Simulation hybride des réseaux IP-DiffServ-MPLS multi-services sur environnement d'exécution distribuée

David Gauchard

DOCTORAT DE L'UNIVERSITE TOULOUSE III
PAUL SABATIER

Spécialité : **SYSTEMES INFORMATIQUES**

GAUCHARD David

Laboratoire :

LABORATOIRE D'ANALYSE ET
D'ARCHITECTURE DES SYSTEMES
(LAAS-CNRS)

Titre de la thèse :

Simulation Hybride des Réseaux IP-DiffServ-MPLS Multi-services sur Environnement d'Exécution Distribuée

Soutenance le : 25 avril 2003 au LAAS CNRS à Toulouse

Directeur de thèse : **GARCIA Jean-Marie**
Co-directeur de thèse : **MONTEIL Thierry**

JURY :

Gérard AUTHIE (LAAS-CNRS)
Pierre Bacquet (DELTA PARTNERS - groupe ANITE)
Jean-Marie GARCIA (LAAS-CNRS)
Thierry MONTEIL (LAAS-CNRS)
Philippe NAIN (INRIA)
Jean-Gabriel REMY (CEGETEL)
Catherine ROUCAIROL (PRISM)

Remerciements

Les travaux présentés dans ce manuscrit sont l'aboutissement de quatre années et demi d'études effectuées au Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS (LAAS-CNRS) au sein du groupe de recherche RST (Réseaux et Systèmes de Télécommunications).

Je tiens à exprimer toute ma reconnaissance à Messieurs Jean-Claude Laprie et Malik Ghallab, successivement directeurs du laboratoire, pour m'avoir accueilli au LAAS et pour avoir mis à ma disposition tous les moyens nécessaires à la réalisation de ces travaux. Je voudrais également signaler ici l'exceptionnelle qualité des moyens informatiques et logistiques mis à disposition des chercheurs dans ce laboratoire.

J'adresse tous mes remerciements à Messieurs Jean-Marie Garcia, chargé de recherche CNRS et Thierry Monteil, Maître de Conférences à l'INSA de Toulouse, respectivement directeur et co-directeur de cette thèse, pour m'avoir encadré et conseillé durant les années que j'ai passées au LAAS. Je tiens ici à leur témoigner mon amitié, ainsi qu'à l'équipe de recherche dans laquelle j'ai été intégré.

Je remercie également Monsieur Jean-Michel Parot, PDG de la société Delta-Partners qui a co-financé cette thèse, pour sa confiance dans le travail que j'ai réalisé.

Je suis très particulièrement reconnaissant envers Monsieur Philippe Nain, Directeur de Recherche INRIA, Monsieur Jean-Gabriel Remy, Directeur Scientifique SFR et Madame Catherine Roucairol, Professeur à l'Université de Versailles et Chargée de Mission à la Direction de la Recherche qui m'ont fait l'honneur d'avoir accepté la tâche d'être les rapporteurs de cette thèse.

Enfin, je remercie Gérard Authié, Professeur à l'Université Paul Sabatier, actuel directeur du groupe de recherche RST, pour avoir accepté de présider le Jury de soutenance, et Monsieur Pierre Bacquet, directeur technique de la société Delta-Partners, en sa qualité de membre du Jury, et avec lequel ce fut un plaisir de collaborer tout au long de ce travail.

Je n'oublierai pas les autres membres de l'équipe : Charly et ses idées, avec lequel j'ai eu de nombreux échanges, et en particulier Olivier Brun, Chargé de Recherche CNRS pour ses incessants conseils éclairés et la très précieuse aide qu'il m'a apportée sur la correction du manuscrit. Je leur témoigne ici toute mon amitié.

Je dois également d'exprimer ma gratitude au service Informatique et Instrumentation, dans lequel j'ai été intégré avant la fin de ces quatre années et demi. D'une part les membres de ce service, qui sont ceux qui assurent l'exceptionnelle qualité de l'environnement de travail informatique au LAAS, et d'autre part son Directeur Marc Vaisset qui m'a autorisé à terminer ce travail de recherche avant de prendre mes fonctions.

Je remercie également toute ma famille pour son soutien moral, et également pour son impatience à vouloir assister à la soutenance, et particulièrement mon amie Vera qui m'a accompagné dans ce travail et qui a su supporter avec bienveillance mon humeur des derniers moments.

Résumé

La technologique utilisée dans les réseaux de télécommunication à commutation de paquets est en développement permanent. La tendance actuelle est orientée vers la mutualisation des services voix et données, de leurs infrastructures et des techniques associées (réseaux IP/MPLS, QoS dans Internet).

Les méthodes classiques utilisées pour l'évaluation de performances de ces réseaux sont la modélisation analytique et la simulation événementielle. La simulation événementielle requiert des temps de calcul prohibitifs, tandis que les modèles issus des méthodes analytiques manquent parfois de précision.

Le travail présenté dans cette thèse définit un cadre de modélisation appelé Simulation Hybride Distribuée qui combine de manière rigoureuse la théorie différentielle du trafic et la simulation événementielle.

Dans une première technique, les ressources du réseau sont partitionnées en domaines dont certains sont simulés par événements, et les autres sont modélisés par des équations intégrées numériquement. La seconde technique proposée permet la circulation de trafics simulés par événements sur les ressources du réseau modélisées par des équations.

La simulation hybride permet ainsi de prolonger la modélisation analytique au-delà de ses limites théoriques. Elle permet également de concevoir des modèles d'évaluation de performances très généraux et de développer des logiciels plus performants en temps de calculs.

Un prototype de simulateur hybride a été conçu. Il permet de modéliser précisément des routeurs Internet Diffserv et MPLS, le protocole TCP ainsi que diverses sources de trafic multimédia (Audio, Vidéo).

Pour réduire les temps de calcul, le simulateur peut être parallélisé. En ce sens, un nouveau noyau de communication de l'environnement de parallélisme LANDA a été conçu. Il offre une bibliothèque à la norme MPI et permet d'utiliser simultanément et efficacement (latence, bande passante) plusieurs media de communication haut-débit ainsi que la mémoire commune SMP.

Mots-clés :

Théorie différentielle du trafic, files d'attente, simulation événementielle, simulation hybride distribuée, réseaux IP/MPLS multiservices, sources de trafic multimédia, modèles TCP, parallélisme haut-débit, passage de messages.

Introduction

Depuis plus d'une décennie, nous assistons à une évolution permanente des technologies de transport de données numériques. Par la démocratisation du réseau Internet, le spectre des services proposés à un public en croissance permanente ne cesse de s'agrandir. Les réseaux actuels de transport de données utilisent la technologie de commutation de paquets, tandis que les réseaux de télécommunications (ainsi que les réseaux mobiles) utilisent la commutation de circuits plus adaptée aux exigences des services temps réels.
Le déploiement des réseaux à haut débit et le développement des technologies MPLS permettent la conception de réseaux multiservices capables de transporter aussi bien les flux de données que les flux temps réel (voix, vidéo). Ces nouveaux réseaux seront capables de satisfaire les exigences différentes des différents flux qu'ils transportent.

L'IETF a ainsi défini le nouveau protocole MPLS avec deux objectifs principaux :

- Permettre un acheminement rapide des paquets IP en remplaçant la fonction de routage par une fonction de commutation beaucoup plus rapide. Ceci est possible grâce à la substitution des tables de routage classiques par des matrices de commutation beaucoup plus petites,
- Faciliter l'ingénierie réseau en fournissant aux opérateurs la maîtrise de l'acheminement des données, qui était très complexe avec les protocoles de routage classiques comme OSPF.

Le protocole MPLS, basé sur le paradigme de changement de label, dérive directement de l'expérience acquise avec les réseaux ATM (canaux virtuels, chemins virtuels). Les grandes innovations de MPLS par rapport au routage IP traditionnel sont la manipulation de tables de routage de bien plus petites tailles, des temps de commutation extrêmement rapides, des chemins (LSP) par classe de service, l'ingénierie du trafic, une meilleure maîtrise de la qualité de service, des mécanismes de reroutage en cas de panne. MPLS, associé à des routeurs implémentant les mécanismes de différenciation de service (DiffServ) permet de traiter les flux séparément selon leur caractéristiques (type de service, type d'application) par un routage plus précis et par le traitement différencié des paquets. C'est cette double fonctionnalité qui permet une meilleure utilisation des ressources de bout en bout afin d'assurer la qualité de service requise par les applications.

L'ensemble des mécanismes mis en jeu dans un réseau IP-DiffServ-MPLS sont extrêmement complexes. Un administrateur réseau ou un opérateur aura de grandes difficultés à dimensionner ces ressources, à ajuster les différents paramètres de gestion de la différentiation de service, à proposer une ingénierie du trafic optimale, à garantir à des applications des contrats de qualité de service etc. De ce fait une estimation précise des performances de tels réseaux est un élément clé d'aide à la décision et à la planification.

La modélisation des réseaux à commutation de paquets et à commutation de circuits a été largement étudiée au cours des trente dernières années. D'autre part, les nouveaux mécanismes de gestion de qualité de service dans les routeurs du monde IP ont également conduit à l'étude de nouveaux modèles de trafic. Concernant les outils de modélisation à proprement parlé, les simulateurs «ns» (pour Network Simulator de l'université de Berkeley), Opnet et NetQUAD sont les plus couramment utilisés (resp. en milieu académique, industriel et par opérateurs). Les outils comme Wandl (pour IP-MPLS) ou NetScene (SDH, ATM) sont quant à eux dépourvus de modèles précis d'évaluation de performance.

Le travail présenté dans cette thèse est centré sur la conception et la réalisation d'un simulateur général de files d'attente et de réseaux à commutation de paquet de type IP intégrant les mécanismes

de différentiation de services et la commutation MPLS (IP-DiffServ-MPLS). Résoudre l'ensemble des problèmes posés par l'évaluation de performance de tels réseau est une tâche très complexe. En effet, les modèles utilisés doivent être :

- suffisamment précis pour évaluer correctement la qualité de service de chaque trafic dans un grand réseau,
- capables de représenter la réponse transitoire du système à des perturbations du trafic et le fonctionnement en boucle fermée de certains protocoles (TCP, OSPF...),
- capables de prendre en compte les caractéristiques statistiques spécifiques du trafic généré par chaque source (voix, vidéo, données...),
- suffisamment rapides pour être utilisés dans des boucles d'optimisation itératives pour la planification de grands réseaux de télécommunications.

Les approches traditionnelles dans ce domaine sont le prototypage, la modélisation analytique et la simulation événementielle. Le prototypage est en général hors de coût même pour des réseaux de taille modeste. La modélisation analytique, basée sur la théorie des files d'attente, est une approche puissante mais dont la capacité à représenter finement certains mécanismes complexes, et notamment le comportement réactif des protocoles, reste limitée. Par conséquent, une modélisation analytique exacte ayant toutes les propriétés précédentes semble aujourd'hui complètement inenvisageable compte tenu de nos connaissances théoriques. En particulier, il est bien connu qu'une modélisation analytique des phénomènes transitoires dans les réseaux de files d'attente est très complexe et peu de résultats généraux existent dans ce domaine ([ABA87] [KEL85] [LEE93] [MUR88] [SAA60] [TAK62] [WAN99]). Enfin, les modèles de simulation à événements discrets n'ont pas ces limitations, mais requièrent des temps de calcul prohibitifs pour des réseaux de tailles réelles.

Pour dépasser ces limitations, l'approche adoptée consiste à construire un modèle fluide du réseau, basé sur une approximation des équations différentielles exactes gouvernant le trafic moyen transitoire de chaque flot de communication dans le réseau. Cette approximation est obtenue en utilisant la notion de trafic fictif. Ce type de modélisation, qui a été introduit dans le logiciel commercial de planification de réseaux NetQUAD, a été beaucoup utilisé pour les réseaux à commutation de circuits (téléphonie) et a montré à la fois sa précision et sa rapidité.

Une caractéristique intéressante de cette modélisation différentielle est qu'elle offre un cadre théorique global permettant de combiner des modèles analytiques et des modèles de simulation à événements discrets. Ce nouveau concept, appelé simulation hybride, se révèle bien adapté à la modélisation des grands systèmes complexes de télécommunications et peut être efficacement parallélisé, de manière à réduire considérablement les temps d'exécution. L'approche de simulation hybride a été développée dans le cadre du projet européen «$STAR_1$» qui avait pour objectif de porter la première partie du code de l'environnement de simulation de réseaux NetQUAD sur une plate-forme HPCN en utilisant le paradigme de l'échange de messages [STR99].

Cette thèse est organisée de la façon suivante. Dans une première partie nous détaillerons l'ensemble les mécanismes utilisés dans le réseau Internet ou plus généralement dans les réseaux IP-DiffServ-MPLS. Le chapitre II rappelle quelques résultats de la théorie différentielle du trafic qui constitue la base de la simulation hybride. Des résultats sur la modélisation analytique du paradigme GPS sont également présentés. Le chapitre III est consacré à la modélisation des sources de trafic multimedia. Nous y présentons également quelques résultats d'agrégations de sources, intéressants pour la simulation hybride. TCP étant le protocole de transport principalement utilisé par les trafics du réseau Internet, nous détaillons son fonctionnement dans le chapitre IV, et également son

implémentation pour la simulation hybride. De plus, nous y proposons une nouvelle approche différentielle dynamique représentant une source TCP. Deux modèles fluides issus de travaux récents représentant les agrégations de sources TCP sont également abordés. La simulation hybride est explicitée dans le chapitre V. Ce chapitre présente le modèle de simulation hybride que nous avons développé et utilisé dans simulateur DHS (*Distributed Hybrid Simulator*), qui a été conçu et développé dans le cadre de cette thèse. Les modèles de trafic utilisés dans DHS sont les trafics multimedia et TCP.

Afin de pouvoir réaliser des simulations rapides, le simulateur peut être parallélisé en utilisant un environnement d'exécution multi-processeurs. L'évolution des techniques de communication numérique ne concernent pas seulement les réseaux de télécommunication grande distance, mais aussi les réseaux locaux. De multiples technologies propriétaires pour l'interconnexion très haut débit d'ordinateurs « grand public » ont également été développées ces dernières années et permettent l'interconnexion de ces ordinateurs en grappes (*clusters*). Elles permettent également l'interconnexions de grappes en grilles (*grids*). Les technologies employées sont nombreuses, et souvent hétérogènes en ce qui concerne les grilles. L'exécution d'applications parallèles sur ces architectures suppose l'utilisation d'environnements d'exécution distribuée. Pour des raisons de performance, ceux-ci sont souvent limités par l'utilisation d'un seul type de réseau d'interconnexion locale. Le chapitre VI est consacré à ces environnements. Plus précisément, ce travail concerne une re-conception du noyau de communication de l'environnement LANDA (*Local Area Network for Distributed Applications*), rebaptisé LANDA-HSN (*High Speed Network*).

Table des matières

Chapitre V - Simulateur hybride 123

Chapitre I - Les réseaux IP Multiservices et MPLS

I.1 Introduction

Les protocoles Internet constituent la famille la plus populaire de protocoles de systèmes ouverts non propriétaires car ils peuvent être utilisés pour communiquer entre n'importe quel ensemble de réseaux interconnectés et sont aussi bien adaptés aux communications dans les réseaux locaux que dans les réseaux à grande échelle. Les plus connus de ces protocoles sont IP (*Internet Protocol*) [RFC791] et TCP (*Transmission Control Protocol*) [STE94][RFC793][RFC2001]. Les protocoles IP non seulement traitent des couches basses mais incluent aussi des applications comme le courrier électronique (*e-mail*), l'émulation de terminal (*telnet*), le transfert de fichiers (*ftp*), ou la «navigation internet» (*World Wide Web*).
Les protocoles IP ont été développés dans les années 70 quand l'agence américaine DARPA (*Defense Advanced Research Projects Agency*) voulait mettre en place un réseau à commutation de paquets permettant la communication de systèmes hétérogènes entre plusieurs institutions de recherche. La DARPA finança ainsi l'Université de Stanford (Bolt, Beranek et Newman, BBN). Le résultat de cet effort de développement fut le protocole IP à la fin des années 70.
TCP/IP fut introduit ensuite dans Unix BSD (*Berkeley Software Distribution*) et amena la brique de base de ce qu'est actuellement l'Internet et le «*Web*». Les documentations et normalisations de l'Internet sont consignées dans des RFC (*Request For Comments*). Par exemple, les améliorations des protocoles sont consignées dans de nouvelles RFC en particulier lorsque de nouvelles technologies sont mises en place.

Dans un premier temps, nous développons quelques éléments essentiels à la compréhension du fonctionnement d'un réseau IP : l'adressage IP, les protocoles UDP (mode non-connecté) et TCP (mode connecté), la hiérarchie des domaines, le concept de système autonome et les protocoles de routage intra et inter-domaines.
Nous présentons ensuite les évolutions majeures d'IP pour la mise en place d'applications multiservices nécessitant des garanties de qualité de service. En particulier seront développés les principes de Intserv (*Integrated Services* : services intégrés) et Diffserv (*Differentiated Service*s : services différenciés), les mécanismes de gestion des tampons, le protocole MPLS (*Multi-Protocol Label Switching*), ainsi que l'adaptation du protocole de routage OSPF (*Open Shortest Path First*)[RFC1247] à l'ingénierie des trafics (OSPF-TE: *Traffic Engineering*).
Nous exposerons enfin la problématique posée par l'évaluation des performances de ces réseaux.

I.2 Quelques éléments sur IP

IP est un protocole qui permet l'adressage des machines et le routage des paquets de données [RFC791]. Il correspond à la couche 3 (réseau) de la hiérarchie des couches ISO. Son rôle est d'établir des communications sans connexion de bout en bout entre des réseaux, de délivrer des trames de données (*datagram*) « au mieux (*best effort*) », et de réaliser la fragmentation et le réassemblage des trames pour supporter des liaisons n'ayant pas la même MTU (*Maximum Transmission Unit*, c'est-à-dire la taille maximum d'un paquet de donnée).

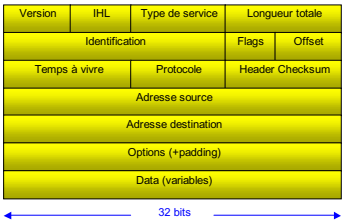
I.2.1 L'adressage IP

Un des éléments essentiels d'IP est la couverture mondiale qu'assure le protocole. Ceci est fait grâce à une gestion d'adresses uniques. L'adressage IP permet d'identifier de façon unique une interface dans une machine connectée à un réseau. Une machine ayant plusieurs interfaces est communément appelée un routeur. Elle est capable de recevoir et de renvoyer des paquets de données par le biais de différentes interfaces. Elle utilise pour cela une table de routage qui, dans sa version la plus simple, contient un certain nombre d'adresses internet complètes ou sous forme de masques (liste non exhaustives), l'interface à utiliser pour atteindre chacune d'elles, et une ou plusieurs interfaces par défaut vers lesquelles envoyer les paquets d'adresse inconnues. Cette correspondance peut être enrichie d'un coût, appelé aussi métrique, servant aux protocoles de routage que nous verrons plus loin.

I.2.2 Format du paquet IP

Un paquet IP (figure 1) est composé d'une entête et de données.

Figure 1: **Format du paquet IP**



L'entête contient des champs. Leurs descriptions sont les suivantes :

- *Version :* indique la version d'IP,
- *IP Header Length (Longueur d'entête IP) :* indique la taille du datagram header en mots de 32 bits,
- *Type de Service* : constitué de trois bits, spécifie comment un protocole de couche plus haute utilise les données et indique quel effort est à faire pour le traitement du paquet (délai, débit, fiabilité),
- *Longueur Totale :* spécifie la longueur totale en octets du paquet IP (entête et données),
- *Identification :* contient un entier qui identifie la trame actuelle. Ce champ est nécessaire pour le réassemblage des fragments de la trame,
- *Flags (drapeaux) :* est constitué de trois bits. Les deux bit de poids plus faible contrôlent la fragmentation. Le premier bit indique si le paquet peut être fragmenté, le deuxième bit indique si le paquet est le dernier d'une série de paquets fragmentés. Le troisième bit n'est pas utilisé,
- *Offset (déplacement) :* Dans le cas d'une fragmentation, ce champ indique la position du fragment relative au début des données dans le datagram original. Ceci permet au processus IP destinataire de reconstruire proprement le datagram,
- *Temps à Vivre :* maintient un compteur qui est décrémenté d'une unité à chaque traversée d'un routeur. La trame est éliminée lorsque ce compteur devient nul. Ceci permet de protéger le réseau contre les erreurs de routage (cycles),
- *Protocole :* indique le protocole de couche supérieure qui est à l'origine du paquet,

• *Header Checksum (Vérificateur d'entête)* : permet de garantir l'intégrité de l'entête,

• *Source Address (Adresse de source) :* permet de spécifier le noeud émetteur,

• *Destination Addres (Adresse de destination) :* permet de spécifier le noeud récepteur,

• *Options :* permettent à IP de supporter plusieurs options dont celle de sécurité,

La zone *Data* (Données) contient les données utiles (provenant des / à transférer aux couches supérieures).

I.2.3 Format de l'adresse IP

Dans la version 4 du protocole IP (*IPv4*), une adresse est codée sur 32 bits et associée à une interface physique. Un point important de l'adressage IP est qu'il est hiérarchique : une adresse IP est constituée d'au moins deux parties, l'adresse du domaine auquel appartient l'interface et l'adresse de l'interface dans le domaine. L'adresse du domaine est une adresse officielle, qui lui permet d'être connue dans le réseau Internet (c'est-à-dire mondialement). L'adresse de l'interface dans le domaine peut être ou non subdivisée en une adresse de sous-domaine et l'adresse de l'interface dans le sous domaine, et ceci autant de fois que les 32 bits le permettent.

La première partie (adresse de domaine) doit être déclarée au centre d'information du réseau Internet (*InterNIC : Internet Network Information Center*). Il existe trois classes d'adresse de domaine, appelées A, B et C qui déterminent les tailles en bits de l'adresse du domaine. Elles sont respectivement de 8, 16 et 24 bits, ce qui permet d'avoir respectivement 2^{24}, 2^{16} et 2^8 adresses d'interfaces possibles. La classe d'une adresse est déterminée par les 2 bits de poids fort de l'adresse.

Une adresse IP est généralement représentée par quatre entiers de 8 bits séparés par des points (x.y.z.w).

Certaines adresses de domaines de classe A sont réservée, et d'une manière générale les valeurs *0* et 2^n*-1* sont réservées, *n* étant la taille en bits de la partie de l'adresse que l'on considère (domaine, sous-domaine, interface). Cela laisse un nombre de valeurs possibles de 2^n*-2* pour chaque partie.

I.2.4 Protocoles de base d'IP

Citons quelques protocoles qui sont nécessaires au fonctionnement de base des réseaux IP :

• ARP (*Address Resolution Protocol*) : Il permet d'associer une adresse d'équipement physique (interface) à une adresse IP pour coopérer avec les couches inférieures [RFC826],

• ICMP (*Internet Control Message Protocol*) : Il permet le transport de messages relatifs au bon fonctionnement d'internet, et concerne principalement les informations de routage,

• DNS (*Domain Name Service*) : Ce protocole permet la résolution d'adresses nommées (FQDN - *Fully Qualified Domain Name*) en adresses IP. Par exemple il permet d'obtenir la réponse 195.83.132.133 à la question « *www.laas.fr* »,

Citons également TCP et UDP. Ils sont très largement utilisés, et décrits en détails dans les paragraphes suivants. Ces deux protocoles permettent le transport de données, via IP, entre deux applications exécutées sur les hôtes correspondants aux adresses source et destination de l'entête du paquet IP.

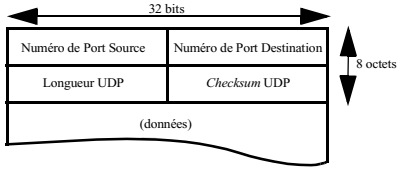
I.3 Les Protocoles UDP et TCP

I.3.1 UDP

UDP (*User Datagram Protocol*) est un protocole de communication de niveau ISO 4 (transport) sans connexion : Cela signifie qu'il n'y a aucune garantie qu'une trame UDP émise arrive à destination. Les trames UDP sont encapsulées par la couche réseau IP sous-jacente. Elles peuvent être de longueur quelconque, la couche IP se charge de leur fragmentation et de leur réassemblage de façon transparente.

La qualité de service offerte par UDP est la même que celle fournie par IP. UDP ajoute cependant quelques informations importantes à l'entête, outre la longueur des données utiles et un checksum : ce sont les numéros de port source et de port destination (figure 2). Ces numéros de port permettent de différencier plusieurs connexions ou services différents entre deux extrémités identiques dans le réseau IP, afin de pouvoir multiplexer les connexions entre deux mêmes machines.

Figure 2: **Structure d'une trame UDP**



UDP est principalement utilisé dans les réseaux locaux (dans lesquels la probabilité de perte d'un paquet IP est moindre) ce qui justifie la non-nécessité de connexion virtuelle (par opposition à TCP). Les services l'utilisant sont généralement NFS (*Network File System*), et NIS (*Network Information Service*), respectivement pour le partage de fichiers et la centralisation d'informations relatives aux réseaux. Ils sont en effet conçus pour fonctionner en mode déconnectés. De fait, ces services doivent détecter et éventuellement corriger eux-même les erreurs de transmission.
Le protocole UDP sert aussi à transporter des données autorisant des pertes mais ayant des contraintes fortes sur les délais. En effet un protocole en mode connecté procède à des retransmissions en cas de perte, ce qui peut considérablement augmenter le délai de bout en bout. Par exemple le protocole RTP (*Real-Time Protocol*) s'appuie sur UDP pour transporter des données de type communication audio.

I.3.2 TCP

I.3.2.1 Généralités

TCP/IP (*Transmission Control Protocol on IP*) est un protocole de transport. Ce protocole est utilisé à grande échelle sur Internet. C'est un protocole de niveau 4 (transport) qui assure un transfert bidirectionnel de données, de façon fiable et sans erreur, avec contrôle et retransmission des données effectués aux extrémités de la liaison.

Il est utilisé par les services réseaux qui nécessitent de transférer des données de façon fiable et sans erreur. Les services ou protocoles l'utilisant peuvent aller de la simple liaison type terminal (*telnet, rlogin*) interactive et à bas débit, aux transferts type FTP (*File Transfert Protocol*) de plusieurs Méga-Octets non-interactive, en passant par les requêtes HTTP (*HyperText Transfert*

Protocol), le mail (SMTP : *Simple Mail Transport Protocol*), ou la visio-conférence. La notion de port permettant le multiplexage des données entre deux extrémités est la même que dans le protocole UDP.

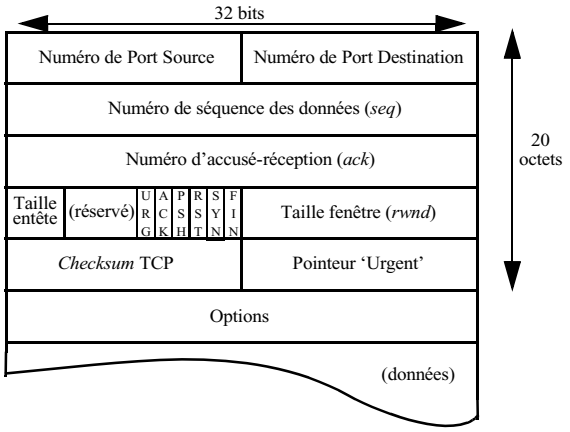
La principale caractéristique de TCP est qu'il est un protocole dit en mode connecté. Cela signifie qu'avant tout échange de données, une connexion entre les deux extrémités de la liaison doit être établie. Une fois réalisée, cette connexion, qualifiée de *virtuelle*, demeure existante jusqu'à terminaison explicite et peut-être considérée comme un tube particulier offrant une communication sûre reliant les ports respectifs des deux extrémités. Cela s'oppose aux protocoles de transport sans connexion comme UDP.

