République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE AMAR TELIDJI LAGHOUAT

FACULTE DES SCIENCES ET DE L'INGENIERIE

DEPARTEMENT DE GENIE INFORMATIQUE

PROJET DE FIN D'ETUDES

Pour l'Obtention Du Diplôme

D'INGENIEUR D'ETAT EN INFORMATIQUE

Option: Systèmes Parallèles et Distribués

Thème:

Etude et simulation des algorithmes de recouvrement arrière : application des protocoles MS et BCS

Réalisé par:

Fatima BARKAT.

Leila OULED KOUIDER.

Proposé et encadré par:

Melle. Zohra ABDELHAFIDI.

N° d'ordre: ……../2009-PFE/DGI

Dédicaces

À nos parents,
Qu'ils trouvent ici le fruit de leur patience et du soutien permanent et quotidien qu'ils nous ont prodigué pour affronter tous les moments difficiles.

À nos familles et nos amis sans oublier ceux qui ont participés de prés ou de loin à la réalisation de ce travail.

Leila & Fatima

Remerciements

Nous devons tout d'abord remercie Dieu notre créateur, pour le courage et la patience qui nous a donné afin de mener ce projet à terme.

Nous ne saurions oublier de trop remercier nos familles pour leur encouragement et leur soutien tout au long de ce parcours.

Nous tenons à exprimer notre profonde gratitude et reconnaissance à l'égard de notre encadreur Melle. ABDELHAFIDI Zohra pour son aide, sa constante disponibilité et ses précieux conseils qui ont permis ce travail de voir le jour.

Nous remercions les enseignants qui nous avons fait l'honneur de participer au jury de ce mémoire.

Nos respects et nos remerciements à tous les enseignants du département de l'informatique, sincèrement à Melle. BELABACI Amel enseignante à l'université Amar Telidji de Laghouat pour ses conseils, pour la validation des diagrammes UML sans oublier Melle. BEN KOUIDER Sarah pour ses remarques précises pour la partie de programmation.

Table des matières

Table des figures

Table des tableaux

Résumé

Dans notre projet, nous nous intéressons au recouvrement arrière basé sur les points de reprise lequel permet au processus de revenir à un état antérieur sauvegardé dans la mémoire stable. Plusieurs protocoles de recouvrement utilisant les points de reprise sont introduis dans le but de revenir à un état antérieur le plus proche possible et d'éviter problème de l'effet domino.

A cet effet, nous avons simulé et étudié les comportements (via plusieurs scénarios) des algorithmes MS et BCS dus respectivement à D.Manivannan et M.Singhal (1996) et Briatico, Ciufoletti et Simoncini (1984).

Deux approches ont été adoptées:

La première approche (simulation sans fautes) où nous avons calculé le nombre de points de reprise forcés, le nombre de messages sauvegardés en fonction de nombre de processus et la longueur d'intervalle entre deux points de reprise spontanés.

Dans la deuxième approche (simulation avec fautes), nous avons calculé le nombre de points de reprise forcés, le rapport de nombre des accusés de réception sur le nombre de messages sauvegardés, nombre de points de reprise forcés pris et le nombre de points de reprise supprimés lors du processus de recouvrement. Trois cas de processus ont été considérés : processus symétriques, un seul processus rapide et un seul processus lent.

Les résultats obtenus nous ont permis de mieux cerner l'importance de la tolérance aux fautes dans les systèmes répartis et le recouvrement arrière.

Mots clés : NS-2, points de reprise, recouvrement arrière, système réparti, tolérance aux pannes.

Abstract

In our project, we are interested of rollback recovery based in checkpointing, it allows the process to start from the recent state saved in stable storage, several algorithm using Checkpointing are introduced with the aim of starting the recent state and avoiding the domino effect problem.

We simulated and studied the behavior of two algorithms MS and BCS due respectively to D.Manivannan and M.Singhal (1996) and Briatico, Ciufoletti and Simoncini (1984).

Two approaches are adopted:

In the first approach (simulation without faults), we calculate forced checkpoint number and logged messages, as function of processes number and interval length that separate two checkpoints.

In the second approach (simulation with faults), we calculate forced checkpoint number, the ratio between acknowledgement number and logged messages number and forced checkpoint taken, deleted checkpoint number during recovery. Three cases are considered (symmetrical processes, one of processes is fast and one is slow).

The results show the importance of fault tolerance and rollback recovery.

Keywords: checkpoint, distributed system, fault tolerance, NS-2, rollback recovery.

Introduction générale

Les systèmes distribués aujourd'hui sont les plus utilisés car ils incluent : les systèmes client-serveur, système transactionnels, web et l'informatique scientifique. Pendant l'exécution de certaines applications, il peut y avoir des pannes. Pour résoudre un tel problème on a recours à un ensemble de techniques tel que la tolérance aux pannes (ou aux fautes).

La tolérance aux pannes est divisée en deux grandes tâches : La détection des erreurs et le recouvrement des erreurs.

Le recouvrement arrière traite l'application distribuée comme étant un ensemble des processus communiqués via un réseau, il réalise la tolérance aux pannes par l'enregistrement périodique de l'état de processus durant l'exécution normale (sans faute), lors d'une panne, le système doit redémarrer depuis un état sauvegardé. L'état sauvegardé est appelé point de reprise et la procédure de retour en arrière est appelé recouvrement arrière.

Dans les systèmes répartis, si chaque processus prend un point de reprise indépendamment des autres, il est susceptible d'avoir le problème de l'effet domino, il est fortement désirable d'éviter un tel problème. Pour cela plusieurs techniques ont été développées.

Parmi ces techniques, il y'a l'approche de point de reprise coordonné où les processus coordonnent leurs efforts pour former un point de reprise global cohérent. En cas de panne, l'état de système est restauré depuis cet ensemble de points de reprise cohérent.

La deuxième approche, est l'approche induite par communication (CIC : Communication-Induced Checkpointing). Les processus prennent des points de reprise indépendants, mais ils se coordonnent via les informations transportées sur les messages et éventuellement prendre des points de reprise supplémentaires (appelé forcés) pour assurer toujours l'existence d'un point de reprise global cohérent.

Dans ce mémoire nous traitons le problème de recouvrement arrière pour les algorithmes de la classe CIC. Pour cela nous avons simulé deux algorithmes (MS et BCS) et analysé leur performance via différents scénarios.

Ce mémoire est composé de trois chapitres et trois annexes:

- Le premier chapitre est donné sous forme de généralités sur la définition des systèmes répartis avec leurs inconvénients et leurs avantages, la tolérance aux pannes (fautes) avec les fautes d'un processus, et les classes de points de reprise et le recouvrement arrière.

- Dans le deuxième chapitre on donne une description des algorithmes simulés. On a utilisé les diagrammes UML pour simplifier la compréhension de ces algorithmes.

- Le troisième chapitre regroupe les étapes de simulation, l'outil de simulation utilisé et l'analyse des résultats obtenus, ce chapitre contient aussi des diagrammes UML pour expliquer les étapes de simulation et l'intégration des algorithmes simulés dans NS-2.

À la fin de ce mémoire, on termine par une conclusion générale.

- Les annexes sont organisées de la manière suivante:

    I. Définition de simulateur NS-2.

    II. Un exemple de code source (un exemple de protocole MS avec faute).

    III. Le mode d'utilisation pour voir la réalisation de la simulation.

Chapitre I

Systèmes répartis et tolérance aux pannes
Notions et définitions

Le but de ce chapitre est de donner des notions générales sur le système réparti avec leurs avantages et leurs inconvénients, les points de reprise et le recouvrement arrière.

1. Les systèmes répartis

1.1. Définition

Un système réparti (ou distribué de «distributed system») est un ensemble d'ordinateurs (dits sites) ou processeurs reliés entre eux par un système de communication permettant l'échange de données et d'informations [10]. Les deux principaux modèles d'un système réparti sont:

1.1.1. Un système synchrone

Dans un système synchrone :

- Ø Les délais de transmission des messages sont bornés.
- Ø Il existe une borne supérieure pour le temps d'exécution d'une étape par un processus.

La réponse à une sollicitation s'effectue toujours dans un délai borné [7].

1.1.2. Un système asynchrone

Dans un système asynchrone :

- Ø Les délais de transmission des messages ne sont pas bornés.
- Ø Pas de borne sur les vitesses relatives des processus.

Ainsi, un système est dit asynchrone si le délai de réponse à une sollicitation n'est pas borné [7].

1.2. Avantages d'un système réparti

Les avantages d'un système réparti sont regroupés dans les points suivants [10]:

- **La sécurité :** les applications sont conçues selon une approche modulaire permettant d'isoler les données, et donc protéger les accès.
- **Accélération des calculs :** un calcul peut être découpé en sous calculs réalisables en même temps (en parallèle), le système réparti permet alors de répartir les calculs sur les différents processus afin de les exécuter simultanément. Lorsqu'un site est surchargé, certaines tâches peuvent être déplacées ou allouées à un site moins chargé (la répartition des charges).
- **La transparence :** c'est l'influence des systèmes répartis sur plusieurs niveaux en cas d'utilisation des ressources, des usages, l'endroit d'accès.

- **La flexibilité :** l'ajout ou la suppression d'un ordinateur (ou un site) est une opération simple, et n'influe pas sur le fonctionnement total du système.

- **La fiabilité :** un des buts des systèmes répartis est d'obtenir des systèmes plus fiables que les monoprocesseurs. Si une machine tombe en panne, une autre machine devrait prendre la relève, cette situation entraîne généralement la duplication des données partagées pour accroître la robustesse du système, mais cette duplication nécessite une gestion de copies multiples des fichiers.

1.3. Inconvénients d'un système réparti

Les inconvénients des systèmes répartis sont les difficultés concernant surtout la communication [10]:

- **La lenteur de la communication :** malgré le degré de fiabilité offert par les réseaux de communication, et les vitesses de transmission qu'ils proposent (avec l'arrivée des fibres optiques), les réseaux restent relativement lents par rapport à la vitesse de calcul sur les machines, ce qui implique un grand temps d'attente dans le cas d'un travail coopératif entre les processus, et donc on obtient une perte concernant la performance du système.

- **La perte du message:** parfois, les réseaux de communication ne sont pas fiables à 100%, quelques messages transportés par ces réseaux peuvent être perdus, et donc on doit réémettre ces messages, les protocoles de détection de perte, et la génération des nouveaux messages reste jusqu'à présent un des domaines de recherche dans les systèmes répartis.

1.2. La tolérance aux pannes

1.2.1. Définition

La tolérance aux pannes (ou bien aux fautes) est l'ensemble des techniques de conception des systèmes qui continuent de fonctionner même en présence de la panne de l'un de leurs composants. On peut dire qu'un ensemble d'architectures, considérée comme un tout, continue par l'utilisation de redondances de rendre le service attendu en dépit de l'existence de fautes [7].

1.2.2. Les fautes d'un processus

Il existe quatre types de fautes: les fautes par arrêt définitif, les fautes par omission, les fautes de performance, et les fautes arbitraires (byzantines) [8].

A. Faute par arrêt définitif :

Dans la faute par arrêt définitif (fail-stop fault), un processus s'arrête prématurément et ne fait rien à partir de ce point. Dans ce type de défaillance, le processus ne peut pas reprendre son exécution. La terminologie panne franche (ou crash) est aussi utilisée pour décrire un arrêt définitif.

B. Faute par omission :

Dans ce type de fautes (fail-stutter fault**)**, un processus fautif peut omettre certaines actions. Ces actions concernent l'envoi et la réception de messages.

C. Fautes de performance :

Dans ce type de fautes (performance fault), un processus fautif ne respecte pas les délais d'exécution des tâches.

D. Faute arbitraire (byzantine) :

Dans ce type de fautes (byzantine fault), un processus fautif peut se comporter arbitrairement. Un processus byzantin peut, par exemple, corrompre des messages ou encore ne pas exécuter volontairement des parties entières de l'algorithme. Ce type de fautes caractérise principalement les comportements malveillants volontaires.

La figure 1.1 illustre la classification des fautes de processus.

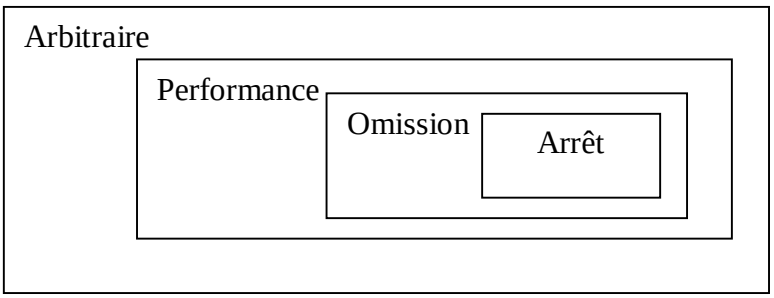


Figure 1. 1 Classification des fautes de processus

Dans le cadre de ce mémoire, nous nous sommes intéressés à la faute par omission (fail-stutter fault**).**

1.2.3. Classes de tolérance aux pannes

La tolérance aux pannes peut être classifiée en deux :

- La tolérance aux pannes par duplication.
- La tolérance aux pannes par mémoire stable.

1.2.3.1. Tolérance aux pannes par duplication

La tolérance aux pannes par duplication est réalisée par masquage d'erreur, la défaillance d'une copie est masquée par le comportement des copies non défaillantes. Cette technique permet de tolérer un nombre limité de pannes.

Trois stratégies pour réaliser la duplication: la duplication active, la duplication passive, et la duplication semi-active. Ces différentes stratégies visent à garantir une cohérence forte entre les copies d'un composant dupliqué [12].

ü **Duplication active:**

Elle est définie par la symétrie du comportement des copies d'un composant dupliqué où chaque copie joue un rôle identique à celui des autres.

ü **Duplication passive:**

Contrairement à la duplication active, la duplication passive (passive replication ou primary-backups replication) est asymétrique. Elle distingue deux comportements pour les copies d'un composant dupliqué : la copie primaire (primary copy) et les copies secondaires (backups). La copie primaire est la seule à effectuer tous les traitements. Les copies secondaires, oisives, surveillent la copie primaire. En cas de défaillance de la copie primaire, une des copies secondaires devient la nouvelle copie primaire.

ü **Duplication semi-active:**

La duplication semi-active (semi-active replication ou leader followers replication) se situe à mi-chemin entre la duplication active et la duplication passive. Comme cette dernière, la duplication semi-active est une stratégie asymétrique. Contrairement à la duplication passive, les copies secondaires ne sont pas oisives.

1.2.3.2. Tolérance aux pannes par mémoire stable

La tolérance aux pannes par mémoire stable utilise un support persistant de stockage dont le rôle principal est d'assurer une accessibilité et une protection des données contre les pannes pouvant affecter le système. Ainsi, suite à une panne, un état correct ayant été stocké antérieurement à cette panne sur la mémoire stable reste accessible; cela permet au système un retour à un état antérieur [12].

Il existe deux types de recouvrement :

- Recouvrement basé sur la journalisation.
- Recouvrement basé sur les points de reprise.

1.3. Recouvrement arrière basé sur la journalisation

Dans la tolérance aux pannes par journalisation on peut distinguer trois approches différentes: pessimiste, optimiste, et causale [12].

1.3.1. Journalisation pessimiste

Le principe de la journalisation pessimiste est que tous les évènements (messages en transit) sont enregistrés sur la mémoire stable d'une façon synchrone où l'ordre de ces évènements doit être conservé. Si un événement n'est pas enregistré sur la mémoire stable alors aucun processus ne dépend de cet événement.

L'avantage de cette approche est qu'elle ne crée jamais de processus orphelin (c'est un processus dont un message qui lui est envoyé a été perdu). Mais son inconvénient est qu'elle bloque le processus avant chaque retrait de message et donc la journalisation pessimiste introduit un surcoût important pendant l'exécution normale (sans panne).

1.3.2. Journalisation optimiste

Le principe de la journalisation optimiste est que les évènements (pas forcément les messages reçus) dans le système sont enregistrés de façon asynchrone.

L'avantage de cette approche est qu'un processus n'a pas besoin de se bloquer pendant le stockage sur la mémoire stable. Mais son inconvénient est que pour éviter les processus orphelins, un processus peut revenir à l'état initial de calcul (problème de l'effet domino). De plus, un surcoût important dû au calcul de l'état global cohérent est introduit durant la reprise.

1.3.3. Journalisation causale

Le principe de la journalisation causale est d'assurer que le déterminant de chaque événement non déterministe (la réception d'un message ou d'un événement interne au processus) qui précède causalement l'état d'un processus se trouve soit en mémoire stable, soit est disponible localement pour ce processus.

La journalisation causale a l'avantage des deux méthodes précédentes en termes de performances à l'exécution et en cas de reprise. La journalisation causale évite d'une part, la synchronisation avec la mémoire stable à l'exception du moment de la publication de résultats (journalisation optimiste). D'autre part, la journalisation causale ne crée jamais de processus orphelin (comme la journalisation pessimiste). Cela permet la reprise de n'importe quel processus défaillant à partir de son dernier état sauvegardé sans le problème de l'effet

domino. Mais l'inconvénient de cette technique réside en la complexité du protocole de recouvrement arrière suite à une panne.

1.4. Recouvrement arrière basé sur les points de reprise

1.4.1. Définitions

Avant d'entamer les différentes classes de recouvrement arrière qui sont basées sur les points de reprise, nous citons quelques définitions nécessaires:

1.4.1.1. Point de reprise local

Un point de reprise local (ou un état local) est défini comme un enregistrement complet de l'état de processus. Cet enregistrement sera utilisé en cas de panne ce qui évite au processus de repartir depuis le début de son exécution [9].

Un point de reprise $C_{i,x}$ représente le $x^{ème}$ point de reprise de processus P_i.

1.4.1.2. Point de reprise global

Un point de reprise global (ou un état global) d'un système réparti est un ensemble de points de reprise locaux un par processus [9].

Ce point de reprise peut être cohérent ou incohérent, un point de reprise global est cohérent si pour chaque message (m) tel que sa réception est enregistrée dans le point de reprise global, son émission est aussi enregistrée, ce point de reprise global reste cohérent même si un message a été envoyé mais non reçu (message en transit) [1].

Un message en transit : est un message qui a été envoyé avant un point de reprise appartenant à un point de reprise et reçu après un point de reprise appartenant à cet point de reprise global [3].

La figure 1.2 représente un point de reprise global cohérent ($C_{i,1}$, $C_{j,1}$, $C_{k,1}$) et le message m_3 est en transit.

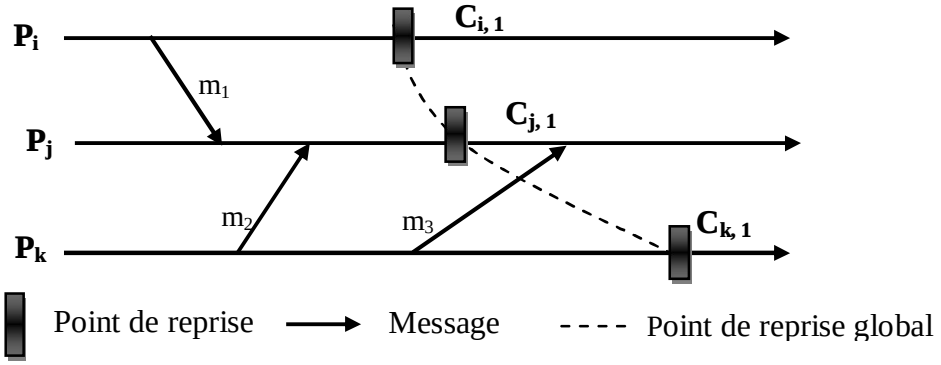


Figure 1. 2 Un Point de reprise global cohérent

Un point de reprise global est incohérent à cause du message orphelin.

Un message orphelin : est un message qui a été envoyé après un point de reprise appartenant à un état global et reçu avant un point de reprise appartenant à cet état global [3].

La figure 1.3 représente un point de reprise global incohérent ($C_{i,1}$, $C_{j,1}$, $C_{k,1}$), le message m_3 est orphelin.

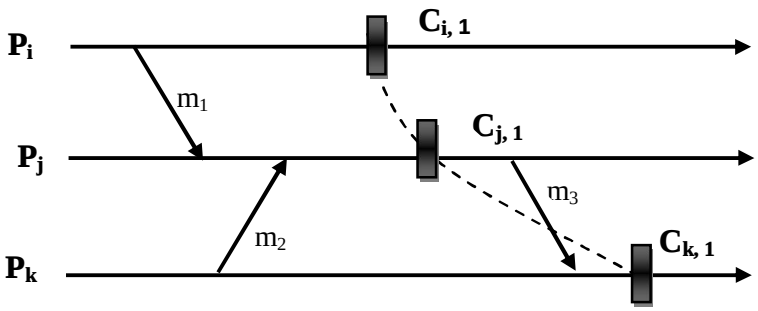


Figure 1. 3 Un Point de reprise global incohérent

1.4.1.3. L'effet domino

Le problème de l'effet domino apparaît lorsque chaque processus prend des points de reprise sans coordination avec les autres processus, il est caractérisé par une cascade de retours en arrière lors de la reprise du système après une panne. Lors de la panne du processus (exemple P_k dans la figure 1.4), le système va tenter de reprendre depuis le point de reprise global le plus récent.

À cause d'un message orphelin, chaque point de reprise global choisi reste incohérent et le système va redémarrer à partir de point de reprise initial (début d'exécution) et donc une grande partie de travail sera perdue [1] [3].

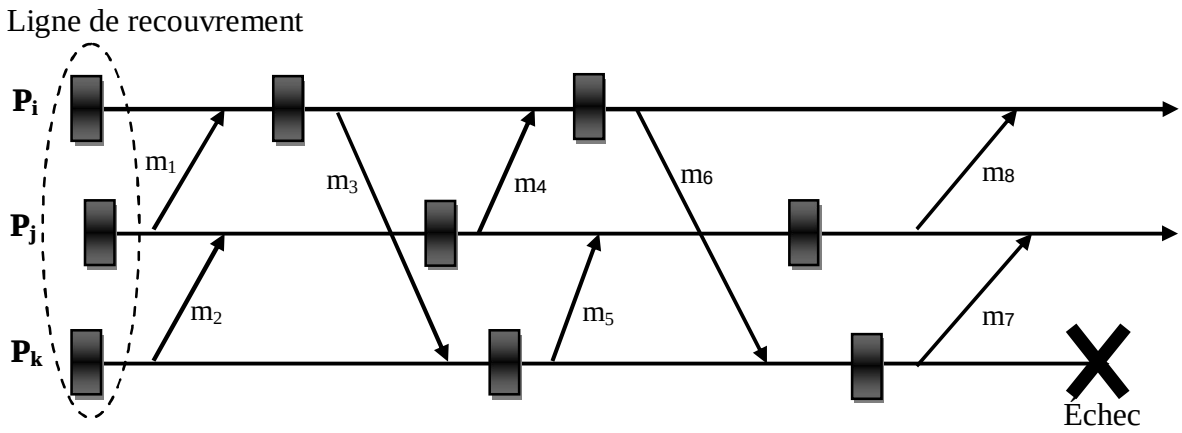


Figure 1. 4 L'effet domino

1.4.1.4. Z-Chemin

Un Z-Chemin 'Zig-Zag' entre deux point de reprise $C_{i,x}$ et $C_{j,y}$ est une séquence particulière de messages $\{m_1, m_2,\ldots, m_q\}$ tel que l'événement émission du message m_i produit par un processus appartient au même intervalle ou à l'intervalle suivant qui contient la réception du message m_{i+1}[1].

Dans la figure 1.5, La séquence $[m_1, m_2]$ est un Z-Chemin.

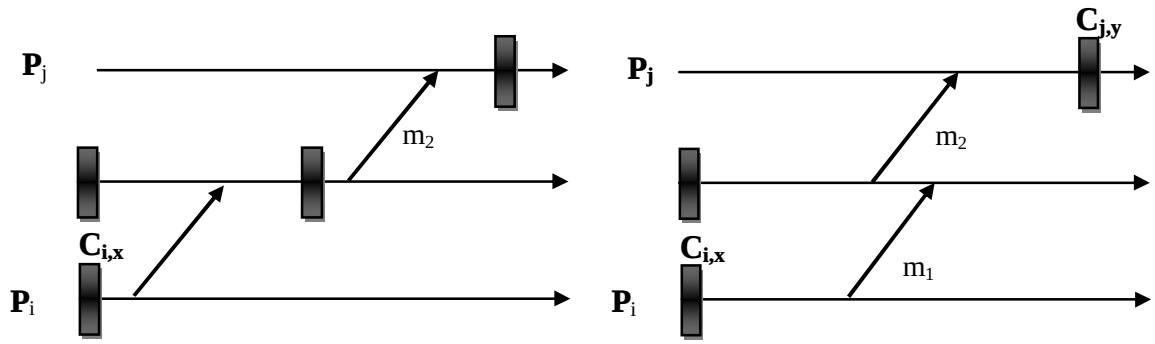


Figure 1. 5 Les Z-Chemins

Définition : un intervalle (noté $I_{i,x}$) est défini par une séquence des événements produits par un processus P_i entre deux points de reprise successifs $C_{i,x-1}$ et $C_{i,x}$, un intervalle $I_{i,x}$ peut être vu comme un « macro-évènement» qui fait passer le processus P_i du point de reprise $C_{i,x-1}$ au point de reprise $C_{i,x}$ [1].

