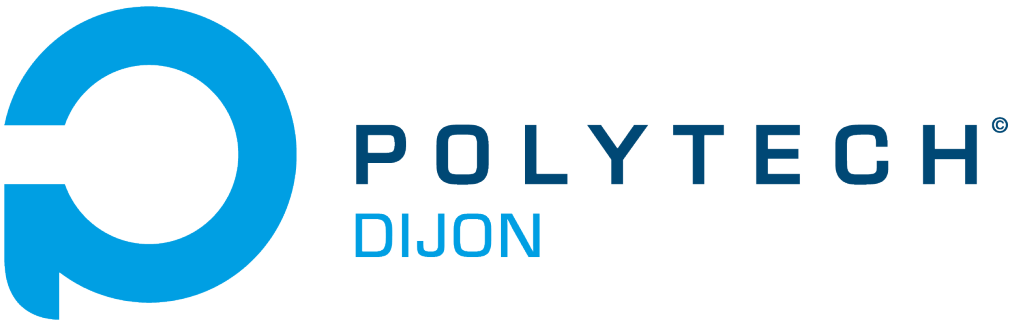
Polytech Dijon 3A options : Informatique & Réseaux

Module : ITC312

Compte rendu TP Simulation NS-2

Auteurs :
HUBERT Matéo, AGONGLO Marel

Professeur référent :
MBAREK Nader



2023-2024

Table des matières

Table des figures

Table des codes

1 Introduction générale

1.1 Contexte

Le but de ces travaux pratiques de simulation est de vérifier la bonne compréhension des notions vues en cours. Pour cela, on va utiliser le simulateur NS-2. Ce logiciel est un outil de simulation de réseaux informatiques développé à Lawrence Berkeley National Laboratory. L'objectif de ce simulateur est de faciliter l'étude de l'interaction entre différents protocoles mais aussi d'évaluer les performances de systèmes complexes. Ainsi, il est possible de faire des tests par ordinateurs avant de modifier ou créer son installation afin de gagner du temps et de ne pas bloquer l'accès aux réseaux des autres utilisateurs sur une periode trop importante. NS-2 est particulièrement bien adapté pour les réseaux à commutation de paquets. Il permet la mise en place de nombreux composants par catégorie comme :

— Application : FTP, telnet, CBR...
— Transport : TCP, UDP, RTP...
— Routage : Statique, Dynamique (DV, LS)...
— File d'attente : RED, DropTail...

1.2 Problématique

On découvrira le simulateur à travers l'élaboration de scripts qui feront appel à des outils comme nam ou encore xgraph. Pour cela on va notamment utiliser les scripts OTCL. On mettra également en place des protocoles de routage de type Vecteur Distance et Etat de Liens. Pour finir, on visualisera les performances d'un réseau avec xgraph pour mettre en évidence les phases de controle de flux, controle de congestion, Slow Start, Congestion Avoidance etc...

1.3 Annonce du plan

Dans la suite de ce compte rendu, chaque chapitre sera consacré à un TP. Pour chaque TP, on proposera une introduction et une conclusion. Pour chaque question du TP, on commentera uniquement les parties du code qui ne l'ont pas encore été pour éviter d'avoir un rapport global d'une longueur trop importante. L'ensemble des codes seront disponibles en annexe **6**.

2 Rapport Travaux Pratiques 1

2.1 Introduction

Le but du TP est de se familiariser avec le simulateur de réseaux NS-2. Pour cela, un script général permettant la simulation est fourni. L'objectif est alors de créer une communication entre deux noeuds dans une topologie simple puisque ces deux noeuds sont reliés directement. La communication entre ces deux noeuds se fera grâce à un agent UDP attaché au noeud n0. Une fois le script réalisé, on visualisera la simulation avec nam pour vérifier que les paramètres de communication spécifiés dans le script sont bien effectifs. Par la suite, on essayera de recréer la même simulation mais en utilisant directement l'interface graphique de nam. Dans la suite du TP, on réalisera une simulation de réseau plus complexe notamment en spécifiant une taille de buffer, en positionnant les noeuds selon une topologie spécifique et mettant en place des agents TCP et UDP etc... La finalité du TP est de réussir à décrire les résultats obtenus lors de la simulation en indiquant les phases et les causes de la congestion, pertes de paquets etc...

2.2 Analyse de la forme générale d'un script de simulation

```
1  set ns [new Simulator]
2
3  set nf [open out.nam w]
4  $ns namtrace-all $nf
5
6  proc finish {} {
7      global ns nf
8      $ns flush-trace
9      close $nf
10     exec nam out.nam &
11     exit 0
12 }
13
14 $ns at 5.0 "finish"
15
16 $ns run
```

Code 1 – Script général

La commande : set ns [new Simulator] permet de créer une instance de l'objet Simulator attribué à la variable ns.

La commande : set nf [open out.nam w] permet d'ouvrir le fichier out.nam en mode écriture pour récupérer les données de simulation pour nam. Par la suite, la variable nf pourra être utilisée pour faire référence à ce fichier.

La commande : $ns namtrace-all $nf permet d'enregistrer tous les événements de la simulation obtenus en passant par le simulateur via la variable ns et de les écrirent dans le fichier de sortie pointé par la variable nf.

La commande : proc finish {} {} permet la déclaration sans paramètres d'une fonction appelée finish.

La commande : global ns nf permet de spécifier la portée globale des variables ns et nf, c'est à dire modifier la portée d'accessibilité de ces variables afin de pourvoir les utiliser dans la fonction.

La commande : $ns flush-trace permet d'écrire toutes les données acquises dans le fichier correspondant ici à out.nam.

La commande : close $nf permet de fermer le fichier de sortie out.nam.

La commande : exec nam out.nam & permet d'exécuter la commande nam out.nam qui ouvre la simulation nam avec les informations récupérées dans le fichier de sortie out.nam.

La commande : exit 0 permet de mettre fin à l'éxécution du script.

La commande : $ns at 5.0 "finish" permet d'appeler la fonction finish après 5 secondes de simulation.

la commande : $ns run permet d'éxécuter la simulation.

2.3 Exercice : 1

2.3.1 But de l'exercice

Le but de cet exerice est de créer une communication entre un Agent UDP et un Agent Null avec un trafic CBR. Les deux noeuds sont reliés directement. Il faut respecter certains paramètres donnés dans l'énoncé pour le trafic CBR comme la taille des paquets ou encore l'intervalle de transmission de paquets. Dans la seconde partie de l'exercice, on verra les différences entre les implémentations au niveau du code et l'implémentations graphique en utilisant nam.

2.3.2 Explications du code

```
19 #Création de deux noeuds nommé n(0) et n(1) sous la forme d'un tableau
20 set n(0) [$ns node]
21 set n(1) [$ns node]
```

Code 2 – Création de deux noeuds n(0) et n(1)

Les deux commandes ci-dessus permettent de créer les deux noeuds nécessaires à l'exercice. On utilise set car on veut créer le noeud, n(0) car les noeuds sont créés sous la forme d'un tableau et le [$ns node] permet d'appeler la fonction node du Simulator pointée par la variable ns afin de créer un noeud.

```
23 #Création d'un lien duplex entre n(0) et n(1) de capacité 1Mb,
24 #10ms de temps de propagation et d'une file d'attente de type
25 #DropTail
26 $ns duplex-link $n(0) $n(1) 1Mb 10ms DropTail
```

Code 3 – Connexion de deux noeuds avec un lien duplex

Grâce à la commande ci-dessus, on peut connecter deux noeuds ensemble. On utilise $ns pour accéder à l'instance de la simulation, en spécifiant duplex-link, on peut indiquer le nom des deux noeuds à relier ici, n(0) et n(1) sans oublier le $ car ce n'est pas une déclaration. Par la suite, on précise comme demandé dans l'énoncé, que la Bandwith est de 1Mb/s, que le Delay est de 10ms et que le TypeOfBuffer est de type DropTail. La Bandwith représente le débit, le Delay représente le temps de propagation et le TypeOfBuffer représente le comportement du buffer des noeuds. Ici DropTail nous indique que quand le buffer est plein alors tous les nouveaux paquets seront dropés jusqu'à ce que de la place se libère.

```
28 #Création d'un agent UDP pour le noeud n(0)
29 set udp [new Agent/UDP]
30 $ns attach-agent $n(0) $udp
```

Code 4 – Création d'un agent UDP attacher à n(0)

La commande set udp permet de créer une variable nommée udp qui va être un agent UDP. Comme UDP hérite de la classe Agent, il faut alors spécifier la hiérarchie complète, il est donc indispensable d'indiquer Agent/UDP. Par la suite on doit attacher ce nouvel Agent au noeud émetteur n(0). On utilise donc la méthode du simulateur attach-agent en spécifiant dans un premier temps le noeud auquel on va attacher l'Agent puis l'Agent que l'on veut lui attacher.

```
32 #Création d'une source de trafic CBR avec une taille des paquets
33 #de 500 octets et d'intervalle de transmission de paquets de 0.005s
34 set cbr [new Application/Traffic/CBR]
35 $cbr set packetSize_ 500
36 $cbr set interval_ 0.005
```

Code 5 – Créer une source de trafic CBR

Cette fois ci, on doit créer une source de trafic CBR c'est donc une Application de type Traffic la hiérarchie est donc Application/Traffic/CBR. Cette Application sera pointée par la variable cbr. Il faut ensuite modifier quelques paramètres de la source CBR. Avec la commande $cbr set packetSize_ 500 il est possible de spécifier la taille des paquets pour l'application CBR créée précédement via la variable cbr. L'unité par défaut de la taille des paquets pour une source CBR sur NS-2 est l'octet (bytes) ainsi chaque paquet émit aura une taille de 500 octets. La commande $cbr set intervalle_ 0.005 permet d'indiquer à la source que les paquets sont envoyés toutes les 0.005 secondes.

```
38 #Connexion de la source CBR à l'agent UDP
39 $cbr attach-agent $udp
```

Code 6 – Connexion de la source CBR à l'agent UDP

La commande ci-dessus permet d'indiquer à la source CBR qu'elle doit utiliser l'Agent UDP pour envoyer ses paquets.

```
41 #Création de l'agent NULL pour la réception des paquets sur le noeud n(1)
42 set null [new Agent/Null]
43 $ns attach-agent $n(1) $null
```

Code 7 – Création de l'Agent Null pour la réception des paquets

On crée l'agent Null et on l'attache de la même manière que pour l'Agent UDP mais en attachant l'Agent au noeud n(1).

```
45 #Connexion des deux agents Null et UDP
46 $ns connect $udp $null
```

Code 8 – Connexion des deux Agents UDP et Null

Pour connecter deux Agents on utilise la commande connect du simulateur en spécifiant les deux Agents à connecter, ici UDP avec $udp et Null avec $null.

```
59 #Déclenchement du trafic CBR à t=1s puis arrêt à t=4.5s
60 $ns at 1 "$cbr start"
61 $ns at 4.5 "$cbr stop"
```

Code 9 – Déclenchement du trafic CBR à 1s et arrêt à 4.5s

Pour déclencher le trafic on utilise $ns pour spécifier l'instance du simulateur auquel on souhaite déclencher un événement, avec at, on indique à quel temps se produit l'événement et avec "événement" on indique les actions à réaliser. On peut donc avec "$cbr start" lancer le trafic CBR et avec "$cbr stop" le stopper. Une colorisation des trafics ainsi que la mise en forme de la simulation ont été réalisé mais ont été traité dans le TP2. Vous pourrez retrouver l'ensemble du code dans l'Annexe 6.

2.3.3 Visualisation sur nam

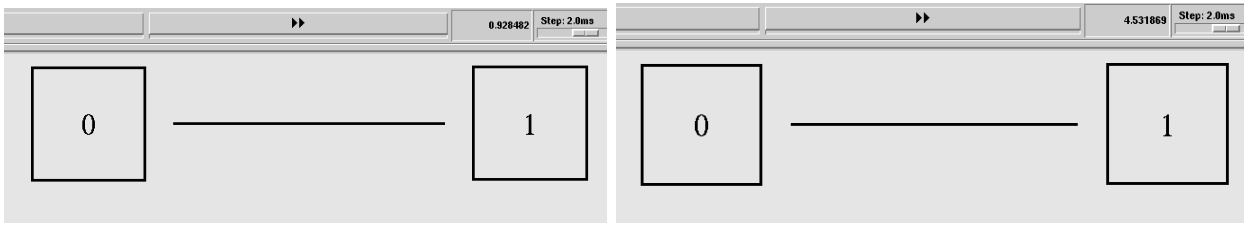


FIGURE 1 – Absence de communication avec 1s et après 4.5s

Comme on peut le voir sur la figure ci-dessus, il n'y a pas de communication avant 1 seconde et après 4.5 secondes ce qui correspond bien à l'énoncé.

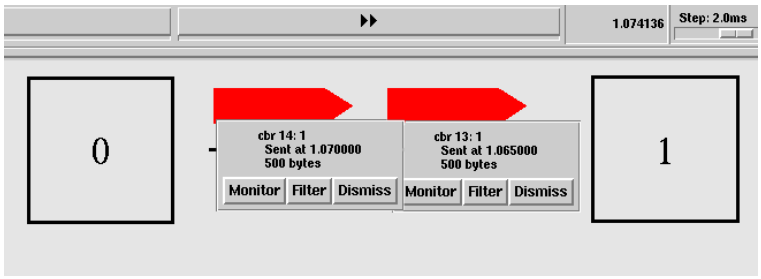


FIGURE 2 – Trafic CBR via UDP vers Null

On voit bien que les paquets font 500 octects. De plus, l'intervalle de transmission de paquets est bien de 0.005s.

2.3.4 Découverte de l'interface et des fonctionnalités

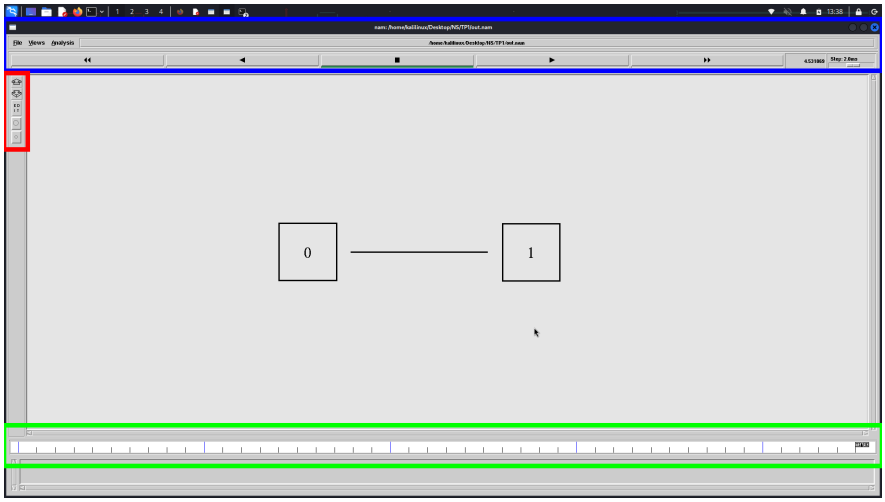


FIGURE 3 – Interface nam

Dans le rectangle bleu, on peut voir qu'on nous indique le chemin absolu jusqu'au fichier de simulation. On voit également que l'on peut lancer la simulation, l'arrêter, la faire aller dans le sens inverse ou encore faire des avances rapides. On peut également régler l'incrémentation du temps avec la jauge en haut à droite pour avancer très rapidement jusqu'à un moment cible ou au contraire ralentir très fortement pour capturer des paquets de routage.

Dans le rectangle rouge, on peut observer 2 flèches qui permettent de zoomer ou de dézoomer. Il y a également un bouton EDIT/VIEW qui permet, soit de modifier l'emplacement des noeuds par exemple, soit d'avoir des informations en cliquant sur un noeud ou sur un paquet.

Dans le rectangle vert, on peut observer une barre temporelle qui permet de déplacer le curseur pour aller dans un endroit précis de la simulation sans attendre que le temps s'écoule, ce qui peut être pratique notamment lors de longue simulation.

2.3.5 Utilisation de l'outil graphique

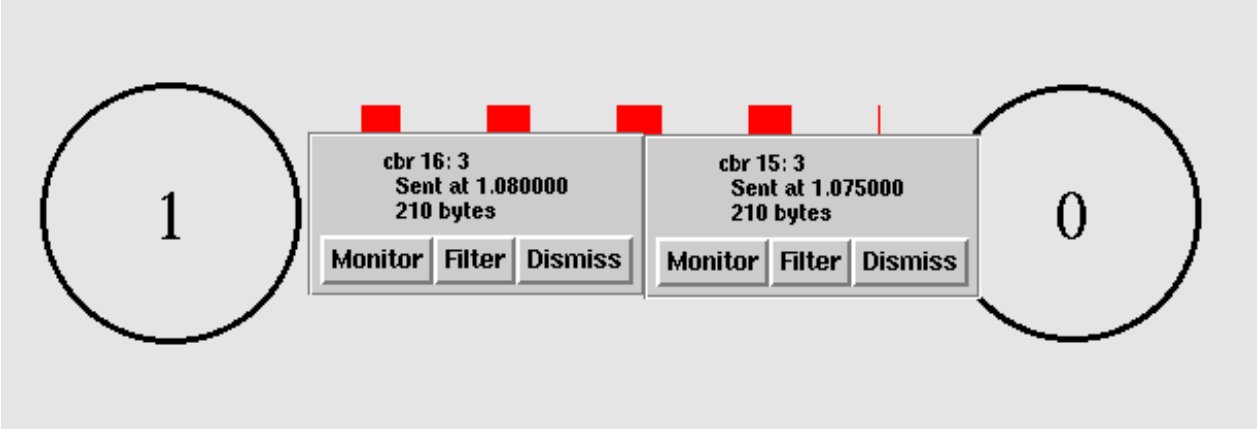


FIGURE 4 – Utilisation de l'outil graphique

Comme on peut le voir, même en ayant spécifié une taille de paquets de 500 octets, la taille des paquets n'est que de 210 octets. L'intervalle a été pris en compte tout comme la coloration des paquets. Il n'est par contre pas possible de spécifier l'ensemble des paramètres comme dans un script. L'interface graphique est donc grandement limitée et ne sera donc plus utilisée par la suite en raison de ses limitations.

2.4 Exercice : 2

2.4.1 But de l'exercice

Le but de cet exercice est de simuler un réseau avec quatre noeuds. Deux d'entre eux sont émetteurs. n(0) possède un Agent TCP avec une Application FTP. n(1) possède un Agent UDP avec une Application CBR. Ces deux noeuds communiquent avec le noeud n(2) qui redirige l'ensemble du trafic vers n(3) récepteur qui possède un Agent TCPSink et un Agent Null. De plus on verra comment colorer les différents trafics et comment orienter les différents noeuds pour créer la topologie voulue.

2.4.2 Explications du code

```
1  #Creation 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  #Definir la procedure de terminaison de la simulation
9  proc finish {} {
10     global ns nf
11     $ns flush-trace
12     #Fermer le fichier trace
13     close $nf
14     #Executer le nam avec en entrée le fichier trace
15     exec nam out.nam &
16     exit 0
17 }
18 #Création des noeuds
19 set n0 [$ns node]
20 set n1 [$ns node]
21 set n2 [$ns node]
22 set n3 [$ns node]
```

Code 10 – Structure générale et création des noeuds

La structure générale du code est la même que celle de l'exercice 1 et a dejà été expliquée dans la partie **2.2**. Par la suite, on crée simplement 4 noeuds.

```
24 #Connexion des différents noeuds
25 $ns duplex-link $n0 $n2 2Mb 10ms DropTail
26 $ns duplex-link $n1 $n2 2Mb 10ms DropTail
27 $ns duplex-link $n2 $n3 1.7Mb 20ms DropTail
```

Code 11 – Création des différents liens

Les liens entre n0 et n2 ainsi qu'entre n1 et n2 ont un débit de 2Mb/s, un temps de propagation de 10ms et un buffer de type DropTail. Le détail des options est disponible dans la partie **2.3.2**.

```
29 #Fixe la taille du buffer entre le noeud n2 et n3
30 $ns queue-limit $n2 $n3 10
```

Code 12 – Réglage du buffer

Grâce à cette ligne de code, on peut fixer la taille du buffer à l'entrée du lien entre n2 et n3 à 10, 10 étant un nombre de paquets. Cela signifie que le noeuds n2 peut avoir en mémoire jusqu'à 10 paquets. Par la suite, ils seront dropés.

```
32 #Modification de l'emplacement des noeuds
33 $ns duplex-link-op $n0 $n2 orient right-down
34 $ns duplex-link-op $n1 $n2 orient right-up
35 $ns duplex-link-op $n2 $n3 orient right
```

Code 13 – Orientation des noeuds

Avec la commande duplex-link-op on peut orienter les noeuds les uns par rapport aux autres. Par exemple, la ligne 33 permet de positionner le noeud n2 en bas à droite de n0. En faisant cette opération pour tous les noeuds, on peut ainsi obtenir la topologie voulue comme le montre l'image ci dessous.

