# Table des matières

# Liste des figures

# Liste des acronymes

ACK: Acknowledgment

AIMD: Additive Increase/Multiplicative-Decrease

CBQ: Class Based Queuing

CBR: Constant Bit Rate

CSMA/CA: Carrier Sense Multiple Access with Collision Avoidance

CSMA/CD: Carrier Sense Multiple Access with Collision Detection

DRR: Deficit Round Robin

DVMRP: Distance Vector Multicast Routing Protocol

EC: efficiency Controller

ECN: Explicit Congestion Notification

FC: Fairness Controller

FIFO: First In First Out

FTP: File Transfer Protocol

HTTP:Hyper Text Transfer Protocol

IETF: Internet Engineering Task Force

IP: Internet Protocol

LAN: Local Area Network

MAN: Metropolitan Area Network

NAM: Network AniMator

NS: Network Simulator

PIM: Protocol Independent Multicast

RED: Random Early Detection

RFC: requests For Comments

RTP: Real time Transport Protocol

RTT: Round Trip Time

SFQ: Stochastic Fairness Queuing

SRM: Scalable Reliable Multicast

TK: Tool Kit

TCL: Tool Command Language

TCP: Transfer Control Protocol

UDP: User Datagram Protocol

WAN: Wide Area Network

XCP: eXplicit Congestion Protocol

# Introduction générale

De nos jours, les réseaux véhiculent simultanément un nombre important de données hétérogènes. Ainsi, lors de la circulation de plusieurs flux, les caractéristiques du réseau engendrent des aspects négatifs sur la transmission de ces informations. Ces aspects peuvent se traduire par des délais d'acheminement importants, une incapacité d'obtenir une bande passante suffisante ou encore par la perte des paquets.

Pour remédier à ces problèmes, des mécanismes de gestion de file d'attente sont implémentés dans les routeurs, ce qui permet d'administrer les paquets plus ou moins d'une manière équitable et apporter des solutions au problème de congestion qui se produit lorsque le flux dans le réseau est supérieur aux capacités des ressources disponibles sur ce même réseau.

Dans le cadre de notre projet de modélisation de flux dans un réseau qui est la conception d'un modèle qui représente la réalité, nous allons simuler plusieurs types de files d'attente et cela afin d'étudier et évaluer les modalités de leurs fonctionnements, en utilisant le simulateur de réseau NS2 (Network simulator 2).

Après une présentation générale des différents types de réseaux, de topologies et la simulation, nous allons entamer une étude approfondie de notre problématique qui est la gestion des files d'attente.

En effet, un réseau se comporte d'un ensemble de files d'attente qui interviennent entre autres en commutation[1], routage[2], réception des paquets[3] …etc.

[1] Commutation est une technique fondée sur le découpage des données afin de les transférer sur le réseau vers sa destination.
[2] Routage est le mécanisme d'acheminement de données dans un réseau.
[3] Paquet est une unité de transmission utilisée pour communiquer, en réseau.

# Chapitre I Généralité

# I- Introduction

Le réseau est un système de communication qui relie des ordinateurs et des équipements informatiques dans un espace géographique défini. Il sert à la transmission des données (fichiers, message, etc.), au partage de données (logiciel et base de données) et au partage de matériels (imprimantes, modem, etc.).

Dans ce qui suit nous allons voir les différents types de réseaux et de topologies qui existent. Vue l'importance de la simulation et le rôle majeur qu'elle joue dans notre étude nous allons l'introduire dans ce chapitre en soulignant ses différents avantages et inconvénients.

# II- Les types de réseaux et topologies

## 1- Les types de réseaux

Un réseau informatique est un ensemble de machines connectées entre elles qui échangent des informations, il permet la communication entre ordinateurs, le partage de fichiers, d'imprimantes et différents services. [14]

On distingue différents types de réseaux selon leur taille, leur vitesse de transfert et leur étendue. Généralement il y a trois catégories de réseaux :

- LAN (local area network)
- MAN (metropolitan area network)
- WAN (wide area)

On peut distinguer deux modes de fonctionnement en réseau :

- Dans un environnement d'"égal à égal" (en anglais peer to peer), dans lequel il n'y a pas d'ordinateur central et chaque ordinateur a un rôle similaire.

- Dans un environnement "client/serveur", dans lequel un ordinateur central fournit des services réseau aux utilisateurs.

### 1.1- Les LAN

LAN signifie Local Area Network (en français Réseau Local). Il s'agit d'un ensemble d'ordinateurs reliés entre eux dans une petite aire géographique par un réseau, souvent à l'aide d'une même technologie (la plus répandue étant Ethernet).

### 1.2- Les MAN

Les MAN (Metropolitan Area Network) interconnectent plusieurs LAN géographiquement proches (au maximum quelques dizaines de km) à des débits importants. Ainsi un MAN permet à deux nœuds distants de communiquer comme si ils faisaient partie d'un même réseau local.

Un MAN est formé de commutateurs ou de routeurs interconnectés par des liens hauts débits (en général en fibre optique).

### 1.3- Les WAN

Un WAN (Wide Area Network ou réseau étendu) interconnecte plusieurs LANs à travers de grandes distances géographiques.

Les débits disponibles sur un WAN résultent d'un arbitrage avec le coût des liaisons (qui augmente avec la distance) et peuvent être faibles.

Les WAN fonctionnent grâce à des routeurs qui permettent de "choisir" le trajet le plus approprié pour atteindre un nœuds du réseau. Le plus connu des WAN est Internet. [15]

## 2- Les topologies

Les dispositifs matériels mis en œuvre ne sont pas suffisants à l'utilisation du réseau local. En effet, il est nécessaire de définir une méthode d'accès standard entre les ordinateurs, afin que ceux-ci connaissent la manière de laquelle les ordinateurs échangent les informations, notamment dans le cas où plus de deux ordinateurs se partagent le support physique. Cette méthode d'accès est appelée topologie logique. La topologie logique est réalisée par un protocole d'accès. Les protocoles d'accès les plus utilisés sont : Ethernet[4] et Token ring[5]. [17]

La façon de laquelle les ordinateurs sont interconnectés physiquement est appelée topologie physique. Les topologies physiques basiques sont :

[4] Ethernet est un protocole de réseau local à commutation de paquets.
[5] Token ring est un protocole de réseau local qui fonctionne sur les couches physique et liaison du modèle OSI.

## 2.1- La topologie en bus

Une topologie en bus désigne le fait que lors de l'émission de données sur le bus par une station de travail, l'ensemble des stations de travail connectées sur le bus la reçoivent. Seule la station de travail à qui le message est destiné la recopie.

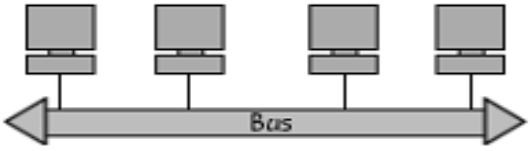



**Figure 1 : Topologie en bus**

## 2.2- La topologie en étoile

L'ensemble des stations de travail est connecté à un Hub qui examine le contenu du message, qui le régénère, et qui le transmet qu'à son destinataire. C'est en réalité un réseau de "n" liaisons point par point, car il établit un circuit entre une paire d'utilisateurs.

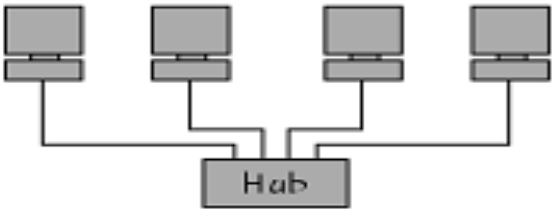



**Figure 2 : Topologie en étoile**

## 2.3- La topologie en anneau

L'information circule le long de l'anneau dans un seul sens. A chaque passage d'un message au niveau d'une station de travail, celle-ci regarde si le message lui est destiné, si c'est le cas elle le recopie.

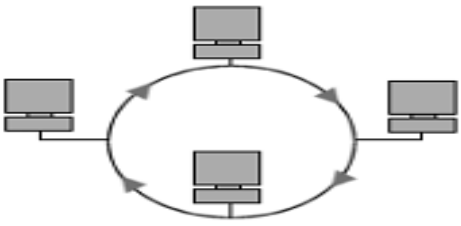


**Figure 3 : Topologie en anneau**

# III- Simulation

La simulation est l'un des outils permettant de simuler des phénomènes réels.

Les objectifs principaux de la simulation sont l'évaluation des protocoles, d'architecture des réseaux et prévoir leur fonctionnement.

Il existe une multitude de simulateurs tel que : OPNET, QNAP, SimuLog, OMNET++ et NS.

Parmi ces simulateurs nous utilisons NS car c'est un outil libre, plus utilisé et que nous avons déjà apprit à travailler avec.

Le logiciel NS-2 et son outil de visualisation NAM sont particulièrement adaptés à l'étude de réseaux complexes (filaires, sans fils) mettant en œuvre de nombreuses files d'attente, procédés de routage et types de trafics.

**Intérêt de la simulation**

- Quand on ne peut pas facilement observer les états du système,
- Quand on désire analyser l'enchaînement des événements dans le système, ainsi que les relations de causes à effets,
- Quand on désire valider une solution analytique,
- Quand la complexité des interactions dans le système est telle qu'elle ne peut être étudiée qu'au travers de simulations,
- Quand on désire visualiser les états d'un système,
- Quand on veut tester différentes optimisations pour améliorer un système déjà existant.

## Avantages de la simulation

- Observations des états du système,
- Etudes des points de fonctionnement d'un système,
- Etudes de systèmes à échelle de temps variable,
- Etudes de l'impact des variables sur les performances du système,
- Etude d'un système sans les contraintes matérielles.

## Inconvénients

- La conception de modèles peut nécessiter des compétences spéciales,
- Une autre forme d'analyse plus proche de la réalité est peut être nécessaire,
- Résultats difficilement interprétables,
- Résultats pas forcément généralisables,
- Résultats sont fonction des entrées du système.

# IV- Network Simulator NS

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

Le simulateur NS actuel est particulièrement bien adapté aux réseaux à commutation de paquets et à la réalisation de simulations de petite taille. Il contient les fonctionnalités nécessaires à l'étude des algorithmes de routage uni point ou multipoint, des protocoles de transport, de session, de réservation, des services intégrés, des protocoles d'application comme HTTP. De plus le simulateur possède déjà une palette de systèmes de transmission (couche 1 de l'architecture TCP/IP) d'ordonnanceurs et de politiques de gestion de files d'attente pour effectuer des études de contrôle de congestion. La liste des principaux composants actuellement disponible dans NS par catégorie est:

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

Prises ensembles, ces capacités ouvrent le champ à l'étude de nouveaux mécanismes au niveau des différentes couches de l'architecture réseau. NS est devenu l'outil de référence pour les chercheurs du domaine. Ils peuvent ainsi partager leurs efforts et échanger leurs

résultats de simulations. Cette façon de faire se concrétise aujourd'hui par l'envoi dans certaines listes de diffusion électronique de scripts de simulations NS pour illustration. [12]

# V- Conclusion

La plupart des recherches utilisent la modélisation et la simulation. Les finalités sont diverses : le modèle peut être utilisé à des fins de contrôle et de prédiction ou à des fins de compréhension et de confrontation de points de vue.

Notre objectif est de même car nous allons utiliser la simulation afin de montrer les différents phénomènes engendrés dans un réseau et cela au niveau des files d'attente qui sont responsables de la bonne circulation des données (paquets) dans ce réseau.

En résumé dans ce chapitre nous avons défini les différents types de réseau tel que LAN, MAN et WAN et aussi les topologies associées à ces derniers. Ensuite nous avons présenté ce que c'est que la simulation avec ses différents avantages et inconvénients. Ensuite nous avons introduit Le simulateur NS (Network Simulator) que nous allons utiliser pour la partie simulation grâce aux résultats de ces simulations nous allons pouvoir effectuer notre étude.

# Chapitre II

# Gestion des files d'attente : Etat de l'art

# I- Introduction

La gestion de file d'attente se préoccupe de la gestion d'acheminement, de retardement ou d'élimination des paquets appartenant à une même classe de flux.

Dans ce chapitre nous allons voir les notions élémentaires de la théorie des files d'attentes: notion de Kendall, processus des arrivées et des services, des files markoviennes (M/M/1, file à capacité limitée) ainsi que le phénomène de congestion.

# II- Les files d'attente

## 1- Définition

Une file d'attente est une structure qui stocke de manière ordonnée des éléments, mais rend accessible uniquement un seul d'entre eux, appelé la tête de la file. Quant on ajoute un élément, celui-ci devient le dernier élément qui sera accessible. Quant on retire un élément de la file, on retire toujours la tête, celle-ci étant le premier élément qui a été placé dans la file. Pour résumer, le premier élément ajouté dans la file est le premier élément à en être retiré. Cette structure est également appelée une liste FIFO (*First In, First Out*). [16]

## 2- Théories des files d'attente [13]

### 2.1- Processus des Arrivées

On observe les arrivées de clients à l'entrée du système. Pour décrire le phénomène, la première idée venant à l'esprit sera d'utiliser l'intervalle du temps entre arrivées successives, ou bien le nombre des arrivées dans un intervalle donné.

Pendant la durée T, n(T) arrivées se produisent. On chiffre le volume du flux arrivant par le taux d'arrivée dont la définition intuitive est:

$$\lambda \equiv \lim_{T\to\infty} \frac{n(T)}{T}$$

- **Processus de renouvellement**

Dans le cas le plus favorable, la description statistique du temps entre les arrivées est très simple. Supposons que la situation puisse être décrite de la façon suivante:

Chaque fois qu'une arrivée se produit, je tire au sort, selon une loi donnée, l'intervalle jusqu'à la prochaine arrivée, de telle sorte que les intervalles successifs soient indépendants.

On est alors dans le cas très particulier d'un processus de renouvellement.

- **Processus de Poisson**

Supposons que le processus des arrivées obéisse aux règles suivantes:

- La probabilité d'une arrivée dans un intervalle [t, t+Dt[ne dépend pas de ce qui s'est passé avant l'instant t. C'est la propriété dite ``sans mémoire".
- La probabilité d'apparition d'un client est proportionnelle à Dt, la probabilité de plus d'un événement étant ``négligeable" (infiniment petit d'ordre supérieur). Le coefficient de proportionnalité est noté l (intensité du processus).

## 2.2- Processus des Services

Le processus de service pourra être d'une complexité extrême, mais on se borne le plus souvent à supposer que chaque durée de service est indépendante des autres, et qu'elles obéissent toutes à une même loi de distribution: on parle de variables indépendantes et identiquement distribuées (i.i.d.). On décrira cette loi par sa distribution de probabilité :

$$B(x) = P\ \{\text{temps de service } \leq x\}$$

- **La loi exponentielle**

La loi de service la plus populaire est la loi exponentielle, qu'il est traditionnel d'écrire en utilisant comme 'taux de service' la lettre μ :

$$B(x) = P\ (\text{service} \leq x) = 1 - e^{-\mu x}$$

La densité correspondante est $b(x) = \mu e^{-\mu x}$

La loi exponentielle doit une bonne partie de son prestige à la propriété 'sans mémoire'.

La loi exponentielle se caractérise par ses moments :

- Moyenne de la variable : $1/\mu$

- Variance de la variable : $1/(\mu^2)$

- **La loi d'Erlang**

Supposons que le processus de service soit composé d'une cascade de k serveurs élémentaires exponentiels, identiques (c'est à dire, de même paramètre μ), et indépendants les uns des autres. Le temps du service est la somme des temps passés dans chaque serveur.

Supposons k =2. Notons X le temps total, $X_1$ et $X_2$ les durées des deux temps services, la distribution B de X est donnée par la convolution de $B_1$ et $B_2$ :

$$P\{X \leq x\} = P\{X_1 + X_2 \leq x\} = \int_{u=0}^{x} B_1(x-u)dB_2(u)$$

Evidemment, $B_1$ et $B_2$ sont identiques, et correspondent à l'exponentielle.

$$P\{X \leq x\} = \int_{u=0}^{x} \left[1 - e^{-\mu(x-u)}\right]\mu e^{-\mu x} du = 1 - e^{-\mu x} - \mu x e^{-\mu x}$$

Plus généralement, la mise en cascade de k serveurs exponentiels de même paramètre μ conduit à une distribution :

$$B(x) = P\{X_1 + X_2 + \cdots + X_k \leq x\} = 1 - \sum_{i=0}^{k} \frac{(\mu x)^j}{j!} e^{-\mu x}$$

Puisqu'il s'agit d'une somme de variables aléatoires indépendantes, moyenne et variance s'obtiennent facilement, comme la somme de la moyenne et de la variance de chaque variable exponentielle :

- Moyenne de la variable : k/μ.

- Variance de la variable : k/ ($\mu^2$).

- Le coefficient de variation est $1/\sqrt{k}$.