1.4.1.5. Z-Dépendance

Un point de reprise local $C_{i,x}$ Z-Dépendance d'un autre point de reprise $C_{j,y}$ ($C_{i,x} \overset{z}{\rightarrow} C_{j,y}$), soit parce que tous les deux sont du même processus (i = j) et $C_{i,x}$ précède $C_{j,y}$ ou parce qu'il existe un Z-Chemin de $C_{i,x}$ à $C_{j,y}$ [1].

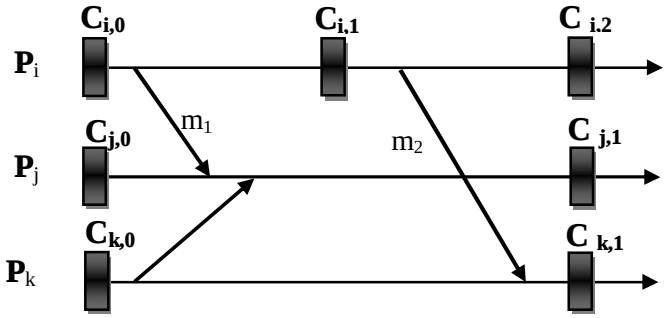


Figure 1. 6 Z-Dépendance (exécution répartie)

Remarque : Un point de reprise global G est cohérent si et seulement si :

$$\forall\, C_{i,x}, C_{j,y} \qquad G: \neg (C_{i,x} \overset{z}{\rightarrow} C_{j,y}) .$$

1.4.1.6. Z-Cycle

Un Z-Cycle est un Z-Dépendance du point de reprise local $C_{i,x}$ à lui-même $C_{i,x}$, un Z-Cycle est dit **inutile**, car quelque soit le point de reprise global contenant $C_{i,x}$, ce point est toujours incohérent [1].

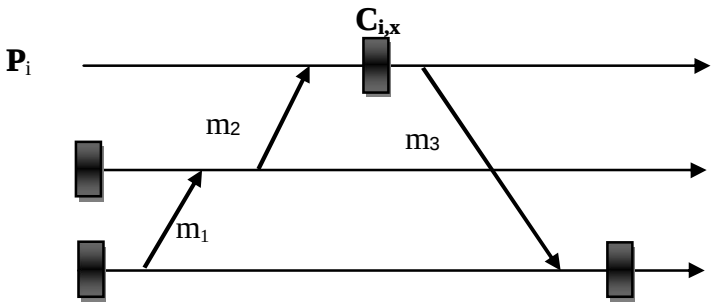


Figure 1. 7 Z-Cycle

1.4.2. Classes de recouvrement arrière basé sur les points de reprise

Le recouvrement arrière survient dés que le système est en panne à cause d'un échec d'un ou de plusieurs processus. Le mécanisme de recouvrement arrière à l'aide des techniques organise le retour en arrière d'un ensemble de processus au dernier point de reprise global cohérent pour se ré-exécuter [6].

Les techniques de recouvrement arrière peuvent être classifiées en trois catégories : point de reprise non coordonné (Uncoordinated Checkpointing), point de reprise coordonné (Coordinated Checkpointing), et point de reprise induit par communication (Communication-Induced Checkpointing).

1.4.2.1. Point de reprise non coordonné

Dans cette technique (Uncoordinated Checkpointing), chaque processus prend d'une manière indépendante (asynchrone) son point de reprise local (point de reprise spontané). L'avantage principal de cette technique est que chaque processus sauvegarde son point de reprise sans aucune nécessité de coordination avec les autres processus. Pendant l'exécution, les processus sauvegardent les dépendances causales entre leurs points de reprise causés par l'échange des messages qui sont suivies par une estampille temporelle. La dépendance causale est généralement utilisée dans un point de reprise non coordonné.

Mais, le retour en arrière est lent parce que les processus doivent recommencer pour trouver un point de reprise cohérent, ces points peuvent être des points de reprise inutiles qui ne font jamais partie d'aucun point de reprise global cohérent et ne contribuent pas à

l'avancement de la ligne de recouvrement. De plus, pendant le retour en arrière les processus peuvent revenir au début d'exécution (problème de l'effet domino) [6].

1.4.2.2. Point de reprise coordonné

Cette technique (Coordinated Checkpointing) permet aux processus de prendre des points de reprise d'une manière synchrone. Quand un processus reçoit un message pour déterminer un état global cohérent, il arrête son exécution et revient au dernier point de reprise sauvegardé. L'avantage est que l'effet domino ne se produit pas et l'espace de stockage est réduit.

Mais l'inconvénient principal de cette technique est de bloquer les communications afin de déterminer un point de reprise global cohérent qui est nécessaire avant que les messages ne peuvent être envoyés [6].

1.4.2.3. Point de reprise induit par communication

Dans cette technique (Communication-Induced Checkpointing), chaque processus prend des points de reprise locaux (spontanés) d'une manière indépendante, mais à partir de l'information superposée par des messages envoyés par d'autres processus, le processus peut prendre des points de reprise supplémentaires appelés forcés. Cette technique permet d'empêcher la création des points de reprise inutiles et donc d'éviter la présence de l'effet domino puisqu'elle utilise une théorie basée sur les notions de Z-Chemin et de Z-Cycle [6]**.**

1.5. Conclusion

Dans ce chapitre, on a donné des notions générales sur le système réparti et la tolérance aux pannes avec les différents types des fautes d'un processus. On a vu également les points de reprise et le mécanisme de recouvrement.

Chapitre II

Les algorithmes simulés

Le but de ce chapitre est de donner une description complète pour les algorithmes simulés (protocole MS et protocole BCS) en utilisant le langage UML (Unified Modeling Language).

2.1. Les algorithmes de point de reprise sans recouvrement

2.1.1. Description de protocole MS

Le protocole MS (l'algorithme Quasi-Synchrone (the Quasi-Synchronous Algorithm « Communication-Induced Checkpointing»)) est proposé par D.Manivannan et M.Singhal en 1996, cet algorithme est géré de la manière suivante [5]:

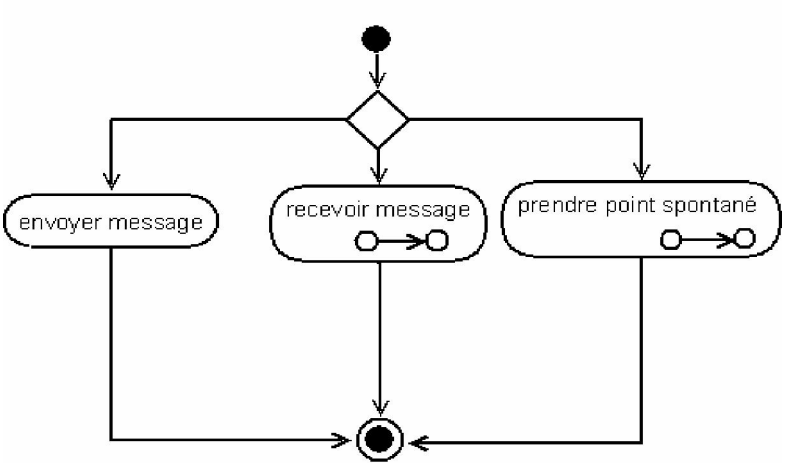


Figure 2. 1 Diagramme d'activité «les étapes nécessaires de l'algorithme (sans recouvrement)»

Note :

O→O Signifie que cette activité (exemple : recevoir message) est détaillée par un autre diagramme d'activité.

Chaque processus (P_i) dispose d'une variable $next_i$ et sn_i pour garder la trace du nombre actuel de point de reprise de processus.

1) Avant que le processus (P_i) ne prenne un point de reprise spontané, il incrémente $next_i$ d'une unité de temps et associe cette valeur à sn_i du point de reprise.

2) Quand un processus P_i envoie un message M, il superpose le numéro de séquence *sn* sur le message noté *M.sn*.

3) Chaque processus (en état normal) peut prendre un point de reprise spontané (basic checkpoint) (la figure 2.2):

   - Si *next* > *sn*, il est possible qu'un processus P_i prend un point de reprise spontanés.

- Si la condition *next* > *sn* n'est pas vérifiée, l'ajout d'un autre point de reprise spontané est annulé (un point de reprise échappé) à cause de la présence d'un point de reprise forcé enregistré.

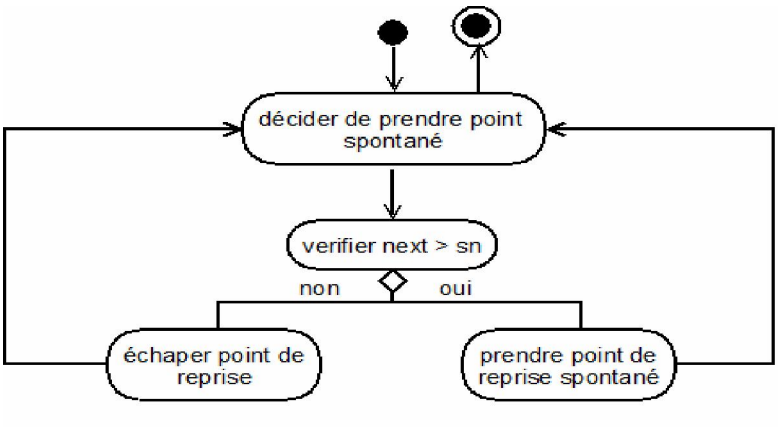


Figure 2. 2 Diagramme d'activité «prendre point spontané»

4) Lors de la réception d'un message, chaque processus en état normal vérifie les deux valeurs de *M.sn* et *sn* (la figure 2.3) :

- Si *M.sn* > *sn*, alors le processus prend un point de reprise supplémentaire (forced checkpoint) et poursuit le message.

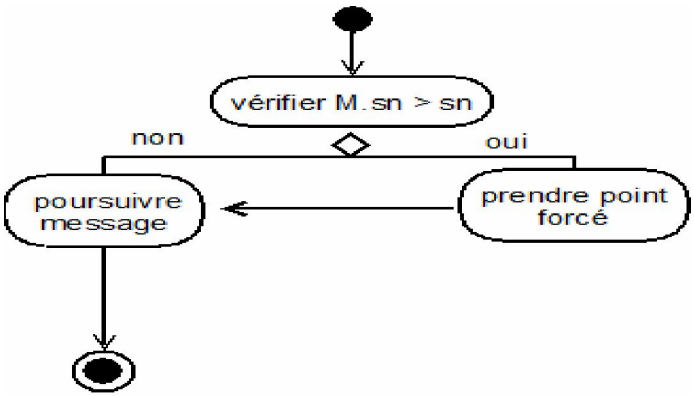


Figure 2. 3 Diagramme d'activité «recevoir message (sans recouvrement)»

2.1.1.1. La structure de données

sn_i : entier ; // numéro de séquence de point de reprise, initialisé à 0.

$next_i$: entier ; // $next_i$ = x (unités de temps), numéro de séquence à affecter pour le prochain point de reprise, initialisé à 1.

2.1.1.2. L'algorithme

Début

1. Quand un processus P_i décide d'incrémenter « $next_i$ » :

$next_i$:= **$next_i$** + 1;

2. Quand un processus P_i prend un point de reprise spontané (basic checkpoint):

Si **($next_i$ > sn_i)** alors

//prendre un point de reprise C

C.sn : = **$next_i$** ; // C.sn reçoit $next_i$ comme numéro de séquence

sn_i : = **C.sn** ; // le sn_i est mis à jour

Finsi

3. Quand un processus P_i envoie un message M :

M.sn: = sn_i ; //sn superposé (piggybacked) avec M

envoyer **(M)**

4. Quand un processus P_j reçoit un message M d'un processus P_i :

Si (**M.sn > sn_j**) alors

//prendre un point de reprise forcé C

C.sn : = **M.sn** ;

sn_j : = **C.sn** ;

sinon

//poursuivre le message

Finsi

Fin

2.1.1.3. Exemple

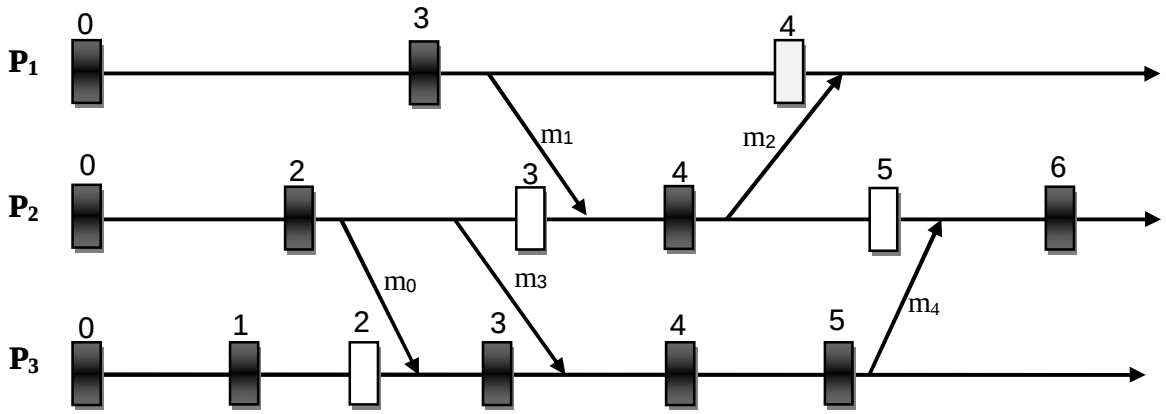


Figure 2. 4 Exemple de protocole MS

Un point de reprise spontané (basic checkpoint).

Un point de reprise forcé (forced checkpoint).

Soit un exemple (sans recouvrement) d'un système composé de trois processus (la figure 2.4), chaque processus incrémente la valeur de *next* périodiquement ($next_1$=3 unités de temps, $next_2$=2 unités de temps et $next_3$=une unité de temps).

Lors de la réception des messages, certains processus sont forcés pour prendre des points de reprise supplémentaires (forced checkpoint) et font une mis à jour de la valeur de *sn* par rapport à la valeur de *M.sn* superposée par le message (le processus P_2 reçoit le message m_0 et prend un point de reprise forcé ($M.sn$=2 > sn=1) par contre la réception de message m_3 ne nécessite pas la création d'un point de reprise forcé ($M.sn$= 2 < sn= 3).

2.1.2. Description de protocole BCS

Le protocole BCS est proposé par Briatico, Ciufoletti et Simoncini en 1984, basé sur la stratégie d'estampillage normale qui est gérée de la manière suivante [4]:

1) Chaque processus P_i dispose d'une estampille sn_i.

2) Avant que le processus P_i ne prenne un point de reprise spontané, il incrémente sn_i de '1' et associe cette valeur à l'estampille du point de reprise.

3) A l'émission d'un message M, le processus P_i estampille ce message par sn_i (noté par *M.sn*).

4) Quand un processus P_i reçoit un message m, il met sn_i égale au maximum entre son estampille et l'estampille du message reçu (sn_i = max (*M.sn*, sn_i)), alors le processus P_i

prend un point de reprise forcé avant la consommation du message si *M.sn* est supérieure à sn_i.

2.1.2.1. La structure de données

sn_i : entier ; // initialisé à 0.

sn : entier ; **//**entête de message M.

2.1.2.2. L'algorithme

Début

1. Quand un processus P_i décide de prendre un point de reprise spontané (basic checkpoint):

sn_i : = **sn_i** +1; // incrémentation de l'estampille local de processus P_i

2. Quand un processus P_i envoie un message M :

M.sn := **sn_i** ;

envoyer M au processus P_j

3. Quand un processus P_i reçoit un message M:

Si **(M.sn > sn_i)** alors

sn_i := Max (sn_i, M.sn) ;

//Prendre un point de reprise forcé

Finsi

Fin

2.1.2.3. Exemple

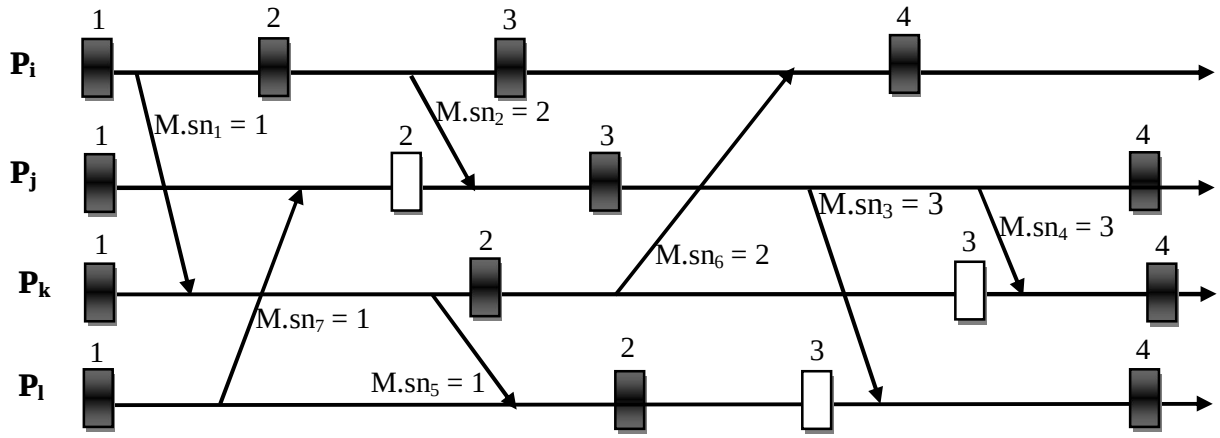


Figure 2. 5 Exemple de protocole BCS

Soit un exemple (sans recouvrement) d'un système composé de quatre processus (la figure 2.5), le processus P_j prend un point de reprise forcé après la réception de message M_2 ($M.sn_2 = 2 > sn_j = 1$). Par contre le processus P_i continue son exécution sans faire un point de reprise forcé ($M.sn_6 = 2 < sn_i = 3$).

2.2. Les algorithmes de point de reprise avec recouvrement

Les hypothèses de cet algorithme sont [5]:

- La panne traitée est de type arrêt définitif (fail-stop fault).
- Un seul processus tombe en panne dans le système.
- Lors de recouvrement, les processus ne bloquent pas leurs exécutions.

2.2.1. Les états d'un Processus

Un processus peut avoir trois états (normal, en panne, en recouvrement) :

- Dans un état normal, un processus peut envoyer des messages simples aux autres, recevoir des messages de recouvrement ou des messages portant l'information qu'il y a un processus en panne. La réception de ces deux types de message met le processus dans un état de recouvrement.
- A cause d'une erreur, un processus peut tomber en panne et devient en état de recouvrement.

- A la fin de recouvrement, le processus peut revenir à son état normal et continuer son exécution.

La figure 2.6 représente les états d'un processus.

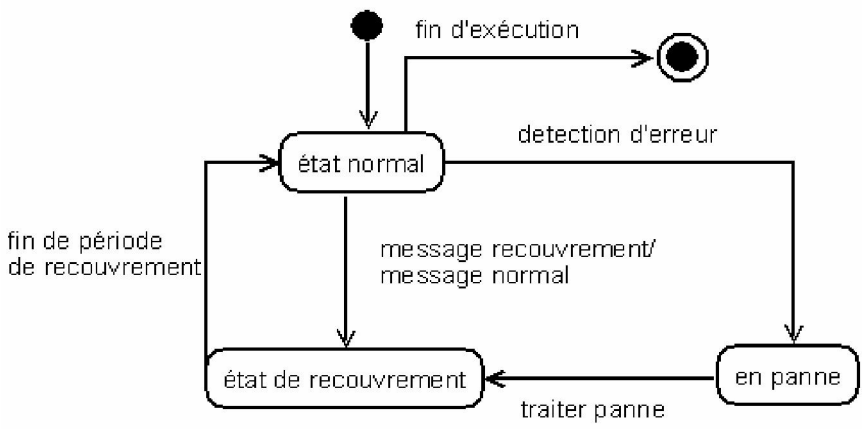


Figure 2. 6 Diagramme d'état / transition «les états d'un Processus»

2.2.2. Description de mécanisme de recouvrement

Cet algorithme (Basic Recovery Algorithm) [5] est une amélioration de l'algorithme précédent pour qu'il traite le cas de panne, deux variables importantes sont ajoutées:

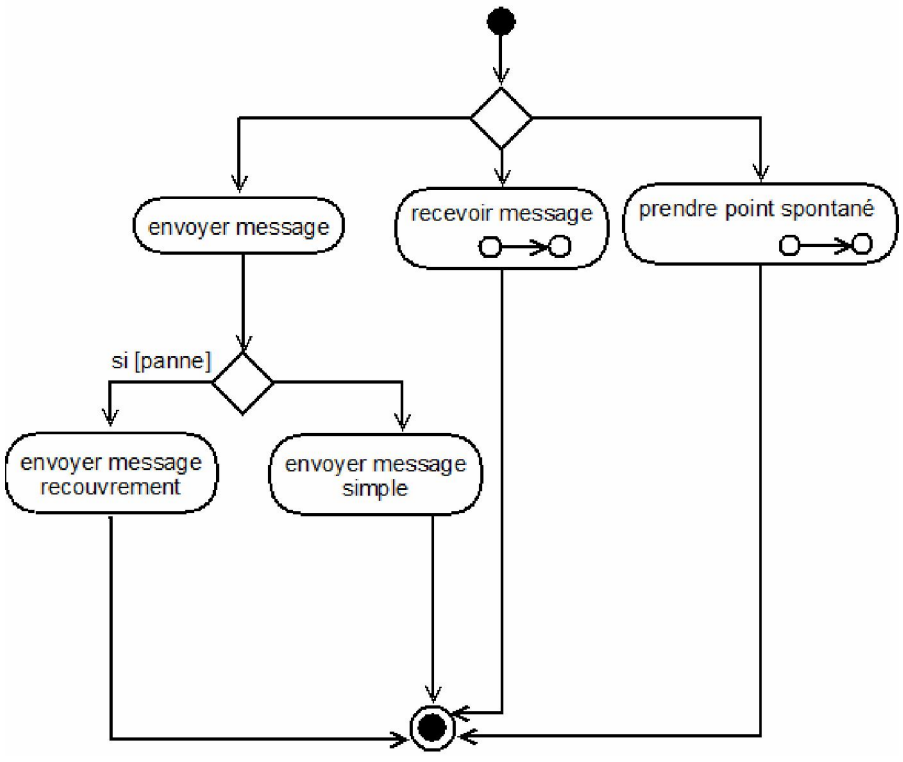


Figure 2. 7 Diagramme d'activité «les étapes nécessaires de l'algorithme (avec recouvrement)»

1) Chaque processus (P_i) dispose d'une variable inc_i pour garder le nombre d'erreur pour chaque processus et une variable rec_line_i pour garder le numéro de la ligne de recouvrement.

2) Un processus qui tombe en panne, envoie des messages de recouvrement aux autres processus et retourne à un état précédent (recouvrement).

   - Quand un processus reçoit un message normal, il vérifie si ce message porte une nouvelle information concernant les pannes (*M.inc>inc*) et éventuellement fait un recouvrement, sinon il poursuit son exécution.
   - Lors de la réception d'un message de recouvrement le processus vérifie s'il y'a une autre panne (*M.inc>inc*), si c'est le cas le processus exécute la procédure de recouvrement, sinon il ignore ce message.
   - Pendant le recouvrement, un processus peut prendre un point forcé si *M.rec_line >sn* sinon il restaure un point de reprise déjà sauvegardé.