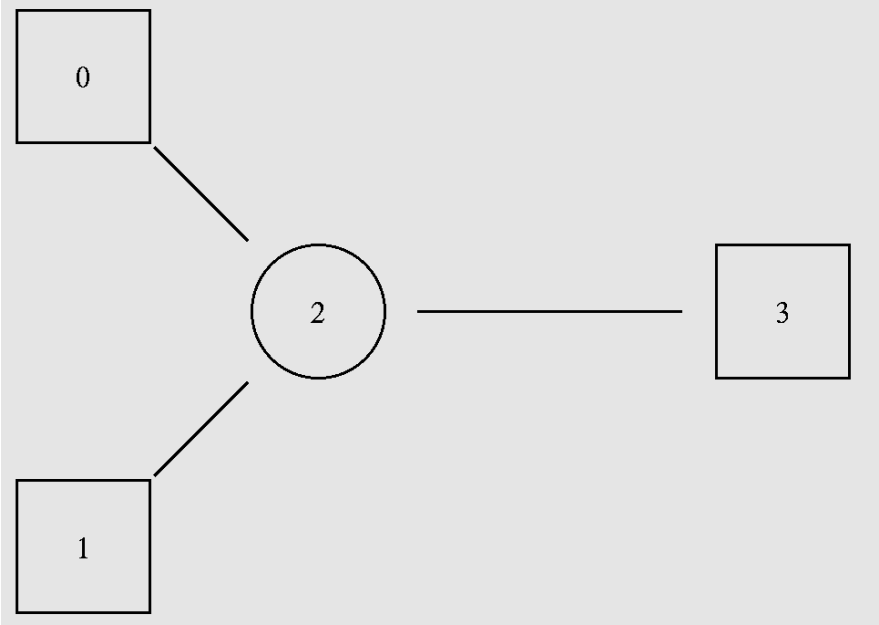


FIGURE 5 – Topologie de l'exercice 2

```
42 #Création d'un Agent TCP
43 set tcp [new Agent/TCP]
44 $ns attach-agent $n0 $tcp
45
46 #Création d'un Agent UDP
47 set udp [new Agent/UDP]
48 $ns attach-agent $n1 $udp
49
50 #Création d'un Agent TCPSink pour l'envoi d'ACK
51 set tcpsink [new Agent/TCPSink]
52 $ns attach-agent $n3 $tcpsink
53
54 #Création de l'Agent Null pour l'Agent UDP
55 set null [new Agent/Null]
56 $ns attach-agent $n3 $null
```

Code 14 – Création des agents

Toute cette section de code permet de créer les différents Agents nécessaires. Le noeud n0 possède un Agent TCP. Le noeud n1 possède un Agent UDP. Le noeud n3 quand à lui possède un Agent TCPSink ainsi qu'un Agent Null. Le détail de la création d'Agent est détaillé dans la partie **2.3.2**.

```
58 #Création d'une Application FTP
59 set ftp [new Application/FTP]
60 $ftp attach-agent $tcp
61
62 #Création d'une source de trafic CBR et modification des paramètres
63 set cbr [new Application/Traffic/CBR]
64 $cbr set packetSize_ 1000
65 $cbr set interval_ 0.008
66 $cbr attach-agent $udp
```

Code 15 – Création des Applications

Cette partie du code permet la création des deux Applications nécessaires. Une Application FTP pour le noeud n0 qui communiquera avec n3 via TCP. Une Application CBR qui communiquera avec le noeud n3 via UDP. Pour la source de trafic CBR il est précisé que la taille des paquets

doit être de 1Ko et que le débit d'envoi doit être de 1Mb/s. Le détail de la création d'Application est disponible dans la partie **2.3.2**. Ainsi avec un calcul simple on peut en déduire que l'intervalle d'envoi des paquets est :

$1\ octet = 8\ bits \rightarrow 1\ 000\ octets = 8\ 000\ bits$

$1\ Mbits = 1\ 000\ 000\ bits \rightarrow intervalle = \frac{8\ 000}{1\ 000\ 000} = 0.008\ seconde$

Donc si chaque paquet est envoyé toutes les 0.008 secondes alors le débit d'envoi sera bien de 1Mb/s.

```
68 #Connexion des différents Agents
69 $ns connect $tcp $tcpsink
70 $ns connect $udp $null
```

Code 16 – Connexion des Agents

On connecte les différents Agents source à leurs Agents de réception.

```
72 #Coloration des différents trafics
73 $ns color 1 Blue
74 $tcp set class_ 1
75 $ns color 2 Red
76 $udp set class_ 2
```

Code 17 – Coloration des différents trafics

A la ligne 73 on peut voir que l'on configure la couleur de la classe numéro 1 en bleu. A la ligne suivante, on indique que l'instance tcp appartient à la classe 1 ce qui permettra par la suite de visualiser les paquets de communication de l'instance tcp en bleu lors de la simulation. On fait de même avec la classe 2 que l'on spécifie de couleur rouge et que l'on attribut à l'instance udp.

```
78 #Création des événements de la simulation
79 $ns at 0.1 "$cbr start"
80 $ns at 1 "$ftp start"
81 $ns at 4 "$ftp stop"
82 $ns at 4.5 "$cbr stop"
83
84 #Appeler la procedure de terminaison apres un temps t
85 $ns at 5.0 "finish"
86
87 #Executer la simulation
88 $ns run
```

Code 18 – Évènements de simulation

Cette partie du code regroupe tous les événements de la simulation. On voit bien que le trafic CBR commence à 0.1 seconde et fini à 4.5 secondes. Le trafic FTP quand à lui commence à 1 seconde et fini à 4 secondes. Vous trouverez le détail de création d'événements dans la partie **2.3.2**. Vous trouverez également l'entièreté du code en Annexe **6**.

Il ne reste plus qu'à préciser que la simulation s'arrête après 5 secondes et l'on peut ensuite passer à la visualisation sur nam.

2.4.3 Visualisation sur nam

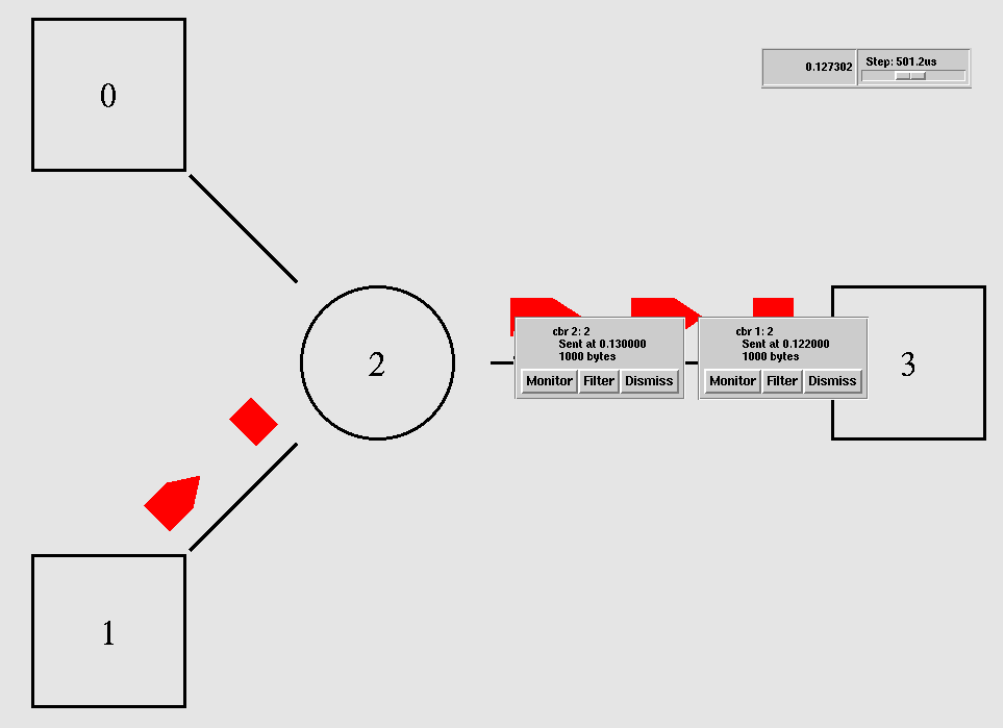


FIGURE 6 – Démarrage communication UDP

Comme on peut le voir sur la figure ci-dessus, la communication UDP commence bien à 0.1 seconde. De plus si l'on regarde l'intervalle de temps entre deux paquets, il correspond bien à 0.008 seconde. Chaque paquet fais bien 1000 octets donc les paramètres pour la communication UDP sont bien respectés.

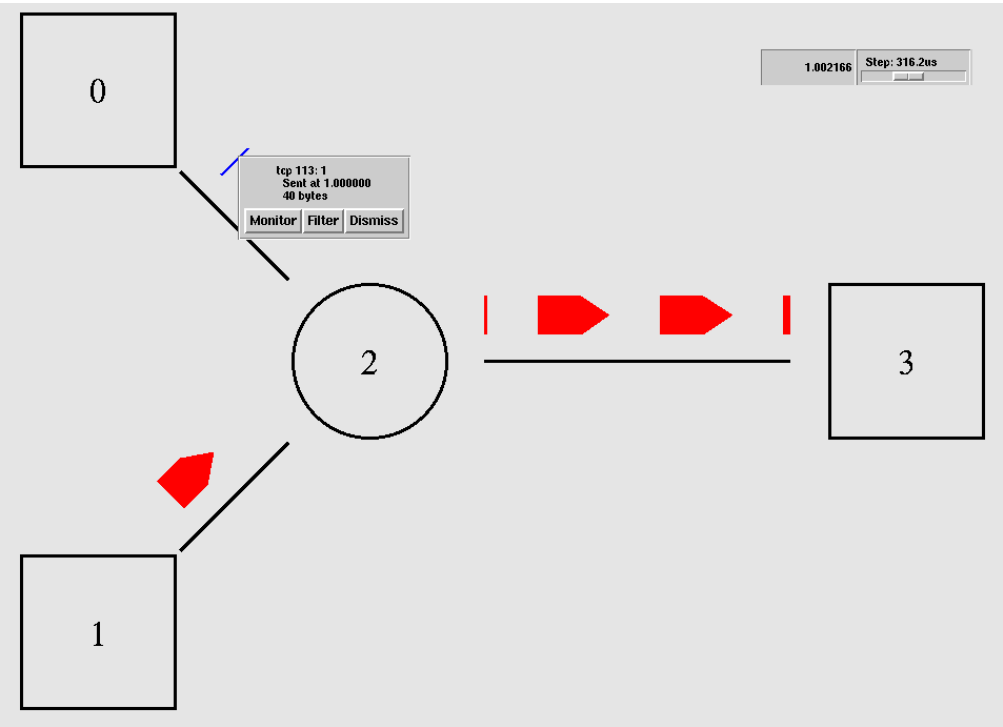


FIGURE 7 – Démarrage de la communication TCP

On voit sur la figure ci-dessus que la communication TCP commence à 1 seconde. De plus le premier paquet envoyé fait 40 octets car c'est une demande de connexion c'est à dire qu'il n'y a aucun octet de données, uniquement les en-têtes dont le flag SYN pour établir la connexion.

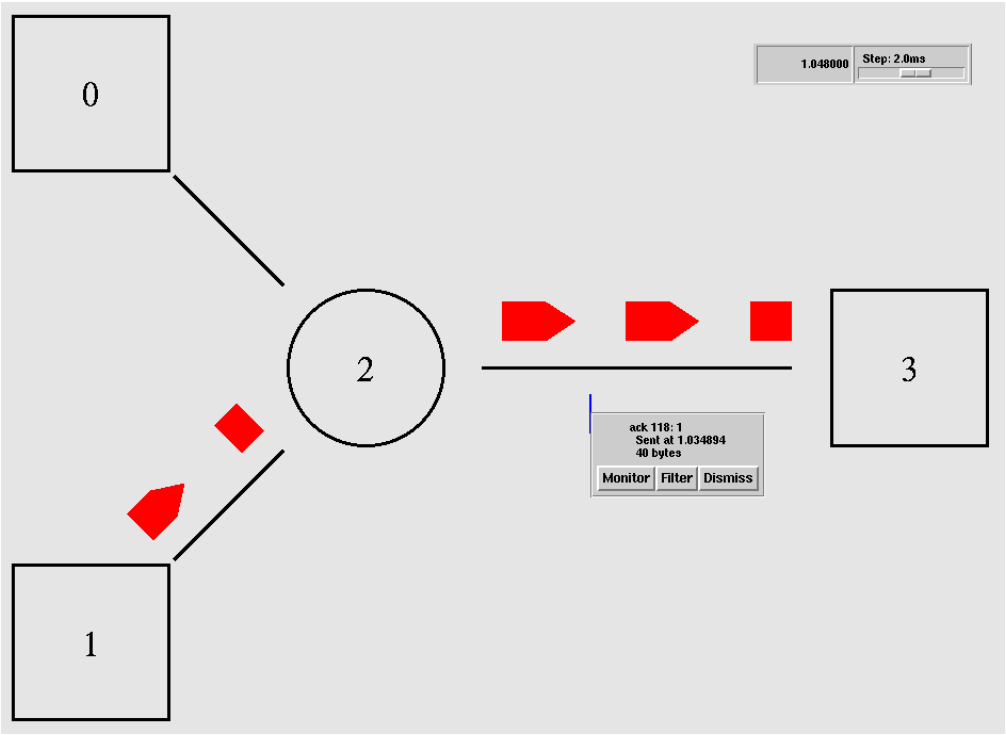


FIGURE 8 – ACK du premier segment TCP

Sur l'image ci-dessus, on visualise clairement la réponse au premier segment TCP qui fait également 40 octets car c'est un ACK. A la réception de cet ACK, le noeud n0 va doubler la valeur de son cwnd ce qui va doubler la valeur de la fenêtre TCP car il à reçu un seul ACK ce qui signifie que le rwnd n'est pas encore atteint. Il ne sature donc pas le réseau et peut se permettre d'envoyer plus rapidement ses paquets. C'est la phase dite de Slow Start. Tant que des drops ou des ACK dupliqués ne sont pas détectés, la communication est dans une phase de contrôle de congestion, c'est à dire que l'émetteur s'adapte aux capacités du réseau.

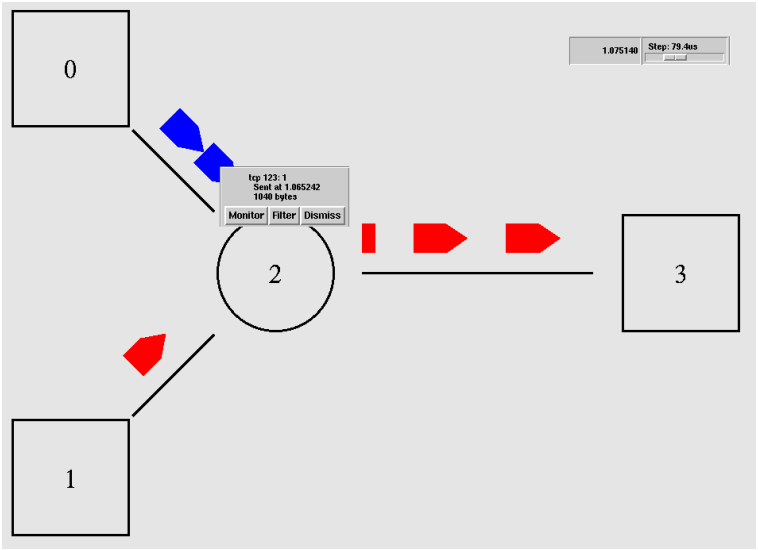


FIGURE 9 – Deuxième RTT

Comme décrit précédemment, il y a bien 2 paquets qui sont envoyés. On rappelle que l'on a paramétré un buffer pouvant contenir 10 paquets. La question que l'on se pose alors est : va t'il y avoir des pertes et si oui quand ? Les deux liens reliant n0 à n2 et n0 à n3 ont une capacité de 2Mb/s tandis que le lien reliant n2 et n3 a une capacité de 1.7Mb/s. Étant donné que la communication UDP se fait à 1Mb/s, quand la communication TCP atteindra plus 0.7Mb/s, il y aura des drops.

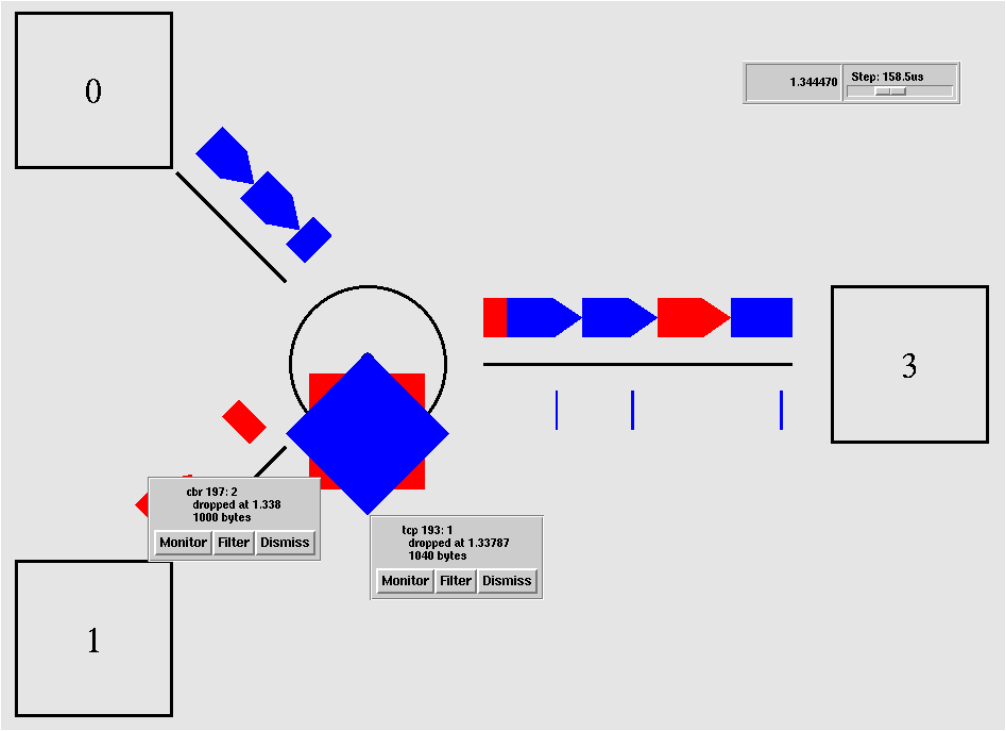


FIGURE 10 – Premier Drop

On voit sur l'image ci-dessus que des drops se produisent à la fois pour UDP et pour TCP. Suite à ces drops, UDP ne se rendra compte de rien et va continuer d'envoyer à une vitesse constante de 1Mb/s sans renvoyer les paquets perdus car UDP est un mode de transport pour le temps réel et n'est pas en mode connecté. TCP quand à lui va se rendre compte de la perte de paquets et les renvoyer car il ne va pas recevoir d'ACK avant la fin du temporisateur. On est donc dans la situation d'épuisement d'un temporisateur de retransmission c'est à dire RTO. Dans ce cas, la valeur du cwnd va revenir à un SMSS (Sender Maximum Segment Size). Par la suite, le ssthresh (Slow Start Threshold) qui détermine jusqu'à quelle valeur de cwnd sera effectué le Slow Start pour la ré-émission, va devenir la dernière valeur de fenêtre TCP n'ayant pas provoqué de perte divisée par 2.

Par la suite, on peut observer que lorsque cwnd est supérieur à sshthresh, les incrémentations de cwnd ne sont plus doublées, mais un ajout de un SMSS est réalisé à chaque RTT. Cela permet de saturer moins vite le réseau. Ce phénomène s'appelle Congestion Avoidance. Cela reste du contrôle de congestion et on reste dans le Congestion Avoidance jusqu'à la prochaine détection de congestion.

2.5 Conclusion

Dans ce TP on a pu découvrir les bases de la programmation OTCL ainsi que le fonctionnement de nam. On s'est également rendu compte que l'utilisation de nam via interface graphique est grandement limitée c'est pourquoi nous utiliserons des codes par la suite. On à également pu voir en application les différents phénomènes vus en cours des modes de transports UDP et TCP et pourquoi il vaut mieux utiliser UDP pour le temps réel et TCP pour la fiabilité.

3 Rapport Travaux Pratiques 2

3.1 Introduction

Le but du TP est de mettre en place des protocoles de routage à Vecteur de Distance (DV) et Etat des Liens (LS). Plus particulièrement on s'intéressera aux paquets de routage échangés lors du début de la simulation, lors de la rupture d'un lien, du rétablissement d'un lien et éventuellement les mises à jour périodiques. Ainsi, nous pourrons comparer le temps de mise à jour entre ces deux protocoles de routage. Par la suite on visualisera les performances d'un réseau grâce à xgraph et l'on détaillera les différentes phases telles que le Slow Start, Congestion Avoidance, cwnd, rwnd, ssthresh etc... De plus, on procédera à une comparaison entre la perte des paquets dans ce TP et dans le TP précédent.

3.2 Exercice : 1

3.2.1 But de l'exercice

Le but de l'exercice 1 est de créer un réseau contenant 8 noeuds. Les noeuds numéros 1 et 2 seront émetteurs. Ils auront tous deux un protocole de transport UDP avec une taille de paquets de 500 octets et un intervalle de transmission de 0.005 secondes. Tous les liens reliant les noeuds seront de types full-duplex avec un débit de 10Mb/s, un delai de 10ms et un buffer de type DropTail. L'objectif est de capturer les moments d'échanges de paquets de routage afin de faire un tableau comparatif du temps de mise à jour du chemin des paquets. Grâce à cela, on pourra déduire les caractéristiques des deux protocoles de routage comme le temps de convergence, le comportement au niveau du noeud qui subit la rupture etc...