## 2.3- Chaînes de Markov

Une chaîne de Markov est une suite de variables aléatoires ($X_n$, n ∈ N) qui permet de modéliser l'évolution dynamique d'un système aléatoire : $X_n$ représente l'état du système à l'instant n. La propriété fondamentale des chaînes de Markov, dite propriété de Markov, est que son évolution future ne dépend du passé qu'au travers de sa valeur actuelle.

Considérons une variable aléatoire X, à espace d'état discret (X ne prend ses valeurs que dans un espace discret. Pour se fixer les idées, X peut représenter le nombre des clients dans un tampon: X =0, 1, 2,…). On peut noter les états $E_1$, $E_2$, …, $E_k$. Notons $t_1$, $t_2$,…, $t_n$ les instants successifs des changements d'état de X, et $x_1$, …, $x_n$ la suite des états visités. Les $t_n$ pourraient être, par exemple, les instants d'arrivée et de départ dans le tampon. Dans le cas général, l'étude de l'évolution de X est très difficile. Il existe une circonstance capitale, qui est la base de tous les développements qui suivent: Dans le cas général, la valeur de X à l'instant $t_{n+1}$ dépend de tout le passé, c'est à dire de X ($t_1$), X ($t_2$), etc. On dira que X obéit à la propriété de Markov si :

$$P[X\,(t_{n+1}) = x_{n+1}\,|T(t_n) = \ x_n\,, X\,(t_{n-1}) = x_{n-1}\,, \dots\ X\,(t_1) = x_1]$$
$$= \ \ P[X\,(t_{n+1}) = x_{n+1}\,|T(t_n) = \ x_n\,]$$

## 2.4- La Notation de Kendall

La notation de Kendall permet de décrire le système à files d'attente par une suite de paramètres.

Pour identifier un système à files d'attentes, le formalisme suivant a été proposé et est unanimement adopté:

A/B/n/K/N

La première lettre identifie la loi du processus des arrivées, la seconde le processus des services, avec dans les deux cas les conventions:

M : loi ``sans mémoire" (arrivées poissonniennes, service exponentiel);

D : loi déterministe;

Ek : loi ``Erlang-k"

Hk : loi ``hyper exponentielle" d'ordre k;

GI : loi générale, les variables successives étant indépendantes;

G : loi générale, sans hypothèse d'indépendance.

Le chiffre qui suit donne le nombre de serveurs, les lettres qui suivent (facultatives) identifient la taille de la file d'attente et la taille de la population source (si ces valeurs sont omises, elles sont supposées infinies).

Par exemple, M/M/1 fait référence au modèle de base: arrivées poissonniennes, service exponentiel, un seul serveur, file d'attente non bornée. M/D/K/K indique un service de durée constante, K serveurs et K places au total (c'est à dire pas d'attente: modèle d'Erlang).

Dans une file d'attente, un paramètre de première importance est le taux d'utilisation, noté traditionnellement ρ. C'est le produit du taux d'arrivée des clients par le temps de service: ρ = l×E(s). On peut montrer que 1- ρ donne la probabilité de trouver le serveur inactif. Il faut (pour une file de capacité infinie) que ρ < 1 (condition de stabilité).

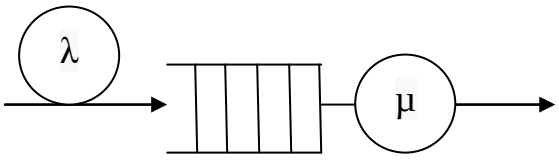
## 2.5- File d'attente M/M/1 [2]

Une file d'attente M/M/1 est un système formé d'une file de capacité infinie et d'un unique serveur. Les clients arrivent dans le système selon un processus de Poisson de taux λ (eq. 1) sachant que λ est le temps d'inter-arrivée, A(t) est une variable aléatoire de distribution exponentielle de paramètre λ (eq. 2). Le temps de service est distribué selon une loi exponentielle de paramètre μ sachant que μ est la taille moyenne d'un paquet.

$P_k(t) = \frac{(\lambda t)^k}{k!} e^{-\lambda t}$ Avec $P_k(t)$ : probabilité d'avoir k clients qui soient arrivés pendant t) (eq. 1)

$A(t) = 1 - e^{-\lambda t}$ Avec A(t) : probabilité d'une arrivée pendant t (eq. 2)

L'ordre de service est FIFO. Une file d'attente M/M/1 se représente selon la figure suivante :




Clients Services

**Figure 4 : File d'attente M/M/1**

# 3- Congestion

Malgré tous les efforts pour contrôler le flux d'information dans un réseau, celui-ci peut se trouver face à un problème de congestion. Il s'agit alors de résoudre le problème sans l'aggraver. En effet, les problèmes de congestion arrivent lorsque les nœuds d'un réseau saturent leurs files d'attente et donc perdent des paquets.

La congestion est statiquement inévitable sur un réseau. Elle résulte de l'augmentation des délais d'acheminement. Si la taille des files d'attente augmente dans les nœuds, les blocs ne sont pas acheminés dans les délais et sont donc retransmis, ce qui augmente encore le flux. [7]

La congestion se produit quand le flux (charge) en réseau est supérieur aux capacités des ressources. Parmi les causes majeures de congestion en réseau sont :

- Bande passante faible.
- Performances des processeurs (mise en file d'attente, mise à jour des tables de routage, retransmission).
- Files d'attente des lignes d'entrées, leur taille finie implique la perte de paquets.

# 4- Contrôle de congestion

L'objectif du contrôle de congestion est d'essayer de garantir une bonne qualité de service pour les usagers. Lors d'un problème de congestion (charge du nœud trop élevé), les mémoires tampon des nœuds sont saturés, les nouveaux paquets reçus sont perdus entrainant des retransmissions ; d'où la nécessité d'un contrôle de congestion. [6]

## 4.1- Explicit Congestion Notification ECN

L'Explicit Congestion Notification ou ECN est une addition à la pile TCP/IP conçue en 1994 et standardisée en septembre 2001 par l'IETF sous la RFC 3168 [3].

C'est un mécanisme qui permet d'anticiper la congestion d'un routeur. ECN est seulement employé lorsque les deux protagonistes (source et destination) signalent qu'ils sont d'accord pour employer cette extension. Avec cette méthode, un drapeau ECN est placé dans l'en-tête IP pour informer explicitement l'émetteur que le routeur est congestionné.ECN est une alternative à la notification indirecte de congestion faite par l'effacement d'un paquet exécuté par RED. [10]

## 4.2- eXplicit Congestion control Protocol XCP

XCP est un protocole de contrôle de congestion, suggéré par Mark Hadley et développé par Dina Katabi. Ce protocole peut être appliqué à n'importe quel protocole de transport.

Il est performant particulièrement dans les grands réseaux à large bande. L'efficacité du TCP diminue en cas de multiple perte de paquets et devient instable indépendamment du système de file d'attente utilisé.

Cependant en environnement similaires, XCP en utilisant une théorie de contrôle basée sur un modèle de rétroaction (feedback) réalise beaucoup plus d'efficacité, de stabilité et d'équité en envoyant la rétroaction du réseau à l'expéditeur pour ajuster le flux de données en réseau. [11]

- **Fonctionnement du XCP**

XCP [5] (eXplicit Control Protocol) est un protocole qui utilise l'assistance des routeurs pour informer précisément l'émetteur des conditions de congestion du réseau. Les paquets de données XCP comportent un en-tête de congestion, rempli par l'émetteur, qui contient la taille actuelle de la fenêtre de congestion de celui-ci (le champ H_cwnd), l'estimation du temps aller-retour (le champ rtt) et une valeur appelée feedback (le champ H_feedback), qui indique à l'émetteur un incrément (si elle est positive) ou un décrément (si elle est négative) à appliquer à sa fenêtre de congestion. Le champ H_feedback est le seul qui peut être modifié par les routeurs XCP en fonction des valeurs des 2 autres champs. Quand le récepteur reçoit un paquet de données, il recopie l'en-tête du paquet dans l'en-tête d'un paquet d'accusé de réception (ACK) qui sera envoyé vers l'émetteur.

A la réception de l'ACK, l'émetteur mettra à jour la taille de sa fenêtre de congestion de la manière suivante : cwnd = max (cwnd + H feedback; packetsize), où cwnd est exprimé en octets. Le mécanisme central d'un routeur XCP est basé sur l'utilisation d'un contrôleur d'efficacité (EC) et d'un contrôleur d'équité (FC) qui réalisent la mise à jour de feedback pendant un intervalle de contrôle équivalent à la moyenne des RTT. L'EC a la responsabilité de maximiser l'utilisation de la bande passante tout en minimisant le nombre de paquets rejetés et le FC de partager les ressources de façon équitable. Il va assigner à feedback une valeur proportionnelle à la bande passante disponible (S), déduite de la différence entre le trafic entrant total et la capacité du lien de sortie. La taille résiduelle Q de la file d'attente est également prise en compte. Dans une deuxième étape, le FC traduit cette valeur feedback globale (qui peut être assimilée à une valeur agrégée de bande passante disponible positive ou

négative) en une valeur feedback par paquet (qui sera ensuite placée dans l'en-tête de chaque paquet de données) en suivant des règles d'équité par flux similaires aux règles AIMD[6] de TCP. Il faut noter qu'il n'y a pas d'états par flux conservés par le routeur XCP pour exécuter toutes ces opérations. En effet, comme les paquets de données d'un flux donné portent dans leur en-tête la valeur actuelle de la fenêtre de congestion et le RTT, il est possible de calculer pour chaque flux le nombre de paquets envoyés par fenêtre de congestion afin d'assigner la bande passante disponible de manière proportionnelle.

XCP est donc capable d'atteindre très rapidement le débit optimal et de réagir aussi rapidement aux variations de bande passante. Cependant, dans le cas de cohabitation avec TCP, XCP récupérera moins de bande passante à cause du contrôle strict qu'il impose sur le taux de pertes. [8]

# III- Conclusion

Une file d'attente est une structure qui stocke de manière ordonnée des éléments et qui rend accessible uniquement un seul d'entre eux.

Dans ce chapitre, nous avons présenté les notions de gestion de file d'attente telle que les processus d'arrivés qui décrit l'arrivé des clients à la file avec l'intervalle du temps entre arrivées successives, processus de service qui est défini en fonction du temps de service selon une loi exponentielle et la notion de Kendall qui décrit le système de files d'attente par une suite de paramètres A/B/n/K/N. Puis nous avons détaillé le système M/M/1 le plus connu, le phénomène de congestion et son contrôle avec le protocole XCP.

[6] AIMD est un algorithme d'évitement de congestion utilisé par TCP

# Chapitre III

# Modélisation et évaluation des performances

# I- Introduction

La simulation nous permet de rendre visible le fonctionnement d'une file d'attente et par ce fait nous permet d'analyser la circulation du flux dans notre réseau.

Pour accomplir notre objectif et pouvoir modéliser le trafic en réseau nous avons simulé le fonctionnement des files d'attente que sont FIFO, SFQ et RED. Après visualisation des résultats obtenus nous avons affiché les graphes correspondant à chacune d'entre elles après quoi nous avons discuté les différents résultats obtenus. Pour plus de détails ce reporter a la section II. Par la suite nous avons simulé les systèmes de file d'attente de type M/M/1 et M/M/1/K. Dérivés de la notation de Kendall ces systèmes admettent des lois exponentielles des processus d'arrivée ainsi que le processus de service. Le principe était de varier les paramètres que sont la fréquence d'arrivée des paquets et le taux de service, afin d'étudier le comportement de ces deux systèmes. En dernier nous avons abordé le protocole XCP qui est un exemple de protocole de contrôle de congestion.

# II- Simulations des files d'attente

## 1- FIFO First In First Out

Sur Internet, qui est le réseau des réseaux, tous les paquets arrivant dans un routeur sont placés dans une file d'attente de type First In First Out (FIFO) pour un traitement des flux.

Cet algorithme ne s'applique que dans le cas de la présence d'une seule file d'attente. Les paquets sont ajoutés en queue de la file d'attente et c'est le paquet en tête de file qui est traité en premier. Cependant cette gestion des paquets entraîne une variation des délais (gigue) entre l'arrivée des paquets dans le cas où plusieurs flux sont présents. [9]

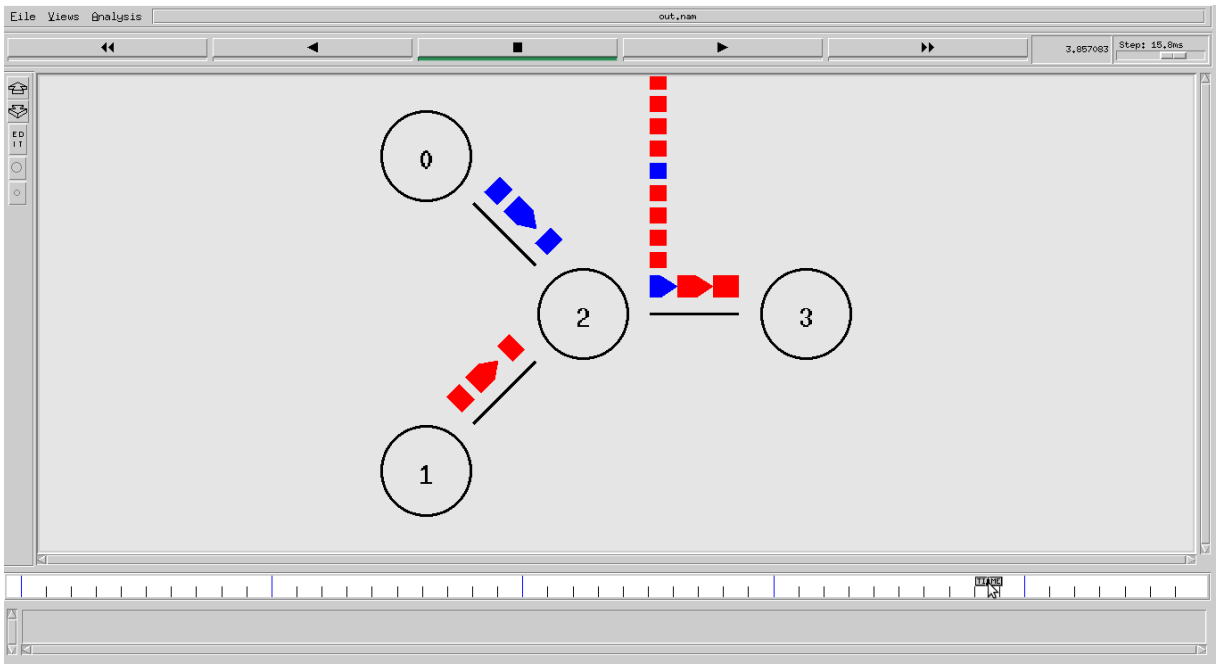



**Figure 5 : Simulation 1 (File d'attente FIFO)**

Dans cette simulation, on a deux nœuds (n0, n1) émettant vers la destination (n3) en passant par le nœud 2 (commutateur ou routeur). Le lien qui relie les nœuds entre eux est défini par :
$ns duplex-link $n0 $n3 1Mb 100ms DropTail
Cette ligne permet de connecter les deux nœuds n0 et n3 avec 1 Mb de bande passante, 100 ms de délai et une file d'attente FIFO.

Une saturation du lien reliant les nœuds 2 et 3 se produit à partir du moment où la quantité de paquets envoyés dépasse la capacité du lien, alors la file d'attente se sature et les paquets seront rejetés. Les premiers paquets arrivés sont transmis et le reste des paquets sont rejetés.

Drop Tail met en file d'attente un certain volume du trafic et élimine tout ce qui déborde. Ce n'est pas équitable et entraine des retransmissions et inondera à nouveau le routeur congestionné.

### 2- SFQ Stochastic Fairness Queueing

Stochastic Fairness Queueing (SFQ) est une file d'attente stochastiquement équitable, c'est une implémentation simple de la famille des algorithmes de mise en file d'attente équitable. Cette implémentation est moins précise que les autres, mais elle nécessite aussi moins de calculs tout en étant presque parfaitement équitable.

SQF divise le trafic en flux, et affecte chaque flux à une file d'attente. Ceci permet de donner à chaque connexion l'opportunité d'envoyer des données de façon très équitable et empêche qu'une seule conversation étouffe les autres. [4]

Le trafic est divisé en file d'attente FIFO, en envoyant un paquet de chaque FIFO à chaque tour. C'est pour cette raison qu'elle est appelée équitable.

