Esirem Informatique & Réseaux options : Sécurité et Qualités Réseaux

Réseaux

Travaux Pratiques : Compte Rendu

Auteur :
SOULAIROL Lilian

2023-2024

Table des matières

Table des figures

1 Objectif

Le but de ces différents travaux pratiques est de pouvoir mettre en application les différentes notions vues en cours. On s'interresse plus particulèrement sur le principe de routage. Pour réaliser cela on va utiliser un logiciel de simulation sous Linux. Nous allons utilisé plus particulièrement le logiciel NS sous sa version 2. Ns est un logiciel libre de simulation. Le projet ns a débtuté en 1989 comme une variante du simulateur REAL. En 1995 ns a été soutenu par l'agence pour les projets de recherche avancée de défense des Etats-Unis pour créer sa première version en 2000.

La problématique étant de réaliser des réseaux plus ou moins complexes, nous allons donc utiliser ce logiciel en utilisant des scripts en OTCL. Il est donc nésseraire de connaitre les bases de ce langage pour pouvoir réaliser nos premiers scripts. Pour visualiser les réseaux nous allons également utiliser nam qui permet de voir de manière graphique les différents scripts réalisés sous ns-2.

Pour cela, nous allons réaliser 3 TP différents pour comprendre les principes fondamentaux de ce langage et de la mise en place de réseaux informatiques. Le premier TP nous permettra de réaliser des scripts simples pour commencer à comprendre les fonctionnalités basiques de ns-2 et de nam. Dans le deuxième TP nous verrons la mise en place de routage dynamique tel que le routage à vecteur de distance ou à état de lien. Et pour finir, nous verrons dans le troisième TP la visualisation des pérformances de un réseau.

2 TP1

2.1 Introduction

Ce TP a pour but de se familiariser avec le simulateur de réseaux ns-2 et de découvrir les fonctionnalités de base de ce dernier. Pour ce faire, on va créer des scripts en utilisant également nam pour visualiser ces réseaux. On va donc créer deux scripts de base pour pouvoir comprendre le fonctionnement de ns-2 et de nam. Ces scripts seront écrit en OTCL. Le premier exercice se concentrera sur un réseau simple avec seulement deux nœuds tandis que le second exercice aura plus de nœuds pour simuler plusieurs communications en même temps.

2.2 Mise en pratique

2.2.1 Exercice 1

Ce script sera composé de deux noeuds avec un lien duplex de capacité 1Mb avec un temps de propagation de 10ms et une file d'attente de type DropTail avec un agent UDP.

```
#Création d'une instance de l'objet Simulator
set ns [new Simulator]

#Ouvrir le fichier trace pour nam
set nf [open out.nam w]
$ns namtrace-all $nf

#Definir la procédure de terminaison de la simulation
proc finish {} {
        global ns nf
        $ns flush-trace
        #fermer le fichier trace
        close $nf
        #Exécuter le nam avec en entrée le fichier trace
        exec nam out.nam &
        exit 0
}

set n(0) [$ns node]
set n(1) [$ns node]

$ns duplex-link $n(0) $n(1) 1Mb 10ms DropTail

set UDP [new Agent/UDP]
$ns attach-agent $n(0) $UDP

set CBR [new Application/Traffic/CBR]
$CBR set packetSize_ 500
$CBR set interval_ 5ms

$CBR attach-agent $UDP

set Null [new Agent/Null]
$ns attach-agent $n(1) $Null

$ns connect $UDP $Null

$ns at 1 "$CBR start"
$ns at 4.5 "$CBR stop"



#Appeler la procédure de terminaison après un temps t (ex t=5s)
$ns at 5.0 "finish"

#Exécuter la simulation
$ns run
```

FIGURE 1 – Script Exercice 1

La ligne set n(0) [$ns node] permet de créer le nœud numéro 0. Pour pouvoir ensuite créer le lien qu'il y aura entre les deux nœuds, il suffit d'écrire la commande suivante :

$ns duplex-link $n(0) $n(1) 1Mb 10ms DropTail

Les deux premiers paramètres sont les nœuds que l'on veut lier, ensuite on choisi le débit du lien, puis le temps de propagation et pour finir le type de file d'attente. On veut ensuite créer un agent UDP et l'attacher à n(0). Pour cela, il faut utiliser la commande suivante :

set nom_agent [new Agent/UDP]

Cela permet de créer un agent UDP du nom de nom_agent.
On veut ensuite créer une source de trafic CBR avec des paramètres particuliers. Ces paramètres sont les suivants :
Taille de paquet (packetSize_) : 500 octets
Intervalle de transmission de paquets (interval_) : 0.005s

Pour faire cette source de trafic on peut ecrire les lignes suivantes :

set CBR [new Application/Traffic/CBR]
$CBR set packetSize_ 500
$CBR set interval_ 5ms

Une fois l'agent créé et la source également il faut maintenant connecter la source à l'agent avec l'expression suivante :

$CBR attach-agent $UDP

Après cela, il faut créer un deuxième agent pour le lier à l'agent UDP pour permettre la communication entre les deux nœuds. Pour ce faire, il faut écrire les lignes suivantes :

set Null [new Agent/Null]
$ns attach-agent $n(1) $Null
$ns connect $UDP $Null

Pour finir, il suffit de déclencher le trafic à un temps donné et de le stopper également à un temps donné. Pour cet exercice, le trafic CBR doit être déclenché à 1s et doit être arrêté à 4.5s. La simulation doit également être arrêtée à 5s. Il faut donc mettre dans le script les lignes suivantes :

$ns at 1 "$CBR start"
$ns at 4.5 "$CBR stop"
$ns at 5 "finish"

La dernière ligne permet d'appeler la procédure finish à t=5s. En lançant ensuite nam avec la commande "ns Exercice1.tcl" on obtient le schéma suivant :

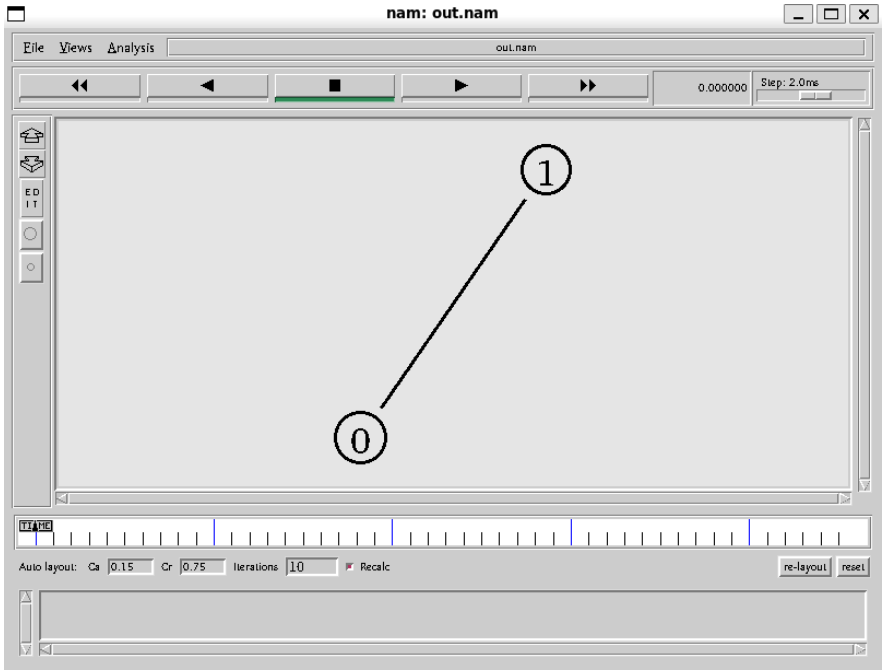

FIGURE 2 – Réseau Exercice 1

On peut voir que les deux nœuds que l'on souhaitait créer ont bien été créés ainsi que le lien entre les deux. Pour vérifier si le lien entre les deux nœuds est bien configuré avec les paramètres que l'on souhaitait, il suffit de cliquer sur celui-ci pour afficher ses paramètres.

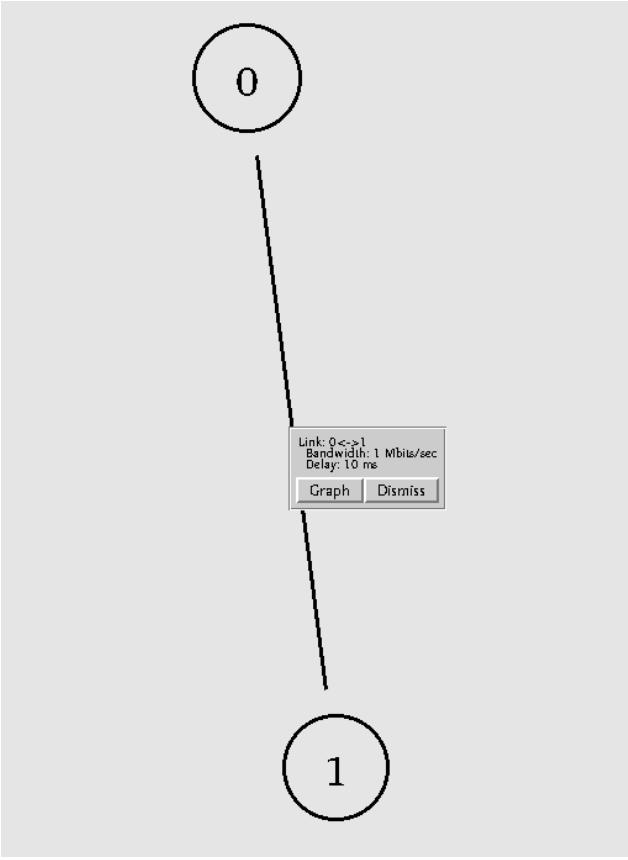

FIGURE 3 – Lien Exercice 1

On peut voir que les paramètres du lien sont bien configurés. Pour lancer la simulation, il suffit de cliquer sur le bouton en forme de flèche. On peut accélérer la simulation à l'aide du bouton en haut à droite en le mettant plus à droite ou ralentir la simulation en le mettant plus à gauche. L'autre flèche permet d'inverser le temps, c'est-à-dire d'aller en marche arrière et les deux flèches permettent de faire un bond dans le temps que ce soit dans le futur ou dans le passé de la simulation. On peut également cliquer sur le bouton EDIT pour modifier l'apparence du réseau en bougeant les différents nœuds.

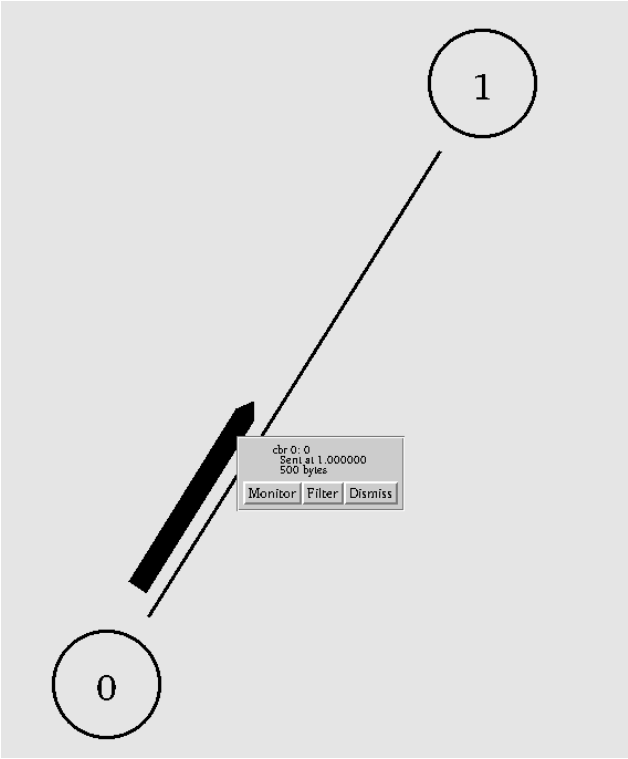

FIGURE 4 – Premier Paquet Exercice 1

On peut voir ici que le premier paquet envoyé est bien envoyé à 1s et que sa taille est de 500 octets ce qui correspond bien à ce que l'on voulait faire. On peut également vérifier si le deuxième paquet envoyé est bien decaler de 0.005s du premier.

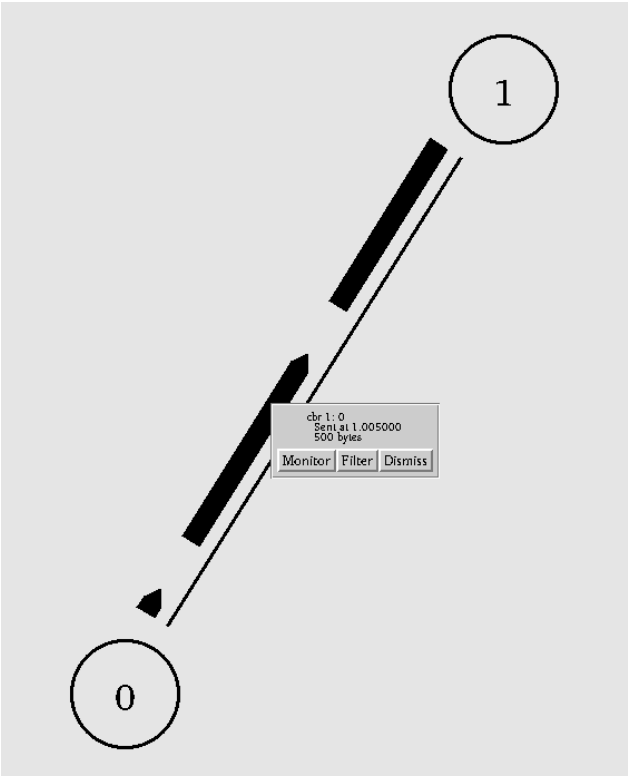

FIGURE 5 – Intervalle Exercice 1

On peut voir que l'intervalle est également respecté. Il manque seulement à vérifier si le dernier paquet est bien envoyé à 4.5s.

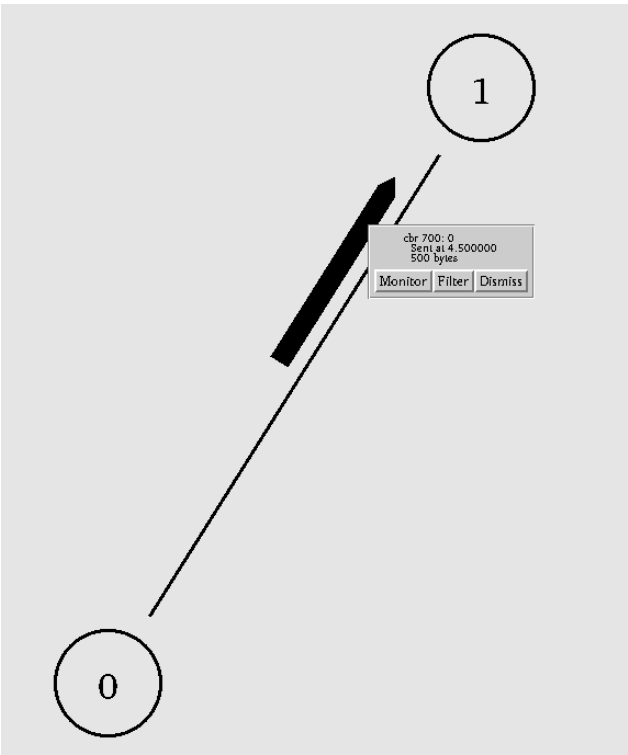

FIGURE 6 – Dernier Paquet Exercice 1

Ce paquet est bien envoyé à 4.5s. Tous les paramètres que l'on souhaitait sont donc bien configurés. En utilisant nam graphiquement on ne peut pas configurés la taille des paquets envoyés par le trafic CBR. On peut donc en conclure que faire les différents réseaux avec le mode graphique n'est pas une bonne idée car les différentes configurations de son pas forcément possible. Ecrire les scripts est donc beaucoup plus utile pour cela.