3.2.2 Explications du code

```
#Creation 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 procedure de terminaison de la simulation
proc finish {} {
    global ns nf
    $ns flush-trace
    #Fermer le fichier trace
    close $nf
    #Executer le nam avec en entrée le fichier trace
    exec nam out.nam &
    exit 0
}
```

Code 19 – Structure générale

Ce code est la structure générale d'un script OTCL dont vous trouverez l'explication détaillée dans la partie **2.2**

```
18 #Déclaration du protocole de routage
19 $ns rtproto DV
```

Code 20 – Mise en place du protocole de routage à vecteur de distance

Grâce à la commande rtproto il est possible de mettre en place un protocole de routage ici DV pour Distance Vector ou Vecteur de Distance en français. Ce protocole de routage permet aux routeurs de construire des tables de routages ou aucun routeur ne possède la vision globale du réseau. La diffusion des informations se fait de proche en proche. Ici la distance est le nombre de sauts.

```
20 #Création des noeuds grâce à une boucle
21 for {set i 1} {$i<=8} {set i [expr $i+1]} {set n($i) [$ns node]}
```

Code 21 – Création des noeuds via une boucle

Cette boucle for permet de créer autant de noeuds que l'on veut en une seule ligne. C'est très pratique quand le nombre de noeuds est important comme dans cet exercice. {set i 1} initialise le compteur de la boucle à 1. $\{i <= 8\}$ indique que tant que la valeur de la variable i est inférieure ou égale à 8, alors les instructions de la boucle s'effectuent. {set i [expr $i+1$]} permet d'incrémenter le compteur à chaque itération de la boucle. Pour réaliser des opérations il est indispensable d'utiliser expr. {set n($i) [$ns node]} représente les commandes à exécuter à chaque itération de la boucle, ici créer un noeud dont le nom change à chaque fois que l'on fait une itération.

```
22 #Création des différents Agents
23 set udp1 [new Agent/UDP]
24 $ns attach-agent $n(1) $udp1
25 set udp2 [new Agent/UDP]
26 $ns attach-agent $n(2) $udp2
27 set null [new Agent/Null]
28 $ns attach-agent $n(8) $null
```

Code 22 – Création des différents Agents

Toute cette section de code permet de créer les différents Agents nécessaires. Les noeuds n(1) et n(2) possèdent un Agent UDP. Le noeud n(8) possède quant à lui un Agent Null pour la réception. Le détail de la création d'Agent est détaillé dans la partie **2.3.2**.

```
29 #Création des différents liens
30 $ns duplex-link $n(1) $n(3) 10Mb 10ms DropTail
31 $ns duplex-link $n(2) $n(3) 10Mb 10ms DropTail
32 $ns duplex-link $n(3) $n(4) 10Mb 10ms DropTail
33 $ns duplex-link $n(4) $n(6) 10Mb 10ms DropTail
34 $ns duplex-link $n(3) $n(5) 10Mb 10ms DropTail
35 $ns duplex-link $n(5) $n(8) 10Mb 10ms DropTail
36 $ns duplex-link $n(6) $n(7) 10Mb 10ms DropTail
37 $ns duplex-link $n(7) $n(8) 10Mb 10ms DropTail
```

Code 23 – Création des différents liens

Toute cette partie de code permet de créer les différents liens reliants les noeuds entre eux. Tous les liens ont les mêmes paramètres c'est à dire un débit de 10Mb/s, un delai de 10ms et un buffer de type DropTail. Vous trouverez le détail de création de lien dans la partie **2.3.2**

```
#Orientation des différents noeuds
$ns duplex-link-op $n(1) $n(3) orient right-down
$ns duplex-link-op $n(2) $n(3) orient left-down
$ns duplex-link-op $n(3) $n(4) orient right-down
$ns duplex-link-op $n(4) $n(6) orient right
$ns duplex-link-op $n(3) $n(5) orient left-down
$ns duplex-link-op $n(5) $n(8) orient down
$ns duplex-link-op $n(7) $n(8) orient left
$ns duplex-link-op $n(6) $n(7) orient left-down
```

Code 24 – Orientation des différents noeuds

Cette portion de code permet d'orienter les noeuds les uns par rapport aux autres afin d'obtenir la topologie démandée disponible sur l'image ci-dessous. Le détail de l'orientation des noeuds est expliqué dans la partie **2.4.2**.

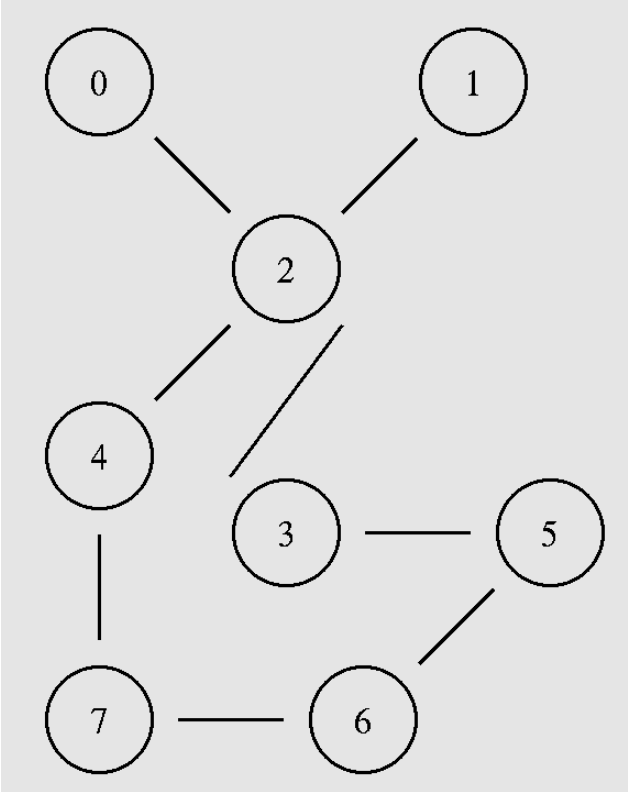


FIGURE 11 – Topologie de l'exercice 1

Comme on peut le voir, les noms des noeuds lors de la définition va de n(1) à n(8) mais lors de la simulation les noms vont de n(0) à n(7).

```
#Connexion des différents Agents
$ns connect $udp1 $null
$ns connect $udp2 $null
```

Code 25 – Connexion des différents Agents

Ces deux lignes de code permettent de connecter les deux sources UDP des noeuds n(1) et n(2) à l'Agent Null du noeud n(8).

```
#Création des différentes Applications
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 500
$cbr2 set interval_ 0.005
```

Code 26 – Création des différentes Applications

Cette partie permet la création des différentes Applications nécessaires ainsi que la modification de certains paramètres. Les deux sources CBR ont une taille de paquets de 500 octets et un intervalle de transmission de 0.005 seconde. Le détail de la création d'Application est disponible dans la partie **2.3.2**.

```
#Connexion des différentes Applications à leur Agent
$cbr1 attach-agent $udp1
$cbr2 attach-agent $udp2
```

Code 27 – Connexion des différentes Applications à leur Agent

Ces lignes de code permettent d'attacher l'Application CBR pointé par cbr1 à l'Agent UDP pointé par udp1. Il en est de même avec cbr2 et udp2.

```
#Coloration des différents trafics
$ns color 1 Blue
$udp1 set class_ 1
$ns color 2 Red
$udp2 set class_ 2
```

Code 28 – Coloration des trafics

Le trafic UDP du noeud n(1) sera de couleur bleu et celui du noeud n(2) sera de couleur rouge. Le détail de la coloration de paquet est disponible dans la partie **2.4.2**.

```
#Evénements de simulation
$ns at 1 "$cbr1 start"
$ns at 2 "$cbr2 start"
#Rupture du lien puis rétablissement
$ns rtmodel-at 4 down $n(5) $n(8)
$ns rtmodel-at 5 up $n(5) $n(8)
$ns at 6 "$cbr2 stop"
$ns at 7 "$cbr2 stop"
#Appeler la procedure de terminaison apres un temps t
$ns at 8.0 "finish"
#Executer la simulation
$ns run
```

Code 29 – Évènements de simulation

Comme indiqué dans l'énoncé, le trafic CBR du noeud n(1) doit commencer à 1 seconde et finir à 7 secondes. Le trafic CBR du noeud n(2) doit quant à lui commencer à 2 secondes et finir à 6 secondes. La partie la plus importante se situe au niveau des lignes 71 et 72. C'est ici que l'on crée la rupture puis le rétablissement. Avec rtmodel-at on peut indiquer à quel temps se produit l'événement puis down pour indiquer une rupture puis les deux noeuds concernés. Pour rétablir le lien, il suffit de modifier down en up. Vous trouverez le code dans l'Annexe **6**.

On peut donc passer à la visualisation sur nam.

3.2.3 Visualisation sur nam protocole de routage DV

L'objectif de cette partie et de repérer les différents moments ou des paquets de routage sont échangés afin de déterminer le temps de convergence des différents protocoles de routage ainsi que le fonctionnement du noeud au niveau de la rupture.

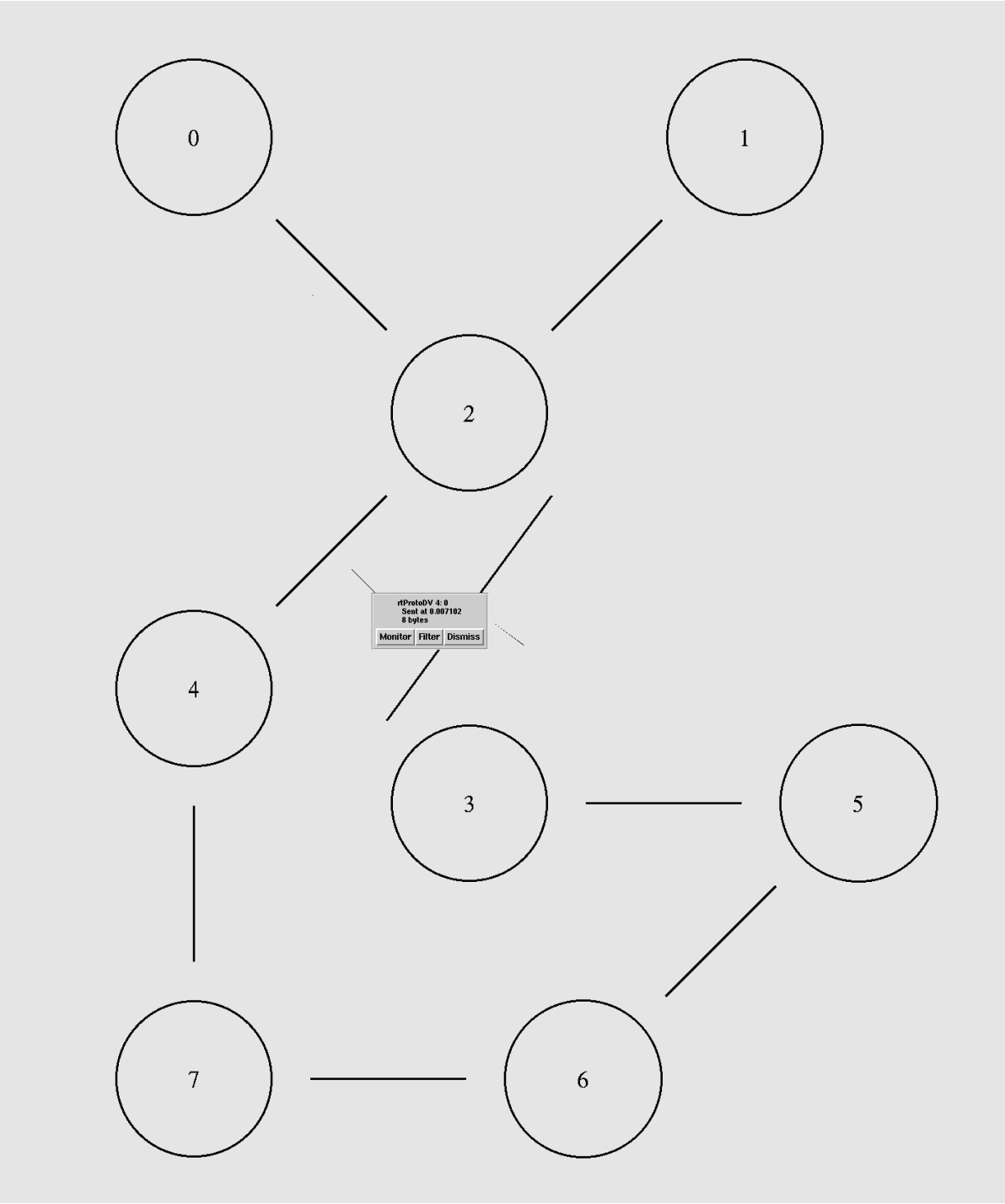


FIGURE 12 – Mise à jour de routage initiale DV

Comme on peut le voir sur l'image ci-dessus, le paquet de routage capturé a été émis à 0.007102 seconde. C'est donc ce qu'on appelle un paquet de routage initial qui permet aux différents routeurs de savoir quels chemins emprunter pour leurs communications.

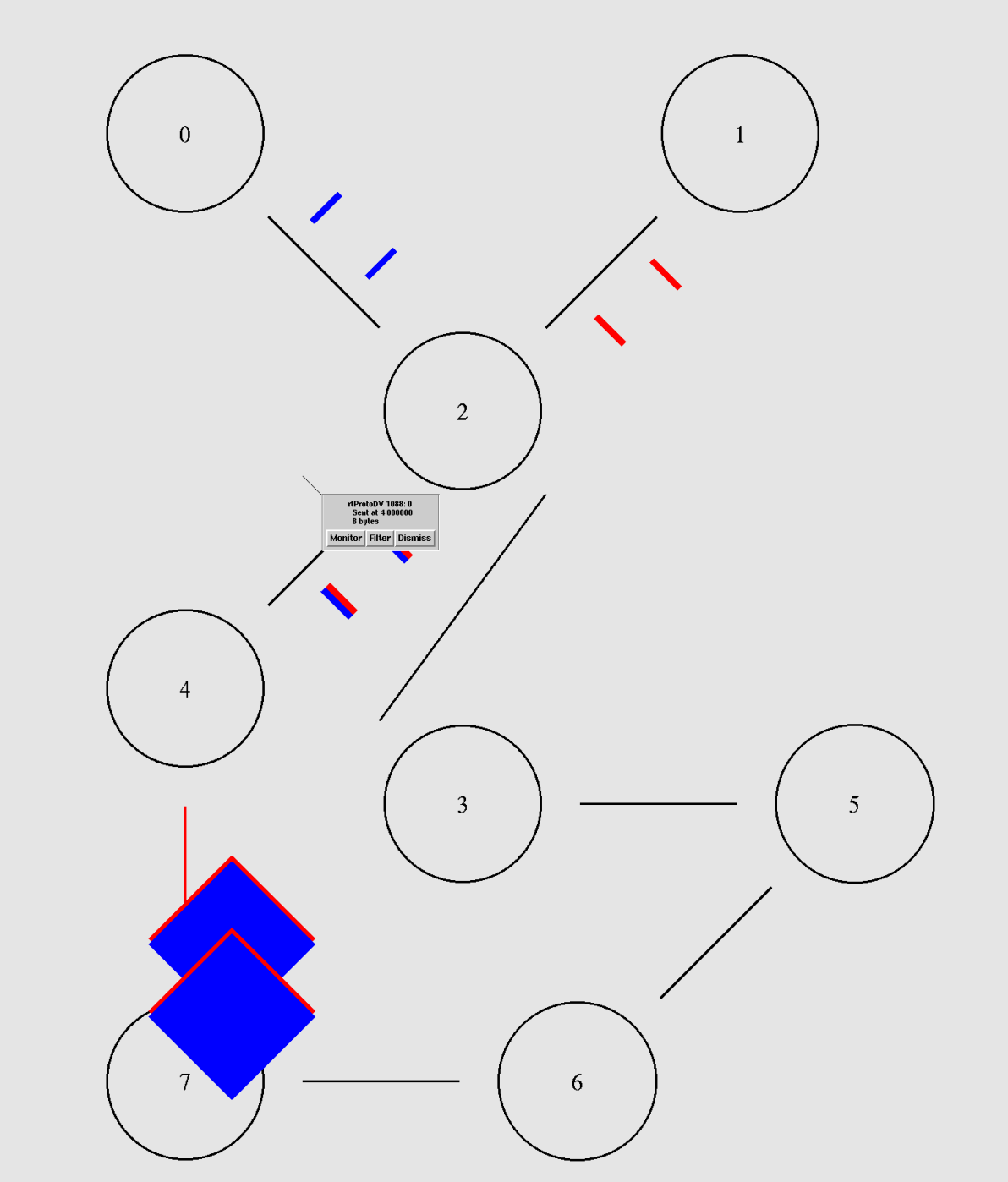


FIGURE 13 – Mise à jour de routage rupture DV

Comme le montre la figure ci-dessus, lors de la rupture, instantanément des paquets de routage sont échangés afin d'informer les autres noeuds de la rupture.

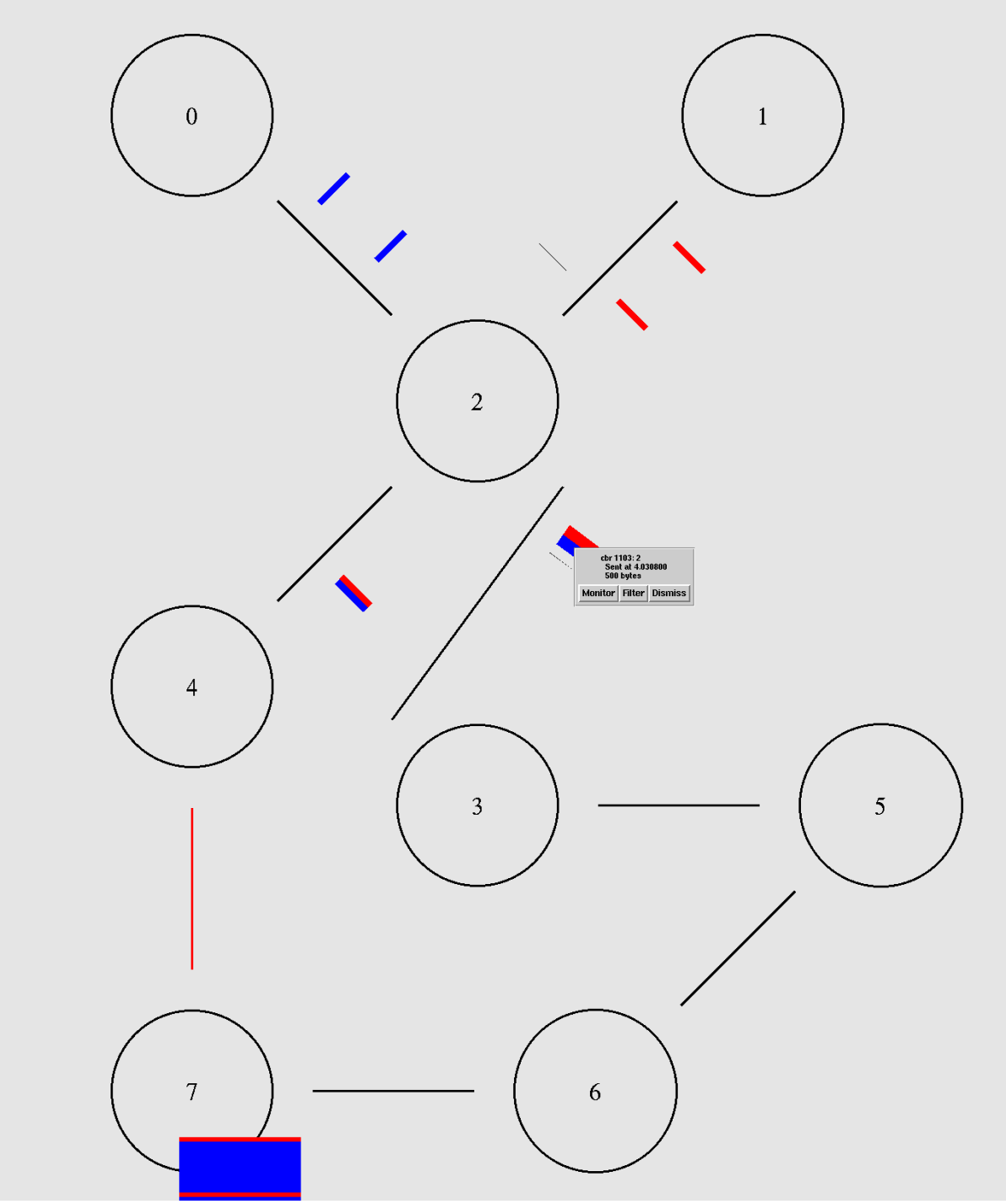


FIGURE 14 – Premier paquet après rupture DV

On voit sur la figure ci-dessus, que le premier paquet à changer de route a été émis à 4.030800 secondes. Il faut donc environ 30ms pour que le réseau se rende compte de la rupture et change de chemin avec un protocole de routage à Vecteur de Distance (DV).