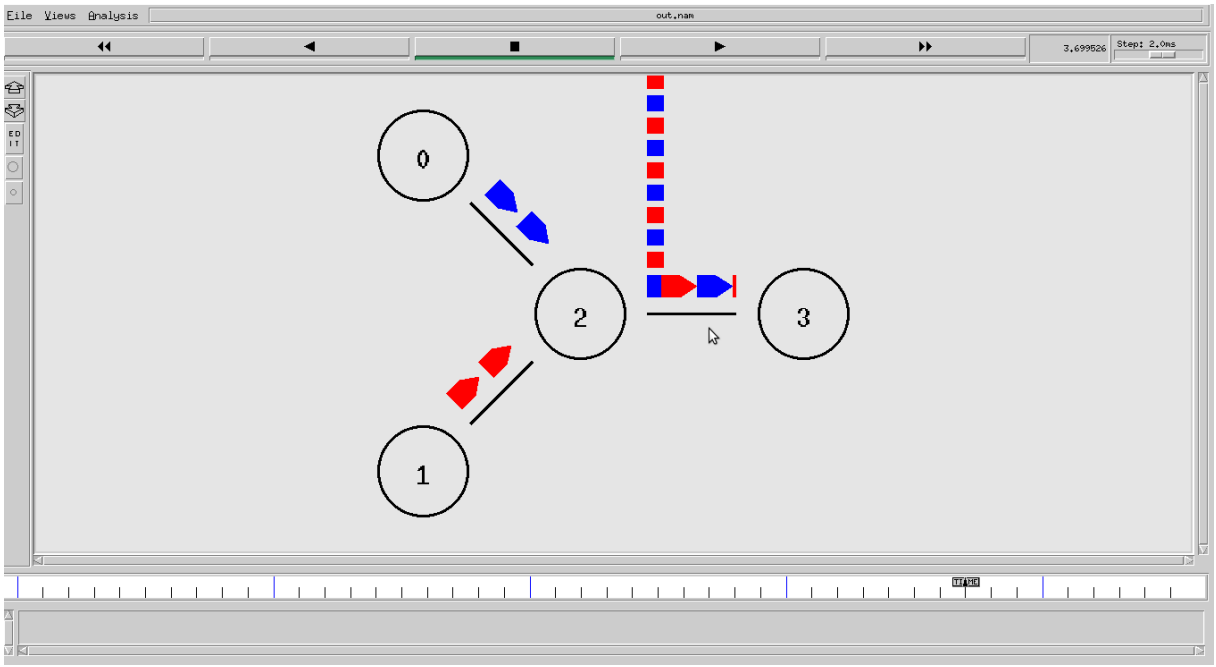



**Figure 6 : simulation 2 SFQ**

Pour simuler SFQ, Nous avons due remplacer la discipline de FIFO par une SFQ

La ligne à modifier est la suivante :

$ns duplex-link $n0 $n3 1Mb 100ms DropTail

On la remplace par :

$ns duplex-link $n0 $n3 1Mb 100ms SFQ

**Remarques et observations :**

On remarque que SFQ est plus équitable que FIFO, le nombre de paquets rejetés des deux sources est le même, les deux flux ont la même priorité.

### 3- RED Random Early Detection

RED (Random Early Detection) ou détection précoce directe a été définie par Sam Floyd et Van Jacobson, ils l'ont développé afin de proposer des avantages par rapport à Drop Tail.

RED permet de détecter la congestion au niveau d'un routeur avant que cette dernière ne se produise. Le routeur surveille la taille de sa file d'attente et suivant sa valeur, rejette suivant une certaine probabilité des paquets à leur arrivée par mesure préventive afin que ces flux pénalisés réduisent leur débit.

Plutôt que d'attendre que la file soit pleine, on élimine un paquet qui arrive avec une certaine probabilité d'élimination (drop probability) quand la file d'attente atteint un certain niveau (drop level). [1]

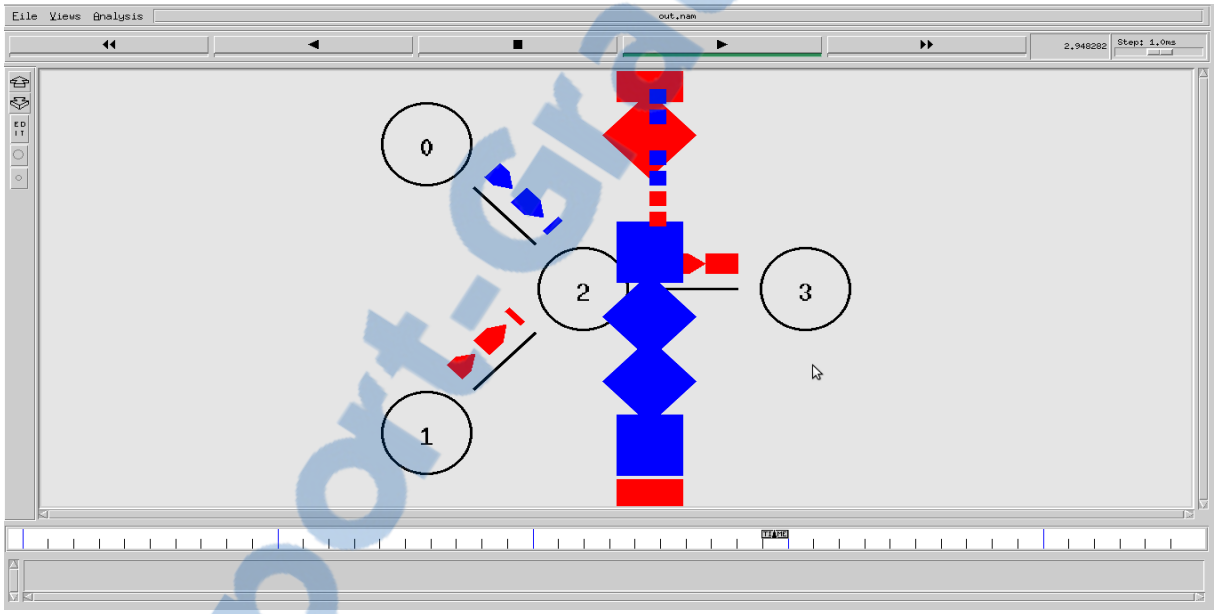



**Figure 7 : Simulation 3 RED**

RED élimine statistiquement des paquets du flux avant qu'il n'atteigne sa limite (saturation de la file d'attente).

## III- Etude et évaluation des résultats

Dans la modélisation des différentes files d'attente on a enregistré les statistiques de la simulation avec la commande suivante :

*puts $f0 "$now [$sink0 set nlost_] [expr $bw0/$time*8/1000000] "*

Elle génère trois fichiers trace, chaque fichier contient trois colonnes : le temps de la mesure, le nombre de paquets perdus (nlost_) et la bande passante mesuré en mbps.

Pour visualiser les résultats des fichiers trace, on appelle XGRAPH par la commande :

*exec xgraph out0.tr out1.tr out2.tr -geometry 800x400*

Le script qui affiche les graphes des paquets perdu et la bande passante ce trouve dans l'annexe A.

## 1- Les graphes de Drop Tail

### 1.1- Paquets perdus

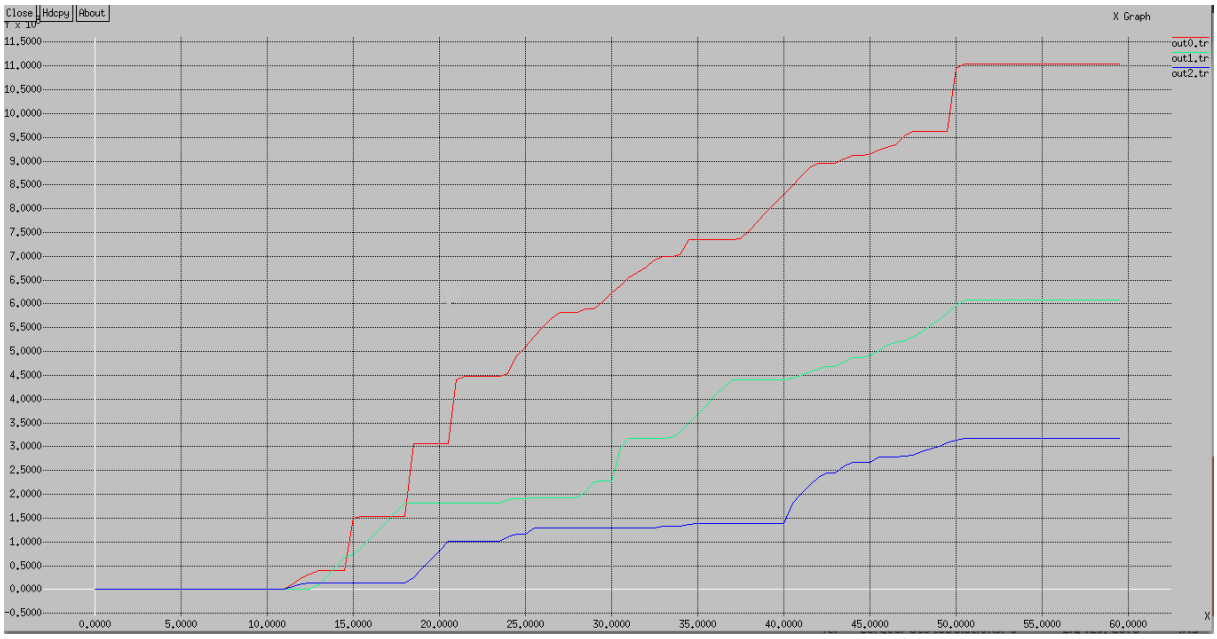



**Figure 8 : Graphe des paquets perdus en fonction du temps**

- **Analyse du graphe**

Le graphe représente le nombre de paquets perdus pendant 60 secondes de simulation par les trois sources.

En période de 0 à 10 secondes la perte des paquets est nulle parce qu'il n'y a pas de paquets rejetés car les sources n'ont pas encore envoyé leur paquets.

De 10 à 50 secondes nous remarquons que le nombre des paquets rejetés augmente considérablement cela se traduit par le fait que le flux a dépassé la capacité maximale du lien et que la file d'attente est saturée (la congestion).

Après la cinquantième seconde, le nombre de paquets rejetés n'augmente pas parce qu'il y a plus de flux sur le lien.

On remarque que le nombre de paquets rejetés de la source 0 est plus important que les deux autres sources, quand la file d'attente est remplie à sa capacité maximum, les paquets nouvellement arrivés sont rejetés jusqu'à ce qu'une place soit libérée dans la file d'attente pour accepter le trafic entrant.

## 1.2- Bande passante

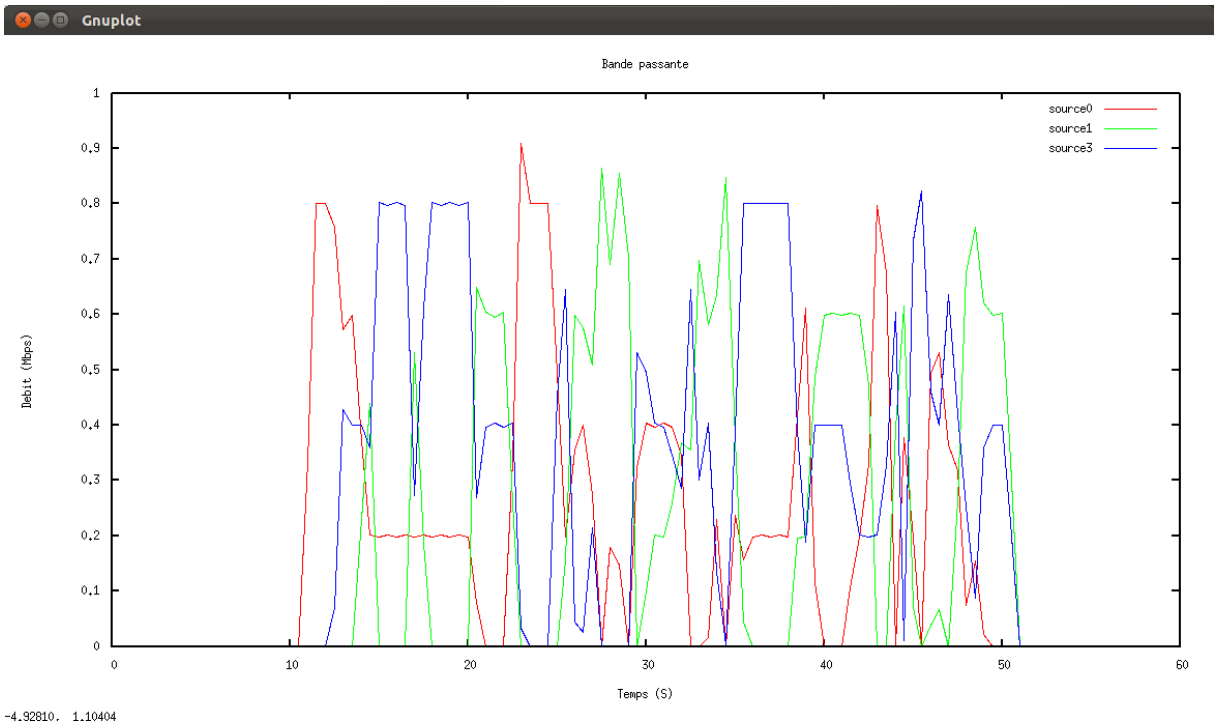



**Figure 9 : Graphique de la bande passante**

La courbe en rouge représente la bande passante utilisée par la source 0, celle en vert représente la bande passante utilisée par la source 1, et la courbe en bleu représente la bande passante utilisée par la source 2.

On remarque que la somme de bande passante des trois sources à un instant donné ne dépasse pas la capacité du lien qui est 1 mbps.

## 2- Les graphes de SFQ

### 2.1- Paquets perdus

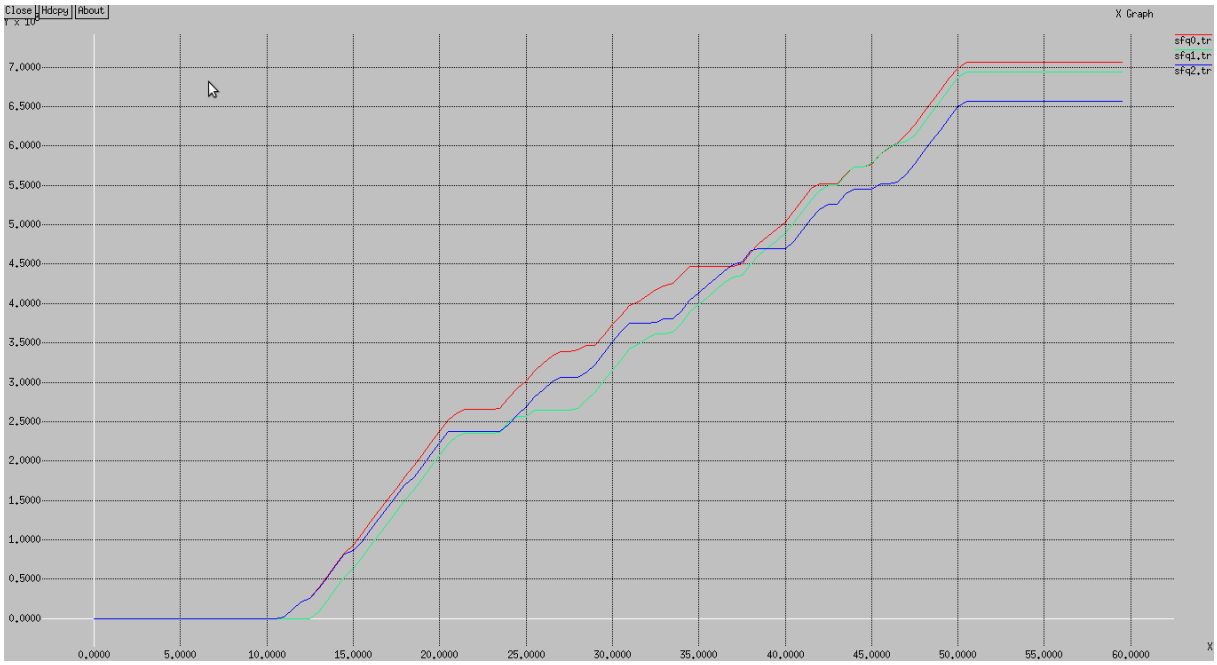



**Figure 10 : graphes des paquets perdus avec la file SFQ**

- **Analyse du graphe**

Le graphe de paquets perdus en fonction de temps nous confirme l'équité de la file d'attente SFQ, qui ne permet pas à une seule source d'étouffer le trafic mais plutôt de donner la même priorité aux paquets venus des trois nœuds sources, le nombre de paquets rejetés est presque le même entre les trois sources.