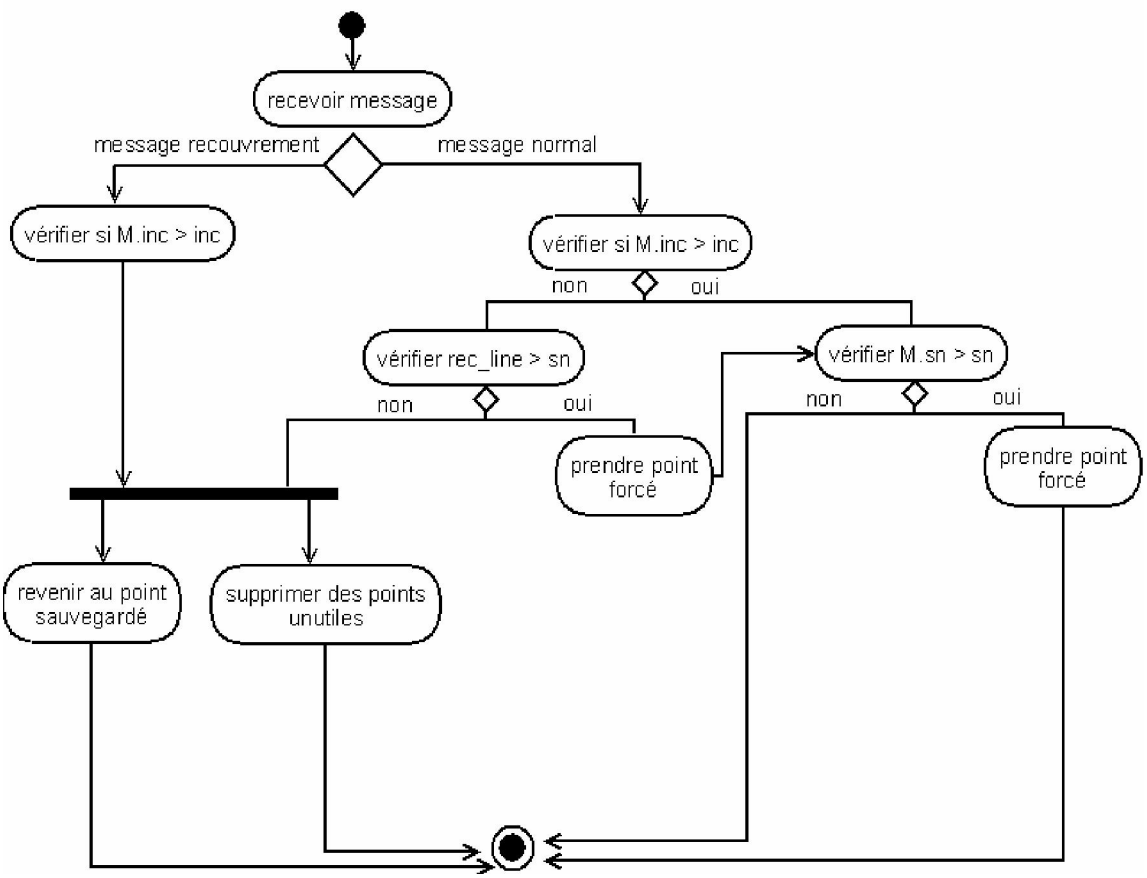


Figure 2. 8 Diagramme d'activité «recevoir message (avec recouvrement)»

2.2.2.1. La structure de données

sn_i : entier;

$next_i$: entier;

inc_i : entier; // numéro d'incarnation de processus, initialisé à 0.

rec_line_i : entier; // numéro de dernier ligne de recouvrement, initialisé à 0.

Les deux premiers procédures de l'algorithme précédent sont les mêmes pour cet algorithme.

2.2.2.2. L'algorithme

Début

3. Quand un processus Pi envoie un message M :

M.sn: = sn_i ; //sn superposé (piggybacked) avec M

M.rec_line: = rec_line_i ; //rec_line superposé (piggybacked) avec M

M.inc: = inc_i ; //inc superposé (piggybacked) avec M

envoyer (M);

4. Quand un processus P_j reçoit un message M :

Si (**M.inc > inc_j**) alors

rec_line_j := **M.rec_line**;

inc_j := **M.inc**;

recouvrement (Pj);

Finsi

Si (**M.sn > sn_j**) alors

//prendre un point de reprise forcé C

C.sn := **M.sn**;

sn_j := **C.sn**;

sinon

//poursuivre le message

Finsi

5. Après l'échec de processus Pi :

//revenir au dernier point sauvegardé

inc_i := **inc_i** + 1;

rec_line_i := **sn_i** ;

envoyer recouvrement (inc_i, rec_line_i) à tous les autres processus

//reprendre l'exécution

6. P_j processus lors de la réception d'un message de recouvrement (inc_i, rec_line_i) d'un processus P_i :

Si (**inc_i** > **inc_j)** alors

inc_j := **inc_i**;

rec_line_j : = **rec_line_i**;

recouvrement(P_j);

//continuer comme d'habitude

Sinon

//ignorer le message de rollback

Finsi

7. La procédure recouvrement (P_j) :

Début

Si (**rec_line_j** > **sn_j**) alors

//prendre un point de reprise spontané C;

C.sn := **rec_line_j**;

sn_j := **C.sn**;

Sinon

//trouver le point le plus proche de C avec C.sn ≥ rec_line_j

sn_j := **C.sn**;

//restaurer le point de reprise C

//supprimer tous les points de reprise après C

Finsi

Fin

Fin

Remarque:

On adapte le même mécanisme de recouvrement pour les algorithmes MS et BCS.

2.2.3. La classification des messages

Pour aider à clarifier la classification des messages, nous utilisons un exemple qui regroupe les types des messages qui sont définis à la suite.

Soit un système composé de quatre processus P_1, P_2, P_3 et P_4 et désigné par la Figure 2.9. Les points de reprise spontanés (basic checkpoint) sont indiqués par des rectangles « ▮ » et les messages sont représentés par des flèches « ⟶ ».

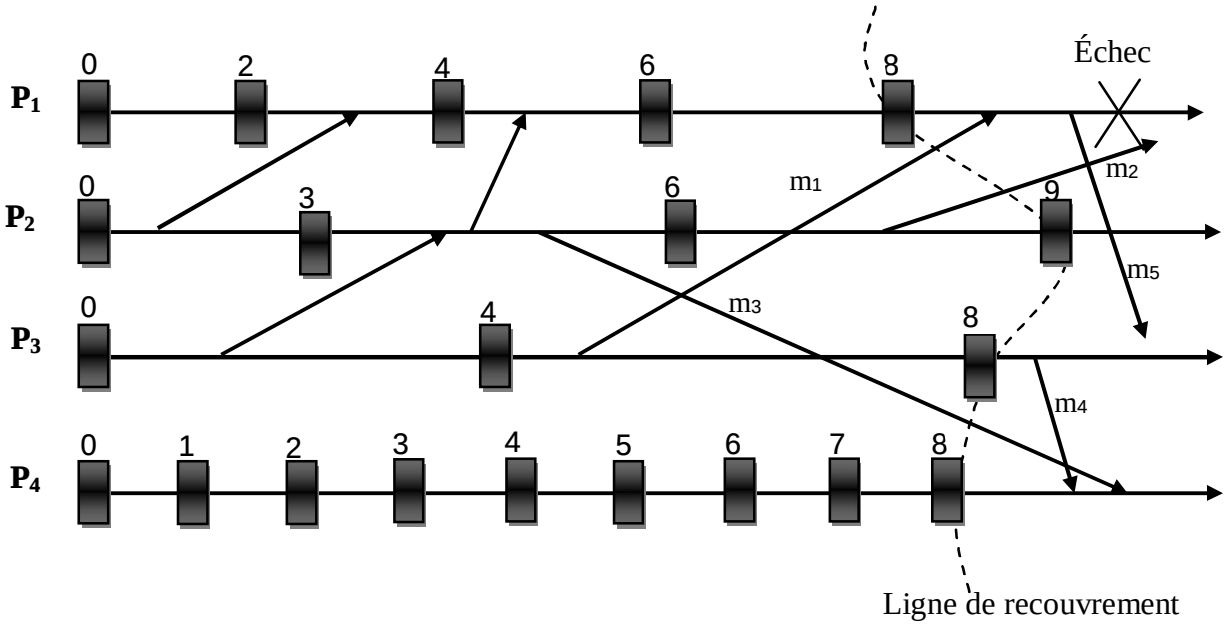


Figure 2. 9 Différent types de messages

2.2.3.1. Message en transit

Ce type de messages se pose quand un processus fait un retour à un point de reprise avant la réception du message alors que l'expéditeur ne doit pas revenir à un point de reprise avant de l'envoyer.

La figure 2.9 montre que le message m_1 est un message en transit par rapport au point de reprise global ($C_{1,8}$, $C_{2,9}$, $C_{3,8}$, $C_{4,8}$)

2.2.3.2. Message omis

Ce type de message apparait si le processus récepteur retourne à un point de reprise qui précède sa réception.

2.2.3.3. Message retardé

Ce type de message apparait quand l'émission est faite avant la ligne de recouvrement, mais il n'est reçu qu'après la ligne de recouvrement.

Dans la figure 2.9, m_2 et m_5 se sont des messages retardés.

2.2.3.4. Message orphelin

Ce sont les messages dont leurs réceptions sont enregistrés mais leurs émissions ne sont pas enregistrés (si on retourne vers un point de reprise global cohérent, on ne peut pas avoir des messages orphelins).

2.2.3.5. Message dupliqué

Ce sont les messages retransmis à cause de recouvrement.

Le message m_4 (Figure 2.9) était envoyé et reçu avant le recouvrement. En raison de recouvrement, P_4 a annulé la réception de m_4 et P_3 a annulé l'envoie de m_4. Ainsi, après P_4 fait un retour au point de reprise avec un numéro de séquence (*sn*=8), il convient de ne pas retransmettre le message m_4 depuis P_3 a annulé l'envoie de m_4 ; Si P_4 retransmit m_4, m_4 sera un message doublé, car il sera envoyé à nouveau par P_3.

2.2.4. Traitement des messages par l'algorithme de recouvrement

Dans ce mécanisme, un processus P_i enregistre les messages en transit et les messages retardés :

$$(M.inc < inc_i \text{ et } M.sn < rec\_line) \text{ ou } (Minc = inc \text{ et } M.sn < sn)$$

Un processus n'envoie des accusés de réception (ACK) que pour les messages enregistrés. Voici la condition pour envoyer un ACK :

$$M \text{ est reçu après } C \text{ et } M.sn < rec\_line$$

Un processus considère le message M comme étant un message doublé si :

$$M.sn > rec\_line$$

Dans ce cas, il annule la réception d'un tel message.

La figure 2.8 représente le diagramme d'activités «recevoir message» qui regroupe les traitements des messages par l'algorithme de recouvrement:

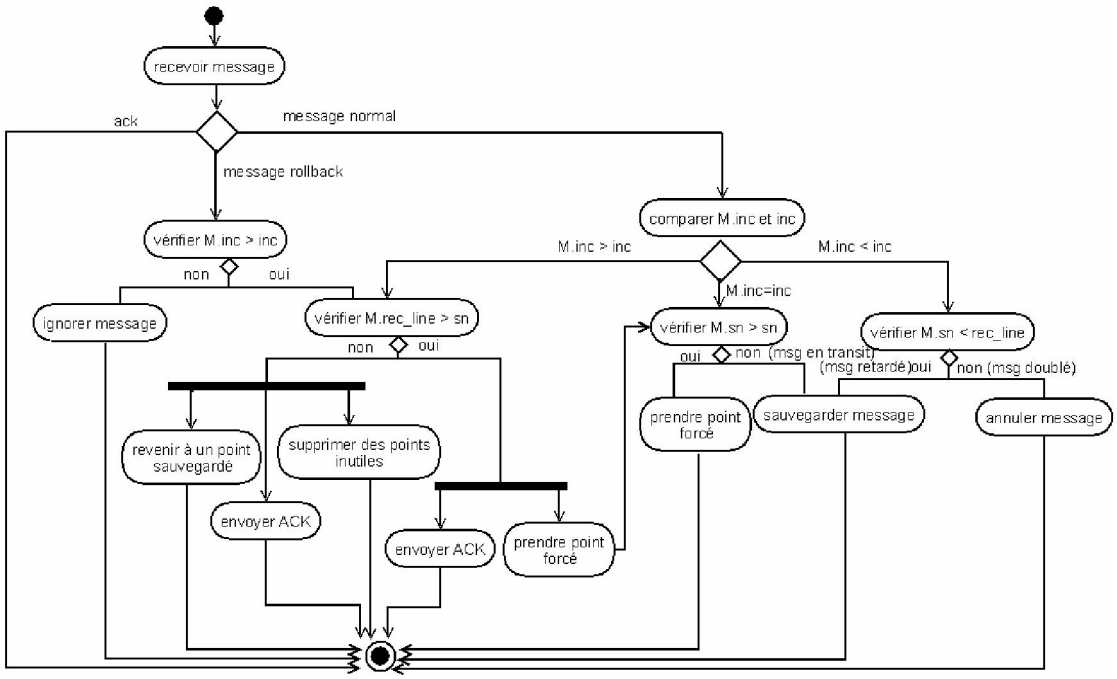


Figure 2. 10 Diagramme d'activité «recevoir message et traitement des messages (avec recouvrement)»

2.2.5. Exemple de l'algorithme de recouvrement:

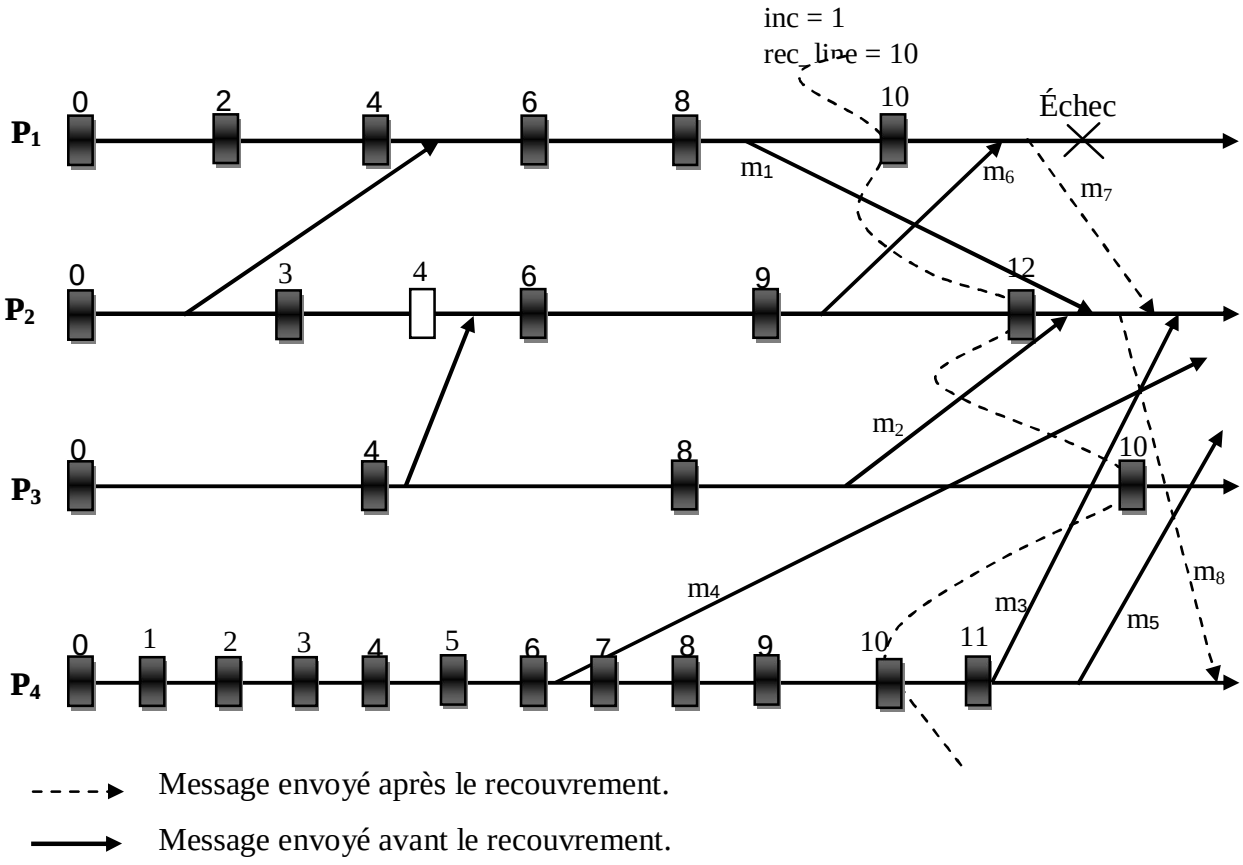


Figure 2. 11 Traitement de messages pendant le recouvrement arrière (algorithme MS)

Chaque processus Pi incrémente sa valeur de $next_i$ à chaque unité de temps x :

- Le processus P_1 prend un point de reprise pour toutes les 2 unités du temps.
- Le processus P_2 prend un point de reprise pour toutes les 3 unités du temps.
- Le processus P_3 a un point de reprise pour toutes les 4 unités du temps.
- Le processus P_4 a un point de reprise chaque unité du temps.

Dans la Figure2.11 Supposons que le processus P_1 tombe en panne (le premier échec dans le système), alors il revient à son dernier point de reprise $C_{1,\ 10}$, incrémente inc_1 à 1, et $rec\text{-}line_1$:=10, et envoie un message recouvrement(1,10) à tous les autres processus. Quand un processus P_i reçoit ce message (recouvrement (1,10)), il met à jour ces variables, inc_i := 1 (*M.inc*) et rec_line_i := 10 (*M.rec_line*) et revient au dernier point de reprise dont le numéro de séquence $sn_i \geq rec\_line_1$. Les messages montrés par des flèches hardies ont un *inc* =0 et les messages montrés par des flèches cassées ont un *inc* =1.

- Le message m_4 est enregistré et ensuite traité par P_2, c'est le même cas pour les messages m_1, m_2, et m_6.
- Le message m_5 est annulé par P_2, car ($M_5.sn = 11 \geq rec\_line_2 = 10$) et ($M_5.inc = 0 < inc_2 = 1$), la même remarque pour le message m_3.
- Le message m_7 doit être enregistré avant le traitement parce que si P_2 fait un retour en arrière au point de contrôle $C_{2,\ 12}$ en raison de son échec ou en raison de l'échec d'un autre processus dans le futur, alors P_2 devra retransmettre m_7 si *rec_line* > 10, cette remarque est la même pour le message m_8.

2.4. Conclusion

Dans ce chapitre, nous avons décrit les algorithmes MS et BCS et le mécanisme de recouvrement utilisé. Nous avons détaillé le traitement de chaque type de message et le comportement de processus pour chaque cas.

Chapitre III

Simulation et analyse de performance

Le but de ce chapitre est de simuler les deux algorithmes (protocole MS et protocole BCS) avec le simulateur NS-2, afin de faire une étude comparative pour évaluer la performance des algorithmes.

3.1. Les techniques d'évaluation des performances

Il existe différentes techniques pour l'évaluation des performances d'un système [2]:

3.1.1. Mesure (émulation)

Le principe de cette technique est de faire des mesures et de les analyser directement sur un système réel. Cette technique permet de comprendre le vrai comportement du système. Mais elle n'est pas toujours réalisable car le fonctionnement de système réel peut être perturbé, en plus les résultats issus de cette mesure ne reflète qu'une seule trajectoire du système.

3.1.2. Modélisation

Le principe de cette technique est de réduire le système en un modèle mathématique (les automates, les réseaux de pétri, les approches probabilistes,…) et de l'analyser numériquement. Généralement, certaines hypothèses sont posées pour simplifier l'étape de modélisation du système et rendent l'évaluation numérique faisable. Ces hypothèses simplificatrices peuvent toucher la fidélité de la représentation du système.

3.1.3. Simulation

Cette technique est basée sur l'implémentation d'un modèle simplifié du système à l'aide d'un programme de simulation adéquat. Cette méthode traduit le comportement du système à évaluer d'une manière réaliste. La simulation permet en plus de visualiser les résultats sous forme de graphes faciles à analyser et à interpréter. La simulation n'est pas une méthode exacte, et nécessite de prêter une attention particulière aux interprétations des résultats de simulation.

À partir de ces trois définitions, nous avons conclu que la simulation permet de tester à moindre coût les nouveaux protocoles et d'anticiper les problèmes qui pourront se poser dans le futur afin d'implémenter la technologie la mieux adaptée aux besoins, c'est pour cette raison que nous avons choisi la simulation comme une technique pour évaluer la performance des algorithmes étudiés.

3.2. L'outil de simulation

Il existe Plusieurs outils de simulation, comme Glomosim (global mobile simulator) et NS2 (network simulator 2), …etc. Nous avons utilisé NS-2 (Network Simulator 2) comme outil de simulation pour ces importantes raisons :

- Ø NS-2 est le simulateur de réseau le plus utilisé puisqu'il est gratuit et son code source est disponible.

- Ø NS-2 permet l'intégration des nouveaux protocoles au choix.

Cet outil fournit un ensemble d'objets TCL spécialement adaptés à la simulation de réseaux. Il est bâti autour d'un langage de programmation appelé Tcl dont il est une extension. Du point de vue de l'utilisateur, la mise en œuvre de ce simulateur se fait via une étape de programmation qui décrit la topologie du réseau et le comportement de ses composants, puis vient l'étape de simulation proprement dite et enfin l'interprétation des résultats. Cette dernière étape peut être prise en charge par un outil annexe, appelé *nam* qui permet une visualisation et une analyse des éléments simulés [11].

3.3. L'objectif de travail

Notre objectif est décrit sous la forme suivante :

- L'installation du simulateur (NS-2) et l'intégration des nouveaux protocoles («MS: *D.Manivannan et M.Singhal*» et «BCS: Briatico, Ciufoletti et Simoncini»).
- La réalisation de la simulation des algorithmes étudiés.
- L'analyse de performance des algorithmes simulés à partir des fichiers (*.dat) afin d'évaluer cette performance pour plusieurs scénarios.
- Rédaction d'un document qui aide les étudiants intéressés à ce genre de travail.

3.4. Environnement de travail

3.4.1. Environnement logiciel

Nous avons réalisé la simulation sous Linux Mandriva 2008 Spring, en utilisant la version NS-allinone-2.32. Car NS-2 est nettement plus facile à installer et à configurer sous Linux que sous Windows.

3.4.2. Environnement matériel

Notre projet a été réalisé en utilisant un ordinateur dont la configuration est suivante:

- Processeur : Intel® Core™ 2 Duo CPU 1.66 GHz.
- Mémoire : 2 Go DDR2 SDRAM.
- Disque dur : 250 GB /Go.

3.5. Réalisation de simulation

Pour rendre plus facile la compréhension de notre travail, nous avons opté pour l'utilisation des diagrammes UML :

Le premier diagramme introduit sera, celui des cas d'utilisation qui donne une vision globale du comportement fonctionnel de notre solution (figure 3.1).

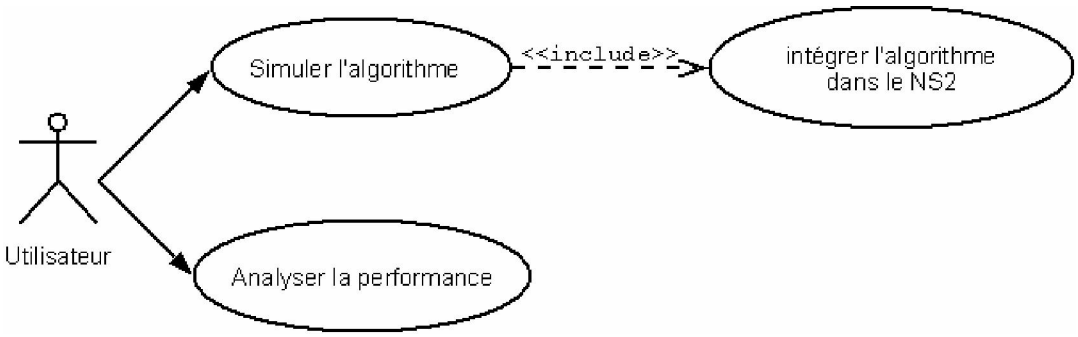


Figure 3. 1 Diagramme de cas d'utilisation «simulation de l'algorithme et analyse de performance»

Afin de réaliser la simulation des deux algorithmes (MS, BCS), nous avons intégré les deux protocoles dans le simulateur NS-2

L'intégration des deux algorithmes entraînent la création de deux fichiers sources écrits en langage c++ (*.h, *.cc), et deux autres fichiers scripts (*.tcl) pour chaque algorithme:

- le fichier *.h pour la déclaration des variables.
- le fichier *.cc pour définir l'algorithme.

Avant l'intégration de l'algorithme, il faut copier l'agent (MS ou BCS) dans le simulateur NS-2 :

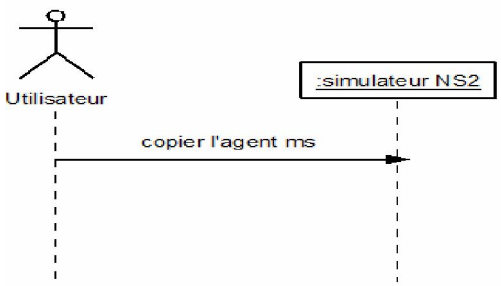


Figure 3. 2 Diagramme de séquence «copier agent dans NS-2»

Le scénario du cas d'utilisation « intégrer l'algorithme» est détaillé de la manière suivante:

- Modifier le fichier makefile (ajouter: diffusion3/apps/agent_ms/ms.o \).
- Modifier le fichier packet.h pour connaître notre agent (dans le répertoire common, ajouter : PT_MS, et name_[P_MS]="ms" ;).
- Modifier le fichier ns-default.tcl (dans le répertoire tcl/lib, ajouter : Agent/MS set packetSize_ 64 et à la suite, initialiser les variables utilisées dans le programme).
- À la fin de ces modifications, il faut recompiler NS-2 par la commande *make*. Un fichier de type objet (*.o) sera généré après la recompilation de NS-2.