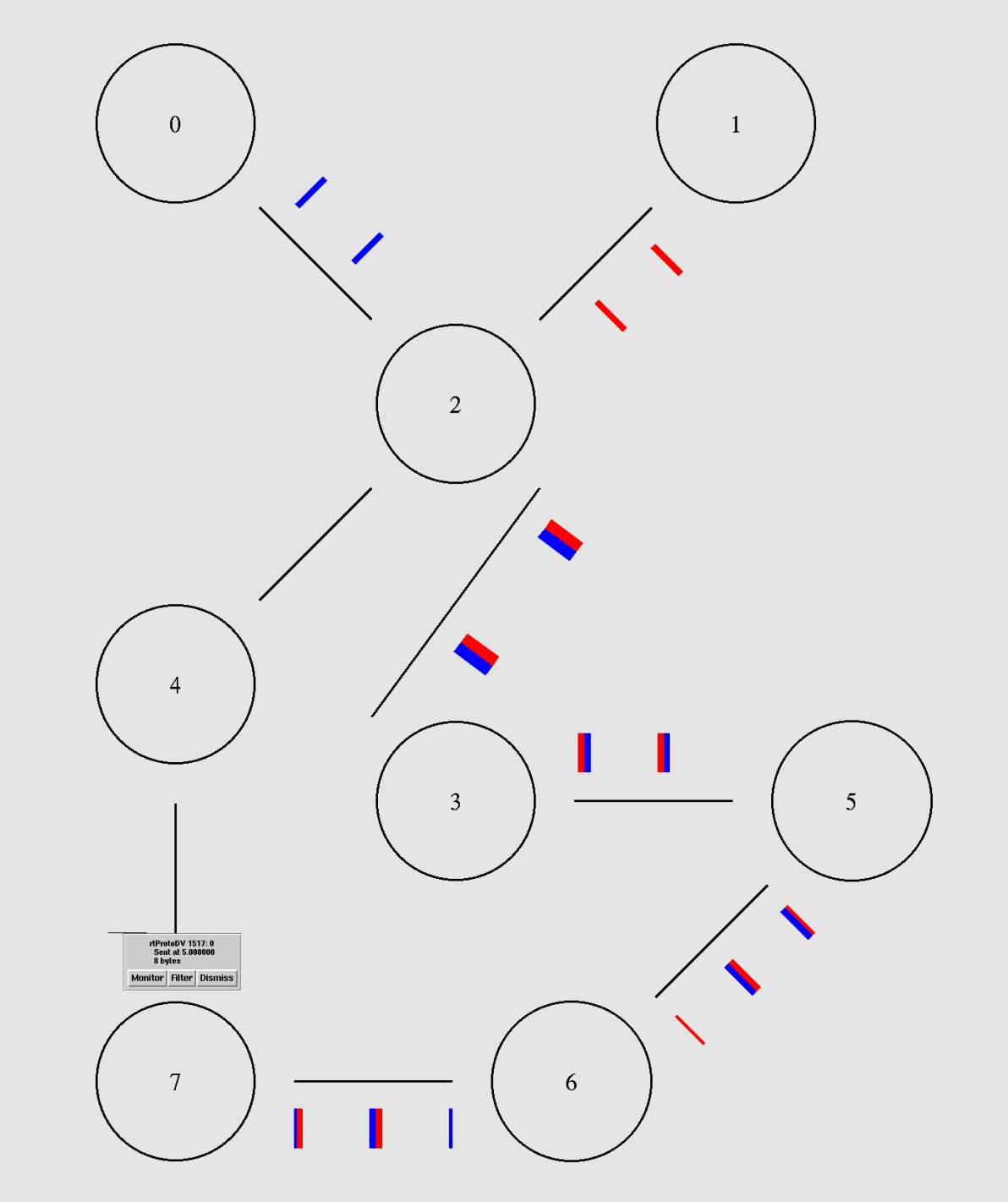


FIGURE 15 – Mise à jour de routage rétablissement DV

On peut voir sur l'image du dessus que lors du rétablissement du lien, des paquets de routage sont émis pour mettre à jour les tables de routage de tous les routeurs afin de retrouver le meilleur chemin.

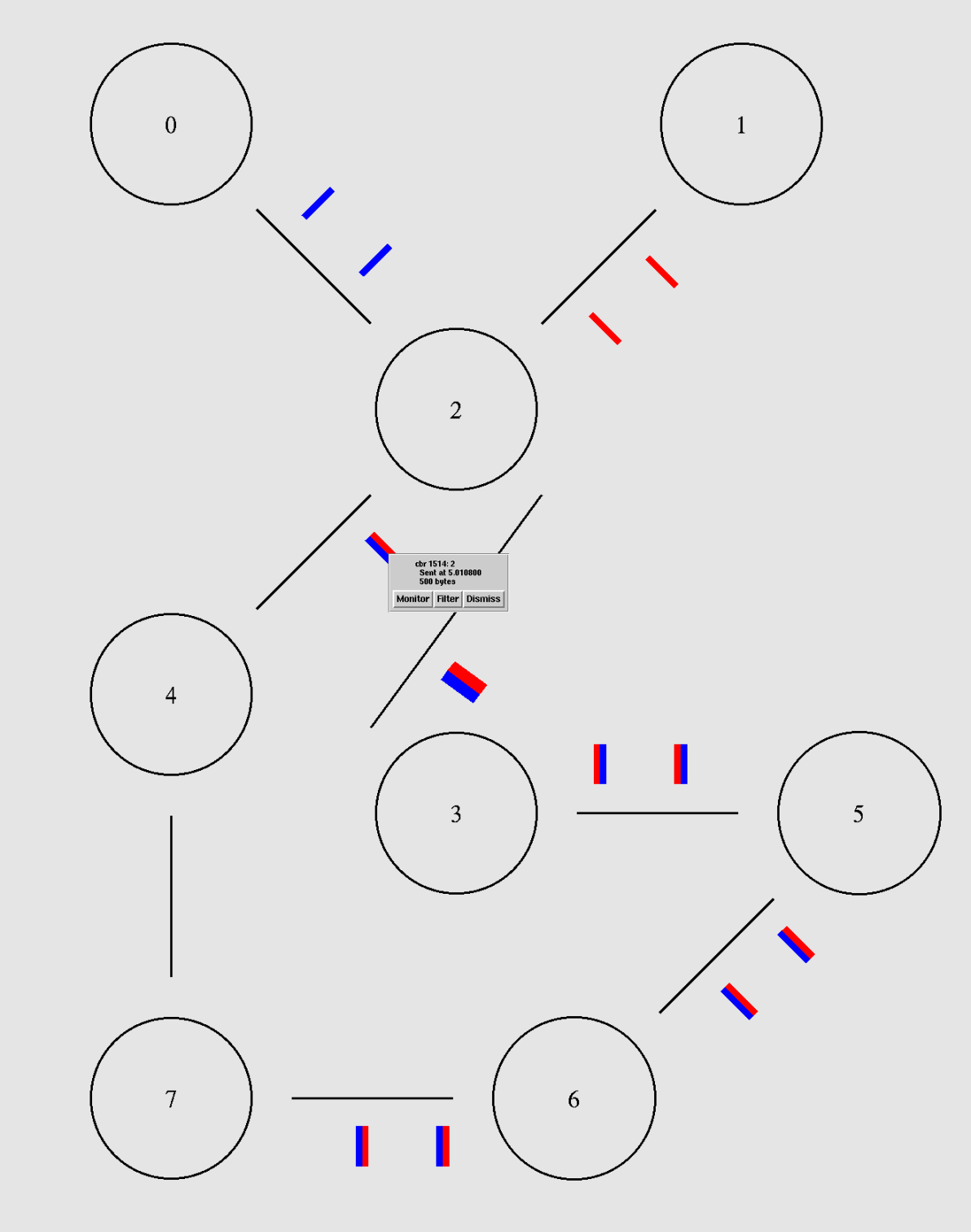


FIGURE 16 – Premier paquet après rétablissement DV

Le premier paquet à reprendre le meilleur chemin après rétablissement est émis à 5.010800. Il faut donc environ 10ms au réseau pour se rendre compte qu'il y a eu rétablissement et modifier son comportement. On peut donc noter que pour le protocole de routage à Vecteur de Distance (DV) il est plus long de réagir à une rupture qu'à un rétablissement.

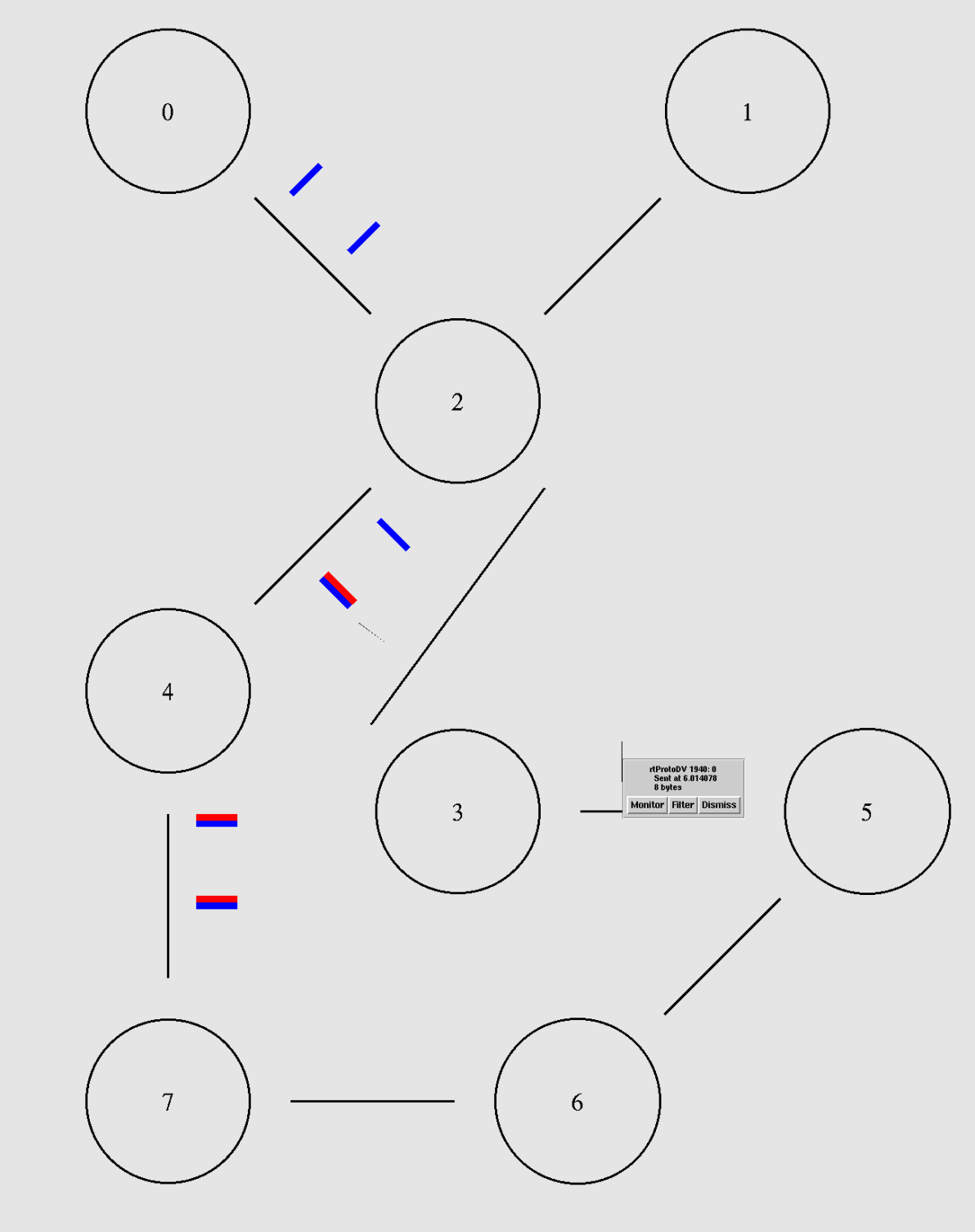


FIGURE 17 – Mise à jour de routage périodique DV

Il existe également un autre moment ou des paquets de routage sont échangés, ce sont les paquets de routage périodiques. Ils permettent d'échanger entre chaque routeur les routes dont ils ont connaissance.

3.2.4 Visualisation sur nam protocole de routage LS

```
18 #Déclaration du protocole de routage
19 $ns rtproto LS
```

Code 30 – Mise en place du protocole de routage LS

Le seul changement à effectuer dans le code précédent est le suivant : remplacer DV par LS pour passer du protocole de routage à Vecteur de Distance à un protocole de routage à Etat de Liens. Ce protocole de routage permet aux routeurs de construire une carte complète de la topologie du réseau. Chaque routeur calcul localement le meilleur chemin vers chaque destination.

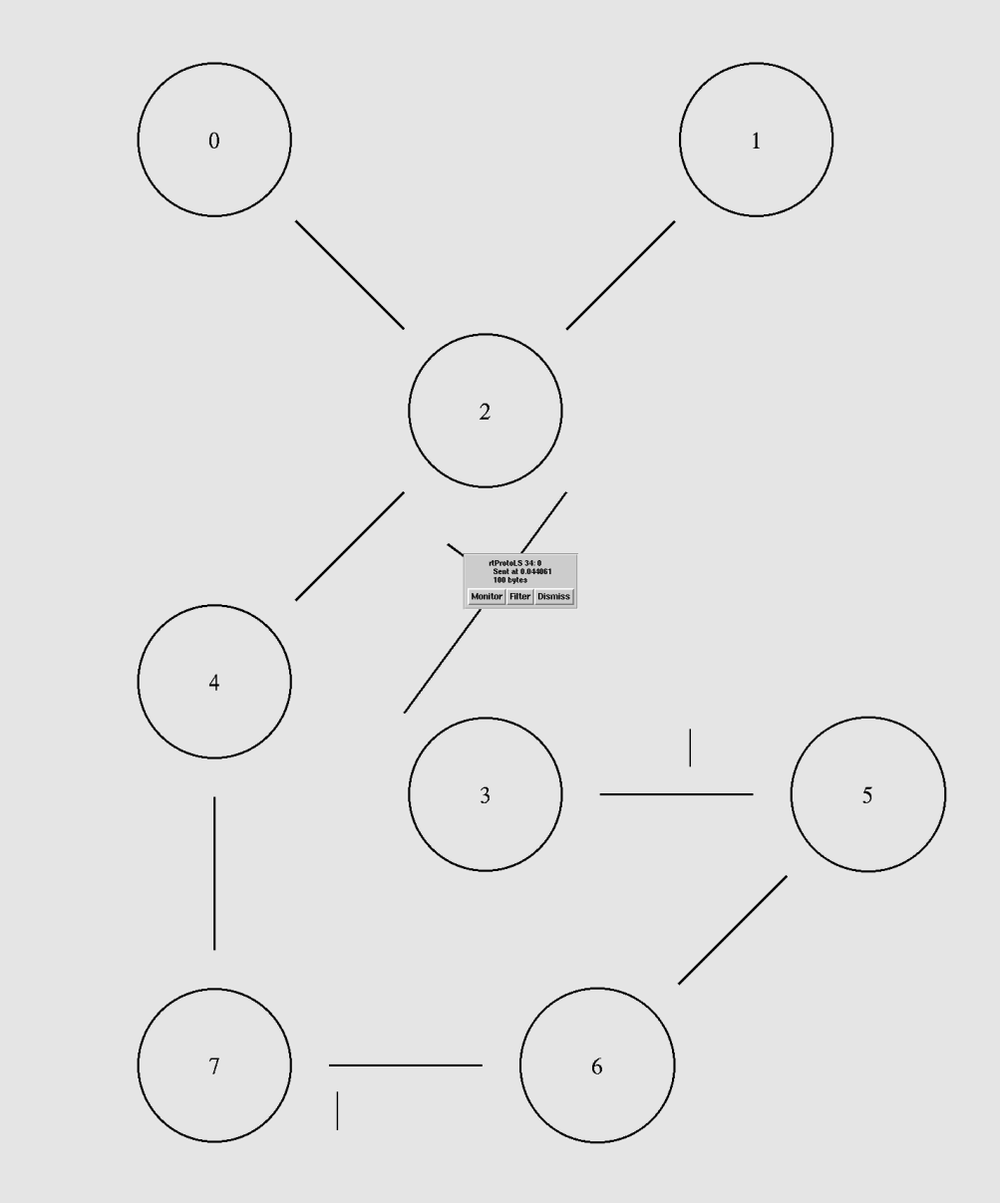


FIGURE 18 – Mise à jour de routage initiale LS

Tout comme le protocole de routage Dv, le protocole de routage LS échange des paquets de routage dès sa mise en route afin que les routeurs puissent prendre connaissance de la topologie du réseau et déterminer les meilleurs chemins.

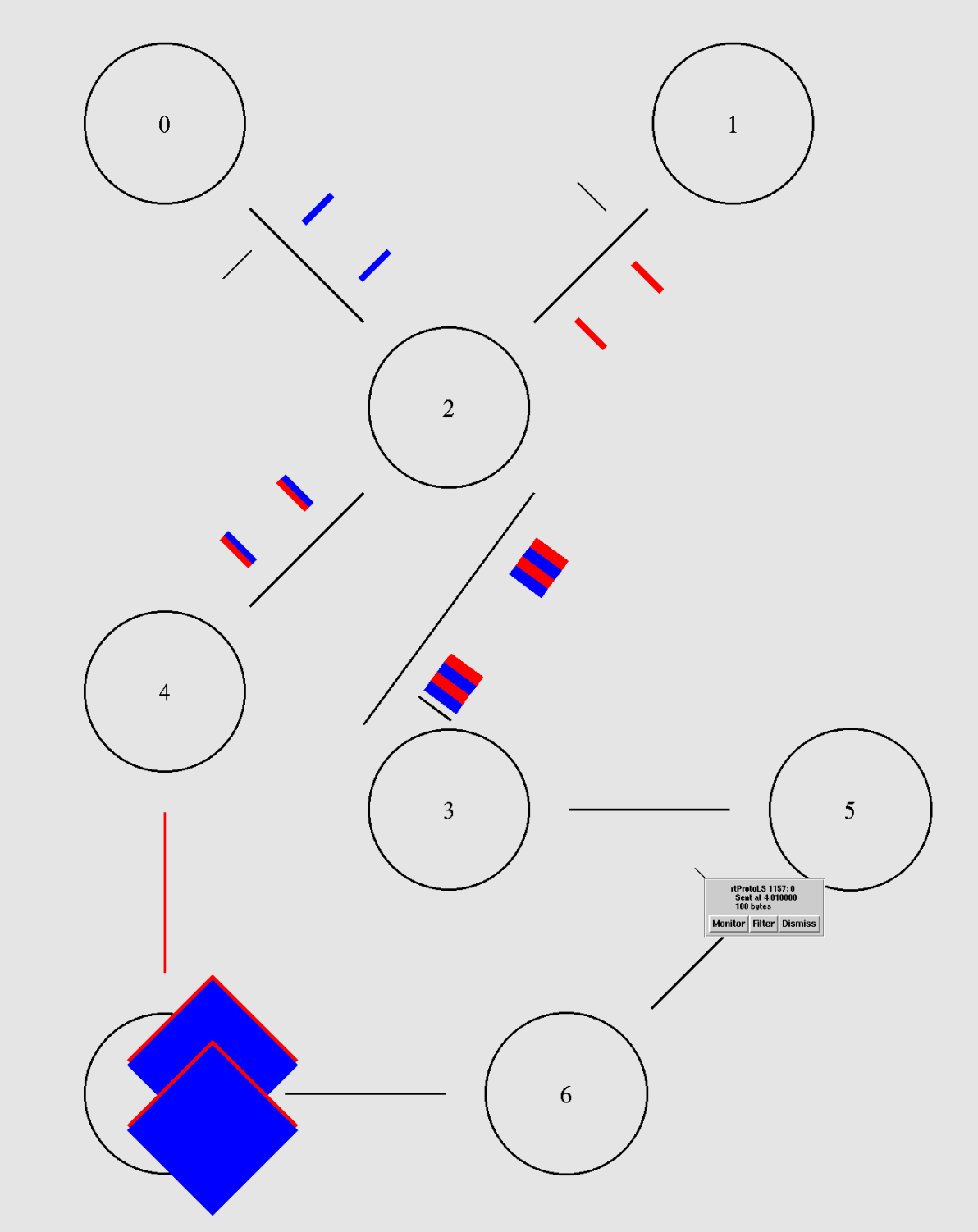


FIGURE 19 – Mise à jour de routage rupture LS

Comme le montre la figure ci-dessus, lors de la rupture, instantanément, des paquets de routage sont échangés afin d'informer les autres noeuds de la rupture.