2.2.2 Exercice 2

On va maintenant s'intéresser à un réseau avec 4 nœuds pour simuler deux communications différentes en même temps.

```
26 $ns duplex-link $n0 $n2 2Mb 10ms DropTail
27 $ns duplex-link $n1 $n2 2Mb 10ms DropTail
28 $ns duplex-link $n2 $n3 1.7Mb 20ms DropTail
29
30 $ns duplex-link-op $n0 $n2 orient right-down
31 $ns duplex-link-op $n1 $n2 orient right-up
32 $ns duplex-link-op $n2 $n3 orient right
33
34 $ns queue-limit $n2 $n3 10
```

FIGURE 7 – Liens Exercice 2

Comme pour l'exercice précédent, on peut utiliser la même expression, mais on l'utilise plusieurs fois pour créer les différents liens. L'expression duplex-link-op permet d'orienter les nœuds les uns par rapport aux autres. Enfin, la ligne queue-limit permet de choisir la taille du buffer.

```
48 $ns connect $udp $Null
49
50 set tcp [new Agent/TCP]
51 $ns attach-agent $n0 $tcp
52
53 set ftp [new Application/FTP]
54 $ftp attach-agent $tcp
55
56 set sink [new Agent/TCPSink]
57 $ns attach-agent $n3 $sink
58
59 $ns connect $tcp $sink
```

FIGURE 8 – Agent TCP

Pour créer un Agent TCP, on peut utiliser le même principe qu'avec un agent UDP seulement, il ne faut pas créer un agent Null, car avec un agent TCP le nœud qui reçoit les paquets doit fournir des Ack à l'émetteur pour lui indiquer la bonne réception des paquets. Il faut donc à la place faire un agent TCPSink.

```
51 $ns color 1 Blue
52 $tcp set class_ 1
53
54 $ns color 2 Red
55 $udp set class_ 2
```

FIGURE 9 – Couleur Paquets

Pour réaliser cet exercice, il est demandé de colorier les flux différents avec des couleurs. Pour ce faire, il suffit d'écrire ces lignes. La première permet de définir la couleur bleue avec le nombre 1. On initialise ensuite l'agent TCP avec le numéro 1 pour indiquer la couleur bleue. On fait pareil pour l'agent UDP avec la couleur rouge.

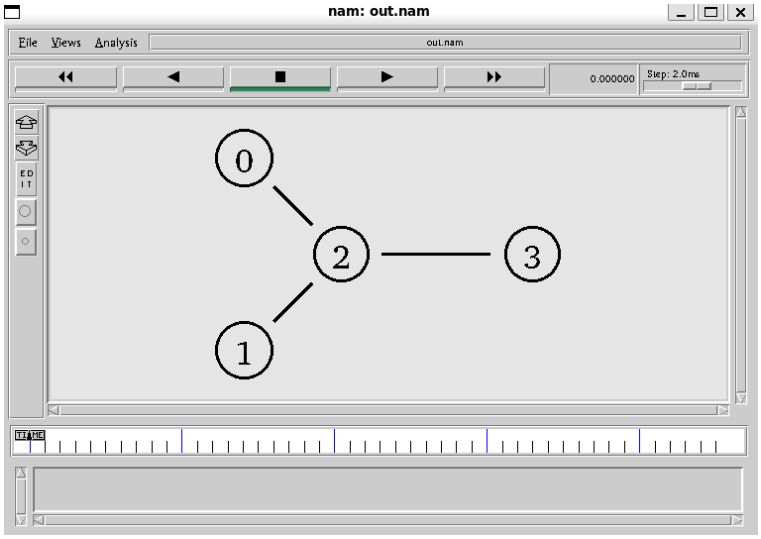

FIGURE 10 – Réseau Exercice 2

Avec l'aide des lignes permettant l'orientation des nœuds, on peut voir que le réseau est correctement configuré.

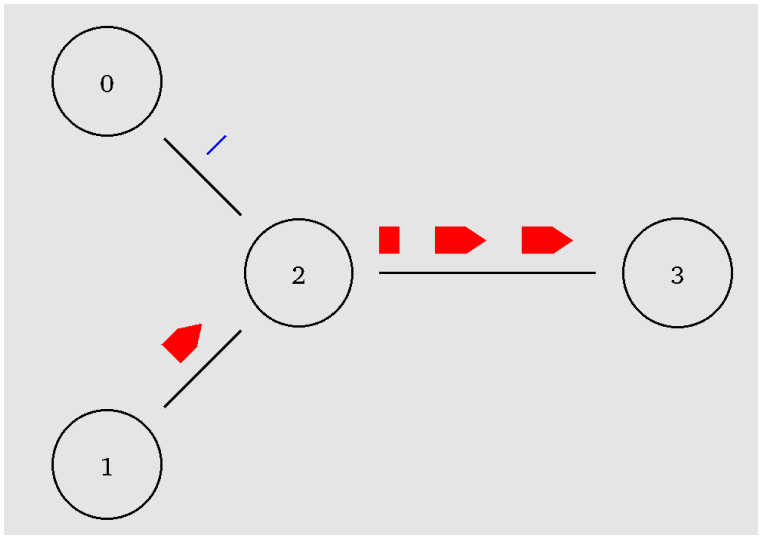

FIGURE 11 – Premier Paquet TCP

On peut voir sur cette capture les deux flux différents avec la couleur bleue pour TCP et le rouge pour l'UDP.

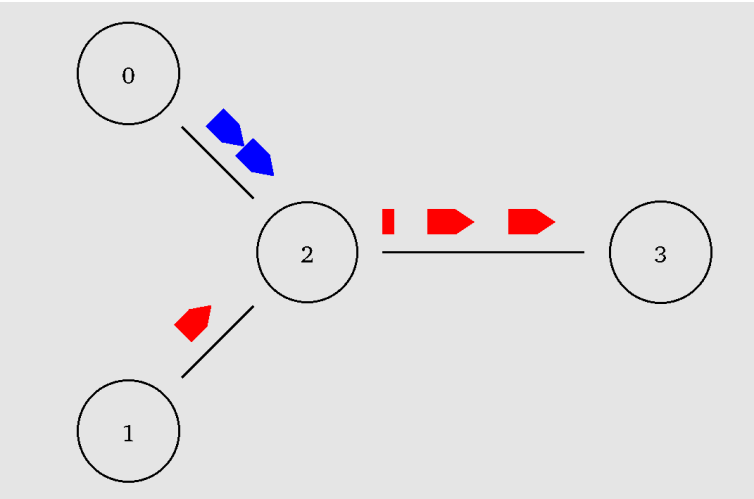

FIGURE 12 – Deuxieme Paquet TCP

On peut voir qu'en recevant l'ack, le nœud 0 envoi 2 paquets à la fois, c'est-à-dire le double. Il va ensuite à envoyer 4 puis 8...

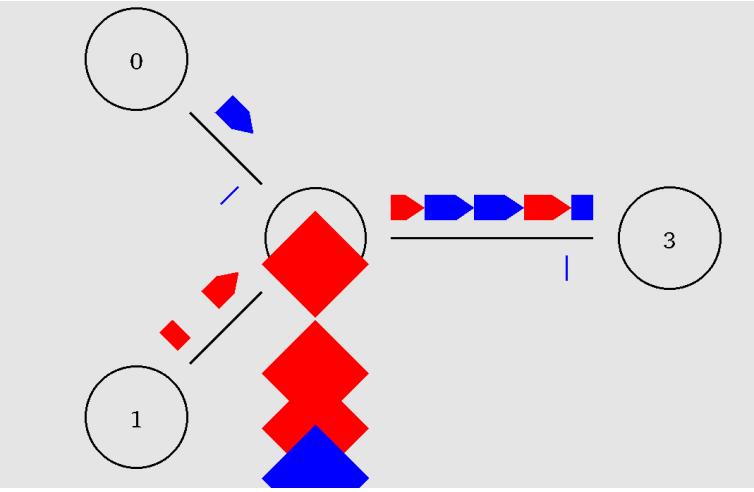

FIGURE 13 – Fenetre TCP

Avec cette capture, on peut voir la perte des paquets ainsi que la différence entre TCP et UDP. En effet, UDP continu d'envoyer autant de paquets qu'il y ait des pertes ou pas, car il ne peut pas connaître l'existence de pertes dans le réseau. Tandis que TCP à l'aide des ACK peut connaître l'existence de pertes dans le réseau. Ce qui permet de modifier la taille de la fenetre en fonction de l'état du réseau. Après avoir pris connaissance des pertes. L'agent TCP recommence à envoyer un seul paquet pour éviter les pertes. Dans ce réseau, on peut observer des pertes à cause du buffer que l'on a configuré au début du script. Quand le buffer est remplit, les paquets qui arrivent au nœud 2 sont dropper que ce soit les paquets TCP ou les paquets UDP.

2.3 Conclusion

Ce TP nous a permis de comprendre les différentes fonctionnalités de NS-2 ainsi que de nam à l'aide de deux exercices différents. Cela a permis de comprendre le fonctionnement d'un réseau avec un agent UDP ou un agent TCP ainsi que le principe de buffer dans un routeur ou un ordinateur. Les principes de base d'un réseau ont pu être vu dans ce TP avec par exemple le débit, l'intervalle entre les paquets envoyés.

3 TP2

3.1 Introduction

Le but de ce TP est de créer et de tester des réseaux avec des routages dynamiques tels que le routage à vecteur de distance et à état de lien. Après avoir fait ces réseaux, cela permettra de tester le fonctionnement de ces routages avec des ruptures dans le réseau et la différence entre ces deux types de routage dynamique avec leurs avantages. Dans un second temps, on testera un réseau avec un lien possédant un coût supérieur à 1 pour vérifier le fonctionnement du routage à vecteur de distance (RIP).

3.2 Mise en pratique

3.2.1 Exercice 3

Dans cet exercice on faire un réseau avec 8 nœuds différents et tester le fonctionnement de celui-ci avec les deux types de routage pendant une perte d'un lien ainsi que durant le rétablissement de ce lien.

Commençons par le routage à vecteur de distance. Pour simuler un routage dynamique à vecteur de distance on va avoir besoin de la ligne suivante :

$ns rtproto DV

Cela permet de simuler un routage DV (Distance Vector). Ce type de routage permet de construire les tables de routage où aucun routage n'a de vision globale du réseau.
Ensuite, pour créer un nombre de nœuds important, il est long d'écrire une ligne par nous dans le script. Pour éviter cela, il est possible de faire une boucle for pour automatiser la création de nous. Pour faire cette boucle, il suffit d'écrire l'expression suivante :

for set i 1 $i<=8 set i [expr $i+1] set n($i) [$ns node]

"set i 1" permet d'initialiser la variable i à 1. Ensuite, "$i<=8" permet de choisir le nombre d'itération ce qui permet de choisir le nombre de nœuds que l'on souhaite créer. "set n($i) [$ns node]", ceci est semblable à l'expression utilisée dans le TP1 pour créer les nœuds, seulement à la place de mettre le numéro du nœud il y a $i pour l'automatisation de la création des nœuds dans la boucle for.

```
$ns duplex-link $n(1) $n(3) 10Mb 10ms DropTail
$ns duplex-link $n(2) $n(3) 10Mb 10ms DropTail
$ns duplex-link $n(3) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(3) $n(5) 10Mb 10ms DropTail
$ns duplex-link $n(5) $n(8) 10Mb 10ms DropTail
$ns duplex-link $n(4) $n(6) 10Mb 10ms DropTail
$ns duplex-link $n(6) $n(7) 10Mb 10ms DropTail
$ns duplex-link $n(7) $n(8) 10Mb 10ms DropTail


$ns duplex-link-op $n(1) $n(3) orient rigth-down
$ns duplex-link-op $n(2) $n(3) orient left-down
$ns duplex-link-op $n(3) $n(4) orient rigth-down
$ns duplex-link-op $n(3) $n(5) orient left-down
$ns duplex-link-op $n(4) $n(6) orient rigth
$ns duplex-link-op $n(6) $n(7) orient left-down
$ns duplex-link-op $n(7) $n(8) orient left
$ns duplex-link-op $n(5) $n(8) orient down
```

FIGURE 14 – Liens Exercice 3

Cela permet la création des différents liens demandé dans l'exercice et l'orientation de ces liens les uns par rapport aux autres.
On créer ensuite les différents agents et sources demandés dans cet exercice en suivant la même méthode que dans le TP précédent. Il ne reste plus qu'à simuler la perte du lien entre le nœud 5 et 8 à 4s et le rétablissement de celui-ci à 5s. On utilise pour faire cela les lignes suivantes :

$ns rtmodel-at 4 down $n(5) $n(8)
$ns rtmodel-at 5 up $n(5) $n(8)

La premiere ligne permet d'indiquer de couper le lien entre $n(5) et $n(8) à t=4s avec le mot clé "down", alors que la deuxième ligne indique le rétablissement de ce lien à l'aide du mot clé "up". Cela donne le réseau suivant :

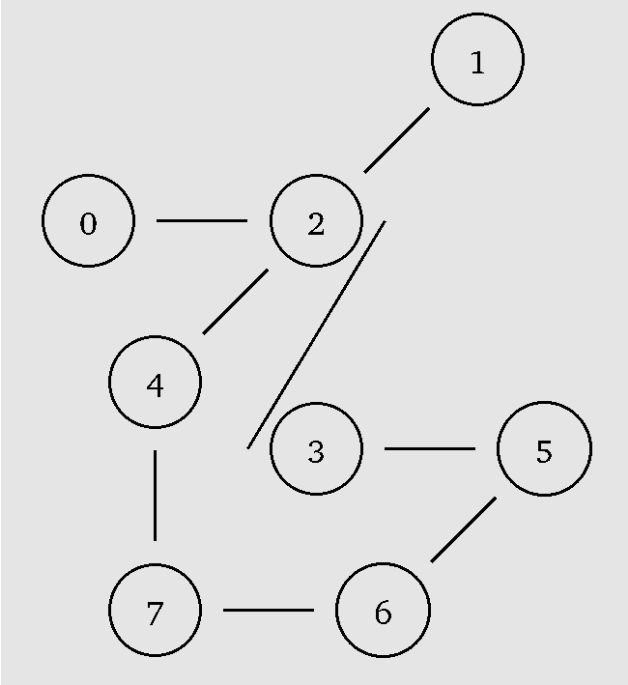

FIGURE 15 – Réseau Exercice 3

Regardons maintenant si la coupure du lien a bien lieu à 4 secondes.