Le diagramme de séquence suivant décrit le scénario « intégrer l'algorithme » (la figure 3.3)

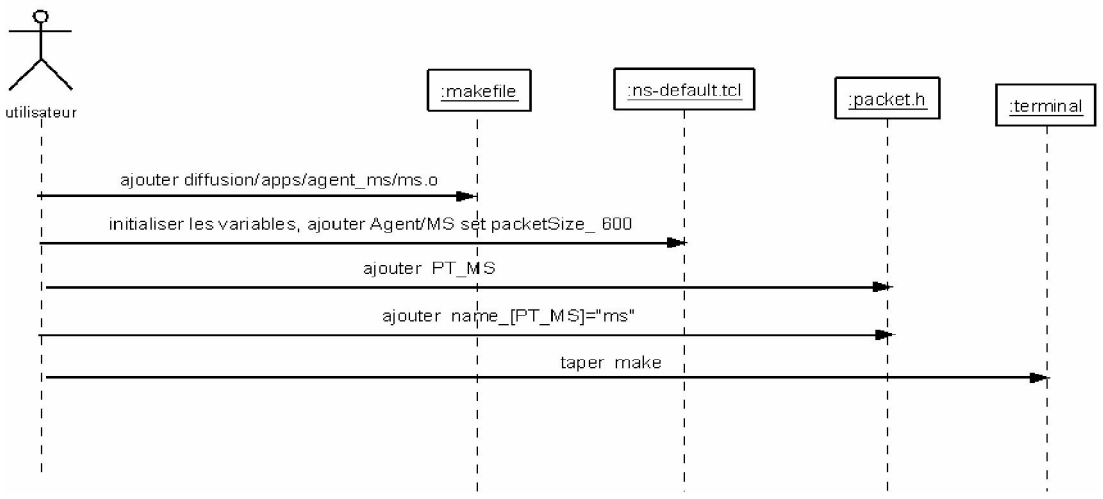


Figure 3. 3 Diagramme de séquence du scénario «intégrer l'algorithme»

3.6. Les étapes de simulation

Après l'ouverture du terminal et celle du répertoire où se trouvent les fichiers contenant le programme à exécuter :

- Ø Le fichier (*scenario.tcl*) permet à l'utilisateur de saisir les informations nécessaires pour la simulation (le temps de simulation, le nombre de processus, le nombre de messages,…) ces informations sont enregistrées dans un fichier appelé *rollback.txt*.
- Ø Le fichier *scenario.txt* devenu aussi comme résultat de premier fichier, contient les informations d'envoi des messages (le processus source, le processus destination, et le moment d'envoi).
- Ø Le fichier *basic.txt* contient l'identifiant de processus et le moment de prise d'un point de reprise spontané (basic checkpoint).
- Ø le fichier *panne.txt* contient le moment et l'identifiant de processus qui tombe en panne.
- Ø le fichier *final.txt* est un fichier global sous forme d'un tableau trié qui regroupe tous les temps nécessaires pour la simulation (temps d'envoi ou réception de message, temps de panne et temps pour prendre un point basic).

Ces fichiers (*.txt) sont nécessaires pour l'exécution des autres fichiers (*ms_sans_faut.tcl*, *bcs_sans_faut.tcl, …etc.)*, ces derniers fichiers permettent de:

- Lancer le Nam où on peut animer la simulation. Le lancement de simulation permet de voir les nœuds, les liens entre les nœuds, l'envoi et la réception des messages,…etc.
- Obtenir les résultats de la simulation pour tracer des courbes afin d'analyser la performance de l'algorithme simulé.

La figure3.4 résume les étapes de simulation d'un scénario.

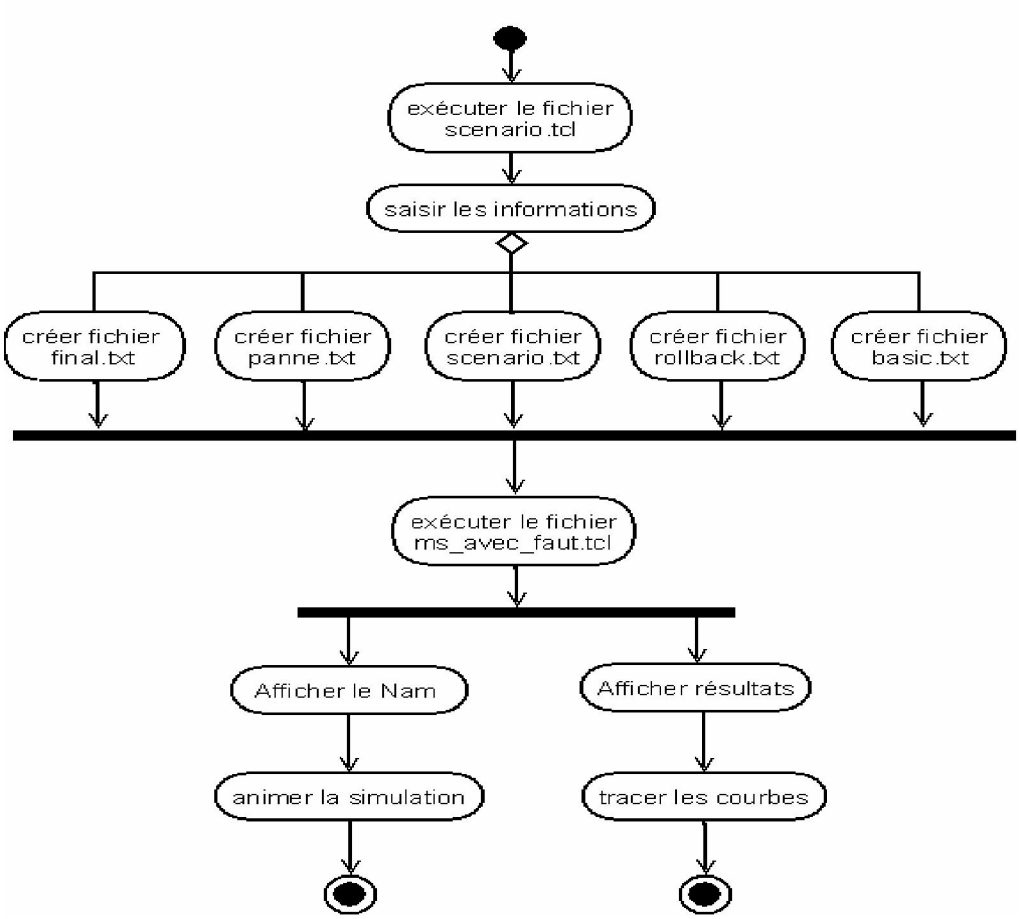


Figure 3. 4 Diagramme d'activité «les étapes de la simulation d'un scénario»

3.7. Paramètres et métriques de simulation

Afin de comparer les deux protocoles (MS et BCS) nous avons un ensemble des paramètres et des métriques.

Les paramètres	Descriptions
Nb_processus	nombre de processus
L	longueur d'intervalle qui sépare deux points de reprise
Nb_pannes	nombre de pannes

Tableau 3. 1 Les paramètres de simulation

Les métriques (sans faute)	Descriptions
F	nombre de points de reprise forcés
nb_sauv	nombre de messages sauvegardés
E	nombre de points échappés

Tableau 3. 2 Les métriques de simulation (sans faute)

Les métriques (avec faute)	Descriptions
F	nombre de points de reprise forcés
nb_ack/nb_sauv	rapport, nombre d'ACKs sur le nombre des messages sauvegardés
Deleted	nombre des points de reprise supprimés
FR	nombre de points forcés lors de recouvrement

Tableau 3. 3 Les métriques de simulation (avec faute)

3.7.1. Simulation sans faute

Scénario1 : Varier le nombre de processus (Nb_processus)

Dans ce scénario nous avons varié le nombre de processus de 4 à 64, et nous calculons le nombre de points de reprise forcés (F), le nombre de messages sauvegardés (nb_sauv) et le nombre de points de reprise échappés (Skipped (E)) en fonction de la variation de nombre de processus.

Dans ce scénario, le nombre de messages est égal à 5000, la longueur d'intervalle (le nombre d'évènements (L)) est 10 où L est la longueur d'intervalle qui sépare deux points de reprise spontanés.

Scénario2 : Varier la longueur d'intervalle(L)

L'objectif de ce scénario est d'étudier l'influence de changement de la longueur d'intervalle (L) qui sépare deux points spontanés sur le nombre de points de reprise forcés (F), nombre de message sauvegardés (nb_sauv) et le nombre des points échappés (Skipped (E)).

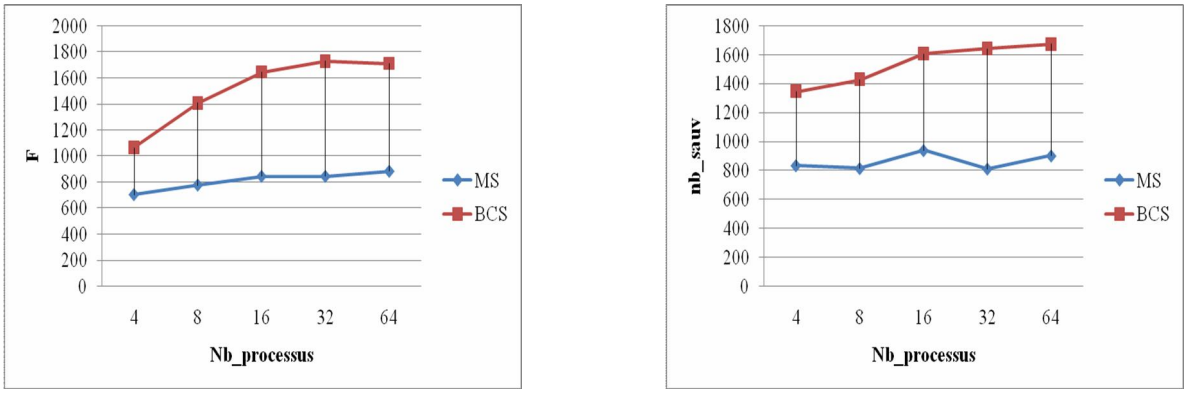
Le tableau suivant résume les différents paramètres utilisés dans cette simulation :

	Nombre de processus	Temps de simulation	Nombre de messages	Longueur d'intervalle (L)
Scénario1	4<Nb_processus<64	100	5000	10
Scénario2	10	100	5000	5< L <50

Tableau 3. 4 Variation des paramètres de simulation (sans faute)

Après la simulation, nous avons obtenu les résultats ci-dessous :

Scénario1 : Varier le nombre de processus (Nb_processus)



(a) **(b)**

Figure 3. 5 (a) F versus Nb_processus, (b) nb_sauv versus Nb_processus

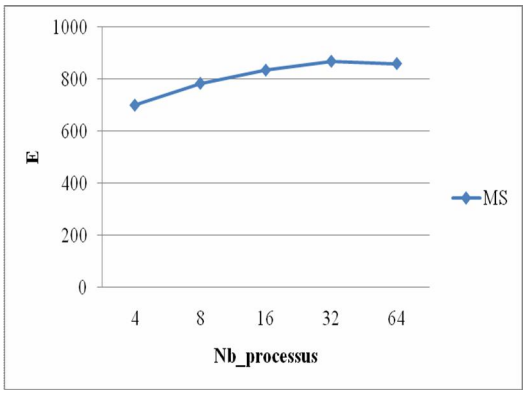


Figure 3. 6 E versus Nb_processus

La figure 3.5 (a) montre qu'avec l'augmentation de nombre de processus, le nombre de point de reprise forcés (F) augmente aussi.

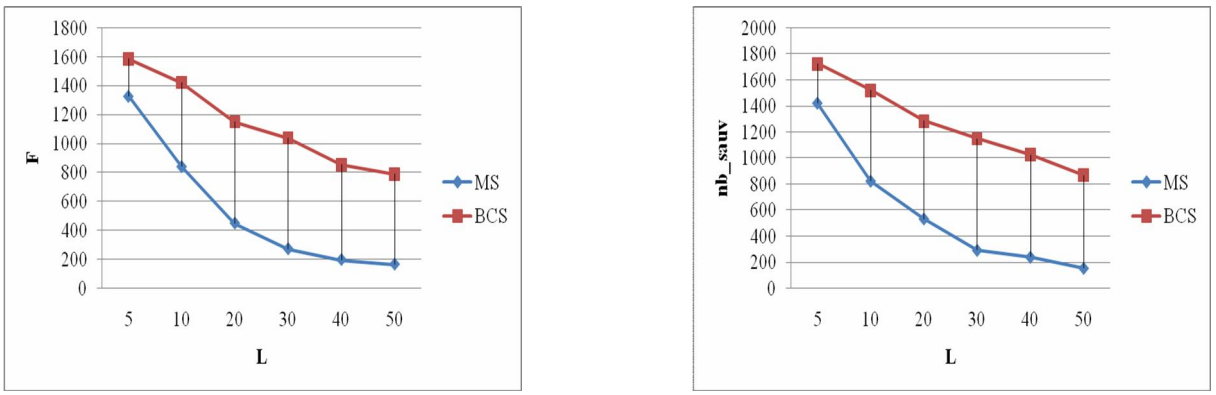
Nous voyons que la différence entre le nombre F pris par l'algorithme MS et l'algorithme BCS est très grande, égal 1500 et augmente avec l'augmentation de nombre de processus.

La figure 3.5 (b) montre la variation de nombre de messages sauvegardés en fonction de nombre de processus.

Nous remarquons que l'algorithme BCS enregistre 50% de messages échangés tandis que l'algorithme MS enregistre seulement 20%.

La figure 3.6 illustre le nombre de points de reprise spontanés n'ayant pas être pris par l'algorithme MS (E). C'est à cause de la stratégie de cet algorithme qui empêche l'apparition des points de reprise spontanés après des points de reprise forcés. Pour cela le nombre de points de reprise forcés de l'algorithme MS est inférieur à celui de l'algorithme BCS.

Scénario2 : Varier la longueur d'intervalle (L)



(a) **(b)**

Figure 3. 7 (a) F versus L, (b) nb_sauv versus L

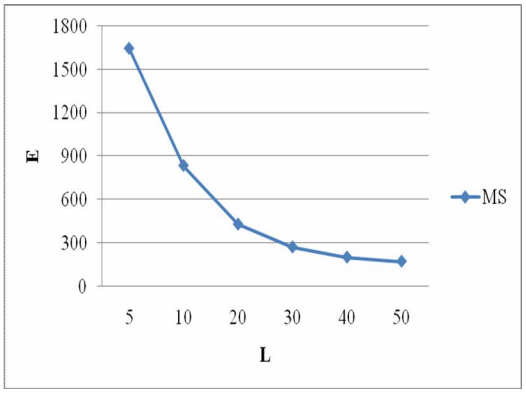


Figure 3. 8 E versus L

Pour un système composé de dix (10) processus avec un nombre de messages égal à 5000. On remarque que le nombre de points de reprise forcés diminue chaque fois que la longueur d'intervalle augmente.

Nous voyons une grande différence entre MS et BCS, par exemple pour L=5 MS prend 1500 points de reprise forcés et BCS prend 3000 points forcés.

Comme conclusion, le nombre de points de reprise forcés augmente quand le nombre de points de reprise spontanés est grand.

Dans la figure 3.7 (b) le nombre de messages sauvegardés diminue avec l'augmentation de l'intervalle (L).

- Pour L = 5 : BCS enregistre 50% des messages échangés et MS enregistre 30%, ce pourcentage diminue quand la longueur L augmente.

La figure 3.8 représente la variation de nombre de points de reprise échappés (E) en fonction de la longueur L, le nombre E diminue avec l'augmentation de L.

Ces scénarios nous ont permis de tirer les remarques suivantes :

- Pour les grands systèmes composés d'un grand nombre de processus, la performance des algorithmes est dégradée.
- La manière dont les processus prennent des points de reprise influe sur la performance des algorithmes ainsi que la surcharge de mémoire stable.

3.7.2. Simulation avec faute

3.7.2.1. Calcul de métrique F et (nb_ack/nb_sauv)

Scénario1 : Varier le nombre de processus (Nb_processus)

Dans ce scénario nous avons étudié l'influence du nombre de processus sur le nombre des points de reprise forcés (F) et le rapport (nb_ack/nb_sauv) (le nombre d'ACKs sur le nombre de messages sauvegardés).

Scénario2: Varier la longueur d'intervalle (L)

Dans ce cas, nous avons simulé selon la variation de la longueur L qui sépare deux points de reprise spontanés et présenté l'influence de cette variation sur le nombre de points de reprise forcés (F) et le rapport (nb_ack/nb_sauv).

Scénario3 : Varier le nombre de pannes (Nb_pannes)

Dans ce scénario, nous avons varié le nombre de pannes de 5 à 60 afin de voir l'influence de ce dernier sur le nombre de points de reprise forcés (F) et le rapport (nb_ack／nb_sauv).

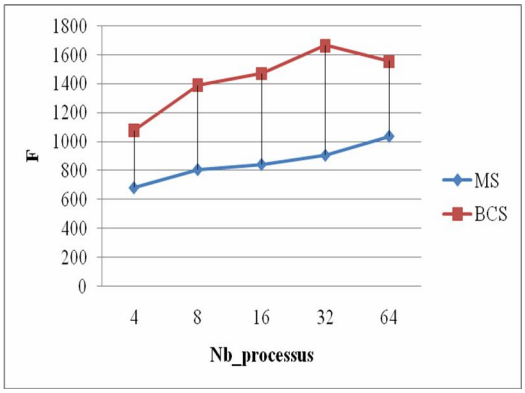
Le tableau suivant résume les différents paramètres utilisés dans la simulation :

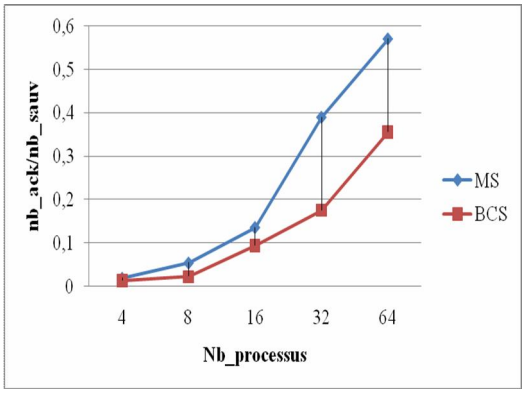
	Nombre de processus	Temps de simulation	Nombre de messages	Longueur d'intervalle (L)	Nombre de pannes
Scénario1	4<Nb_processus<64	100	5000	10	10
Scénario2	10	100	5000	5<L<50	10
Scénario3	10	100	5000	10	5<Nb_panne<60

Tableau 3. 5 Variation des paramètres de simulation (avec faute)

Scénario1 : Varier le nombre de processus (Nb_processus)

Dans ce scénario, nous nous intéressons au calcul de paramètre F et (nb_ack /nb_sauv):





(a) **(b)**

Figure 3. 9 (a) F versus Nb_processus, (b) (nb_ack/nb_sauv) versus Nb_processus

Nb_processus / **nb_ack/nb_sauv**	4	8	16	32	64
MS	16 /828	48/881	126/924	406/1040	660/1156
BCS	18/1334	34/1489	155/1649	296/1684	577/1620

Tableau 3. 6 L'influence de nombre de processus sur le rapport (nb_ack/nb_sauv)

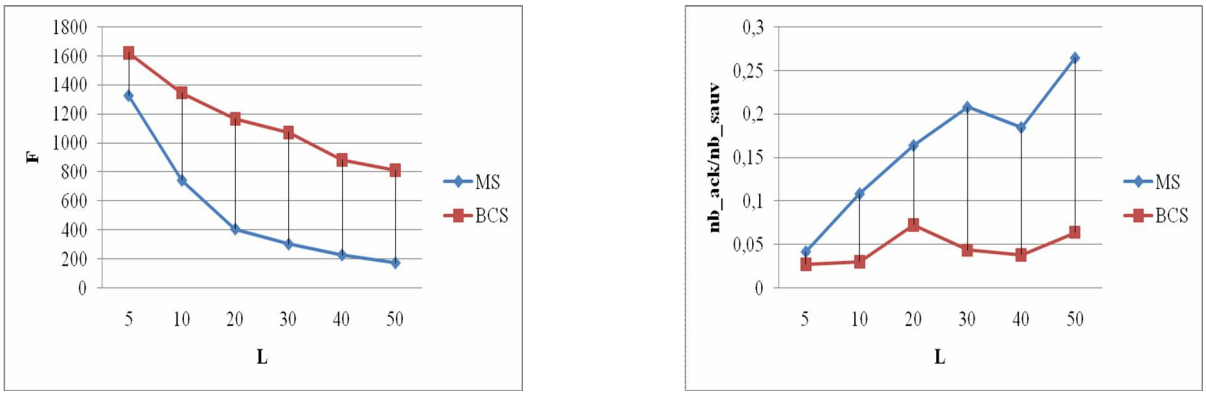
Premièrement, la figure 3.9 (a) illustre la variation de nombre de points de reprise forcés (F) en fonction de nombre de processus. Comme dans le scénario 1 de simulation sans faute, le nombre F croit avec la croissance de nombre de processus, alors en peut tirer les mêmes remarques.

Deuxième, si nous comparons la figure 3.9 avec la figure 3.5 nous pouvons remarquer que le nombre F a diminue dans ce scénario, ceci est dû peut être aux messages qui sont considérés comme étant dupliqués, et ils sont annulés bien qu'ils portent de nouvelles informations (qui peuvent éventuellement forcer les processus de prendre des points de reprise supplémentaires).

- Pour 4< Nb_processus <16, pratiquement il n'a y pas une grande différence entre MS et BCS. Ces algorithmes envoient environ de 10% des accusés de réception pour les messages enregistrés, d'une part ce pourcentage est meilleur et ne charge pas le réseau. D'autre part, 90% de messages sauvegardés seront inutiles pour le recouvrement (il occupe inutilement la mémoire).

- Pour 16< Nb_processus<64, la différence entre MS et BCS est très grande, par exemple pour Nb_processus = 32, MS envoie 40% des accusés de réception et BCS envoie seulement 20%. MS surcharge le réseau par les accusés de réception bien qu'il enregistre un nombre moins que BCS (voir le tableau 3.6).

Scénario2: Varier la longueur d'intervalle (L)



(a) **(b)**

Figure 3. 10 (a) F versus L, (b) (nb_ack/nb_sauv) versus L

L / nb_ack/nb_sauv	5	10	20	30	40	50
MS	57/1376	101/933	81/494	93/447	60/325	80/302
BCS	47/1725	48/1576	96/1317	55/1258	39/1018	64/996

Tableau 3. 7 L'influence de la longueur L sur le rapport (nb_ack/nb_sauv)

Dans ce scénario, nous étudions l'influence de changement de la longueur d'intervalle (L) qui sépare deux points spontanés (5<L<50) sur le nombre de points de reprise forcés (F) et le rapport (nb_ack/nb_sauv).

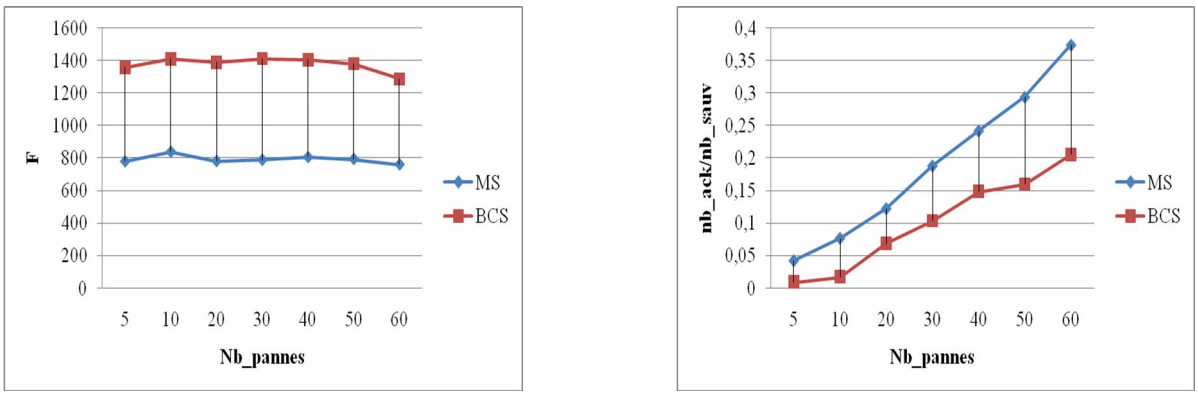
- Dans la figure 3.10(a), d'une part nous remarquons que le nombre de points de reprise forcés diminue quand la longueur L augmente pour les deux algorithmes. D'autre part, nous voyons que la différence entre ces algorithmes augmente avec l'augmentation de L, pour L=5, la différence est presque de 200 points mais pour L=50 la différence peut atteindre 600 points.