I.3.2.2 Une transmission sans perte et sans erreur

Plusieurs mécanismes permettent aux couches supérieures de ne pas se préoccuper du contrôle et de la validité des données reçues par la couche de niveau transport TCP. Les mécanismes peuvent se résumer ainsi:

- Les tailles des paquets assemblés par la couche TCP et transmis à la couche IP ne correspondent pas forcément aux tailles des paquets reçus des couches supérieures (utilisateur, application), et ceci à des fins d'optimisation de la bande passante. Un paquet transmis par la couche TCP à la couche IP est appelé un segment de donnée,
- A l'émission d'un segment, un temporisateur est armé. A l'expiration de celui-ci, et si le correspondant n'a pas renvoyé d'accusé-réception pour ce segment, il est considéré comme étant perdu, et est retransmis,
- Quand la couche TCP reçoit une donnée du réseau, un accusé-réception est envoyé après un délai optionnel. Ce délai autorise la prise en compte de nouvelles données à envoyer, afin d'optimiser la bande passante. Il ne doit pas excéder 500ms, et est habituellement de 200ms,
- Chaque segment est accompagné de son *checksum*. Un segment arrivant avec un *checksum* invalide est détruit et ignoré. Le temporisateur de l'émetteur détectera alors la perte. Il est admis que moins de 1% des erreurs de transmissions sont dues à des erreur de *checksum* [STE94],
- Le protocole IP ne garantie pas que l'ordre de réception des paquets à la destination est le même que l'ordre d'émission à la source. Les segments TCP peuvent donc arriver à la destination dans le désordre, et dans ce cas, la couche TCP se charge de les remettre dans l'ordre afin que l'application reçoive correctement les données,
- Les segments reçus en double sont éliminés,
- Chaque extrémité de la connexion TCP a un tampon de réception de taille limitée. TCP assure qu'une extrémité n'enverra pas plus de données que ce que son correspondant ne peut recevoir.

Figure 3: **Structure d'un segment TCP**



I.3.2.3 Fonctionnement

Structure d'une entête TCP

Un segment TCP est encapsulé dans un paquet IP à la source, et n'est décapsulé puis analysé que lors de l'arrivée du paquet IP dans le noeud de destination : le contrôle se fait de bout-en-bout. Une fois le paquet reçu, la couche IP extrait le segment TCP (figure 3) et le transfère à la couche transport (TCP) où l'entête est analysée. Elle contient notamment les champs:

- Ports source et destination :
  Ils jouent le même rôle que dans l'entête UDP : multiplexage des connexions entre les deux extrémités,
- Numéros de séquence et d'acquittement :
  Le numéro de séquence correspond à l'adresse du premier octet transporté dans le segment (modulo 2^{32}) vers la destination, tandis que le champ accusé-réception correspond à l'adresse du prochain octet attendu (modulo 2^{32} également).
  Le champ accusé-réception n'est valide que si l'indicateur ACK (*Acknowledge*: accusé-réception) est activé. Il contient dans le cas contraire une valeur indéterminée.
  Un segment TCP peut combiner des données et un accusé (optimisation de la bande passante), contenir des données sans accusé, ou contenir un accusé sans donnée utile (lorsqu'un accusé doit impérativement être envoyé à la source alors qu'aucune donnée n'est dans le tampon d'envoi),
- Indicateurs :
  Nous précisons ici la liste des fonctions des autres indicateurs de l'entête, mais nous ne détaillerons pas leur fonctionnement.
  L'indicateur PSH a une signification variable selon les implémentations. Il permet en général à l'émetteur de notifier au récepteur de vider ses tampons de réception en passant les données reçues aux couches supérieures.
  L'indicateur URG, associé au pointeur 'Urgent', définit une zone de données spéciales indépendantes des données du tampon, dans la zone de données du segment TCP.
  Les indicateurs SYN (*synchronization*), FIN (*finalize*), et RST (*reset*) servent respectivement lors de l'établissement, la terminaison, et la réinitialisation de la connexion.
- Le champ Option :
  Ce champ est optionnel, et sa taille maximum est de 40 octets.

Transfert de données

Il existe plusieurs types de transfert de données. Par exemple du point de vue de la couche applicative, ils peuvent être interactifs ou non. Une connexion de type terminal (*telnet*) est interactive, tandis qu'une connexion de type transfert de données brutes (*ftp*) ne l'est pas. Des études ont montré qu'environ 10% du volume en octets, correspondant à la moitié du trafic en terme de segments TCP est du trafic interactif [STE94].

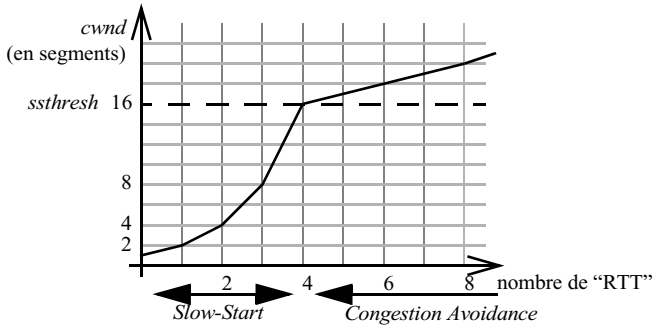
Le protocole TCP est conçu pour utiliser au mieux la bande passante pour les différents types de trafics pouvant exister, grâce à des mécanismes adaptatifs de débit.

Nous détaillerons la version *New-Reno* de TCP, qui est la plus répandue dans le réseau Internet [RFC2582].

Les deux modes de fonctionnement de TCP

Une transmission de données fonctionne en deux temps. Elle commence toujours en mode *Slow-Start* (démarrage lent, mais augmentation rapide du débit) qui correspond à la découverte de la qualité de la liaison entre les deux extrémités. L'autre mode, appelé *Congestion-Avoidance* (protection contre la congestion) est moins agressif, il permet d'utiliser la bande passante acquise en mode *Slow-Start*, tout en essayant de l'augmenter par incréments plus faibles (figure 4).

Figure 4: Slow-Start* et *Congestion-Avoidance



Une différence notable entre les deux modes réside dans la manière de modifier la taille de la fenêtre de congestion (appelée *cwnd* pour *Congestion WiNDow*) qui représente le débit instantané d'envoi par temps d'aller-retour (RTT - *Round Trip Time* : temps d'aller-retour).

L'algorithme TCP-NewReno est détaillé au chapitre IV.

I.4 Le routage dans les domaines IP

Nous avons vu au I.2.1 que dans un routeur, une table de routage IP contient la liste des adresses connues localement afin de leur associer une interface de sortie pour orienter les paquets vers leurs destinations. Bien que l'adressage IP soit de niveau mondial, un seul routeur ne peut pas connaître toutes les adresses de destination dans sa table de routage. Une gestion hiérarchisée de l'adressage et du routage est donc nécessaire.

Le réseau internet est subdivisé en domaines. Chaque domaine peut lui-même être encore subdivisé en plusieurs sous-domaines (I.2.3: Format de l'adresse IP). Toutes les interfaces d'un

même domaine ont en général la même adresse réseau.

Les protocoles de routage dans un domaine IP sont appelés IGP (*Interior Gateway Protocol :* Protocole de routage intérieur). Entre les domaines IP, on parle de EGP (*Exterior Gateway Protocol* : Protocole de routage extérieur).

I.4.1 Le routage IP: routage de proche en proche

La table de routage dans chaque routeur peut être positionnée par un opérateur, ou par un algorithme de routage. Si, une fois positionnée, le routage ne change pas, on le qualifie de routage statique, et au contraire, de routage dynamique si un algorithme est capable de le modifier en fonction des changements d'état du réseau.

Au minimum, chaque routeur doit avoir une vision partielle du réseau, limitée aux groupes d'adresses qui sont accessibles pour chacun de ses liens, permettant de transporter un paquet de proche en proche (*next-hop*). Une connaissance centralisée de l'ensemble des routeurs empruntés de la source jusqu'à la destination n'est pas nécessaire : Lorsqu'un paquet de destination inconnue (donc non locale) est reçu, le paquet est envoyé vers le routeur de frontière par défaut.

I.4.2 Les deux familles de protocoles IGP et EGP

Les IGP découvrent d'eux même la topologie du réseau pour trouver les chemins les mieux adaptés. Ils sont basés sur les algorithmes de type *distance-vector* qui n'ont qu'une vision partielle du réseau, ou sur les algorithmes de type *link-state* où chaque routeur construit une vision du domaine complet qu'il gère.

Les EGP travaillent principalement sur les routeurs de bordure des réseaux. Ils ne s'échangent pas les adresses de sous-réseaux.

Le tableau 1 présente ces deux types de protocoles.

Tableau 1: **Protocoles de routage dans Internet**

Type de protocole		Internet	ISO
Intra-domaine (*IGP*)	Entre terminaux	ICMP router discovery ICMP redirect *Distance-Vector :* RIP, RIP II	ES-IS
	Entre routeurs	*Distance-Vector :* RIP, RIP II, IGRP, EIGRP *Link-State :* OSPF	*Link-State :* IS-IS
Inter-domaines (*EGP*)		*Distance-Vector :* BGP	IDRP

I.4.3 Les « Autonomous System »

Dans le réseau Internet, les routeurs sont organisés de manière hiérarchique. Les domaines dans

l'Internet sont une collection de systèmes autonomes (AS : *Autonomous System*). Un système autonome est un réseau ou un groupe de réseaux placé sous une autorité administrative de routage unique.

Les AS utilisent à l'intérieur de leur domaine des protocoles de routage de type IGP tels que OSPF ou IS-IS (tableau 1). OSPF a été créé par l'IETF (*Internet Engineering Task Force*) et IS-IS par l'ISO (*International Organization for Standardization*). Ces deux protocoles calculent les routes par des algorithmes de type *Link-State* basés sur l'algorithme de plus court chemin de Dijkstra (I.4.4.2).

Les routeurs agissant dans un AS sont appelés *Interior Gateway Router*, et utilisent des protocoles de routage appelés *Interior Gateway Protocols* (IGP). D'autres routeurs permettent à différents AS de communiquer entre eux. Ils utilisent des protocoles de routage appelés *Exterior Gateway Protocols* (EGP).

I.4.4 Les protocoles de routage de type IGP

Un protocole de routage est un agent capable de modifier les tables de routage dans chaque routeur afin de rendre possible le transport des paquets de bout en bout, mais aussi de le faire le plus efficacement possible. Les principaux objectifs à atteindre par un protocole de routage sont:

- *Optimisation* (sélectionner le meilleur chemin selon les critères choisis),
- *Robustesse et souplesse* (pour faire face à tout imprévu),
- *Convergence rapide* (pour rapidement faire face aux boucles et aux pannes réseau),
- *Simplicité* (pour ne pas surcharger le réseau de données de contrôle).

En échangeant régulièrement des informations, les routeurs construisent une image de la topologie du réseau, qui n'est pas nécessairement connue initialement. Cela permet de créer les tables de routage.

Il existe deux grands types de protocoles de routage dynamique.

I.4.4.1 Distance-Vector

Le fonctionnement général d'un algorithme de routage de type *distance-vector* (vecteur de distance) est le suivant : Chaque routeur i associe un coût à chacun de ses liens vers un autre routeur j dans une table $d_i(j)$. Ce coût peut représenter n'importe quel type de mesure (une constante, un prix, un délai, ...). Chaque routeur i échange d_i avec ses voisins, et peut ainsi construire une nouvelle table $D_i(i,k)$ représentant le coût minimum pour atteindre un routeur k non directement connecté à i. Cette valeur est mise à jour à chaque réception de la table $D_j(j,k)$, pour toutes les valeurs de k connues par le routeur voisin.

$$D_i(i,k) = \min_{j \text{ connecté à } i} (d_i(j) + D_j(j,k)) \qquad (1)$$

Le choix du lien à emprunter pour aller de i à k est alors le lien (i,j) qui a permis la dernière mise à jour de la valeur actuelle de $D_i(i,k)$.

Ces échanges de tables entre les routeurs se font tant que l'algorithme n'a pas convergé. Une fois la convergence atteinte, les échanges continuent afin de pouvoir détecter les pannes et reconverger vers une nouvelle table de routage.

Cet algorithme est l'algorithme de Bellman-Ford distribué.

Exemples

- RIP [RFC1058], RIPv2 [RFC1388] (*Routing Information Protocol*) : la métrique utilisée est 1 par lien, 15 au maximum, 16 indique l'infini (ce qui limite le diamètre d'un domaine routé par RIP),
- IGRP, EIGRP ((*Extended) Interior Gateway Routing Protocol*, [CIS99]) : C'est une évolution de RIP par le constructeur CISCO, concernant principalement la métrique. Elle permet de tenir compte, en plus du nombre de routeurs intermédiaires, de la vitesse, du délai de transmission, de la charge et de la qualité des liens. La limitation à 16 est levée. Le partage de charge sur plusieurs routes est possible.

Avantages et Inconvénients

Le problème des algorithmes de routage de type *distance-vector* est la lenteur de leur convergence et l'instabilité (boucles de routage) de l'algorithme avant sa convergence. RIP résout cela en limitant les valeurs possibles de la métrique entre 1 et 15, 16 représentant une distance infinie, mais cela limite la taille du domaine. (E)IGRP lève cette limitation en introduisant d'autres techniques (*Split Horizon, Poison-Reverse Updates* [CIS99]).

I.4.4.2 Link State : OSPF

OSPF [RFC1247] (*Open Shortest Path First* : Le plus court chemin d'abord) est un protocole de routage dynamique et hiérarchique basé sur l'adressage IP. C'est un protocole de routage adapté aux grands réseaux à commutation de paquets (grand nombre de routeurs, redondance sur les chemins) dans lequel chaque routeur a une vision globale du réseau.
OSPF est un protocole de type Link-State (état de lien) : les données propagées par un routeur sont uniquement les informations sur ses liens avec ses voisins immédiats du même niveau hiérarchique.

Résumé de l'algorithme du *Link-State*

Lors de la phase d'initialisation de l'algorithme, chaque routeur découvre et identifie ses voisins immédiats (les routeurs voisins connectés à ses liens) grâce au protocole *Hello*. Des informations, appelées *link-states* sont alors construites. Elles contiennent des informations sur tous les liens du routeur, et notamment leur métrique. Les LSA (*Link-State Advertisement*: publication d'état des liens) regroupent ces informations pour les diffuser à grande échelle sur l'ensemble du domaine OSPF, de sorte que chaque routeur ait une vision (décalée dans le temps) de la topologie du domaine et de sa métrique. Chacun d'eux peut alors appliquer l'algorithme de Dijkstra du plus court chemin entre lui-même et l'ensemble des autres routeurs, pour construire sa table de routage IP.

Principes de base d'OSPF

La diffusion des *link-states* de chaque routeur vers tous les autres routeurs peut engendrer un trafic de fond non négligeable sur le réseau. Un routeur appelé DR est alors désigné (*Designated Router*) pour centraliser puis redistribuer l'ensemble des *link-states* (LSA) sur tous les routeurs.
Pour diminuer encore le trafic de contrôle, le domaine OSPF est scindé en zones (*OSPF areas*), et chaque zone fonctionne de manière indépendante suivant l'algorithme décrit précédemment. Une zone particulière, appelée zone *backbone* (*backbone area* : zone fédératrice) joue le rôle d'interconnexion entre toutes les zones.

La métrique propagée dans les LSA est un coût sans dimension. Pour chaque lien, elle peut représenter son état (l'abscence de LSA indique une panne), la capacité, la capacité non-utilisée, le délai ou la charge, ou tout autre coût. Par exemple dans les routeurs CISCO la métrique par défaut est donnée par 10^8/débit(bits/s) ("plus le débit est grand, plus le saut est court"). Cette métrique peut également être une constante imposée par l'opérateur, arbitraire ou résultante d'un processus d'optimisation externe.

OSPF est également capable de différencier les classes de service, permettant ainsi un routage différent selon le type de trafic dans le réseau. Les tables de routages pouvant être différentes selon les classes de services, une métrique différente peut être utilisée pour chacune d'elle (I.6.2.4). Le partage de charge (*Load Balancing*) entre routes de longueur égales peut également être employé.

Mise en oeuvre

• Initialisation

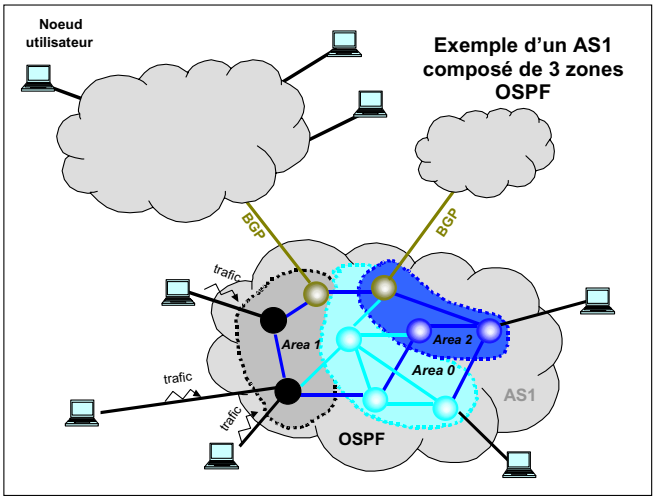
- Le routage initial pour permettre la diffusion des LSA et des informations de contrôle est positionné par l'opérateur. Les zones constituent un sous domaine avec une adresse de sous-réseau de l'AS,
- La diffusion des LSA se fait par un protocole *multicast* (multi-diffusion) afin de réduire le trafic de contrôle,
- Si un DR (routeur désigné) doit exister dans le réseau, l'élection se fait de proche en proche : chaque routeur se proclame routeur désigné, et au fur et à mesure de la réception des informations topologiques, celui-ci peut perdre cette élection si un routeur plus prioritaire s'est auto-proclamé.

• Pannes

L'élection d'un b-DR (*backup-DR*: DR de secours) est également organisée. le b-DR reste passif, mais est capable d'assurer les fonctions du DR en cas de panne de celui-ci.
La détection d'une panne d'un routeur de la zone se fait par les routeur voisins grâce aux paquets du protocole *Hello*. En effet, ces paquets sont régulièrement échangés pour rendre compte du bon fonctionnement des liens. Des LSA informant la suppression d'un lien sont alors propagés sur l'ensemble de la zone de sorte que les tables de routage soient recalculées.

• ***Interconnexion de zones et d'AS***

Figure 5: **Exemple d'interconnexion de zones et d'AS**



Au moins un routeur de chaque zone a une interface connectée à la zone *backbone* (zone *0*). Ces routeurs sont appelés ABR (*Area Border Router*: Routeur de bordure de zone). Ils possèdent la table de routage de leur zone et celle de la zone 0. Un routeur voulant s'adresser à un routeur extérieur à sa zone s'adresse à son ABR.
De même, nous avons les ASBR (*AS Border Router*: Routeur de bordure d'AS), qui ont la même fonction que l'ABR, mais pour l'ensemble de l'AS. Ces routeurs possèdent un algorithme de routage de type BGP.
Certaines connexions (extérieur - AS, zone *z* - zone *0*) ne peuvent se faire topologiquement qu'avec un routeur d'une zone non *backbone*. Dans ce cas une liaison virtuelle est établie entre ce routeur et un ASBR, ou ABR respectivement, dans la zone *0*.

• ***Métriques de type 1 et 2***

Les métriques dont nous avons parlé jusqu'ici sont les métriques de type 1, prioritaires par rapport aux métriques de type 2, qui sont essentiellement utilisées pour les données sortantes de l'AS. Les métriques de type 2 sont par nature plus grandes que le coût de n'importe quel chemin à l'intérieur de l'AS (diamètre de l'AS en terme de métrique de type 1).

• ***Routage OSPF basé sur le champ TOS (type de service)***

OSPF peut calculer une route différente pour des trafics IP appartenant à différents types de service. Cela implique qu'il peut y avoir autant d'entrées de table de routage que de champ TOS (*Type Of Service* : type de service) d'IP pour n'importe quelle destination. Cette fonctionnalité de routage interne au domaine OSPF est aussi étendue aux routeurs de frontière (I.6.2.4).

I.4.5 Le routage entre les domaines IP: BGP

Le protocole BGP (*Border Gateway Protocol*) est un protocole de routage entre systèmes autonomes. Il est utilisé pour échanger les informations de routage entre AS dans le réseau Internet. Ces informations de routage se présentent sous la forme de la suite des numéros d'AS à traverser pour atteindre la destination.
Quand BGP est utilisé entre AS, le protocole est connu sous le nom de e-BGP (*exterior BGP*). Si BGP est utilisé à l'intérieur d'un AS pour échanger des routes entre les routeurs de bordure d'un même AS afin de le traverser, alors le protocole est connu sous le nom de i-BGP (*interior BGP*).

BGP est un protocole très robuste et très scalable : les tables de routage BGP du réseau Internet comprennent plus de 90000 routes. Si BGP a atteint ce niveau de scalabilité, c'est essentiellement parce qu'il associe aux routes des attributs permettant de définir des politiques de routage entre AS.

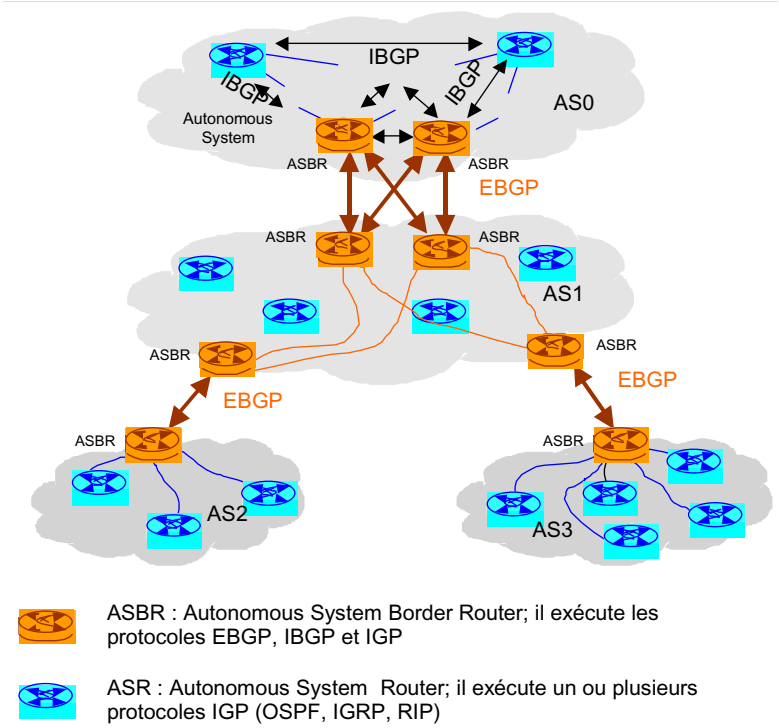


Figure 6: **.Exemple d'interconnexion d'AS par BGP**

Les routeurs de frontière communiquant par i-BGP doivent être complètement maillés (mais pas forcément directement connectés) afin de permettre l'échange des informations. Les évolutions de BGP utilisent cependant des techniques permettant de réduire le nombre de messages à échanger afin de réduire le trafic de contrôle dans les grands AS.

Les voisins BGP échangent initialement toutes leurs informations de routage par une connexion TCP. Les routeurs BGP n'envoient ensuite à leur voisins que des mises à jour des tables de routage lorsqu'un changement de route est détecté ou qu'une nouvelle route est apprise. Ainsi, les routeurs BGP n'envoient pas de mises à jour périodiques mais ne diffusent que le chemin «optimal» vers un réseau destination.

BGP peut servir aussi à mettre en commun les tables de routages des différents algorithmes IGP pouvant exister dans différents sous-domaines d'un même AS, par exemple pour pouvoir router des données d'un sous-domaine à l'autre, sans sortir de l'AS.

I.5 Réseaux IP Multiservices et la gestion de QoS

I.5.1 Introduction

L'Internet de demain devra permettre le déploiement d'applications multimedia ayant des exigences spécifiques en terme de QoS (*Quality of Service* : Qualité de service). Certains services comme les services vocaux auront besoin d'un faible délai point à point et d'une faible gigue. D'autres, comme les trafics de données, nécessiteront des faibles taux de pertes ou d'erreurs, sans retransmission, avec éventuellement une certaine garantie de bande passante pour les trafics critiques comme les données transactionnelles.

Pour pouvoir garantir la QoS des flux transportés, il va donc falloir utiliser des mécanismes permettant de traiter de manière différenciée les différentes catégories de trafic dans les organes du réseau ainsi que des protocoles de signalisation de la QoS pour pouvoir allouer des ressources en fonction des besoins des applications.

On peut distinguer deux grandes problématiques pour la gestion de la QoS dans un réseau IP :

- La gestion des phénomènes de congestion :
  Il s'agit un point fondamental pour garantir la QoS des flux. En effet, les mécanismes de gestion des QoS n'auront un impact effectif que lorsque le réseau sera chargé. Il existe deux grandes approches pour la gestion des congestions : les méthodes réactives et les méthodes préventives.
  La philosophie du contrôle réactif est d'accepter un maximum de connexions. Lors de la congestion d'un équipement réseau (mesurée en terme de pertes, de délais et de remplissage des tampons), les sources réduisent leurs débits. Ainsi, dans le modèle *Best-Effort* (modèle « au mieux ») du réseau Internet actuel, la gestion du phénomène de congestion est faite de manière réactive par le mécanisme de fenêtre glissante (adaptation du débit) du protocole TCP. Toutefois, ce contrôle n'est pas adapté pour pouvoir offrir des garanties de QoS par exemple à des flux audio et vidéo temps réel : ils ne peuvent pas adapter leur débit.
  A l'inverse, le contrôle préventif consiste à prendre des mesures « à priori », afin de minimiser l'apparition du phénomène de congestion et/ou pour qu'il n'affecte pas les services garantis. Ainsi, les techniques de contrôle d'admission ou de mise en forme des trafics (*Policing*, *Shaping*) permettent de réduire la fréquence des congestions tandis que les mécanismes de gestion de tampon (RED, WRED, etc.) permettent de respecter les QoS des services les plus prioritaires en cas de congestion.
- L'ordonnancement (*Scheduling*) des paquets est aussi un mécanisme fondamental pour garantir la QoS des flux transportés. Ceci est évident si on considère des flux hétérogènes : les rafales de certaines connexions peuvent perturber le trafic temps réel même s'il n'y a pas congestion. Ainsi, bien que la mise en oeuvre d'ordonnancements autres que FIFO (*First in, First out* : Premier Entré, Premier Sorti ou PAPS) soit difficile sur des routeurs à très haut débit, les équipementiers réseau fournissent de plus en plus des mécanismes sophistiqués réalisant un ordonnancement prenant en compte les classes de trafic (WRR, WFQ, etc).

Ces différents mécanismes de contrôle de congestion et d'ordonnancement des paquets sont présents dans toutes les architectures développées pour le contrôle de la QoS dans les réseaux IP. Historiquement, la première architecture qui a été proposée associe, comme ATM, une QoS à chaque flux que le réseau transporte. Il s'agit du modèle IntServ (*Integrated Services* : Service intégré). Pour des raisons de d'échelle principalement, la communauté Internet a également proposé un modèle appelé DiffServ (*Differentiated Services :* Services différenciés) dans lequel la QoS est associée à une agrégation de flots. Ce regroupement est appelé « classe de service ».
Ces deux modèles de réseaux IP-Multiservices peuvent être utilisés avec le protocole de commutation MPLS (*Multi Protocol Label Switching*) qui permet un traitement très rapide des paquets sur les routeurs du coeur du réseau en remplaçant la fonction de routage IP par une fonction de commutation (*switching*), beaucoup plus adaptée au débit des réseaux de transport actuels. Le protocole MPLS encapsule ainsi le protocole IP.