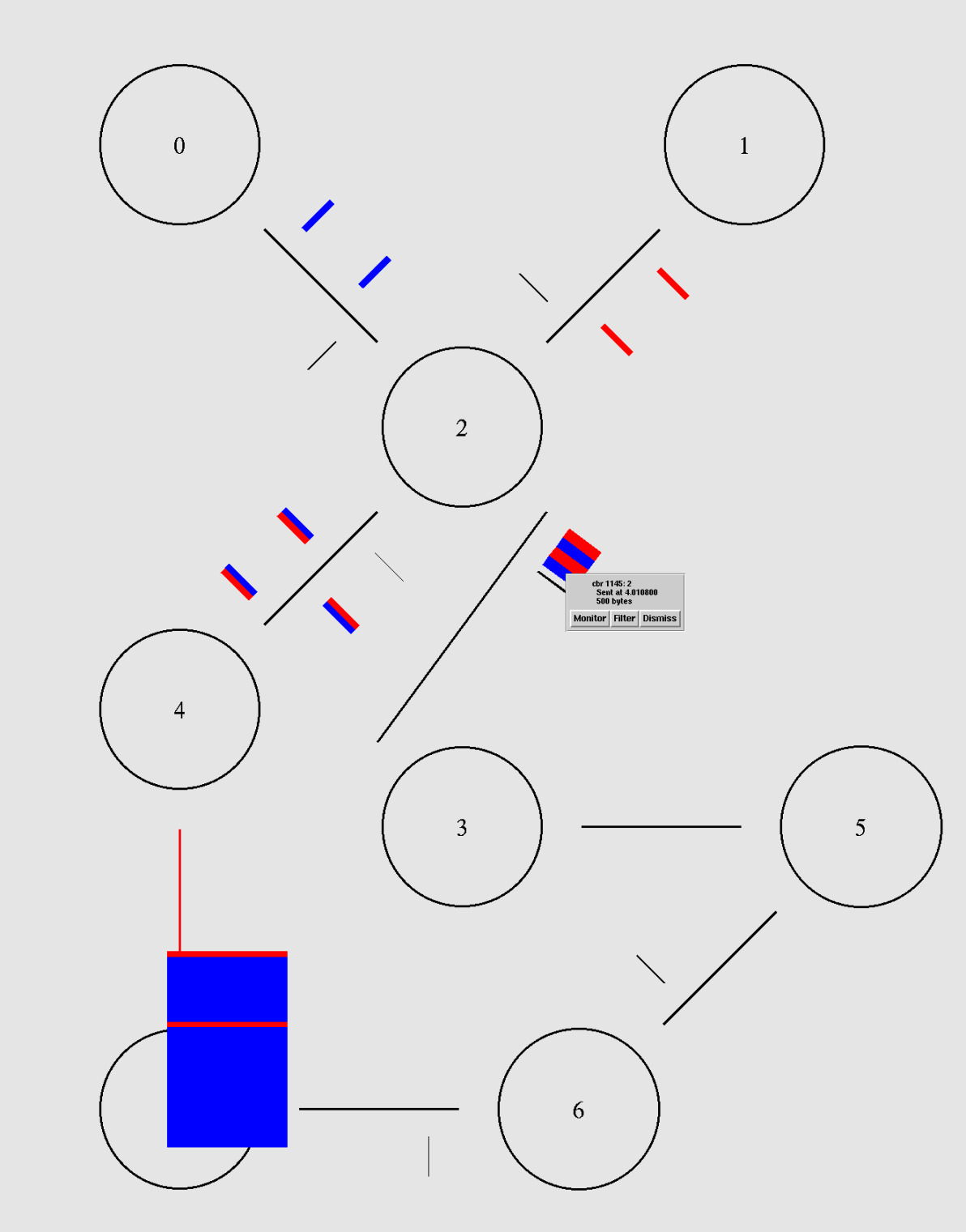


FIGURE 20 – Premier paquet après rupture LS

Lors de la rupture, le premier paquet à emprunter la nouvelle route le fait à 4.010800 secondes. Il faut donc environ 10ms au réseau pour réagir face à la rupture du lien.

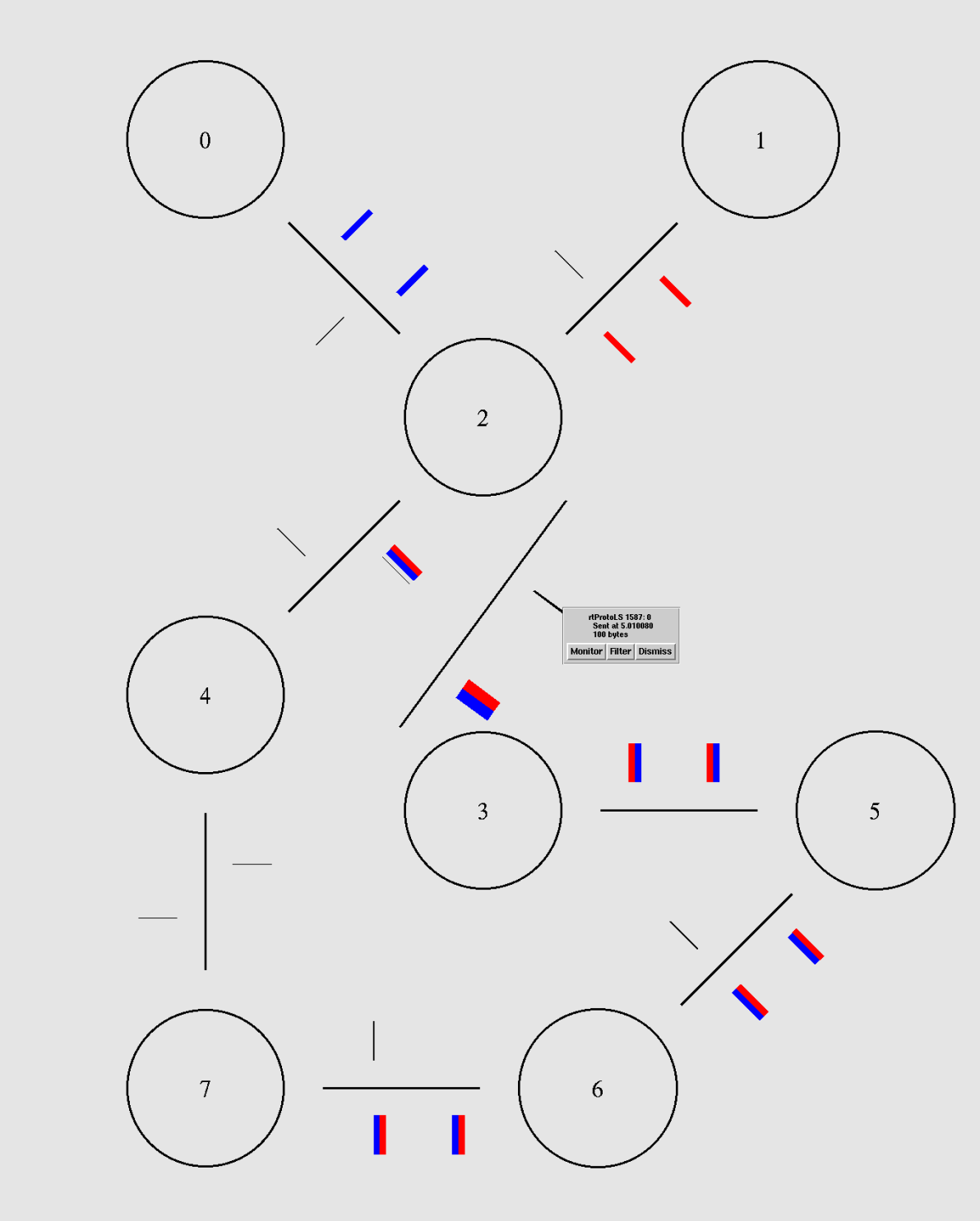


FIGURE 21 – Mise à jour de routage rétablissement LS

On peut voir sur l'image du dessus que lors du rétablissement du lien, des paquets de routage sont émis pour mettre à jour les tables de routage de tous les routeurs afin de retrouver le meilleur chemin.

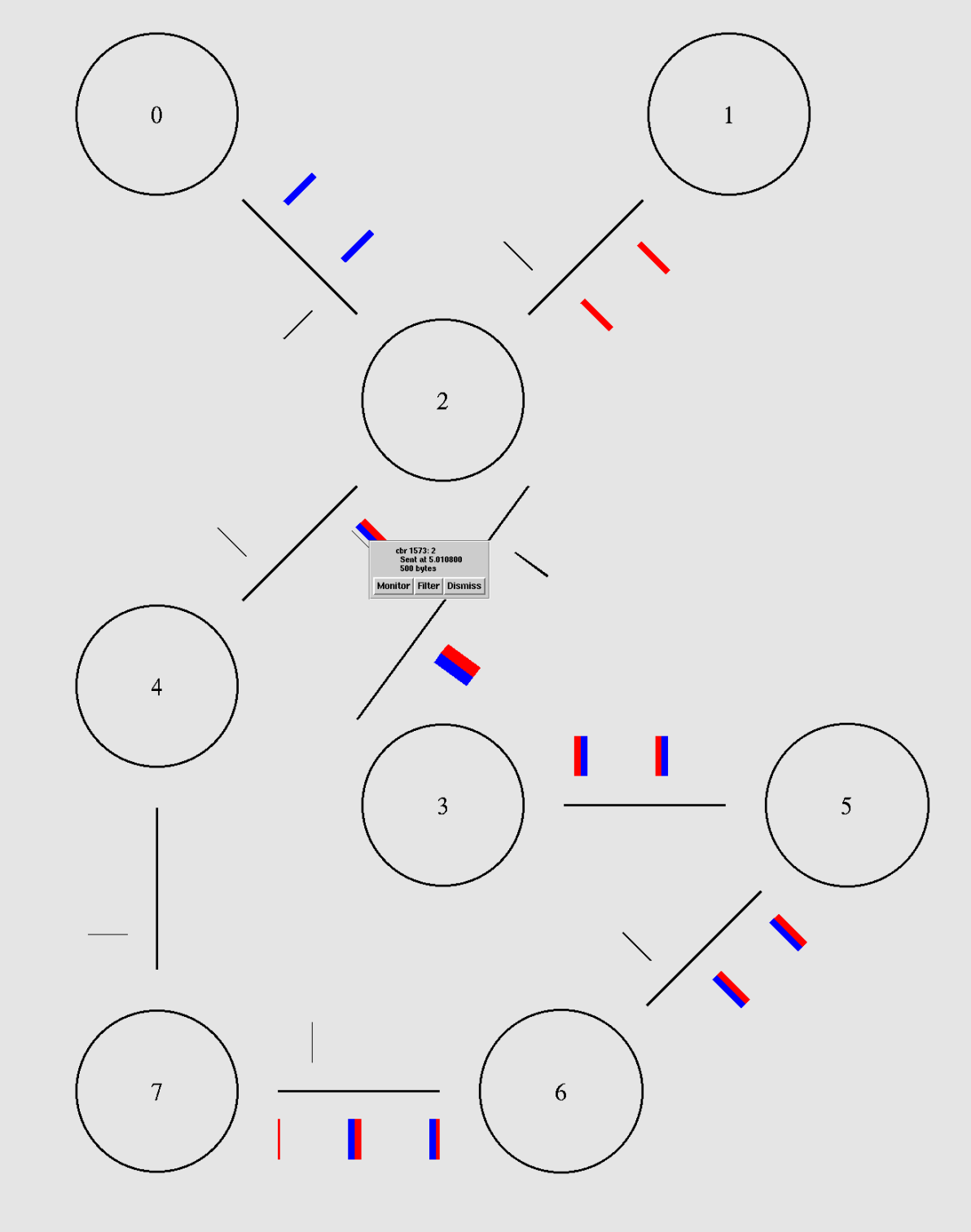


FIGURE 22 – Premier paquet après rétablissement LS

Le premier paquet à emprunter le nouveau meilleur chemin après rétablissement le fait à 5.010800 secondes. Ainsi il faut donc au protocole de routage à État de Liens environ 10ms pour réagir lors du rétablissement du lien.

On peut également noter qu'il n'y a pas la présence de paquet de routage périodique.

3.2.5 Comparaison entre le protocole de routage DV et LS

//////////////////	Routage DV	Routage LS
Convergence	30ms rupture / 10ms rétablissement	10ms rupture / 10ms rétablissement
MAJ de routage	Initiale, Périodique, Rupture, Rétablissement	Initiale, Rupture, Rétablissement
Comportement du noeud n(5) au moment de la rupture du lien entre n(5) et n(8)	continue d'envoyer les paquets au noeud n(8) malgré la rupture du lien ce qui entraîne de nombreuses pertes jusqu'à ce que le réseau réagisse	redirige les paquets dans le sens inverse pour qu'il soit directement renvoyer par le meilleur chemin ce qui entraîne beaucoup moins de perte

3.3 Exercice : 2

3.3.1 But de l'exercice

Le but de l'exercice est de simuler un réseau avec une rupture et un rétablissement de lien afin de voir l'impact des pertes sur l'évolution d'une communication TCP. Le réseau simulé possédera 4 noeuds principaux A, B, C et D. Toutes les liaisons auront un coût de un sauf la liaison entre C et D qui aura un coût de quatre. Ce coût sera matérialisé par trois noeuds entre C et D puisque l'on ne peut pas spécifier un coût lors de la connexion entre deux noeuds. Le noeud A sera émetteur d'un trafic FTP via TCP vers le noeud D. L'objectif final de l'exercice est de tracer une courbe de performance avec xgraph et de l'exploiter pour mettre en évidence toutes les phases de TCP vues en cours comme le contrôle de congestion, contrôle de flux, Slow Start, Congestion Avoidance etc...

3.3.2 Explication du code

```
#Creation 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 procedure de terminaison de la simulation
proc finish {} {
    global ns nf
    $ns flush-trace
    #Fermer le fichier trace
    close $nf
    #Executer le nam avec en entree le fichier trace
    exec nam out.nam &
    exit 0
}
```

Code 31 – Structure générale

Les premières lignes de code sont semblables à celles de tous les scripts. Le détail de ces lignes est disponible dans la partie **2.2**.

```
16 #Mise en place du protocole de routage
17 $ns rtproto DV
```

Code 32 – Mise en place du protocole de routage

Le protocole de routage indiqué dans l'énoncé est le protocole à Vecteur de Distance. Le détail du fonctionnement de ce protocole est disponible dans la partie **3.2.2**.

```
20 #Nommer les noeuds et changer la forme
21 $n(1) label "A"
22 $n(2) label "B"
23 $n(3) label "C"
24 $n(4) label "D"
25 $n(1) shape box(hexagon)
26 $n(4) shape box(hexagon)
```

Code 33 – Nommage des noeuds et changement de leur forme

Grâce à ces commandes, il est possible de donner un nom au noeud. On nomme ainsi les noeuds principaux A, B, C et D. Pour identifier plus facilement le noeud émetteur et le noeud récepteur, on modifie leur forme pour la forme "hexagon" qui donne un carré à la place d'un rond.

```
27 #Créations des différents liens
28 $ns duplex-link $n(1) $n(2) 10Mb 10ms DropTail
29 $ns duplex-link $n(1) $n(3) 10Mb 10ms DropTail
30 $ns duplex-link $n(2) $n(3) 10Mb 10ms DropTail
31 $ns duplex-link $n(2) $n(4) 10Mb 10ms DropTail
32 $ns duplex-link $n(3) $n(5) 10Mb 10ms DropTail
33 $ns duplex-link $n(5) $n(6) 10Mb 10ms DropTail
34 $ns duplex-link $n(6) $n(7) 10Mb 10ms DropTail
35 $ns duplex-link $n(7) $n(4) 10Mb 10ms DropTail
```

Code 34 – Créations des différents liens

On crée les différents liens en faisant attention aux paramètres demandés dans l'énoncé qui sont un débit de 10Mb/s, un delai de 10ms et un buffer de type DropTail. Le détail de création des liens est disponible dans la partie **2.3.2**.

```
36 #Orientation des différents noeuds
37 $ns duplex-link-op $n(1) $n(2) orient left-down
38 $ns duplex-link-op $n(1) $n(3) orient right-down
39 $ns duplex-link-op $n(2) $n(3) orient right
40 $ns duplex-link-op $n(2) $n(4) orient right-down
41 $ns duplex-link-op $n(3) $n(5) orient down
42 $ns duplex-link-op $n(5) $n(6) orient down
43 $ns duplex-link-op $n(6) $n(7) orient down
44 $ns duplex-link-op $n(7) $n(4) orient left-down
```

Code 35 – Orientation des différents noeuds

On oriente les noeuds de manière à avoir une topologie semblable à celle demandée. Le détail d'orientation des noeuds est disponible dans la partie **2.4.2**. On obtient donc la topologie ci-dessous.

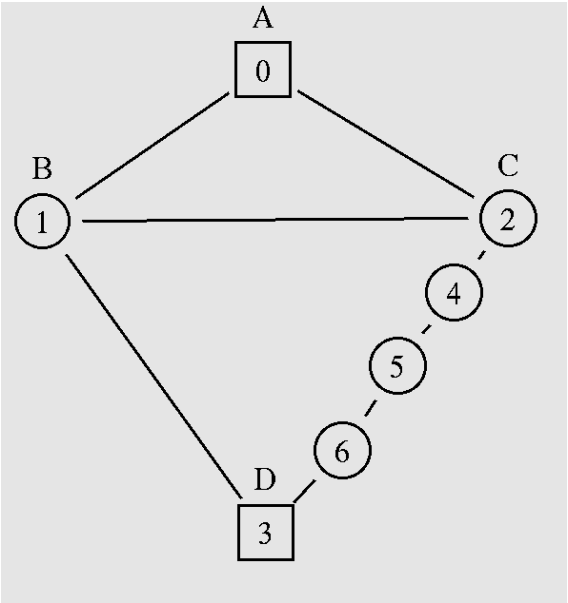


FIGURE 23 – Topologie Exercice : 2

```
45 #Création et connexion des différents Agents
46 set tcp [new Agent/TCP]
47 $ns attach-agent $n(1) $tcp
48 set tcpsink [new Agent/TCPSink]
49 $ns attach-agent $n(4) $tcpsink
```

Code 36 – Créations des différents Agents

On crée l'Agent TCP pour le noeud A et TCPSink pour envoyer des ACK depuis le noeud D. Le détail de création d'Agent est disponible dans la partie **2.3.2**.

```
50 #Création et connexion de l'Application FTP
51 set ftp [new Application/FTP]
52 $ftp attach-agent $tcp
53 $ns connect $tcp $tcpsink
```

Code 37 – Création des différentes Applications

On crée une Application FTP qui utilisera l'Agent TCP pour communiquer. Le détail de création d'Application est disponible dans la partie **2.3.2**.

```
54 #Evenements de simulation
55 $ns at 1 "$ftp start"
56 $ns rtmodel-at 3.48 down $n(2) $n(4)
57 $ns rtmodel-at 4.95 up $n(2) $n(4)
58 $ns at 7 "$ftp stop"
59 #Appeler la procédure de terminaison apres un temps t
60 $ns at 8.0 "finish"
61 #Executer la simulation
62 $ns run
```

Code 38 – Evénements de simulation

Comme indiqué, le trafic FTP commence à 1 seconde et s'arrête à 7 secondes. La rupture du lien entre B et D à lieu à 3.48 secondes et le rétablissement à lieu à 4.95 secondes. Vous trouverez l'entièreté du code dans l'Annexe **6**.

3.3.3 Visualisation sur nam

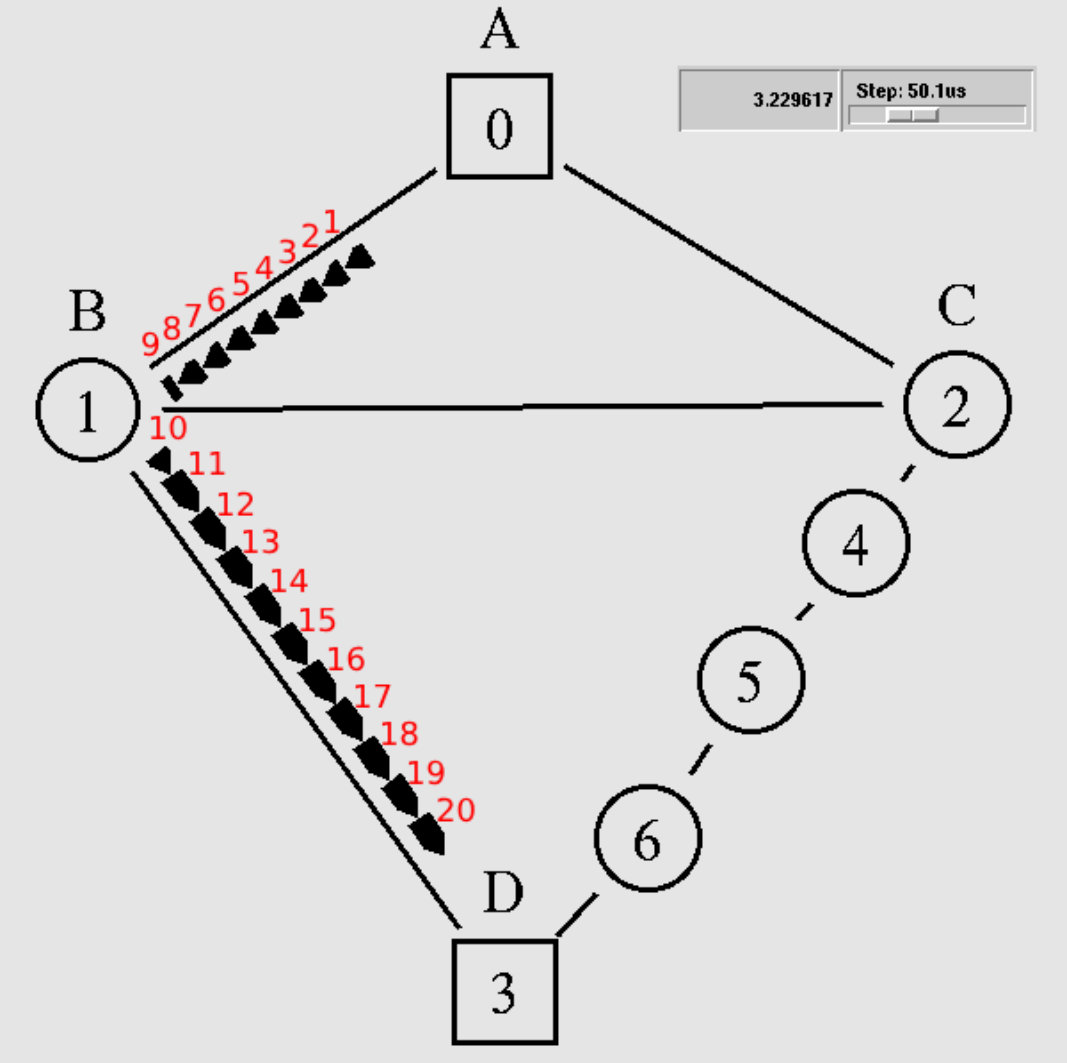


FIGURE 24 – Fenêtre TCP avant rupture

Comme on le voit sur l'image ci-dessus, la fenêtre TCP avant la rupture est de 20 SMSS et ce depuis un certain nombre de RTT. Comme il n'y a pas eu de drop on peut en conclure que c'est donc la valeur de rwnd qui limite la fenêtre TCP. On verra en détail l'évolution de la fenêtre TCP dans la partie analyse xgraph.