- La figure 3.10(b), le rapport (nb_ack/nb_sauv) augmente avec l'augmentation d'intervalle (L) et la différence aussi entre les algorithmes, la figure montre que pour L=50, MS enregistre 25% de messages et BCS enregistre seulement 5%.

Le tableau 3.7 montre bien que le nombre de messages sauvegardés diminue et le nombre d'ACKs augmente pour cela le rapport (nb_ack/nb_sauv) augmente aussi.

Scénario3 : Varier le nombre de pannes (Nb_pannes)

Dans ce scénario, nous avons calculé le nombre de points de reprise forcés (F) et le rapport (nb_ack/nb_sauv) en fonction du nombre de pannes (Nb_pannes) :



(a) **(b)**

Figure 3. 11 (a) F versus Nb_pannes, (b) (nb_ack/nb_sauv) versus Nb_pannes

Nb_pannes / Nb_ack/Nb_sauv	5	10	20	30	40	50	60
MS	41/969	76/989	114/931	177/941	251/1037	324/1102	418/1117
BCS	15/1652	27/1576	108 /1567	167/1608	232/1558	270/1684	324/1572

Tableau 3. 8 L'influence de nombre de pannes sur le rapport (nb_ack/nb_sauv)

Lorsque le nombre de pannes (Nb_pannes) varie de 5 à 60, nous remarquons que la croissance du nombre de pannes n'implique pas la croissance du nombre de points de reprise forcés pour les deux algorithmes (BCS et MS), car le nombre de points de reprise forcés n'est influé que par le nombre de processus ou la longueur L.

Par contre le rapport (nb_ack/nb_sauv) croit avec la croissance de nombre de pannes. Pour MS, ce rapport varié de 5% jusqu'à 35% et pour BCS, il varie de 5% jusqu'à 20%.

Selon la figure 3.13 (a), on remarque que l'algorithme BCS donne plus de points de reprise forcés (F) que l'algorithme MS donc l'algorithme MS est meilleur (plus performant) que l'algorithme BCS.

3.7.2.2. Calcul de métrique FR et Deleted

Nous calculons le nombre de points de reprise forcés pris (FR) et le nombre de points de reprise supprimés (Deleted) lors de recouvrement par rapport au nombre de processus (Nb_processus), la longueur d'intervalle (L) et le nombre de pannes (Nb_pannes) selon les différents cas de simulation (processus symétriques, un des processus est rapide et un des processus est lent).

Scénario1 : Varier le nombre de processus (Nb_processus)

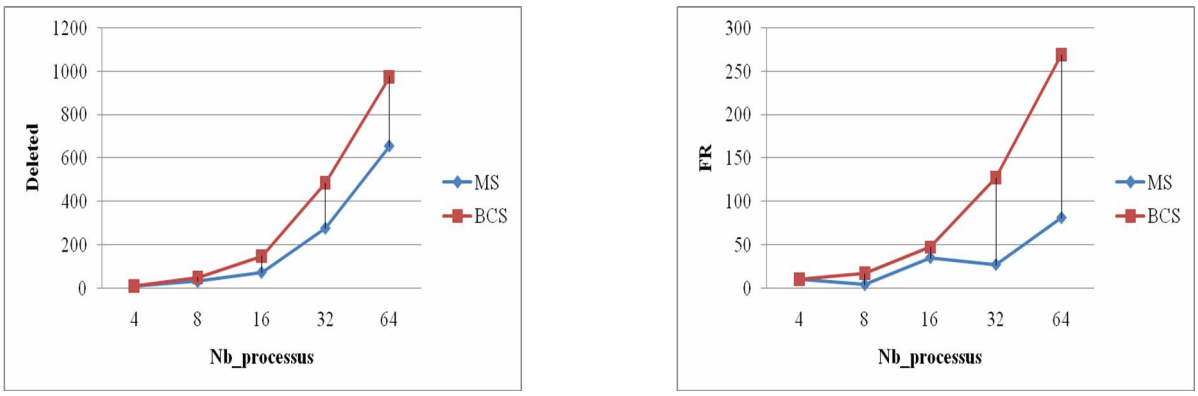
	Nombre de processus	Temps de simulation	Nombre de messages	Longueur d'intervalle (L)	Longueur d'intervalle (L')	Nombre de pannes
Processus symétrique	4<Nb_processus<64	100	*5000*	*10*	-	10
Processus rapide	4<Nb_processus<64	100	*5000*	10	5	10
Processus lent	4<Nb_processus<64	100	*5000*	10	55	10

Tableau 3. 9 Variation du nombre de processus (Nb_processus)

Remarque :

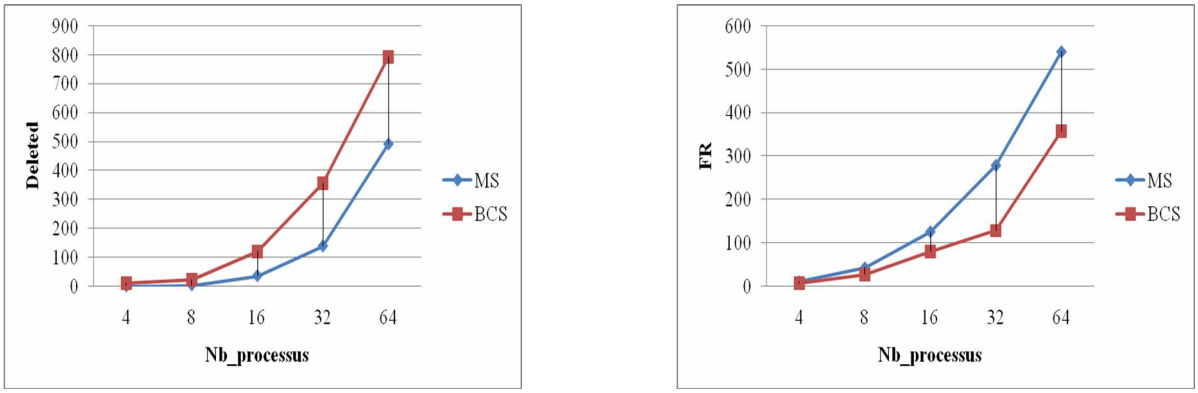
Le processus qui tombe en panne dans le cas 2 est le processus rapide.

Le processus qui tombe en panne dans le cas 3 est le processus lent.



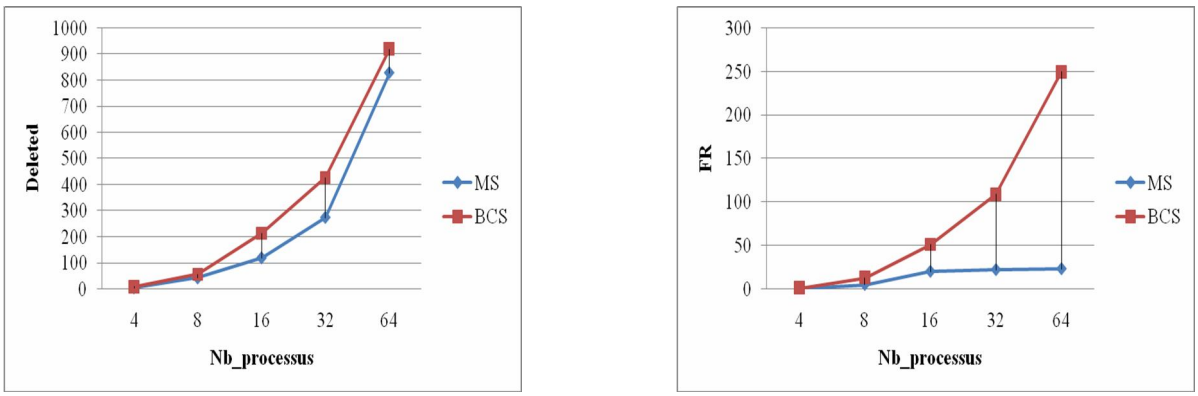
(a) **(b)**

Figure 3. 12 (a) Deleted versus Nb_processus, (b) FR versus Nb_processus (symétrique)



(a) **(b)**

Figure 3. 13 (a) Deleted versus Nb_processus, (b) FR versus Nb_processus (rapide)



(a) **(b)**

Figure 3. 14 (a) Deleted versus Nb_processus, (b) FR versus Nb_processus (lent)

Pour le premier cas, où on a des processus symétriques :

- Le nombre de points de reprise supprimés lors de recouvrement augmente avec l'augmentation de nombre de processus (pour les deux algorithmes) (figure 3.11 (1)).
- Pour 4<Nb_processus<16, il n'a y pas une grande différence entre MS et BCS. Mais pour 16< Nb_processus <32, BCS supprime un grand nombre de points par rapport MS. Ce qui implique un retour à un état plus ancien par rapport à l'état courant.

Dans le deuxième cas, (ce processus est plus rapide que les autres) :

- Nous pouvons tirer les mêmes remarques.
- En plus nous voyons une différence plus grande que celle de premier cas entre BCS et MS, ce qui est l'inverse pour le troisième cas où nous avons un processus lent (il n'a y pas une grande différence).

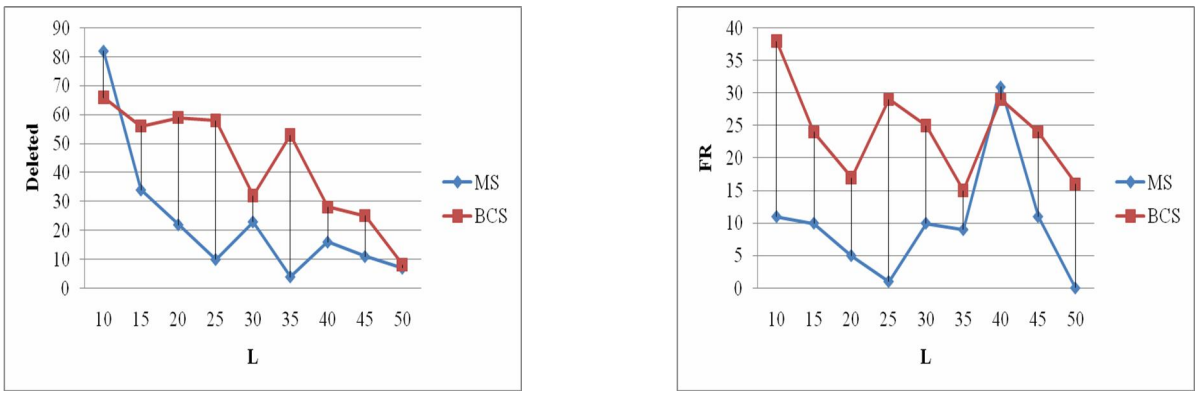
En parallèle nous avons calculé le nombre de points de reprise forcés à cause de recouvrement (FR) en fonction de nombre de processus pour les trois cas, quand le nombre de processus augmente le nombre FR augmente aussi (pour les trois cas) sauf pour le deuxième cas, nous voyons que MS a un grand nombre par rapport à BCS.

Plus que cela, si nous voyons tous les graphes parallèlement nous peuvons dire que BCS supprime des points de reprise et prend des points forcés plus que MS, c.-à-d. que MS ne charge pas la mémoire, en plus il ne retourne pas à un état plus ancien.

Scénario2: Varier la longueur d'intervalle (L)

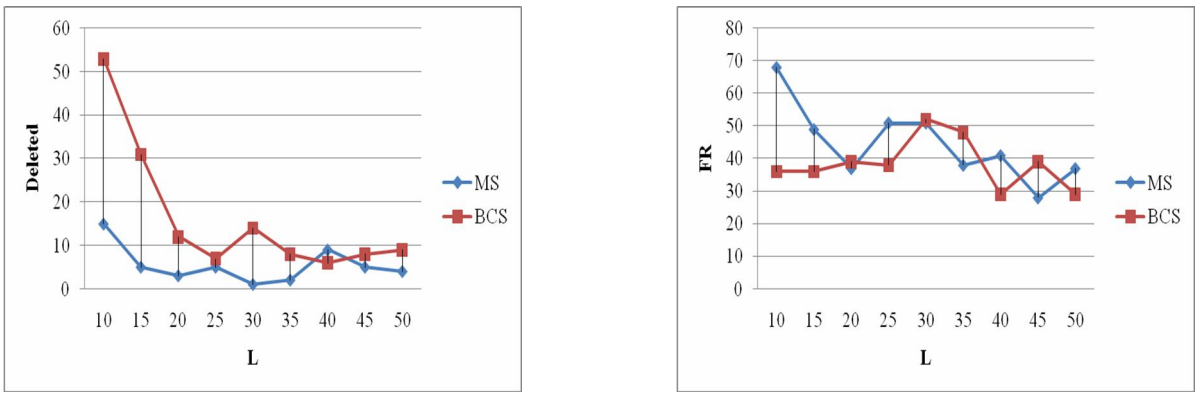
	Nombre de processus	Temps de simulation	Nombre de messages	Longueur d'intervalle (L)	Longueur d'intervalle (L')	Nombre de pannes
Processus symétrique	10	100	*5000*	5<L<50	-	10
Processus rapide	10	100	*5000*	10<L<50	5	10
Processus lent	10	100	*5000*	10<L<50	55	10

Tableau 3. 10 Variation de la longueur d'intervalle (L)



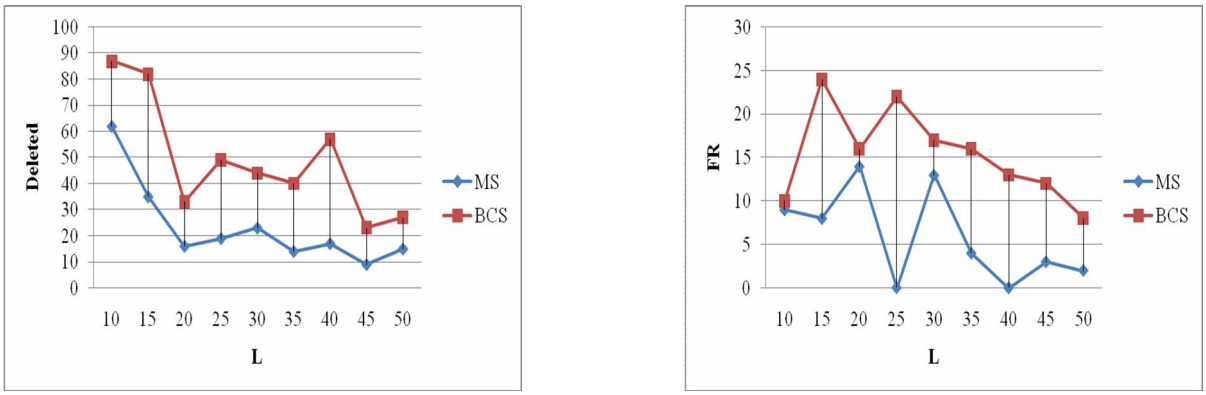
(a) **(b)**

Figure 3. 15 (a) Deleted versus L, (b) FR versus L (symétrique)



(a) **(b)**

Figure 3. 16 (a) Deleted versus L, (b) FR versus L (rapide)



(a) **(b)**

Figure 3. 17 (a) Deleted versus L, (b) FR versus L (lent)

L'objectif de ce scénario est d'étudier l'influence de la longueur L sur deleted et FR.

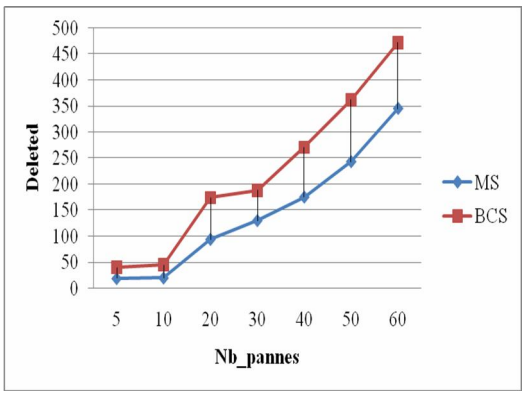
Nous remarquons que pour certaines valeurs de L, la différence entre MS et BCS augmente et que pour les autres elle diminue, on peut remarquer aussi la différence pour le nombre Deleted entre MS et BCS est grande, la différence est petite pour le nombre FR (surtout pour le cas 1).

Pour le deuxième cas, la figure 3.16 montre que les valeurs de nombre FR sont supérieures aux valeurs de Deleted, par contre dans le troisième cas, on nous voyons que les valeurs de Deleted sont plus grandes que les valeurs de FR. Ceci est expliqué par le fait que le processus qui tombe en panne est le processus asymétrique.

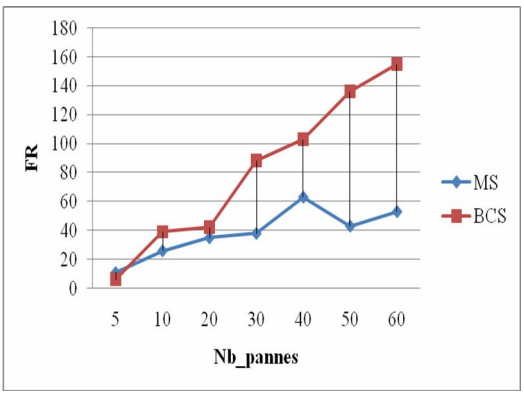
Scénario3 : Varier le nombre de pannes (Nb_pannes)

	Nombre de processus	Temps de simulation	Nombre de messages	Longueur d'intervalle (L)	Longueur d'intervalle (L')	Nombre de pannes
Processus symétrique	10	100	*5000*	10	-	5<Nb_pannes<60
Processus rapide	10	100	*5000*	10	5	5<Nb_pannes<60
Processus lent	10	100	*5000*	10	55	5<Nb_pannes<60

Tableau 3. 11 Variation du nombre de pannes (Nb_pannes)

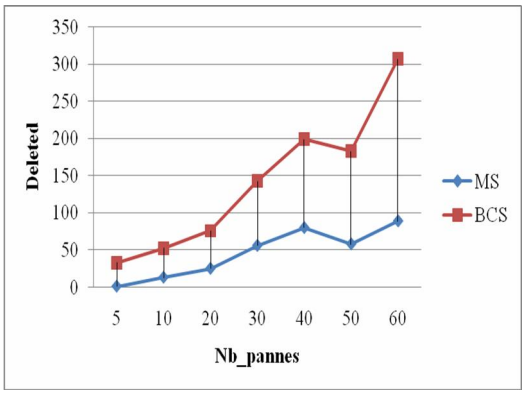


(a)

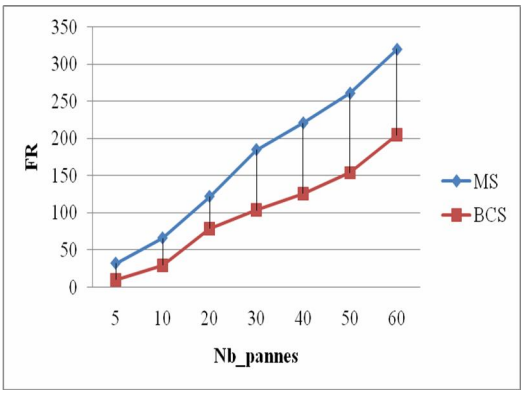


(b)

Figure 3. 18 (a) Deleted versus Nb_pannes, (b) FR versus Nb_pannes (symétrique)

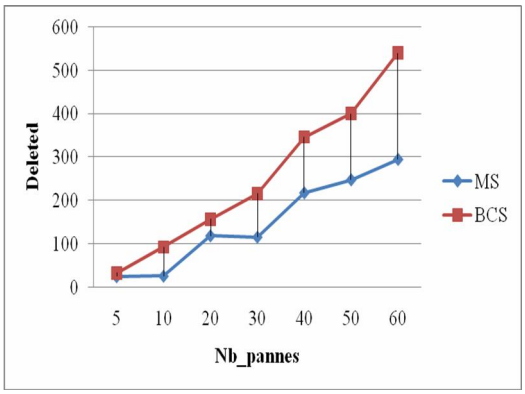


(a)

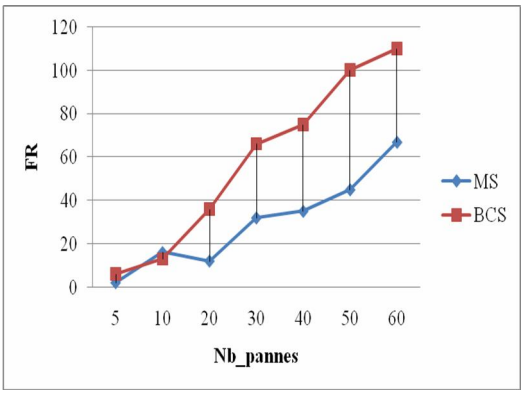


(b)

Figure 3. 19 (a) Deleted versus Nb_pannes, (b) FR versus Nb_pannes (rapide)



(a)



(b)

Figure 3. 20 (a) Deleted versus Nb_pannes, (b) FR versus Nb_pannes (lent)

Dans ce scénario, nous étudions l'influence du nombre de pannes (Nb_pannes) sur FR et Deleted.

Pour les trois cas, les figures montrent que l'augmentation de nombre de pannes implique l'augmentation de nombre de points de reprise forcés (FR) et le nombre de points de reprise supprimés (Deleted) lors de recouvrement.

Pour le premier cas (figure 3.18), la différence entre MS et BCS arrive jusqu'à 150 points supprimés et 50 points forcés (BCS supprime et prend des points plus que MS).

Dans le deuxième cas (figure 3.19), pour un nombre de panne égal à 60, BCS supprime 300 points et MS supprime juste 100 points, par contre MS prend plus de points que BCS et la différence atteint jusqu'à 100 points pour un nombre de panne égale 60.

Pour le troisième cas (figure 3.20), nous pouvons tirer les mêmes remarques de premier cas. En générale on peut dire que MS fait un retour vers un état plus récent que celui de BCS.

Nous concluons que le nombre de pannes influe sur la performance des algorithmes plus que le nombre de pannes augmente.

3.8. Conclusion

La simulation nous a permet de tirer les remarques suivantes :

- Pour les grands systèmes, constituant de grands processus, la performance de deux algorithmes est dégradée.
- L'algorithme MS enregistre un petit nombre de messages échangés par rapport aux BCS avec une différence de 30%.
- La stratégie adoptée par MS influx sur le nombre de points de reprises forcés pris.
- Lors de recouvrement, le protocole MS envoie des accusés de réception plus que le protocole BCS, il surcharge le réseau. D'autre part, les messages enregistrés par BSC occupent inutilement le support de stockage.

- Pour les systèmes qui ont un risque d'avoir plusieurs fautes, la performance des algorithmes est dégradée car ils toujours prisent un grand nombre de points supprimés (grande partie de travail sera supprimée à cause de retour).

- La manière de prise de points de reprises spontanées a un impact sur la performance des deux algorithmes.

Conclusion générale

Le recouvrement arrière permet aux processus de système un retour à un état précédent en cas de panne. Parmi les techniques existantes celles qui sont basées sur les points de reprise.

Dans ce mémoire, nous avons étudié et simulé deux algorithmes (MS et BCS) de points de reprise de la classe CIC avec un mécanisme de recouvrement en arrière.

La simulation nous a permis de tirer les remarques suivantes :

- la performance des algorithmes est dégradée dans les grands systèmes composés de plusieurs processus.
- Les algorithmes simulés enregistrent un grand nombre de messages dans la mémoire stable, en conséquence envoient un grand nombre des accusés de réception qui peut surcharger le réseau.
- Dans le cas de système ayant un grand nombre de pannes, ces algorithmes dégradent leur performance car ils augmentent le nombre de points de reprise forcés et suppriment un grand nombre de points de reprise enregistré, se qui implique un retour vers un état plus ancien que l'état courant.
- La manière et la stratégie adoptées pour la prise des points de reprise influent sur la performance des algorithmes.

Nous proposons comme perspectives :

- Améliorer les algorithmes pour traiter le cas de N pannes.
- Minimiser le nombre de processus participant dans le recouvrement arrière.
- Simuler les algorithmes de recouvrement arrière pour le cas de réseaux mobile.
- Améliorer les algorithmes pour traiter les autres types des pannes.
- Minimiser le surcoût des algorithmes et trouver un compromis entre le nombre de points de reprise forcés et le nombre de points de reprise supprimés.

Bibliographie

[1]. Abdelhafidi.Z. *Points de reprise dans les systèmes répartis Etude basée sur la simulation des protocoles CIC assurant la propriété RDT.* Thèse Magistère de l'université Amar Telidji-Laghouat Spécialité informatique, p11-27, 2007.

[2]. Amer.M, Yahyaoui.A. *Etude et simulation des protocoles de routage dans les réseaux mobiles sans fil (AD HOC).* Thèse d'ingénieur de l'université Amar Telidji-Laghouat Spécialité informatique, PFE 2008.