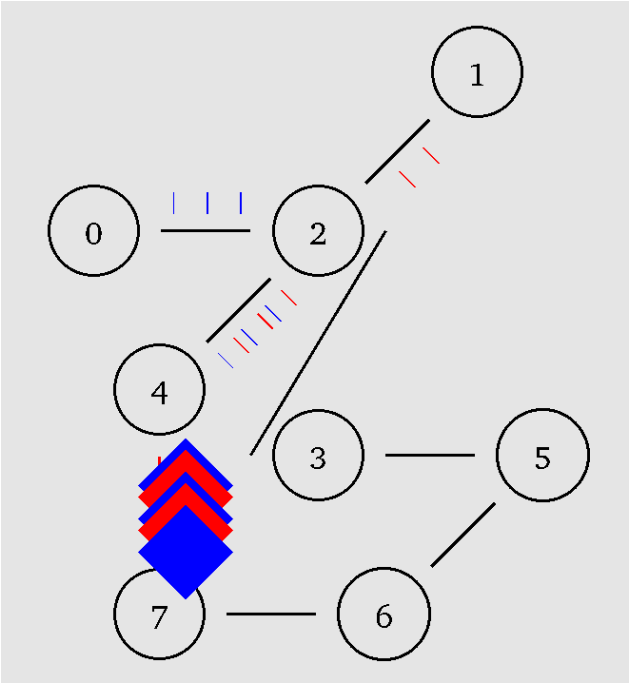

FIGURE 16 – Coupure Exercice 3

On peut voir que le lien est bel est bien coupé. Regardons aussi si le lien est rétablit apres 5 secondes du début de la simulation.

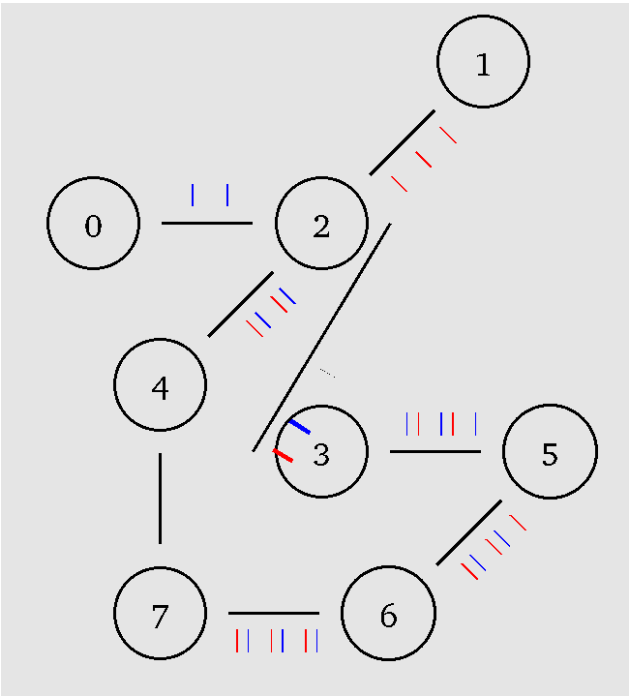

FIGURE 17 – Rétablissement Exercice 3

Maintenant que la coupure et que le rétablissement du lien entre le nœud 5 et 8, pour comprendre le fonctionnement du routage à vecteur de distance, il faut maintenant chercher les différents paquets de routage échangés par les différents routeurs du réseau.

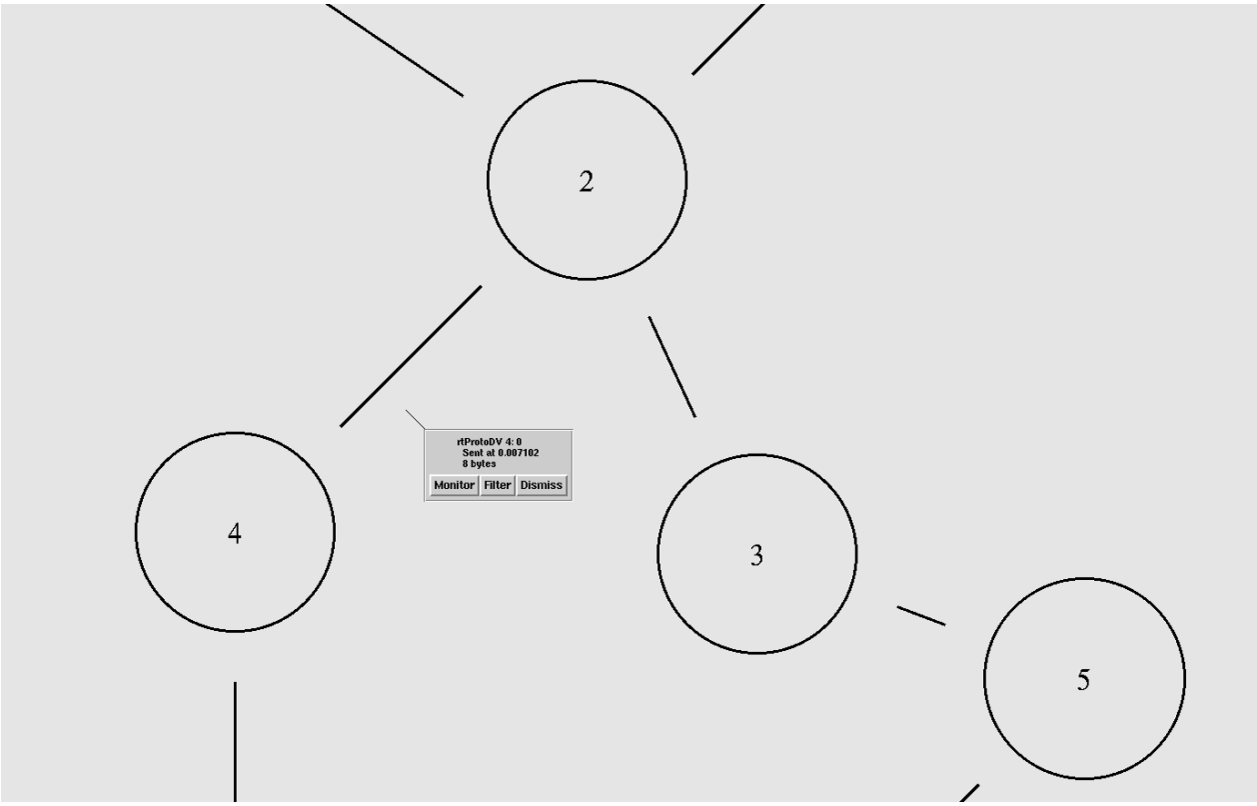

FIGURE 18 – Initialisation DV Exercice 3

On peut voir qu'au début de la simulation les différents routeurs s'échange des paquets pour échanger sur la configuration du réseau. Cela permet aux différents routeurs de remplir leur table de routage en mettant la distance vers chaque destination.

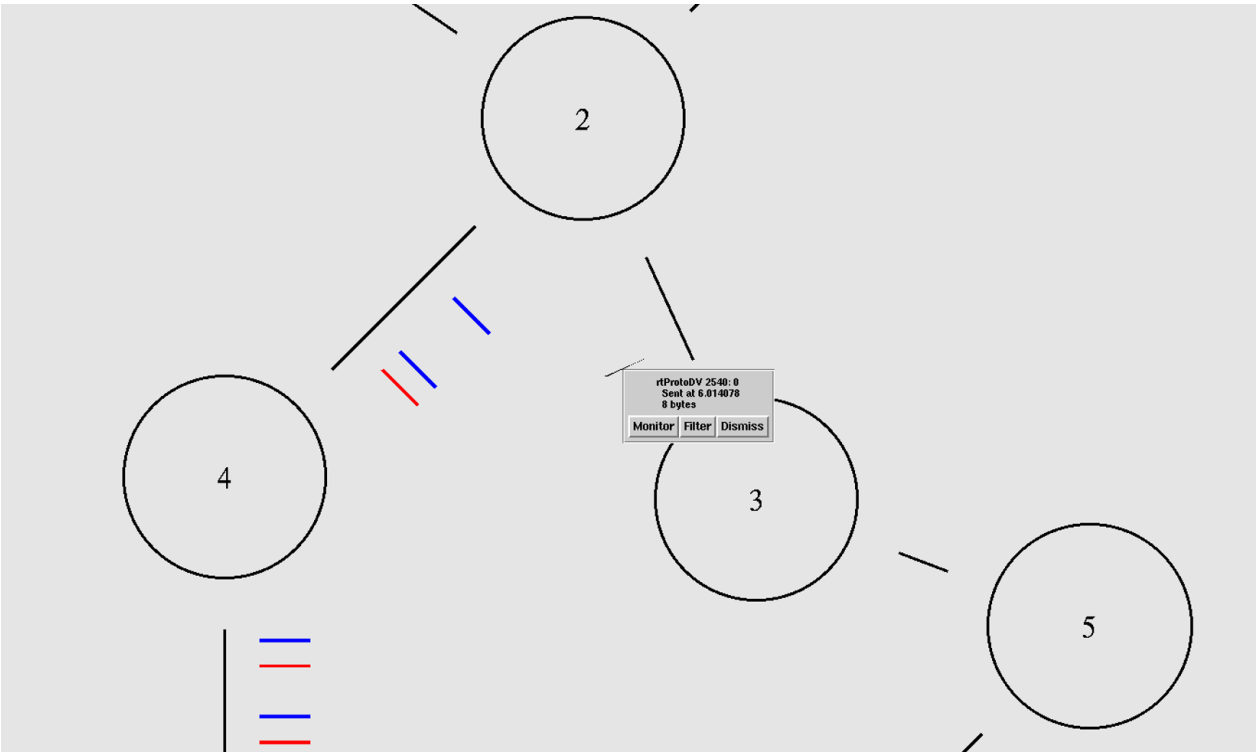

FIGURE 19 – Periodique DV Exercice 3

On peut également voir que tous les intervalles de temps des paquets de routage sont également échangé pour mettre à jour les tables des différents routages. Cela permet d'avoir les routes les plus optimisées et cela le plus souvent possible.

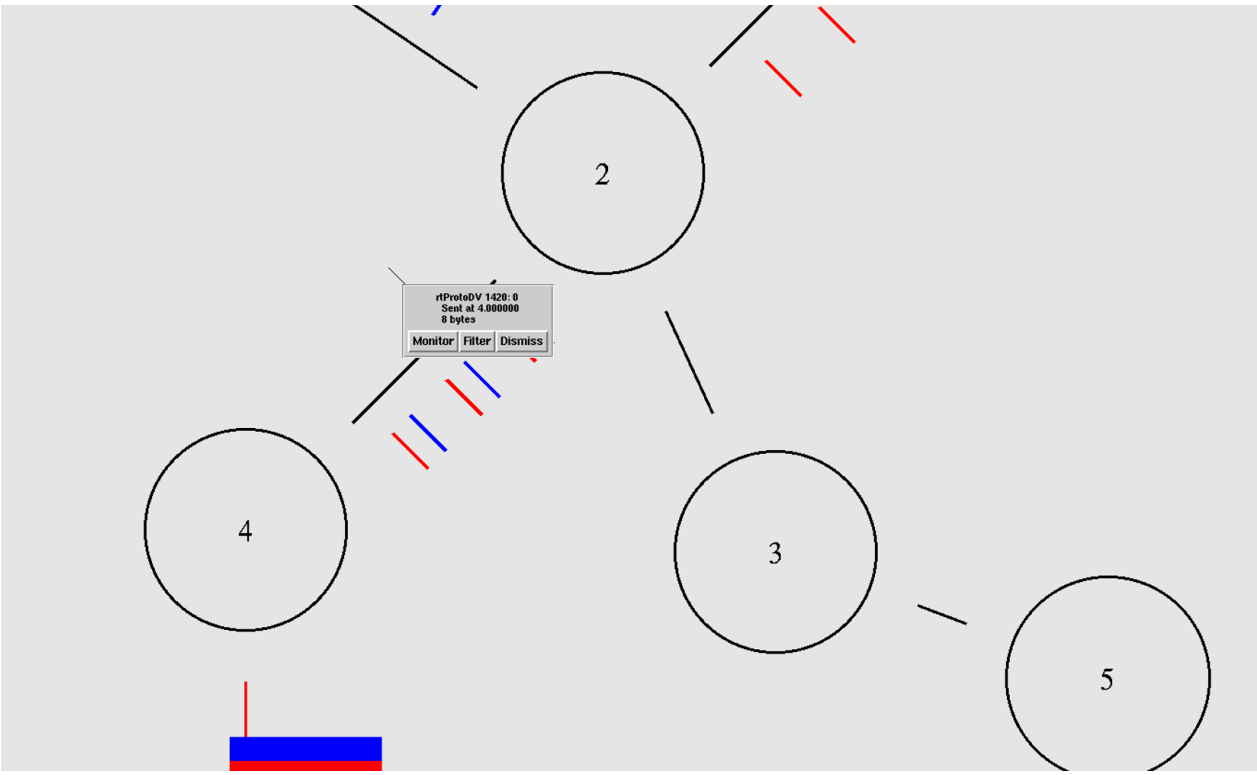

FIGURE 20 – Coupure DV Exercice 3

Des paquets sont également échangés au moment de la rupture du lien entre le nœud 5 et 8 pour informer tous les routeurs d'un changement dans le réseau. Cela permet aux paquets de passer par un autre chemin pour pouvoir toujours accéder à la destination voulue, sauf s'il n'y a plus aucun chemin possible pour l'atteindre.