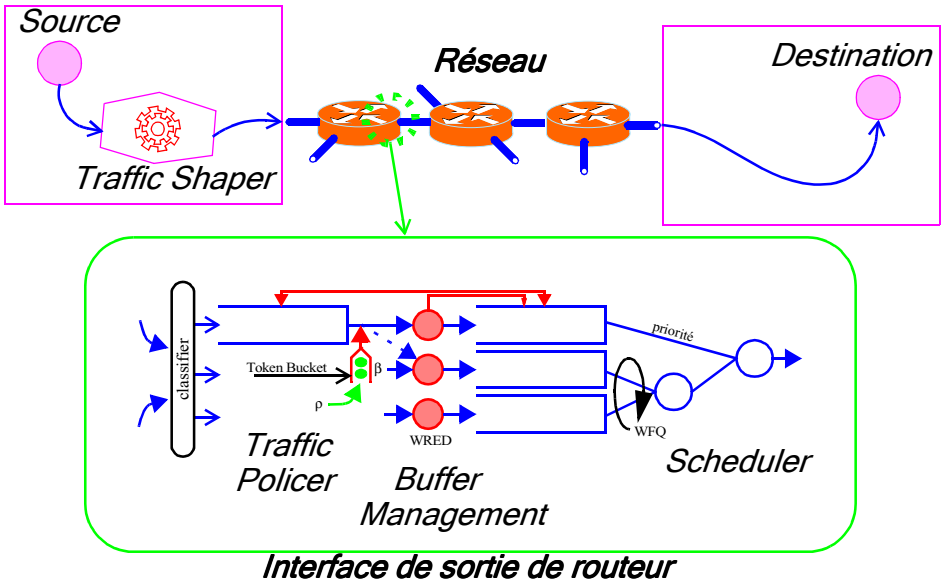
Dans ce paragraphe, nous présentons les principaux mécanismes permettant la gestion de la qualité de service dans les réseaux IP. Puis nous décrivons les deux architectures proposées, IntServ et DiffServ. Enfin, nous présentons rapidement le protocole MPLS qui combine les avantages du monde IP avec ceux du monde ATM.

I.5.2 Les Principaux Mécanismes de Gestion de la QoS

La gestion de la QoS dans les réseaux IP suppose un ensemble de traitements différenciés, réalisés soit au niveau de la source de trafic (terminal émetteur) soit dans les organes du réseau. Les

principaux traitements réalisés sont illustrés ci-dessous (figure 7).

Figure 7: **Architecture générale d'un réseau à QoS.**



On peut distinguer les mécanismes suivants :

- Le *traffic shaping* (mise en forme du trafic) consiste principalement à introduire du délai entre les émissions de paquets de manière à éviter ou limiter les rafales de paquets. Le *shaper* fournit au réseau une description simple du trafic,
- Le *traffic policing* (contrôle du trafic) est une opération qui consiste à vérifier que le trafic émis est bien conforme à la description qui en a été faite par la source,
- Le mécanisme de *buffer management* (gestion de tampon) consiste à éliminer des paquets en cas de congestion du tampon d'une interface de sortie de manière sélective, en fonction des paramètres de QoS associés aux flux,
- Le *traffic scheduling* (ordonnancement du trafic) consiste à ordonnancer la transmission des paquets présents dans les tampons de l'interface de sortie en fonction des paramètres de QoS associés aux flux.

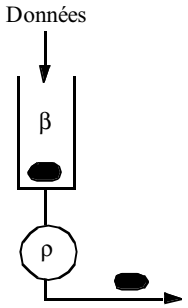
Dans la suite, nous décrivons ces différents mécanismes.

I.5.2.1 Traffic Shaping (mise en forme du trafic)

La mise en forme du trafic a trois objectifs principaux. Tout d'abord, il s'agit d'un moyen de fournir au réseau une description du trafic généré par les flots. Ensuite, étant donné cette description, le réseau peut éventuellement déterminer si ce flot peut être autorisé à émettre (contrôle d'admission) et comment gérer son trafic. Enfin, grâce à cette information, le réseau peut périodiquement contrôler le trafic par le mécanisme de *policing* (I.5.2.2) pour vérifier que le flot se comporte comme prévu.

Leaky Bucket

Figure 8: **Leaky Bucket**



Le mécanisme de *traffic shaping* le plus simple, le *leaky bucket,* (seau percé) consiste à permettre le passage de quantités régulières de données à intervalles réguliers. Il permet ainsi de réguler des rafales de trafic en un flot régulier de paquets.

Conceptuellement, le *leaky bucket* fonctionne comme indiqué sur la figure 8. Chaque flot a son propre leaky bucket. Lors de l'émission de données, l'hôte émetteur place les paquets dans un tampon et ces paquets sont émis sur le réseau au débit ρ. La taille du tampon β limite le nombre de paquets en attente d'émission. Si un flot accumule plus de données que le tampon ne peut en contenir, l'excès est éliminé.

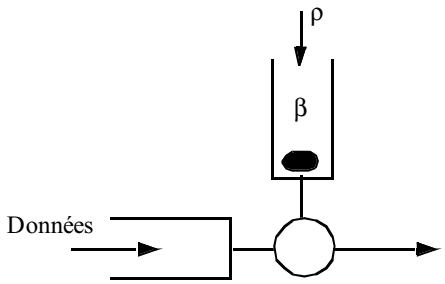
L'effet principal du *leaky bucket* est de transformer des rafales de trafic en un flot régulier de paquets et émis toutes les 1/ρ unités de temps. Ainsi le réseau sait que le flot n'émettra pas de paquets plus rapidement qu'à ce débit. La taille du tampon β permet de limiter le délai avant émission d'un paquet ainsi que la taille maximale des rafales que la source tente d'émettre.

L'intérêt principal du *leaky bucket* est qu'il est très simple à implémenter et qu'il fournit une description simple (le paramètre ρ) du trafic injecté dans le réseau par la source, facilitant ainsi l'opération de contrôle (*policing*) du trafic dans le réseau. La principale limitation de ce mécanisme est liée au faible nombre de comportements des sources de trafic qu'il peut décrire. Le *leaky bucket* entraîne un gaspillage de ressources pour des trafics à débit fortement variable (obligation d'allouer des ressources associées au tampon pour le débit crête).

Token Bucket

Le *token bucket* (réservoir à jetons) est une extension du *leaky bucket* dans laquelle les données sont placées dans un tampon et émises à un débit régulé par un réservoir *(bucket)* dans lequel sont placés des jetons (*tokens*) à intervalles réguliers, chaque jeton correspondant à une autorisation d'envoi.

Figure 9: **Token Bucket**



Dans un *token bucket*, le débit ρ est le débit auquel les jetons sont placés dans le réservoir qui a une capacité β. Si le bucket est plein (β), les jetons arrivant sont éliminés. Les paquets à émettre sont placés dans le tampon sur la gauche de la figure 9. Pour transmettre un paquet, le régulateur doit enlever du réservoir un jeton.

Ainsi, alors qu'un simple *leaky bucket* transforme les trafics avec rafales en flots réguliers, le *token bucket* permet des rafales de trafic, tout en les bornant. Plus précisément, le *token bucket* garantit que le flot n'émettra pas plus de l'équivalent en données de $(\beta + \tau\rho)$ jetons sur un intervalle de longueur τ, et que son débit moyen de transmission sur le long terme n'excédera pas ρ. Un autre avantage du *token bucket* est qu'il n'élimine pas directement les données mais les jetons, et laisse au réseau la responsabilité de gérer le tampon de transmission.

Le *token bucket* a deux inconvénients. Le premier est que la fonction de police du trafic dans le réseau devient un peu plus complexe qu'avec un simple *leaky bucket*. Le second est qu'un flot qui est

resté silencieux suffisamment longtemps va pouvoir disposer de β jetons d'émission et donc consommer beaucoup de bande passante d'un coup. Pour remédier à ce dernier problème, certains auteurs ont proposé de mettre en oeuvre un *token bucket* et de réguler le débit d'émission par un *leaky bucket* de débit C, avec C > ρ.

I.5.2.2 Traffic Policing

La qualité de service (QoS: *Quality of Service*) ne peut être garantie que si tous les flots injectent du trafic dans le réseau conformément à leurs spécifications. Le *traffic policing* sert à détecter "à la volée" les applications frauduleuses ou mal spécifiées. Sa mise en oeuvre dépend étroitement de la façon dont sont spécifiées les QoS des flux. Elle peut être simple, notamment pour des spécifications de type réservoir (*bucket*), ou plus problématique pour des spécifications de type paramètres statistiques.

Plusieurs politiques peuvent être mises en oeuvre, de la plus draconienne à la plus lâche. En cas de non respect de la spécification, les données en trop peuvent être soit supprimées, soit traitées sans garantie (*best effort*) et en cas de saturation du réseau, supprimées. Cette dernière politique permet d'optimiser les ressources du réseau.

On inclut généralement dans ce mécanisme la régulation des flux, et ceci parce qu'il s'agit souvent d'un même algorithme. La régulation de flux étant faite par un algorithme de type *token bucket*, il est alors aisé de contrôler les paquets supplémentaires: détection de non-conformité (débordement du tampon de transmission) et traitement adéquat (par exemple suppression).

I.5.2.3 Buffer Management

Le mécanisme de gestion de tampon (*buffer management*) consiste à éliminer des paquets en cas de congestion du tampon d'un port de sortie d'un routeur. L'élimination est réalisée de manière sélective, en fonction des critères de QoS des trafics. Ainsi, dans le cas de DiffServ (I.5.4), les algorithmes de gestion de tampon se basent sur le PHB (*Per Hop Behavior*) qui spécifie le comportement à adopter en fonction de la classe du trafic.

Plusieurs algorithmes de gestion de tampon ont été proposés pour les réseaux à QoS: WRED, RIO, FRED. Ils sont tous basés sur l'algorithme RED (*Random Early Detection/Discard:* Détection/ Suppression précoce aléatoire) qui a été conçu pour limiter le phénomène d'oscillation du débit des connexions TCP (voir IV.2.3). Pour éviter ce phénomène de congestion, RED supprime aléatoirement quelques paquets bien avant que n'apparaissent des problèmes de congestion. Ce type d'algorithme est recommandé dans les réseaux IP dans [RFC2309].

L'algorithme RED ne peut toutefois pas être directement utilisé dans des réseaux IP multiservices car son principe d'élimination aléatoire et aveugle de paquets n'est pas du tout adapté à des trafics de données de type temps-réel. Ce mécanisme est donc généralement activé sur les classes de trafic faisant intervenir le protocole TCP.

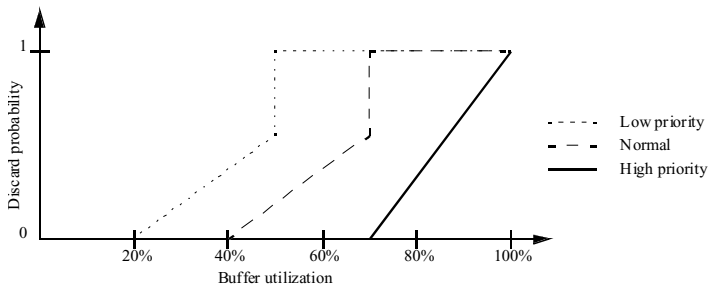
De cette manière, le trafic *Best Effort* peut continuer à être géré suivant l'algorithme RED, tandis que les autres trafics ne sont affectés par les pertes aléatoires plus ou moins sévères qu'en cas de congestion, suivant les poids affectés aux classes.

WRED est basé sur un estimateur de l'occupation de la file d'attente dont il régule la congestion. C'est un estimateur à oubli exponentiel. Soit *w* le poids (*weight*), *X(t)* le nombre de

paquet dans la file, et $\overline{X(t)}$ l'estimateur utilisé par l'algorithme WRED. À chaque arrivée d'un nouveau paquet, l'estimateur est mis à jour comme suit :

$$\overline{X(t_{i+1})} = \overline{X(t_i)} \cdot (1 - w) + X(t_i) \cdot w \qquad (2)$$

Figure 10: **Principe de WRED**



Le principe de WRED est illustré sur la figure 10. Les paquets de faible priorité commencent à être éliminés aléatoirement à partir de 20% d'utilisation du tampon et le sont systématiquement au dessus de 50%. A partir de 40%, le routeur commence à éliminer des paquets de priorité moyenne et les élimine systématiquement lorsque l'utilisation du tampon dépasse 70% de la capacité. Enfin, ce n'est que lorsque l'on dépasse les 70% d'utilisation de la capacité que des paquets de haute priorité commence à être éliminés. Plus précisément, les paramètres pour une file sont la capacité C_0 à partir de laquelle les paquets commencent à être supprimés, la probabilité maximale P_{max} de suppression lorsque l'estimateur du nombre de client $\overline{X}$ de la file atteint la capacité C_{max}. La probabilité de suppression est donnée par l'expression suivante:

$$P = 0 \qquad \text{si } \overline{X} < C_0$$

$$P = P_{max}\frac{\overline{X} - C_0}{C_{\max} - C_0} \qquad \text{si } C_0 < \overline{X} < C_{max}$$

$$P = 1 \qquad \text{si } \overline{X} > C_{max}$$

Cet algorithme simple n'est donné qu'à titre de clarification. En réalité, il est un peu plus complexe que cela [FLO93]. Les chiffres donnés ne sont qu'illustratifs. Diverses politiques peuvent être mises en place suivant les besoins des flux en terme de délai et de taux de perte.

I.5.2.4 Ordonnancement des trafics (*Traffic Scheduling*)

C'est l'un des composants les plus critiques dans l'architecture d'un réseau à QoS. En effet, même en l'absence de congestion, les paquets des différents flux subissent des délais d'attente plus ou moins importants. Ces délais d'attente dans les tampons sont néfastes pour les applications sensibles aux délais et surtout pour les applications sensibles à la gigue. L'idée du *traffic scheduling* est d'adapter la politique de transmission des paquets en attente dans un tampon, en fonction des besoins en QoS des flux. Le choix de la politique d'ordonnancement a un impact important non seulement sur le délai et la gigue mais aussi sur la taille des tampons. Suivant la politique, la garantie de QoS sera déterministe ou statistique.

Les principales qualités attendues d'un ordonnanceur sont une bonne isolation des flots (traitement séparé des flots, ou découplage des différents flots en classe de trafic), une implémentation aisée et scalable, un temps de traitement faible et la qualité des garanties de QoS apportées.
Chacun des algorithmes présentés ci-après peut être vu de deux manières différentes : La vision

IntServ, qui consiste à traiter chaque flux indépendamment, et la vision DiffServ, qui consiste à regrouper tous les flux par classe de service, et à traiter chaque classe de service comme s'il s'agissait d'un seul flux (plus de détails en I.5.3 et I.5.4). Les flux d'une même classe sont alors traités selon la discipline FIFO à l'intérieur de cette classe. Dans la suite, nous parlerons de sessions. Une session sera un flux dans la vision IntServ, ou une classe regroupant plusieurs flux dans la vision DiffServ.
Un problème se pose cependant dans la vision DiffServ : Si un flux est prépondérant par rapport aux autres au sein d'une même classe de service, alors aucun mécanisme DiffServ ne pourra empêcher la gêne occasionné aux autres flux de la même classe.

Principalement trois types d'ordonnancement ont été envisagés : l'ordonnancement à priorité fixe, l'ordonnancement basé sur le paradigme GPS (*Generalized Processor Sharing :* Temps partagé généralisé) et celui basé sur la notion de trame temporelle.

Ordonnanceurs à Priorité Fixe

Les ordonnanceurs de type priorité fixe transmettent les paquets dans l'ordre de leurs priorités. Un paquet ne peut être transmis s'il y a un paquet de priorité supérieure en attente. Ces ordonnanceurs garantissent les délais les plus faibles possibles pour les paquets de priorité haute (temps réel). Toutefois, un des inconvénients majeurs de ce type de politique est que les sessions de priorités inférieures ne sont absolument pas isolées (elles sont défavorisées). Cet algorithme est néanmoins utilisé car très simple à implémenter et donc rapide.

Ordonnanceurs basés sur le paradigme GPS

Les ordonnanceurs basés sur le paradigme GPS [TOU98] (*Virtual Clock*: horloge virtuelle, *Weighted Fair Queueing*: gestion pondérée et équitable de file d'attente) fournissent un partage différencié de la bande passante avec un minimum garanti à chaque session. Cette bande passante minimale est basée sur un poids associé à la session qui est fonction de ses paramètres de QoS. Une session peut disposer temporairement d'un débit supérieur quand les paquets d'une ou plusieurs autres sessions ne sont pas en attente de transmission. Dans ce cas, le débit en surplus est partagé équitablement entre les sessions présentes suivant leurs poids. Ces ordonnanceurs possèdent la notion de temps virtuel associé à une sortie. Ils divisent la bande passante de la sortie pour chaque session en émulant un multiplexage TDM (*Time-Division Multiplexer*). Leur complexité est souvent de l'ordre de *O(log(n))*, ce qui les rend peu scalables dans la vision IntServ. Ceci est cependant acceptable dans la vision DiffServ car *n* est réduit au nombre de classes de services.

Définition 1: Soit une collection de réels positifs $(\phi_i)_{i=1...n}$ associés aux sessions 1...*n*. Le service est conforme à la discipline de service GPS s'il est non-oisif et si, étant donné deux instants t_1 et t_2, les quantités de services reçues par deux sessions *i* et *j* actives dans l'intervalle $[t_1...t_2]$ sont telles que :

$$\frac{W_i[t_1 \dots t_2]}{\phi_i} = \frac{W_j[t_1 \dots t_2]}{\phi_j} \qquad (3)$$

Un serveur est non-oisif s'il n'est jamais inactif lorsque des paquets à servir sont présents.
Les réels (ϕ_i) sont les poids (*weights*) et $W_i[t_1...t_2]$ représentent la quantité de travail reçu (*Work*) entre les temps t_1 et t_2. Seul l'importance relative (4) de ces poids a de l'intérêt. Pratiquement, pour

assurer les débits relatifs entre sessions dans le cas où elles sont toutes actives, on peut donner à ϕ_i la valeur de la bande passante requise pour la session i, la somme de tous les $(\phi_i)_{i=1...n}$ étant égale à la bande passante totale.

$$\frac{W_i[t_1 \ldots t_2]}{W_j[t_1 \ldots t_2]} = \frac{\phi_i}{\phi_j} \qquad (4)$$

La quantité W_i/ϕ_i est la quantité normalisée de service, également appelée temps virtuel : le temps virtuel est le même pour chaque session.

Ordonnanceurs basés sur la notion de trame temporelle

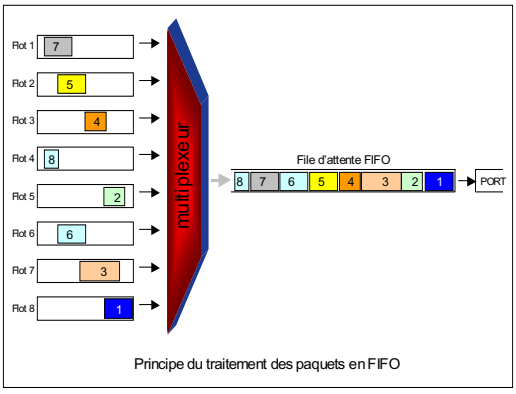
Les ordonnanceurs basés sur la notion de trame temporelle, comme l'algorithme WRR (*Weighted Round Robin*: Partage du service (*round robin*) Pondéré), divisent le temps en trames de tailles variables ou non. Ces trames sont allouées aux sessions à la façon *round robin*. En terme de garanties de délai et de gigue, ces ordonnanceurs sont moins performants que les ordonnanceurs basés sur le paradigme GPS. En revanche, ils sont plus simples à implémenter. Cela peut être un facteur décisif, si la vitesse des liens augmente plus vite que la puissance de calcul.

Nous allons illustrer les différents type d'ordonnanceurs utilisés dans les routeurs modernes.

I.5.2.5 Exemples d'ordonnanceurs

FIFO (First-In First-Out)

Figure 11: **Ordonnancement type FIFO (*First In First Out*)**

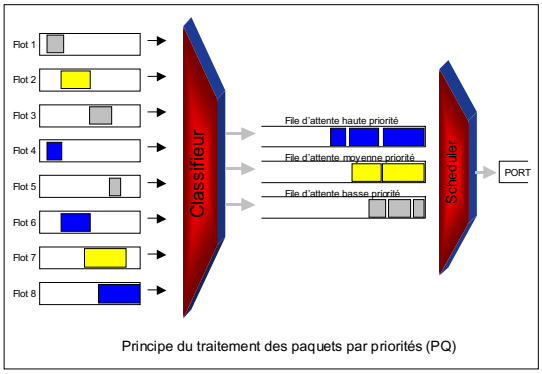


La discipline FIFO (PAPS) est la discipline de traitement la plus simple rencontrée (figure 11). Les paquets sont placés dans une file d'attente suivant leur ordre d'arrivée et sont ensuite envoyés sur le port de sortie, après traitement, suivant ce même ordre. Cette discipline est très simple à implémenter. Cependant lorsqu'une session présente une surcharge ponctuelle de trafic, toutes les autres sessions utilisant le même port sont bloquées conduisant ainsi à un ensemble de problèmes. Par exemple l'ensemble des connexions TCP utilisant ce port risque d'avoir le débit extrêmement réduit tant que la surcharge de trafic n'est pas passée.

PQ (Priority Queueing)

Le traitement par files à priorités permet de faire de la différentiation de service (figure 12). Les paquets qui entrent dans le système sont tout d'abord classifiés et envoyés sur la file d'attente correspondant à la session à laquelle ils appartiennent. Les paquets contenus sur la file de haute priorité sont servis (envoyés sur le port de sortie) en premier. Quand cette file est vide c'est la file de priorité moyenne qui est traitée. Les paquets de la file de basse priorité ne sont traités que lorsque tous les paquets des deux autres files ont été servis (et que les files sont vides). Dans chaque file, le traitement se fait suivant une discipline FIFO.

Figure 12: **Ordonnancement type PQ (*Priority Queueing*)**



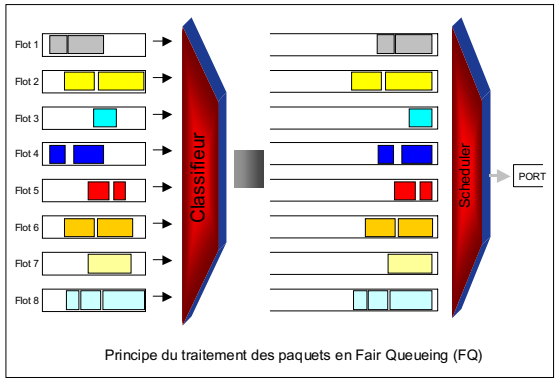
Le problème du traitement par priorité vient du fait que si certains trafics connaissent des pointes ou un débit supérieur à ce qui est prévu nominalement, alors les trafics affectés aux files moins prioritaires ont leur taux de service qui diminue rapidement. Pour éviter un tel problème certains constructeurs proposent des files prioritaires avec une bande passante maximum utilisable. Ainsi, en cas de surcharge de trafic de haute priorité, les autres files sont tout de même servies.

Un exemple immédiat de ce type d'ordonnancement est le traitement du trafic de contrôle du réseau. En cas de congestion ou de problème dans le réseau, il est primordial que les trafics de reconfiguration des protocoles puissent circuler entre les routeurs, quelque soit la charge de ces derniers.

FQ (Fair Queueing)

Cette discipline de traitement des paquets a été proposée par John Nagle en 1985 [RFC970]. C'est la première contribution aux disciplines de traitement qui permettent d'assurer que chaque session a droit à un accès équitable aux ressources de bande passante, évitant ainsi qu'une session avec une pointe de trafic ne consomme plus de ressource qu'on ne veut lui attribuer.
Chaque session est affectée à une file d'attente. La discipline de service est de type «*Round-Robin* paquet par paquet», c'est à dire un paquet de chaque file à tour de rôle. Les paquets dans chaque file sont servis suivant une discipline FIFO.

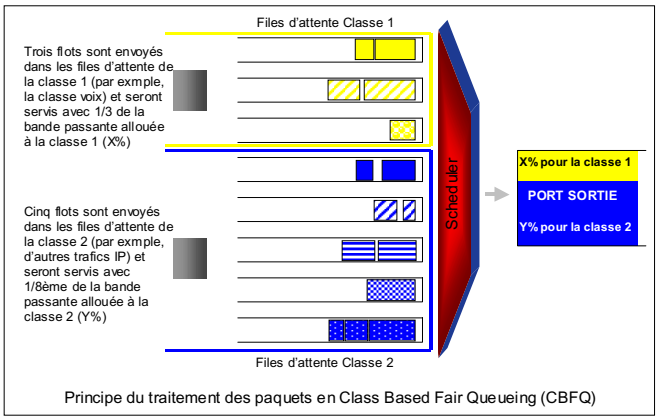
Figure 13: **Ordonnancement type FQ (*Fair Queueing*)**



L'inconvénient de FQ est tout d'abord de fournir un service égal sur toutes les files, et de servir des paquets indépendamment de leur taille. Les files actives (toujours avec des paquets en attente) ayant des paquets de longueur plus importante utiliseront donc plus de bande passante que les autres. En fait FQ peut être vu non pas comme un partage équitable de la bande passante affectée à chaque file mais comme un équilibrage du nombre de paquets traités.

La solution FQ dans la vision IntServ n'est pas beaucoup utilisée à cause du nombre de flux qui peut exister dans un cœur de réseau IP. Par contre, elle fournit une bonne manière d'isoler les flux utilisateurs entrant dans un réseau les uns par rapport aux autres en garantissant une certaine qualité à tous les flux quelles que soient les conditions de trafic.

***Figure 14:* Ordonnancement type CBFQ (*Class-Based Fair Queueing*)**

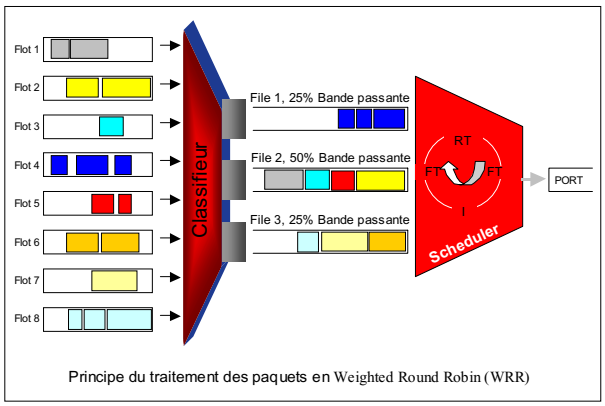


Avec la solution FQ-DiffServ (appelée aussi CBFQ : *Class-Based Fair Queueing*, figure 14), un pourcentage de la bande passante du port de sortie est affecté à chaque classe. Cette bande passante est garantie pour chaque classe, quelle que soit l'intensité des flux dans les autres classes.

FQ-DiffServ est plus facilement exploitable dans les logiciels d'exploitation de routeurs, du fait du nombre plus réduit de classes de services différentes existant dans les réseaux.

WRR (Weighted Round Robin)

***Figure 15:* Ordonnanceur type WRR (*Weighted Round Robin*)**



WRR s'appelle aussi CBQ (*Class Based Queueing*) ou CQ (*Custom Queueing*). Avec WRR, chaque file peut avoir un pourcentage de la bande passante du port de sortie. WRR corrige ainsi les défauts de PQ (plus de famine) et de FQ (affectation d'un poids à chaque file).

Dans WRR, les paquets sont tout d'abord classifiés suivant des classes de service (DiffServ) et affectés à une file d'attente spécialisée pour cette classe de service.

La pondération du principe *Round Robin* peut être réalisé de deux manières différentes :

- Chaque file est servie à tour de rôle. Plusieurs paquets d'une file peuvent être servis lorsque celle-ci est visitée (nombre de paquets au prorata du poids relatif de la file),
- Chaque file est visitée plusieurs fois pendant un tour *Round-Robin*, en servant un seul paquet à la fois. La fréquence de visite est calculée au prorata du poids relatif de la file.