[3]. C.Delbé. *Tolérance aux pannes pour objets actifs asynchrones : protocole, modèle et expérimentations.* Thèse de doctorat de l'université de Nice - Sophia Antipolis Spécialité informatique, 2007.

[4]. D. Briatico, A. Ciufoletti et L. Simoncini. "A Distributed Domino-Effect Free Recovery Algorithm. Proc, 4th IEEE Symposium on Reliability in Distributed Software and Database Systems." 1984. 207-215 pages.

[5]. D.Manivannan, M.Singhal. "A low-overhead recovery technique using Quasi-Synchronous checkpointing." The 16th International Conference on Distributed Computing Systems. Computer and Info.science Dept. The Ohio state University Columbus, OH 43210, May 1996. 100-107 pages.

[6]. E. N. Elnozahy, L. Alvisi, Y.-M. Wang, D. B. Johnson. *A survey of rollback recovery protocols in message-passing systems.* ACM Comput. Surv., 34(3) :375–408, 2002.

[7]. Florin.G. *La tolérance aux pannes dans les systèmes répartis.* Laboratoire CEDRIC CNAM.

[8]. Hamouma.M. *Évaluation de mécanismes de détection de défaillances dans un système réparti asynchrone.* Mémoire de Magistère En Informatique de l'université A/Mira de Bejaia, 2004 / 2005.

[9]. Kebir.M, Alliliche.A. *Etude comparative des protocoles de points de reprise de type CIC.* Thèse d'ingénieur de l'université Amar Telidji-Laghouat Spécialité informatique, PFE 2008.

[10]. Krouba.M, Allaoui.T. *Etude des systèmes répartis et réalisation d'un simulateur des algorithmes répartis d'exclusion mutuelle.* Thèse d'ingénieur de l'université Amar Telidji-Laghouat Spécialité informatique, p15-16, 2004.

[11]. P. Anelli, E. Horlait. *NS-2: Principes de conception et d'utilisation (Manuel NS-2).* UPMC - LIP 6 : Laboratoire d'Informatique de Paris VI, 1999.

[12]. S.Jafar. *Programmation des systèmes parallèles distribués : tolérance aux pannes, résilience et adaptabilité.* Thèse pour obtenir le grade de docteur de L'INPG (Institut National Polytechnique de Grenoble) Spécialité : "Informatique : Systèmes et Logiciels", le 30 Juin 2006.

Annexe I

Le simulateur NS-2

1. Définition : [11]

NS-2 (Network Simulator-2) est un outil logiciel de simulation de réseaux informatiques. Il est principalement bâti avec les idées de la conception par objets, de réutilisabilité du code et de modularité. Il est devenu aujourd'hui un standard de référence en ce domaine. C'est un logiciel dans le domaine public disponible sur l'Internet. Ce logiciel est exécutable tant sous Unix que sous Windows.

Application	Web, ftp, telnet, générateur de trafic (CBR, ...)
Transport	TCP, UDP, RTP, SRM
Routage	Statique, dynamique (vecteur distance) et routage multipoint (DVMRP, PIM)
Gestion de file d'attente	RED, DropTail, Token bucket
Discipline de service	CBQ, SFQ, DRR, Fair queuing
Système de transmission	CSMA/CD, CSMA/CA, lien point à point

Tableau A1. 1 Principaux composants

Il présente principalement les informations permettant de construire un modèle de simulation et montre comment utiliser l'interpréteur. Il n'y a que peu d'explications sur la création de nouveaux composants et sur le fonctionnement du simulateur.

L'utilisateur décrit le modèle de son réseau, c'est-à-dire les éléments qui le constituent, dans le langage OTCL (une extension objet du langage Tcl). Ce fichier est ensuite passé au simulateur proprement dit. Il est possible de rajouter des objets réseau au simulateur, en les programmant en C++, les principes de l'OTCL et les explications sur le mécanisme qui permet à un programme C d'utiliser un interpréteur Tcl.

Tcl (Tool Command Language) offre des structures de programmation telles que les boucles, les procédures ou les notions de variables. Il y a deux principales façons de se servir de Tcl: comme un langage autonome interprété ou comme une interface applicative d'un programme classique écrit en C ou C++. En pratique, l'interpréteur Tcl se présente sous la forme d'une bibliothèque de procédures C qui peut être facilement incorporée dans une application. Cette application peut alors utiliser les fonctions standards du langage Tcl mais également ajouter des commandes à l'interpréteur.

2. Principe de fonctionnement [11]:

NS-2 est composé de plusieurs classes structurées sous une forme arborescente. Ces classes sont utilisées par le simulateur pour le fonctionnement de ses composantes. Les classes visibles au niveau de l'interpréteur comprennent une déclaration dans la classe Tclcl qui est la racine de toutes les autres classes à la fois dans l'arborescence compilée et interprétée.

Les principales classes sont :

- **Applicatio**n : La classe mère de toutes les applications (ftp, telnet)
- **Agent** : La classe mère de tous les protocoles qui tournent au niveau 3 ou 4 (TCP, UDP, SRM, RIP, OSPF)
- **Queue :** La classe mère de tous les buffers (DropTail, RED)
- **LinkDelay** : Simule un délai de propagation. Avec la classe Queue, elle permet de définir la classe des liens du réseau.
- **Packet** : La classe des paquets. Cette classe définit un pointeur vers l'en-tête du paquet (Header) et un pointeur vers le champ de données.
- **TimerHandler**: La classe mère de tous les timers utilisés par les différents éléments du réseau. Chaque objet du simulateur (à l'exception de l'application de destination) dispose d'un pointeur vers l'objet suivant auquel le paquet doit être fourni après être traité. Pour passer le paquet à un objet, il suffit d'appeler sa fonction recv avec un pointeur vers le paquet comme paramètre.
- **Le Scheduler:** Le simulateur dispose d'un Scheduler qui permet l'exécution des événements dans le futur (délivrance d'un paquet à un objet après un certain temps, exécution d'une action lors de l'expiration d'un temporisateur, etc.).

3. Structure du simulateur:

3.1. Le code [11]:

La manipulation des bits, des entêtes de paquet, mais aussi de pouvoir implémenter des algorithmes capables de parcourir plusieurs types de données, donc la création rapide et efficace des objets et variables manipulés lors de la simulation. Pour cette tâche une rapidité

d'exécution est requise, et est importante (*la découverte des erreurs, la correction, recompilation et en fin réexécution est moins importante*), ceci est offert part le C++ qui gère le code avec l'interpréteur Tcl. D'autre part la configuration des objets et la gestion des évènements, ou autre l'exploration d'un grands nombre de scénarios (*Changement du modèle et réexécution*), donc le temps d'itération est plus important par rapport à la rapidité d'exécution, ceci par contre est offert par l'OTCL, qui le permet d'une manière interactive. Notons aussi que le tclcl permet aux objets et variables d'apparaître et d'être utilisé par les deux langages.

3.2. L'interface TCL/OTCL [11]:

L'ensemble des actions spécifie un scénario de simulation donné est passé à la simulation sous forme d'un fichier. Le langage TCL ou OTCL sont utilisés pour l'écriture de ce fichier.

3.2.1. TCL (Tool Command Language):

Tcl (Tool Command Language) est un langage interprété portable sur Mac, PC et station Unix. Tcl est un langage script non typé, Tcl fut créé en 1990 par John OUSTERHOUT à l'Université de Berkeley. C'est un langage de "collage" pour attacher ensemble plusieurs applications.

Tcl est un langage de commande comme le Shell UNIX mais qui sert à contrôler les applications. Tcl offre des structures de programmation telles que les boucles, les procédures ou les notions de variables. Il y a deux principales façons de se servir de Tcl:

Comme un langage autonome interprété ou comme une interface applicative d'un programme classique écrit en C ou C++. En pratique, l'interpréteur Tcl se présente sous la forme d'une bibliothèque de procédures C qui peut être facilement incorporée dans une application. Cette application peut alors utiliser les fonctions standards du langage Tcl mais également ajouter des commandes à l'interpréteur.

Programmation Tcl :

A. Variables Tcl :

En Tcl, les variables sont non typées, et peuvent être suivant le cas, considérées comme chaînes de caractères (le cas le plus large), listes (ensemble ordonné d'éléments non

typés qui peuvent aussi être eux mêmes des listes), entiers ou flottants. Elles peuvent aussi être des tableaux à une ou plusieurs dimensions. Une même variable peut être considérée de types différents suivant l'opération effectuée.

L'accès à une variable *nom_variable* se fait par *$nom_variable*.
L'affectation se fait par la commande *set*.

Exemple :

*% **set** a 18* # la variable *a* reçoit "18".
*% **puts $**a* # affichage du contenu de la variable *a*.

B. Structures de programme :

- Une conditionnelle s'écrit sous la forme :

% ***if*** condition bloc # l'écriture de condition
La condition doit, contrairement au C, être un "vrai" booléen (0 pour faux ou 1 pour vrai). Si la condition est composée de plus d'une opération arithmétique, il faut la mettre sous la forme *[expr* condition*]* qui force une évaluation mathématique et/ou booléenne de la condition.

- Une boucle " pour " s'écrit sous la forme :

*% **for*** initialisation condition incrémentation bloc # l'écriture de boucle
De la même manière que les tests, une boucle "pour" s'écrit sur une même ligne, il ne peut y avoir des sauts de ligne qu'à l'intérieur d'accolades.

Exemple :

*% **for** {**set** i 1} {$i<=10} {**incr** i} {*
***puts** $i #affichage des nombres de 1à 10.*
}

- Le branchement à choix multiples s'écrit de la manière suivante :

Switch chaîne $modèle_1$ $bloc_1$ ... $modèle_n$ $bloc_n$ # structure de branchement
Ou bien
Switch chaîne $modèle_1$ $bloc_1$ ... $modèle_n$ $bloc_n$ ***default*** bloc default avec bloc default qui est exécuté si aucun modèle ne correspond à la chaîne.

Exemple :

```
% Switch $a {
        0{puts "rollback"}
        1{puts "scenario"}
default {puts " fin"}
}
```

- Les fonctions permettent d'introduire une modularité (relativement faible) dans un programme Tcl. La définition d'une fonction se fait de la façon suivante:

*% **proc*** nom paramètres corps. En fait elle s'écrira le plus souvent:

***Proc** nom_proc {param$_1$ param$_2$ ... param$_n$} {*
bloc
}

Les fonctions sont toutes censées renvoyer une valeur (qui peut être une chaîne vide). Dans le cas où une instruction ***return*** est rencontrée lors de l'évaluation de la fonction, son évaluation est stoppée et la valeur suivant le ***return*** est renvoyée.

Toutes les variables affectées à l'extérieur des fonctions sont considérées comme globales. Ainsi l'accès en lecture ou en écriture à une variable à l'intérieur d'une fonction se fait sur une variable locale par défaut. Pour accéder à une variable globale (exemple : nb_node_) à l'intérieur d'une fonction, il faut déclarer celle-ci comme globale à l'intérieur de la fonction (de préférence avant tout accès à cette variable) avec le mot clé ***global*** (par exemple *global nb_node_*).

- Les commandes d'entrées sorties sont les suivantes :

Open gets seek flush close read tell puts file

La commande open retourne un identifiant qui sera utilisé lors des appels aux autres commandes.

Exemple:

***set** f [open "scenario.txt" "r"]* * **set** table [read $f]* * # instructions* * **close** $f*	***set** f [open "scenario.txt" w]* * # instructions* ***puts** $f* ***close** $f*

puts permet d'écrire dans un canal déterminé (défaut # sortie standard)

3.2.2. OTCL (Object Tool Command Language):

A. L'interface OTCL :

L'ensemble des actions spécifiques à un scénario de simulation donné est passé au simulateur sous forme d'un fichier. Le langage OTCL est utilisé pour l'écriture sur cette interface. OTCL est un langage de programmation orienté objet offrant les mêmes fonctionnalités que C++.

Un interpréteur OTCL est ajouté au simulateur pour traduire les instructions, OTCL passées par l'interface au simulateur en des instructions C++.Lorsque le simulateur est appelé, une hiérarchie de classes similaires à celle au niveau C++ est créée au niveau OTCL.

L'utilisateur programme sur l'interface comme si le simulateur a été écrit en OTCL. En particulier, il peut instancier des classes existantes ou bien définir de nouvelles classes.

- objets crées par la commande new {}.
- set ns [new Simulator] #création d'instance de Simulator.
- objets topologiques (nœud et lien) ont une syntaxe différente: on n'utilise pas new mais une procédure de la classe Simulator.
- la classe est uniquement OTCL.
- référence des instances mémorisée dans un tableau de la classe Simulator.
- $ns node #création d'un nœud.
- objets manipulables avec les méthodes (de leur classe ou de leurs super-classes).
- les affectations sur variables membres se font par la commande set{}.
- <référence instance> set <instance variable > <valeur>.
- héritage indiqué par l'option -superclass <nom de la super-classe> après la directive Class.
- Class class1 -superclass {class2 class3}.
    - ü lors de création d'objets en NS, la première ligne est souvent *eval $self next $args* la méthode *next* appelle la méthode de même nom dans l'hiérarchie de classe. Elle effectue une agrégation de méthode.

B. Liens C++ et OTCL :

- Faciliter l'interaction entre hiérarchie interprétée (Tcl) et hiérarchie compilée (C++).
    - ü NS fournit la classe Tcl.
    - ü documentation de l'API C de OTCL.

- Construire une application avec un interpréteur Tcl=inclure une bibliothèque Tcl qui définit les commandes de base de Tcl dans l'application.
- L'interpréteur effectue l'analyse syntaxique et appelle la fonction C correspondant à la commande Tcl.
- Ajouter une commande Tcl consiste à établir un lien entre un mot (nom de commande) et une fonction C.
- La fonction C est définie dans le code source de l'application.
- Au démarrage l'application procède dans son main () aux initialisations nécessaires et passe la main à l'interpréteur.
- L'application passe en mode interactif: à chaque commende tapée par l'utilisateur, la fonction C correspondante est appelée afin de réaliser la commande demandée.

C++	**OTcl**
Unique déclaration de classe	Méthodes attachées à un objet ou à une classe
	Méthodes appelées l'objet en préfixe
Constructeur/Destructeur	init{}/destrooy{}
this(C++)	$self(Otcl) (le $self est explicite)
	Méthodes sont tout le temps « virtual » : Détermination de la méthode à appeler effectuée à l'exécution
	Méthodes non définies dans la classe appelées explicitement avec l'opérateur de portée « : : »
Appel de méthode de même nom de classe parant	Méthodes implicitement applées dans l'arbre d'héritage par « $self next »
L'héritage multiple possible	
Attributs méthodes void XTP : : Send (char *data)	Déclaration de variable d'instance : instvar Déclaration méthode de classe : instproc XTP instproc Send {data}

Tableau A1. 2 Liens C++ vers OTCL

3.3. Simulateur [w1]:

La simulation est configurée, contrôlée et exploitée via l'interface fournie par la classe OTCL Simulator. Cette classe joue le rôle d'API (Application Programming Interface). Elle

n'existe que dans l'interpréteur. Un script de simulation commence toujours par créer une instance de cette classe par la commande:

***Set** ns [new Simulator*]

L'objet Simulator contiendra une référence sur chaque élément de la topologie du réseau simulé et sur les options mises pour la simulation comme par exemple les noms des fichiers de trace. La topologie du réseau est construite avec des instances des classes OTCL Node et Link.

L'initialisation du Simulator par la méthode *init {}* initialise le format des paquets de la simulation et crée un ordonnanceur.

3.4. Les outils de simulateur NS-2 [w1]:

3.4.1. Xgraph:

Xgraph est un autre utilitaire utilisé par NS-2 et qui permet lui aussi de fournir un compte rendu graphique Pour tracer les résultats d'une simulation sous forme de courbes.

3.4.2. NAM:

Lors de la simulation, les résultats sont consignés dans un fichier de trace que l'outil de visualisation NAM (Network AniMator) va permettre de traiter basé sur le langage Tcl/Tk (Tool Command Language/ToolKit*)*. Dans notre programme, l'outil de visualisation est appelé directement à la fin de la simulation. Deux éléments intéressants sont proposés à la visualisation : un dessin de la topologie du réseau étudié, et une visualisation dynamique du déroulement du programme dans le temps.

On peut ouvrir le NAM avec la commande « nam-file », le nom d'un dossier de trace de NAM qui a été produit par NS ou vous pouvez l'exécuter directement hors du script de simulation de Tcl pour la simulation que vous voulez visualiser. Voici la fenêtre NAM (voir la figure A1.3):

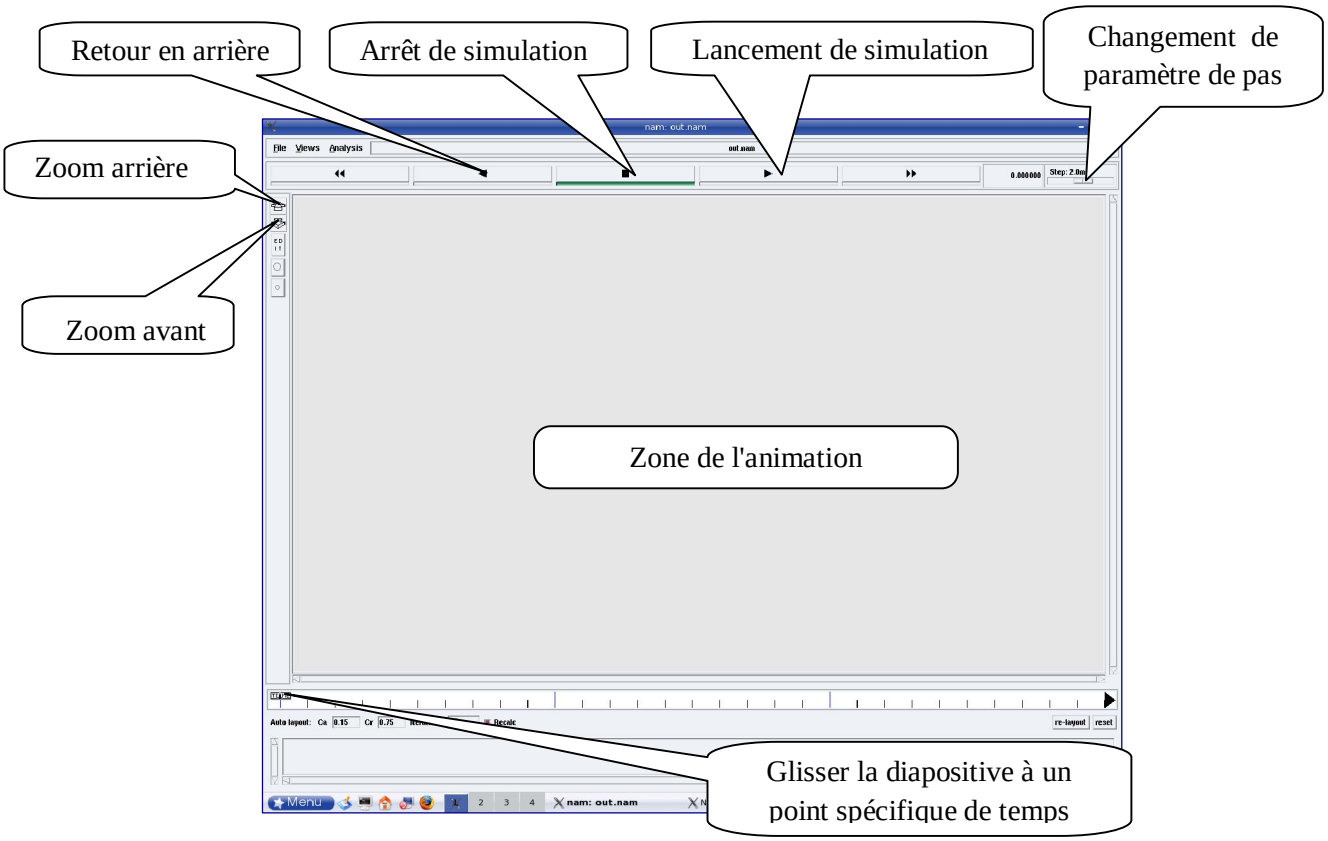


Figure A1. 1 La fenêtre de NAM

3.5. L'interprétation [11]:

NS fournit deux moyens pour extraire les données de la simulation:

3.5.1. Trace:

La trace enregistre dans un fichier les changements d'états d'un paquet ou de valeur d'une variable. Il existe deux types de traces: les traces effectuées à partir d'une file d'attente d'un lien et les traces de variables.

- Pour la première (*queue*), chaque type d'événement pouvant faire l'objet d'un enregistrement se définit par une sous-classe de Trace. La classe Trace consiste à réception d'un paquet à effectuer l'écriture des principales caractéristiques du paquet dans le fichier et à passer le paquet à l'objet aval. Une ligne de trace commence par une lettre indiquant le type de trace afin de différencier chaque sous-classe de Trace:
  - \+ mise en file d'attente.
  - \- sortie de la file d'attente.
  - d suppression de la file d'attente (dropped).
  - r réception au nœud (receive).

- s envoi au nœud (send).

Ø Pour la deuxième (*variable*), dans NS, les variables pouvant être tracées sont dérivées de la classe de base TracedVar. Deux classes sont fournies: TracedInt et TracedDouble. Pour être exploitables, ces informations doivent être traitées par un filtre qui analyse chaque ligne du fichier pour donner des informations interprétables. NS2 ne propose pas ce filtre et l'utilisateur est libre de choisir le programme approprié pour lire le fichier texte. Nous avons choisi d'utiliser Awk. Il nous transformer cet ensemble de données brutes en un sous-ensemble intéressant pour la compréhension de la simulation.

La commande *awk* s'utilise de la façon suivante :

***awk** –f fichier_script fichier_trace.*

3.5.2. Moniteur:

Le moniteur est un objet pouvant faire des calculs sur différentes grandeurs tel que le nombre de paquets ou d'octets arrivés, un moniteur utilise des objets observateurs dits sniffeurs (snoop) qui sont insérés dans la topologie réseau. Le rôle de ces petits sniffeurs est de faire remonter les informations d'états du réseau au moniteur. Celui-ci est un point de ralliement de ces informations, où des calculs sont effectués. Montre l'assemblage des composants du lien lorsqu'il y a un QueueMonitor. La classe QueueMonitor est définie dans le fichier *queue-monitor.cc.* Le moniteur est un composant qui effectue des calculs relatifs au lien. Dans un souci de performance, il est préférable d'utiliser des traces et de procéder à des traitements post-mortem à la simulation grâce à des outils comme *awk.*

[W1]. http: // www.isi.edu\nsnam\ns \doc\index.html

Annexe II

Exemple de code source

Dans cette annexe, nous présentons le code source de l'algorithme MS avec recouvrement (simulation avec faute).