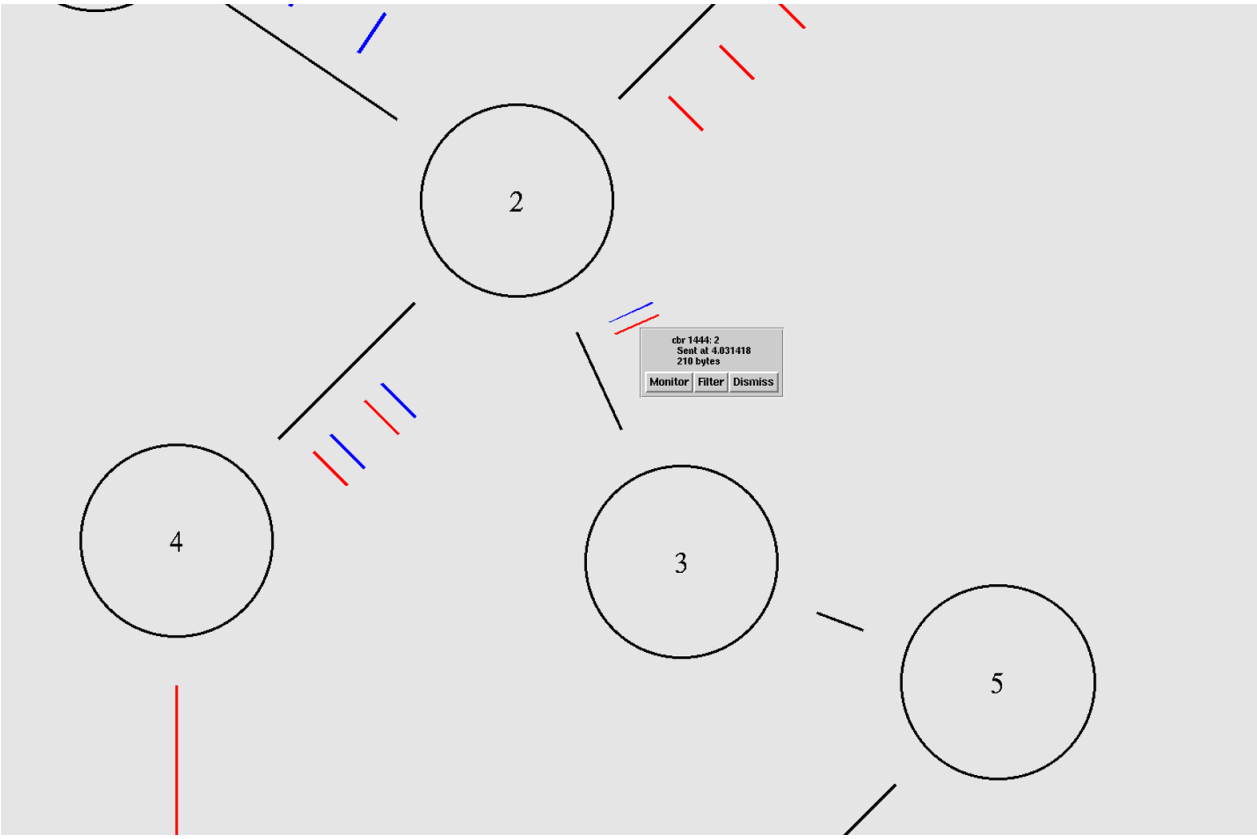

FIGURE 21 – Premier paquet rupture DV Exercice 3

On peut se rendre compte que le premier paquet à passer par un autre chemin n'est pas le paquet envoyé au moment de la rupture, mais un paquet qui a été envoyé après. Le premier paquet à passer par un autre chemin a été envoyé 30 ms après la coupure. Ce qui signifie que des paquets seront droppés et devront être renvoyé pour permettre au destinataire de recevoir la totalité de la communication.

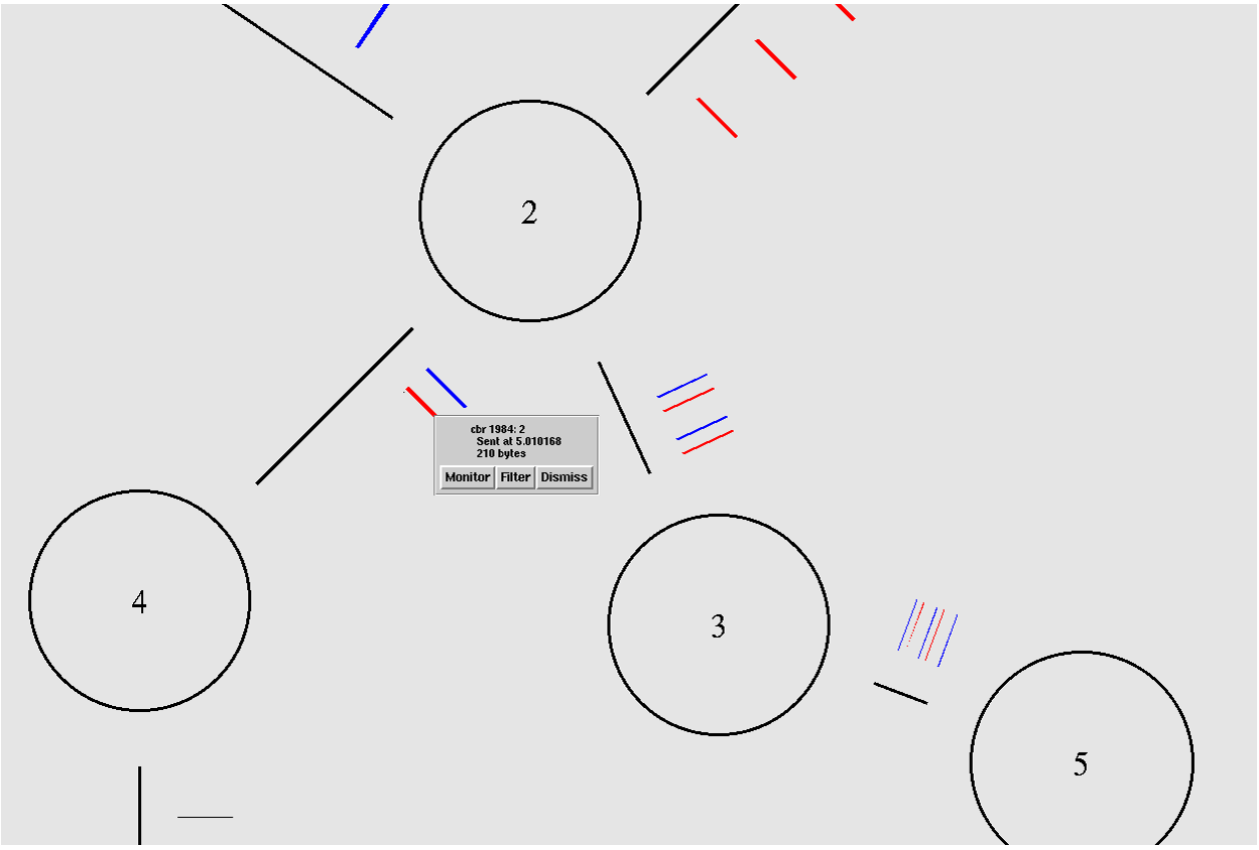

FIGURE 22 – Premier paquet rétablissement DV Exercice 3

Au moment du rétablissement du lien, il y a 10 ms de décalage pour permettre au premier paquet de repasser par le chemin initial. On peut donc se rendre compte que la convergence est trois fois plus rapide au moment du rétablissement qu'au moment de la rupture du lien.

Maintenant, on peut s'intéresser au routage à état de lien pour pouvoir comparer les deux types de routage et pour pouvoir comprendre les avantages de ces deux types de routage.
Pour simuler un réseau avec un routage à état de lien, il suffit de modifier la ligne suivante :

$ns rtproto DV
Il suffit de modifer le DV en LS :
$ns rtproto LS

Il ne reste plus qu'à chercher les différents paquets de routage échangés par les routeurs du réseau que l'on souhaite simuler.

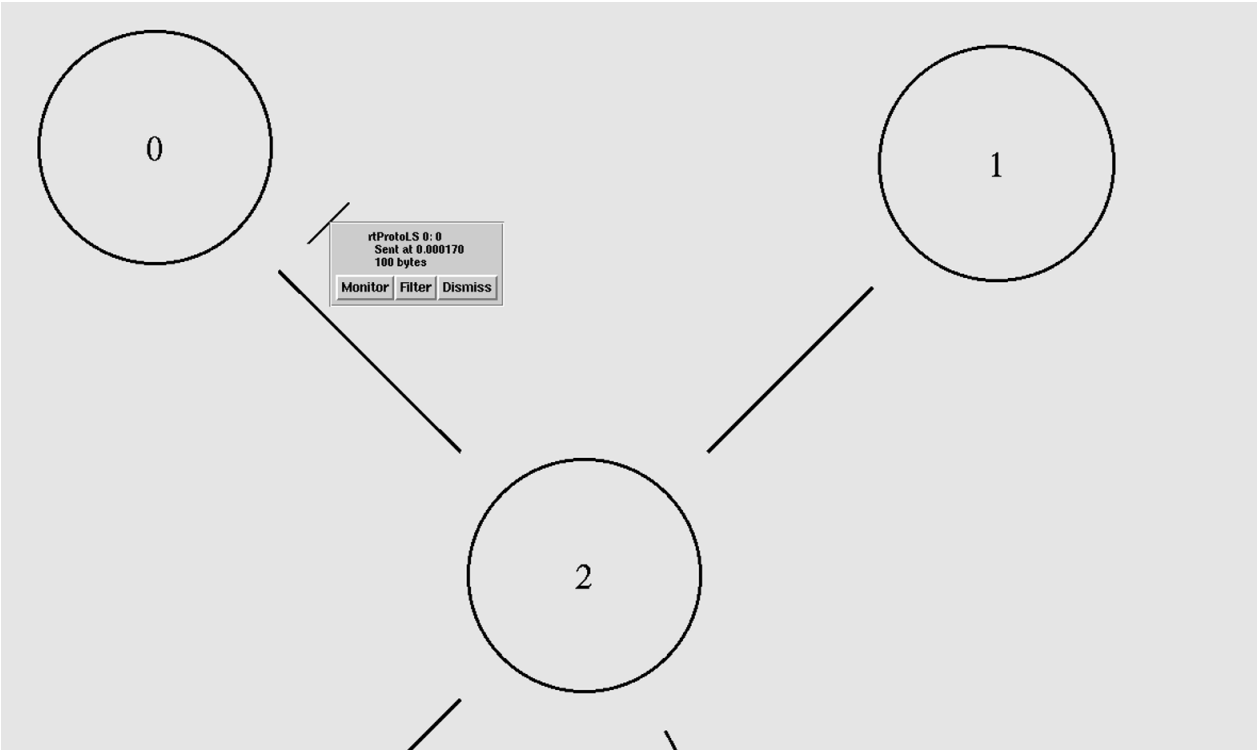

FIGURE 23 – Initialisation LS Exercice 3

Comme pour le routage à vecteur de distance, le routage à état de lien échange des paquets de routage au début de la simulation. Cela permet de compléter les tables des différents routeurs.

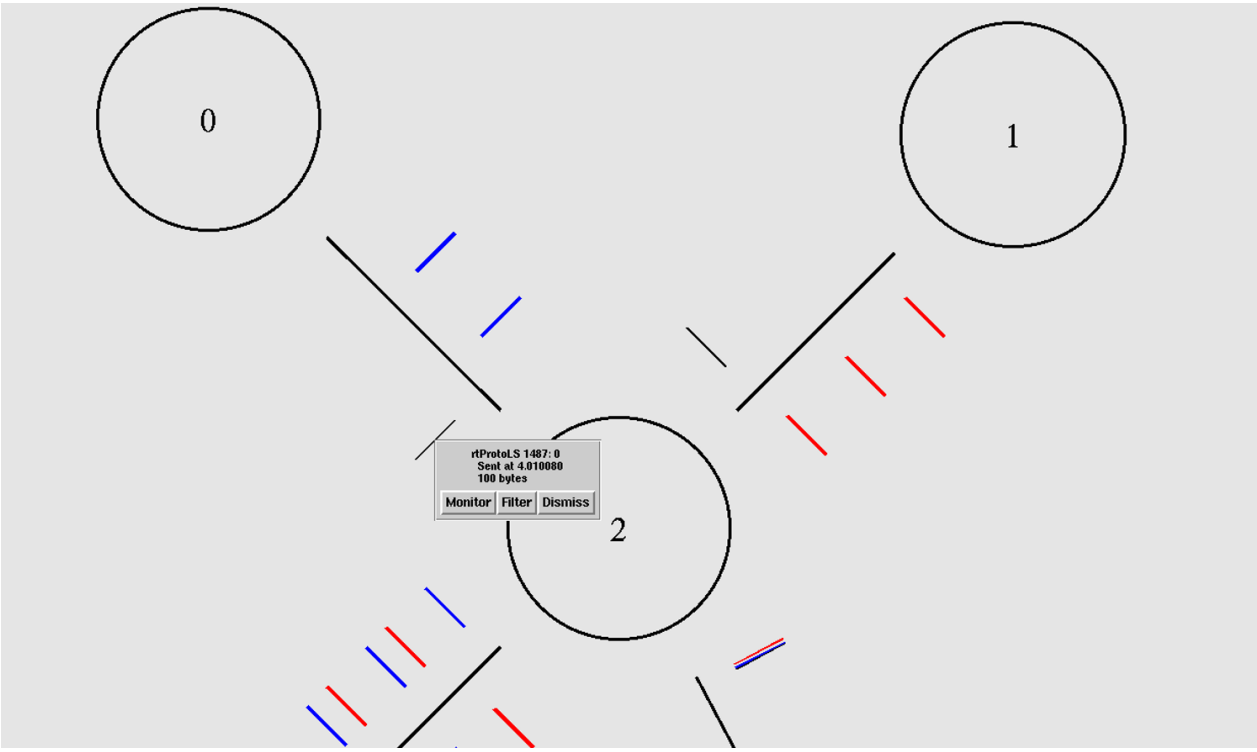

FIGURE 24 – Coupure LS Exercice 3

Il y a également des paquets de routage échangés lorsqu'il y a une rupture de lien dans le réseau. Cependant, à la différence du routage à vecteur de distance, il n'y a pas de paquets de routage échangés périodiquement. En effet, contrairement au routage à vecteur de distance, les différents routeurs du réseau ont connaissance de la globalité du réseau. Il n'est donc pas nécessaire d'avoir des mises à jour régulier, mais il reste tout de même nécessaire de s'échanger les informations lors d'un changement dans le réseau tel qu'une rupture de lien.

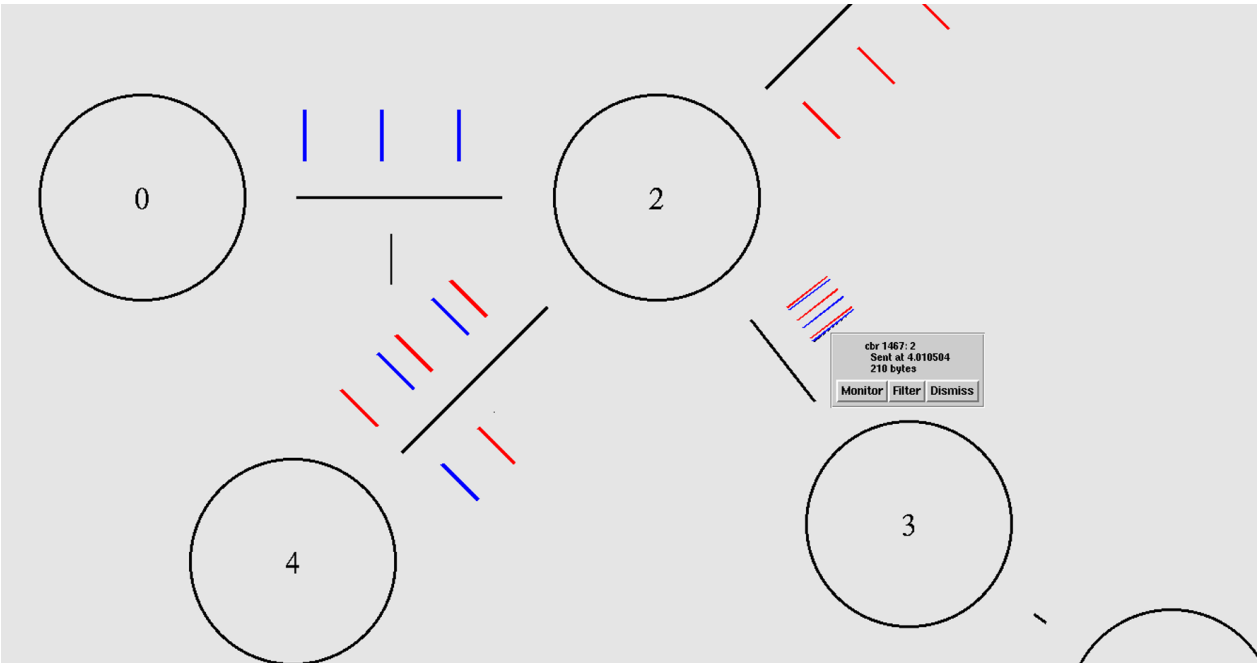

FIGURE 25 – Premier paquet coupure LS Exercice 3

Nous pouvons voir qu'avec un routage à état de lien il faut 10ms pour avoir un paquet sur un autre chemin.