## 2.2- Bande passante

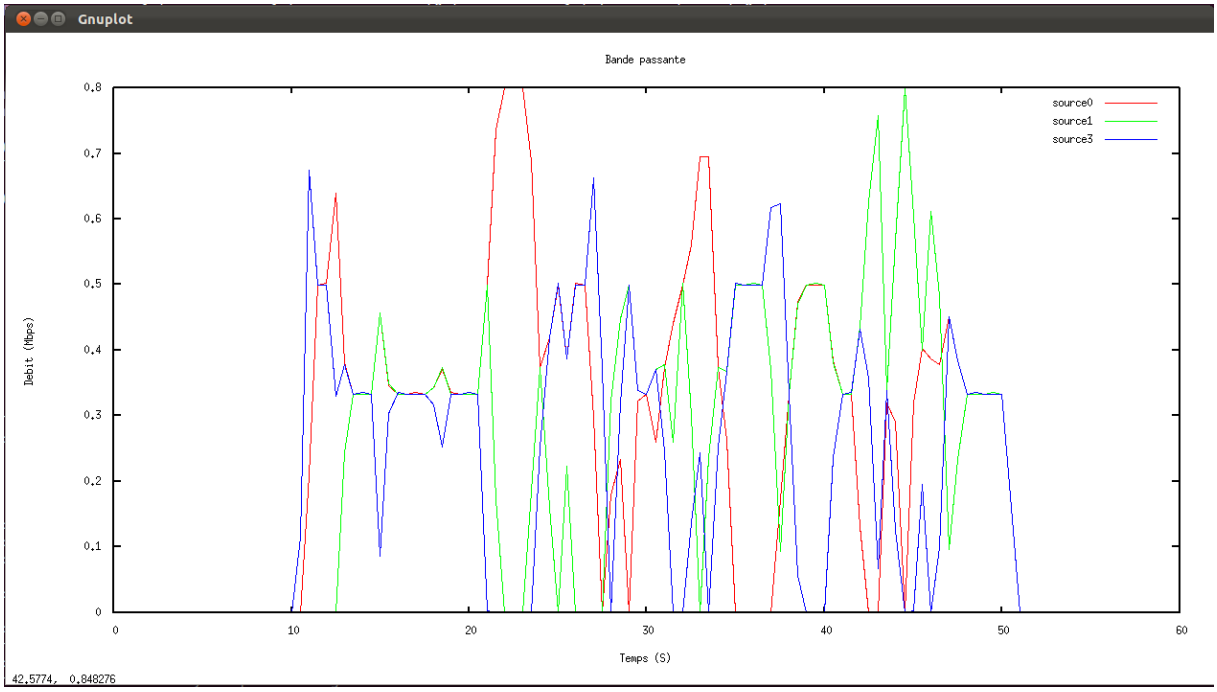



**Figure 11 : Graphique de la bande passante avec SFQ**

# 3- Les graphes de RED

## 3.1- Paquets perdus

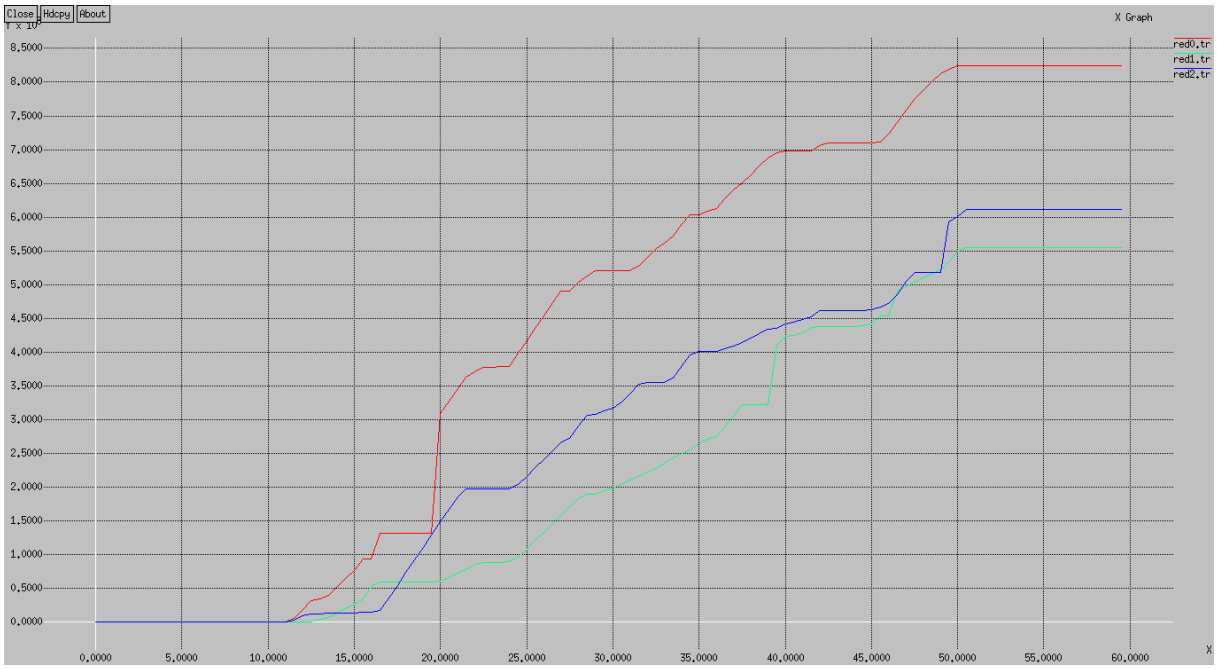



**Figure 12 : graphes des paquets perdus avec RED**

- **Analyse du graphe**

On remarque dans le graphe que les pertes des paquets sont moins importantes qu'avec Drop Tail car RED est une gestion active, à mesure préventive contre la congestion, RED surveille le taux de remplissage de la file d'attente et quand il dépasse le seuil (drop level), les paquets seront rejetés pour réduire le flux a fin que la file d'attente ne se sature pas.

### 3.2- Bande passante

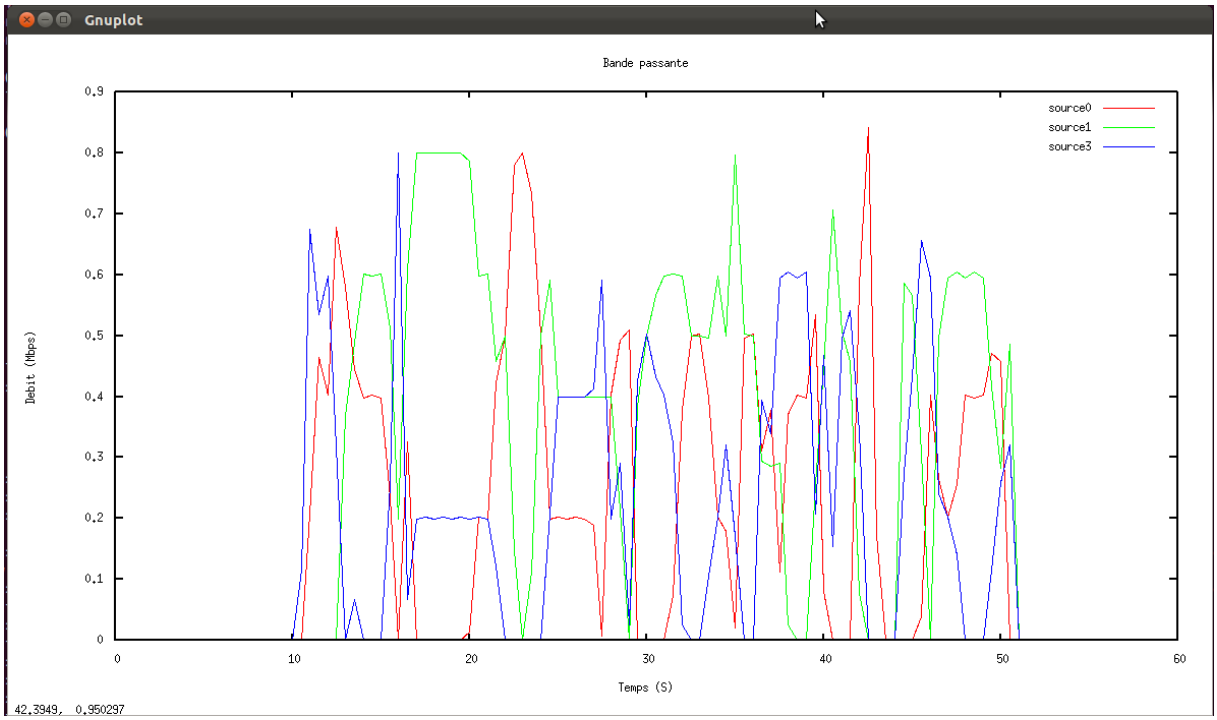



**Figure 13 : Bande passante avec RED**

## IV- Simulation d'un système M/M/1

La notation de Kendal vu dans le chapitre précédent, M/M/1 signifie un système aux arrivées Markoviennes / Poissoniennes (M), à temps de traitement Markovien / Exponentiel (M), composé d'une file d'attente FIFO de taille illimitée et d'un seul serveur (1)

### 1- La démarche

On initialise les paramètres λ et μ où λ est le nombre moyen de paquets arrivés par seconde (la fréquence des paquets) et μ présente la taille moyenne d'un paquet.

*set lambda 30.0*

*set mu 33.0*

En suite on crée deux nœuds, un lien simplex avec la file FIFO d'une taille illimitée et sans temps d'accès.

*set n1 [$ns node]*

*set n2 [$ns node]*

*set link [$ns simplex-link $n1 $n2 100kb 0ms DropTail]*

*$ns queue-limit $n1 $n2 100000*

Et après on génère l'intervalle aléatoire de temps et la taille des paquets et on crée les exponentielles d'arrivée pour le temps et la taille des paquets.

*set InterArrivalTime [new RandomVariable/Exponential]*

*$InterArrivalTime set avg_ [expr 1/$lambda]*

*set pktSize [new RandomVariable/Exponential]*

*$pktSize set avg_ [expr 100000.0/(8*$mu)]*

On monitore la file d'attente au cours du temps avec le pas de 0.1

*set qmon [$ns monitor-queue $n1 $n2 [open qm.out w] 0.1]*

*$link queue-sample-timeout*

Pour plus de détail voir l'annexe A.

## 2- Etude approfondie du M/M/1

Nous allons varier les paramètres λ et μ et étudier le comportement de la file d'attente à l'aide des graphiques.

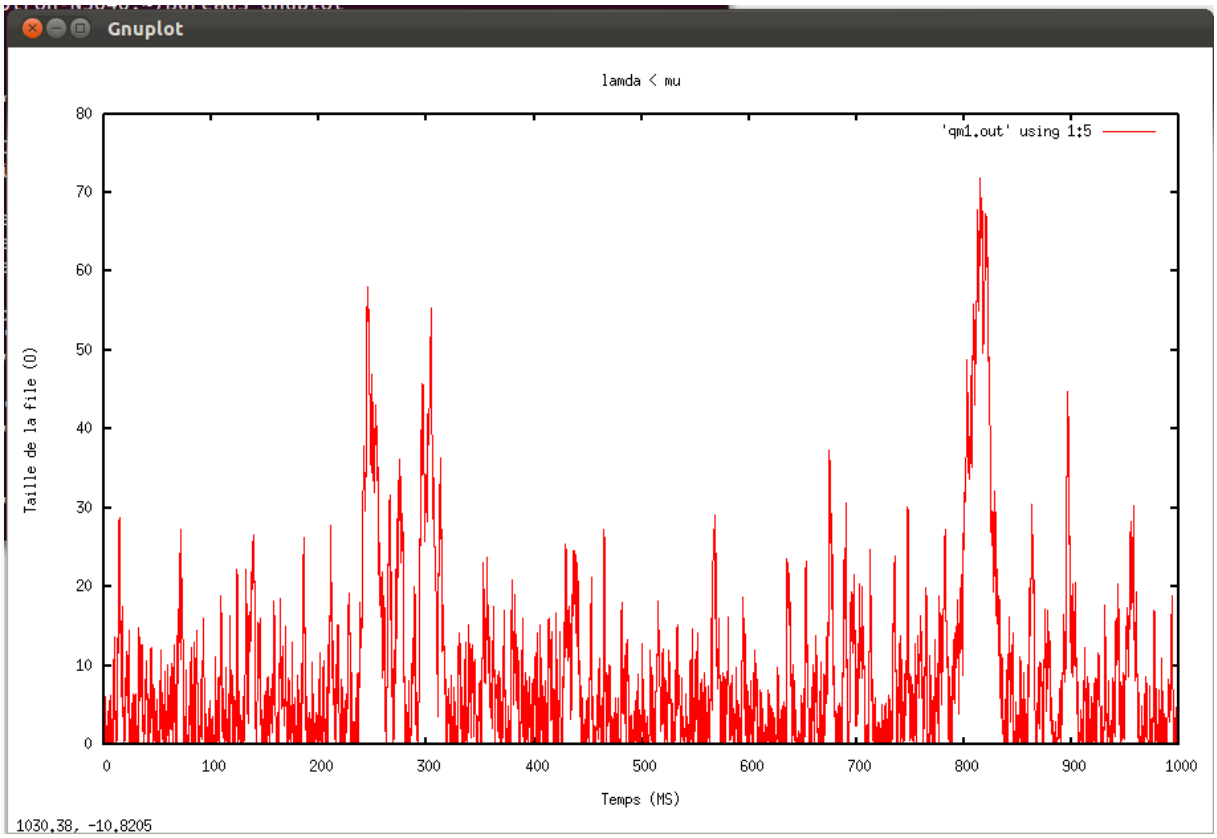



**Figure 14 : La taille de la file d'attente avec λ < μ**

- **Analyse du graphe**

Dans ce cas où la fréquence des paquets qui est calculé en fonction de λ (lambda[7]) est inférieur au taux de service qui est calculé en fonction de μ (mu[8]) {voir section II.2.5}, nous remarquons que la taille ne dépasse pas 80 octets qui est très bas cela signifie que le traitement (mise en file d'attente et acheminement de paquets) est plus rapide que l'arrivée des paquets.

[7] Intervalle de temps d'arrivée des paquets.
[8] Taille moyenne des paquets

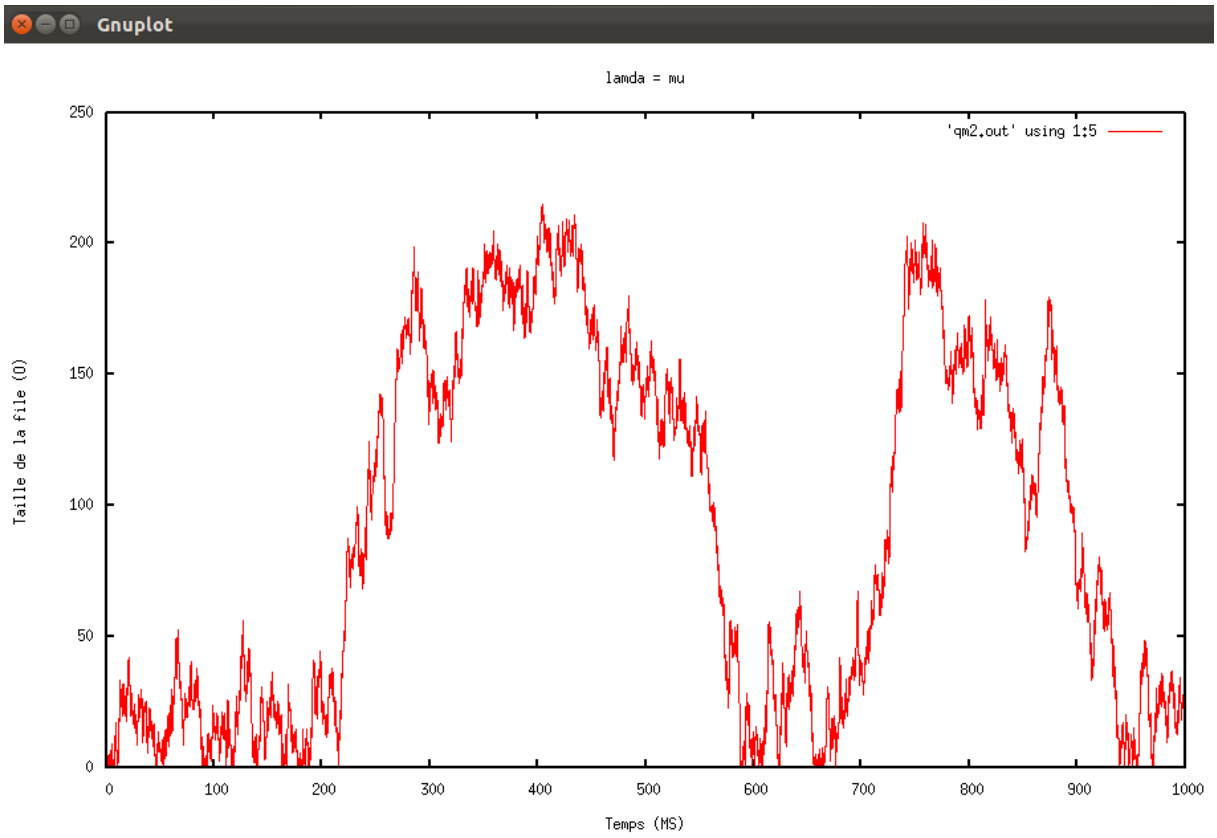



**Figure 15 : La taille de la file d'attente avec λ =μ**

- **Remarques et observations**

Quand λ = μ qui veut dire que les paquets quittent la file autant qu'ils n'arrivent, la taille de la file augmente mais reste stable, elle ne dépasse pas 250 octets et arrive à se vider (les paquets sont évacués a peu près a la même vitesse qu'ils arrivent).