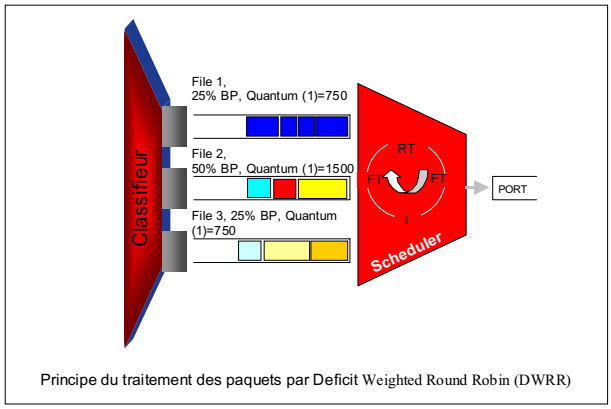
L'ordonnancement *Round-Robin* peut être facilement implémenté directement dans le matériel et ceci présente un avantage certain. Toutefois, le partage de bande passante effectué par *Round*

Robin, se fait sur le nombre de paquets servis. Cela implique que pour obtenir un partage vraiment équitable de la bande passante, toutes les files doivent être occupées et tous les paquets doivent avoir la même taille.

DWRR (Deficit Weighted Round Robin)

DWRR permet de corriger les défauts de WRR, en particulier pour obtenir un juste partage de bande passante.

Figure 16: **Ordonnanceur type DWRR (*Deficit Round Robin*)**



Principe du traitement des paquets par Deficit Weighted Round Robin (DWRR)

Dans DWRR, chaque file est configurée avec certains paramètres :

- Un poids (pourcentage de bande passante, comme dans WRR),
- Un quantum de quantité de service, proportionnel au poids de la file, et qui est exprimé en octets,
- Un compteur de déficit, qui indique le nombre d'octets maximum que peut transmettre une file chaque fois qu'elle est visitée.
  Par exemple, pendant le cycle *Round-Robin* un paquet ne peut pas être transmis parce qu'il est trop grand. La file ne le transmet pas et garde alors son crédit. Ce crédit (compteur de déficit) sera ajouté au crédit gagné pendant le cycle suivant (le quantum de temps de service de la file), ce qui permettra à terme de transmettre ce paquet.

Principe de l'algorithme :

L'ordonnanceur visite les files non vides. Le compteur de déficit est incrémenté de la valeur du quantum offert à la file.

- Si la taille du premier paquet en attente est plus grande que la valeur du compteur de déficit, l'ordonnanceur ne prend pas le paquet et passe à la file suivante non vide.
- Si la taille du paquet est plus petite que le compteur de déficit, le paquet est servi. et le compteur de déficit est décrémenté de la valeur de la taille du paquet servi. L'ordonnanceur continue sur la même file tant que la taille du paquet de tête est plus petite que la valeur du compteur. Il s'arrête et change de file dans le cas inverse. Quand la file est vide, le compteur de déficit est mis à zéro et l'ordonnanceur passe à une autre file.

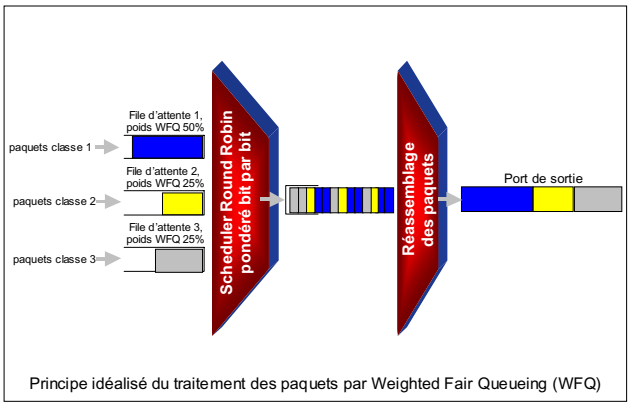
DWRR n'est pas aussi précis que d'autres algorithmes (GPS par exemple) mais il a la grande qualité de sa simplicité. Cependant pour des liens à très grande bande passante, la précision de l'ordonnancement n'est pas nécessairement aussi critique que pour des liens à faible bande passante. Etant donné que l'algorithme peut être implémenté directement dans le matériel (pas de logiciel), c'est donc une bonne solution pour les nœuds d'accès mais aussi pour les nœuds de cœur de réseau.

GPS (Generalized Processor Sharing) et approximations

L'ordonnancement de type GPS [TOU98] (Partage de processeur généralisé) donne un pourcentage de la bande passante à chaque session. Le partage de la bande passante ne s'effectue pas

en traitant les paquets, le partage (*sharing*) de type GPS se fait au niveau du bit de donnée, ce qui rend effectivement le partage beaucoup plus équitable.

***Figure 17:* Ordonnancement type GPS (*Generalized Processor Sharing*)**



Principe idéalisé du traitement des paquets par Weighted Fair Queueing (WFQ)

GPS est un mécanisme d'ordonnancement théorique et impossible à mettre en oeuvre pratiquement (les paquet ne sont jamais traités au niveau du bit comme en figure 17). Il existe plusieurs algorithmes discrets (niveau paquet) permettant d'approcher le paradigme GPS.

WFQ (V.5.2.1 page 143) (*Weighted Fair Queueing* : mise en attente équitable pondérée, figure 18) est l'un de ces algorithmes. Il est également appelé PGPS (*Packetized-GPS* : GPS paquétisé). Ce n'est pas le meilleur algorithme du point de vue performances [TOU98], mais il est souvent utilisé dans les routeurs.

Dans WFQ, c'est donc bien la somme des longueurs des paquets qui va être comptée comme le service offert et non le nombre de paquets. Il en résulte que c'est bien la bande passante qui est partagée (et pondérée) entre les sessions.

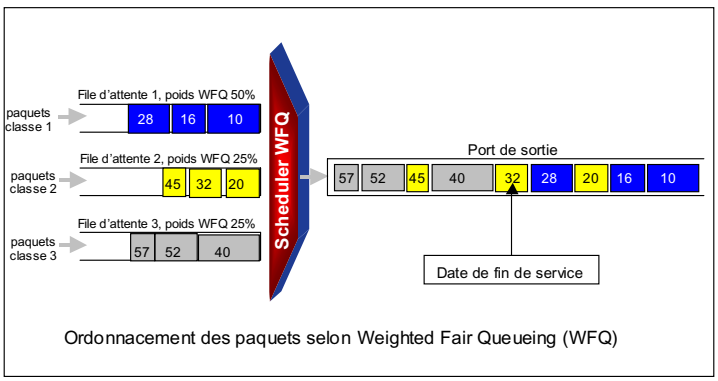
Pour donner une idée générale, WFQ tend, en moyenne, à faire ce qu'aurait fait un algorithme de type *Round Robin* pondéré si les paquets avaient tous la même taille. Le principe de WFQ est de prédire les dates virtuelles de fin de service des paquets, et de les classer par ordre de traitement, afin de les servir dans cet ordre.

En général WFQ est implémenté dans les routeurs par logiciel, ce qui limite le nombre de files d'attente pouvant être traitées. Il sera donc difficile de traiter chaque flux individuellement (WFQ-IntServ) pour lui garantir une certaine qualité de service.

Plusieurs variantes à WFQ sont proposées.

- ***CBWFQ (Class-Based Weighted Fair Queueing)***

***Figure 18:* Ordonnanceur type CBWFQ (*Class-Based Weighted Fair Queueing*)**



Ordonnacement des paquets selon Weighted Fair Queueing (WFQ)

Certains constructeurs (CISCO par exemple) proposent dans leurs routeurs un ordonnancement de type CBWFQ. C'est un ordonnancement WFQ associant les classes de service aux files. Cela décomplexifie grandement la partie logiciel WFQ du routeur afin d'obtenir de meilleures performances.

Dans la figure 17, le poids WFQ de la classe 1 est de 50 % et le poids WFQ des classes 2 et 3 est de 25%. En conséquence, chaque fois qu'un bit des classes 2 et 3 est transmis, deux bits de la classe 1 doivent être transmis. Les paquets ne pouvant être éclatés bit par bit, WFQ ne va prédire que la date de

traitement du dernier bit des paquets de classe 1 2 et 3. Ceci n'est qu'une prédiction basée sur la notion de temps virtuel, et non la vraie date de transmission effective du paquet (quand il sera transmis).

• SCFQ (Self Clocking Fair Queueing))

SCFQ (*Self Clocking Fair Queueing*) est une variante de WFQ qui propose une simplification du calcul de l'estimation du temps virtuel associé à GPS.

• WF^2Q (Worst Case Fair Queueing)

WF^2Q (*Worst-Case Fair Weighted Fair Queueing*) est une variante de WFQ qui utilise le temps de fin de traitement au temps de début de traitement d'un paquet associé afin de réaliser une meilleure approximation de GPS.

• WF^2Q+ (Worst Case Fair Queueing +)

WF^2Q+ est une amélioration de WF^2Q, qui est basée sur un autre algorithme de calcul du temps virtuel moins complexe mais étant une meilleur approximation de GPS. Sa mise en oeuvre dans les routeurs (mise en oeuvre matérielle) est cependant plus lourde.

I.5.2.6 Les délais de transmission des paquets

Il y a plusieurs éléments qui contribuent au délai de transmission d'un paquet entre la source et la destination :

• Le temps de commutation ou de routage (*Forwarding*)
C'est le temps, interne au routeur, mis par un paquet pour passer de la sortie d'une interface d'entrée à l'entrée d'une interface de sortie, c'est à dire le temps pris par le fonction de routage,

• Le temps de stockage (*bufferization*) :
c'est le temps d'attente du paquet dans sa file d'attente avant son service, en général dans l'interface de sortie,

• Le délai de sérialisation :
c'est le temps qui est mis pour envoyer le paquet sur le lien physique. Cette valeur est proportionnelle à la taille du paquet et au débit du lien physique,

• Le temps de propagation :
Il s'agit du temps mis par le premier bit pour aller de la sortie de l'interface du noeud à l'entrée de l'interface du noeud suivant (environ 75% de la vitesse de la lumière dans le cuivre, et 65% dans les fibres optiques),

• Le délai d'encapsulation et de décapsulation
Un exemple typique de ce délai est le passage d'un paquet d'un réseau IP vers un réseau ATM. La trame IP doit être découpée en cellules de 48 octets puis chaque cellule doit être augmentée de l'entête ATM de 5 octets. Cela augmente de 10% environ la tailles des données qui transitent, à cela s'ajoutant les temps d'encapsulation puis de décapsulation correspondant.

I.5.3 Le modèle Integrated Service (IntServ)

I.5.3.1 Principe de IntServ

La première proposition d'architecture Internet avec support pour la QoS a été faite par l'IETF avec le modèle IIS (*Internet Integrated Services*) ou IntServ. L'idée de base du modèle IntServ est de fournir une QoS individualisée à chaque connexion en utilisant un mécanisme de contrôle d'admission et de réservation de ressources via le protocole RSVP. Avec ce protocole, chaque routeur est interrogé sur ses ressources disponibles. S'il est possible de trouver un chemin entre les deux extrémités de connexion sur lequel tous les routeurs disposent de suffisamment de ressources pour assurer la QoS de la connexion, alors celle-ci est établie, sinon elle est refusée.

Le modèle IntServ définit trois types de service appelés GS (*Garanteed Service*), CL (*Controlled Load*) et BE (*Best Effort*):

- GS utilise des limites supérieures (prouvées mathématiquement) sur les délais d'attente de chaque flot pour fournir une QoS dite dure (limite en terme de délai d'attente dans un routeur) en réservant la bande-passante et la mémoire nécessaires. Ce service est prévu pour les applications temps-réel comme la voix [RFC2212].
- CL simule un réseau peu chargé lors de la congestion du réseau grâce à un multiplexage statistique. Il fournit de faibles taux de pertes et de faibles délais, mais reste qualitatif. De ce fait, ce service peut se dégrader si le réseau est très chargé. Il convient à des applications adaptatives [RFC2211].
- BE est le service traditionnel, sans aucune garantie de QoS.

I.5.3.2 Les composants du modèle IntServ

Outre les mécanismes de *traffic policing*, de *buffer managment* et de *traffic scheduling* déjà vus, la mise en oeuvre du modèle IntServ nécessite l'utilisation de 3 mécanismes :

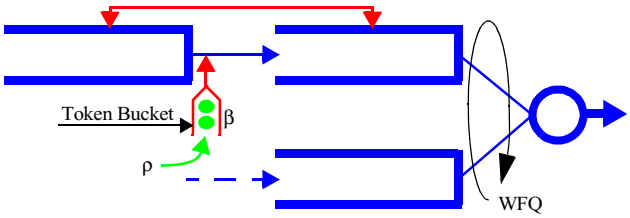
- Un mécanisme de spécification de la QoS et de caractérisation du trafic:
  Afin d'autoriser une nouvelle connexion sur le réseau, il faut spécifier les caractéristiques du nouveau flux et de la QoS demandée. Les caractéristiques du flux sont soit représentées par celles d'un *leaky bucket* ou d'un *tocken bucket* correspondant, soit en donnant les paramètres statistiques du flux (débit moyen, crête, gigue, tolérance de rafale, ...). Les caractéristiques de la QoS sont spécifiées par le délai point à point, la gigue et les taux de perte [RFC2210][RFC2215],
- Un mécanisme de contrôle d'admission :
  Afin de déterminer si un nouveau flux peut-être accepté sur le réseau, on utilise le protocole RSVP [RFC2205]. Il utilise la bande passante équivalente du flux, déduite des caractéristiques vues au point précédent et de modèles issus de la théorie des files d'attentes (notamment le modèle M/D/1/N) afin de chercher et réserver un chemin possible à travers le routeur pour le flux. Si aucun chemin n'est trouvé, la demande est rejetée.
  En cas de refus, des mécanisme supplémentaires peuvent proposer une QoS plus dégradée pour tenter d'établir le flux malgré tout,

• Un processus de négociation et renégociation de la QoS:
Ce protocole est l'un des plus importants de IntServ. Il est le medium pour le transfert d'informations sur les besoins en ressources des connexions et pour la négociation des besoins utilisateurs en QoS. Il est utilisé non seulement pour le contrôle d'admission des connexions mais aussi pour la signalisation de la QoS des connexions établies : allocation de ressources et ordonnancement des connexions.

I.5.3.3 L'ordonnanceur (*scheduler*)

L'ordonnancement réalisé dans les routeurs IntServ est de type GPS. Son principe, illustré sur la figure 19, est qu'à chaque source est associé un tampon de transmission et un *token bucket*. La régulation du flot effectuée par le *token bucket* est nécessaire même si la source est déjà régulée par un mécanisme de shaping car certains traitements dans le réseau peuvent entraîner de la gigue.

Figure 19: **Ordonnancement dans IntServ**



Considérons alors une source donnée, régulée par un token bucket de paramètres (ρ, β) et demandant un service caractérisé par un taux de transmission *R*. Le trafic généré dans un intervalle de taille *t* est borné par ρ*t*+β. Une fois la bande passante réservée au moyen de RSVP, le scheduler garantit un taux *R* de transfert pour cette session. Le délai maximum d'attente d'un paquet est β/*R*, aussi longtemps que $R < \rho$.

I.5.4 Le modèle DiffServ (Differentiated Services)

Le modèle DiffServ différencie les traitement des flux de façon indépendante dans chaque routeur, à la différence de IntServ qui considère les flux de bout en bout. Les flux sont regroupés en classes de service (par exemple temps-réel mais pertes autorisées pour la voix, sans perte mais sans garantie de délai pour le transactionnel et *best effort*).

I.5.4.1 Principe du modèle DiffServ

DiffServ n'utilise pas de mécanisme de réservation de ressources pour les flux, mais leur offre une QoS située entre le *best-effort* (IP classique) et la QoS garantie (GS d'IntServ). L'avantage de ce modèle est d'une part sa proximité avec IP (RSVP n'est plus nécessaire), ce qui permet une implémentation aisée, et d'autre part ses capacités d'évolution face à l'introduction de nouveaux services. Ce modèle est d'ailleurs d'une complexité moins grande que celle d'IntServ.
Certains bits de l'entête IP sont utilisés pour associer au paquet une classe de service (champ de l'entête appelé DSCP), et une information indiquant si le paquet peut être éliminé en cas de congestion.

I.5.4.2 Les composants du modèle DiffServ

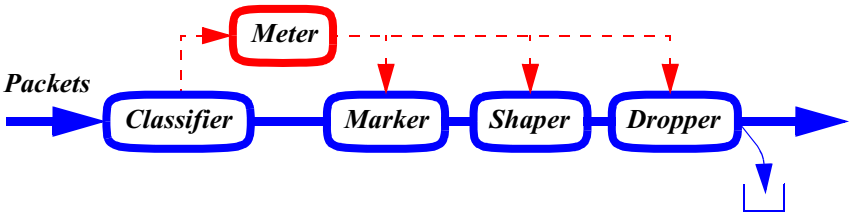
Dans un domaine DiffServ, il faut distinguer les routeurs d'accès, ou routeurs de bordure (*edge routers*), et les routeurs du coeur de réseau (*core routers*).

Les routeurs de bordure

Les routeurs de bordure sont les portes d'entrées obligatoires pour un flot pénétrant dans le domaine DiffServ. Ils effectuent des traitements (figure 20) sur les paquets entrant qui ont pour effet :

- de déterminer leur classe de service (*classifier*) grâce à l'entête du paquet (champ Type de Service (ToS), adresse source et/ou destination, protocole, port) sur la base d'un contrat avec l'opérateur,
- de leur affecter l'information d'élimination possible ou non, selon les mesures qui sont faites (*meter*) sur les débits moyens et crête, et en accord avec le contrat passé avec l'opérateur. En cas de congestion dans le coeur de réseau, un paquet éliminable (*out profile*) sera traité dans une classe moins prioritaire, voire détruit.
- de réguler leur débit selon leur classe, souvent par un *token bucket* (*shaper*),
- de les détruire ou de les déclasser (*dropper, policer*) en cas de congestion (s'ils sont marqués *out profile*). L'opération de déclassement n'est possible que dans les routeurs de bordure.

Figure 20: Traitement des paquets dans un routeur de bordure



Les routeurs de coeur de réseau

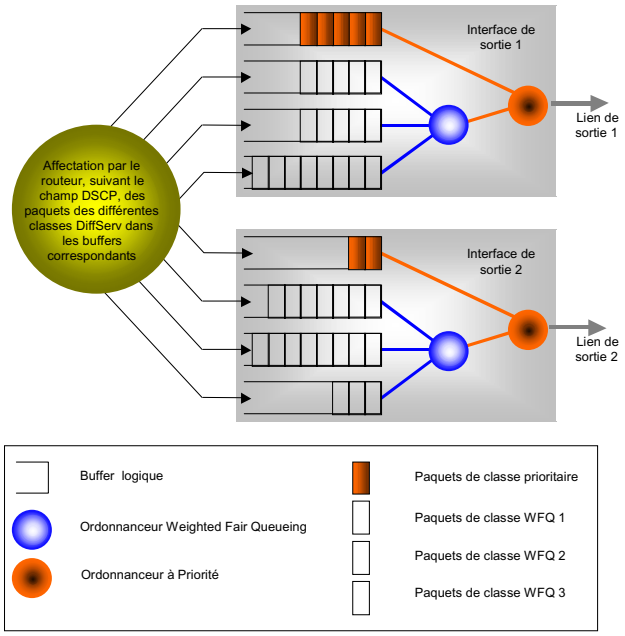
Les routeurs du coeur de réseau réalisent des opérations simples de *bufferization* et de routage des paquets en se basant uniquement sur le marquage effectué par les routeurs situés en bordure de domaine DiffServ. La différenciation de service se fait au niveau des deux mécanismes cruciaux du modèle DiffServ: le *scheduling (*I.5.2.4) et le *buffer management* (I.5.2.3).
Chaque sortie du routeur possède un nombre fixe de files logiques où le routeur dépose les paquets arrivant selon leur classe de service. Les files sont servies en accord avec l'algorithme d'ordonnancement.

Le routeur de coeur de réseau est constitué de trois fonctions principales :

- **Routage** : Il consiste à la détermination de l'interface suivante vers laquelle diriger le paquet,

- **Gestion de tampon** : les algorithmes utilisés sont le WRED ou RIO. On a ainsi des éliminations de paquets sélectives en fonction de la classe de trafic. La configuration de ces algorithmes est toutefois relativement délicate.
- **Ordonnancement** : Plusieurs politiques d'ordonnancement peuvent être utilisées : WFQ, WRR ou priorités fixes.
  A l'heure actuelle, il semblerait que les opérateurs s'orientent vers une combinaison de ces politiques avec une priorité fixe pour le trafic temps réel et un ordonnancement WFQ pour les autres classes. C'est le champ DSCP (classe de service) dans l'entête des paquets qui permet de les affecter à une file d'ordonnancement particulière. La figure 21 montre l'affectation qui est faite en fonction du routage (choix de l'interface de sortie) et de la classe de service (choix du tampon logique de la classe de service).

Figure 21: **Routeur DiffServ de coeur de réseau**



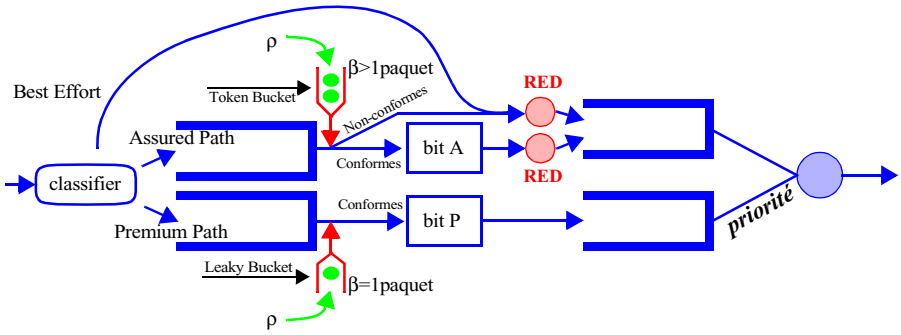
Contrairement à IntServ, il n'est plus nécessaire pour DiffServ d'associer un régulateur de type *token bucket* à chaque file des routeurs de coeur de réseau (*core router*). En effet, tous les trafics entrant dans le coeur de réseau ont déjà subi la fonction de police dans les routeurs de bordure (*border router*), et l'on vérifie à posteriori que la gigue n'est pas modifiée par les *core routers* s'ils sont suffisamment rapides (ils sont en général sur dimension nés).

I.5.4.3 Exemple d'architecture DiffServ

Une des premières architectures proposées pour DiffServ est l'architecture TDSA (*Two-bit Differentiated Service Architecture :* Architecture de service codée sur deux bits). Cette architecture comprend trois classes de service codées sur deux bits: 10 = *Premium*, 01 = *Assured Forwarding*, 00 = *Best-Effort*.

Le *Premium service* est destiné au trafic temps réel (voix en particulier). L'*Assured Forwarding* reçoit un meilleur traitement que le *Best-Effort* (moins de pertes). Le *Premium service* est régulé par un *leaky bucket* et les paquets non-conformes (*out profile*) sont systématiquement supprimés. Le service *Assured Forwarding* offre la garantie que si les paquets restent dans le profil négocié (par un *token bucket*), ils ne sont supprimés qu'en dernière extrémité. Par contre, s'ils sont hors-profil, ils peuvent être déclassés (en *Best-Effort*) par le routeur de bordure. La figure 22 décrit l'architecture générale d'un routeur de frontière.

Figure 22: **Principe de fonctionnement d'un *Two-bit edge router* (frontière*)***



Au niveau du routeur de bordure, le *P-bit classifier* sépare les paquets marqués 10 (*Premium service*) des paquets marqués 01 (*Assured Forwarding*) et de ceux marqués 00 (*Best Effort*). Si les paquets sont marqués 10 ou 01, ils sont alors marqués suivant leurs champs IP puis sont régulés suivant le profil associé (SLA) avec un *leaky bucket* pour le service *Premium* ou avec un *token bucket* pour l'*Assured Forwarding*.

Au niveau des routeurs de coeur de réseau, la classe *Premium* a une priorité haute tandis que les autres classes (*Assured* et *best-effort*) ont une priorité basse. Le *scheduling* étant à priorités fixes, la classe *Premium* n'est que peu gênée par les autres. Dans la file à faible priorité, la probabilité d'élimination des paquets est plus faible pour la classe *Assured Forwarding* (principe de RIO: deux niveaux WRED).

I.6 Le protocole MPLS

I.6.1 Les principes de base

I.6.1.1 Introduction

Avec l'augmentation du débit des liens de communication et du nombre de routeur dans l'Internet, l'IETF a défini le nouveau protocole MPLS (*Multi Protocol Label Switching* : commutation de label multi-protocol) pour faire face au goulot d'étranglement induit par la complexité de la fonction de routage. Ce protocole a deux objectifs principaux qui sont :

- Permettre un acheminement rapide des paquets IP en remplaçant la fonction de routage par une fonction de commutation qui est beaucoup plus rapide, la taille des matrices de commutation devenant très petite,
- Faciliter l'ingénierie réseau en fournissant aux opérateurs la maîtrise de l'acheminement des données, qui pouvaient être très complexes avec des protocoles comme OSPF (gestion des métriques).

Le protocole MPLS basé sur le paradigme de la commutation de label dérive directement de l'expérience acquise avec ATM (étiquettes VPI/VCI : *Virtual Path/Channel Identifier*, ATM : *Asynchronous Transfer Mode*). Ce mécanisme est aussi similaire à celui de *Frame Relay* ou des liaisons PPP (PPP: *Point to Point Protocol*). L'idée de MPLS est de rajouter un label de niveau liaison point-à-point (niveau 2 dans la hiérarchie ISO) aux paquets IP dans les routeurs de frontière (routeurs de bordure de coeur de réseau). Le réseau MPLS va pouvoir créer des LSP (*Label*

Switching Path : Chemin à commutation de label), similaires aux VPC (*Virtual Path Connection* : Chemin de connexion virtuelle) d'ATM. Ces LSP définissent une route de bout en bout dans le coeur de réseau, par une concaténation de labels. Les routeurs du cœur de réseau MPLS, appelés LSR (*Label Switching Routers* : Routeurs à commutation de label), vont acheminer les paquets de proche en proche par commutation de labels, sans tenir compte des adresses IP.

Le fonctionnement général est le suivant : Dans un LER (*Label Edge Router* : Router à label de bordure) situé sur la bordure du réseau MPLS, le label est rajouté à l'entête IP. Une fois que le paquet a traversé le coeur de réseau (les LSR), le LER de sortie retire le label ce qui permet de retrouver le paquet IP originel.

Les grandes innovations par rapport au routage IP traditionnel (IGP, EGP), sont :

- La manipulation de tables de routage de bien plus petite taille et des champs de petite taille pour les labels conduisant ainsi à des temps de commutation extrêmement rapides (traitement matériel plutôt que logiciel),
- La détermination des LSP (les chemins) est basée sur d'autres critères que l'adresse de destination uniquement (classe de service, groupes d'adresses de destination),
- Un ensemble de règles et de paramètres permettant de calculer un routage « optimal » (autre que le plus court chemin) pour l'ingénierie du trafic et la qualité de service,
- Des mécanismes de reroutage en cas de panne avec la possibilité de réserver des LSP de secours pour certains trafics.

I.6.1.2 Principes de MPLS

MPLS est particulièrement adapté au modèle DiffServ. En effet, on y retrouve la notion de routeurs de bordure et de routeurs de coeur de réseau (*Edge/Core router*) ainsi que la notion d'agrégation de flots.