Le fichier (ms.h):

```
#ifndef ns_ms_h
#define ns_ms_h
#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
struct hdr_ACK {
        int id_proc;
        // la méthode pour accéder à l'entête
        static int offset_;
        inline static int &offset() { return offset_; }
        inline static hdr_ACK* access(const Packet* p) {
                return (hdr_ACK*) p->access(offset_);
        }
};
struct hdr_message {
        char type;
        int Msn;        //numéro de séquence superposé avec le message M
        int Minc;      //numéro d'incarnation (compteur d'erreur) superposé avec le message M
        int M.rec_line;   //numéro de la ligne de recouvrement superposé avec le message M
        int Mid_proc;     //l'identifiant de processus superposé avec le message M
                // La méthode pour accéder à l'entête du paquet
        static int offset_;
        inline static int &offset() { return offset_; }
        inline static hdr_message* access(const Packet* p) {
                return (hdr_message*) p->access(offset_);
        }
};

struct buf{
// utilisé pour sauvegarder le sn de point de reprise c, le sn superposé avec le message M
// et l'identifiant de processus
        int csn;
        int msn;
        int pid;
};
        //      Création de la classe msAgent héritée de la classe Agent
class msAgent : public Agent {
public:
        msAgent();
        int skipped;            //Nombre de points échappés
        int nb_fr_;         //Nombre de points forcés ajoutés lors de recouvrement
        int nb_force_;   //Nombre de points forcés
        int nb_basic_;    //Nombre de points basics
        int deleted;        //Nombre de points supprimer lors de recouvrements
        int nb_msg_;       //Nombre de messages
        int nb_node_;     //Nombre de processus
        int nb_sauv_;      //Nombre de messages sauvegardés
```

```
        int nb_annul_;    //Nombre de messages annulés
        int nb_retard_;    //Nombre de messages retardés
        int nb_ack_;       //nombre d'ACKs
        int recieved;        //Nombre de messages reçus
        struct buf buffer[1000];
        virtual int command(int argc, const char*const* argv);
         virtual void recv(Packet*, Handler*);
protected:
        int id_proc;       //l'identifiant de processus
        int sn;           // numéro de séquence de processus
        int next;         // numéro de séquence du prochain point de reprise;
        int inc;     // incrémenté à chaque fois qu'un processus échoue
        int rec_line;     //le numéro de la ligne de recouvrement.
        int checkpoint[12000];
        int s;
        int t;
};
#endif   // ns_ms_h
```

Le fichier (ms.cc):

```
#include "ms.h"
#include "agent.h"
int hdr_ACK::offset_;
int hdr_message::offset_;
//*****************************************************************************
//                    fonction min pour donner le minimum de deux variables
//*****************************************************************************
int min(int x, int y)
{
 if (x < y) return(x);
 else return(y);
}
//*****************************************************************************
//                              pour calculer offset de l'en tête hdr_ms
//*****************************************************************************
static class msHeaderClass : public PacketHeaderClass {
public:
        msHeaderClass() : PacketHeaderClass("PacketHeader/ms",
                                  sizeof(hdr_message)) {
                bind_offset(&hdr_message::offset_);
        }
} class_mshdr;

static class ACKHeaderClass : public PacketHeaderClass {

public:
        ACKHeaderClass() : PacketHeaderClass("PacketHeader/ms",
                                  sizeof(hdr_ACK)) {
                bind_offset(&hdr_ACK::offset_);
        }
} class_ackhdr;
```

```
//*****************************************************************************
//                    pour créer une intance à partir de tcl
//*****************************************************************************
static class msClass : public TclClass {
public:
        msClass() : TclClass("Agent/ms") {}
        TclObject* create(int, const char*const*) {
                return (new msAgent());
        }
} class_ms;

//*****************************************************************************
//        pour faire la lien entre les variables en C++ et en TCL (bind)
//*****************************************************************************
msAgent::msAgent(): Agent(PT_MS), nb_node_(0), nb_msg_(0), nb_basic_(0), nb_force_(0),
id_proc(0),   sn(0), next(1),
rec_line(0),inc(0),nb_retard_(0),nb_annul_(0),nb_sauv_(0),s(0),deleted(0),nb_ack_(0), recieved(0),
skipped(0),t(0),nb_fr_(0)
                                                // PT_MS identifier un protocole dans les traces
{
        bind("packetSize_", &size_);
        bind("nb_node_", &nb_node_);
        bind("nb_msg_", &nb_msg_);
        bind("nb_basic_", &nb_basic_);
        bind("nb_retard_", &nb_retard_);
        bind("nb_annul_", &nb_annul_);
        bind("nb_sauv_", &nb_sauv_);
        bind("nb_ack_", &nb_ack_);
        bind("nb_force_", &nb_force_);
        bind("id_proc",&id_proc);
        bind("recieved",&recieved);
        bind("inc",&inc);
        bind("rec_line",&rec_line);
        bind("sn",&sn);
        bind("deleted",&deleted);
        bind("next",&next);
        bind("skipped",&skipped);
        bind("indice",&s);
        bind("nb_fr_",&nb_fr_);
}
//*****************************************************************************
//                                    pour envoyer les paquets
//*****************************************************************************
int msAgent::command(int argc, const char*const* argv)
{int j;

  if (argc == 2) {
    if (strcmp(argv[1], "send-message") == 0) {
                //Pour la taille de paquet
            size_ =600;
                //Pour la couleur de paquet
            fid_ = 0;
                 // Création d'un nouveau paquet
            Packet* pkt = allocpkt();
            // L'accès à l'entête ms pour un nouveau paquet
```

```
            hdr_message* hdr = hdr_message::access(pkt);
                    hdr->type = 'M';
                    hdr->Mid_proc = here_.addr_;
                    hdr->Msn = sn;
                    hdr->M.inc = inc;
                    hdr->M.rec_line = rec_line;
                    nb_msg_ = nb_msg_ +1;
          //Envoyer le paquet
            send(pkt, 0);
   return (TCL_OK); }
   else if (strcmp(argv[1], "take-basic") == 0) {
            if(next > sn) {
                                    nb_basic_=nb_basic_+1;
                                    sn = next;
                                    checkpoint[s]= sn;
                                    s=s+1;}else{skipped=skipped+1;}
                                    next=next+1;
    return (TCL_OK);}
   else if (strcmp(argv[1], "take-initial") == 0) {
                    sn=0;
                    nb_basic_=nb_basic_+1;
                    checkpoint[s]= sn; //ce tableau permet de sauvegarder les points de reprise spontanés
                    s=s+1;
    return (TCL_OK);}
   else if (strcmp(argv[1], "send-rollback") == 0) {
                    //Pour la taille de paquet
                     size_ =600;
                    //Pour la couleur de paquet
                    fid_ = 1;
                     // Création d'un nouveau paquet
                    Packet* pkt = allocpkt();
                    // L'accès à l'entête des paquets ms
                     inc=inc+1;
                     hdr_message* hdr = hdr_message::access(pkt);
                     hdr->type = 'R';
                     hdr->Mid_proc = here_.addr_;
                     hdr->Msn = sn;
                     hdr->M.inc = inc;
                     rec_line=sn;
                     hdr->M.rec_line = rec_line;
                  //Envoyer le paquet
                     send(pkt, 0);
   return (TCL_OK); }
  }
          return (Agent::command(argc, argv));
}
//*****************************************************************************
//                    pour traiter les paquets envoyés
//*****************************************************************************
void msAgent::recv(Packet* pkt, Handler*)
{
hdr_message* hdr = hdr_message::access(pkt);
int i,ind,j,a,b;
char find;
i=0; find='f'; ind=0;
```

```
if(hdr->type=='M'){                              // message normal
        recieved =recieved+1;
        if (hdr->Minc< inc){// message reçu après recouvrement mais envoyé avant le recouvrement
                if(hdr->Msn<=rec_line){          // message retarde
                        nb_sauv_=nb_sauv_+1;   // enregistrer message
                        nb_retard_=nb_retard_+1;
                        buffer[t].csn=sn;
                        buffer[t].msn=hdr->Msn;
                        buffer[t].pid=hdr->Mid_proc ;
                        t=t+1;
                        }else{                   // message duplique
                        nb_annul_=nb_annul_+1;
                        }
        }
        if (hdr->Minc> inc){      //message reçu qui indique qu'il y a un processus qui tombe en panne
                rec_line = hdr->M.rec_line;
                inc = hdr->Minc;
                                        //rollback
                if (rec_line> sn){          //prendre un point de reprise
                        nb_force_=nb_force_+1;
                        nb_fr_=nb_fr_+1;
                        sn = hdr->Msn;
                        checkpoint[s]= sn;
                        s=s+1;
                }else{                   // trouver un point de reprise avec sn >= rec_line
                        while ((i<s)&&(find=='f'))
                        {
                        if(checkpoint[i]>= rec_line)
                                {
                                find='t';
                                sn=checkpoint[i];
                                }
                        i=i+1;
                        }
                deleted=deleted+s-i;
                next=sn+1;
                s=i;
                }                               //fin rollback
                for (j=0; j<=t; j=j+1){
                        a=buffer[j].csn;
                        b=buffer[j].msn;
                        if ((a>= sn)&&(b< rec_line)){
                        fid_ = 2;
                        Packet* pktack = allocpkt();
                        size_ =500;
                        hdr_ACK* hdrack = hdr_ACK::access(pktack);
                        dst_.addr_= buffer[j].pid;
                        nb_ack_=nb_ack_+1;
                        send(pktack,0);
                        }j=j+1;
                        }
        }
       if (hdr->Minc==inc){//message indique que les deux processus en même état "normal"
                if(hdr->Msn> sn) {
                        nb_force_=nb_force_+1;
```

```
                    sn = hdr->Msn;
                    checkpoint[s]= sn;
                    s=s+1;
                    }
                if(hdr->Msn< sn) {
                nb_sauv_=nb_sauv_+1;
                buffer[t].csn=sn;
                buffer[t].msn=hdr->Msn;
                buffer[t].pid=hdr->Mid_proc ;
                t=t+1;
                }
        }
}else if(hdr->type=='R'){               // message reçu est un message de recouvrement
        if (hdr->M.inc> inc){           //rollback
                rec_line = hdr->M.rec_line;
                inc = hdr->Minc;
                if (rec_line> sn){      //prendre un point de reprise forcé
                        nb_fr_=nb_fr_+1;
                        nb_force_=nb_force_+1;
                        sn = rec_line;
                        checkpoint[s]= sn;
                        s=s+1;
                }else{                  // trouver un point de reprise avec sn >= rec_line
                        while ((i<s)&&(find=='f'))
                        {
                        if(checkpoint[i]>= rec_line)
                                {
                                find='t';
                                sn=checkpoint[i];
                                }
                        i=i+1;
                        }
                deleted=deleted+s-i;
                next=sn+1;
                s=i;
                }                       //fin rollback
                for (j=0; j<=t; j=j+1){
                        a=buffer[j].csn;
                        b=buffer[j].msn;
                        if ((a>= sn)&&(b< rec_line)){
                        fid_ = 2;
                        Packet* pktack = allocpkt();
                        size_ =500;
                        hdr_ACK* hdrack = hdr_ACK::access(pktack);
                        dst_.addr_= buffer[j].pid;
                        nb_ack_=nb_ack_+1;
                        send(pktack,0);
                        }j=j+1;
                        }
 }                                      //sinon ignorer le message de recouvrement
}
}
```

Le fichier (scenario_avec_faut.tcl) :

```
#############################################################################
#          L'ouverture de fichier rollback.txt qui contient les informations de simulation          #
#############################################################################
set r [open "rollback.txt" w]

#***************Demande de l'utilisateur pour entrer le nbre de processus**************#
puts "Entrez le nombre de processus de votre simulation : "
gets stdin nb_node_
puts $r "$nb_node_"
#***************Demande de l'utilisateur pour entrer la durée de simulation*************#
puts "Entrez le temps de simulation : "
gets stdin tps
puts $r "$tps"
#*************Demande de l'utilisateur pour entrer le nombre de messages**************#
puts "Entrez le nombre de message : "
gets stdin nb_msg_
puts $r "$nb_msg_"
#**************Demande de l'utilisateur pour entrer le nombre du fautes***************#
puts "Identifiez le nombre du fautes : "
gets stdin nb_faut
puts $r "$nb_faut"
#***Demande de l'utilisateur pour entrer le nombre d'événements pour faire point basic***#
puts "Après combien d'évènements faire point basic ? "
gets stdin nb_eve_
puts $r "$nb_eve_"
#*********Donner chaque processus un identifiant et numéro de séquence initial**********#
for {set j 0} {$j < $nb_node_} {incr j} {
        for {set k 0} {$k < 2} {incr k} {
                                switch $k {
                                        # identifiant de processus
                                        0 { set table($j,$k) $j }
                                        # initialisé sn de chaque site  à '0'
                                        1 { set table($j,$k) 0 }
                                }
    puts $r "$table($j,$k)"
        }
}
close $r
#############################################################################
#                              Création de scénario d'envoie de message                              #
#############################################################################
#############################################################################
#                          L'accès au fichier rollback.txt                                          #
#############################################################################
set r [open "rollback.txt" "r"]
set nb_node_ [gets $r]
set tps [gets $r]
set nb_msg_ [gets $r]
set nb_faut [gets $r]
set nb_eve_ [gets $r]
close $r
```

```
#############################################################################
#                    Prendre les temps pour faire les envoies des messages                    #
#############################################################################
set tcom  $tps
for {set i 0} {$i < $nb_msg_} {incr i} {
set exp [expr rand()*$tcom]
        for {set j 0} {$j < 2} {incr j} {
                                switch $j {
                                        0 { set envoi_table_($i,$j) $i }
                                        1 { set envoi_table_($i,$j) $exp }
                                }
      }
}

#############################################################################
#                    Tri de tableau des envois des messages par décalage                    #
#############################################################################
for {set i 0} {$i <[expr $nb_msg_-1]} {incr i} {
        for {set j [expr $i+1]} {$j <$nb_msg_} {incr j} {
                if { $envoi_table_($i,1) > $envoi_table_($j,1)} {
                        set interr $envoi_table_($i,1)
                        set envoi_table_($i,1) $envoi_table_($j,1)
                        set envoi_table_($j,1) $interr
           }
      }
}
#############################################################################
#                    L'ouverture de fichier scenario.txt qui contient                    #
#                    les informations d'envoies et de réceptions des messages                    #
#############################################################################
set s [open "scenario.txt" w]
for {set j 0} {$j <$nb_msg_} {incr j} {
        set src_ 0
        set dest_ 0
        while { $src_ == $dest_} {
                set src_ [expr int(rand()*$nb_node_)]
                set dest_ [expr int(rand()*$nb_node_)]
        }
        for {set k 0} {$k < 3} {incr k} {
                                switch $k {
                                        0 { set scenario($j,$k) $envoi_table_($j,1) }
                                        1 { set scenario($j,$k) $src_ }
                                        2 { set scenario($j,$k) $dest_ }
                                }
        puts $s "$scenario($j,$k)"
        }
}
close $s
#############################################################################
#                    L'ouverture de fichier panne.txt qui contient                    #
#                    les processus qui tombent en panne et le moment de panne                    #
#############################################################################
set pa [open "panne.txt" w]
set id_panne 0
set taille 0
```

```
set inter [expr $tps/[expr $nb_faut+1]]
set time $inter
set j 0
while { $time < $tps} {
        set id_panne [expr int(rand()*$nb_node_)]
        for {set k 0} {$k < 2} {incr k} {
                                        switch $k {
                                                0 { set panne($j,$k) $time }
                                                1 { set panne($j,$k) $id_panne }
                                        }
                puts $pa "$panne($j,$k)"
                }
        set time [expr $time+$inter]
        set taille [expr $taille+1]
        incr j
        }
close $pa
#############################################################################
#                                  L'ouverture de fichier basic.txt                                  #
#############################################################################
set ba [open "basic.txt" w]
set time 0.02
set long1 0
set long2 0
for {set j 0} {$j < $nb_node_} {incr j} {
        for {set k 0} {$k < 2} {incr k} {
                                switch $k {
                                        # identifiant de processus
                                        0 { set even_table($j,$k) $j }
                                        # even de caque processus initialisé à '0'
                                        1 { set even_table($j,$k) 0 }
                        }
        }
}
for {set i 0} {$i < $nb_msg_} {incr i} {
        set src $scenario($i,1)
        incr even_table($src,1)
        set des $scenario($i,2)
        incr even_table($des,1)
        set heurs $scenario($i,0)
        if {$even_table($src,1)==$nb_eve_} {
                set even_table($src,1) 0
                for {set k 0} {$k < 2} {incr k} {
                                        switch $k {
                                                0 { set basic($i,$k) $heurs }
                                                1 { set basic($i,$k) $src }
                                }
                puts $ba "$basic($i,$k)"
                }
        set long1 [expr $long1 +1]
        }
        if {$even_table($des,1)==$nb_eve_} {
                set even_table($des,1) 0
                for {set k 0} {$k < 2} {incr k} {
                                        switch $k {
```

```
                                                0 { set basic($i,$k) $heurs }
                                                1 { set basic($i,$k) $des }
                                        }
                puts $ba "$basic($i,$k)"
                }
        set long2 [expr $long2 +1]
        }
}
close $ba
#############################################################################
#                                        L'accès au fichier basic.txt                                #
#############################################################################
set ba [open "basic.txt" "r"]
set long [expr $long1 +$long2]
for {set j 0} {$j < $long} {incr j} {
        for {set k 0} {$k < 2} {incr k} {
        set basic($j,$k) [gets $ba]
        }
}
close $ba
#############################################################################
#                                        L'accès au fichier panne.txt                                #
#############################################################################
set pa [open "panne.txt" "r"]
for {set j 0} {$j < $taille} {incr j} {
        for {set k 0} {$k < 2} {incr k} {
        set panne($j,$k) [gets $pa]
        }
}
close $pa
#############################################################################
#                                L'ouverture de fichier global.txt qui regroupe                #
#                                   tous les moments dans un tableau glob_tp                #
#############################################################################
set global [open "global.txt" w]
for {set i 0} {$i < $nb_msg_} {incr i} {
                        set envoi $scenario($i,0)
for {set j 0} {$j < 2} {incr j} {
                        switch $j {
                        0 { set glob_tp($i,$j) $envoi}
                        1 { set glob_tp($i,$j) 1 }
                        }
puts $global "$glob_tp($i,$j)"
}
}
for {set i 0} {$i < $taille} {incr i} {
                                set pan $panne($i,0)
for {set j 0} {$j < 2} {incr j} {
                        switch $j {
                                0 { set glob_tp($i,$j) $pan}
                                1 { set glob_tp($i,$j) 3 }
                }
puts $global "$glob_tp($i,$j)"
}
}
```

```
for {set i 0} {$i < $long} {incr i} {
                        set bas $basic($i,0)
for {set j 0} {$j < 2} {incr j} {
                        switch $j {
                                0 { set glob_tp($i,$j) $bas}
                                1 { set glob_tp($i,$j) 2 }
                }
puts $global "$glob_tp($i,$j)"
}
}
close $global
#############################################################################
#                                        L'ouverture de fichier final.txt                                #
#############################################################################
set fi [open "final.txt" w]
#**************************L'accès au fichier global.txt************************#
set global [open "global.txt" "r"]
set tot [expr $nb_msg_ + $long]
set total [expr $taille +$tot]
puts $fi "$total"
puts $fi "$long"
puts $fi "$taille"
for {set j 0} {$j <= $total} {incr j} {
        for {set k 0} {$k < 2} {incr k} {
        set glob_tp($j,$k) [gets $global]
        }
}
#**************************Tri de tableau glob_tp*****************************#
for {set i 0} {$i <=$total} {incr i} {
        for {set j [expr $i+1]} {$j <=$total} {incr j} {
                set x $glob_tp($i,0)
                set y $glob_tp($j,0)
        if { $x > $y } {
                        set var $glob_tp($i,0)
                set glob_tp($i,0) $glob_tp($j,0)
                set glob_tp($j,0) $var

                        set val $glob_tp($i,1)
                set glob_tp($i,1) $glob_tp($j,1)
                set glob_tp($j,1) $val
                  }
        }
puts $fi "$glob_tp($i,0)"
puts $fi "$glob_tp($i,1)"
}
close $global
close $fi
```

le fichier (ms_avec_faut.tcl):

```
#################################################################################
#                                  Création d'un simulateur                     #
#################################################################################
set ns [new Simulator]
set bw rol.dat
set f0 [open $bw w]
set bw res.dat
set ck [open $bw w]
set bw pa.dat
set f1 [open $bw w]
#################################################################################
#                                  L'ouverture de fichier trace                 #
#################################################################################
set nf [open l.nam w]
$ns namtrace-all $nf
#################################################################################
#                                  Definition de procedure ‘finishes'           #
#################################################################################
proc finish {} {
      global ns nf f0 nb_node_ p  nmsg  del c basi forced ck sauv annul retard ack deleted  sk rec
forcplus
         puts $ck "n   sn   next   basic   force   deleted   nb_sauv_   nb_annul_   nb_retard_
nb_ack_   skipped   received   nb_fr_"
         for {set i 0} {$i < $nb_node_} {incr i} {
         set n [$p($i) set agent_addr_]
         set l [$p($i) set sn]
         set s [$p($i) set next]
         set a [$p($i) set nb_basic_]
         set b [$p($i) set nb_force_]
         set c [$p($i) set deleted]
         set d [$p($i) set nb_sauv_]
         set e [$p($i) set nb_annul_]
         set f [$p($i) set nb_retard_]
         set j [$p($i) set nb_ack_]
         set k [$p($i) set recieved]
         set md [$p($i) set skipped]
         set ind [$p($i) set indice]
         set nbf [$p($i) set nb_fr_]
         puts $ck "$n   $l   $s   $a   $b   $c   $d   $e   $f   $j   $md
$k   $nbf"
         set del [expr [$p($i) set deleted]+$del]
         set basi [expr [$p($i) set nb_basic_]+$basi]
         set forced [expr [$p($i) set nb_force_]+$forced]
         set nmsg [expr [$p($i) set nb_msg_]+$nmsg]
         set sauv [expr [$p($i) set nb_sauv_]+$sauv]
         set annul [expr [$p($i) set nb_annul_]+$annul]
         set retard [expr [$p($i) set nb_retard_]+$retard]
         set ack [expr [$p($i) set nb_ack_]+$ack]
         set rec [expr [$p($i) set recieved]+$rec]
         set sk [expr [$p($i) set skipped]+$sk]
         set forcplus [expr [$p($i) set nb_fr_]+$forcplus]
         }
         puts $f0 "################################"
```

```
        puts $f0 "$del est le nombre de points supprimés"
        puts $f0 "$basi est le nombre de points spontanés"
        puts $f0 "$forced est le nombre de points forcés"
        puts $f0 "$nmsg est le nombre total de messages envoyés"
        puts $f0 "$sauv est le nombre de messages sauvegardés "
        puts $f0 "$annul est le nombre de messages annulés "
        puts $f0 "$retard est le nombre de messages retardés "
        puts $f0 "$ack est le nombre d'ack "
        puts $f0 "$rec est le nombre d'msg recu "
        puts $f0 "$sk est le nombre de point echappé "
        puts $f0 "$forcplus est le nombre de points forcés à cause de recouvrement "
    $ns flush-trace
        #Close the trace file
    close $nf
        #Execute nam on the trace file
    exec nam ck.nam &
    exit 0
}
#############################################################################
#                    Procédure permet d'afficher le processus qui prend un point forcé        #
#############################################################################
proc forced {p q n} {
set ns [Simulator instance]
set nowe [$ns now]
set time 0.1
set d [$p set sn]
set b [$q set sn]
if { $d > $b} {
$ns at $nowe "$n color red"
            $ns at $nowe "$n label \" point force ($d) \""
            $ns at [expr $nowe+0.01] "$n color black"
            $ns at [expr $nowe+0.01] "$n label \" \""
}
}
#############################################################################
#            Procédure permet d'afficher le processus qui prend un point spontané              #
#############################################################################
proc basicinitial {p n} {
set ns [Simulator instance]
set nowe [$ns now]
$ns at $nowe "$p take-initial"
set b [$p set sn]
$ns at $nowe "$n color blue"
$ns at $nowe "$n label \" point basic ($b) \""
$ns at [expr $nowe+0.01] "$n color black"
$ns at [expr $nowe+0.01] "$n label \" \""
}
proc basic {p n} {
set ns [Simulator instance]
set nowe [$ns now]
$ns at $nowe "$p take-basic"
set b [$p set sn]
set b [expr $b+1]
$ns at $nowe "$n color blue"
$ns at $nowe "$n label \" point basic ($b) \""
```

```
$ns at [expr $nowe+0.01] "$n color black"
$ns at [expr $nowe+0.01] "$n label \" \""
}
#############################################################################
#                    Procédure permet d'afficher le processus qui tombe en panne                    #
#############################################################################
proc panne {p n } {
global inter tps nb_faut
set ns [Simulator instance]
set nowe [$ns now]
set time 0.01
set b [$p set agent_addr_]
                    $ns at $nowe "$n color brown"
            $ns at $nowe "$n label \" p($b) en panne  \""
                    $ns at [expr $nowe+$time] "$n color black"
            $ns at [expr $nowe+$time] "$n label \" \""
}
#############################################################################
#                              Procédure pour diffuser le message de Rollback                              #
#############################################################################
proc diffusion {p n} {
global nb_node_
set ns [Simulator instance]
set nowe [$ns now]
set b [$p set agent_addr_]

for {set i 0} {$i < $nb_node_} {incr i} {
          if { $b != $i} {
          $ns at $nowe "$p set dst_addr_ $i"
          $ns at $nowe "$p send-Rollback"
          }
}
}
#############################################################################
#                                                L'accès au fichier rollback.txt                                                #
#############################################################################
set r [open "rollback.txt" "r"]
set nb_node_ [gets $r]
set tps [gets $r]
set nb_msg_ [gets $r]
set nb_faut [gets $r]
set nb_eve_ [gets $r]
for {set j 0} {$j < $nb_node_} {incr j} {
          for {set k 0} {$k < 2} {incr k} {
          set table($j,$k) [gets $r]
          }
}
close $r
#############################################################################
#                                              L'accès au fichier scenario.txt                                              #
#############################################################################
set s [open "scenario.txt" "r"]
for {set j 0} {$j < $nb_msg_} {incr j} {
          for {set k 0} {$k < 3} {incr k} {
          set scenario($j,$k) [gets $s]
```

```
        }
}
close $s
#############################################################################
#                                  L'accès au fichier final.txt                                  #
#############################################################################
set fi [open "final.txt" "r"]
set total [gets $fi]
set long [gets $fi]
set taille [gets $fi]
for {set j 0} {$j <= $total} {incr j} {
          for {set k 0} {$k < 2} {incr k} {
          set glob_tp($j,$k) [gets $fi]
}
   } close $fi
#############################################################################
#                                  L'accès au fichier basic.txt                                  #
#############################################################################
set ba [open "basic.txt" "r"]
for {set j 0} {$j < $long} {incr j} {
          for {set k 0} {$k < 2} {incr k} {
          set basic($j,$k) [gets $ba]
          }
}
close $ba
#############################################################################
#                                  L'accès au fichier panne.txt                                  #
#############################################################################
set pa [open "panne.txt" "r"]
for {set j 0} {$j < $taille} {incr j} {
          for {set k 0} {$k < 2} {incr k} {
          set panne($j,$k) [gets $pa]
          }
}
close $pa
#############################################################################
#************************Identification du couleur de paquet*********************#
$ns color 0 Green
$ns color 1 Blue
$ns color 2 Red
#************************Création des nœuds*********************************#
for {set i 0} {$i <$nb_node_} {incr i} {
                          set n($i) [$ns node]
                           }
#*******Création d'une ligne de communication "full duplex" entre les nœuds*********#
for {set i 0} {$i <$nb_node_} {incr i} {
          for {set j [expr $i+1]} {$j <$nb_node_} {incr j} {
                          $ns duplex-link $n($i) $n($j) 1Mb 10ms DropTail
          }
}

#**Création d'un agent MS devenue comme un générateur de paquet pour chaque nœud **#
for {set i 0} {$i <$nb_node_} {incr i} {
                                        set p($i) [new Agent/ms]
}
```

```
for {set i 0} {$i <$nb_node_} {incr i} {
                                        $ns attach-agent $n($i) $p($i)
}
#**********************Connection entre deux agents**************************#
for {set i 0} {$i <$nb_node_} {incr i} {
        for {set j [expr $i+1]} {$j <$nb_node_} {incr j} {
                                                $ns connect $p($i) $p($j)
        }
}

#**********************Initialisation des variables utilisés**********************#
set nmsg 0
set del 0
set basi 0
set forced 0
set sauv  0
set annul 0
set retard 0
set ack 0
set now 0
set time 0.02
set j 0
set k 0
set ind 0
set sk 0
set rec 0
set forcplus 0
#############################################################################
#############################################################################
#          Chaque processus peut prendre son point de reprise spontané initial                #
#############################################################################
for {set i 0} {$i < $nb_node_} {incr i} {
        $ns at 0.0 "basicinitial $p($i) $n($i)"
}
#############################################################################
#                  Parcours de tableau "glob_tp" trié qui contient les moments                #
#                    et l'accès aux fichiers utilisés selon leurs types "1,2 ou 3"            #
#############################################################################
for {set i 0} {$i <= $total} {incr i} {
        set heur $glob_tp($i,0)
        set var $glob_tp($i,1)
        if {$var == 1} {
                set src $scenario($j,1)
                set des $scenario($j,2)
                $ns at $heur "$p($src) set dst_addr_ $des"
                $ns at $heur "$p($src) send-message"
                $ns at $heur "forced $p($src) $p($des) $n($des)"
                incr j
        }
        if {$var == 2} {
                set pre $basic($k,1)
                $ns at [expr $heur+$time] "basic $p($pre) $n($pre)"
                incr k
        }
        if {$var == 3} {
```

```
                set id_panne $panne($ind,1)
                $ns at $heur "aff-sn"
                $ns at $heur "panne $p($id_panne) $n($id_panne) "
                $ns at $heur "diffusion $p($id_panne) $n($id_panne) "
                $ns at [expr $heur+0.07] "aff-sn"
        incr ind
        }
}
#############################################################################
#               Appel de procédure "finish" après un temps "tps" de simulation          #
#############################################################################
$ns at $tps "finish"
#############################################################################
#                                   lancer la simulation                                  #
#############################################################################
$ns run
```