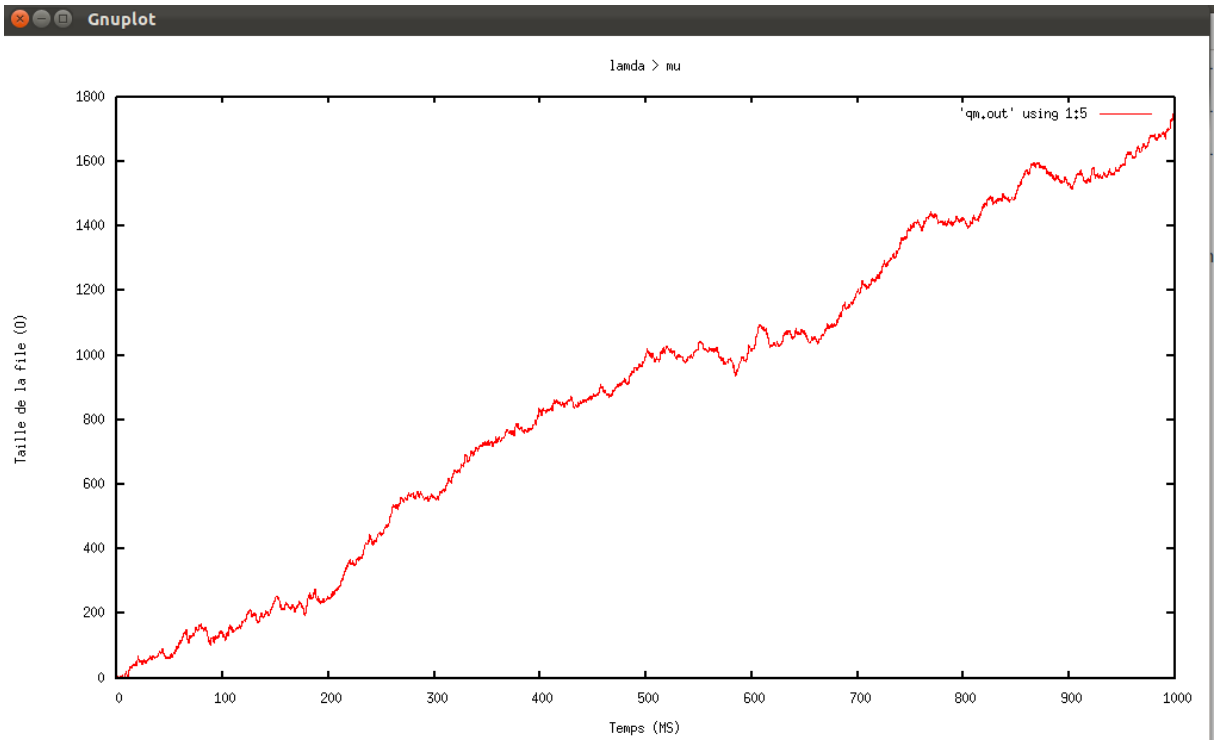



**Figure 16 : La file d'attente avec λ > μ**

- **Analyse du graphe**

Dans le cas où lambda est supérieur à mu, les paquets entrent en file d'attente plus rapidement qu'ils ne sortent du système. Aussi, le nombre de paquets en file d'attente augmente de façon constante et atteint presque 1800 octets en fin de simulation.

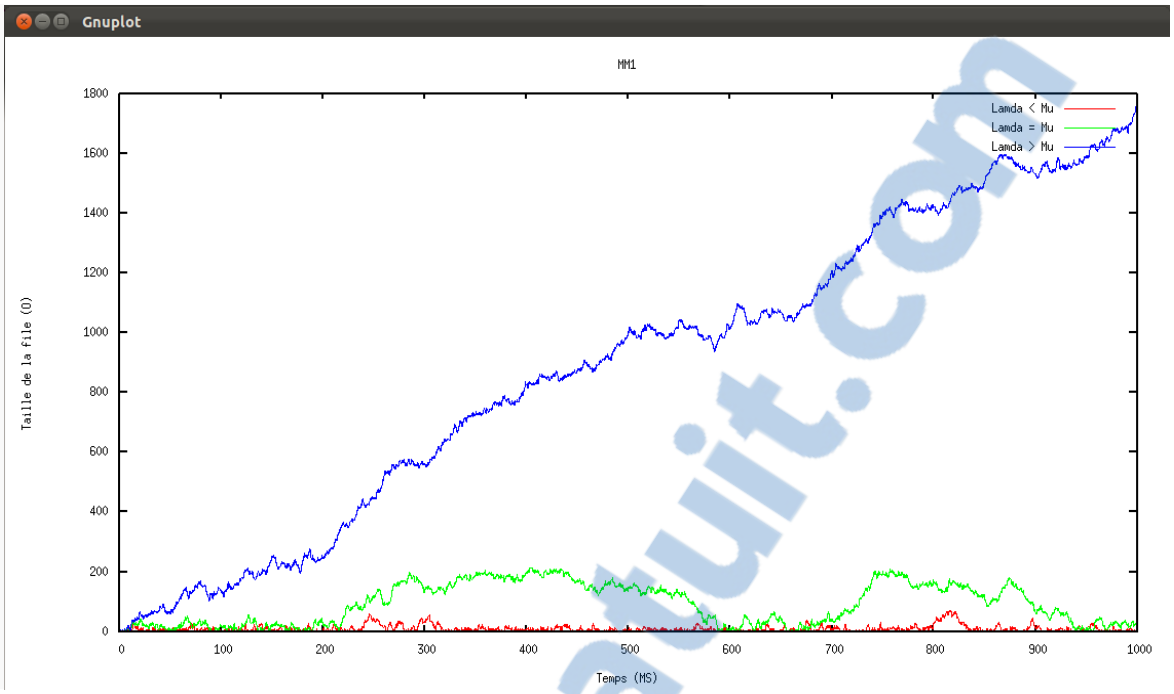



**Figure 17 : graphique de la taille de la file en fonction de lambda et mu**

- **Remarque**

Ce graphe regroupe les trois cas, pour mieux voir la différence de variation des paramètres, quand lambda est inférieure ou égale à mu, la taille de la file d'attente est assez stable et arrive à se vider, mais une fois que lambda dépasse mu, la taille de la file d'attente augmente considérablement.

## V- Simulation du système M/M/1/K

Un système M/M/1/K est un système dont la loi d'arrivée des clients est Markovienne (Poissonienne), donc la loi de traitement est Markovienne (Exponentielle), qui possède un seul serveur, et dont la file d'attente est limité à K, où K est un entier qui représente la taille limitée de la file.

La démarche est la même que MM1, il faut juste modifier la limite de la file d'attente à 50 octets, qui est fait par :

*$ns queue-limit $n1 $n2 50*

### 1- Etude de M/M/1/K

Les variations des paramètres lambda et mu donnent les résultats suivants :

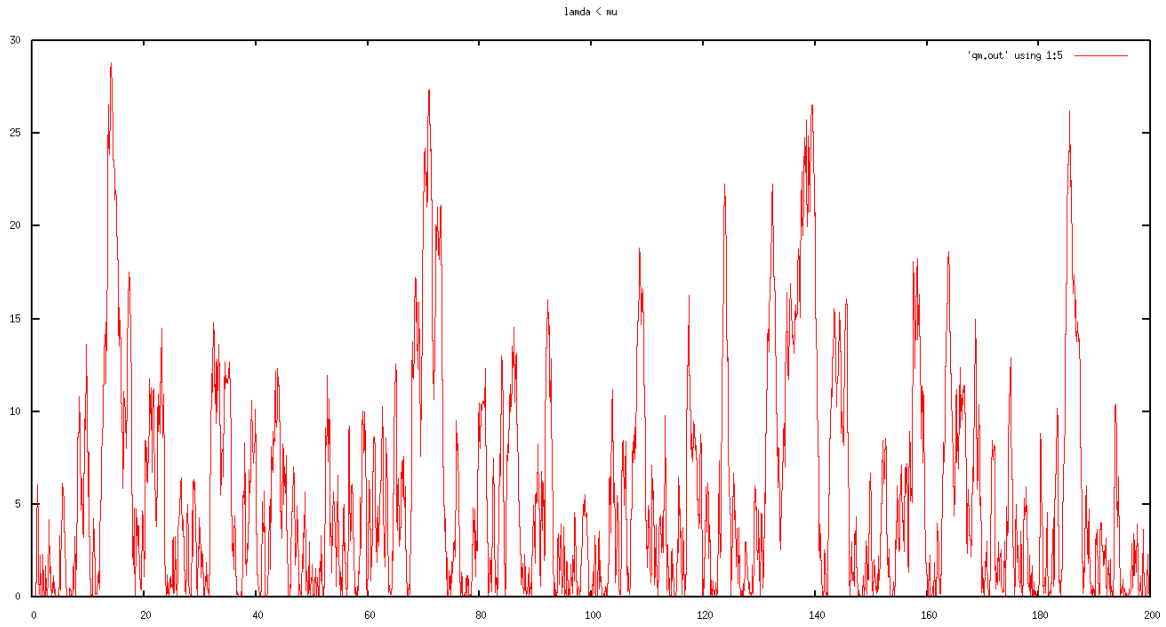



**Figure 18 : La taille de file d'attente avec λ < μ**

- **Analyse du graphe**

Dans le cas où λ < μ, la file ne se sature pas sa taille ne dépasse pas 30 octets, ce qui veut dire qu'il n'y a pas de paquets perdus.

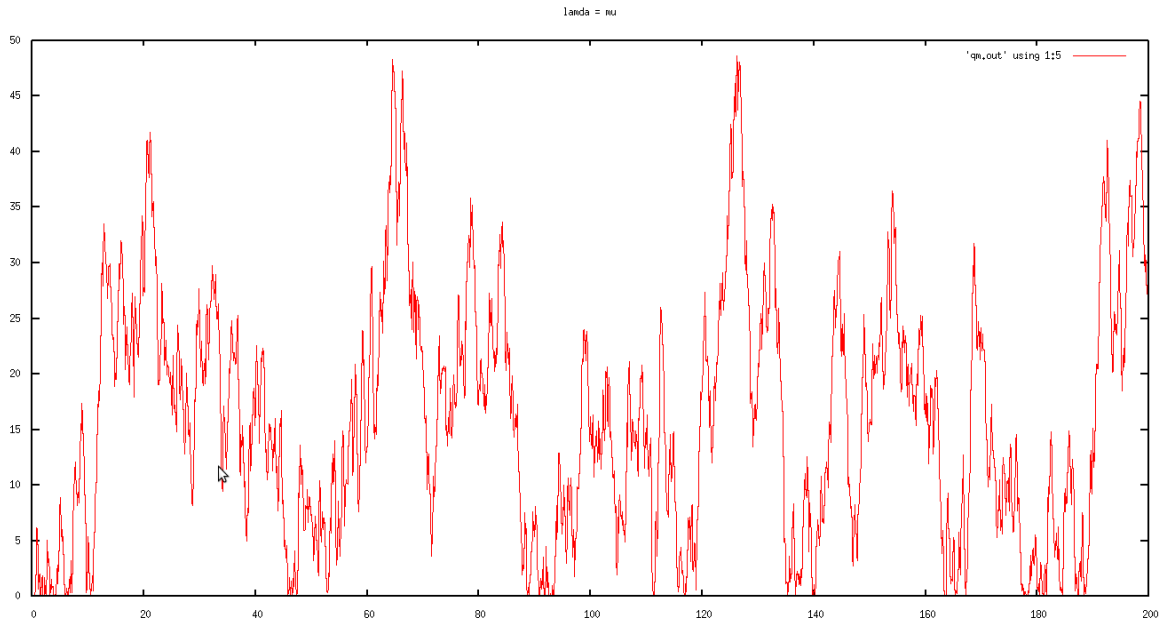



**Figure 19 : la taille de la file d'attente avec λ = μ**

Quand λ = μ, la quantité de paquets perdus est faible, car la file se sature occasionnellement, par exemple à l' instant 64 ms et 126 ms.

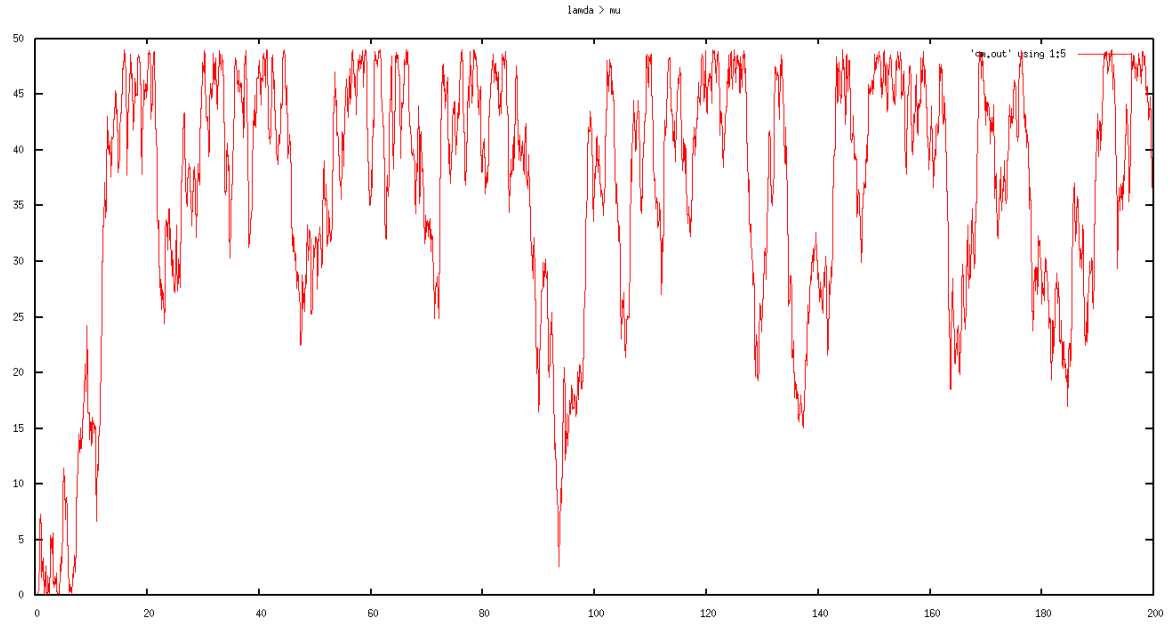



**Figure 20 : la taille de la file d'attente avec λ > μ**

- **Observations et remarques**

Dans ce cas où la fréquence des paquets qui est calculé en fonction de λ (Intervalle de temps d'arrivée des paquets) {voir section II.2.5} est supérieur au taux de service qui est calculé en fonction de μ (Taille moyenne des paquets), la file d'attente se sature régulièrement ce qui engendre une congestion, car l'arrivée des paquets dépasse la capacité de traitement du serveur.

## VI- Simulation du Protocol de contrôle de congestion XCP

XCP est un protocole de contrôle de congestion, il permet d'avoir beaucoup d'efficacité, de stabilité et d'équité en réseau en envoyant le retour d'information (feedback) à la source pour adapter le flux en réseau.

XCP informe l'émetteur des conditions de congestion, l'en-tête des paquets XCP contient la taille actuelle de la fenêtre de congestion (H_cwnd[9]).

A la réception du paquet, l'en-tête du paquet reçu sera recopié dans l'ACK[10] alors l'émetteur mettra à jour la taille de sa fenêtre de congestion de la manière suivante :

Cwnd= max (cwnd+H_feedback, packet size)

[9] High congestion Window (la taille de la fenêtre de congestion).
[10] ACKnowlegment est le paquet d'accusé de réception.

XCP est basé sur l'utilisation de deux contrôleurs EC[11] et FC[12].

EC maximise la bande passante et minimise le rejet des paquets et FC partage les ressources équitablement.

### 1- Démarche

Le script complet de la simulation du XCP se trouve dans l'annexe A.

On commence par la création de la topologie avec les étapes suivantes :

- Création des nœuds du réseau.
- Définition des liens avec le protocole XCP
- Définition des paramètres de la file d'attente RED, Limiter la taille de la file d'attente.
- Création du trafic et l'attacher à l'agent TCP.