Comme on peut le voir sur la figure ci-dessous, comme un drop a été détecté, cwnd est passé à un SMSS. Ainsi la valeur la plus faible entre cwnd et rwnd est 1 donc un seul paquet est émit. La fenêtre TCP va doubler grâce au Slow Start jusqu'à ce que cwnd soit supérieur au ssthresh qui est de 10 SMSS. Le détail du fonctionnement sera expliqué dans la partie suivante mettre ref. Une fois que la fenêtre d'envoi de TCP a atteint 10 SMSS, on passe à la phase de Congestion Avoidance c'est à dire que la fenêtre s'incrémente de un SMSS à chaque RTT.

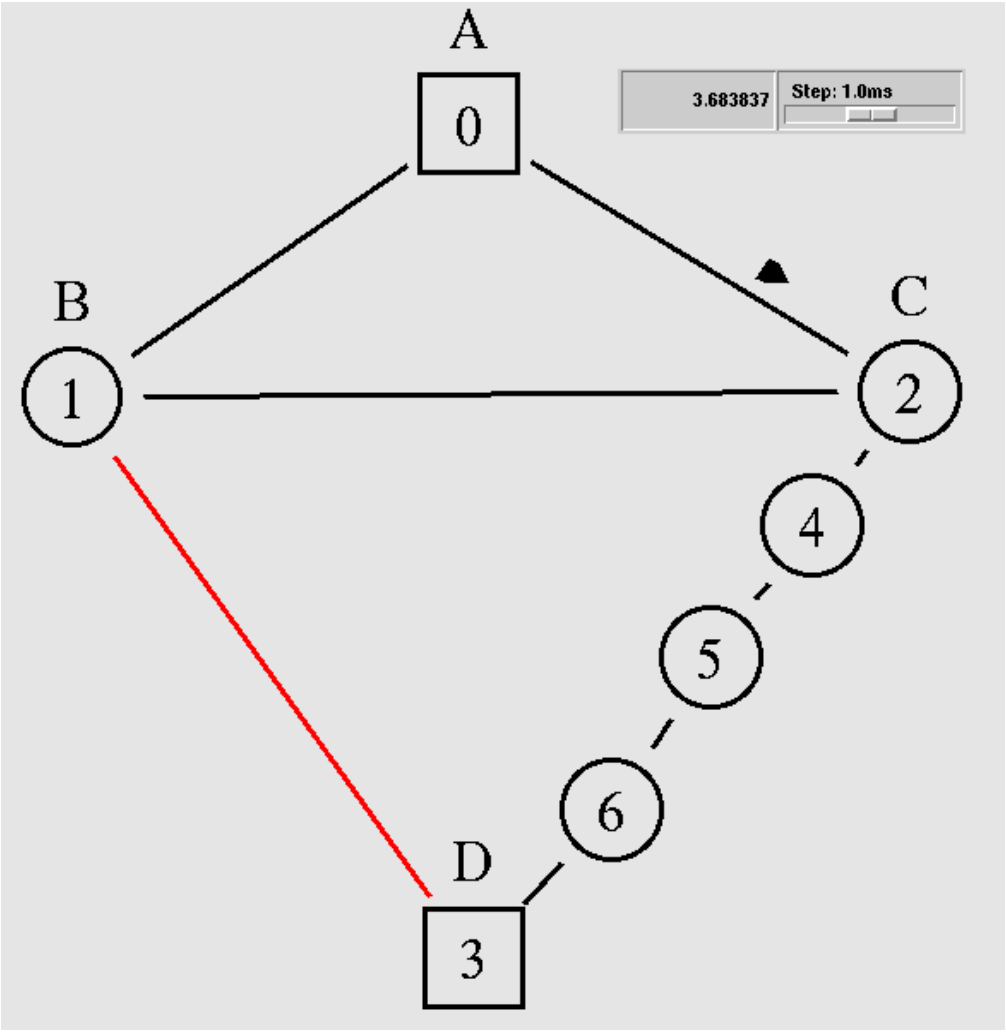


FIGURE 25 – Premier paquet après rupture

Comme on peut le voir sur la figure 26 ci-dessous, la fenêtre TCP utilise bien le Slow Start pour arriver au ssthresh de 10 SMSS. On voit à gauche qu'il y a 8 paquets et à droite il y a 10 paquets. On note également la présence de paquets de routage périodique sur l'image gauche de la figure 26. On voit également sur l'image gauche de la figure 27 la phase de Congestion Avoidance. Étant donnée qu'aucun type de buffer n'a été spécifié, les noeuds n'ont pas de file d'attente. Il est donc impossible de saturer le réseau ainsi la phase de Congestion Avoidance va durer jusqu'à ce que la fenêtre TCP atteigne le rwnd soit 20 SMSS.

Après avoir anaylsé le comportement du réseau avec nam, on passe à l'analyse de ses performances avec xgraph. Dans la partie suivante on analysera le code permettant de créer des fichiers de sortie utilisables par xgraph et on analysera dans le détail les différentes phases TCP.

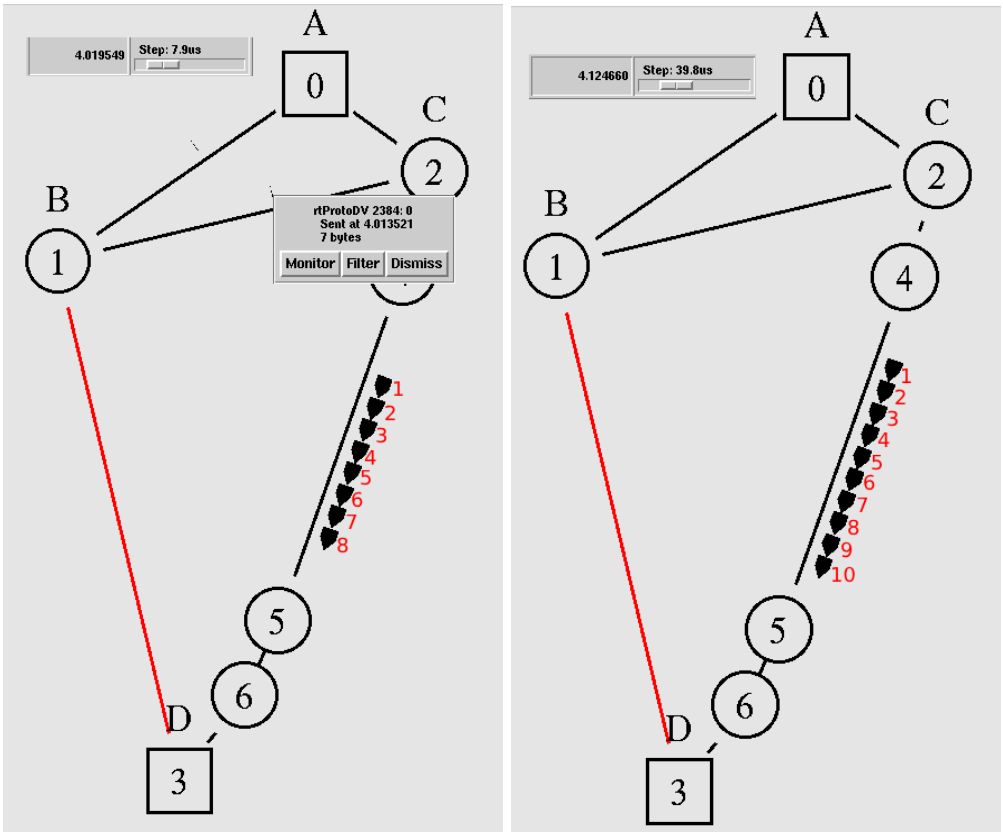


FIGURE 26 – Évolution de cwnd jusqu'à ssthresh

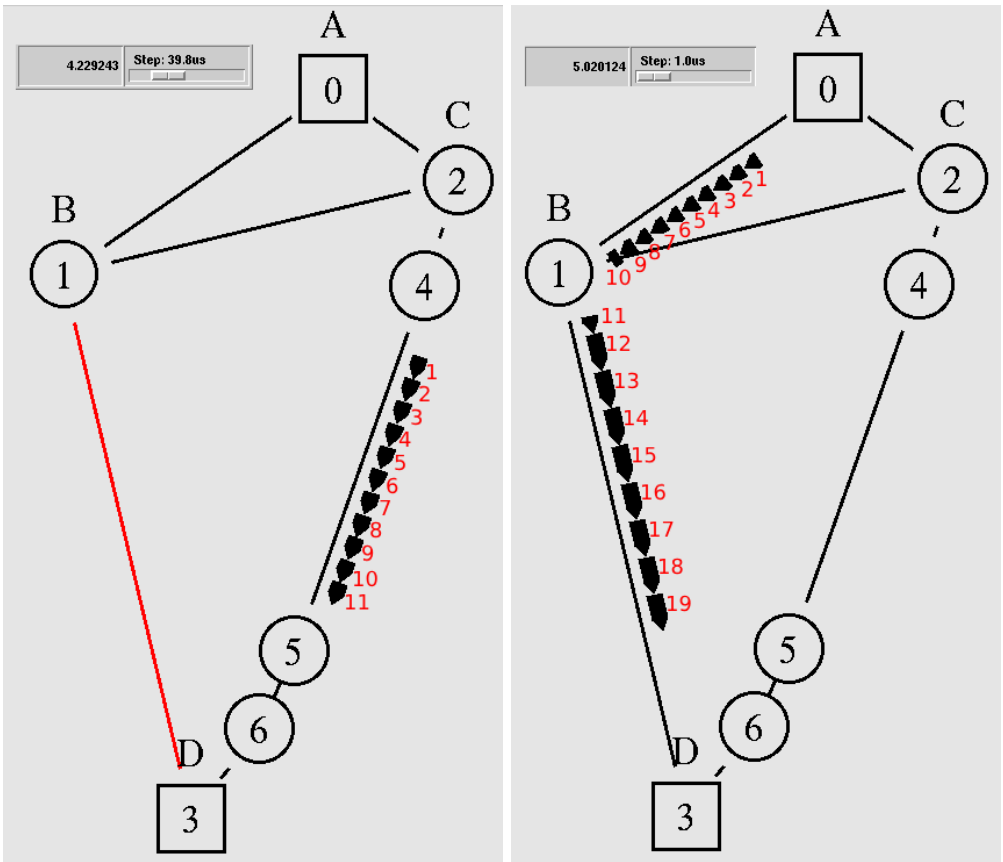


FIGURE 27 – Congestion Avoidance et rétablissement

3.3.4 Analyse des performances du réseau avec xgraph

Dans un premier temps on va commenter le code utilisé. Le code est le même que celui de l'exercice 2 mais on a rajouté les lignes suivantes :

```
7  #Création d'un fichier nommé WindowVsTime en écriture
8  set windowVsTime [open WindowVsTime w]
```

Code 39 – Création du fichier de sortie

On crée un fichier de sortie différent de out.nam car on va écrire des informations différentes qui seront utilisés par xgraph et non par nam. Le nom du fichier de sortie est WindowVsTime car les informations qui vont être enregistrées sont les valeurs de la fenêtre TCP au cours du temps. Ce fichier est ouvert en mode écriture comme le précise le w. Par la suite, il sera possible d'accéder à ce fichier via la variable windowVsTime en la précédent d'un $.

```
9  #Création de la fonction plotWindow avec 2 paramètres
10 proc plotWindow {tcpSource windowVsTime} {
```

Code 40 – Déclaration de la fonction de sauvegarde

On déclara ensuite une fonction appelée plotWindow car cette fonction va écrire dans le fichier la valeur de la fenêtre à un certain temps. Cette fonction prend deux paramètres en entrée. Un Agent TCP source qui sera indiqué lors de l'appel de la fonction et qui sera accessible via la variable tcpSource. Le deuxième paramètre est un fichier de sortie pour écrire les informations. Une fois ce fichier passé en paramètre lors de l'appel de la fonction, il sera accessible via la variable windowVsTime.

```
11     #Modification de la portée de la variable
12     global ns
```

Code 41 – Modification de la portée de la variable ns

On modifie la portée de la variable ns pour la rendre accessible à l'intérieur de la fonction car comme on va le voir par la suite, on va appeler la fonction dans la fonction pour qu'elle s'exécute automatiquement tous les x temps.

```
13     #Création de la variable time
14     set time 0.1
```

Code 42 – Déclaration du temporisateur

On crée la variable time qui a comme valeur 0.1. On va s'en servir comme temporisateur pour appeler la fonction toutes les 0.1 seconde. On prend un intervalle de temps assez faible afin d'avoir une précision suffisamment précise pour avoir des résultats exploitables.

```
15     #Affectation de la valeur de cwnd dans la variable cwnd
16     set cwnd [$tcpSource set cwnd_]
```

Code 43 – Mise à jour de la valeur de la fenêtre

A l'aide de la commande $tcpSource set cwnd_ on récupère la valeur de cwnd de l'Agent TCP pointé par $tcpSource qui n'est autre que l'Agent donné en paramètre lors de l'appel de la fonction c'est à dire l'Agent TCP de la simulation vu précédement dans la partie **3.3.2**.

```
17      #Limitation à rwnd au maximum
18      if {$cwnd >= 20} {
19          set cwnd 20
20      }
```

Code 44 – Limitation à rwnd

La valeur de la fenêtre TCP est la valeur minimale entre cwnd et rwnd, cwnd étant incrémenté à chaque RTT, il est indispensable de spécifier que si cwnd est supérieure à rwnd alors la taille de la fenêtre se limite à rwnd. Comme vu précédement dans la partie **3.3.3** la valeur de rwnd est de 20 SMSS. On crée donc une condition qui vérifie si cwnd est supérieure à 20 SMSS. Si c'est le cas alors la valeur de la variable cwnd devient 20 SMSS sinon elle garde la valeur précédemment obtenue.

```
21      #MAJ du temps
22      set now [$ns now]
```

Code 45 – Mise à jour du temps

Cette ligne de code permet de mettre à jour le temps dans la fonction en fonction du temps de la simulation.

```
23      #Enregistrement de la valeur de la fenêtre TCP au cours du temps
24      puts $windowVsTime "$now $cwnd"
```

Code 46 – Enregistrement de la valeur et du temps

Grâce à cette commande, on écrit dans le fichier fournit en paramètre pointé par $windowVsTime le temps actuel de simulation puis la valeur de la fenêtre TCP à ce temps.

```
11      #Modification de la portée de la variable
12      global ns
```

Code 47 – Modification de la portée de la variable ns

```
26      #Appel de la fonction par elle même toutes les 0.1 seconde
27      $ns at [expr $now+$time] "plotWindow $tcpSource $windowVsTime"
28  }
```

Code 48 – Appel automatique de la fonction

Avec ces dernières lignes, on appelle la fonction plotWindow toutes les 0.1 seconde. En effet, la commande expr $now+$time incrémente le temps de l'appel de 0.1 seconde. Par exemple si l'on a enregistré une valeur à 3 secondes, la valeur de $now est 3 et comme $time est 0.1 alors à 3.1 secondes, l'évenement sera exécuté. En passant en paramètres $tcpSource et $windowVsTime, on indique les paramètres donnés lors de l'appel initial comme expliqué dans la partie **3.3.4**. Avec ce système il est donc possible d'enregistrer la valeur de la fenêtre toutes les 0.1 seconde en appelant une seule fois la fonction comme suit :

```
79  #Appel initial de la fonction avec paramètres
80  $ns at 0 "plotWindow $tcp $windowVsTime"
```

Code 49 – Appel initial de la foncction

En appelant la fonction à t = 0s et en indiquant en paramètres $tcp et $windowVsTime la fonction va s'éxecuter correctement toutes les 0.1 secondes et on aura ainsi à notre disposition toutes les informations nécessaires pour utiliser xgraph. Vous trouverez le code dans l'Annexe **6**. Avec la commande xgraph WindowVsTime on obtient :

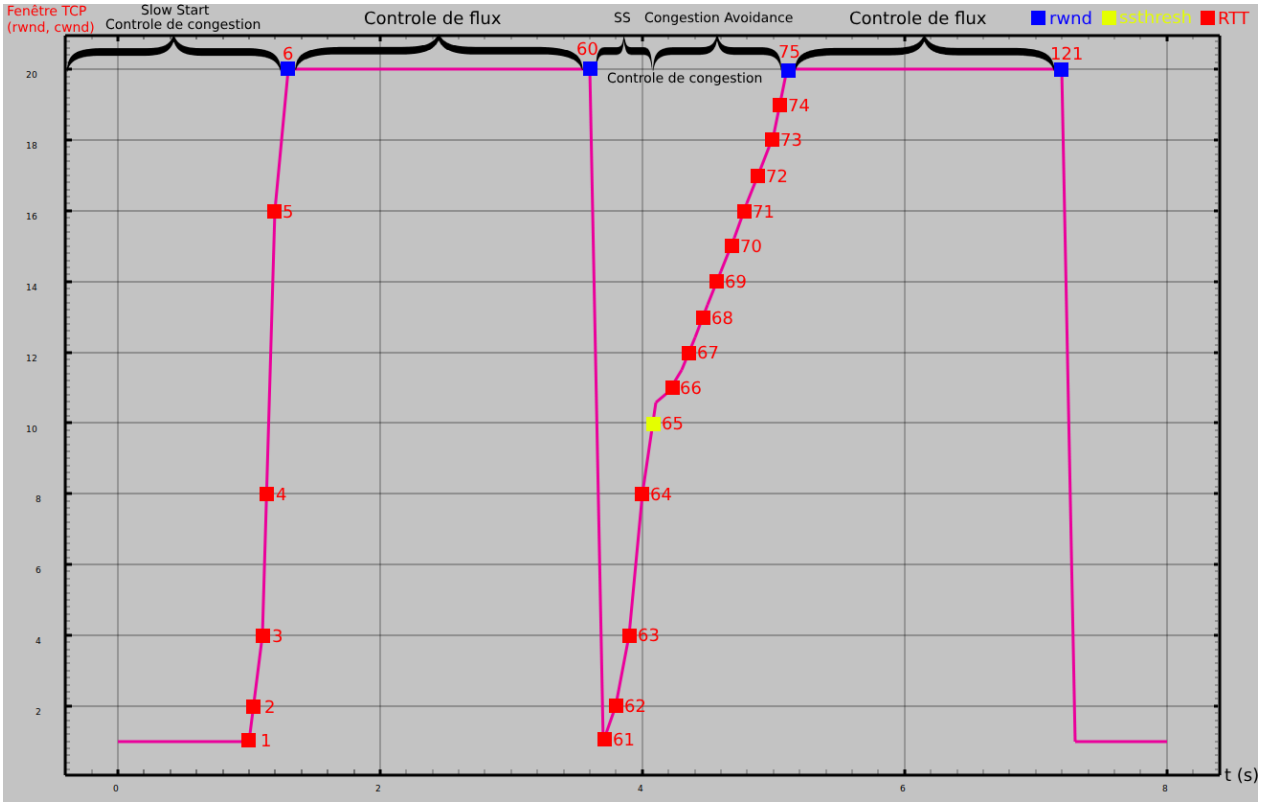


FIGURE 28 – Courbe de performances sous xgraph

Sur le graphique ci-dessus, l'axe des abscisses représente le temps en secondes et l'axe des ordonnées représente la fenêtre TCP en SMSS. Le SMSS est la Sender Maximum Segment Size c'est à dire la taille du plus grand segment que l'émetteur peut transmettre (dépend du RMSS et du MTU). La fenêtre TCP est la valeur minimale entre le cwnd qui est la congestion window, représentant la variable TCP limitant la quantité d'informations que TCP peut envoyer. rwnd est le receiver window c'est la fenêtre de réception la plus récemment annoncée par le récepteur. Si la valeur minimale est le cwnd alors TCP est dans une phase de contrôle de congestion car elle se limite aux capacités du réseau. Si à l'inverse la valeur minimale est le rwnd alors TCP est dans la phase de contrôle de flux car il se limite aux capacités du récepteur. Sur le graphique, les rectangles rouges représentent la "salve" actuellement transmise en fonction du RTT. Le RTT étant le Round-Trip Time qui représente la durée totale nécessaire pour qu'un paquet de données soit envoyé d'un émetteur à un récepteur, puis que l'accusé de réception correspondant soit renvoyé du récepteur à l'émetteur. Ainsi on peut aisément savoir à quel round d'envoi correspond la fenêtre TCP. Les rectangles bleus représentent le rwnd qui est de 20 SMSS comme vu dans la partie **3.3.3**. C'est à dire le moment où la fenêtre d'envoi TCP est limitée par le rwnd. Le rectangle jaune représente le ssthresh pour Slow Start Threshold qui détermine quel algorithme utiliser pour contrôler la transmission des données. Si la valeur de cwnd est inférieure au ssthresh alors l'algorithme utilisé est le Slow Start. Si la valeur du cwnd est supérieure au ssthresh alors l'algorithme utilisé est le Congestion Avoidance.
Passons maintenant à une analyse en détail des performances du réseau. Au démarrage, l'algorithme utilisé est le Slow Start. La transmission des données commence sans avoir connaissance de l'état du réseau. TCP teste donc ces capacités avec un accroissement exponentiel pour déterminer la capacité disponible. Lorsque TCP transmet des données sans avoir connaissances des capacités du réseau il choisit un cwnd inférieur à 2 SMSS. Ici on peut voir sur le graphique que la valeur choisie est 1 SMSS. Durant le slow start, la valeur de cwnd est augmentée de SMSS octets pour chaque ACK reçu acquittant la bonne réception des données c'est à dire qu'à chaque RTT la valeur de cwnd double. Comme il n'y à pas de drop, la valeur de cwnd pour le 2ème RTT est de 2 SMSS.