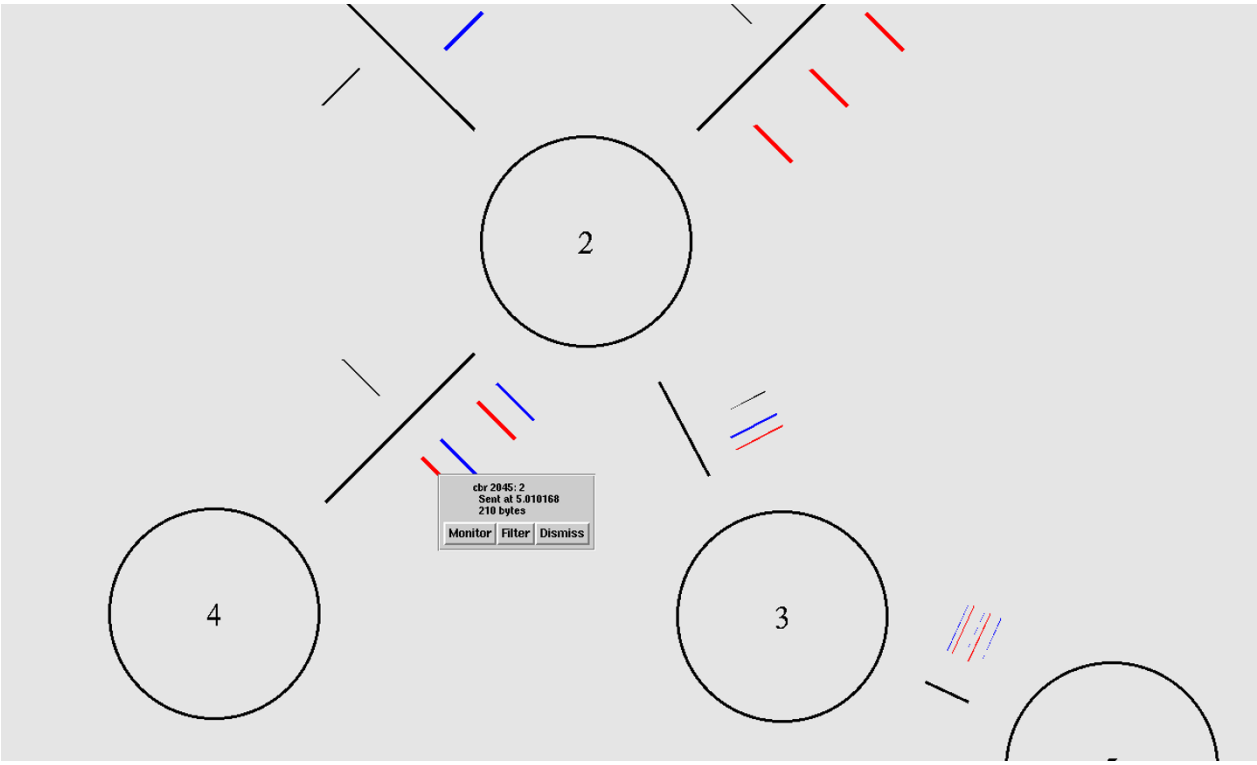

FIGURE 26 – Premier paquet rétablissement LS Exercice 3

Il y a également 10ms de latence pour permettre à un paquet de repasser par le chemin initial.

On peut donc maintenant comparer les deux types de routage dynamique. Tout d'abord, au moment de la coupure le temps pour avoir un paquet sur un nouveau chemin est trois fois plus rapide avec un routage à état de lien qu'avec un routage à vecteur de distance. De plus, avec un routage à état de lien, il n'y a pas de paquets de routage périodiques. On peut enfin également voir une différence de comportement durant la coupure du lien entre le nœud 5 et 8. En effet, avec un routage à vecteur de distance, les paquets sont droppés jusqu'au changement de chemin parcouru par les paquets. Tandis qu'avec un routage à état de lien, des paquets peuvent faire demi-tour pour passer par un autre chemin ce qui rend la destination atteignable.

3.2.2 Exercice 4

Le but de cet exercice est de simuler un réseau avec un protocole à vecteur de distance, plus précisément RIP. On va tester ce protocole sur un réseau qui subira une perte de lien à un moment donné puis ce lien sera rétabli. De plus, on va tester RIP sur un réseau qui possède un coût différent sur l'un de ces liens pour pouvoir observer le choix des routes. La seule métrique utilisée par RIP étant le nombre de sauts, pour pouvoir augmenter le coût entre deux nœuds du réseau, il va falloir ajouter des nœuds intermédiaires entre les deux.
Dans cet exercice, il est demandé de mettre des noms aux nœuds et de changer également la forme du nœud qui envoie et du nœud qui reçoit les paquets.

```
43 $n(1) label "A"
44 $n(2) label "B"
45 $n(3) label "C"
46 $n(4) label "D"
47
48 $n(1) shape box(hexagon)
49 $n(4) shape box(hexagon)
50
```

FIGURE 27 – Label et forme Exercice 4

Les quatres premiere lignes permettre de mettre aux noeuds 1, 2, 3 et 4 les noms A, B, C et D. Cela permettra de différencier les noeuds du réseau et ceux qui servent seulement à augmenter le coût entre C et D. Les deux autres lignes permettent de changer la forme d'un noeud (le 1 et le 4 ici) et de les mettres sous la forme d'un hexagon pour pouvoir faciliter la comprhéension du réseau. Il suffit après de configurer le réseau comme demandé en utilisant les lignes de commande précédemment utilisé en les modifiant pour cet exercice. Il suffit donc de créer les différents liens, les différents nœuds (dont 3 de plus pour avoir un coût de 4 entre C et D), l'agent TCP avec une application FTP. Il ne faut pas oublier de couper le lien entre B et D à t=3.48s et de le rétablir à t=4.95s avec les deux commandes utilisées dans l'exercice précédent.

On obtient donc le réseau suivant :

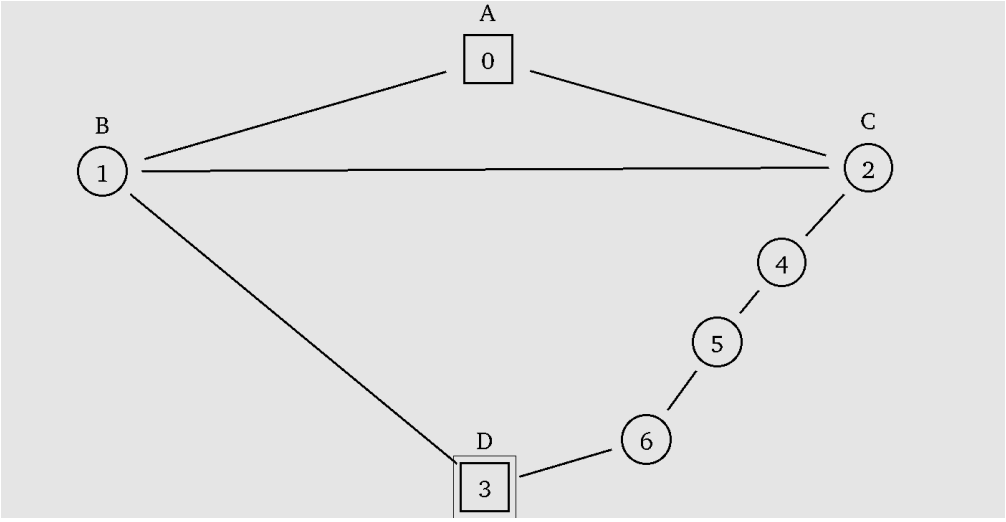

FIGURE 28 – Réseau Exercice 4

Vérifions maintenant le fonctionnement du réseau avant la rupture du lien entre B et D, après la rupture et enfin après le rétablissement de ce lien.

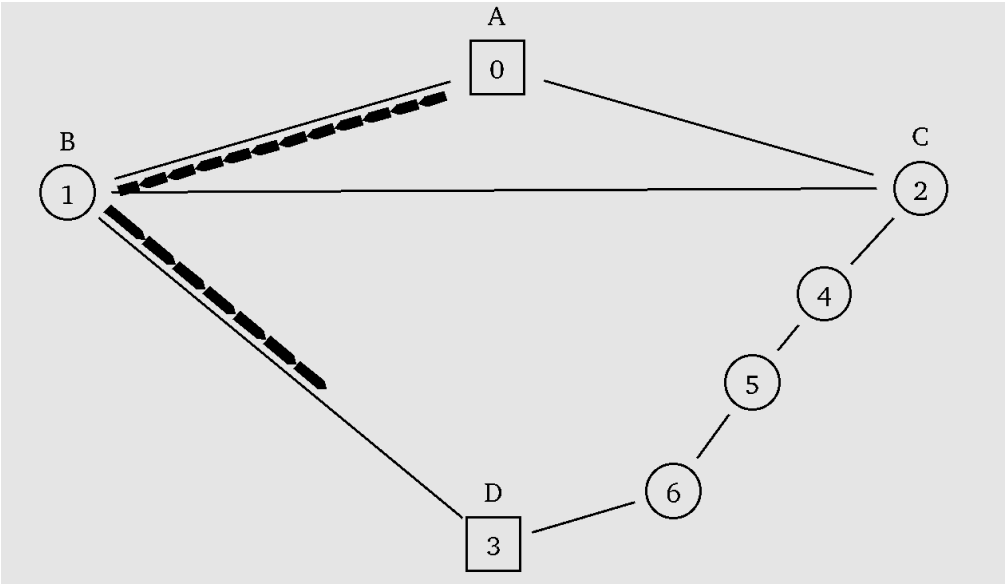

FIGURE 29 – Avant rupture du lien Exercice 4

Avant la rupture du lien, on peut voir que la communication entre A et D passe par le nœud B. En effet, le protocole de routage à vecteur de distance RIP utilise l'algorithme de plus court chemin Bellman-Ford. Cet algorithme ne prend en compte que le nombre de sauts. En passant par le nœud B, les paquets envoyés par A à destination de D n'auront qu'un seul nœud intermédiaire. En passant par C puis B par exemple, il y aura deux nœuds intermédiaires, et 4 pour la route de C vers D en passent par les nœuds 4, 5 et 6. La route A-B-D est donc celle choisie par le protocole RIP.

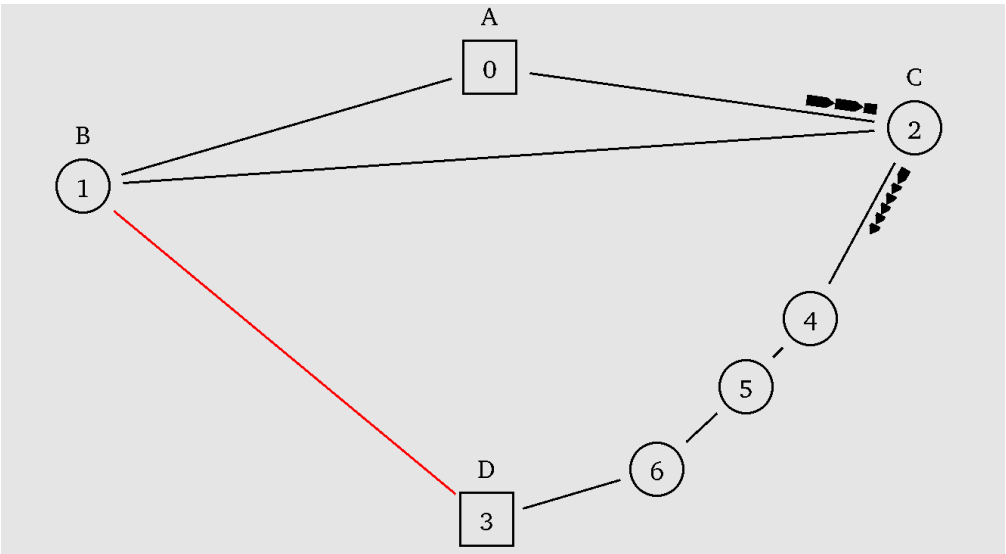

FIGURE 30 – Après rupture du lien Exercice 4

Quand la coupure à lieu les routeurs s'échangent des paquets de routage comme vu précédemment. Le nœud A envoi donc ses segments vers C pour avoir le plus cours chemin entre A et D. Avec l'absence de lien entre B et D, le plus court chemin devient A-C-4-5-6-D. Les segments passeront donc par cette route tant qu'il n'y aura pas de nouveaux changements dans le réseau.

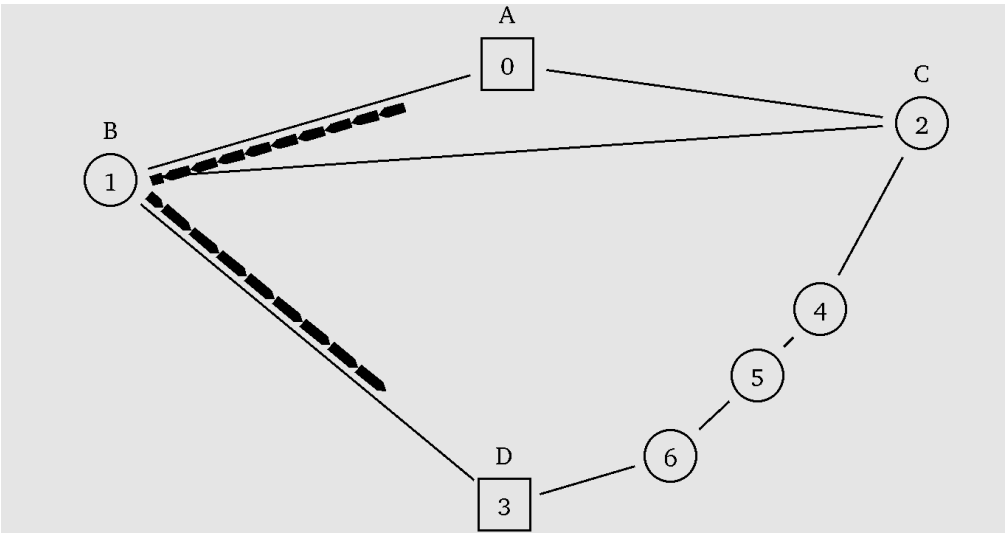

FIGURE 31 – Après rétablissement du lien Exercice 4

Après rétablissement du lien entre B et D, les segments envoyés par A repasseront par le nœud B, car cette route est redevenue accessible et est maintenant le plus court chemin entre A et D dans ce réseau.
Regardons aussi ce qu'il se passe avec les segments envoyés, plus précisement l'évolution de la taille de la fenêtre.