En suite, on initialise la simulation en appelant NS. Après, on génère les fichiers traces qui contiennent les données de la simulation. En fin, on affiche le graphe de la taille de la fenêtre de congestion.

- **Graphe résultants**

[11] Efficiency Controler est le contrôleur d'efficacité.
[12] Fairness Controler est le contrôleur d'équité.

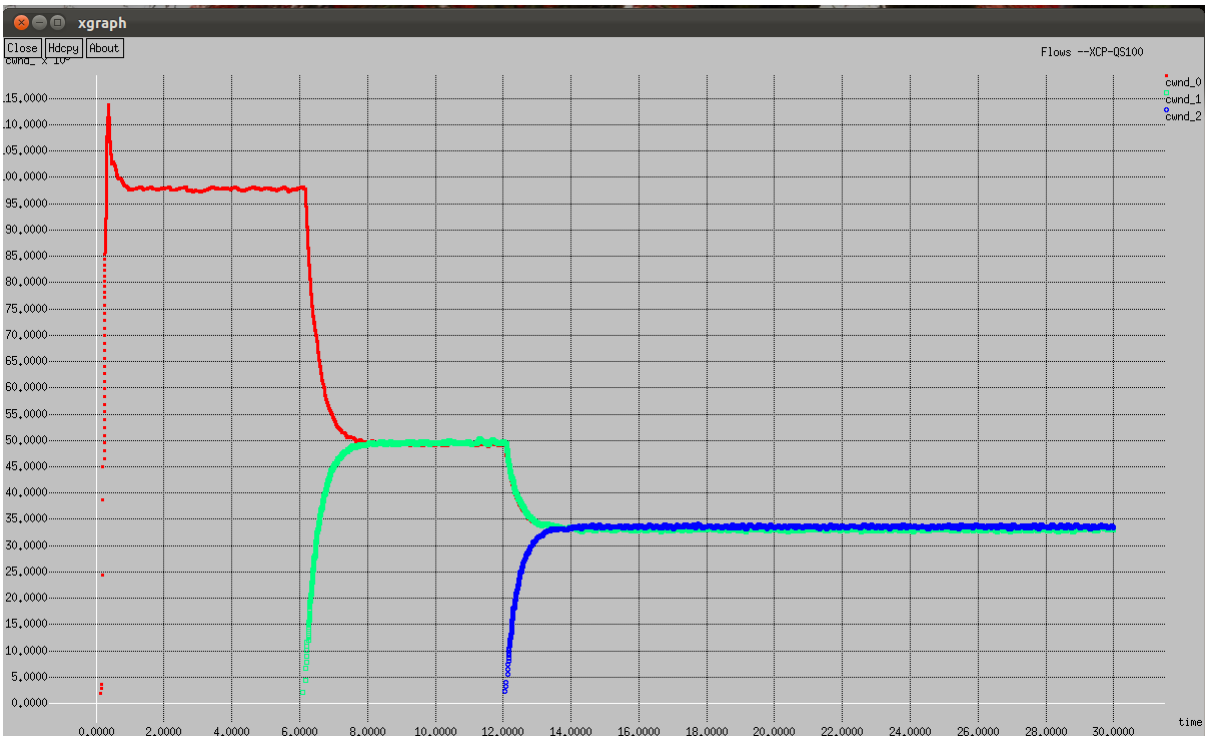



**Figure 21 : Graphique de la taille de la fenêtre de congestion**

- **Observations**

Le graphe représente la taille de la fenêtre de congestion des trois sources en fonction du temps.

La première source commence l'envoi des paquets dés le début de la simulation, alors la taille de la fenêtre de congestion augmente considérablement jusqu'à 115 000 octets car le contrôleur d'efficacité EC du protocole XCP cherche à utiliser le maximum de la bande passante puis cherche à l'optimiser vers 100 000 Octets pour éviter la congestion et le rejet des paquets en réduisant le flux.

A sixième seconde la deuxième source débute son envoi de paquets, on remarque que la taille de la fenêtre de congestion de la deuxième source augmente ainsi que celle de la première source diminue et les deux se stabilisent au niveau de 50 000 Octets ; cela signifie que le contrôleur d'équité partage la bande passante entre les deux sources équitablement, sachant que EC maximise toujours l'utilisation de la bande passante et minimise le rejet des paquets.

La troisième source envoie les paquets à la 12 sec et on remarque le même comportement qu'à la sixième seconde la taille de fenêtre de congestion des deux premières sources se réduit et la troisième augmente jusqu'à 33 000 Octets, car le partage de la ressource (bande passante) est équitable.

## VII- Conclusion

Quelle que soit la politique de qualité de service utilisée qui est définie par ces paramètres : la bande passante, le délai, la gigue et le taux de perte. C'est au niveau des files d'attente des routeurs que cette politique est mise en œuvre, pour cela nous avons simulé les différentes files d'attente et étudié leurs résultats. Nous avons constaté qu'elles se caractérisent par deux gestions actives et passives, la gestion passive s'occupe de savoir que s'il reste de la place dans la fille d'attente avant d'accepter un paquet ou de le rejeter, elle est illustré par DROP TAIL par contre la gestion active prévient la congestion et réduise le flux avant la saturation de la file d'attente en rejetant des paquets telle que RED.

Après la modélisation du système M/M/1 et la variation de ses paramètres, on conclue que le système est stable quand le taux d'arrivé $\lambda$ est inférieur ou égale au taux de service $\mu$.

Finalement, la simulation du protocole de contrôle congestion XCP induit que c'est une solution à la congestion, car XCP informe en temps actuel l'émetteur du taux de remplissage de la fenêtre de congestion et adapte son flux en fonction de celle ci.

## Conclusion générale

En modélisation de flux dans les réseaux, nous avons simulé avec NS (Network Simulator) différentes files d'attente. FIFO connue aussi par 'Drop TAIL' qui serve le premier paquet arrivée et rejette le reste des paquets une fois la file saturée, SFQ est plus équitable par rapport a Drop Tail car elle divise le flux arrivé en plusieurs partie avec la même priorité et RED gère la file activement, elle prévient contre la congestion en rejetant quelques paquets avant la saturation de la file pour que les sources réduisent leur flux, nous avons réalisé aussi une étude approfondie sur les systèmes M/M/1 et M/M/1/K est un système aux arrivées Markoviennes (M), à temps de traitement Markovien / Exponentiel (M), composé d'une file d'attente FIFO de taille illimitée ou limité K et d'un seul serveur (1) et en fin un protocole de contrôle de congestion XCP qui offre l'efficacité, la stabilité et l'équité en réseau grâce aux deux contrôleurs EC et FC. L'objectif de la simulation est d'étudier leurs performances et évaluer leurs résultats.

Ce qui nous a permis de comprendre les causes principales du phénomène de congestion, de le gérer avec les files d'attente et d'utiliser une solution à la congestion tel que le protocole XCP. Puis comprendre le fonctionnement des systèmes de files d'attente et de constater leur utilité en réseau.

Il existe d'autres types de files d'attente à titre d'exemple CBQ (Class Based Queuing) permet d'allouer une certaine proportion de bande passante pour une classe de trafic donnée et DRR Deficit Round Robin.

Les files d'attente permettent de gérer la congestion en minimisant la perte des paquets qui évite la retransmission ce qui économise la bande passante et réduit la latence qui se définie comme la variation des délais d'acheminement ce qui enrichit la couche réseau en assurant une bonne qualité de service.

# Références bibliographiques

[1] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. IEEE/ACM Transactions on Networking, 1993.

[2] B. Baynat, Hernès, Théories des files d'attente : Des chaines de Markov aux réseaux, 2000.

[3] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001

[4] Bert Hubert, Thomas Graf, Routage avancé et contrôle de trafic sous linux, Gestionnaire de mise en file d'attente pour l'administration de bande passante, livre, p 23, 2001.

[5] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In ACM SIGCOMM, 2002.

[6] Djillali Seba, Cisco: installation, configuration et maintenance de réseaux, Contrôle de congestion, ENI, Livre, p 381, 2003

[7] M. Rahoual, P. Siarry, Réseaux informatique : conception et optimisation, THECNIP, page 117, 2006.

[8] Dino M. Lopez Pacheco, Laurent Lefèvre al. XCP-i : "eXplicit Control Protocol" pour l'interconnexion de réseaux haut-débit hétérogènes, Le Protocole XCP, Rapport de recherche, 2007.

[9] Sébastien LINCK, Optimisation et adaptation des communications dans un réseau hétérogène, gestion de la file d'attente, L'ordonnancement des paquets, Thèse, 2008.

[10] S. LINCK, Optimisation et adaptation des communications dans un réseau hétérogène, gestion de la file d'attente, Explicit Congestion Notification, Thèse, 2008.

[11] K. Fall, K. Varadhan, The NS Manual, eXplicit Congestion control Protocol, pages 204 – 209, rapport technique, 2011

[12] P. Anelli, E. Horlait, Manuel_NS1.3, http://www-sop.inria.fr/rodeo/personnel/Pierre.Ansel/Manuel_NS1.3.pdf, 2000.

[13] G. Hébuterne, T. Atmaca, Théories des files d'attente, http://www-rst.int-evry.fr/~hebutern/IT21/PolyIT21.pdf, 2002.

[14] Anthony, Les réseaux, http://www.vulgarisation-informatique.com/types-reseaux.php, 2007.

[15] A. Jeff, Réseaux/Internet, Initiation aux réseaux, http://www.commentcamarche.net/contents/initiation/types.php3, 2010.

[16] B. Bachelet, Structure de données, http://www.nawouak.net/?doc=course.data_structures+ch=queues+lang=fr, 2011.

[17] Sophia Antipolis, Les réseaux informatiques, Université de Nice, Bioinfo.unice.fr, 2011.

# Annexe A

L'annexe A comporte les scripts suivants :

- **Les scripts de simulation de FIFO, SFQ et RED**

```
#Create a simulator object
set ns [new Simulator]

# Création du fichier de traces NS-2
set nf [open out.nam w]
$ns namtrace-all $nf
set f3 [open trace.tr w]

#Open the output files
set f0 [open out0.tr w]
set f1 [open out1.tr w]
set f2 [open out2.tr w]

#Create 5 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

#Connect the nodes
$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

#Define a 'finish' procedure
proc finish {} {
        global ns nf f0 f1 f2 f3
        $ns flush-trace
        #Close the output files
        close $f0
        close $f1
        close $f2
        close $f3
        exec nam out.nam &
        #Call xgraph to display the results
        exec xgraph out0.tr out1.tr out2.tr -geometry 800x400 &
      exit 0
}


#Define a procedure that attaches a UDP agent to a previously created node
#'node' and attaches an Expoo traffic generator to the agent with the
```

```
#characteristic values 'size' for packet size 'burst' for burst time,
#'idle' for idle time and 'rate' for burst peak rate. The procedure connects
#the source with the previously defined traffic sink 'sink' and returns the
#source object.
proc attach-expoo-traffic { node sink size burst idle rate } {
        #Get an instance of the simulator
        set ns [Simulator instance]

        #Create a UDP agent and attach it to the node
        set source [new Agent/UDP]
        $ns attach-agent $node $source

        #Create an Expoo traffic agent and set its configuration parameters
        set traffic [new Application/Traffic/Exponential]
        $traffic set packetSize_ $size
        $traffic set burst_time_ $burst
        $traffic set idle_time_ $idle
        $traffic set rate_ $rate

      # Attach traffic source to the traffic generator
      $traffic attach-agent $source
        #Connect the source and the sink
        $ns connect $source $sink
        return $traffic
}


#Define a procedure which periodically records the bandwidth received by the
#three traffic sinks sink0/1/2 and writes it to the three files f0/1/2.
proc record {} {
      global sink0 sink1 sink2 f0 f1 f2
        #Get an instance of the simulator
        set ns [Simulator instance]
        #Set the time after which the procedure should be called again
      set time 0.5
        #How many bytes have been received by the traffic sinks?
      set bw0 [$sink0 set bytes_]
      set bw1 [$sink1 set bytes_]
      set bw2 [$sink2 set bytes_]

        #set p0 [$sink0 set nlost_]
        #set p1 [$sink1 set nlost_]
        #Get the current time
      set now [$ns now]
        #Calculate the bandwidth (in MBit/s) and write it to the files
      puts $f0 "$now [$sink0 set nlost_] [expr $bw0/$time*8/1000000] "
      puts $f1 "$now [$sink1 set nlost_] [expr $bw1/$time*8/1000000] "
      puts $f2 "$now [$sink2 set nlost_] [expr $bw2/$time*8/1000000]"
        #Reset the bytes_ values on the traffic sinks
      $sink0 set bytes_ 0
```

```
        $sink1 set bytes_ 0
        $sink2 set bytes_ 0
           #Re-schedule the procedure
        $ns at [expr $now+$time] "record"
}


#Create three traffic sinks and attach them to the node n4
set sink0 [new Agent/LossMonitor]
set sink1 [new Agent/LossMonitor]
set sink2 [new Agent/LossMonitor]
$ns attach-agent $n4 $sink0
$ns attach-agent $n4 $sink1
$ns attach-agent $n4 $sink2

#Create three traffic sources
set source0 [attach-expoo-traffic $n0 $sink0 200 2s 1s 800k]
set source1 [attach-expoo-traffic $n1 $sink1 200 2s 1s 800k]
set source2 [attach-expoo-traffic $n2 $sink2 200 2s 1s 800k]

#Start logging the received bandwidth
$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"
#Stop the traffic sources
$ns at 50.0 "$source0 stop"
$ns at 50.0 "$source1 stop"
$ns at 50.0 "$source2 stop"
#Call the finish procedure after 60 seconds simulation time
$ns at 60.0 "finish"

#Run the simulation
$ns run
```

Pour la simulation des filles d'attente SFQ et RED on utilise le même scénario il suffit de modifier les lignes suivantes dans le scripte précédent :

```
$ns duplex-link $n0 $n3 1Mb 100ms DropTail
```

*Par:*

```
$ns duplex-link $n0 $n3 1Mb 100ms SFQ ou RED
```

- **Le script de simulation du système M/M/1 et M/M/1/K**

```
set ns [new Simulator]

set tf [open out.tr w]
$ns trace-all $tf

set lambda 30.0
set mu     33.0
set n1 [$ns node]
set n2 [$ns node]

set link [$ns simplex-link $n1 $n2 100kb 0ms DropTail]
$ns queue-limit $n1 $n2 100000

# generate random interarrival times and packet sizes
set InterArrivalTime [new RandomVariable/Exponential]
$InterArrivalTime set avg_ [expr 1/$lambda]
set pktSize [new RandomVariable/Exponential]
$pktSize set avg_ [expr 100000.0/(8*$mu)]

set src [new Agent/UDP]
$src set packetSize_ 100000
$ns attach-agent $n1 $src

# queue monitoring
set qmon [$ns monitor-queue $n1 $n2 [open qm.out w] 0.1]
$link queue-sample-timeout

proc finish {} {
   global ns tf
   $ns flush-trace
   close $tf
   exit 0
}

proc sendpacket {} {
   global ns src InterArrivalTime pktSize
   set time [$ns now]
   $ns at [expr $time + [$InterArrivalTime value]] "sendpacket"
   set bytes [expr round ([$pktSize value])]
   $src send $bytes
}

set sink [new Agent/Null]
$ns attach-agent $n2 $sink
$ns connect $src $sink
$ns at 0.0001 "sendpacket"
$ns at 1000.0 "finish"
$ns run
```

Pour le script de M/M/1/K nous allons rajouter une instruction au script de MM1 qui limitera la taille de la file d'attente après la définition du lien avec ses paramètres, l'instruction et la suivante :

```
$ns queue-limit $n1 $n2 $qsize
```

Où qsize est une constante déclarée au par avant.