La valeur de rwnd étant de 20 SMSS, la fenêtre TCP se limite donc à cwnd et envoi donc 2 SMSS. Encore une fois aucune perte n'est constatée ainsi la valeur de cwnd pour le 3ème RTT est de 4 SMSS. TCP se limite donc encore à cwnd. On est donc dans une phase de contrôle de congestion. Ainsi de suite jusqu'au 6ème RTT. Lorsque TCP se prépare pour envoyer le 6ème RTT, la valeur précédente de cwnd était de 16 SMSS. Tous les ACK étant reçus normalement, la valeur de cwnd passe à 32 SMSS. La valeur minimale entre cwnd et rwnd est donc rwnd. On entre donc dans un contrôle de flux car si TCP envoyait plus de 20 SMSS, il saturerait le récepteur et des drops auraient lieu ce qui ralentirait drastiquement le trafic. Du 6ème au 60ème RTT, la fenêtre TCP reste limité à rwnd. Ensuite à cause de la rupture du lien, des pertes sont constatées. Il y a alors deux possibilités. Soit TCP recoit 3 ACK dupliqués demandant l'envoi du même segment, soit il ne reçoit aucun ACK. On est dans le second cas. Ainsi, TCP met donc à jour sa valeur de ssthresh qui est la dernière valeur de fenêtre TCP n'ayant entraînée aucune perte divisée par 2. La dernière valeur étant rwnd = 20 SMSS la valeur de ssthresh devient donc 10 SMSS. Cela signifie que la transmission va reprendre avec l'algorithme de Slow Start tant que cwnd est inférieure à ssthresh pour regagner rapidement des performances. Quand cwnd va dépasser le ssthresh, TCP va utiliser l'algorithme Congestion Avoidance qui sonde plus prudemment le réseau. Lors de cette phase, à chaque RTT, cwnd est incrémenté de 1 SMSS. C'est donc un accroissement additif. Cet algorithme est utilisé jusqu'à la détection de congestion. On voit donc sur la figure ci-dessus que lors de la reprise de la transmission, la valeur de la fenêtre double jusqu'au ssthresh puis la valeur augmente de 1 SMSS jusqu'à arriver une nouvelle fois à rwnd.

3.3.5 Comparaison des pertes entre ce TP et le TP précédent

Dans le TP précédent, on a été témoin de la perte de paquets à cause du buffer. En effet le buffer étant plein au bout d'un certain temps, les paquets sont dropés. Dans ce TP, les paquets sont perdus à cause de la rupture du lien. Ainsi tous les paquets en cours d'acheminement seront dropés tant que le réseau ne réagit pas pour trouver un autre chemin. D'autres raisons possibles créant une perte de paquets sont :

Le TTL. En effet le Time To Live permet au paquets de savoir combien de sauts il peut faire avant d'être dropé. Si le TTL arrive à 0 avant que le paquet arrive à sa destination alors il sera perdu.

Pas de route dans la table de routage. En effet, si le paquet est dirigé vers un autre réseau mais que le routeur n'a pas de route vers ce réseau et que la default gateway ne permet pas de joindre la destination le paquet sera donc dropé.

Erreurs dans les informations. En effet, si lors de la réception des erreurs sont détectées le paquet sera également dropé.

3.4 Conclusion

Dans ce TP, on a appris à mettre en oeuvre différents protocoles de routage pour analyser leurs modes de fonctionnement et comparer leurs performances. On a également appris à créer des ruptures de liens pour analyser le fonctionnement de TCP ainsi que l'impacte sur le réseau. On est aussi capable de nommer les noeuds ainsi que de modifier leurs formes pour faciliter la compréhension de la simulation. Pour finir, on sait désormais utiliser xgraph pour voir les performances du réseau au cours du temps et analyser le fonctionnement des différents protocoles de transport.

4 Rapport Travaux Pratiques 3

4.1 Exercice : 1

4.1.1 But de l'exercice

Le but de cet exercice est de vérifier la bonne compréhension des TP précédents. Il est demandé d'expliquer le code et de déterminer son objectif. Pour cela, plusieurs procédures sont à expliquer en détail. Il est également demandé de modifier le code pour pouvoir visualiser la simulation avec nam. Il faut également colorer les flux ce qui est plus complexe que dans les TP précédents car les flux sont crées dans des fonctions. Il faudra donc modifier la fonction pour colorer les flux de manières automatique.

4.1.2 Explication du code

Comme le but du TP est de comprendre donc de décrire le code, on détaillera chaque ligne même si elles ont déjà été expliqué dans les parties précédentes.

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

Code 50 – Création de l'instance de simulation

Cette ligne de code permet de crée une instance de type Simulator qui sera accessible via la variable ns. Grâce à cette variable, on sera en mesure de gérer les événements de simulation, créer des liens entre les différents noeuds etc...

```
3 #Création du fichier de sortie pour nam
4 set nf [open out.nam w]
5 $ns namtrace-all $nf
```

Code 51 – Création du fichier de sortie pour nam

La première ligne permet de créer ou d'effacer le fichier out.nam puis de l'ouvrir en mode écriture comme spécifié avec le w.

$ns namtrace-all $nf permet d'enregistrer tous les événements de la simulation obtenus en passant par le simulateur via la variable ns et de les écrirent dans le fichier de sortie pointé par la variable nf.

```
6 #Création de plusieurs fichier pour xgraph
7 set f0 [open out0.tr w]
8 set f1 [open out1.tr w]
9 set f2 [open out2.tr w]
```

Code 52 – Création des fichiers de sortie pour xgraph

Ces lignes de code servent à créer trois fichiers de sortie pour xgraph. On peut donc s'attendre à visualiser trois courbes lors de l'exécution du script. Les trois fichiers de sortie crées seront accessibles via les variable f0, f1 et f2. Les trois fichiers de sortie sont ouverts en mode écriture.

```
10 #Création de plusieurs noeuds
11 set n0 [$ns node]
12 set n1 [$ns node]
13 set n2 [$ns node]
14 set n3 [$ns node]
15 set n4 [$ns node]
```

Code 53 – Création des noeuds

Le morceau de code ci-dessus permet de créer cinq noeuds nommé n0, n1, n2, n3, n4 et n5.

```
16 #Création des liens entre les noeuds ayant un debit de 1Mb un temps de transmission
17 #de 100ms et un buffer de type DropTail
18 $ns duplex-link $n0 $n3 1Mb 100ms DropTail
19 $ns duplex-link $n1 $n3 1Mb 100ms DropTail
20 $ns duplex-link $n2 $n3 1Mb 100ms DropTail
21 $ns duplex-link $n3 $n4 1Mb 100ms DropTail
```

Code 54 – Création des différents liens

Tous les liens crées grâce à ces lignes de code possède un débit de 1Mb/s, un temps de propagation de 100ms et un buffer de type DropTail qui signifie que si le buffer est plein alors les paquets qui arrivent seront dropés. En analysant les liens ci-dessus, on peut se rendre compte que les liens n0, n1, et n2 sont reliés à n3. n3 quand à lui est relié à n4. On peut donc envisager que les noeuds n0, n1, et n2 vont communiquer avec n4 et que tous leurs trafic sera centralisé sur n3 ou que n4 communiquera avec n0, n1 et n2 et que son trafic sera dispersé au niveau de n3.

```
22 #Création de la foncction finish
23 proc finish {} {
```

Code 55 – Définition de la procédure finish

Cette ligne est la définition de la procédure finish. Cette procédure ne prend pas de paramètre.

```
24     #Récupere les variables global
25     global f0 f1 f2 nf ns
```

Code 56 – Modification de la portée des variables

Cette ligne permet de modifier la portée des variables f0, f1, f2, nf et ns pour pouvoir les utiliser dans la procédure.

```
26     #Ecrit les données dans le fichier de sortie
27     $ns flush-trace
```

Code 57 – Ecriture dans le fichier de sortie

Cette ligne permet d'écrire toutes les données enregistrées jusqu'à présent dans le fichier de sortie. Ici le fichier de sortie est out.nam car on a spécifié précédemment $ns namtrace-all $nf.

```
28     #Close the output files
29     close $nf
30     close $f0
31     close $f1
32     close $f2
```

Code 58 – Clôture des fichiers de sorties

finish étant la procédure de terminaison du script il faut donc fermer tous les fichiers ouverts durant l'exécution du programme.

```
33      #lance xgraph sur nos 3 fichiers de sortie en ouvrant une fenetre de 800 par 400
34      exec xgraph out0.tr out1.tr out2.tr -geometry 800x400 &
35      exec nam out.nam &
```

Code 59 – Exécution de nam et de xgraph

Ces deux lignes permettent, après avoir récupérées les informations de toutes la simulation à la fin du script, d'exécuter xgraph sur les trois fichiers de sortie correspondant et nam sur le fichier de sortie out.nam. Ceci permet de ne pas taper les commandes à la main à la fin du script. Le & à la fin de la ligne permet de lancer la commande en arrière plan et de ne pas bloquer l'exécution des commandes suivantes.

```
36      #Arrete le programme
37      exit 0
38  }
```

Code 60 – Arrêt du script

Avec le exit 0, le script s'arrête.

```
39  #Declaration de la fonction attach-expoo-traffic qui prend comme parametres
40  # node sink size burst idle rate
41  proc attach-expoo-traffic { node sink size burst idle rate color number} {
```

Code 61 – Déclaration de la procédure attach-expoo-traffic

Cette ligne est la déclaration de la procédure nommé attach-expoo-traffic. Comme son nom l'indique, le but de cette procédure est de créer un trafic exponentiel est de l'attacher à un agent. Le trafic exponentiel est un trafic qui est actif pendant un certain temps puis inactif pendant un certain temps. La procédure prend en paramètres :

-node qui représente le noeud auquel on veut attacher l'agent.

-sink qui représente l'Agent auquel sera attaché l'Agent crée dans la procédure.

-size qui représente la taille des paquets envoyés en octets.

-burst qui représente le temps moyen pendant lequel le trafic est actif.

-idle qui représente le temps moyen pendant lequel le trafic est inactif.

-rate qui représente le débit de transmission des paquets lorsque le trafic est actif.

-color qui représente la couleur que l'on souhaite donner au trafic.

-number qui représente lu numéro de la classe à laquelle on souhaite attribuer la couleur.

```
42      #Utilisation de l'instance ns de type Simulator
43      set ns [Simulator instance]
```

Code 62 – Récupération de l'instance Simulator

Cette ligne de code permet de créer la variable ns à l'intérieur de la fonction mais au lieu de créer une nouvelle instance avec new Simulator, on spécifie que ns est un pointeur vers l'instance Simulator déjà existante. Cela reviens au même que de faire global ns.

```
44      #Creation de l'Agent UDP
45      set source [new Agent/UDP]
```

Code 63 – Création de l'Agent UDP

Cette ligne de code permet de créer un Agent UDP accessible via la variable source.

```
46      #On attache l'Agent au noeud
47      $ns attach-agent $node $source
```

Code 64 – Connexion de l'Agent au noeud

Cette ligne permet d'attacher l'Agent UDP précédemment crée au noeud passé en paramètre lors de l'appel de la fonction.

```
48      #Création d'une nouvelle application de type Exponential
49      set traffic [new Application/Traffic/Exponential]
```

Code 65 – Création de l'Application exponentiel

On crée une Application du type Exponentiel qui sera accessible via la variable traffic.

```
50      #Modification de la taille du paquet
51      $traffic set packetSize_ $size
52      #Modification de burst_time
53      $traffic set burst_time_ $burst
54      #Modification de idle_time
55      $traffic set idle_time_ $idle
56      #Modification du rate
57      $traffic set rate_ $rate
```

Code 66 – Modification des paramètres de l'Application

Grâce aux paramètres fournis lors de l'appel de la procédure on peut modifier les paramètres de l'Application exponentiel. Dans un premier temps, on modifie la taille des paquets en utilisant la variable size. Par la suite on change la durée moyenne durant laquelle le trafic est actif avec la variable burst. On modifie ensuite la durée moyenne durant laquelle le trafic reste inactif avec la variable idle. Enfin on choisit le débit d'émission lorsque le trafic est actif avec la variable rate.

```
58      #On attache l'Application à l'Agent
59      $traffic attach-agent $source
```

Code 67 – Déclaration de la procédure attach-expoo-traffic

Une fois toutes les modifications effectuées, on peut attacher l'Application à l'Agent UDP crée précédemment.

```
60      #On connecte l'Agent UDP à l'Agent du noeud passé en paramètre
61      $ns connect $source $sink
```

Code 68 – Connexion de l'Agent crée à l'Agent passé en paramètre

On peut par la suite attacher l'Agent crée dans la procédure à l'Agent passé en paramètre.

```
62      #Coloration du trafic
63      $ns color $number $color
64      $source set class_ $number
```

Code 69 – Coloration des différents trafics

Grâce aux variables number et color, on pourra modifier la couleur de différentes classes avec différentes valeurs passées en paramètres afin de colorer les trafics de différentes couleurs pour aider à la compréhension de la simulation.

```
65      #On renvoie ce que l'on viens de créer
66      return $traffic
67  }
```

Code 70 – Déclaration de la procédure attach-expoo-traffic

Une fois tout crée et paramétré, on peut utiliser return pour que la procédure renvoi l'Application exponentiel ce qui permettra de créer autant d'Application que l'on veut comme on le verra par la suite.

```
68  #Déclaration de la fonction record
69  proc record {} {
```

Code 71 – Déclaration de la procédure record

La deuxième procédure du script que l'on ne connaît pas est la procédure record.

```
70      #Recupere les variables globales
71      global sink0 sink1 sink2 f0 f1 f2
```

Code 72 – Modification de la portée des variables

Dans la procédure on a besoin de plusieurs variables déclarées en dehors de la fonction. Il faut donc modifier leurs portées pour les rendre accessibles.

```
72      #Utilisation de l'instance ns de type Simulator
73      set ns [Simulator instance]
```

Code 73 – Récupération de l'instance Simulator

Tout comme dans la procédure précédente, on récupère l'instance du simulateur.

```
74      #Déclaration de l'intervalle de temps de maj
75      set time 0.5
```

Code 74 – Déclaration de l'intervalle de temps

La variable time a le même objectif que dans le TP2. Permettre l'enregistrement des données toutes les 0.5 seconde.

```
76      #les variables recuperent la valeurs de bytes_ de $sink
77      set bw0 [$sink0 set bytes_]
78      set bw1 [$sink1 set bytes_]
79      set bw2 [$sink2 set bytes_]
```

Code 75 – Récupération du nombre d'octets reçu

Ces lignes permettent de récupérer le nombre d'octets reçu par le récepteur depuis chaque émetteur.

```
80      #Mise a jour du temps
81      set now [$ns now]
```

Code 76 – Récupération de l'instance Simulator

Cette ligne de code permet de mettre à jour le temps dans la procédure en fonction du temps du simulateur.

```
82      #On met dans les fichiers de sorties les différents débits de communication
83      puts $f0 "$now [expr $bw0/$time*8/1000000]"
84      puts $f1 "$now [expr $bw1/$time*8/1000000]"
85      puts $f2 "$now [expr $bw2/$time*8/1000000]"
```

Code 77 – Récupération de l'instance Simulator

De la même manière que dans le TP2, dans chaque fichier de sortie, on écrit le temps ainsi que le débit. Comment sait on que c'est un débit ? Tout d'abord, les variables bw0, bw1 et bw2 contiennent le nombre d'octets reçu pendant un certain temps. On divise se nombre d'octets par un temps, on est donc en octets par seconde. De plus, on multiplie ce résultat par 8 car 1 octet = 8 bits. On est donc en bits par seconde. Finalement, on divise ce résultat par 1 000 000 pour arriver en Mb/s. Le nombre d'octets étant divisé par le temps, il est important de prendre en compte que c'est un débit moyen et non instantané.

```
86      #Remise a 0 du nombre d'octets des sink
87      $sink0 set bytes_ 0
88      $sink1 set bytes_ 0
89      $sink2 set bytes_ 0
```

Code 78 – Remise à zéro des variables

Une fois les données enregistrées, ils est nécéssaire de le remettre à zéro pour ne pas qu'il y ait des conflits lors de la prochaine écriture ce qui pourrait fausser les données et donc le débit.

```
90      #A chaque mise a jour de temps, la fonction record est appelée
91      $ns at [expr $now+$time] "record"
92  }
```

Code 79 – Appel de la fonction

Grâce à cette ligne de code, la fonction est appelée toutes les time secondes afin de récupérer les valeurs moyennes de débits des trois communications dans les fichiers de sortie.

```
96  #Creation de nouveaux agents de type LossMonitor
97  set sink0 [new Agent/LossMonitor]
98  set sink1 [new Agent/LossMonitor]
99  set sink2 [new Agent/LossMonitor]
```

Code 80 – Création des Applications

Dans la partie du code supérieur, on crée trois Agents de type LossMonitor pour la réception et l'analyse de données.

```
97  #On attache les agents aux noeuds
98  $ns attach-agent $n4 $sink0
99  $ns attach-agent $n4 $sink1
100 $ns attach-agent $n4 $sink2
```

Code 81 – Appel de la fonction

Ensuite, on attache les trois Agents au noeuds n4 pour la réception des trois communications.

```
#Creation de differentes sources en appelant la fonction attach-expoo-traffic avec
    des options
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 Yellow 3]
```

Code 82 – Appel de la fonction

Avec les lignes de code ci-dessus, on crée trois Applications exponentiels grâce à la fonction attach-expoo-traffic. La première Application est pour le noeud 0, la taille des paquets sera de 200 octets, le trafic sera actif pendant 2 secondes puis inactif pendant 1 seconde. Le débit de communication lorsque le trafic est actif sera de 100 kb/s, la couleur des paquets sera bleu. Le noeud n1 aura quant à lui un débit de 200 kb/s et sera rouge. Le trafic provenant du noeud n2, aura un débit de 300 kb/s et sera jaune.

```
#appel de la fonction record
$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"
$ns at 60.0 "finish"
#Executer la simulation
$ns run
```

Code 83 – Evénements de simulation

Les différents événements de simulation sont présents dans la portion de code ci-dessus. Tout d'abord, on appelle la procédure record à t = 0 pour démarrer l'enregistrement des données. Les trois trafics démarrent à 10 secondes et s'arrêtent à t = 50 secondes. Vous trouverez le code dans l'Annexe **6**.

4.2 Visualisation sur nam

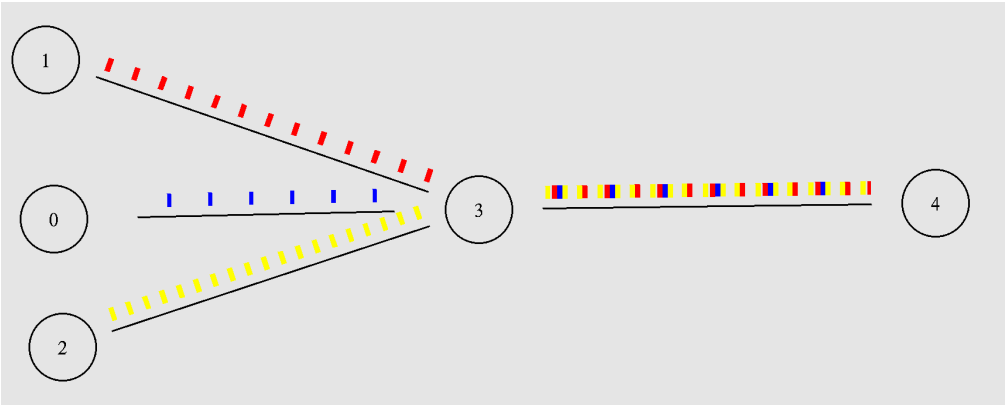


FIGURE 29 – Trafic exponentiel

Comme on peut le voir sur l'image ci-dessus, les différents trafics ont bien une couleur unique.

4.3 Visualisation sous xgraph

Comme indiquer dans l'énoncé, on va observer les courbes de performances sous xgraph quand l'intervalle de temps est de 0.5 seconde, 1 seconde et 0.1 seconde.