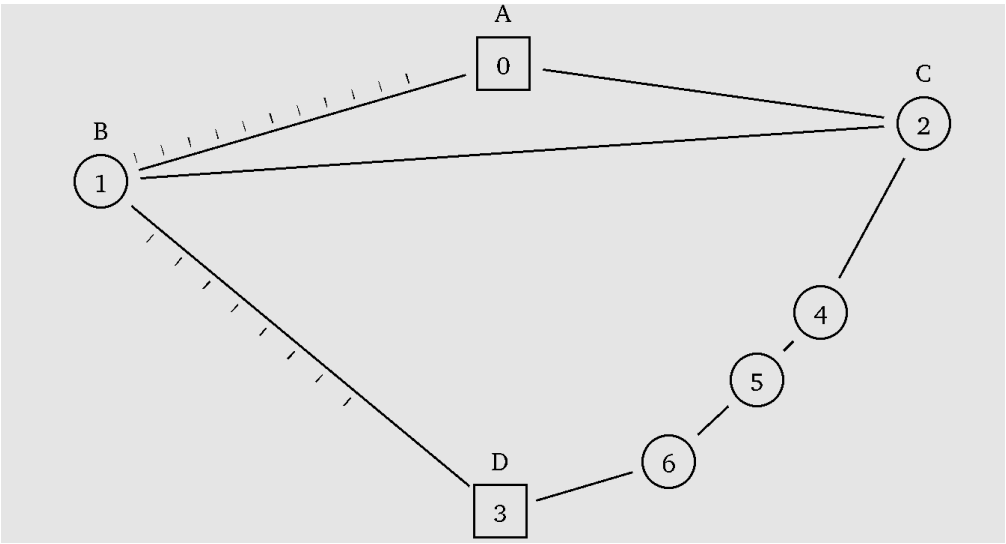

FIGURE 32 – ACK TCP Exercice 4

Avec le protocole TCP, D doit envoyer des ACK à A pour lui signifier qu'il a bien reçu ses segments. A peut donc renvoyer d'autres paquets en augmentant la fenêtre jusqu'à avoir un problème dans le réseau.
Pour pouvoir observer l'évolution de la fenêtre TCP nous allons prendre les différentes valeurs de la fenêtre TCP et faire un graphe sous Excel.

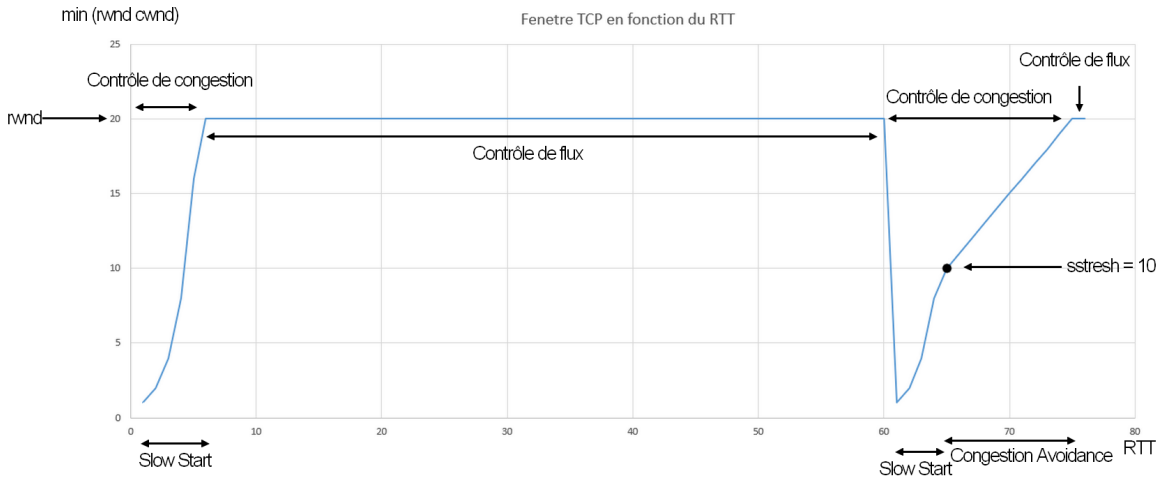

FIGURE 33 – Graphe fenêtre TCP Exercice 4

Pour commencer, le cwnd (congestion window) est la fenêtre de congestion du réseau. Le rwnd est la fenêtre du destinataire échangé lors de la connexion. Le sstresh quant à lui est la taille maximum d'une fenêtre d'émission en slow start, sachant que la taille de la fenêtre avant la perte d'un segment était de 20 le sstresh sera alors de Derniere_Taille/2 = 10.
Au début, la taille de la fenêtre double à chaque fois jusqu'à arrivé au rwnd qui bloque la taille de la fenêtre à 20. Au moment de la perte d'un segment, il y a deux possibilités, si le timer de A arrive à 0. la fenêtre repartira à 1. Tandis que si A reçois trois ACK dupliqués, la fenêtre repartira à la moitié de sa taille avant la perte. Dans notre cas, le timer de A arrive à 0 et c'est pour cela que l'on peut voir une remise à 1 de la fenêtre TCP. Il sera également en slow start (c'est-à-dire faire taille_fenetre_précédente x 2) jusqu'à atteindre le sstresh et passer ensuite en mode congestion Avoidance (c'est-à-dire taille_fenetre_précédente + 1) jusqu'à atteindre la valeur maximale possible si aucune perte n'est détectée (c'est-à-dire 20 dans notre cas).
On peut également tracer un graphe de ce style en utilisant xgraph. Il faut donc modifer le script pour permettre à celui-ci de tracer un graphe semblable à celui-ci.

```
 8 set windowVsTime [open WindowVsTime w]
 9 proc plotWindow {tcpSource windowVsTime} {
10         global ns
11         set time 0.1
12         set cwnd [$tcpSource set cwnd_]
13         set now [$ns now]
14         if {$cwnd >= 20} {
15                 set cwnd 20
16         }
17         puts $windowVsTime "$now $cwnd"
18         $ns at [expr $now+$time] "plotWindow $tcpSource $windowVsTime"
19 }
```

FIGURE 34 – Script pour xgraph Exercice 4

Cette partie du code permet de tracer l'évolution de la fenêtre TCP en fonction du temps. Plus précisement la valeur minimum entre le rwnd et le cwnd en fonction du temps.

Le if permet de bloquer la taille de la fenêtre au maximum à 20 car il faut prendre la valeur minimale entre le rwnd et le cwnd. De plus, ces lignes permettront de créer un fichier WindowVS-Time où il y sera écrit le graphe construit par les lignes suivantes.
Pour commencer la construction de ce graphe, il suffit de mettre cette ligne à la fin du script :

$ns at 0 "plotWindow $tcp $windowVsTime"

Cela permettra de commencer à remplir ce graphe à partir de t=0s. On obtient donc ceci :

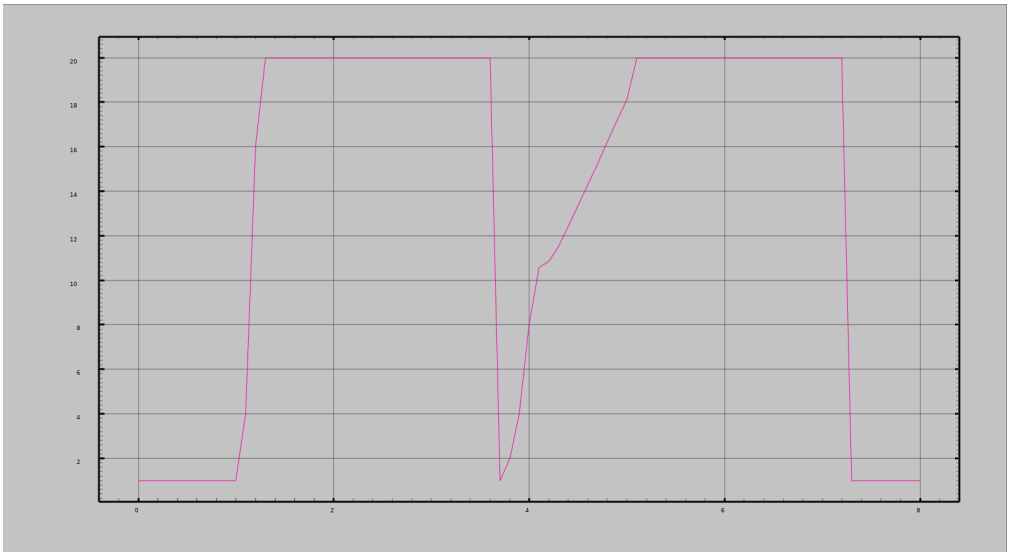

FIGURE 35 – Graphe avec xgraph Exercice 4

On peut voir que ce graphe est le même que celui tracé avec Excel en prenant les valeurs de la fenêtre à chaque échange. Il y a tout de même une différence. En effet, Le graphe tracé par script montre la fenêtre TCP en fonction du temps et non en fonction du RTT.

Dans ce scénario, la perte de segment TCP est provoquée par la rupture d'un lien dans le réseau. Dans l'exercice 2 du TP1 la perte était dû au buffer d'un nœud qui était remplis ce qui provoquait la perte de tous les paquets arrivant à ce nœud. D'autres raisons de perte de segment TCP dans un réseau sont possibles, si un erroné arrive au destinataire, il ne sera pas traiter et il faudra donc le réenvoyé. Il y a aussi par exemple l'absence de route pour atteindre le destinataire qui provoquera également la perte d'un segment TCP.

3.3 Conclusion

Pour finir avec ce TP, celui-ci nous a permis de comprendre le fonctionnement d'un routage d'dynamique dans un réseau. De plus, on a également pu voir la différence entre le routage à vecteur de distance et le routage à état de lien et voir les avantages de ces deux types de routage dans un réseau. Et enfin, le fonctionnement du protocole de routage RIP a également pu être mis en pratique dans un exercice pour comprendre la notion de coût sur un lien.

4 TP3

4.1 Introduction

L'objectif de ce dernier TP est de comprendre un script déjà existant et d'utiliser xgraph pour visualiser les performances d'un réseau. Pour faire cela, on va diviser ce TP en trois parties. Dans un premier temps, on se concentrera sur la compréhension globale du script pour comprendre son fonctionnement et pour comprendre également par la suite le graphe. Dans un second temps, On regardera rapidement avec nam le fonctionnement du réseau que l'on souhaite simuler. .Enfin, dans un troisième temps, on se concentrera sur l'analyse de graphes à l'aide du logiciel xgraph. Cela permettra d'analyser les performances du réseau simulé par ce script.

4.2 Mise en pratique

4.2.1 Compréhension du script

```
1 #Création d'une instance de l'objet Simulator
2 set ns [new Simulator]
3
4 #Ouvrir le fichier trace pour nam
5 set nf [open out.nam w]
6 $ns namtrace-all $nf
7
8 #Création de fichier pour xgraph
9 set f0 [open out0.tr w]
10 set f1 [open out1.tr w]
11 set f2 [open out2.tr w]
12
13 #Création de noeuds
14 set n0 [$ns node]
15 set n1 [$ns node]
16 set n2 [$ns node]
17 set n3 [$ns node]
18 set n4 [$ns node]
19
20 #Creation de lien entre les différents noeuds
21 $ns duplex-link $n0 $n3 1Mb 100ms DropTail
22 $ns duplex-link $n1 $n3 1Mb 100ms DropTail
23 $ns duplex-link $n2 $n3 1Mb 100ms DropTail
24 $ns duplex-link $n3 $n4 1Mb 100ms DropTail
```

FIGURE 36 – Analyse du début du script Exercice 5

Les trois lignes (set f0 [open out0.tr w], ...) permettent d'ouvrir en écriture trois fichiers différents pour pouvoir par la suite y mettre les différents graphes. On créer ensuite les différents nœuds et les différents liens entre ces nœuds pour créer le réseau.

```
#Définir la procédure de terminaison de la simulation
proc finish {} {
        global f0 f1 f2
        global ns nf
        #Fermer le fichier trace
        close $nf
        #Exécuter le nam avec en entrée le fichier trace
        exec nam out.nam
        #Close the output files
        close $f0
        close $f1
        close $f2
        #Executer xgraph
        exec xgraph out0.tr out1.tr out2.tr -geometry 800x400 & exit 0
}
```

FIGURE 37 – Analyse de la procédure finish du script Exercice 5

Les lignes "global ..." permettent d'utiliser les différentes variables définies hors de la procédure à l'intérieur de celle-ci. On exécute également à la fin xgraph pour tracer sur un même graphes les différentes courbes obtenues dans le script.

```
42 #Définir la procédure attach-expoo-traffic qui permet d'attacher l'application exponential à un agent UDP
43 proc attach-expoo-traffic { node sink size burst idle rate color number} {
44         set ns [Simulator instance]
45
46         #Création d'agent UDP et d'une application Exponential
47         set source [new Agent/UDP]
48         $ns attach-agent $node $source
49
50         set traffic [new Application/Traffic/Exponential]
51         $traffic set packetSize_ $size
52         $traffic set burst_time_ $burst
53         $traffic set idle_time_ $idle
54         $traffic set rate_ $rate
55
56         $traffic attach-agent $source
57         $ns connect $source $sink
58
59
60         $ns color $number $color
61         $source set class_ $number
62         return $traffic
```

FIGURE 38 – Analyse de la procédure attach-expoo-traffic du script Exercice 5

Cette procédure permet de créer un agent UDP ainsi qu'un trafic exponential et de les attachés ensemble. Le faire dans une procédure permet de faciliter le script quand on souhaite par la suite créer plusieurs trafics exponential dans le réseau. Il suffira simplement d'appeler la procédure en lui indiquant les valeurs des différents paramètres. Le premier paramètre à indiquer est le nœud sur lequel l'on souhaite attacher l'agent UDP ainsi que le trafic. On indique également avec Sink de deuxième nœud, c'est-à-dire celui qui recevra le flux envoyé par nous. Size permet de définir la taille des paquets envoyés. Ensuite, burst permet de définir le temps moyen pendant lequel le trafic est actif. idle quant à lui permet d'indiquer de temps moyen de silence de ce même trafic. rate permet également de définir le débit du trafic pendant son activité. Pour finir, les deux derniers paramètres (color number) permettent de choisir la couleur du trafic sur le réseau simulé avec nam.

```
#Définir la procédure record pour prendre des mesures pour tracer les graphes
proc record {} {
        global sink0 sink1 sink2 f0 f1 f2
        set ns [Simulator instance]
        #Definition de l'intervalle
        set time 0.5
        #Recuperation de bytes_ des sink
        set bw0 [$sink0 set bytes_]
        set bw1 [$sink1 set bytes_]
        set bw2 [$sink2 set bytes_]
        set now [$ns now]
        #Calcul du débit
        puts $f0 "$now [expr $bw0/$time*8/1000000]"
        puts $f1 "$now [expr $bw1/$time*8/1000000]"
        puts $f2 "$now [expr $bw2/$time*8/1000000]"
        #Réinitialisation de la taille de bytes_ à 0
        $sink0 set bytes_ 0
        $sink1 set bytes_ 0
        $sink2 set bytes_ 0
        $ns at [expr $now+$time] "record"
}
```