L'entête MPLS de 32 bits contient :

- Un label de 20 bits (0 à 1023) servant à identifier le LSP emprunté par le paquet,
- Un champ de 3 bits (0 à 7) contenant un identificateur appelé le PHB (*Per Hop Behaviour* : Comportement par noeud) correspondant à la classe de service DiffServ affectée au paquet grâce à une table de correspondance (*classifier*) dans chaque routeur,
- un bit permettant l'empilement de labels MPLS,
- une durée de vie sur 8 bits (pour faire face aux erreur de routage)

Classes d'équivalences pour le routage : FEC

Lorsqu'un paquet entre dans le coeur de réseau, une entête MPLS lui est ajouté, qui détermine le LSP dans lequel seront routés les paquets. Afin de déterminer cet entête, la FEC (*Forwarding Equivalent Class* : classe d'équivalence pour le routage) et le type de service du paquet jouent un rôle déterminant. Un flot de donnée peut cependant ne pas être encapsulé dans un LSP. Le routage IP classique (et plus lourd) se chargera de déterminer sa route dans le coeur de réseau.

Une FEC est une liste d'adresses IP, chaque adresse correspondant à une machine particulière

ou une adresse de réseau. Cette spécification permet de regrouper par destination les flux d'un même LSP.

Routeurs de bordures : LER

Les routeurs de bordure d'un réseau MPLS sont appelés LER (*Label Edge Router* : Routeur de bordure à label). Le label MPLS encapsule le paquet au niveau du LER d'entrée et est retiré au niveau du LER de sortie.

L'allocation d'un label peut se faire de plusieurs manières. Un label identifie nécessairement un LER d'entrée et un LER de sortie, mais permet de regrouper les flux de façon assez souple. Il peut représenter un port d'entrée, des adresses (de bout-en-bout) source et destination, des classes de services, ou une combinaison de tout cela. La connexion entre deux LER est un circuit virtuel appelé LSP (*Label Switched Path* : Chemin à commutation de label) qui est faite au travers des routeurs du coeur de réseau MPLS : les LSR.

Le routage des paquets IP dans le réseau MPLS est entièrement déterminé au niveau du LER d'entrée par l'affectation d'un label : le label définit un LSP et pourra transporter des paquets dont les adresses de destination correspondent à une FEC et appartenant à un certain ensemble de classes de service.

Routeurs de coeur de réseau: LSR

Au niveau des LSR (*Label Switch Router:* Routeurs à commutation de label), la fonction de routage, que l'on préfère qualifier de commutation, est très simple, afin d'être très rapide (traitement matériel). Tout routeur LSR possède une table de commutation où à chaque label entrant sur un port donné correspond un label et un port de sortie. La taille de cette table de commutation est relativement réduite car elle est locale au coeur de réseau, et découplée du routages IP.

I.6.1.3 Principe MPLS

Chaque LSP a donc pour caractéristique un ensemble de FEC, et un ensemble de classes de services. Une table de correspondances dans chaque routeur de bordure (LER) permet d'affecter chaque flux (en fonction de sa classe et de sa destination) à un LSP. Cette fonction est aussi lourde que la fonction de routage dans les réseaux IP.
Une fois le paquet encapsulé, il sera routé dans son LSP via les routeurs de coeurs de réseau (LER) grâce à leur table de commutation, la fonction de commutation étant beaucoup plus rapide que la fonction de routage IP.
Une fois arrivé dans le LER de sortie, l'entête MPLS est retiré du paquet, et celui-ci est relâché dans le réseau IP pour continuer sa route.

Les LER d'entrée sont appelés *Ingress*, et les LER de sortie *Egress*.

Grâce à la possibilité d'empilement d'entêtes MPLS, certains LSP ayant une portion de chemin commune peuvent être agrégés, permettant par exemple la création de VPN (*Virtual Private Network*).

I.6.1.4 Fonctionnement de base du LSP

Les LSP sont des chemins de LER à LER positionnés par l'opérateur. La construction de ces chemins se fait dans les LSR, qui insèrent de proche en proche leur identification dans des messages de contrôle du protocole LDP (*Label Distribution Protocol* : Protocole de distribution de labels) ou RSVP [RFC2205]. Ces protocoles permettent la détection des boucles de routage par la vérification de l'unicité des identificateurs des LSR dans le LSP ainsi construit. Ces messages se présentent sous forme d'une table de routage de type *next-hop* à la manière du routage IP. Une fois le LSP mis en place, les paquets IP arrivant dans un LER peuvent être étiquetés par l'entête MPLS (le label) et être commutés jusqu'au LER de sortie.

I.6.2 MPLS et l'ingénierie de trafic

I.6.2.1 Motivation

Comme on a pu le voir précédemment avec la description des protocoles de routage RIP ou OSPF, le routage IP est assez pauvre et se limite la plupart du temps à un routage au plus court chemin vers une destination au travers d'une métrique plus ou moins sophistiquée. Il n'est donc pas possible avec ces protocoles de réaliser un routage qui tienne compte de contraintes de QoS ou des propriétés spécifiques de chacun des flux. MPLS offre de multiples possibilités pour réaliser cette ingénierie du trafic.

I.6.2.2 Concept de MPLS-TE

MPLS-TE (*MPLS - Traffic Engineering*) est basé sur le concept de routage de tunnels (*traffic trunk*). Le tunnel est unidirectionnel et est défini par deux LER (*Ingress* et *Egress*), la ou les FEC transportées et des attributs. Le chemin emprunté par le tunnel est un LSP, un LSP peut contenir un ou plusieurs tunnels.
Plusieurs LSP peuvent exister entre deux LER, et emprunter des chemins différents. Cela permet d'une part d'offrir des routes différentes correspondantes à la qualité de service requise pour les flux transportés grâce à la répartition très fine qui peut être faite, et d'autre part de créer des LSP de secours en cas de panne ou de surcharge sur les LSP initiaux.
La spécification des agrégats DiffServ (BA: *Behaviour Aggregate*) est à la charge de l'administrateur. Ce choix a pour effet de séparer ou de regrouper un ensemble de tunnels dans un ou plusieurs LSP, dans le but de protéger certains tunnels, ou de faire subir des traitements différents dans les LSR aux autres par exemple.

L'encapsulation MPLS d'un paquet dans un LSP se fait en connaissant sa classe de service et le routeur *Egress* par lequel il est censé passer selon le routage IP. Le LSP ayant la FEC correspondante sera alors emprunté, et tout se passe comme si le routage initial était oublié jusqu'à ce que le paquet soit décapsulé dans le routeur *Egress*. Plusieurs FEC peuvent correspondre au même *Egress*, la priorité est alors donnée aux LSP ayant la FEC qui caractérise le plus précisément le routeur *Egress*, et qui n'est pas incompatible avec l'adresse de destination réelle du paquet.

I.6.2.3 Ingénierie de trafic avec MPLS-TE

La création des LSP se fait en plusieurs étapes. Une fois les tunnels déterminés (groupes de flux

ayant la même FEC et le même couple *Ingress-Egress*), puis une fois les tunnels ayant le même couple *Ingress-Egress* regroupés en un ou plusieurs LSP, il faut créer les routes dans le coeur de réseau pour propager ces LSP.

Routage des LSP

Il est possible de fournir une route précalculée pour chaque LSP. Il faut alors donner la liste de routeurs traversés afin que le protocole RSVP-TE, sans faire de réservation, configure les tables de labels de l'ensemble des routeurs empruntés.
Il est également possible de ne rien spécifier, ou uniquement une partie de la route qui doit être empruntée par le LSP. Dans ce cas, le protocole RSVP-TE a la charge de créer la partie manquante de la route, en utilisant le protocole OSPF-TE, qui est la version adapté du protocole OSPF pour les réseaux MPLS-Diffserv. La métrique employée pour la recherche des plus courts chemins peut-être différente pour chaque classe de service considérée, ce qui permet d'optimiser les routes en fonction du type de trafic que l'on veut propager. Par exemple, la métrique sera le délai pour le trafic de type voix, ou le nombre de sauts pour les trafics de type donnée.

Création automatique des LSP

Dans le cas de la création automatique des LSP, le protocole RSVP-TE doit également faire de la réservation de bande passante. Les tunnels ont un certain nombre d'attribut dont il faut tenir compte pour les propager et créer les LSP correspondants.

Une première contrainte est l'affinité, c'est un masque qui autorise certains types de trafics sur certains type de liens (lien crypté, lien satellite...).
La recherche du ou des plus courts chemins au sens de la métrique considérée pour chaque tunnel commence alors. Les tunnels sont considérés en fonction du champ *setup priority* (priorité de mise en place). Dans le cas où des LSP sont déjà existant, et que la bande passante restante est insuffisante, les LSP ayant une *holding priority* (priorité de fonctionnement) inférieure à la *setup priority* du tunnel à placer sont retirés de l'algorithme de routage (dans le cas où cette route est retenue pour le tunnel, les ressources de ces LSP moins prioritaires sont libérées, et les LSP sont supprimés ou reroutés sur leur LSP de secours, ou sur un nouveau chemin si cela est possible), il s'agit du mécanisme de contrôle d'admission (LCALC). Le protocole OSPF-TE étant capable de trouver des plus courts chemins de coût équivalents, le groupe de tunnels à propager va pouvoir être distribué sur plusieurs LSP. La priorité est cependant donnée aux chemins ayant la plus grande bande passante résiduelle.

I.6.2.4 Routage des LSP et OSPF-TE

Plusieurs extensions ont été rajoutées à OSPF afin de pouvoir faire de l'ingénierie de trafic. On aurait pu, à la place du terme TE, utiliser le terme « attributs de liens étendus », qui reflète plus la réalité de l'évolution du protocole OSPF (au même titre par exemple qu'OSPF avec TOS, voir I.4.4.2).

OSPF-TE ajoute dans ses LSA des attributs supplémentaires qui permettent de construire une base de données de trafic engineering plus sophistiquée que la base de données standard. Les fonctionnalités sur cette base de données sont le contrôle des nouveaux attributs de liens, le routage à la source avec contraintes induites par les attributs TE et ingénierie de trafic global.

Ces attributs (TLV: *Type/Length/Value*: type/longueur/valeur, inclus dans les LSA) intègrent notamment la métrique de la classe de service TE, la bande passante maximum, réservable, et disponible du lien, ainsi que sa couleur (ou affinité).

I.6.3 Mécanismes de protection dans MPLS

Certaines pannes demeurent invisibles du point de vue MPLS. Ce sont par exemple les pannes survenant sur les liens physiquement protégés, comme la technologie SDH qui intègre une procédure propre de contournement de panne grâce à une boucle optique, ou la redondance matérielle (plusieurs interfaces physiques reliées sur un même bus sur lequel une seule est active).

Les pannes sont souvent le fait des chantiers des travaux publics (tranchées). Les liens physiques sont dans ce cas coupés, et le trafic ne peut plus s'écouler. Le temps de récupération sur une panne doit être typiquement inférieur à 60 millisecondes.

I.6.3.1 Détection des erreurs et mécanismes

La détection des pannes et la propagation de cette information jusqu'aux routeurs responsables du re-routage est une fonctionnalité fondamentale dans les réseaux. Parmi les mécanismes de détection on peut citer, par ordre de rapidité, la perte de connectivité (niveau 1: électronique ou optique), par les protocoles de niveau 2 (liaison, par exemple SDH) ou de niveau 3 (protocoles intégrant des échanges périodiques entre routeurs (*Hello* dans OSPF)). Dans chaque cas, l'information doit remonter aux équipements MPLS chargés de réorienter le routage, via les protocoles de signalisation MPLS (RSVP, LDP).

I.6.3.2 Les mécanismes de protection et récupération

FRR (Fast Re-Routing)

La sécurisation temporaire du chemin d'un tunnel protégé en cas de panne d'un lien ou d'un routeur est faite par FRR (*Fast Re-Routing*). Le principe de FRR est de pré-instancier un LSP de secours sans réservation physique de la bande passante, mais simplement avec la distribution de labels. Lors de l'activation de ce LSP (suite à une panne), la réservation de la bande passante pour le trafic protégé est faite en positionnant le champ *setup prio* et *holding prio* au maximum. Ainsi, ce LSP ne bloque pas de ressource quand il est inactif. Ce mécanisme FRR permet au réseau d'éviter la panne par des chemins de secours en attendant que le LER d'entrée mette en place un LSP secondaire pour le tunnel.

Ce mécanisme suppose que toutes les pannes possibles sur la topologie aient été prévues à l'avance. Il peut protéger soit un lien, soit un routeur. L'évitement de la panne commencera alors respectivement sur le routeur dont un lien est en panne, ou sur le routeur précédent le routeur en panne.

Protection de chemin

Une fois l'information remontée au *Ingress*-LER, le mécanisme de protection de chemin peut alors être mis en place. Il consiste en le calcul d'un second LSP qui ne passe par aucun des routeurs

du LSP primaire. Ce LSP peut avoir été pré-alloué ou non. S'il ne l'a pas été, il faut utiliser la procédure de création automatique vu au I.6.2.3. S'il l'a été, il n'y a rien à faire si le mécanisme de protection 1:1 est employé (les données sont toujours dupliquées sur les deux LSP). Si les données ne sont pas dupliquées, il faut encapsuler le LSP en panne dans le LSP secondaire (par empilement de label), en supprimant si nécessaire les éventuelles connexions de priorité moindre qui l'utilisent.

I.7 Problématique

La maîtrise des réseaux à échelle nationale, voire mondiale tels que ceux qui ont été décrits dans ce chapitre, dotés de tous les protocoles de transport, de routage, de commutation, de tous les mécanismes de gestion de la qualité de service est très complexe, d'autant plus que ces réseaux sont en évolution permanente. De nouvelles technologies performantes sont sans cesse développées et utilisées parallèlement à l'existant ce qui implique une complexité sans cesse croissante.

L'évaluation de performance de ces réseaux suppose l'utilisation de modèles précis et de techniques de simulation performantes. Dans la suite, nous aborderons différentes manières de simuler ces réseaux afin d'en décrire les comportements et d'en évaluer les performances. Ceci constitue une première étape offrant une base solide pour permettre une utilisation efficace de l'ensemble des systèmes décrits dans ce chapitre, et constituer une base de données précise pour optimiser l'utilisation de ces réseaux.

Chapitre II - Modélisation différentielle du trafic

Dans ce chapitre, nous rappelons quelques résultats de la théorie différentielle du trafic qui constituent la base de la simulation hybride. Dans une deuxième partie, nous aborderons le paradigme GPS. Le modèle du paradigme GPS proposé est une approximation par linéarisation à trois classes, et enfin nous conclurons.

II.1 Théorie différentielle du trafic

La théorie différentielle du trafic a été développée au début des années 1980 par Garcia et Legall pour les réseaux à commutation de circuits [GAR86][GAR80]. Cette théorie a été récemment étendue aux réseaux à commutation de circuits multiclasses (ATM, STM) [BRU00a] et aux réseaux à commutation de paquets [BRU00a][GAR98].

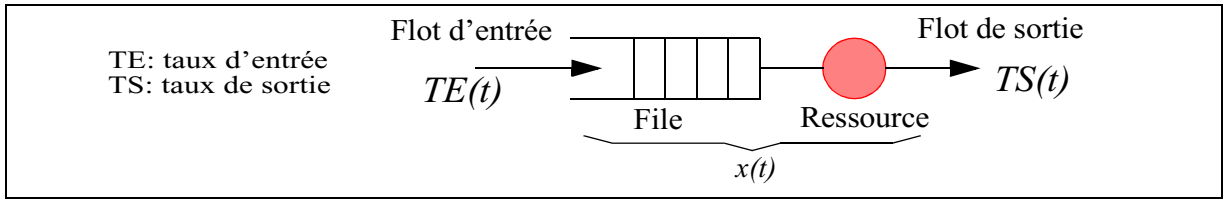
La théorie différentielle du trafic définit une méthodologie de modélisation analytique fluide qui permet l'étude du trafic sur les ressources d'un réseau aussi bien en régime stationnaire qu'en régime transitoire. Les modèles développés sont en général assez précis et d'une faible complexité calculatoire.

Nous décrivons tout d'abord au paragraphe II.1.1 le principe général de la modélisation différentielle. Puis nous montrons au paragraphe II.1.2 comment cette technique peut être appliquée pour obtenir les modèles différentiels de quelques systèmes élémentaires markoviens. Comme montré au paragraphe II.1.3, cette technique n'est toutefois pas uniquement limitée à des systèmes markoviens. Finalement, nous montrons comment elle peut être étendue à des réseaux de files d'attente au paragraphe II.1.4.

II.1.1 Principe général de la théorie différentielle du trafic

Considérons le système représenté sur la figure 23. Soit *x(t)* la variable aléatoire associée au nombre de paquets dans ce système. Notons *TE(t)* et *TS(t)* respectivement les débits instantanés d'entrée et de sortie du système.

Figure 23: **Modèle de file d'attente d'une ressource réseau**



L'idée fondamentale de la théorie différentielle du trafic est d'établir l'expression exacte de l'équation différentielle gouvernant le trafic moyen transitoire $X(t) = E[x(t)]$. Cette équation est obtenue par une analyse du processus markovien ou semi-markovien associé à *x(t)*. La forme générale de cette expression est la suivante (où $P[x(t)]$ est la probabilité de l'état $x(t)$) :

$$\frac{d}{dt}X(t) = TE(t) - TS(t) = \sum_i x_i(t) \cdot \frac{d}{dt}P[x_i(t)] \qquad X(t_0) \text{ donné} \qquad (5)$$

Cette expression est très générale : elle exprime tout simplement que la variation du nombre moyen de paquets dans le système est donnée par la différence entre le débit instantané d'entrée $TE(t)$ et celui de sortie $TS(t)$.

Des expressions explicites de l'équation différentielle (5) ont été obtenues pour tout une batterie de modèles de files d'attente élémentaires (markoviens ou non). Ces équations différentielles font toujours intervenir des probabilités d'états transitoires dont l'expression analytique n'est en général pas connue. La théorie différentielle du trafic procède alors à une approximation de ces équations différentielles exactes en utilisant une relation implicite entre les probabilités d'états et le trafic moyen dans le système. On peut ainsi obtenir le comportement en régime dynamique et à l'équilibre du système en utilisant une méthode d'intégration numérique (méthode d'Euler ou de Runge-Kutta par exemple).

II.1.2 Modèles dynamiques de systèmes markoviens élémentaires

Dans ce paragraphe, nous considérons les files d'attente suivantes : $M/M/1/\infty$, $M/M/1/N$. et $M^k/M^k/1/\infty$.

II.1.2.1 Modèle dynamique de la file $M/M/1/\infty$

La file $M/M/1/\infty$ est un modèle très simple souvent utilisé pour représenter un lien de transmission. Il s'agit d'un modèle à capacité infinie. Les clients arrivent suivant un processus de Poisson de paramètre λ. Ils sont servis suivant la discipline PAPS (premier arrivé premier servi) et la durée du service de chaque client est une variable aléatoire exponentiellement distribuée de moyenne $1/\mu$.

Si nous prenons pour variable d'état $x(t)$ le nombre de clients dans le système, alors la chaîne de Markov associée est un processus de naissance et de mort ergodique de paramètres l et μ à condition que le taux d'utilisation $\rho = \lambda/\mu < 1$. Notons $Pi(t)$ la probabilité que $x(t)=i$. Il est bien connu qu'à l'équilibre, on a :

$$X(\infty) = \frac{\rho}{1-\rho} \qquad \text{et} \qquad P_i(\infty) = (1-\rho)\rho^i \qquad i = 0\ldots\infty \tag{6}$$

L'équation différentielle exacte du trafic moyen transitoire $X(t)$ de la file $M/M/1/\infty$ s'écrit :

$$\frac{d}{dt}X(t) = \sum i\frac{d}{dt}P_i(t) = \lambda - \mu(1-P_0(t)) \tag{7}$$

Malheureusement, cette équation fait intervenir la probabilité transitoire $P_0(t)$ qui a l'expression complexe suivante (où i représente le nombre initial de clients) [KLE75] :

$$P_k(t) = e^{-(\lambda+\mu)t}\left[\rho^{\frac{k-i}{2}} I_{k-i}(at) + \rho^{\frac{k-i-1}{2}} I_{k+i+1}(at) + (1-\rho)\rho^k \sum_{j=k+i+2}^{\infty} \rho^{\frac{-j}{2}} I_j(at)\right]$$

$$\text{avec} \quad a = 2\mu\sqrt{\rho} \quad \text{et} \quad (\forall k \geq -1),\ I_k(x) = \sum_{m=0}^{\infty} \frac{\left(\frac{x}{2}\right)^{k+2m}}{(k+m)!m!} \tag{8}$$

En pratique, cette expression analytique ne peut être utilisée pour calculer la valeur exacte de $P_0(t)$ car elle fait intervenir des sommes infinies. Une autre possibilité serait de calculer le vecteur de probabilités par intégration numérique de la chaîne de Markov. Mais là encore cette solution est

limitée par la dimension infinie de l'espace d'états. Bien que dans les deux cas des approximations soient possibles en tronquant les calculs, les approximations résultantes ne sont pas très efficaces d'un point de vue numérique (ce qui est essentiel pour de grands réseaux).

La solution proposée dans le cadre de la théorie différentielle du trafic consiste à transformer l'équation différentielle associée à $X(t)$ en une équation différentielle autonome en utilisant une approximation basée sur une relation implicite entre le trafic moyen $X(t)$ et $P_0(t)$. Plus précisément, on observe qu'à l'équilibre, la relation suivante est vérifiée :

$$\rho = 1 - P_0(\infty) = \frac{X(\infty)}{1+X(\infty)} \qquad (9)$$

L'idée est alors d'étendre la validité de cette relation au régime transitoire :

$$1 - P_0(t) = \frac{X(t)}{1+X(t)} \qquad (10)$$

On obtient ainsi l'approximation dynamique suivante pour la file $M/M/1/\infty$:

$$\frac{d}{dt}X(t) \approx \lambda - \mu\frac{X(t)}{1+X(t)} \qquad X(t_0) \text{ donné} \qquad (11)$$

Il s'agit d'une équation différentielle autonome qui permet de calculer efficacement $X(t)$ par intégration numérique. Il est facile de montrer que cette approximation différentielle converge vers la valeur stationnaire exacte $X(\infty) = \rho/(1-\rho)$. Remarquons également que bien que l'équation (11) permette d'obtenir une expression explicite de $X(t)$, il est plus intéressant en pratique d'utiliser cette forme « entrée/sortie » (II.1.4).

Tableau 2: **Comparaison des valeurs de $X(\infty)$ obtenues avec la formule exacte et avec le modèle différentiel**

	Solution exacte	Modèle différentiel
$\rho = 0.1$	0.1111	0.1111
$\rho = 0.5$	1.0000	0.9999

Le tableau 2 présente une comparaison des valeurs stationnaires de $X(t)$ obtenues en utilisant la formule exacte et en intégrant le modèle différentiel pour ρ=0.1 et ρ=0.5. On peut constater que le modèle différentiel converge bien vers les valeurs stationnaires exactes. Pour les mêmes valeurs de ρ, la figure 24 compare le comportement transitoire de $X(t)$ obtenu avec le modèle différentiel à celui obtenu avec 10 000 simulations de Monte-Carlo. On peut voir que le modèle différentiel se révèle assez précis : pour ρ=0.1, les erreurs moyennes et maximales sur l'intervalle [0,10] sont respectivement 3.3% et 10%, tandis que pour ρ=0.5, ces erreurs sur l'intervalle [0,30] sont respectivement de 4.4% et 8.1%. Mais l'avantage principal du modèle différentiel est qu'il permet de déterminer la réponse dynamique du système à des variations de trafic, comme illustré sur la figure 25 (dans ce cas, l'erreur moyenne est de 4.4% est l'erreur maximale de 13.4%).

Figure 24: *M/M/1/∞* : comparaison entre le modèle analytique et des simulations

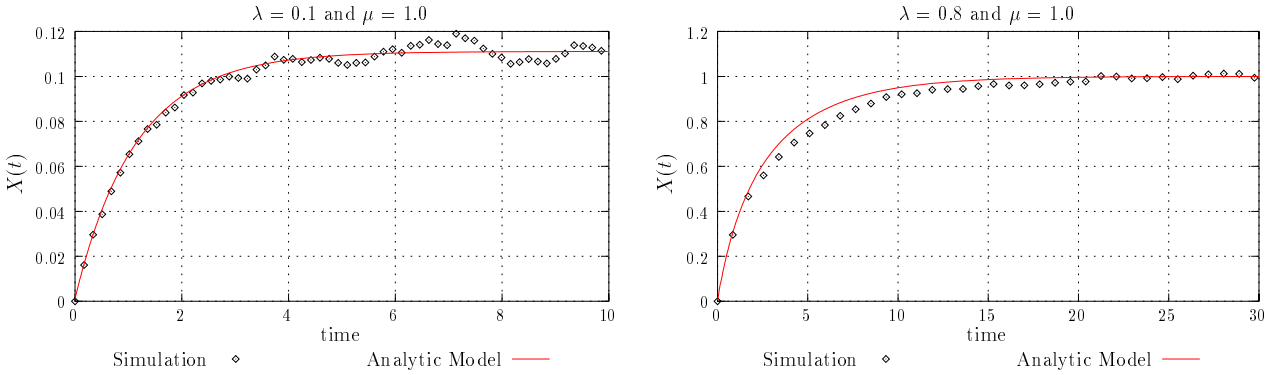
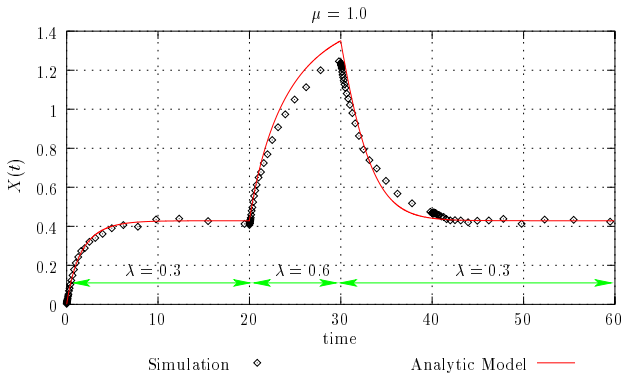


Figure 25: *M/M/1/∞* : comparaison entre le modèle analytique et des simulations dans le cas de variation de trafic



On peut également comparer la précision de l'approximation (11) à celle d'une approximation par un processus de diffusion. Soit *z(t)* le processus de diffusion approximant le nombre de clients dans une file d'attente ayant des durées inter-arrivées de moyenne μ_a et de coefficient de variation au carré c_a, et des durées de service de moyenne μ_s et de coefficient de variation au carré c_s. La densité de probabilité associée à *z(t) (z(0)=z0)* est donnée par ([KOB74]) :

$$p(z,\, t;z_0,\, 0) \;=\; \frac{\partial}{\partial x}\left\{ \Phi\left(\frac{z - z_0 - \beta t}{\sqrt{\alpha t}}\right) - e^{\frac{2\beta z}{\alpha}}\, \Phi\left(-\frac{z + z_0 + \beta t}{\sqrt{\alpha t}}\right) \right\} \qquad (12)$$

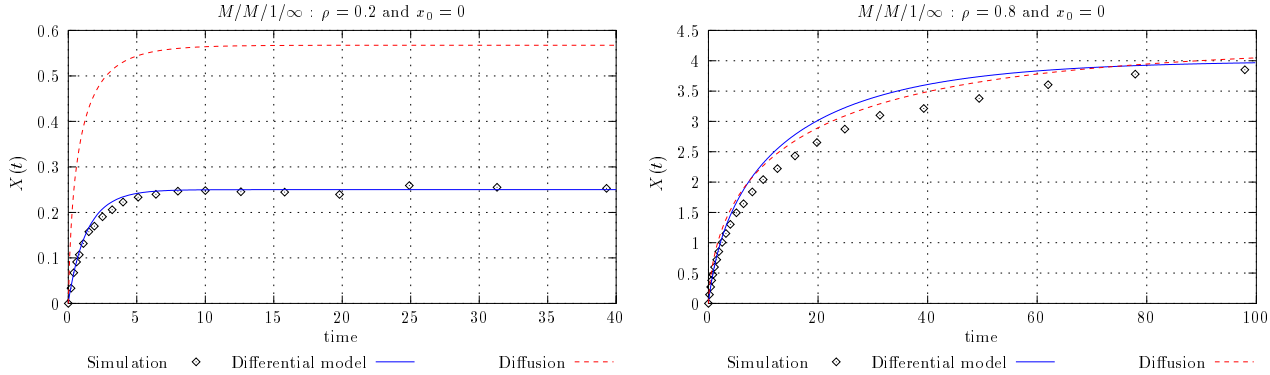
où

$$\Phi(z) \;=\; \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}}\, du \qquad \alpha \;=\; \frac{c_a}{\mu_a} + \frac{c_s}{\mu_s} \qquad \text{et} \qquad \beta \;=\; \frac{1}{\mu_a} + \frac{1}{\mu_s} \qquad (13)$$

Des comparaisons des réponses transitoires obtenues avec le modèle différentiel (11), avec cette approximation diffusion et avec des simulations événementielles sont proposées sur la figure 26 dans le cas d'une file *M/M/1/∞* ayant pour état initial x_0=0. Pour ρ=0.8, on peut observer que l'approximation diffusion est légèrement plus précise que l'approximation différentielle. Cependant, l'approximation diffusion est très grossière pour ρ=0.2. Il s'agit là d'un défaut bien connu des approximations diffusion dû à l'hypothèse sous-jacente d'indépendance des processus d'arrivées et de départs.