Annexe III

Mode d'utilisation

1. Introduction :

Pour mieux comprendre et pour faciliter à l'utilisateur de bien simuler les algorithmes réalisés, nous présentons cette annexe comme un mode d'utilisation avec les étapes nécessaires pour lancer la simulation.

Afin de bien simuler Nous avons divisé le script selon différents cas importants (simulation avec faute, simulation sans faute, et simulation avec des processus asymétriques (rapides ou lents)).

Pour tester notre agent et avant de lancer la simulation on doit d'abord exécuter le programme « *scenario_avec_faut.tcl* » afin de générer et d'enregistrer les cinq fichiers « *rollback.txt* », « *scenario.txt* », « *basic.txt* », « *panne.txt* » et « *final.txt* » pour la simulation (voir Les étapes de la simulation dans le chapitre III).

La figure présente une capture d'écran de la fenêtre qui se lance après le tapement de l'instruction « *ns scenario_avec_faut.tcl* » dans un terminal selon le choix de simulation (dans cette annexe nous avons choisi la simulation avec faute).

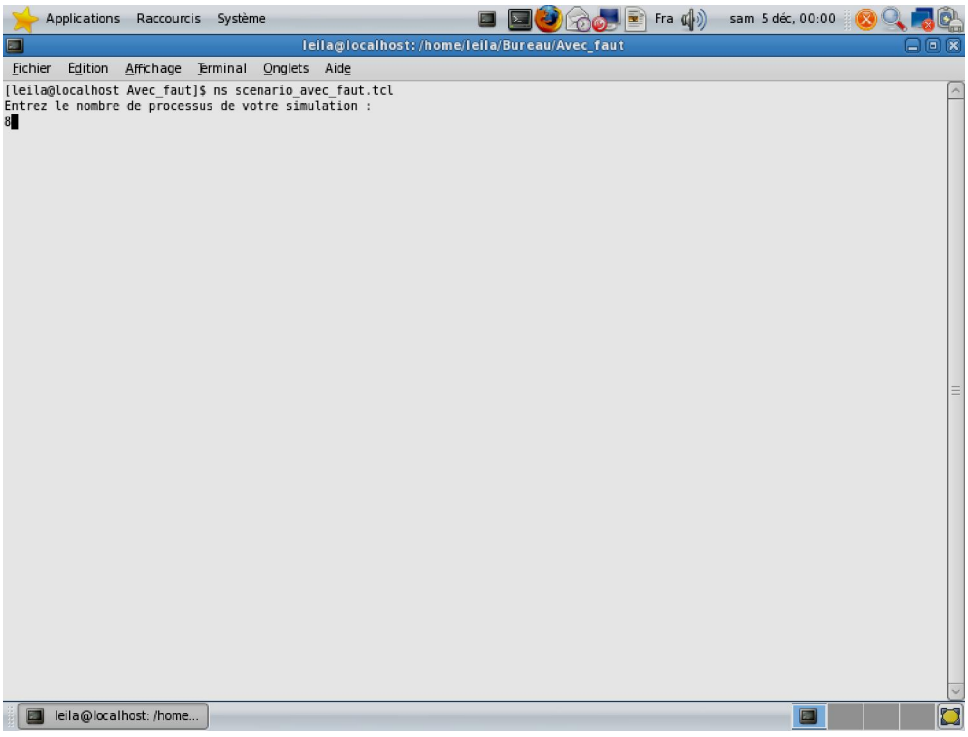


Figure A3. 1 Saisie des paramètres de simulation (avec faute)

Cette fenêtre permet à l'utilisateur de saisir les paramètres de simulation.

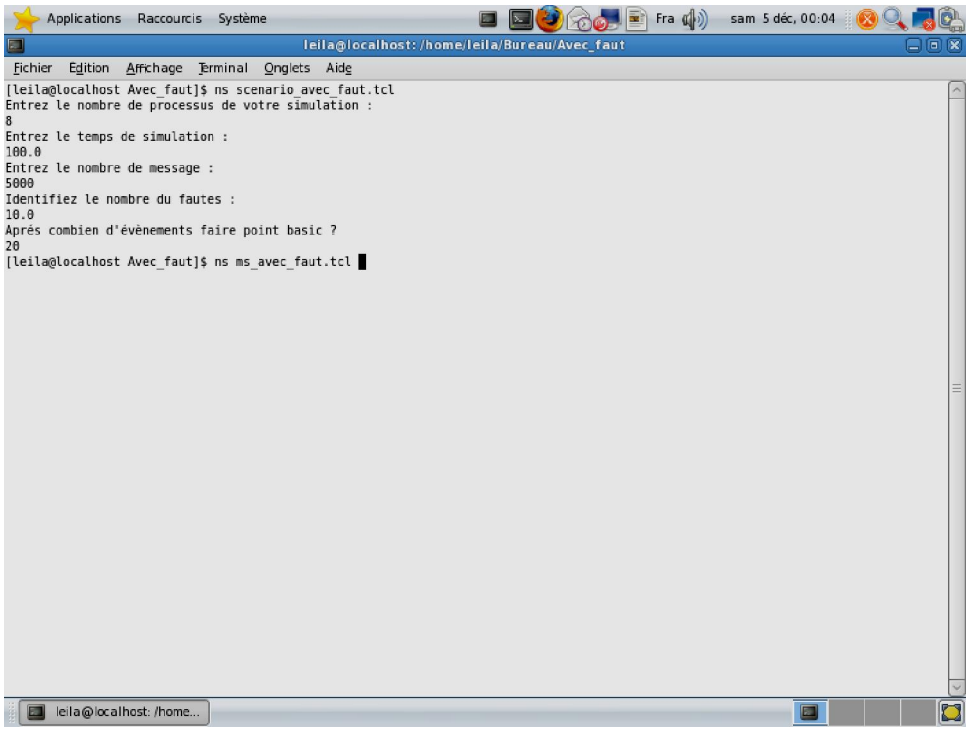


Figure A3. 2 Fin de saisie des paramètres de simulation

Après la saisie des paramètres de simulation, on tape « *ns ms_avec_faut.tcl* ».

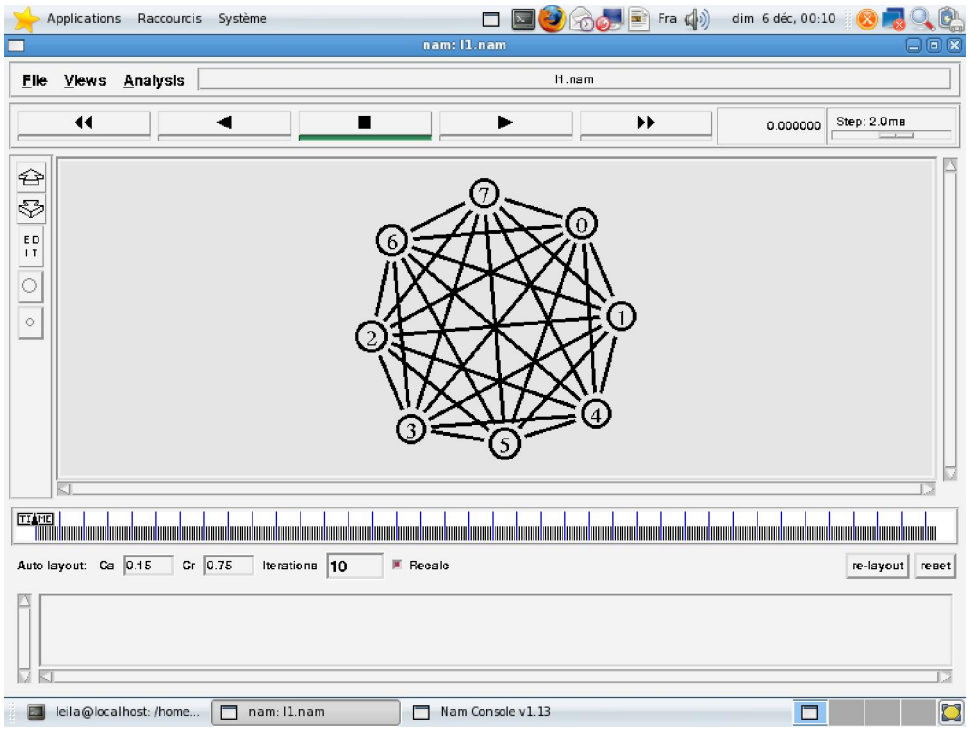


Figure A3. 3 Visualisation de l'algorithme MS (avec faut)

La simulation doit être lancée en cliquant sur le bouton de début de simulation.

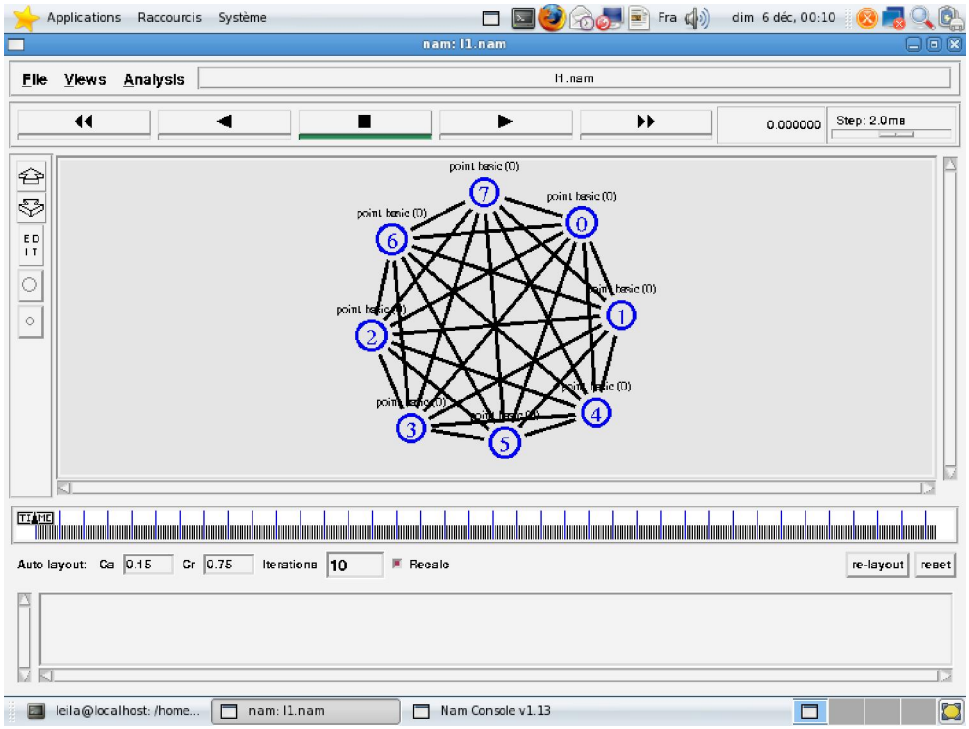


Figure A3. 4 Prendre un point de reprise spontané initial

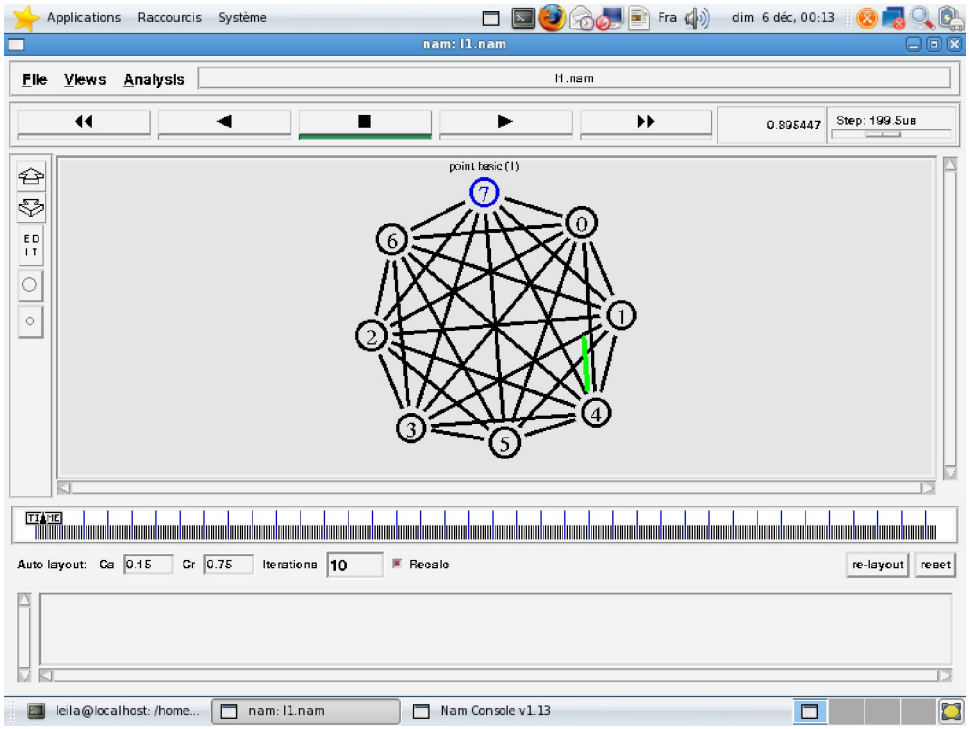


Figure A3. 5 P7 Prend un point de reprise spontané

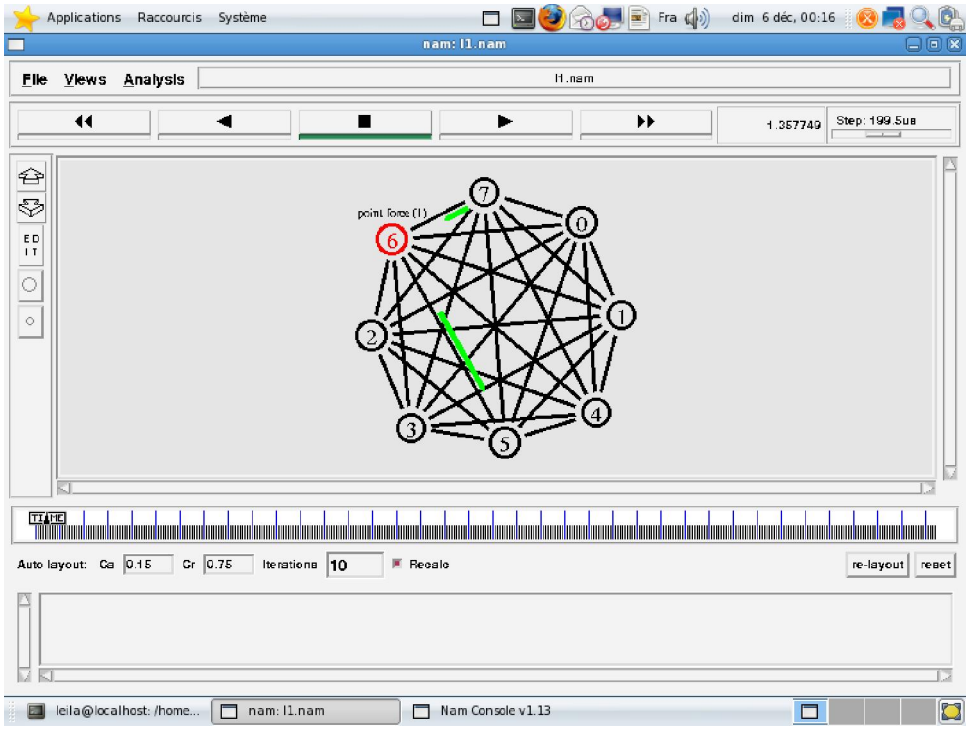


Figure A3. 6 P6 prend un point de reprise forcé

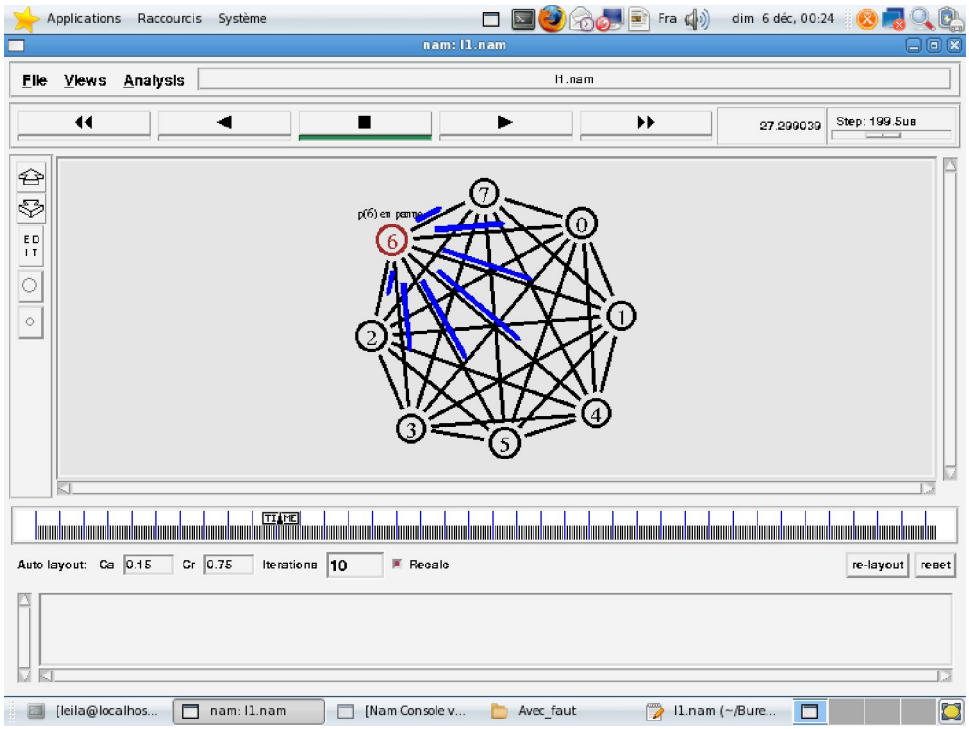


Figure A3. 7 P6 est en panne et diffuse un message de recouvrement

Remarque : pour l'algorithme BCS on va suivre les mêmes étapes.

La couleur de nœuds et de paquets peut être changée pendant la simulation :

Dans la zone d'animation (visualisation), la couleur des nœuds est noire dans l'état normal, cette couleur peut être changée en bleu si un processus prend un point basic (nœud bleu), dans le cas où le processus prend un point forcé, la couleur de nœud est changée au rouge. Si le processus tombe en panne cette couleur est devenue marron, chaque couleur d'un nœud devient noir à la fin de durée de traitement.

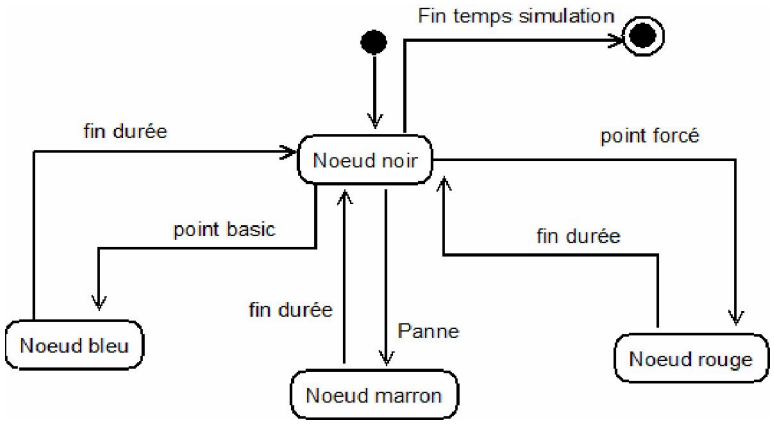


Figure A3. 8 Diagramme d'état / transition «la couleur d'un nœud»

La couleur de packet dans la zone d'animation (visualisation) dépend de la nature de message envoyé, c.-à-d. si le processus envoie un message normal (simple) la couleur de packet devient verte, s'il envoie un ACK cette couleur sera changée au rouge, dés qu'un message est sauvegardé cette couleur revient verte. Dans le cas où le processus envoie un message de recouvrement, la couleur de packet est bleue et à la fin de la durée de recouvrement, cette couleur revient au vert.

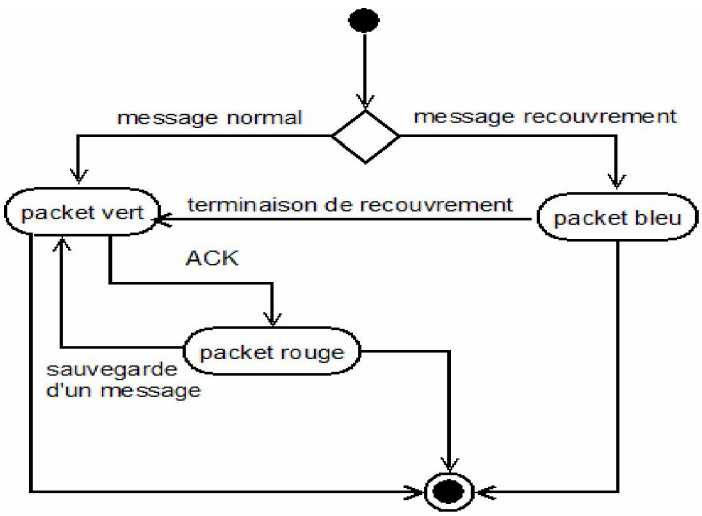


Figure A3. 9 Diagramme d'état / transition «la couleur d'un Paquet»