- **Le script du protocole XCP**

```
# ---------------------- Test XCP Performance -----------------------------#
#-------------------------------------------------------------------------#

#-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*- Utility Functions -*-*-*-*-*-*-*-*--*-*-*-*-*-*-*-*-*-#

#--------- Creating the TOPOLOGY --------------------#
#       n0
#       : \   Bottleneck
#       :  R0-----------------------R1
#        /
#       ni
#

Agent/TCP set minrto_ 1

proc set-red-params { qsize } {

        Queue/RED set thresh_ [expr 0.6 * $qsize]
        Queue/RED set maxthresh_ [expr 0.8 * $qsize]
        Queue/RED set q_weight_ 0.001
        Queue/RED set linterm_ 10
        Queue/RED set bytes_ false ;
        Queue/RED set queue_in_bytes_ false ;
        Agent/TCP set old_ecn_ true
        Queue/RED set setbit_     true
}

proc create-topology2 { BW delay qtype qsize numSideLinks deltaDelay } {
   global ns

   #Set the queue size to the pipe's size assuming the packet size is 1000 KByte
   if { $qsize == 0 } { set qsize [expr round([expr ($BW / 8) * 2 * $delay])] }

   set i 0
   while { $i < 2 } {
        global R$i
        set R$i [$ns node]
        incr i
   }

   $ns duplex-link $R0 $R1 [set BW]Mb [set delay]ms $qtype
   $ns queue-limit $R0 $R1 $qsize
```

```
    $ns queue-limit $R1 $R0 $qsize

    # Give a global handle to the Bottleneck Queue to allow
    # setting the RED paramters
    global Bottleneck rBottleneck
    set  Bottleneck  [[$ns link $R0 $R1] queue]
    set  rBottleneck [[$ns link $R1 $R0] queue]
    global l rl
    set  l  [$ns link $R0 $R1]
    set  rl [$ns link $R1 $R0]

    global all_links
    set all_links "$l $rl "

    # The side links  have the same BW as the Bottleneck
    set i 0
    while { $i < $numSideLinks } {
        global n$i q$i rq$i l$i rl$i
        set n$i  [$ns node]
        $ns duplex-link [set n$i]  $R0  [set BW]Mb [expr $delay + $i * $deltaDelay]ms
$qtype
        $ns queue-limit [set n$i]  $R0  $qsize
        $ns queue-limit $R0 [set n$i]   $qsize
        set  q$i   [[$ns link [set n$i]  $R0] queue]
        set  rq$i  [[$ns link $R0 [set n$i]] queue]
        set  l$i    [$ns link [set n$i]  $R0]
        set  rl$i   [$ns link $R0 [set n$i]]
        set all_links "$all_links [set l$i] [set rl$i] "
        incr i
    }
}
#------- Sender Class :
# This is essentially an ftp sender
Class GeneralSender  -superclass Agent

# otherparams are "startTime TCPclass .."
GeneralSender instproc init { id node rcvrTCP otherparams } {
    global  ns
    $self next
    $self instvar tcp_ id_ ftp_ node_ tcp_rcvr_ tcp_type_
    set id_ $id
    set node_ $node

    if { [llength $otherparams] > 1 } {
        set TCP [lindex $otherparams 1]
    } else {
        set TCP "TCP/Reno"
    }
    set   tcp_type_ $TCP
    set   tcp_ [new Agent/$TCP]
```

```
    $tcp_ set  packetSize_ 1000
    $tcp_ set  class_  $id
    set   ftp_ [new Source/FTP]
    $ftp_ set agent_ $tcp_
    $ns   attach-agent $node $tcp_
    $ns   connect $tcp_  $rcvrTCP
    set   tcp_rcvr_ $rcvrTCP
    set   startTime [lindex $otherparams 0]
    $ns   at $startTime "$ftp_ start"
    puts  "initialized Sender $id_ at $startTime"
}

GeneralSender instproc trace-xcp parameters {
    $self instvar tcp_ id_ tcpTrace_
    global ftracetcp$id_
    set ftracetcp$id_ [open  xcp$id_.tr  w]
    set tcpTrace_ [set ftracetcp$id_]
    $tcp_ attach-trace [set ftracetcp$id_]
    if { -1 < [lsearch $parameters cwnd]  } { $tcp_ tracevar cwnd_ }
    if { -1 < [lsearch $parameters seqno] } { $tcp_ tracevar t_seqno_ }
}
#--- Command line arguments
proc  set-cmd-line-args { list_args } {
    global argv

    set i 0
    foreach a $list_args {
            global $a
            set $a  [lindex $argv $i]
            puts "$a = [set $a]"
            incr i
    }
}

#-------------- Plotting functions -----------#

# plot a xcp traced var
proc plot-xcp { TraceName nXCPs  PlotTime what } {
    if {[string compare $what "cwnd_"] == 0} {
            exec rm -f xgraph_cwnd.tcp
            set f [open xgraph_cwnd.tcp w]
            set a cwnd
    } else {
            exec rm -f xgraph_seqno.tcp
            set f [open xgraph_seqno.tcp w]
            set a seqno
    }
    puts $f "TitleText: $TraceName"
    puts $f "Device: Postscript"
```

```
    foreach i $nXCPs {
        #the TCP traces are flushed when the sources are stopped
        exec rm -f temp.tcp
      exec touch temp.tcp
        global ftracetcp$i
        if [info exists ftracetcp$i] { flush [set ftracetcp$i] }
        set packetsize [expr 100 * ($i +10)]
      set result [exec awk -v PlotTime=$PlotTime -v what=$what -v s=$packetsize {
            {
                if (( $6 == what ) && ($1 > PlotTime)) {
                        tmp=$7*s
                        print $1, tmp >> "temp.tcp";
                }
            }
        } xcp$i.tr]

        puts "$i : $result"

        puts $f \"$what$i
        exec cat temp.tcp >@ $f
        puts $f "\n"
        flush $f
    }
    close $f
    exec xgraph  -nl -m  -x time -y $what xgraph_$a.tcp &
    return
}

# Takes as input the the label on the Y axis, the time it starts plotting, and the trace file tcl var
proc plot-red-queue { TraceName PlotTime traceFile } {

    exec rm -f xgraph.red_queue
    exec rm -f temp.q temp.a temp.p temp.avg_enqueued temp.avg_dequeued temp.x temp.y
    exec touch temp.q temp.a temp.p temp.avg_enqueued temp.avg_dequeued temp.x temp.y

    exec awk -v PT=$PlotTime {
        {
            if (($1 == "q" && NF>2) && ($2 > PT)) {
                print $2, $3 >> "temp.q"
            }
            else if (($1 == "a" && NF>2) && ($2 > PT)){
                print $2, $3 >> "temp.a"
            }
            else if (($1 == "p" && NF>2) && ($2 > PT)){
                print $2, $3 >> "temp.p"
            }
        }
    }  $traceFile

    set ff [open xgraph.red_queue w]
```

```
    puts $ff "TitleText: $TraceName"
    puts $ff "Device: Postscript \n"
    puts $ff \"queue
    exec cat temp.q >@ $ff
    puts $ff \n\"ave_queue
    exec cat temp.a >@ $ff
    puts $ff \n\"prob_drop
    exec cat temp.p >@ $ff

    close $ff
    exec xgraph  -P -x time -y queue xgraph.red_queue &
}

proc plot-red {varname filename PlotTime} {

    exec rm -f temp.$filename
    exec touch temp.$filename

    set result [exec awk -v PlotTime=$PlotTime -v what=$varname -v file=temp.$filename {
          {
             if (( $1 == what ) && ($2 > PlotTime)) {
                   print $2, $3 >> file ;
             }
          }
    } ft_red_Bottleneck.tr]

    exec xgraph -P -x time -y $filename temp.$filename
}

#-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*- Initializing Simulator -*-*-*-*-*-*-*-*--*-*-*-*-*-*-*-*-#
# BW is in Mbs and delay is in ms

#set-cmd-line-args "seed qType BW nXCPs delay "

set seed   472904
set qType  XCP
set BW     20; # in Mb/s
set nXCPs  3; # Number of flows
set delay  10; # in ms


set ns         [new Simulator]
$ns            use-scheduler Heap
set rtg        [new RNG]
$rtg seed      $seed


set f_all [open out.tr w]
$ns trace-all $f_all
```

```
set  qSize  [expr round([expr ($BW / 8.0) * 4 * $delay * 1.0])];#set buffer to the pipe size

set tracedXCPs       "0 1 2"
set SimStopTime      30
set PlotTime         0



#---------- Create the simulation --------------------#

# Create topology
create-topology2 $BW $delay $qType $qSize $nXCPs 0.0

foreach link $all_links {
   set queue [$link queue]
   switch $qType {
         "XCP" {
               $queue set-link-capacity [[$link set link_] set bandwidth_];
         }
         "DropTail/XCP" {
            $queue set-link-capacity-Kbytes [expr [[$link set link_] set bandwidth_] / 8000];
         }
         default {
               puts "Incorrect qType $qType"
               exit 0
         }
   }
}


# Create sources:
set i 0
while { $i < $nXCPs  } {
   set StartTime [expr [$rtg integer 1000] * 0.001 * (0.01 * $delay) + $i  * 6.0]
   set rcvr_XCP      [new Agent/TCPSink/XCPSink]
   $ns attach-agent  $R1 $rcvr_XCP
   set src$i         [new GeneralSender $i [set n$i] $rcvr_XCP "$StartTime TCP/Reno/XCP"]
   [[set src$i] set tcp_]  set  packetSize_ [expr 100 * ($i +10)]
   [[set src$i] set tcp_]  set  window_     [expr $qSize * 10]
   incr i
}


#---------- Trace --------------------#

# Trace sources
foreach i $tracedXCPs {
         [set src$i] trace-xcp "cwnd seqno"
}
```

```
# Trace Queues
foreach queue_name "Bottleneck rBottleneck" {
   set queue [set "$queue_name"]
   switch $qType {
            "XCP" {
                     global "ft_red_$queue_name"
                     set "ft_red_$queue_name" [open ft_red_[set queue_name].tr w]

                     set xcpq $queue ;#[$queue set vq_(0)]
                     $xcpq attach       [set ft_red_$queue_name]
            }
            "DropTail/XCP" {}
   }
}


#---------------- Run the simulation ----------------------#
proc flush-files {} {
   set files "f_all ft_red_Bottleneck ft_red_rBottleneck"
   global tracedXCPs
   foreach file $files {
      global $file
      if {[info exists $file]} {
         flush [set $file]
         close [set $file]
      }
   }
   foreach i $tracedXCPs {
         global src$i
         set file [set src$i tcpTrace_]
         if {[info exists $file]} {
         flush [set $file]
         close [set $file]
      }
   }
}

proc finish {} {
   global ns

   if {[info exists f]} {
         $ns flush-trace
         close $f
   }

   $ns halt
}

$ns at $SimStopTime "finish"
```

```
$ns run
flush-files

#------------ Post Processing ---------------#
set PostProcess 1
if { $PostProcess } {
   #--- Traced TCPs
   set TraceName "Flows --$qType-QS$qSize"
   plot-xcp      $TraceName  $tracedXCPs  0.0  "cwnd_"
 #   plot-xcp      $TraceName  $tracedXCPs  0.0  "t_seqno_"

#    plot-red-queue  $TraceName  $PlotTime   ft_red_Bottleneck.tr
   plot-red "u" util 0.0
}
```

# Annexe B

Cette annexe rapporte plus de détails sur le fonctionnement de NS, sa programmation TCL langage de commande et son outil de visualisation NAM et contient un échantillon du fichier trace généré par les scripts.

## Network Simulator NS

1- TCL Tool Command Language

Présentation

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

Lancement

Toutes les applications qui utilisent Tcl créent et utilisent un interpréteur Tcl. Cet interpréteur est le point d'entrée standard de la bibliothèque. L'application tclsh constitue une application minimale ayant pour but de familiariser un utilisateur au langage Tcl. Elle ne comporte que l'interpréteur Tcl. On retrouve cet interpréteur dans l'application NS. Lorsque l'on tape ns, une série d'initialisations est faite puis l'application passe en mode interactif. On peut alors entrer les commandes Tcl.

Concepts de base

Chaque commande consiste en un ou plusieurs mots séparés par des espaces ou des tabulations. Le langage n'est pas typé. Tous les mots sont des chaînes de caractères. Le premier mot de la commande est le nom de la commande, les autres mots sont les arguments passés à la commande. Chaque commande Tcl retourne le résultat sous forme d'une chaîne de caractères. Le caractère de retour à la ligne termine une commande et lance son interprétation. Le caractère de séparation de commandes sur une même ligne est ";". Une fois la commande

exécutée et terminée, l'application retourne en mode interactif c'est à dire que l'interpréteur est de nouveau prêt à recevoir une nouvelle commande.

Tcl n'est pas un langage compilé, c'est un langage interprété. Il n'y a pas de grammaire fixe qui traite l'ensemble du langage. Tcl est défini par un interpréteur qui identifie les commandes par des règles d'analyse syntaxique. Seules les règles d'analyse des commandes sont fixées. Tcl évalue une commande en deux étapes: analyse syntaxique et exécution. L'analyse syntaxique consiste à identifier les mots et effectuer les substitutions. Durant cette étape, l'interpréteur ne fait que des manipulations de chaînes. Il ne traite pas la signification des mots. Pendant la phase d'exécution, l'aspect sémantique des mots est traité comme par exemple déduire du premier mot le nom de la commande, vérifier si la commande existe et appeler la procédure de cette commande avec les arguments.

2- TCL/TK

2.1- Introduction

Tcl/Tk (prononcer Teackle Teakey à l'anglaise) est un langage interprété portable sur Mac, PC et station Unix. Tcl (Tool Command Language) est un langage script non typé, et Tk (Tool Kit) est un ensemble d'outils permettant de construire des interfaces graphiques.

*2.1.1- Pourquoi Tcl/Tk ?*

• Tcl/Tk permet très simplement de construire des interfaces graphiques. Sa facilité d'utilisation réduit largement le temps de développement de l'interface graphique d'un logiciel (j'estime le temps de développement Interface/Noyau passé de 70%/30% à 30%/70% grâce à l'utilisation de Tcl/Tk).

• De plus, Tcl est suffisamment puissant pour développer une application de petite taille.

Cependant pour des applications de moyenne et grande taille, Tcl/Tk permet un interfaçage simple avec le langage C. Pour les programmeurs Ada, adatcl permet d'utiliser Tcl/Tk avec Ada.

• La portabilité de Tcl/Tk permet de passer d'une plate-forme de développement à une autre (Mac, PC, station Unix) sans aucune modification du programme! De plus, il est envisagé d'intégrer un interpréteur Tcl/Tk en natif dans certains systèmes d'exploitation.

2.1.2- Philosophie générale

Tcl est un script interprété par la commande tclsh, il peut être vu comme une extension du shell Unix sh. Ce script peut être lancé soit en mode interactif:

>tclsh

% set a 1

Soit en mode autonome en lui donnant un fichier d'entrée:

>tclsh -f monfichier.tcl

Le premier cas sera utilisé pour le développement d'une application, alors que le deuxième cas permettra d'utiliser un programme Tcl comme un programme à part entière.

Tk est une surcouche de Tcl, et est aussi un script interprété, mais par l'interpréteur wish (Widget shell) qui inclue tclsh. Wish permet la création d'objets graphiques (fenêtres, boutons, menus, canvas...), nommés widgets, ainsi que leur gestion, destruction, ...

Comme dans tout shell, une instruction doit s'écrire sur une seule ligne. On peut tout de même couper une ligne sur plusieurs pour plus de commodité, en faisant précéder les retours chariot d'un antislash (\). De plus un bloc, c'est à dire du code entre accolades, peut être écrit sur plusieurs lignes. Bien sûr, chaque instruction du bloc doit porter sur une seule ligne ou plusieurs avec \, ou être un bloc...

2.2- Programmation Tcl :

Pour plus de détail sur les instructions nécessaires pour la compréhension de notre travail voir l'annexe

Utilisation de NS

Nous allons commencer par un exemple simple qui illustre assez bien le comportement de ce système et permet d'en apprendre la terminologie élémentaire. Cet exemple est une simple simulation d'un réseau minimal constitué de deux stations communiquant l'une avec l'autre via une liaison spécialisée.


**Figure : Un réseau constitué de deux nœuds**

Dans la terminologie NS, ce que nous appelons machine s'appelle un nœud. Un nœud peut contenir des agents qui représentent des comportements, par exemple des applications. Une bibliothèque assez complète de composants existe de façon standard. Une propriété intéressante de ce système est son extensibilité. En effet, il est assez facile d'étendre la bibliothèque des comportements, des types de liens, ou de tout autre élément du système en programmant ses propres extensions qui deviennent alors intégrées au système.

Pour traiter ce premier cas, nous allons donc écrire un programme simple. En voici le contenu. Les commentaires (introduits par le caractère #) expliquent le rôle de chaque instruction ou partie de programme.

```
# création d'un simulateur
set ns [new Simulator]
# création du fichier de trace utilisé par le visualisateur et indication à ns de l'utiliser
set nf [open out.nam w]
$ns namtrace-all $nf
# lorsque la simulation sera terminée, cette procédure est appelée pour lancer automatiquement le visualisateur
proc finish {} {
global ns nf
$ns flush-trace
close $nf
exec nam out.nam &
exit 0}
# création de deux noeuds
set n0 [$ns node]
set n1 [$ns node]
# création d'une ligne de communication full duplex entre les noeuds n0 et n1
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
# création d'un agent générateur de paquets à vitesse constante
# paquets de 500 octets, générés toutes les 5 ms
# implantation de cet agent dans le noeud n0
```

```
set cbr0 [new Agent/CBR]
$ns attach-agent $n0 $cbr0
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
# création d'un agent vide, destiné à recevoir les paquets
# il est implanté dans n1
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
# le trafic issus de l'agent cbr0 est envoyé  vers null0
$ns connect $cbr0 $null0
# scénario de début et de fin de génération des paquets par cbr0
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
# la simulation va durer 5 secondes de temps simulé
$ns at 5.0 "finish"
# début de la simulation
$ns run
```

Lors de cette simulation, les résultats sont consignés dans un fichier de trace que l'outil de visualisation NAM va permettre de traiter. Dans notre programme, l'outil de visualisation est appelé directement à la fin de la simulation. Deux éléments intéressants sont proposés à la visualisation : un dessin de la topologie du réseau étudié, et une visualisation dynamique du déroulement du programme dans le temps.