Figure 26: *M/M/1/∞* : réponses transitoires obtenues avec le modèle différentiel, l'approximation

diffusion et avec des simulations événementielles quand x_0=0



II.1.2.2 Modèle dynamique de la file $M/M/1/N$

Dans les réseaux à commutation de paquets, certaines ressources peuvent avoir des capacités finies conduisant à la perte de paquets. Le modèle $M/M/1/N$ permet d'étudier de tels systèmes.

La file $M/M/1/N$ est un système à capacité finie, avec N-1 positions d'attente. Les clients arrivent suivant un processus de Poisson de taux l. Ils sont servis suivant la discipline PAPS et la durée du service est une variable aléatoire exponentiellement distribuée de moyenne $1/\mu$. Les clients qui à leur arrivée trouvent un système plein sont rejetés et n'ont plus aucune influence sur le système. Ainsi, la capacité finie agit comme un régulateur sur le nombre de clients en attente, et par conséquent aucune hypothèse sur le taux d'utilisation $\rho = 1/\mu$ n'est nécessaire.

La distribution de probabilités à l'équilibre est donnée par :

$$P_k(\infty) = \rho^k P_0(\infty) = \begin{cases} \dfrac{1-\rho}{1-\rho^{N+1}}\,\rho^k & \text{si} \quad \rho \neq 1 \\ \dfrac{1}{N+1} & \text{si} \quad \rho = 1 \end{cases} \tag{14}$$

et le nombre moyen de clients dans le système à l'équilibre est,

$$X(\infty) = \sum_{k=1}^{N} k P_k(\infty) = \begin{cases} \dfrac{N\rho^{N+2} - (N+1)\rho^{N+1} + \rho}{1-\rho-\rho^{N+1}+\rho^{N+2}} & \text{si} \quad \rho \neq 1 \\ \dfrac{N}{2} & \text{si} \quad \rho = 1 \end{cases} \tag{15}$$

En appliquant la même technique que pour le modèle $M/M/1/N$, il est facile de prouver que l'équation différentielle exacte gouvernant le nombre moyen de clients en régime transitoire est :

$$\frac{d}{dt}X(t) = \lambda\,[1 - P_N(t)] - \mu\,[1 - P_0(t)] \tag{16}$$

Malheureusement, il n'existe pas d'expression simple de $P_0(\infty)$ et $P_N(\infty)$ en fonction de $X(\infty)$. Néanmoins, de façon à pouvoir appliquer le même principe que précédemment, on peut calculer numériquement $P_0(t)$ et $P_N(t)$ en fonction de $X(t)$. L'approximation différentielle est alors la suivante :

$$\frac{d}{dt}X(t) \approx \lambda\left(\frac{1-\rho^N(t)}{1-\rho^{N+1}(t)}\right) - \mu\,\rho\left(\frac{1-\rho^N(t)}{1-\rho^{N+1}(t)}\right) \tag{17}$$

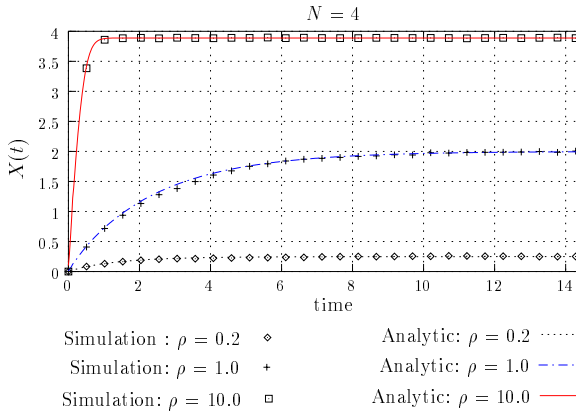
où $\rho(t)$ est solution de :

$$X(t) = f(\rho(t)) = \frac{N\rho^{N+2}(t) - (N+1)\rho^{N+1}(t) + \rho(t)}{1 - \rho(t) - \rho^{N+1}(t) + \rho^{N+2}(t)} \qquad (18)$$

L'intégration numérique est effectuée en résolvant $\rho(t) = f^{-1}(X(t))$ à chaque pas d'intégration (en utilisant par exemple la méthode de Newton). Cette approche est très efficace numériquement et se révèle assez précise comme on le verra plus loin. Evidemment, une autre possibilité, exacte elle, serait d'intégrer numériquement les équations de Chapman-Kolmogorov. Cependant, compte tenu des tailles importantes des tampons dans les routeurs actuels, il est clair que cette approche est beaucoup trop coûteuse numériquement pour étudier des réseaux de taille réelle.

La figure 27 compare l'évolution transitoire de X(t) obtenue avec le modèle analytique avec celle obtenue avec des simulations de Monte-Carlo (10 000 réalisations). Là encore, on peut constater que l'approximation différentielle est relativement précise : pour ρ = 0.2, 1.0 et 10.0, les erreurs moyennes sont respectivement de 2.2%, 1.5% et 0.1% tandis que les erreurs maximales sont de 8.1%, 6.3% et 2.0%.

Figure 27: $M/M/1/N$ **: comparaison entre le modèle analytique et des simulations**



II.1.2.3 Modèle dynamique de la file $M^k/M^k/1/\infty$

Le trafic d'entrée dans un noeud du réseau est en général la superposition de différents flots de trafic. Chacun de ces flots a son propre taux d'arrivée de paquets et éventuellement son propre routage. De plus, il se peut que les besoins en ressource des différents flots soient très différents. Le modèle $M^k/M^k/1/\infty$ a été introduit pour représenter de telles situations.

Il s'agit d'un modèle avec un serveur et une file d'attente de capacité infinie. Le trafic d'entrée dans ce système est la superposition de K flots différents k=1...K. Les clients du flot k arrivent suivant un processus de Poisson de taux λ_k. Ils sont servis avec ceux des autres flots suivant la discipline PAPS. La durée de service des clients du flot k est une variable aléatoire exponentiellement distribuée de moyenne $1/\mu_k$.

Soit $\rho_k=\lambda_k/\mu_k$. Le taux d'utilisation global du serveur est $\rho = \sum_k \rho_k$. Si $\rho < 1$, il existe une distribution de probabilités stationnaires dont l'expression a été déterminée par Marie et Rubino [MAR86]. En particulier, le nombre moyen de clients de la classe k présents dans le système à l'équilibre est donné par :

$$X_k(\infty) = \frac{\rho_k}{1-\rho}\left[1 + \sum_{i=1}^{K}\left(\frac{\mu_k}{\mu_i} - 1\right)\rho_i\right] \qquad k = 1\ldots K \tag{19}$$

L'équation différentielle associée au trafic moyen $X_k(t)$ de la classe k est :

$$\frac{d}{dt}X_k(t) = \lambda_k - \mu_k Q_k(t) \qquad k = 1\ldots K \tag{20}$$

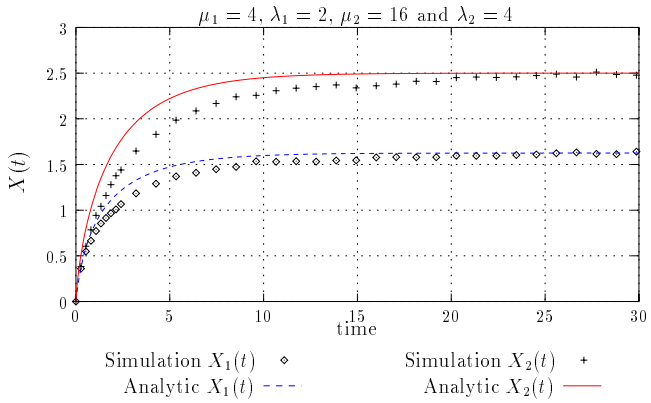
où $Q_k(t)$ est la probabilité qu'un client de classe k soit en service à l'instant t. A l'équilibre, cette probabilité est donnée par :

$$Q_k(\infty) = \rho_k = \frac{X_k(\infty)}{1 + \sum_{i=1}^{K}\frac{\mu_k}{\mu_i}X_i(\infty)} \qquad k = 1\ldots K \tag{21}$$

En prolongeant la validité de cette dernière relation au régime transitoire, on obtient l'approximation différentielle suivante de $X_k(t)$:

La figure 28 présente une comparaison entre cette approximation différentielle et 10 000 simulations de Monte-Carlo. Dans cet exemple, il y a K=2 classes de clients avec les caractéristiques suivantes : μ_1=4, λ_1=2, μ_2=16 et λ_2=4. On peut vérifier que le modèle différentiel fournit une assez bonne approximation de la réponse transitoire du système (pour la classe 2, l'erreur moyenne est de 7% alors que l'erreur maximale est de 24.3%). Là encore, il est important de noter que cette approximation est très peu coûteuse en temps calcul.

Figure 28: $M^k/M^k/1/\infty$ **: comparaison entre le modèle analytique et des simulations**



II.1.3 Modèles dynamiques de systèmes élémentaires non markoviens

Dans ce paragraphe, nous montrons que la théorie différentielle du trafic s'applique également à des systèmes n'ayant pas la propriété markovienne. On étudie en particulier les files d'attente $M/D/1/N$ et $M/G/1/\infty$.

II.1.3.1 Modèle dynamique de la file $M/D/1/N$

Pour l'évaluation des performances des architectures de télécommunications modernes, on utilise souvent des modèles de files d'attente à durée de service déterministe et à capacité finie. Ainsi, par exemple, de tels modèles sont utilisés dans l'étude des multiplexeurs ATM. En fonction du

problème étudié, différents modèles ont été considérés ([GRA90][ROB96][VIC96]) : $M/D/1/N$, $Geo^K/D/1/N$, $K*D/D/1/N$ et $\sum D_i/D/1/N$. La file d'attente $M/D/1/N$ représente de loin le plus simple et le plus général de ces modèles [KLE75].

La file d'attente $M/D/1/N$ est un modèle à capacité finie, avec N-1 positions d'attente. Les clients arrivent suivant un processus de Poisson de taux λ. Ils sont servis suivant la discipline PAPS et la durée de service est une constante T. Comme pour le modèle $M/M/1/N$, aucune hypothèse sur le taux d'utilisation $\rho = \lambda T$ n'est nécessaire.

Bien que ce modèle soit étudié depuis très longtemps et qu'il soit bien résolu d'un point de vue algorithmique, il n'existait pas jusqu'à récemment de résultats analytiques pour cette file d'attente (en dehors du cas particulier N=1, connu sous le nom de modèle d'Erlang B). Dans [BRU00b], nous déterminons la solution analytique du modèle $M/M/1/N$. En particulier, on prouve le résultat suivant :

$$P_0 = \frac{1}{1+\rho b_{N-1}} \tag{22}$$

$$P_j = \frac{b_j - b_{j-1}}{1+\rho b_{N-1}} \qquad j = 1 \ldots N-1 \tag{23}$$

$$P_N = 1 - \frac{b_{N-1}}{1+\rho b_{N-1}} \tag{24}$$

où les coefficients b_n sont donnés par b_0=1 et

$$b_n = \sum_{k=0}^{n} \frac{(-1)^k}{k!}(n-k)^k e^{(n-k)\rho}\rho^k \qquad \forall n \geq 1 \tag{25}$$

Dans [BRU00b] est également proposé un algorithme efficace pour le calcul des coefficients b_n. En utilisant cette expression des probabilités stationnaires, on détermine aisément X, le nombre moyen de clients dans le système à l'équilibre :

$$\bar{X} = N - \frac{\sum_{k=0}^{N-1} b_k}{1+\rho b_{N-1}} \tag{26}$$

Les résultats précédents donnent la solution de la file $M/M/1/N$ en régime stationnaire. Dans [GAR00], nous développons une procédure numérique simple pour le calcul exact de la distribution de probabilités transitoire d'une file $M/M/1/N$. On montre que l'équation différentielle du trafic moyen transitoire $X(t)$ est :

$$\frac{d}{dt}X(t) = \lambda[1 - P_N(t)] - d(t) \tag{27}$$

Là encore, cette dernière équation exprime que la variation du nombre moyen de clients durant un petit intervalle de temps Δt est donnée par la différence entre le nombre de clients qui entrent effectivement dans le système $\lambda[1 - P_N(t)]\Delta t$ et le nombre de clients qui en sortent $d(t)\Delta t$. Ainsi, $d(t)$ est le taux instantané de départ à l'instant t et son expression est donnée par l'équation suivante :

$$\forall \tau < T \qquad d(rT+\tau) = \lambda P_0(\tau) - \sum_{n=1}^{r-1} \frac{dP_0(nT+\tau)}{d\tau} \tag{28}$$

Dans [GAR00], une approche numérique permettant le calcul récursif de $d(t)$ est proposée. Les résultats analytiques et les algorithmes proposés dans cet article fournissent la solution transitoire

exacte de la file $M/M/1/N$. Cependant, en pratique, l'approximation différentielle suivante s'avère suffisante (et nettement moins coûteuse en temps calcul) :

$$\frac{d}{dt}X(t) \approx \lambda[1 - P_N(t)] - \frac{1}{T}[1 - P_0(t)] \tag{29}$$

$$\approx \frac{b_{N-1}(\rho(t))}{1 + \rho b_{N-1}(\rho(t))}\left[\lambda - \frac{\rho(t)}{T}\right] \tag{30}$$

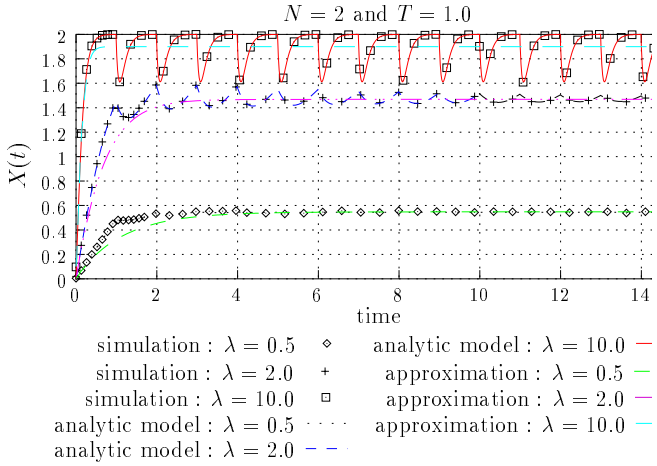
où $\rho(t)$ est solution de :

$$X(t) = f(\rho(t)) = N - \frac{\sum_{k=0}^{N-1} b_k(\rho(t))}{1 + \rho(t) b_{N-1}(\rho(t))} \tag{31}$$

Comme pour la file $M/M/1/N$, l'intégration numérique s'effectue en résolvant à chaque pas de temps $\rho(t) = f^{-1}(X(t))$.

La figure 29 présente les résultats obtenus avec le modèle analytique exact, avec l'approximation différentielle et avec 10 000 simulations de Monte-Carlo. On y a tracé les courbes obtenues pour $X(t)$ pour différentes valeurs de λ. Dans cet exemple, on a N=2 et T=1.0. On peut noter que, pour les grandes valeurs de ρ, le trafic moyen est sujet à des oscillations ([GAR00]), qui ne sont pas obtenues avec le modèle différentiel. Néanmoins, il est possible de vérifier que l'approximation différentielle donne bien l'évolution de la moyenne temporelle du nombre de clients.

Figure 29: **M/D/1/N : comparaison du modèle différentiel avec le modèle analytique exact et des simulations**



II.1.3.2 Modèle dynamique de la file $M/G/1/\infty$

Nous nous intéressons maintenant à la file de capacité infinie $M/G/1/\infty$ [COH69]. Soit $B(x)$ la distribution (arbitraire) du temps de service. Notons $\overline{x^k}$ le $K^{\text{ième}}$ moment de cette distribution. En particulier, on notera $T=\overline{x}$ la moyenne de la durée de service. Dans la suite, on supposera que le taux d'utilisation du serveur $\rho = \lambda T < 1$. Enfin, on notera $C_b^2 = (\overline{X^2}-T^2)/T^2$ le carré du coefficient de variation de la durée du service.

On peut proposer l'approximation suivante pour l'équation différentielle régissant le trafic moyen transitoire $X(t)$:

$$\frac{d}{dt}X(t) = \lambda - \frac{1}{T}[1 - P_0(t)] \tag{32}$$

Comme d'habitude, cette équation fait intervenir la probabilité transitoire $P_0(t)$ qui n'a pas d'expression analytique connue. Cependant, on peut observer qu'à l'équilibre, on a :

$$P_0(\infty) = 1 - \rho \qquad \text{et} \qquad X(\infty) = \frac{\rho + \lambda^2 x^2}{2(1-\rho)} \tag{33}$$

Ainsi, on obtient

$$(C_b{}^2 - 1)\rho^2 + 2[1 + X(\infty)]\rho - 2X(\infty) = 0 \tag{34}$$

L'approximation proposée pour $P_0(t)$ est alors (en supposant que $C_b{}^2 \neq 1$) :

$$P_0(t) \approx 1 - \rho(t) = 1 - \frac{-2(1+X(t)) + \sqrt{4(1+X(t))^2 + 8X(t)(C_b{}^2 - 1)}}{2(C_b{}^2 - 1)} \tag{35}$$

On obtient ainsi l'approximation différentielle suivante du trafic moyen $X(t)$:

$$\frac{d}{dt}X(t) \approx \lambda - \frac{1}{T}\rho(t) \tag{36}$$

où,

$$\rho(t) = \frac{-2(1+X(t)) + \sqrt{4(1+X(t))^2 + 8X(t)(C_b{}^2 - 1)}}{2(C_b{}^2 - 1)} \tag{37}$$

Prenons comme premier exemple celui d'une file $M/M/1/\infty$. On sait que le coefficient de variation de la distribution exponentielle est 1. Par conséquent, l'équation (34) donne :

$$\rho(t) = \frac{X(t)}{1+X(t)} \Rightarrow \frac{d}{dt}X(t) \approx \lambda - \frac{1}{T}\frac{X(t)}{1+X(t)} \tag{38}$$

qui est identique à l'équation (11) et justifie, dans ce cas, les approximations précédentes. Dans la suite, nous proposons différentes comparaisons entre le modèle différentiel et des simulations de Monte-Carlo (10 000 trajectoires) pour des distributions de service déterministe et Gamma.

- Distribution déterministe. On considère ici une file $M/D/1/\infty$. Dans ce cas, on a clairement $C_b{}^2 = 0$ et donc (avec l'équation (37)) :

$$\rho(t) = 1 + X(t) - \sqrt{1 + X(t)^2} \Rightarrow \frac{d}{dt}X(t) \approx \lambda - \frac{1}{T}[1 + X(t) - \sqrt{1 + X(t)^2}] \tag{39}$$

On a comparé cette approximation différentielle avec 10 000 simulations de Monte-Carlo. Les résultats sont représentés sur la figure 30.

- Distribution Gamma. Prenons pour second exemple celui de distributions de service Gamma. Rappelons que la distribution Gamma a une densité de probabilité $f_{\alpha,\beta}(x)$ donnée par :

$$f_{\alpha,\beta}(x) = \frac{\beta^\alpha x^{\alpha-1}}{\Gamma(\alpha)} e^{-\beta x} \qquad x \geq 0 \tag{40}$$

La moyenne T et la variance v de cette distribution sont données par $T=\alpha/\beta$ et $v=1/\beta$. Les valeurs utilisées pour les paramètres α et β ainsi que les courbes correspondantes pour la densité $f_{\alpha,\beta}(x)$ sont précisées dans la figure 43 (page 83). On peut noter que la dernière distribution est survariante ($C_b>1$).

La figure 31 compare les réponses transitoires obtenues avec le modèle différentiel et avec des

simulations événementielles.

On peut remarquer que l'approximation différentielle est assez grossière quand C_b>1. Dans ce cas, la convergence vers l'état stationnaire est beaucoup plus lente.

***Figure 30:* $M/D/1/\infty$: comparaison entre le modèle différentiel et des simulations**

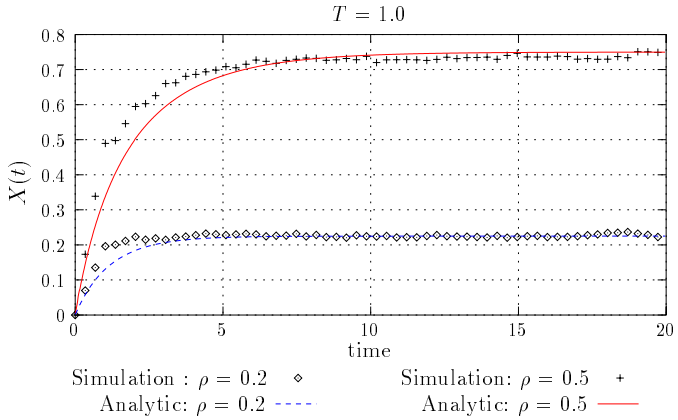
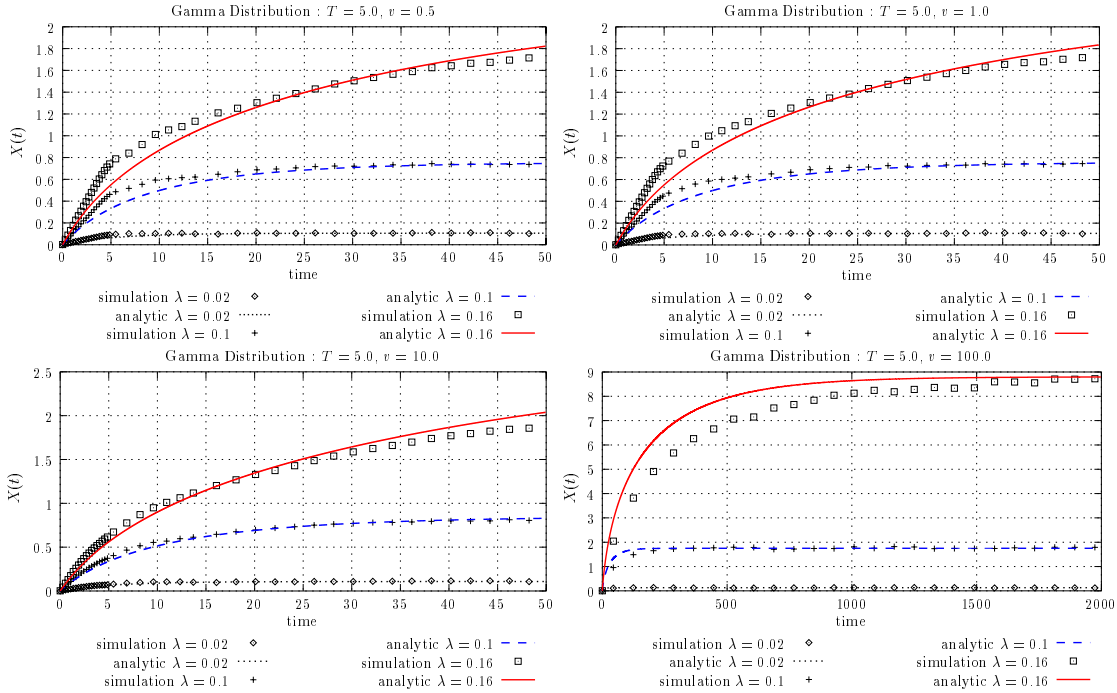


***Figure 31:* $M/G/1/\infty$: réponses transitoires pour des distributions de service Gamma de moyenne T=5.0 et de variance v=0.5, 1.0, 10.0 et 100.0**



II.1.4 Modèles dynamiques de réseaux de files d'attente

Une hypothèse classique pour de grands réseaux est de supposer des arrivées suivant le processus de Poisson et des durées de service exponentiellement distribuées en chaque noeud. Les files d'attente peuvent être à capacités finies ou infinies.

Sous ces hypothèses, considérons un réseau de *n* noeuds. Chaque noeud reçoit du trafic direct (exogène) et du trafic sortant des autres noeuds. Notons :

- λ_i le taux d'arrivée du trafic direct au noeud *i*. On suppose que les paquets arrivent suivant un processus de Poisson,

- μ_i le taux de service au noeud *i*. On suppose que les durées de service sont des variables aléatoires exponentiellement distribuées,
- $r_{i,j}$ le paramètre de routage donnant la proportion de trafic sortant du noeud *i* qui est envoyée vers le noeud *j*.

II.1.5 La forme produit

Soit a_j le taux d'arrivée au noeud *j* (trafic direct et trafic provenant des autres noeuds) :

$$a_j = \lambda_j + \sum_{i=1}^{n} a_i r_{i,j} \tag{41}$$

L'état du système est décrit par le nombre k_i de clients au noeud *i*. La distribution stationnaire Π associée à cet espace d'état est une distribution du type « forme produit » [KLE75]. En d'autres termes, la probabilité d'un état global du réseau est le produit des probabilités des états des différents noeuds pris isolément :

$$\Pi(k_1, k_2, \ldots, k_n) = \prod_{i=1}^{n} (1-\rho_i)\rho_i^k \qquad \text{où} \qquad \rho_i = \frac{a_i}{\mu_i} \tag{42}$$

Ce résultat est très intéressant dans la mesure où il fournit une expression analytique de la distribution à l'équilibre de l'état global du réseau. Une conséquence importante de la forme produit est que le comportement de chaque noeud du réseau est similaire à celui d'une file $M/M/1/\infty$. En effet, même si le trafic sortant en chaque noeud ne suit plus la loi de Poisson, la forme produit de la solution reste valide. Par conséquent, le trafic moyen stationnaire $X_i(\infty)$ au noeud *i* a l'expression suivante :

$$X_i(\infty) = \frac{\rho_i}{1-\rho_i} \qquad i = 1 \ldots n \tag{43}$$

II.1.6 Modèle différentiel d'un réseau

Soit $X_i(t)$ le nombre moyen de paquets au noeud *i* à l'instant *t*. Puisque chaque noeud se comporte comme une file $M/M/1/\infty$ isolée, le modèle différentiel développé au paragraphe II.1.2.1 peut être appliqué. Evidemment, il est nécessaire d'introduire le couplage entre les différents noeuds.

$$\frac{d}{dt}X_i(t) = \lambda_i + \sum_{j=1}^{n} r_{j,i}\mu_j[1 - P_j^0(t)] - \mu_i(1 - r_{i,i})[1 - P_i^0(t)] \tag{44}$$

où $P_i^0(t)$ est la probabilité que le noeud *i* soit vide à l'instant *t*. Ces probabilités peuvent être approximées de la même façon qu'au paragraphe II.1.2.1. On obtient alors l'approximation différentielle du trafic transitoire $X_i(t)$ au noeud *i* :

$$\frac{d}{dt}X_i(t) \approx \lambda_i + \sum_{j=1}^{n} r_{j,i}\mu_j \frac{X_j(t)}{1+X_j(t)} - \mu_i(1 - r_{i,i})\frac{X_i(t)}{1+X_i(t)} \tag{45}$$

Ce modèle différentiel est dynamique : à chaque instant *t*, on obtient l'évolution de l'espérance $X_i(t)$ du nombre de clients en chaque noeud. Si les taux d'arrivées et les paramètres de routage sont stationnaires, alors le système d'équations différentielles couplées converge vers l'état moyen du

système à l'équilibre.