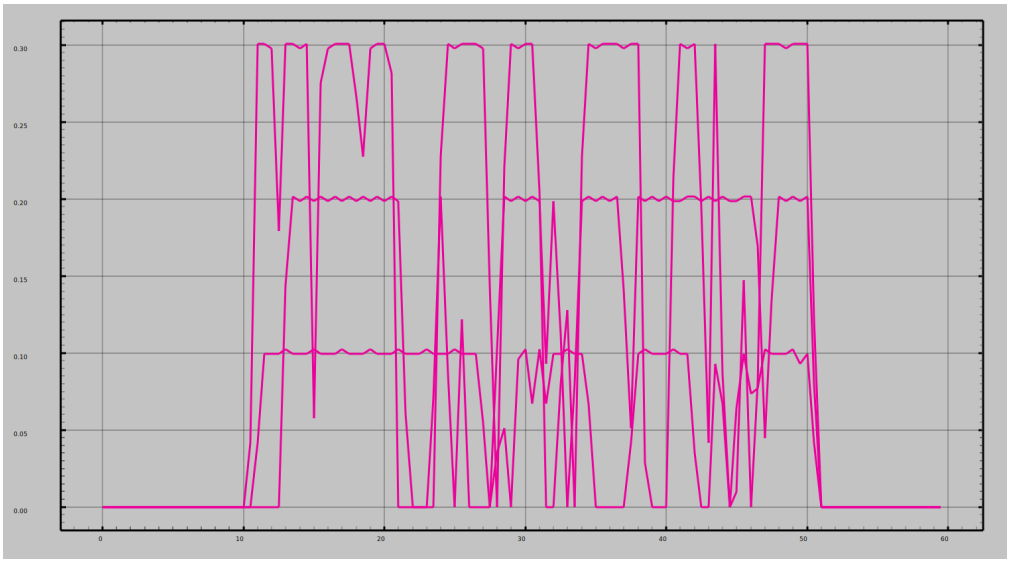


FIGURE 30 – time = 0.5 seconde

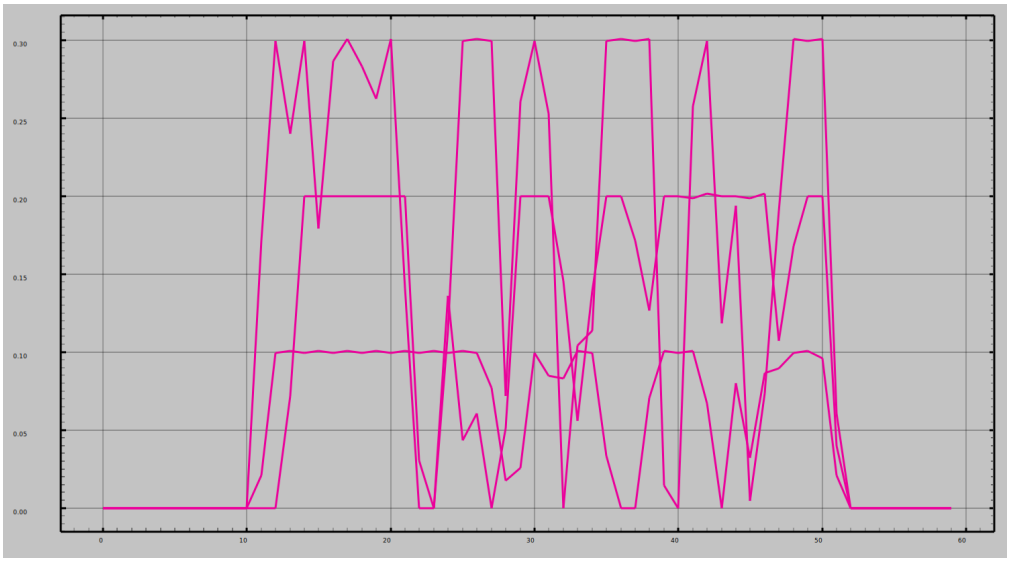


FIGURE 31 – time = 1 seconde

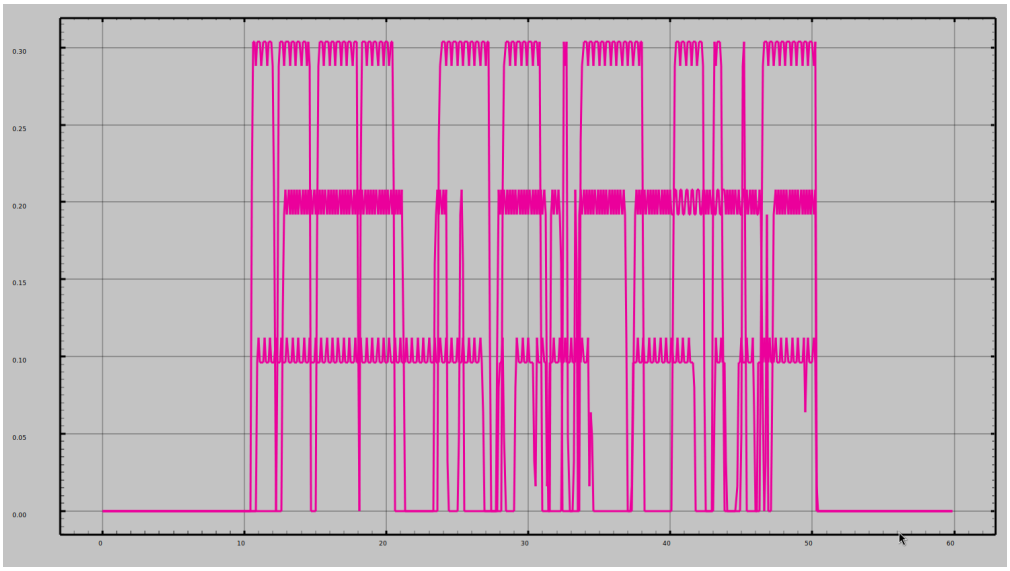


FIGURE 32 – time = 0.1 seconde

Comme on peut le voir sur les images ci-dessus, plus l'intervalle de temps est faible, plus le performances sont précises. Cela s'explique par le fait que les informations sont enregistrées sont le débit moyen. Donc plus le temps est long moins les variations sont précises. Par exemple si l'intervalle de temps est supérieur à la durée de silence de communication alors le débit ne sera jamais mesuré à 0 b/s. Par exemple si l'on prend time = 2 secondes, on obtient la figure ci-dessous :

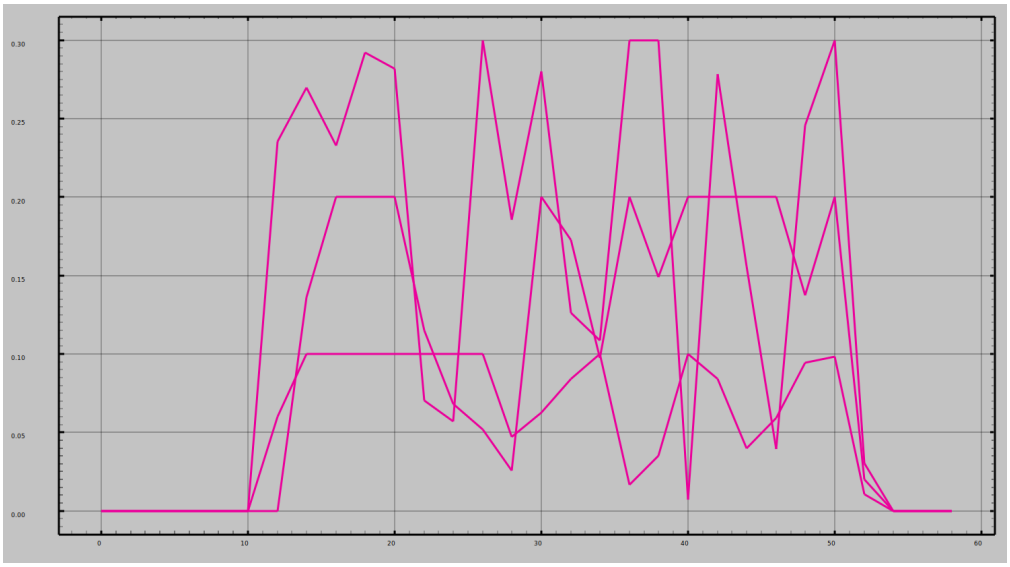


FIGURE 33 – time = 2 secondes

4.4 Conclusion

Dans ce TP, on à donc appris à analyser un script pour déterminer son objectif ainsi que modifier des procédures. On à également vu l'impact de l'intervalle d'enregistrement sur la précision des performances lors de l'exploitation avec xgraph.

5 Conclusion générale

Après tous ces TP on est donc capable de créer nos propres scripts et procédures afin de simuler des réseaux complexes. On peut mettre en place des communications en mode connectés avec des Agents TCP et TCPSink mais aussi des communications temps réels avec des Agents UDP et Null. On sait également mettre en place les différents paramètres des liens reliant deux noeuds comme le débit, le temps de propagation ou encore le type de buffer. Pour faciliter la compréhension des simulations sur nam on a appris à colorer les différents trafics. De plus, on sait comment créer des topologies spécifiques en orientant les noeuds les uns par rapport aux autres. En plus de l'utilisation de nam, on à également appris à analyser les performances d'un réseau avec xgraph. Grâce à cela, on a pu identifier les différentes phases des communications TCP mises en place afin de vérifier la bonne compréhension du cours.

6 Annexe

6.1 Code TP1 Exercice : 1

```
#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
#Définir la procédure de terminaison de la simulation
proc finish {} {
    global ns nf
    $ns flush-trace
    #Fermer le fichier trace
    close $nf
    #2xécuter le nam avec en entrée le fichier trace
    exec nam out.nam &
    exit 0
}
#Création de deux noeuds nommé n(0) et n(1) sous la forme d'un tableau
set n(0) [$ns node]
set n(1) [$ns node]
#Création d'un lien duplex entre n(0) et n(1) de capacité 1Mb,
#10ms de temps de propagation et d'une file d'attente de type
#DropTail
$ns duplex-link $n(0) $n(1) 1Mb 10ms DropTail
#Création d'un agent UDP pour le noeud n(0)
set udp [new Agent/UDP]
$ns attach-agent $n(0) $udp
#Création d'une source de trafic CBR avec une taille des paquets
#de 500 octets et d'intervalle de transmission de paquets de 0.005s
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 500
$cbr set interval_ 0.005
#Connexion de la source CBR à l'agent UDP
$cbr attach-agent $udp
#Création de l'agent NULL pour la réception des paquets sur le noeud n(1)
set null [new Agent/Null]
$ns attach-agent $n(1) $null
#Connexion des deux agents Null et UDP
$ns connect $udp $null
#Coloration de la connexion UDP
$ns color 1 Red
$udp set class_ 1
#Orientation entre les noeuds
$ns duplex-link-op $n(0) $n(1) orient right
#Mise en forme des émetteurs/récepteurs
$n(0) shape box(hexagon)
$n(1) shape box(hexagon)
#Déclenchement du trafic CBR à t=1s puis arrêt à t=4.5s
$ns at 1 "$cbr start"
$ns at 4.5 "$cbr stop"
#Appeler la procédure de terminaison après un temps t
$ns at 5.0 "finish"
#Exécuter la simulation
$ns run
```

Code 84 – Code TP1 Exercice : 1

6.2 Code TP1 Exercice : 2

```
#Creation 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 procedure de terminaison de la simulation
proc finish {} {
    global ns nf
    $ns flush-trace
    #Fermer le fichier trace
    close $nf
    #Executer le nam avec en entrée le fichier trace
    exec nam out.nam &
    exit 0
}
#Création des noeuds
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
#Connexion des différents noeuds
$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
#Fixe la taille du buffer entre le noeud n2 et n3
$ns queue-limit $n2 $n3 10
#Modification de l'emplacement des noeuds
$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
#Modification de la forme des noeuds émetteurs/récepteur
$n0 shape box(hexagon)
$n1 shape box(hexagon)
$n3 shape box(hexagon)
#Création d'un Agent TCP
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
#Création d'un Agent UDP
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
#Création d'un Agent TCPSink pour l'envoi d'ACK
set tcpsink [new Agent/TCPSink]
$ns attach-agent $n3 $tcpsink
#Création de l'Agent Null pour l'Agent UDP
set null [new Agent/Null]
$ns attach-agent $n3 $null
#Création d'une Application FTP
set ftp [new Application/FTP]
$ftp attach-agent $tcp
#Création d'une source de trafic CBR et modification des paramètres
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 1000
$cbr set interval_ 0.008
$cbr attach-agent $udp
#Connexion des différents Agents
$ns connect $tcp $tcpsink
$ns connect $udp $null
```

Code 85 – Code TP1 Exercice : 2

```
58 #Coloration des différents trafics
59 $ns color 1 Blue
60 $tcp set class_ 1
61 $ns color 2 Red
62 $udp set class_ 2
63 #Création des événements de la simulation
64 $ns at 0.1 "$cbr start"
65 $ns at 1 "$ftp start"
66 $ns at 4 "$ftp stop"
67 $ns at 4.5 "$cbr stop"
68 #Appeler la procedure de terminaison apres un temps t
69 $ns at 5.0 "finish"
70 #Executer la simulation
71 $ns run
```

Code 86 – Code TP1 Exercice : 2 suite

6.3 Code TP2 Exercice : 1

```
#Creation 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 procedure de terminaison de la simulation
proc finish {} {
    global ns nf
    $ns flush-trace
    #Fermer le fichier trace
    close $nf
    #Executer le nam avec en entrée le fichier trace
    exec nam out.nam &
    exit 0
}
#Déclaration du protocole de routage LS ou DV
$ns rtproto LS
#Création des noeuds grâce à une boucle
for {set i 1} {$i<=8} {set i [expr $i+1]} {set n($i) [$ns node]}
#Création des différents Agents
set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1
set udp2 [new Agent/UDP]
$ns attach-agent $n(2) $udp2
set null [new Agent/Null]
$ns attach-agent $n(8) $null
#Création des différents liens
$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(4) $n(6) 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(6) $n(7) 10Mb 10ms DropTail
$ns duplex-link $n(7) $n(8) 10Mb 10ms DropTail
#Orientation des différents noeuds
$ns duplex-link-op $n(1) $n(3) orient right-down
$ns duplex-link-op $n(2) $n(3) orient left-down
$ns duplex-link-op $n(3) $n(4) orient right-down
$ns duplex-link-op $n(4) $n(6) orient right
$ns duplex-link-op $n(3) $n(5) orient left-down
$ns duplex-link-op $n(5) $n(8) orient down
$ns duplex-link-op $n(7) $n(8) orient left
$ns duplex-link-op $n(6) $n(7) orient left-down
#Connexion des différents Agents
$ns connect $udp1 $null
$ns connect $udp2 $null
#Création des différentes Applications
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 500
$cbr2 set interval_ 0.005
#Connexion des différentes Applications à leur Agent
```

Code 87 – Code TP2 Exercice : 1

```
$cbr1 attach-agent $udp1
$cbr2 attach-agent $udp2
#Coloration des différents trafics
$ns color 1 Blue
$udp1 set class_ 1
$ns color 2 Red
$udp2 set class_ 2
#Evénements de simulation
$ns at 1 "$cbr1 start"
$ns at 2 "$cbr2 start"
#Rupture du lien puis rétablissement
$ns rtmodel-at 4 down $n(5) $n(8)
$ns rtmodel-at 5 up $n(5) $n(8)
$ns at 6 "$cbr2 stop"
$ns at 7 "$cbr2 stop"
#Appeler la procedure de terminaison apres un temps t
$ns at 8.0 "finish"
#Executer la simulation
$ns run
```

Code 88 – Code TP2 Exercice : 1 suite

6.4 Code TP2 Exercice : 2

```
#Creation 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 procedure de terminaison de la simulation
proc finish {} {
    global ns nf
    $ns flush-trace
    #Fermer le fichier trace
    close $nf
    #Executer le nam avec en entree le fichier trace
    exec nam out.nam &
    exit 0
}
#Mise en place du protocole de routage
$ns rtproto DV
#Boucle pour créer les noeuds
for {set i 1} {$i<=7} {set i [expr $i+1]} {set n($i) [$ns node]}
#Nommer les noeuds et changer la forme
$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)
#Créations des différents liens
$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
#Orientation des différents noeuds
$ns duplex-link-op $n(1) $n(2) orient left-down
$ns duplex-link-op $n(1) $n(3) orient right-down
$ns duplex-link-op $n(2) $n(3) orient right
$ns duplex-link-op $n(2) $n(4) orient right-down
$ns duplex-link-op $n(3) $n(5) orient down
$ns duplex-link-op $n(5) $n(6) orient down
$ns duplex-link-op $n(6) $n(7) orient down
$ns duplex-link-op $n(7) $n(4) orient left-down
#Création et connexion des différents Agents
set tcp [new Agent/TCP]
$ns attach-agent $n(1) $tcp
set tcpsink [new Agent/TCPSink]
$ns attach-agent $n(4) $tcpsink
#Création et connexion de l'Application FTP
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns connect $tcp $tcpsink
#Evenements de simulation
$ns at 1 "$ftp start"
$ns rtmodel-at 3.48 down $n(2) $n(4)
$ns rtmodel-at 4.95 up $n(2) $n(4)
```

Code 89 – Code TP2 Exercice : 2

```
$ns at 7 "$ftp stop"
#Appeler la procédure de terminaison apres un temps t
$ns at 8.0 "finish"
#Executer la simulation
$ns run
```

Code 90 – Code TP2 Exercice : 2 suite

6.5 Code TP2 Exercice Bonus

```
#Création d'un fichier nommé WindowVsTime en écriture
set windowVsTime [open WindowVsTime w]
#Création de la fonction plotWindow avec 2 paramètres
proc plotWindow {tcpSource windowVsTime} {
    #Modification de la portée de la variable
    global ns
    #Création de la variable time
    set time 0.1
    #Affectation de la valeur de cwnd dans la variable cwnd
    set cwnd [$tcpSource set cwnd_]
    #Limitation à rwnd au maximum
    if {$cwnd >= 20} {
        set cwnd 20
    }
    #MAJ du temps
    set now [$ns now]
    #Enregistrement de la valeur de la fenêtre TCP au cours du temps
    puts $windowVsTime "$now $cwnd"
    #Appel de la fonction par elle même toutes les 0.1 seconde
    $ns at [expr $now+$time] "plotWindow $tcpSource $windowVsTime"
}
#Appel initial de la fonction avec paramètres
$ns at 0 "plotWindow $tcp $windowVsTime"
```

Code 91 – Code TP2 Exercice : bonus

6.6 Code TP3 Exercice : 1

```
#Création d'une instance de l'objet Simulator
set ns [new Simulator]
#Création du fichier de sortie pour nam
set nf [open out.nam w]
$ns namtrace-all $nf
#Création de plusieurs fichier pour xgraph
set f0 [open out0.tr w]
set f1 [open out1.tr w]
set f2 [open out2.tr w]
#Création de plusieurs noeuds
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
#Création des liens entre les noeuds ayant un debit de 1Mb un temps de transmission
#de 100ms et un buffer de type DropTail
$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
#Création de la foncction finish
proc finish {} {
    #Récupere les variables global
    global f0 f1 f2 nf ns
    #Ecrit les données dans le fichier de sortie
    $ns flush-trace
    #Close the output files
    close $nf
    close $f0
    close $f1
    close $f2
    #lance xgraph sur nos 3 fichiers de sortie en ouvrant une fenetre de 800 par 400
    exec xgraph out0.tr out1.tr out2.tr -geometry 800x400 &
    exec nam out.nam &
    #Arrete le programme
    exit 0
}
#Declaration de la fonction attach-expoo-traffic qui prend comme parametres
# node sink size burst idle rate
proc attach-expoo-traffic { node sink size burst idle rate color number} {
    #Utilisation de l'instance ns de type Simulator
    set ns [Simulator instance]
    #Creation de l'Agent UDP
    set source [new Agent/UDP]
    #On attache l'Agent au noeud
    $ns attach-agent $node $source
    #Création d'une nouvelle application de type Exponential
    set traffic [new Application/Traffic/Exponential]
    #Modification de la taille du paquet
    $traffic set packetSize_ $size
    #Modification de burst_time
    $traffic set burst_time_ $burst
    #Modification de idle_time
    $traffic set idle_time_ $idle
    #Modification du rate
    $traffic set rate_ $rate
```

Code 92 – Code TP3 Exercice : 1

```
58      #On attache l'Application à l'Agent
59      $traffic attach-agent $source
60      #On connecte l'Agent UDP à l'Agent du noeud passé en paramètre
61      $ns connect $source $sink
62      #Coloration du trafic
63      $ns color $number $color
64      $source set class_ $number
65      #On renvoie ce que l'on viens de créer
66      return $traffic
67  }
68  #Déclaration de la fonction record
69  proc record {} {
70      #Recupere les variables globales
71      global sink0 sink1 sink2 f0 f1 f2
72      #Utilisation de l'instance ns de type Simulator
73      set ns [Simulator instance]
74      #Déclaration de l'intervalle de temps de maj
75      set time 0.5
76      #les variables recuperent la valeurs de bytes_ de $sink
77      set bw0 [$sink0 set bytes_]
78      set bw1 [$sink1 set bytes_]
79      set bw2 [$sink2 set bytes_]
80      #Mise a jour du temps
81      set now [$ns now]
82      #On met dans les fichiers de sorties les différents débits de communication
83      puts $f0 "$now [expr $bw0/$time*8/1000000]"
84      puts $f1 "$now [expr $bw1/$time*8/1000000]"
85      puts $f2 "$now [expr $bw2/$time*8/1000000]"
86      #Remise a 0 du nombre d'octets des sink
87      $sink0 set bytes_ 0
88      $sink1 set bytes_ 0
89      $sink2 set bytes_ 0
90      #A chaque mise a jour de temps, la fonction record est appelée
91      $ns at [expr $now+$time] "record"
92  }
93  #Creation de nouveaux agents de type LossMonitor
94  set sink0 [new Agent/LossMonitor]
95  set sink1 [new Agent/LossMonitor]
96  set sink2 [new Agent/LossMonitor]
97  #On attache les agents aux noeuds
98  $ns attach-agent $n4 $sink0
99  $ns attach-agent $n4 $sink1
100 $ns attach-agent $n4 $sink2
101 #Creation de differentes sources en appelant la fonction attach-expoo-traffic avec
        des options
102 set source0 [attach-expoo-traffic $n0 $sink0 200 2s 1s 100k Blue 1]
103 set source1 [attach-expoo-traffic $n1 $sink1 200 2s 1s 200k Red  2]
104 set source2 [attach-expoo-traffic $n2 $sink2 200 2s 1s 300k Yellow 3]
105 #appel de la fonction record
106 $ns at 0.0 "record"
107 #Start the traffic sources
108 $ns at 10.0 "$source0 start"
109 $ns at 10.0 "$source1 start"
110 $ns at 10.0 "$source2 start"
111 $ns at 50.0 "$source0 stop"
112 $ns at 50.0 "$source1 stop"
113 $ns at 50.0 "$source2 stop"
114 $ns at 60.0 "finish"
```

Code 93 – Code TP3 Exercice : 1 suite

```
115 #Executer la simulation
116 $ns run
```

Code 94 – Code TP3 Exercice : 1 suite