Nous verrons plus loin comment accéder à d'autres informations à partir de cette interface.

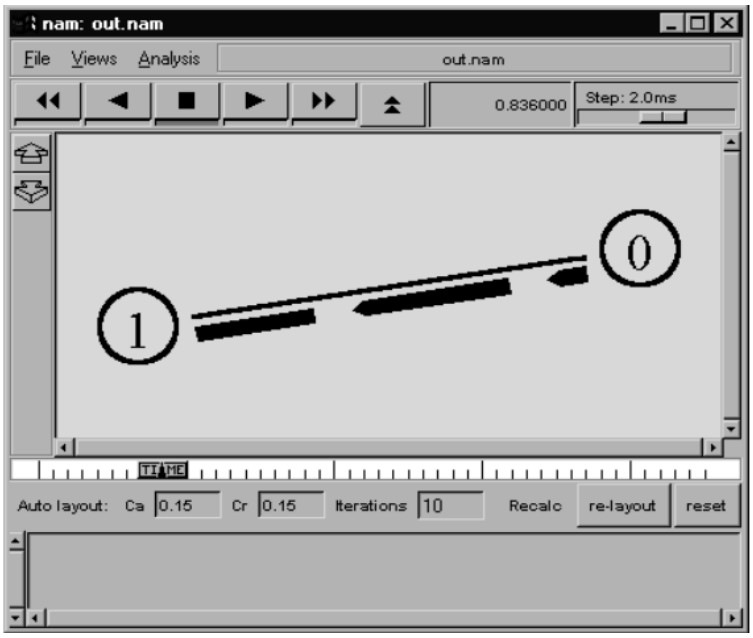



**Figure : Exécution du premier exemple**

Un deuxième exemple va montrer maintenant comment complexifier un peu la topologie du réseau obtenu et rendre plus visible les comportements en construisant un réseau de quatre nœuds avec deux sources de paquets. Lors de la visualisation, on souhaite que les paquets d'une source apparaissent en bleu, les autres en rouge. Voici le texte de ce nouveau programme et la visualisation qui en est obtenue. On remarque la visualisation de la file d'attente de messages qui se forme au nœud 2, puisque les trafics en entrée sont assez proches de la saturation.

```
# création d'un simulateur
set ns [new Simulator]
# création du fichier de trace utilisé par le visualisateur et indication à ns de l'utiliser
set nf [open out.nam w]
$ns namtrace-all $nf
# lorsque la simulation sera terminée, cette procédure est appelée pour lancer automatiquement le visualisateur
proc finish {} {
global ns nf
$ns flush-trace
close $nf
exec nam out.nam &
```

```
exit 0
}
# création de quatre noeuds
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
# création de lignes de communication full duplex
# entre les noeuds
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms DropTail
# création d'agents générateurs de paquets à vitesse constante
# paquets de 500 octets, générés toutes les 5 ms
# implantation de cet agent dans le noeud n0
set cbr0 [new Agent/CBR]
$ns attach-agent $n0 $cbr0
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 set fid_ 1
set cbr1 [new Agent/CBR]
$ns attach-agent $n1 $cbr1
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 set fid_ 2
# création d'un agent vide, destiné à recevoir les paquets
# il est implanté dans n1
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
```

```
# routage des trafics
$ns connect $cbr0 $null0
$ns connect $cbr1 $null0
# surveillance d'une file d'attente du noeud 2 vers le noeud 3
$ns duplex-link-op $n2 $n3 queuePos 0.5
# scénario de début et de fin de génération des paquets par cbr0
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"
# paramétrage des couleurs d'affichage
$ns color 1 Blue
$ns color 2 Red
# la simulation va durer 5 secondes de temps simulé
$ns at 5.0 "finish"
# début de la simulation
$ns run
```

2.2.1- Variables Tcl

En Tcl, les variables sont non typées, et peuvent être, suivant le cas, considérées comme chaînes de caractères (le cas le plus large), listes (ensemble ordonné d'éléments non typés qui peuvent aussi être eux-mêmes des listes), entiers ou flottants. Elles peuvent aussi être des tableaux à une ou plusieurs dimensions. Une même variable peut être considérée de types différents suivant l'opération que l'on fait sur elle.

L'accès à une variable nom_variable se fait par $nom_variable.

L'affectation se fait par la commande set.

```
% set a 12    #la variable a reçoit "12"
% puts a      #affichage du caractère "a" a
% puts $a     #affichage du contenu de la variable a 12
% set a b     #la variable a reçoit la lettre "b"
```

% puts $a  #affichage de la valeur de a b

% puts $b #affichage de la valeur de b

iletaitunebergere

% puts [subst $$a] #affichage de la valeur de la variable contenue dans la variable a,

iletaitunebergere #la fonction subst force l'évaluation le plus loin possible des

#variables. L'emploie des crochets sera expliqué plus loin

**Chaînes de caractères pouvant contenir des espaces:**

% set toto il etait une bergere #utilisation de la fonction set avec trop d'arguments

wrong # args: should be "set varName ?newValue?"

% set toto  "il etait une bergere"  #utilisation correcte de set

% set toto  {il etait une bergere}  #autre utilisation correcte de set

% set a bergere

% set toto "il etait une $a"

% puts $toto il etait une bergere

% set toto {il etait une $a}

% puts $toto il etait une $a

Les guillemets permettent l'utilisation de caractères blancs dans une chaîne, les accolades aussi, mais celles-ci interdisent l'évaluation des variables  éventuelles contenues dans la chaîne.

Il est aussi possible d'utiliser des espaces dans une chaîne de caractères en les faisant précéder d'un anti-slash. Celui-ci permet de considérer le caractère qui le suit comme un caractère normal.

% set toto il\ etait\ une\ $a

% puts $toto

il etait une bergere

Ainsi si une chaîne de caractères doit contenir des guillemets ou des accolades:

% set toto \{il\ etait\"\ une\nbergere # \n force un retour chariot

% puts $toto

{il etait" une bergere

Remarque: la concaténation de chaînes peut se faire de manière automatique:

% set Aut "un aut"

% set Re re

% set toto "un train peut en cacher $Aut$Re"

un train peut en cacher un autre

**Listes**

Tcl/Tk permet de tout considérer comme une liste, par exemple:

% set toto "il etait une bergere"

% llength $toto #fonction retournant le nombre d'éléments de la liste en argument

Il est possible de construire explicitement des listes par l'utilisation du constructeur list ou par l'utilisation d'accolades:

% set toto {il {etait une bergere}} #liste contenant deux éléments

% puts $toto

il {etait une bergere}

% llength $toto  #nombre d'éléments de la liste

% llength [lindex $toto 1]  #nombre d'éléments du deuxième élément de la liste: celles-ci sont indexées à partir de 0

% set toto [list il [list etait une bergere]]  #donne le même résultat qu'en utilisant les #accolades. Cependant, permet l'utilisation de variables dans la fabrication de la liste, ce que les accolades empêchent.

La fonction la plus intéressante pour le parcours de liste est  foreach, qui permet d'effectuer un ensemble de commandes sur (ou a partir de) l'ensemble des éléments d'une liste:

%set maliste "Il etait une bergere"

%foreach mot $maliste {

 puts $mot

}

Il

etait

une

bergere

La propriété intéressante de  foreach est qu'elle permet de traiter les  éléments de l'ensemble par couple, triplet, etc...

```
%foreach {motimpair motpair} $maliste {
 puts $motpair  #seul un mot sur deux est affiché
}
```

etait

bergere

**Entiers et flottants**

Comme Tcl/Tk considère que tout est chaîne de caractères, il est nécessaire de lui spécifier qu'une opération arithmétique est désirée:

```
% set a 23+12
23+12
% set a [expr 23+12]   #utilisation de la fonction expr pour des calculs arithmétiques 35
```

La différence entre entiers et flottants se fait de manière implicite suivant le type des opérandes:

```
% expr 23/3     #calcul entier car les deux arguments sont des entiers 7
% expr 23.0/3   #un des arguments est un réel => calcul flottant 7.66667
```

**Tableaux**

Les tableaux sont des variables possédant des entrées:

```
% set tableau (oncle) Tom   #l'entrée oncle du tableau reçoit la chaîne "Tom"
% puts $tableau (oncle)   #référence au contenu d'une variable de type tableau Tom
% set i oncle
% puts $tableau($i) #encore une référence, mais cette fois ci  par une variable Tom
```

On peut aussi créer des tableaux à plusieurs dimensions, dans ce cas les noms d'entrées sont séparés par des virgules :

% set tab2(case,enpaille) oncle  #l'entrée référencée par case et empaille reçoit "oncle"

% puts $tableau($tab2(case,enpaille))  #affichage de l'entrée oncle de tableau Tom

Limitation: une variable de type tableau ne peut pas être créée dans une variable autre et vice-versa:

% set toto 12  # toto est donc connu par l'interpréteur comme une variable chaîne

% set toto(a) 10  # assignation incorrecte

can't set "toto(a)": variable isn't array

% unset toto  # le contenu de la variable est libéré, de plus la variable toto est "oubliée"

% set toto(a) 10  # assignation maintenant correcte car toto est une nouvelle variable

2.2.2- Structures de programme

**Conditionnelles**

Une conditionnelle s'écrit sous la forme:

if condition bloc

ou bien

if condition bloc1 else bloc2

ou bien

if condition bloc1 elseif condition2 bloc2 ... else bloc n

**Remarque :** cette instruction s'écrit sur la même ligne, on ne peut insérer des sauts de lignes qu'à l'intérieur des accolades, ainsi:

% set a 4

% if {$a > 0} {

puts $a} #incorrect car le else n'est pas sur la même ligne que le bloc

else {puts -$a}

Conseil: toujours écrire les tests sous la forme suivante:

if {condition} {

  bloc

} else {

  bloc

}

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

Attention, les opérations mathématiques sont interdites sur les chaînes de caractères, en particulier il faut remplacer:

$chaîne1 == $chaîne2

Par [string compare $chaîne1 $chaîne2]==0.

String compare est une fonction qui renvoie 0 si les deux chaînes comparées sont identiques.

La bibliothèque  string regroupe un nombre appréciable de commandes de traitement de chaînes de caractères.

**Boucle "pour"**

Un boucle "pour" s'écrit sous la forme:

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

Exemple d'utilisation de la boucle pour:

```
% for {set i 1} {$i<=10} {incr i} {
puts $i
}
```

Conseil d'écriture de la boucle pour:

```
for {initialisations} {condition} {incrémentations} {
bloc
}
```

**Boucle "tant que"**

Une boucle "tant que" s'écrit sous la forme:

```
while condition bloc
```

Comme toute instruction Tcl/Tk, cette commande s'écrit sur la même ligne. Ainsi on ne peut mettre des sauts de lignes que dans des blocs délimités par des accolades:

Exemple d'utilisation:

```
% set a 1
% while {$a>0} {
puts $a
incr a -1 #décrémentation de a
}
```

Conseil d'écriture d'une boucle "tant que":

```
while {condition} {
bloc
}
```

**Branchement à choix multiples**

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

```
switch chaîne modèle1 bloc1 ... modèle n bloc n
```

ou bien

**switch** chaîne modèle1 bloc1 ... modèle n bloc n default blocdefault avec blocdefault qui est exécuté si aucun modèle ne correspond à la chaîne.

Exemple:

```
% switch $a {
toto {puts "c' est toto"}
titi {puts "c'est titi"}
default {puts "je ne le connais pas"}
}
```

Conseil d'écriture:

```
switch chaîne {
modèle1 {
bloc1
}
modèle2 {
bloc2
}
 ...
}
```

**Fonctions**

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

**proc** nom paramètres corps

En fait elle s'écrira le plus souvent:

**proc** nom_proc {param1 param2 ... param n} { bloc }

Les fonctions sont toutes censées renvoyer une valeur (qui peut être une chaîne vide).

Dans le cas où une instruction return est rencontrée lors de l'évaluation de la fonction, son évaluation est stoppée et la valeur suivant le return est renvoyée. Dans le cas où il n'y a aucune instruction return, le résultat de la dernière commande exécutée par la fonction est renvoyé par celle-ci.

La valeur d'une fonction est retournée lorsque celle-ci est appelée. L'appel d'une fonction se fait soit de manière procédurale (on ne conserve pas le résultat):

% nom_procparam1 param2 ... param3 #le résultat de la fonction nom_proc est ignoré bien que la fonction soit interprétée

Soit l'appel se fait par la mise entre crochets de l'appel:

% set a [nom_proc param1 param2 ... param n] #la variable a contiendra le résultat retourné par la fonction appelée.

Exemple:

```
% proc racines {a b c} {
#calcule les racines du polynôme ax2+bx+c
set delta [expr pow($b,2) - (4*$a*$c)]
if {$delta==0} {
return [expr -$b/(2.0*$a)]
 } elseif {$delta>0} {
return [list [expr (-$b+sqrt($delta)) / ($a*2.0)] [expr (-$b-sqrt($delta)) /($a*2.0)]]
} else {
return ""
}
}
% puts [racines 1 2 1]
-1.0
% puts [racines 2 -5 2]
2.0 0.5
```

L'exécution des scripts génère des fichiers trace qui contiennent les données numérique qui sont utilisées soit pour la visualisation de la simulation avec NAM ou pour tracer les graphes.

Ces fichiers contiennent trois colonnes : le temps de la mesure, le nombre de paquets perdus et la bande passante.

```
10.5 0 0.0
11 0 0.20799999999999999
11.5 125 0.40000000000000002
12 249 0.40000000000000002
12.5 318 0.58240000000000003
13 396 0.083199999999999996
13.5 396 0.0
```

Figure : échantillon du fichier trace

## Résumé

Pour remédier aux problèmes de congestion, délai de transmission et perte de paquet, des mécanismes de gestion de files d'attente sont implémentés dans les routeurs pour apporter des solutions aux problèmes qui se produisent lorsque le flux dans le réseau est supérieur aux capacités des ressources.

Dans le but de modéliser le flux dans un réseau afin d'analyser les performances de ce dernier, nous avons simulé plusieurs types de files d'attente tel que FIFO (First In First Out), SFQ (Stochastic Fair Queue), RED (Random Early Detection), le système M/M/1 et XCP (eXplicit Congestion control Protocol) le protocole de contrôle de congestion avec NS (Network Simulator). Nous avons ensuite analysé avec précaution les résultats de chaque simulation.


**Mots-clés :** Gestion de files d'attente, qualité de service, modélisation, simulation, NS 2, FIFO, SFQ, RED, M/M/1, protocole XCP


## Abstract

To solve the problems of congestion, transmission delay and packet loss, management mechanisms of queues are implemented in routers to provide solutions that occur when the flow in the network exceeds the resources capacity.

The target to model the flow in a network to analyze the performance of the latter, we simulated several types of queues as FIFO (First In First Out), SFQ (Stochastic Fair Queue), RED (Random Early Detection), the system M/M/1 and XCP (Explicit Congestion control Protocol) congestion control protocol with NS (Network Simulator). Then we have carefully analyzed the results of each simulation.

This allowed us to understand the main causes of the phenomenon of congestion, To manage this phenomenon with different queues and use a solution to congestion as the XCP protocol. Then understand the function the systems of queues and their utility in the network.


**Keywords:** Management queues, quality of service, simulation, NS2, FIFO, SFQ, RED, M/M/1, XCP protocol


## ملخص

لحل مشاكل الازدحام الشبكة (congestion)، تأخر البث و فقدان البيانات يتم تنفيذ آليات تدبير سلسلة الانتظار (الطابور) {file d'attente } على مستوى جهاز توجيه البيانات هده المشاكل تحدث لما تدفق الشبكة يتجاوز قدرات موارد الشبكة.

لهدف ضبط نموذج تدفق الشبكة لتحليل فعاليات هده الأخيرة قمنا تقليد (simuler) أنواع مختلفة من سلاسل الانتظار مثل FIFO، SFQ، RED، M/M/1 و أيضا بروتوكول XCP باستعمال برنامج التقليد Network Simulator NS. فيما حللنا بتمعن نتائج كل تقليد (simulation).


كلمات الأساسية ادارة سلاسل الانتظار ، نوعية الخدمة ، التقليد ، NS2، FIFO SFQ RED M/M/1 XCP