Deux approches sont possibles pour résoudre ce système d'équations différentielles du premier ordre :

- via des techniques classiques d'intégration numérique si l'on s'intéresse à la solution transitoire. En général, la méthode d'Euler est suffisamment précise,
- via un algorithme d'approximation successive si l'on s'intéresse uniquement à la solution stationnaire.

Cette modélisation dynamique du réseau peut être étendue à des réseaux dont le routage dépend des flots en utilisant des files $M^k/M/1/\infty$ et à des réseaux multiclasses en utilisant des files $M^k/M^k/1/\infty$ [BRU00a].

II.1.7 Réseau test (réseau double X)

Considérons le réseau décrit sur la figure 32a. Ce réseau a 6 noeuds $M/M/1/\infty$ et 8 flots. Les trafics directs entrant aux noeuds 2 et 3 et destinés aux noeuds 4 et 5 n'ont qu'une seule route : le lien direct. Mais les trafics entrant aux noeuds 0 et 1 et destinés aux 4 et 5 peuvent être routés suivant deux chemins différents : 50% du trafic est envoyé sur chacun de ces chemins (routage par partage de charge).

Les valeurs des paramètres λ_{ij} et μ_i sont précisées dans les tableau 3 et 4.

Tableau 3: **Valeurs des taux d'arrivées λ_{ij}**

λ_{04}	λ_{05}	λ_{14}	λ_{15}	λ_{24}	λ_{25}	λ_{34}	λ_{35}
2	3	3	2	2	1	1	2

Tableau 4: **Valeurs des taux de service μ_i**

μ_4	μ_1	μ_2	μ_3	μ_4	μ_5
10	10	15	15	10	10

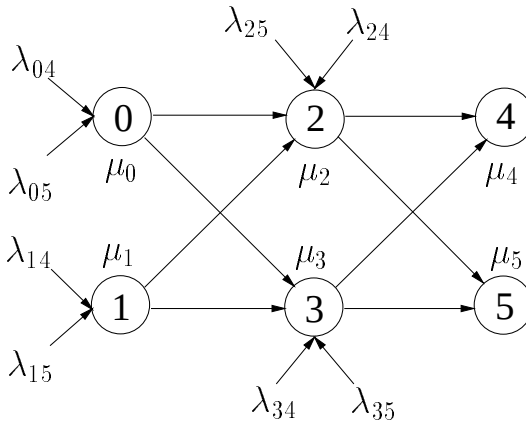
Dans le tableau 5, on compare les valeurs exactes des nombres moyens de clients avec celles obtenues par intégration du modèle différentiel et avec celles obtenues par des simulations de Monte-Carlo. Les résultats montrent la très bonne estimation obtenue avec le modèle différentiel. Les simulations événementielles prennent bien sûr un temps de calcul considérable et sont assez peu précises. De plus, comme on peut le remarquer sur la figure 32b, la réponse transitoire du modèle différentiel fournit une assez bonne approximation de la dynamique du système.

Tableau 5: **Nombre moyens de clients à l'équilibre dans chaque noeud**

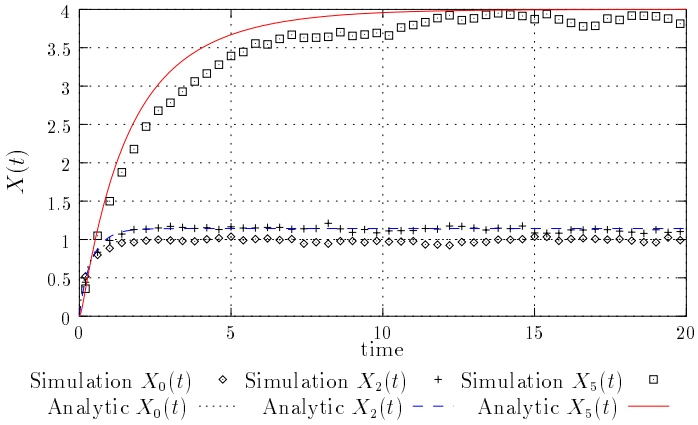
	Exact	Modèle différentiel	Simulations
$X_0(\infty)$	1,000	1,000	0,999

	Exact	Modèle différentiel	Simulations
$X_1(\infty)$	1,000	1,000	1,014
$X_2(\infty)$	1,142	1,142	1,168
$X_3(\infty)$	1,142	1,142	1,193
$X_4(\infty)$	4,000	4,000	4,038
$X_5(\infty)$	4,000	3,999	4,104

Figure 32: **(a) Le réseau double X. (b) Trafics moyens transitoires des noeuds 0, 2 et 5**



(a)



(b)

II.2 Ordonnancement équitable

II.2.1 Paradigme GPS (*Generalized Processor Sharing*)

GPS (*Generalized Processor Sharing*) est un ordonnancement équitable idéal non utilisable dans la réalité, mais qui sert de référence pour les algorithmes WFQ (*Weighted Fair Queueing*) (I.5.2.4).

Le paradigme GPS considère les flux de façon fluides, c'est à dire comme étant composés d'une infinité de petits éléments ne possédant pas de granularité. Les flux sont servis en parallèle. A chaque flux *i* est associé un poids ϕ_i. Le débit de la ligne *C* est alors distribué proportionnellement au poids de chaque flux actif.

Ainsi l'objectif de ce modèle est de garantir une bande passante minimum, c'est à dire un débit minimum à chaque flux.

II.2.1.1 Définition de GPS

La définition de GPS a été vue au paragraphe I.5.2.4. Plus précisément, soit $B(\tau_1,\tau_2)$ l'ensemble des sessions actives durant l'intervalle temporel $[\tau_1,\tau_2]$ durant lequel aucune session ne change

d'état. Dans cet intervalle de temps, la quantité de travail totale allouée aux sessions par le serveur de débit *C* est égal à la somme des quantités de travail que reçoit chaque session :

$$\sum_{i \in B(\tau_1, \tau_2)} W_i(\tau_1, \tau_2) = C(\tau_2 - \tau_1) \qquad (46)$$

La quantité de travail reçue par la session *i* est donc :

$$W_i(\tau_1, \tau_2) = \frac{\phi_i}{\sum_{j \in B(\tau_1, \tau_2)} \phi_j} C(\tau_2 - \tau_1) \qquad (47)$$

Pour obtenir le débit instantané d'une session *i*, on pose $\tau_1 = t$ *et* $\tau_2 = t + \delta t$ et on fait tendre δt vers zero. On obtient alors le débit instantané $d_i(t)$ pour chaque session *i* :

$$d_i(t) = \frac{\phi_i}{\sum_{j \in B(\tau_1, \tau_2)} \phi_j} C \qquad (48)$$

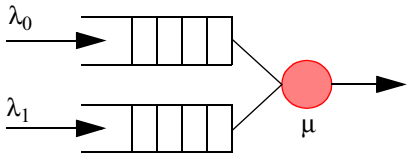
II.2.2 Etat de l'art

Dans la mesure où les algorithmes d'ordonnancement basés sur GPS sont devenus incontournables pour différencier les flux dans les réseaux à qualité de service, l'évaluation des performances des systèmes GPS est capitale. Malheureusement, l'analyse des files d'attente à service de type GPS est connue pour être très complexe [ADA01]. Cependant, il existe une analyse exacte du cas à deux files utilisant des méthodes issues de la théorie des fonctions complexes (la méthode de la valeur limite et la méthode d'uniformisation). Konheim, Meilijson et Melkman ont obtenu, avec une méthode d'uniformisation, la distribution jointe du nombre de clients [KON81] dans le cas complètement symétrique ($\phi_1 = \phi_2$). Ils ont considéré un processus d'arrivé Poissonien et une durée exponentielle de service. Le modèle des processeurs couplés (*coupled processor model*) est très proche de celui de GPS. En analysant ce modèle, Fayolle et Iasnogorodski [GAR99] ont montré dans le cas asymétrique que la résolution de l'équation fonctionnelle associée à la fonction génératrice des moments de la distribution jointe du nombre de clients peut être transformée en un problème aux valeurs limites de Riemann-Hilbert. Cohen et Boxma [COH83] ont poursuivi ces travaux et sont parvenus à une analyse exacte pour une distribution de service générale. Cohen a également obtenu des résultats partiels dans le cas de trois files [COH84][COH85].

II.2.3 Approximation analytique de GPS par linéarisation pour 2 files

Nous présentons dans ce paragraphe des résultats récents sur l'approximation d'un système GPS à deux files d'attente. L'étude du système à *N* files est présenté dans [BOC03]. La figure 33 présente deux files d'attente dont le service est équitablement réparti par le paradigme GPS. Le poids ϕ_0 est associé à la file 0 et le poids ϕ_1 est associé à la file 1 (on impose $\phi_0 + \phi_1 = 1$).

Figure 33: **Système à deux files d'attentes GPS**



En faisant tendre ϕ_k vers 1, la file k tend à se comporter comme une file $M/M/1$:

$$\lim_{\phi_k \to 1} X_k = \frac{\rho_k}{1-\rho_k} \qquad k = 0, 1 \tag{49}$$

En considérant que

$$X = X_0 + X_1 = \frac{\rho_0 + \rho_1}{1-\rho} = \frac{\rho}{1-\rho} \tag{50}$$

on peut écrire que :

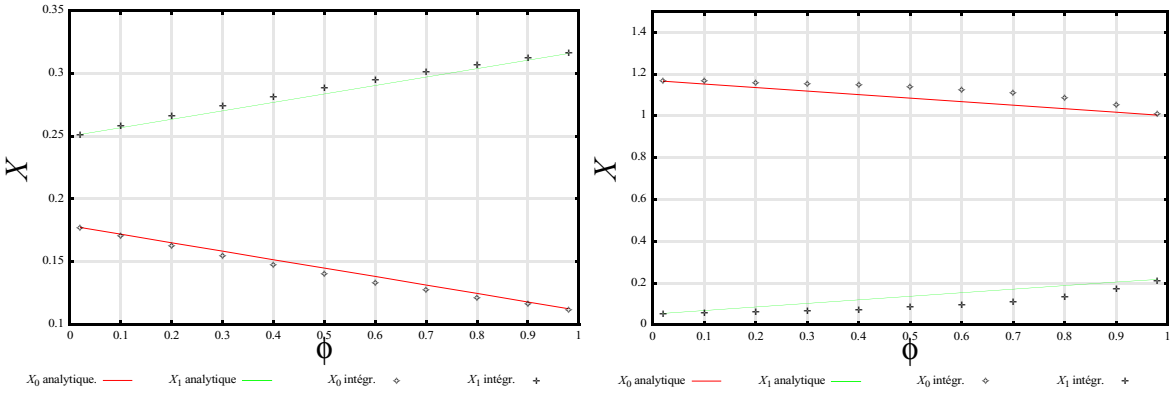
$$X_{1-k} = X - X_k \underset{\phi_k \to 1}{\sim} \frac{\rho}{1-\rho} - \frac{\rho_k}{1-\rho_k} \qquad k = 0, 1 \tag{51}$$

La figure 34 présente la simulation événementielle du système par intégration des chaînes de Markov. On constate sur cette figure que la charge de chaque file est une fonction quasiment linéaire de son poids ϕ. En connaissant les valeurs de X_k lorsque ϕ_k prend les valeurs extrêmes 0 et 1, (équations (49) et (51)) nous écrivons la relation linéaire suivante :

$$X_k \approx \left(\frac{\rho_k}{1-\rho_k} + \frac{\rho_{1-k}}{1-\rho_{1-k}} - \frac{\rho}{1-\rho} \right) \cdot \phi_k - \frac{\rho_{1-k}}{1-\rho_{1-k}} + \frac{\rho}{1-\rho} \tag{52}$$

Cette équation est également tracée sur la figure 34 et montre la validité de l'approximation pour de faibles charges.

Figure 34: **Intégration de la chaîne de Markov et Simulation événementielle**



λ_0=0.1 et λ_1=0.2 μ=1 λ_0=0.5 et λ_1=0.05

Le principe d'obtention du régime transitoire analytique de ce système à ordonnancement GPS est également décrit dans [BOC03].

II.3 Conclusion

Dans ce chapitre, nous avons présenté une méthodologie de modélisation des systèmes avec attente basée sur la théorie du trafic différentiel. Chaque cas étudié est accompagné de l'expression

exacte de l'équation différentielle régissant le nombre moyen de clients dans le système. Cette équation exprime toujours une différence entre un débit instantané d'entrée et un débit instantané de sortie. Elle fait en général intervenir des probabilités transitoires du modèle markovien dont l'expression analytique est inconnue. La méthode proposée consiste alors à faire une approximation de ces probabilités en utilisant une relation implicite entre le trafic moyen et les probabilités d'états. En général, les approximations obtenues sont à la fois assez précises et très rapides à calculer.

Nous avons illustré cette méthode en étudiant plusieurs systèmes markoviens élémentaires : files d'attente $M/M/1/\infty$, $M/M/1/\infty$ multi-flots et multiclasses, ainsi que $M/M/1/N$. Ces files permettent de modéliser différents organes d'un réseau à commutation de paquets.

Nous avons ensuite présenté les files de type $M/D/1/N$ et $M/G/1/\infty$, qui sont en général utilisées pour modéliser les organes de multiplexage ou de routage dans un réseau. Le comportement dynamique et les équations différentielles exactes gouvernant le trafic moyen transitoire de ces files sont décrits ainsi que les approximations du trafic transitoire moyen. La théorie du trafic différentiel permet ainsi de modéliser des systèmes non Markoviens. A ce titre, d'autres modèles ont été développés : $G/M/1/\infty$, $M/G/1/\infty$ multi-flots et multiclasses, $M/G/1/\infty$ multi-file à combinaison d'ordonnancement GPS et priorité.

Nous avons ensuite présenté l'approche utilisée pour établir les équations différentielles approchées gouvernant les flux de traitement sur les ressources d'un réseau à commutation de paquets. Ce résultat est important à plus d'un titre. Tout d'abord parce qu'il fournit un outil d'analyse du comportement dynamique de ces réseaux. Ensuite, parce qu'il constitue un moyen extrêmement efficace de calculer le trafic stationnaire dans le réseau. Nous avons vu, sur le réseau double X, la qualité des résultats obtenus par rapport à des simulations.

Nous avons enfin présenté le paradigme GPS utilisé dans les réseaux à qualité de service. Ce paradigme permet une répartition du service équitable entre les différentes files d'un système. Ce problème est très complexe et l'obtention d'une solution analytique exacte parait difficile. Nous avons présenté une approximation analytique simple, qui se révèle assez précise pour de faibles charges. Cette approximation est généralisable à N classes et permet d'obtenir le régime transitoire de la charge induite par les flux.

Chapitre III - Sources de trafic

Afin de finement modéliser les trafics multimedia, des générateurs de temps d'inter-arrivées et de tailles de paquets sont employés. Les générateurs utilisés doivent représenter correctement les trafics de tout type que l'on rencontre dans les réseaux. De nos jours, le trafic généré par les ordinateurs connectés au réseau Internet est en croissance permanente, et de nouveaux services multimedia apparaissent sans cesse. Nous pouvons distinguer trois grandes classes de trafic : Les données, la vidéo et l'audio. Ces trafics ont par nature certaines caractéristiques intrinsèques et les réseaux les transportant ne doivent pas leur faire subir de transformations trop importantes.

Afin d'évaluer les performances de tels réseaux, il est nécessaire de simuler également les sources de trafics telles qu'elles existent dans les réseaux réels. Pour modéliser ces trafics dans un environnement de simulation, nous devons disposer de modèles de sources précis et variés. Les modèles de trafics que nous présentons ici sont issus d'un document réalisé dans le cadre du projet RNRT Esquimaux. Dans ce document, E. Thibault [THI01] présente un état de l'art sur la modélisation des trafics multimedia dans le réseau Internet. Ce travail est basé sur différentes études métrologiques et propose plusieurs modèles pour les trois grandes classes de trafic.

Nous commencerons par présenter les différents processus de génération utilisé dans la simulation événementielle. Pour cela, nous porterons une attention toute particulière sur la génération d'une variable aléatoire uniforme, qui est à la base de toutes nos générations aléatoires. Nous verrons ensuite de quelle manière représenter les arrivées et les départ d'appels pour modéliser la dynamique de charge du réseau. Les générateurs seront alors paramétrés pour représenter des codecs réels et normalisés utilisés dans les réseaux actuels. Enfin, dans une dernière partie, nous procéderons à l'agrégation de sources de trafic pour en observer leur comportement statistique dans le but d'en dégager des propriétés utiles pour la simulation hybride.

III.1 Processus de génération aléatoire

La génération de données aléatoires est incompatible avec les machines intrinsèquement déterministes que nous utilisons pour nos simulations. Cependant, il est possible, par transformation, de ramener le problème de génération aléatoire de distribution quelconque au problème de génération aléatoire uniforme.

III.1.1 Générateur aléatoire uniforme

Un générateur aléatoire uniforme produit des données uniformément réparties entre 0 et 1. Il y a plusieurs manières d'obtenir un générateur aléatoire uniforme, mais il est admis qu'il est quasiment impossible d'obtenir un générateur uniforme parfait avec un simple algorithme sur une machine déterministe. La méthode à la fois la plus répandue, la plus simple à implémenter et la plus rapide à l'exécution est la méthode LCG (*Linear Congruential Generator*). Cependant les paramètres de l'algorithme sont à choisir avec très grand soin pour obtenir un générateur de qualité satisfaisante.

Les générateurs de type *LCG(m, a, c)* sont de la forme suivante (53):

$$I_{j+1} = (aI_j + c) \mod m \qquad (53)$$

I_j est la variable générée, m est le module, a le multiplicateur et c l'incrément. Ce type de générateur est cyclique. La périodicité maximale du générateur est bornée par *m et* dépendra du choix des paramètres a, c, m, et I_0. Si la périodicité est de m, toutes les valeurs entre 0 et *m-1* sont générées, et n'importe laquelle de ces valeurs pourrait être utilisée pour l'initialisation (I_0). Il est préférable d'obtenir une période assez grande (donc m grand) pour éviter tout cycle dans les valeurs générées pendant le temps d'une simulation.

Outre les caractéristiques générales d'un générateur aléatoire uniforme, deux points importants apparaissent immédiatement concernant la simulation de Monté-Carlo:

- La période du générateur doit être assez grande pour qu'elle ne soit pas atteinte avant la fin de la simulation. Les données générées seraient sinon corrélées, puisque ce type de générateur est cyclique.
- Une simulation événementielle utilisant plusieurs sources de trafic nécessite plusieurs générateurs aléatoires (au moins un par source). Il est délicat d'utiliser un générateur par source en parallèle dans une même simulation, car les risques de corrélation dûs à la méthode d'initialisation des paramètres sont non négligeables. De même il est difficile d'utiliser un générateur différent pour chaque source, car nous ne pouvons pas en disposer en quantité suffisante.

La solution que nous adoptons est l'utilisation d'un générateur aléatoire uniforme unique pour l'ensemble des sources. Il conviendra alors, outre les tests généraux, de s'assurer que la période de ce générateur est assez grande. Il faudra aussi être capable de relancer le générateur à partir d'un état précédent ou d'un état sauvegardé, et connaître à l'avance des états initiaux suffisamment distants pour les simulation parallèles, afin d'éviter les corrélations.

Nous utiliserons plusieurs tests afin de vérifier qu'un générateur a les caractéristiques attendues pour la simulation événementielle. Ces tests sont :

- Test de période:
  Nous générons un grand nombre de valeurs aléatoires jusqu'à retomber sur une des valeurs générées. Ce test est délicat si m est très grand.
- Tests d'adéquation
  Ce test permet de vérifier si la distribution du générateur est bien uniforme.
  Les tests d'adéquation que nous utilisons sont:
  - Le test du χ^2
  - Le test de Kolmogorov - Smirnov
  - Le test de Cramer - Von Mises
- Tests d'indépendance
  Ce test permet de vérifier que les échantillons du générateur uniforme sont indépendants. Nous avons utilisé deux tests d'indépendance :
  - Le test des différences premières
  - Le test de Spearman.

Nous avons cependant précisé que nous ne voulions disposer que d'un seul générateur pour alimenter les différentes sources. Or chaque source a besoin d'un générateur uniforme. Le test doit donc être effectué en oubliant N-1 valeurs parmi N tirées, avec différentes valeurs de N afin de simuler plusieurs sources en parallèle tirant chacune leur tour une valeur aléatoire. Il peut être intéressant aussi de faire ce même test en oubliant un nombre aléatoire de tirage entre chaque tirage effectif.

III.1.2 Générateur aléatoire à partir d'une PDF

Nous sommes amenés lors de la génération de sources de trafic à produire des inter-arrivées distribuées selon une distribution (PDF : *Probability Distribution Function*) particulière. Une technique générale pour les obtenir consiste à utiliser l'inverse de la fonction de répartition (CDF : *Cumulative Distribution Function*) correspondante et le générateur aléatoire uniforme [KLE75].
Prenons l'exemple de la distribution exponentielle largement utilisée dans nos simulations. On a :

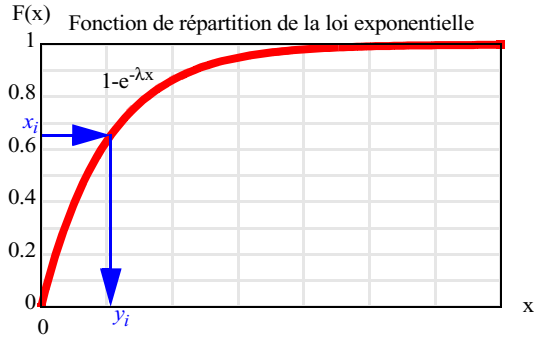
$$\text{PDF: } f(x) = \lambda e^{-\lambda x}$$

$$\text{CDF: } F(x) = 1 - e^{-\lambda x}$$

$$F^{-1}(x) = -\frac{\log(1-x)}{\lambda}$$

À partir d'une suite aléatoire uniforme $X=(x_i)$ ($0 < x_i < 1$ pour tout i), on génère la suite $Y=(y_i)$, avec $y_i=F^{-1}(x_i)$ (figure 35).

Figure 35: **Utilisation d'une CDF pour la génération d'une V.A.**



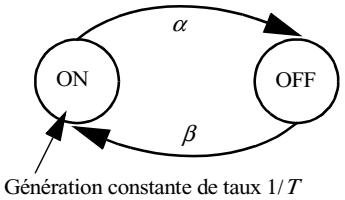
III.1.3 Processus ON-OFF

Le processus ON-OFF (figure 36) est principalement utilisé pour modéliser la voix. Une conversation est caractérisée par des intervalles de parole entrecoupés de silences. Les paramètres du processus sont les suivants :

- α le taux moyen de mort (durée moyenne de l'état ON: $T_{on}=1/\alpha$)
- β le taux moyen de naissance (durée moyenne de l'état OFF: $T_{off}=1/\beta$)
- T le temps constant inter-arrivée durant la période ON.

α et β caractérisent les durées exponentielles de parole et de silence, tandis que T correspond au temps de paquetisation constant pendant toute la durée de l'état ON.

Figure 36: **Processus ON-OFF**



Les études menées sur le trafic téléphonique et sur le trafic audio dans les réseaux à commutation de paquet ont permis de donner des valeurs aux paramètres α, β, et T [THI01]. Ces

valeurs dépendent grandement de la sensibilité des codecs (COdeurs/DÉCodeurs) à la détection des silences, et de la validité de l'approximation Markovienne des périodes ON et OFF. Le tableau 6 présente les valeurs de ces paramètres en fonction du type de conversation mesurées.

Tableau 6: **.Paramètres du modèle ON-OFF**

Type	T_{on}(s)	T_{off}(s)
Modèle classique	0.352	0.650
Conversation	7.24	5.69
Lecture	3.23	0.41

On peut calculer la valeur moyenne de $\overline{\lambda}$ du débit en paquets par seconde sur l'ensemble des deux états de la manière suivante :

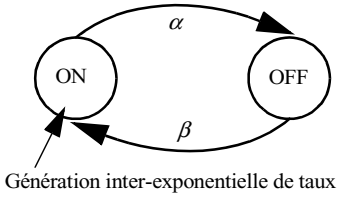
$$\overline{\lambda} = \frac{P_{on}}{T} = \frac{T_{on}}{(T_{on} + T_{off}) \cdot T} = \frac{\beta}{(\beta + \alpha) \cdot T} \qquad (54)$$

III.1.4 Processus IPP

Le processus IPP (*Interrupted Poisson Process* : processus de Poisson interrompu, figure 37) permet également de modéliser la voix. La différence avec le modèle ON-OFF du paragraphe précédent est que les inter-arrivées sont exponentielles pendant la période ON. Il est caractérisé par trois paramètres:

- α le taux moyen de mort (durée moyenne de l'état ON: $T_{on}=1/\alpha$)
- β le taux moyen de naissance (durée moyenne de l'état OFF: $T_{off}=1/\beta$)
- λ le taux d'arrivée durant la période ON.

Figure 37: **Processus IPP**



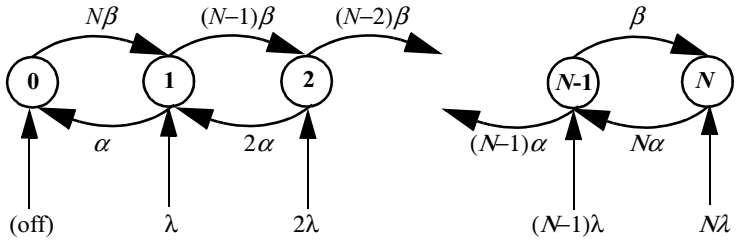
Bien qu'il existe plusieurs méthodes d'approximation de processus ON-OFF par un processus IPP, on peut prendre les mêmes valeurs pour α et β, et $1/T$ pour λ.

L'intérêt du modèle IPP par rapport au modèle ON-OFF réside dans le fait que l'on sait calculer le modèle exact de l'agrégation de plusieurs modèles IPP (voir Processus MMPP-N+1).

III.1.5 Processus MMPP-N+1

Le processus MMPP-N+1 (*Markov Modulated Poisson Process* : processus de Poisson modulé par un processus de Markov à N+1 états, figure 38), est utilisé pour représenter la superposition de N (constant) sources homogènes et identiques de type IPP.

Figure 38: **Processus MMPP-N+1**



Les paramètres α, β et λ ont la même signification que dans le paragraphe précédent (processus IPP). La probabilité π_i d'être dans l'état *i* et les caractéristiques moyennes du processus sont données par :

$$\pi_i = P(\bar{\lambda} = i\lambda) = \frac{N!}{i!(N-i)!} P_{on}^{i} (1 - P_{on})^{N-i}$$

$$E(\bar{\lambda}) = N\lambda P_{on}$$

$$Var(\bar{\lambda}) = N\lambda^2 P_{on}(1 - P_{on})$$

avec:

$$P_{on} = \frac{\beta}{\beta + \alpha}$$

III.1.6 Approximation d'un processus MMPP-N+1 par un processus MMPP-2

On peut utiliser un processus MMPP-2 pour approcher un processus MMPP-N+1 (figure 39). L'état UL (*UnderLoad*) du processus MMPP-2 est associé à l'ensemble des états 0 à M du processus MMPP-N+1, tandis que l'état OL (*OverLoad*) est associé aux états M+1 à N.