FIGURE 39 – Analyse de la procédure record du script Exercice 5

Cette procédure permet de prendre les différentes mesures nécéssaires pour pouvoir tracer les différents graphes.
"set bw0 [$sink0 set bytes_]" permet de récuperer la valeur de bytes_ (la taille en octet) de sink0.
"puts $f0 "$now [expr $bw0/$time*8/1000000]"" permet de calculer le débit et de le mettre dans le pointeur f0. Le débit est calculé initialement en octets/s. Cependant, en multipliant par 8, cela permet de calculer le débit en bits/s. Enfin, le diviser par 1000000 permet finalement de calculer le débit en Mb/s.
On réinitialise ensuite la taille des différents sink à 0 pour que cela n'influence pas sur la prochaine itération de cette procédure. Enfin, "$ns at [expr $now+$time] "record"" permet d'appeler cette procédure tous les "time" secondes pour pouvoir avoir plusieurs points pour tracer les graphes.

```
set sink0 [new Agent/LossMonitor]
set sink1 [new Agent/LossMonitor]
set sink2 [new Agent/LossMonitor]
$ns attach-agent $n4 $sink0
$ns attach-agent $n4 $sink1
$ns attach-agent $n4 $sink2

set source0 [attach-expoo-traffic $n0 $sink0 200 2s 1s 100k Blue 1]
set source1 [attach-expoo-traffic $n1 $sink1 200 2s 1s 200k Red 2]
set source2 [attach-expoo-traffic $n2 $sink2 200 2s 1s 300k Green 3]



$ns at 0.0 "record"
#Start the traffic sources
$ns at 10.0 "$source0 start"
$ns at 10.0 "$source1 start"
$ns at 10.0 "$source2 start"
$ns at 50.0 "$source0 stop"
$ns at 50.0 "$source1 stop"
$ns at 50.0 "$source2 stop"
#Appeler la procédure de terminaison apres un temps de 60s
$ns at 60.0 "finish"

#Exécuter la simulation
$ns run
```

FIGURE 40 – Analyse de la fin du script Exercice 5

Dans cette dernière partie du script, on crée d'abord les différents sink que l'on attaches au nœud 4. On créer ensuite les différentes sources en appelant la procédure attach-expoo-traffic avec leurs différents paramètres comme expliqué précédemment. On lance ensuite la procédure record ainsi que les différentes sources, puis on arrêtes ces sources à un moment donné. Enfin, on exécute la simulation avec $ns run.

4.2.2 Analyse du réseau

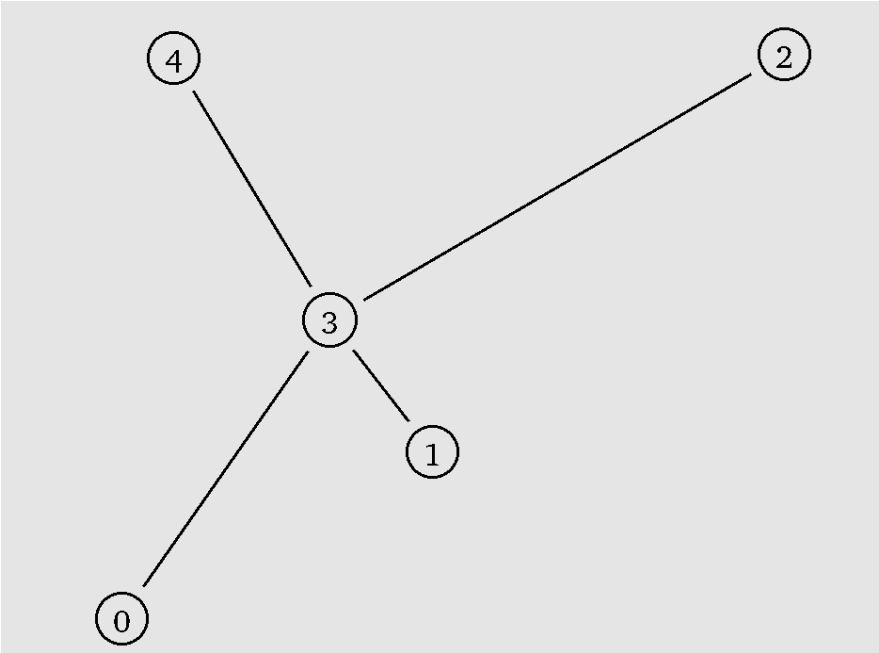

FIGURE 41 – Réseau Exercice 5

On peut observer les différents nœuds et liens que l'on souhaitait créer pour ce réseau. Il faut maintenant regarder si les différents trafics fonctionnent correctement.

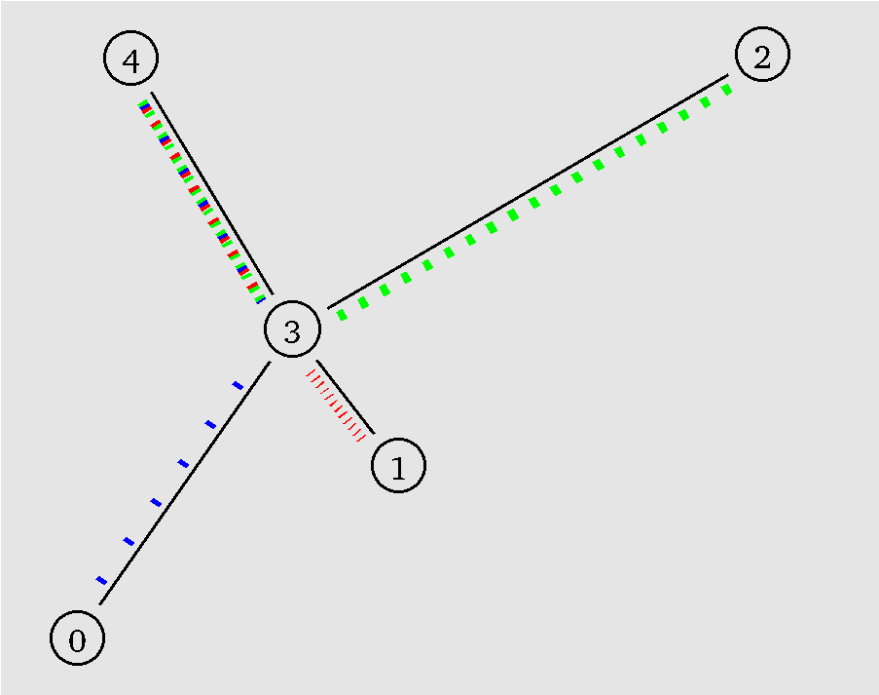

FIGURE 42 – Trafics réseau Exercice 5

On peut voir sur cette capture d'écran le fonctionnement simultané des trois trafics différents. Comme l'on souhaitait, les trafics sont affichés avec des couleurs différentes pour faciliter la compréhension du réseau. Le temps pendant lequel les différents nœuds envoient des paquets est calculé en fonction du temps burst que l'on a choisi lors de la création des différentes sources. Ce temps moyen, pendant lequel le trafic est actif, permettra aux nœuds d'envoyer leurs paquets pendant un temps aléatoire autour de ce temps moyen.

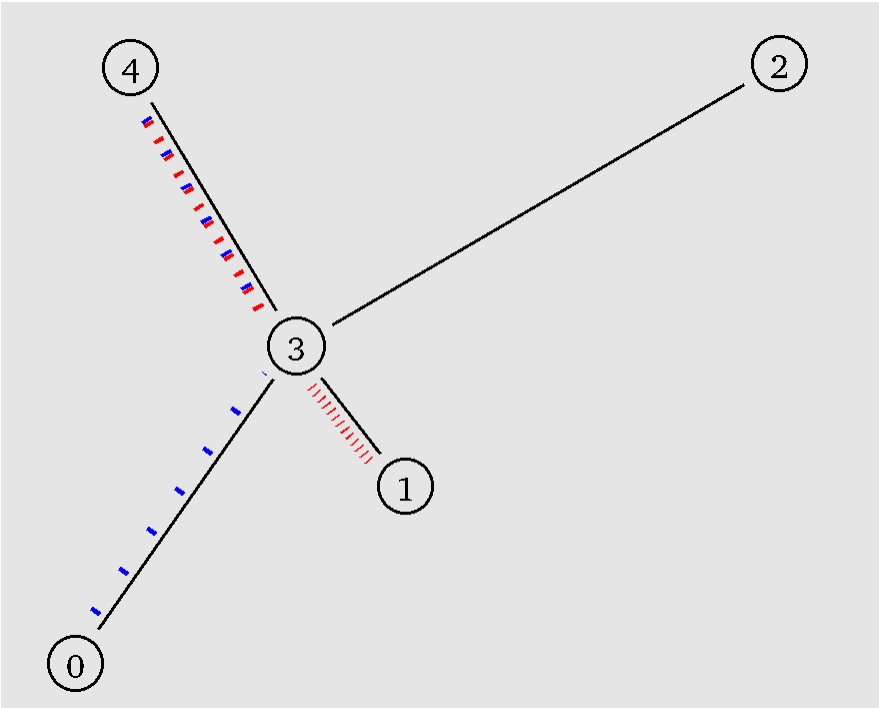

FIGURE 43 – Silences réseau Exercice 5

Comme l'on peut voir ici, le nœud 2 a arrêté l'envoi de ses paquets vers le nœud 4. Cela s'explique par le paramètre idle lors de la création des sources. En effet, sachant que la variable idle correspond au temps moyen de silence du trafic, il est donc normal d'avoir des moments où les différents nœuds n'envoient plus aucun paquet vers le nœud 4.

4.2.3 Analyse des performances

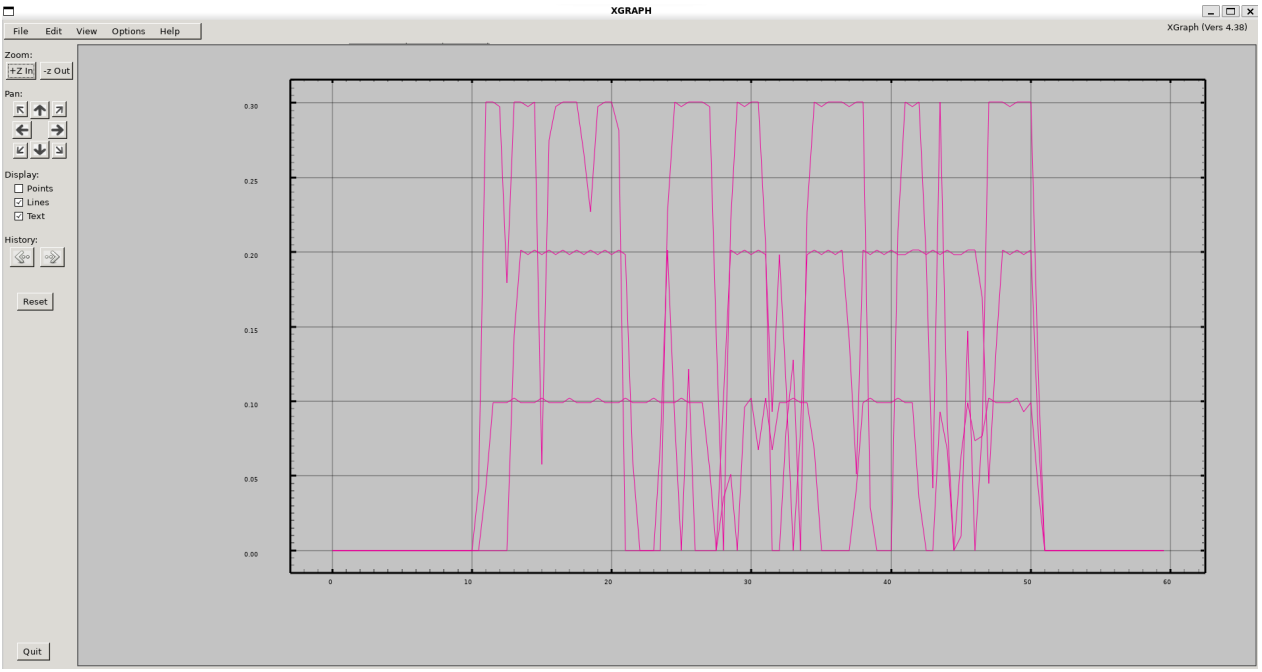

FIGURE 44 – xgraph avec t=0.5s Exercice 5

On peut voir ici l'évolution du débit en Mb/s en fonction du temps pour les trois trafics. On peut également remarquer le retour à zéro des débits à des moments donnés. Cela s'explique par le fait que les sources arrêtent d'envoyer des paquets à certains temps. L'arrêt de trafic provoque alors un débit de 0 qui se voit sur le graphe avec un retour à zéro.

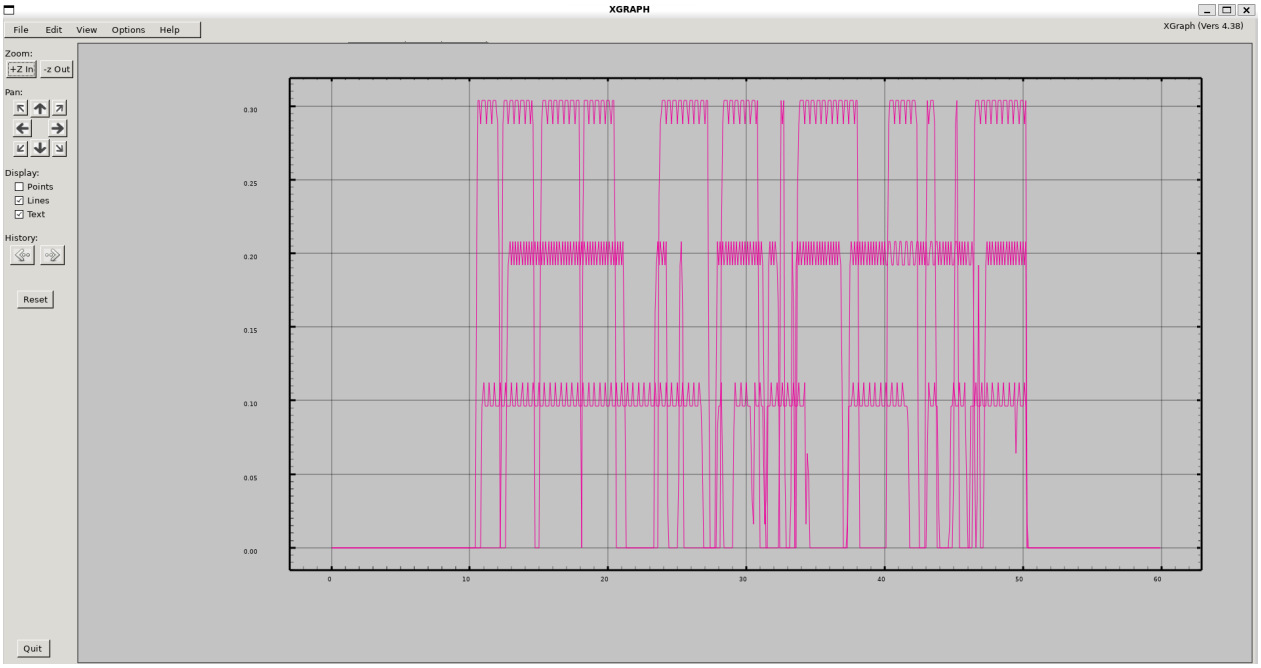

FIGURE 45 – xgraph avec t=0.1s Exercice 5

En modifiant la valeur de la variable time on peut voir une modification du graphe. En effet, la variable temps correspond à l'intervalle entre deux points du graphe. Ce qui signifie qu'en passant du time = 0.5s à 0.1s le nombre de points de mesure sur le graphe a augmenter par 5, ce qui permet d'avoir un graphe plus précis. Pour résumer, plus la valeur de time est petite, plus il y aura de points de mesure sur le graphe et augmentera donc la précision de celui-ci. Cependant, cela augmente le nombre de données, on ne peut donc pas prendre une valeur de time trop petite, sinon cela saturerai la mémoire de l'ordinateur.
Pour éviter des retours à 0 sur le graphe il faudrait choisir une valeur de time supérieure à 1. En effet, en regardant dans le script, les différentes valeurs de idle sont toutes égales à 1s. Donc, pour éviter d'apercevoir ces retours à 0, un intervalle de temps entre deux mesures supérieur à 1s est donc nécessaire.

4.3 Conclusion

Ce TP nous a permis d'analyser un script déjà fait, ce qui permet de tester nos connaissances sur le langage TCL et également de pouvoir apprendre de nouvelles fonctionnalités de ce langage. De plus, il nous a également permis de comprendre l'utilisation du logiciel xgraph ainsi que son fonctionnement avec l'analyse des performances d'un réseau. Enfin, on a également pu observer l'évolution du débit dans un réseau et remarqué que celui-ci n'est absolument pas constant dans le temps.

5 Conclusion Générale

Ces différents TP nous, on permit d'apprendre les bases du langage TCL. De plus, on a également pu apprendre le fonctionnement de plusieurs logiciels tels que ns-2, nam ou xgraph par exemple. Ces TP nous on également permit de comprendre de manière plus concrète le fonctionnement de différents réseaux. En effet, on a testé des réseaux avec des routages différents, on a pu tester le routage statique ainsi que le routage dynamique avec le routage à vecteur de distance, comme RIP par exemple et le routage à état de lien comme IS-IS.
La notion de performance dans un réseau a également pu être traiter dans ces différents réseaux, dont l'analyse du débit en fonction de temps ou encore le fonctionnement de ces réseaux face à une perte ou une rupture de lien.
En résumé, ces TP nous ont permis de mettre en application les différentes notions vues en cours, ce qui permet de les comprendre ou de mieux comprendre leurs principes.

6 Annexes

6.1 Scripts TP1

6.1.1 Script Exercice 1

```
#Cr ation d'une instance de l'objet Simulator
set ns [new Simulator]

#Ouvrir le fichier trace pour nam
set nf [open out.nam w]
$ns namtrace-all $nf

#Definir la proc dure de terminaison de la simulation
proc finish {} {
        global ns nf
        $ns flush-trace
        #fermer le fichier trace
        close $nf
        #Ex cuter le nam avec en entr e le fichier trace
        exec nam out.nam &
        exit 0
}

set n(0) [$ns node]
set n(1) [$ns node]

$ns duplex-link $n(0) $n(1) 1Mb 10ms DropTail

set UDP [new Agent/UDP]
$ns attach-agent $n(0) $UDP

set CBR [new Application/Traffic/CBR]
$CBR set packetSize_ 500
$CBR set interval_ 5ms

$CBR attach-agent $UDP

set Null [new Agent/Null]
$ns attach-agent $n(1) $Null

$ns connect $UDP $Null

$ns at 1 "$CBR start"
$ns at 4.5 "$CBR stop"



#Appeler la proc dure de terminaison apr s un temps t (ex t=5s)
$ns at 5.0 "finish"

#Ex cuter la simulation
$ns run
```

6.1.2 Script Exercice 2

```
#Cr ation d'une instance de l'objet Simulator
set ns [new Simulator]

#Ouvrir le fichier trace pour nam
set nf [open out.nam w]
$ns namtrace-all $nf

#Definir la proc dure de terminaison de la simulation
proc finish {} {
        global ns nf
        $ns flush-trace
        #fermer le fichier trace
        close $nf
        #Ex cuter le nam avec en entr e le fichier trace
        exec nam out.nam &
        exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]



$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

$ns queue-limit $n2 $n3 10

set udp [new Agent/UDP]
$ns attach-agent $n1 $udp

set CBR [new Application/Traffic/CBR]
$CBR set packetSize_ 1000
$CBR set interval_ 0.008

$CBR attach-agent $udp

set Null [new Agent/Null]
$ns attach-agent $n3 $Null

$ns connect $udp $Null

set tcp [new Agent/TCP]
```

```
$ns attach-agent $n0 $tcp

set ftp [new Application/FTP]
$ftp attach-agent $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink

$ns connect $tcp $sink

$ns color 1 Blue
$tcp set class_ 1

$ns color 2 Red
$udp set class_ 2

$ns at 0.1 "$CBR start"
$ns at 1 "$ftp start"
$ns at 4 "$ftp stop"
$ns at 4.5 "$CBR stop"


#Appeler la proc dure de terminaison apr s un temps t (ex t=5s)
$ns at 5.0 "finish"

#Ex cuter la simulation
$ns run
```

6.2 Scripts TP2

6.2.1 Script Exercice 3

```
#Cr ation d'une instance de l'objet Simulator
set ns [new Simulator]

#Ouvrir le fichier trace pour nam
set nf [open out.nam w]
$ns namtrace-all $nf

#Definir la proc dure de terminaison de la simulation
proc finish {} {
        global ns nf
        $ns flush-trace
        #fermer le fichier trace
        close $nf
        #Ex cuter le nam avec en entr e le fichier trace
        exec nam out.nam &
        exit 0
}

$ns rtproto DV
```

```
for {set i 1} {$i<=8} {set i [expr $i+1]} {set n($i) [$ns node]}

$ns duplex-link $n(1) $n(3) 10Mb 10ms DropTail
$ns duplex-link $n(2) $n(3) 10Mb 10ms DropTail
$ns duplex-link $n(3) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(3) $n(5) 10Mb 10ms DropTail
$ns duplex-link $n(5) $n(8) 10Mb 10ms DropTail
$ns duplex-link $n(4) $n(6) 10Mb 10ms DropTail
$ns duplex-link $n(6) $n(7) 10Mb 10ms DropTail
$ns duplex-link $n(7) $n(8) 10Mb 10ms DropTail


$ns duplex-link-op $n(1) $n(3) orient rigth-down
$ns duplex-link-op $n(2) $n(3) orient left-down
$ns duplex-link-op $n(3) $n(4) orient rigth-down
$ns duplex-link-op $n(3) $n(5) orient left-down
$ns duplex-link-op $n(4) $n(6) orient rigth
$ns duplex-link-op $n(6) $n(7) orient left-down
$ns duplex-link-op $n(7) $n(8) orient left
$ns duplex-link-op $n(5) $n(8) orient down




set udp1 [new Agent/UDP]
$udp1 set packetSize_ 500
$udp1 set interval_ 5ms
$ns attach-agent $n(1) $udp1

set udp2 [new Agent/UDP]
$udp2 set packetSize_ 500
$udp2 set interval_ 5ms
$ns attach-agent $n(2) $udp2

set Null [new Agent/Null]
$ns attach-agent $n(8) $Null

$ns connect $udp1 $Null
$ns connect $udp2 $Null


set CBR1 [new Application/Traffic/CBR]
set CBR2 [new Application/Traffic/CBR]

$CBR1 attach-agent $udp1
$CBR2 attach-agent $udp2


$ns color 1 Blue
$udp1 set class_ 1
$ns color 2 Red
$udp2 set class_ 2
```

```
$ns at 1 "$CBR1 start"
$ns at 2 "$CBR2 start"
$ns rtmodel-at 4 down $n(5) $n(8)
$ns rtmodel-at 5 up $n(5) $n(8)
$ns at 6 "$CBR2 stop"
$ns at 7 "$CBR1 stop"

#Appeler la proc dure de terminaison apr s un temps t (ex t=5s)
$ns at 8.0 "finish"

#Ex cuter la simulation
$ns run
```

6.2.2 Script Exercice 4

```
#Cr ation d'une instance de l'objet Simulator
set ns [new Simulator]

#Ouvrir le fichier trace pour nam
set nf [open out.nam w]
$ns namtrace-all $nf

set windowVsTime [open WindowVsTime w]
proc plotWindow {tcpSource windowVsTime} {
        global ns
        set time 0.1
        set cwnd [$tcpSource set cwnd_]
        set now [$ns now]
        if {$cwnd >= 20} {
                set cwnd 20
        }
        puts $windowVsTime "$now $cwnd"
        $ns at [expr $now+$time] "plotWindow $tcpSource $windowVsTime"
}


#Definir la proc dure de terminaison de la simulation
proc finish {} {
        global ns nf
        $ns flush-trace
        #fermer le fichier trace
        close $nf
        #Ex cuter le nam avec en entr e le fichier trace
        exec nam out.nam &
        exit 0
}

$ns rtproto DV
```

```
set n(1) [$ns node]
set n(2) [$ns node]
set n(3) [$ns node]
set n(4) [$ns node]
set n(5) [$ns node]
set n(6) [$ns node]
set n(7) [$ns node]

$n(1) label "A"
$n(2) label "B"
$n(3) label "C"
$n(4) label "D"

$n(1) shape box(hexagon)
$n(4) shape box(hexagon)


$ns duplex-link $n(1) $n(2) 10Mb 10ms DropTail
$ns duplex-link $n(1) $n(3) 10Mb 10ms DropTail
$ns duplex-link $n(2) $n(3) 10Mb 10ms DropTail
$ns duplex-link $n(2) $n(4) 10Mb 10ms DropTail
$ns duplex-link $n(3) $n(5) 10Mb 10ms DropTail
$ns duplex-link $n(5) $n(6) 10Mb 10ms DropTail
$ns duplex-link $n(6) $n(7) 10Mb 10ms DropTail
$ns duplex-link $n(7) $n(4) 10Mb 10ms DropTail


$ns duplex-link-op $n(1) $n(2) orient left-down
$ns duplex-link-op $n(1) $n(3) orient right-down




set tcp [new Agent/TCP]
$ns attach-agent $n(1) $tcp

set ftp [new Application/FTP]
$ftp attach-agent $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n(4) $sink

$ns connect $tcp $sink

$ns color 1 Blue
$ftp set class_ 1

$ns at 0 "plotWindow $tcp $windowVsTime"
$ns at 1 "$ftp start"
$ns rtmodel-at 3.48 down $n(2) $n(4)
$ns rtmodel-at 4.95 up $n(2) $n(4)
$ns at 7 "$ftp stop"
```

```
#Appeler la proc dure de terminaison apr s un temps t (ex t=5s)
$ns at 8.0 "finish"

#Ex cuter la simulation
$ns run
```

6.3 Script TP3

```
#Cr ation d'une instance de l'objet Simulator
set ns [new Simulator]

#Ouvrir le fichier trace pour nam
set nf [open out.nam w]
$ns namtrace-all $nf

#Cr ation de fichier pour xgraph
set f0 [open out0.tr w]
set f1 [open out1.tr w]
set f2 [open out2.tr w]

#Cr ation de noeuds
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

#Creation de lien entre les diff rents noeuds
$ns duplex-link $n0 $n3 1Mb 100ms DropTail
$ns duplex-link $n1 $n3 1Mb 100ms DropTail
$ns duplex-link $n2 $n3 1Mb 100ms DropTail
$ns duplex-link $n3 $n4 1Mb 100ms DropTail

#D finir la proc dure de terminaison de la simulation
proc finish {} {
        global f0 f1 f2
        global ns nf
        #Fermer le fichier trace
        close $nf
        #Ex cuter le nam avec en entr e le fichier trace
        exec nam out.nam
        #Close the output files
        close $f0
        close $f1
        close $f2
        #Executer xgraph
        exec xgraph out0.tr out1.tr out2.tr -geometry 800x400 & exit 0
}

#D finir la proc dure attach-expoo-traffic qui permet d'attacher l'application ex
proc attach-expoo-traffic { node sink size burst idle rate color number} {
```

```
        set ns [Simulator instance]

        #Cr ation d'agent UDP et d'une application Exponential
        set source [new Agent/UDP]
        $ns attach-agent $node $source

        set traffic [new Application/Traffic/Exponential]
        $traffic set packetSize_ $size
        $traffic set burst_time_ $burst
        $traffic set idle_time_ $idle
        $traffic set rate_ $rate

        $traffic attach-agent $source
        $ns connect $source $sink


        $ns color $number $color
        $source set class_ $number
        return $traffic
}

#D finir la proc dure record pour prendre des mesures pour tracer les graphes
proc record {} {
        global sink0 sink1 sink2 f0 f1 f2
        set ns [Simulator instance]
        #Definition de l'intervalle
        set time 0.5
        #Recuperation de bytes_ des sink
        set bw0 [$sink0 set bytes_]
        set bw1 [$sink1 set bytes_]
        set bw2 [$sink2 set bytes_]
        set now [$ns now]
        #Calcul du d bit
        puts $f0 "$now [expr $bw0/$time*8/1000000]"
        puts $f1 "$now [expr $bw1/$time*8/1000000]"
        puts $f2 "$now [expr $bw2/$time*8/1000000]"
        #R initialisation de la taille de bytes_    0
        $sink0 set bytes_ 0
        $sink1 set bytes_ 0
        $sink2 set bytes_ 0
        $ns at [expr $now+$time] "record"
}

set sink0 [new Agent/LossMonitor]
set sink1 [new Agent/LossMonitor]
set sink2 [new Agent/LossMonitor]
$ns attach-agent $n4 $sink0
$ns attach-agent $n4 $sink1
$ns attach-agent $n4 $sink2

set source0 [attach-expoo-traffic $n0 $sink0 200 2s 1s 100k Blue 1]
set source1 [attach-expoo-traffic $n1 $sink1 200 2s 1s 200k Red 2]
```

```
set source2 [attach-expoo-traffic $n2 $sink2 200 2s 1s 300k Green 3]



$ns at 0.0 "record"
#Start the traffic sources
$ns at 10.0 "$source0 start"
$ns at 10.0 "$source1 start"
$ns at 10.0 "$source2 start"
$ns at 50.0 "$source0 stop"
$ns at 50.0 "$source1 stop"
$ns at 50.0 "$source2 stop"
#Appeler la proc dure de terminaison apres un temps de 60s
$ns at 60.0 "finish"

#Ex cuter la simulation
$ns run
```