Plusieurs méthodes existent pour déterminer M et les paramètres du processus MMPP-2. La validité des méthodes est vérifiée en examinant les caractéristiques du processus agrégé dans un multiplexeur ou un routeur (taux de perte, temps de service) [BAI91][LEN98].

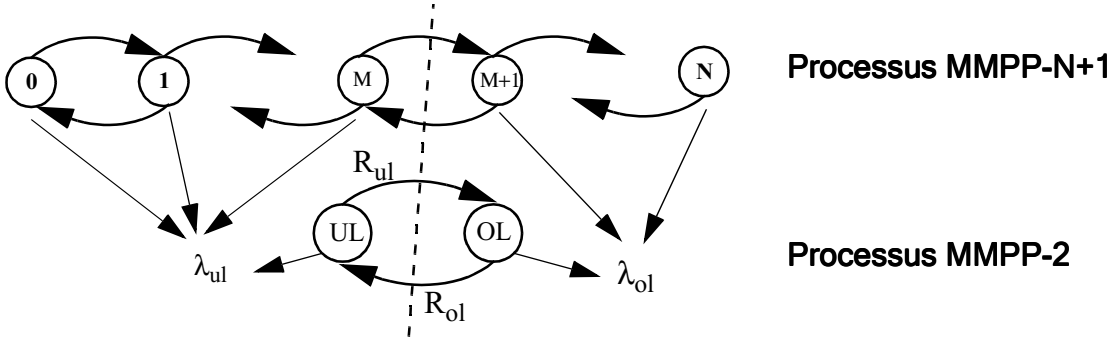
Connaissant M, un calcul possible des paramètres du processus MMPP-2 est donné par :

$$\lambda_{ul} = \lambda \sum_{i=0}^{M} i \frac{\pi_i}{\pi_{ul}} \qquad \lambda_{ol} = \lambda \sum_{i=M+1}^{N} i \frac{\pi_i}{\pi_{ol}}$$

avec:

$$\pi_{ul} = \sum_{i=0}^{M} \pi_i \qquad \pi_{ol} = \sum_{i=M+1}^{N} \pi_i$$

Figure 39: **Approximation d'un processus MMPP-N+1 par un processus MMPP-2**



III.1.7 Processus corrélé basé sur le modèle $M/G/\infty$

Pour représenter la corrélation présente dans la distribution des temps d'inter-arrivées de certaines sources de trafic vidéo, on peut utiliser le processus d'occupation du système $M/G/\infty$ [KRU98]. Les arrivées dans le système sont distribuées exponentiellement (de paramètre λ) et le temps de service est distribué selon une loi G de paramètre σ.

Le processus d'occupation noté (X_n) représente le nombre de clients dans le système au début de l'intervalle temporel (*slot*) $[nT, (n+1)T[$, où T est l'échelle de temps représentant le temps de génération d'une image (1/25 secondes) ou d'un GOP (Groupe d'Image - *Group Of Images*).

Le processus (X_n), stationnaire et corrélé représente bien les flux vidéos. Il modélise la taille des images (ou des GOP) du trafic source. La propriété qui nous intéresse ici est la corrélation. Elle peut être à courte ou à longue dépendances (SRD/LRD: *Short / Long Range Dependency*).

Pour un slot n donné, la distribution de la variable aléatoire X_n est exponentielle de paramètre $\lambda E(\sigma)$. Il convient de la transformer en une autre variable Y_n issu d'une autre loi (à déterminer) représentant correctement la taille des images (GOP) de la source réelle. La corrélation temporelle est conservée par la transformation du processus (X_n) en (Y_n) .

La transformation est réalisée en deux temps : Tout d'abord la variable X_n distribuée exponentiellement est transformée en une variable aléatoire uniforme par application de la fonction de répartition de la distribution exponentielle, puis retransformée afin d'obtenir la distribution voulue en appliquant l'inverse de la fonction de répartition correspondante.

III.1.7.1 La corrélation

Les codecs vidéo normalisés utilisent les redondances spatiales (*motion compensation* : fortes corrélations entre deux points voisins sur une même image) et / ou temporelles (forte corrélation entre deux images successives) pour compresser les images. Quelques codecs associés sont succinctement présentés dans le tableau 9 page 85.

Les dépendances temporelles sont définies de la manière suivante :

Définition 2: Soit $(X_n)_{n \geq 0}$ un processus discret stationnaire représentant les tailles de parties d'un document. La fonction d'auto corrélation associée est définie par :

$$\rho(k) = \frac{Cov(X_n, X_{n+k})}{Var(X_n)} \quad (55)$$

où $\rho(k)$ représente la dépendance qui peut exister entre X_n et X_{n+k}, pour n et k positifs ou nuls.

Définition 3: Un processus $(X_n)_{n \geq 0}$ stationnaire à temps discret est à mémoire longue (LRD) si il vérifie la relation suivante:

$$\sum_k \rho(k) = \infty \quad (56)$$

Définition 4: Un processus $(X_n)_{n \geq 0}$ stationnaire à temps discret est à mémoire courte (SRD) si il vérifie la relation suivante:

$$\sum_k \rho(k) < \infty \quad (57)$$

Concernant les sources vidéo, il a été mis en évidence trois types de corrélations différentes, (58) selon des associations film vidéo / codecs [KRU98]. Ces corrélations ne sont pas associées à un type de film particulier. On les associe cependant à un codec particulier dans nos modèles de source.

$$\begin{aligned} \rho(k) &= e^{-bk} && \text{(SRD)} \\ \rho(k) &= e^{-b\sqrt{k}} && \text{(SRD)} \\ \rho(k) &= (k+1)^{-b} && \text{(LRD)} \end{aligned} \quad (58)$$

La fonction de corrélation à utiliser est un paramètre d'entrée du générateur de source (tableau 10 page 85).

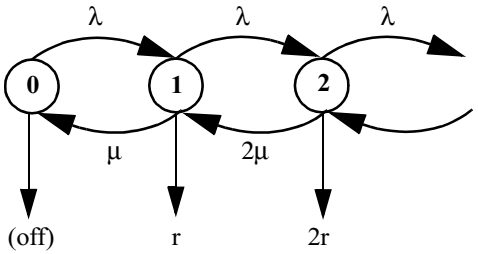
III.1.7.2 Génération des tailles d'image

Une fois les corrélations entres images déterminées, il faut caractériser les tailles des images. Pour cela, il a été montré [ROS95] que les lois Gamma et Lognormal sont bien appropriés. Des mélanges de distribution [KRU98] (Gamma + Pareto) peuvent aussi être utilisés. Le tableau 10 page 85 montre les paramètres de génération de trois codecs utilisant le processus $M/G/\infty$.

III.2 Modèles de trafic agrégé

Pour générer N sources de trafics simultanément, on peut soit utiliser N sources identiques indépendantes, soit utiliser des modèles agrégés. Cependant ceux-ci ne permettent pas de tenir compte du comportement des sources réelles de trafic. En effet, il est intéressant de considérer les fluctuations du nombre de sources actives dans le temps. Ces sources correspondent à des appels d'abonnés à des services particuliers (téléphonie, vidéo, ...), arrivant à des intervalles distribués exponentiellement de taux moyen λ, et restant en communication pendant une durée exponentielle de taux moyen μ. Un processus de naissance et de mort modélise correctement ces arrivées et ces départ d'appels, avec un taux de mort proportionnel au nombre de sources actives (figure 40).

Figure 40: **Processus de naissance et de mort d'appels**



Chaque état représente le nombre de sources actives simultanément. Chaque source génère un taux moyen *r* de paquets par seconde.

Le modèle de file d'attente correspondant est $M/M/\infty$. Appelons *N(t)* le nombre de sources de trafics dans le système à l'instant *t*. Le nombre moyen $\overline{N}$ de sources dans l'état stationnaire, et le taux moyen $\overline{r}$ de génération sont donnés par $\overline{N} = \lambda/\mu$ et $\overline{r} = \overline{N}r$ [KLE75].

Les paramètres d'entrée λ et μ d'un tel modèle peuvent provenir de statistiques faites en métrologie. En effet, les paramètres du système sont complètement déterminés en connaissant le nombre moyen de communications simultanées *N(t)* et la durée moyenne $1/\mu(t)$ de chacune d'elles.

En réalité, le nombre de sources actives simultanément ne peut pas excéder une certaine valeur *C*. Il faut donc pouvoir limiter le nombre d'états du processus de naissance et de mort. On utilise alors un processus d'Erlang B représentant une file d'attente de type *M/M/C/0*. Le nombre moyen de sources dans l'état stationnaire ainsi que le taux global moyen de génération sont donnés par [PHI76] :

$$\overline{N} = \rho(1 - \Pi_C) \quad \text{et} \quad \overline{r} = r\overline{N}$$

avec

$$\rho = \frac{\lambda}{\mu} \quad \text{et} \quad \Pi_j = \frac{\rho^j / j!}{\sum_{i=0}^{C} \frac{\rho^i}{i!}} \qquad (59)$$

Deux types d'utilisations de ce système sont possibles :

- soit on connaît λ et μ, et alors on en déduit N,
- soit on connaît *N* et μ, et on peut en déduire λ.

Un exemple d'utilisation de ce modèle est présenté au paragraphe III.3.3.1.

III.3 Générateurs de paquets

La validation des générateurs aléatoires nécessite des outils statistiques. Nous présentons ici une manière de construire la distribution empirique d'un générateur complexe, ainsi qu'une campagne de tests menées sur différents générateurs aléatoires uniformes (qui sont à la base de générateurs plus complexes). Nous présenterons ensuite une série de modèles de sources représentant des codecs audio et vidéo basés sur le générateur aléatoire uniforme choisi.

III.3.1 Générateurs et distribution empirique

Les générateurs servent principalement à produire les inter-arrivées et les tailles de paquets des sources de trafic dans une simulation événementielle. Ils servent aussi à générer des temps aléatoires

de service lors des traitement des paquets dans le réseau.

Il est très utile de pouvoir tracer une distribution empirique à partir d'une réalisation d'un générateur quelconque. Une courbe empirique (une distribution ou une fonction de répartition) est un outil d'analyse permettant de mieux comprendre les modèles de générateurs agrégés. Nous verrons plus loin, par exemple, qu'une agrégation de générateurs audio (basés sur le processus ON-OFF) tend vers une distribution exponentielle. Ce type de constatation peut être également retrouvé par des calculs statistiques, cependant les distributions empiriques permettent d'orienter plus rapidement la recherche de modèles.

Le principe de construction d'une distribution empirique à partir d'une réalisation du générateur concernée est le suivant : Un grand nombre de générations aléatoires est effectué. L'espace des valeurs possibles est discrétisé en intervalles $I_n=[nT..(n+1)T[$ de taille T, et chaque valeur générée v_i incrémente le compteur S_n ($n = v_i/T$) de l'intervalle I_n.
La suite (S_i) est ensuite normalisée (60) de telle sorte que $S(x)=\alpha S_{x/T}$ représente la fonction de distribution (PDF) empirique du générateur. La fonction de répartition (CDF) peut-être également obtenue en intégrant $S(x)$.

$$\alpha^{-1} = \sum_{i=0}^{\infty} S_i \qquad (60)$$

La taille T de l'intervalle est empirique. Une valeur trop grande peut fournir des données imprécises. Une valeur trop petite engendre une consommation de mémoire pouvant dépasser les capacités du calculateur.

III.3.2 Tests de validité pour les générateurs aléatoires uniformes

Nous avons testé quelques *LCG* pris dans la littérature [HEL98][RAN02]. Chacun d'eux a subi une série de tests statistiques de deux classes : les tests d'adéquation, qui vérifient l'uniformité des valeur générées, et les tests d'indépendance, qui permettent de vérifier l'absence de corrélation dans ces valeurs. Ces tests statistiques utilisent un paramètre α de confiance, pris égal à 0.05 (valeur usuelle pour ces tests).

Nous avons vu au III.1.1 qu'afin de tester les générateurs dans des conditions réelles d'utilisation, les tests statistiques doivent être effectués sur un sous ensemble de tirages effectifs. En effet, si le même générateur uniforme est utilisé pour alimenter successivement plusieurs générateurs de source à tour de rôle de façon cyclique, il convient de réaliser les tests statistiques sur des données d'entrée représentant l'utilisation réelle du générateur aléatoire uniforme.

Il n'est pas possible de déterminer avec exactitude le nombre de tirages effectués sur le générateur aléatoire uniforme entre deux prises de valeur effectives par un générateur de source. Ce nombre dépend du nombre de sources utilisées dans la simulation. Ainsi, pour modéliser les cas d'utilisation du générateur uniforme, nous avons réalisé deux campagnes de test. Soit n compris entre 0 et N. Les générations aléatoires ont été effectuées en ne gardant :

- qu'une valeur sur n tirages,
- qu'une valeur sur un nombre aléatoire de tirages compris entre 0 et n.

Les tests statistiques sont réalisés sur *M* valeurs retenues au total. Un test est compté faux quand il ne satisfait pas l'hypothèse H_0 du test statistique.

Pour chaque test statistique, pour chaque valeur de *M* choisie, et pour chaque type d'oubli (constant, aléatoire), *N* tests sont réalisés (pour chaque valeur de *n* dans [0...*N*]). Le tableau 7 reporte la proportion des tests ne satisfaisant pas la condition H_0.

III.3.2.1 Caractéristiques des générateurs testés

Les générateur de type *LCG(m, a, c, I_0)* ont été présentés au paragraphe III.1.1. Nous en présentons trois.

«Minimal Standard Generator» *LCG*(2^{31} - 1, 16807, 1)

Ce générateur date de 1969 (IBM), et offre d'assez bonnes performances. C'est un générateur de période mesurée 2^{31}-2.

«ANSI-C system generator» *LCG*(2^{31}, 1103515245, 12345, 12345)

Ce générateur est le générateur défini par la norme C-ANSI, qui est utilisé par la fonction *rand()* en langage C. Sa période est maximale (soit 2^{31}) mais trop petite. La totalité de la période est malheureusement couverte en quelques secondes sur les processeurs d'aujourd'hui.

«NAG's generator» *LCG*(2^{59}, 13^{13}, 0, 123456789(2^{32} + 1))

La période maximale de ce générateur est 2^{59}. Il n'est pas question de la tester en totalité : le test de périodicité a été arrêté après environ $5x10^{12}$ valeurs générées en approximativement un mois de calculs (environ 1/100'000 de la période annoncée).

III.3.2.2 Tests statistiques

Tableau 7: **Tests statistiques sur des générateur uniforme de type LCG**

$\alpha = 0.05$ $N = 100$			Tests d'adéquation			Tests d'indépendance	
LCG	Nb. échan-tillons	Nb de sauts entre 2 tirages	χ^2	Kolmogorov Smirnov	Cramer Von Mises	Différences premières	Spearman
Minimal Standard	$M=10^4$	constant	2%	7%	5%	4%	4%
		aléatoire	3%	5%	4%	6%	5%
	$M=10^6$	constant	4%	6%	4%	3%	6%
		aléatoire	5%	0%	1%	2%	8%

$\alpha = 0.05$ $N = 100$			Tests d'adéquation			Tests d'indépendance	
LCG	Nb. échan-tillons	Nb de sauts entre 2 tirages	χ^2	Kolmogorov Smirnov	Cramer Von Mises	Différences premières	Spearman
ANSI C	$M=10^{4\backslash}$	constant	7%	5%	4%	4%	5%
		aléatoire	5%	5%	1%	3%	2%
	$M=10^6$	constant	4%	5%	6%	6%	6%
		aléatoire	11%	4%	6%	5%	6%
NAG	$M=10^4$	constant	3%	6%	4%	5%	4%
		aléatoire	4%	5%	5%	3%	3%
	$M=10^6$	constant	4%	2%	4%	1%	3%
		aléatoire	3%	5%	5%	2%	5%

Le tableau 7 montre que la plupart des tests sont corrects (moins de 5% d'erreur dans la plupart des cas). Nous savions que ces générateurs ne sont pas de mauvaise qualité. Ce tableau montre qu'ils restent valides pour nos cas d'utilisation. C'est à dire que le nombre de tirages aléatoires effectués entre deux tirages effectifs pour une source de trafic donnée importe peu.

III.3.2.3 Conclusion

D'après le paragraphe III.1.1, il est clair que le générateur NAG est le plus intéressant grâce à sa grande période. Les deux autres générateurs testés (MSG, ANSIC) sont corrects du point de vue des tests statistiques, mais ont une période beaucoup trop courte et pourraient induire des corrélations indésirables dans la génération des valeurs aléatoires.

Concernant l'utilisation du générateur pour la simulation distribuée, un problème se pose quand à l'initialisation de sa graine. En effet la simulation distribuée implique que différentes instances du simulateurs soient exécutées en parallèle sur des processeurs et des mémoires indépendantes. Chacun d'eux utilise le même générateur aléatoire uniforme. Pour éviter des résultats identiques dans chacune des tâches, il faut initialiser les générateurs avec des graines différentes, suffisamment « distantes » (en nombre d'itération) les unes des autres pour éviter un recouvrement des valeurs générées.

Ces graines ne peuvent pas être tirées aléatoirement (par exemple par un autre générateur) parce que l'on est par sûr qu'elle fasse partie de l'ensemble des valeurs recouvertes par le LCG (sauf pour le générateur ANSI-C).

La solution que nous adoptons est de faire fonctionner le générateur pendant un certain temps, et de retenir certaines valeurs entre des intervalles assez long, c'est à dire plus long que le temps de simulation complète d'un réseau. Ces valeurs serviront de graine pour chaque tâche de la simulation distribuée.

III.3.3 Générateurs issus de distributions

Nous présentons ici quelques générateurs issus de fonctions de distribution connues. Nous illustrerons l'utilisation d'un processus de naissance et de mort pour l'agrégation des générateurs avec la distribution exponentielle. Chaque générateur a un ou deux paramètres. Le premier est appelé *M* pour Moyenne, et le second *V* pour Variance (si nécessaire). Les générateurs sont tels qu'ils génèrent des valeurs qui correspondent à ces paramètres *M* et *V*, mais écartent les valeurs négatives (par retirage).

III.3.3.1 Distribution exponentielle

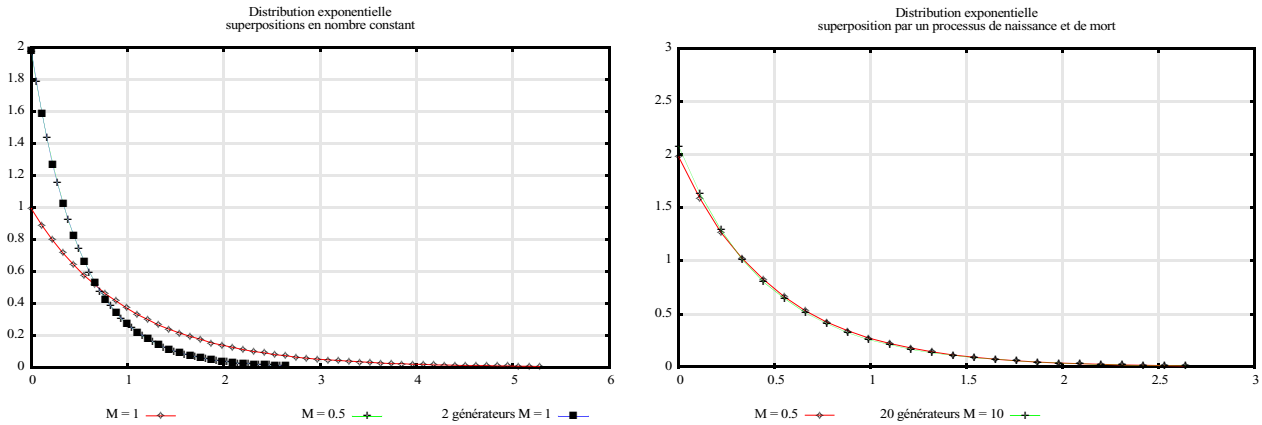
La distribution exponentielle n'a qu'un seul paramètre (la moyenne, notée $M = 1/\lambda$). Sa particularité est de pouvoir représenter beaucoup de phénomènes naturels.
Nous illustrerons dans ce paragraphe la superposition de plusieurs générateurs de même type, en nombre constant, ou en nombre variable issu d'un processus de naissance et de mort, décrit en III.2. La distribution exponentielle a été choisie pour cette illustration car d'une part, elle est connue, et d'autre part la superposition de deux générateurs de paramètre λ issus de la distribution exponentielle est équivalente à un générateur de paramètre 2λ.

La partie gauche de la figure 41 représente :

- La distribution exponentielle de paramètre $M = 1$ ($\lambda = 1$),
- La distribution exponentielle de paramètre $M = 0.5$ ($\lambda = 2$),
- Deux distributions exponentielles superposées avec M = 1 chacune.

La partie droite présente d'une part la distribution exponentielle de paramètre $M = 0.5$ et d'autre part la distribution d'un processus de naissance et de mort de sources exponentielles.
Dans le processus de naissance et de mort, la durée de vie moyenne d'une source est trois minutes (durée exponentielle). La courbe correspond à 20 sources en moyenne, chacune de paramètre M=10.

Figure 41: **Distribution exponentielle**

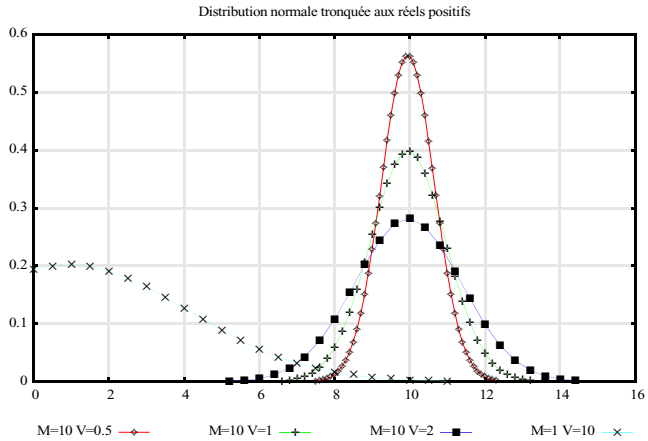


III.3.3.2 Distribution Normale

La distribution normale étant définie sur l'ensemble des réels, on considère sa restriction à l'ensemble des réels positifs. Les deux paramètres du générateur sont la moyenne (*M*) des inter-arrivées et leur variance (*V*). Il faut noter que ces paramètres sont respectés si *M* est grand devant *V*,

car les valeurs générées négatives sont écartées par le générateur.

Figure 42: **Distribution normale tronquée**

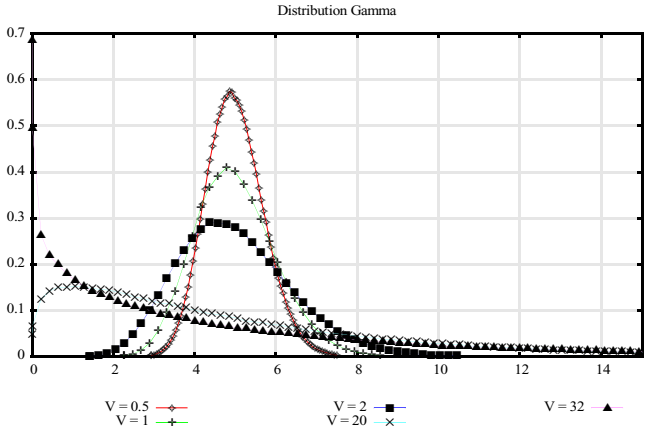


La courbe ci-dessus présente des distributions Normales. Les trois premières sont de moyenne 10 et de variance respectives 0.5, 1 et 2. La dernière est de moyenne 1 et de variance 10.

III.3.3.3 Distribution Gamma

Le générateur issu de la distribution Gamma a deux paramètres qui sont également la moyenne et la variance des inter-arrivées.

Figure 43: **Distribution Gamma**



La courbe ci-dessus présente différentes générations, ayant toutes une moyenne égale à 5, et des variances (V) respectivement de 0.5, 1, 2, 20 et 32 Les différentes formes que peut prendre la distribution en fonction de la variance fait qu'elle est souvent utilisée pour représenter une loi Générale. Des méthodes d'approximation (moyenne, variance) d'une distribution générale à partir d'une distribution Gamma existent.

III.3.4 Sources Audio

III.3.4.1 G711, G726 et G729, RealAudio

Les codecs G711, G726, G729 et RealAudio sont utilisés pour coder la voix. On peut utiliser un

processus IPP ou ON-OFF pour les représenter. Les paramètres du codec G711, par exemple, sont les suivants:

- $\alpha = 1/T_{on} = 1/0.352\ s^{-1}$,
- $\beta = 1/T_{off} = 1/0.650\ s^{-1}$,
- $\lambda = 1/T = 1/0.012\ s^{-1}$ pendant la période ON,
- Taille paquet = 136 octets:
  - 64000bits/s / 8 * 12ms soit 96 octets de données
  - 40 octets d'entête (IP: 20 octets, UDP: 8 octets, et RTP: 12 octets)
  T est le temps de paquétisation pendant la période ON,
- Le λ moyen sur une longue période est : $\overline{\lambda} = 0.352 / (0.352 + 0.650) / 0.012 = 29.3\ s^{-1}$,
- $\overline{D} = \overline{\lambda} * 136 * 8 / 1024 = 31.1$ Kb/s.

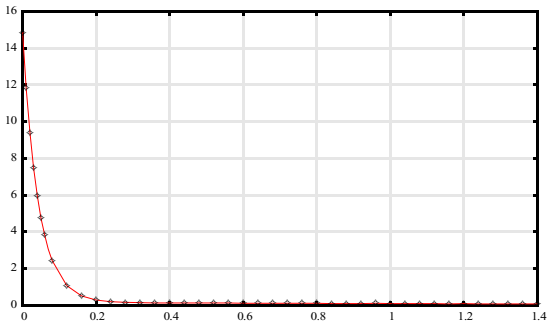
Les mesures faites sur un échantillon de 1 million d'inter-arrivées sont résumées dans le tableau 8.

Tableau 8: **Caractéristiques et mesures (processus IPP) des codecs G711, G726 et G729**

Codec	α (s^{-1})	β (s^{-1})	λ (s^{-1})	$\overline{\lambda}$ (s^{-1})	Débit (état ON) (octets/s)	Taille paquet (octets)	$\overline{D}$ (Kbits/s)	mesure $\overline{\lambda}$ (s^{-1})	mesure $\overline{D}$ (Kbits/s)
G711	1/0.352	1/.0650	1/0.012	29.3	64000	136	31.1	29.2	31.0
G726	1/0.352	1/0.650	1/0.016	22.0	32000	104	17.8	22.1	17.9
G729	1/0.352	1/0.650	1/0.030	11.7	8000	70	6.40	11.6	6.36
RA1	1/0.2	1/1.8	18	1.8	28132	535	7.52	1.80	7.54
RA2	1/0.2	1/1.8	13.3	1.33	22505	333	3.46	1.33	3.46

Pour exemple, la distribution empirique du codec Realaudio2/IPP est représentée en figure 44.

Figure 44: **Distribution du codec RealAudio2 (IPP)**



III.3.4.2 $M/G/\infty$: Sources vidéo

Les caractéristiques générales des codecs vidéo sont présentées dans le tableau 9. Nous ne présentons dans le tableau 10 que les trois modèles dont nous avons parlé précédemment (III.1.7). Pour chacun des générateurs, la fonction de corrélation $\rho(k)$, ainsi que les distributions et les

paramètres permettant de générer la bonne taille des paquets sont données. Les paramètres des processus ont été calculés pour une taille moyenne de 1000 octets par paquet généré.

Tableau 9: **Caractéristiques et débits de plusieurs codecs Vidéo**

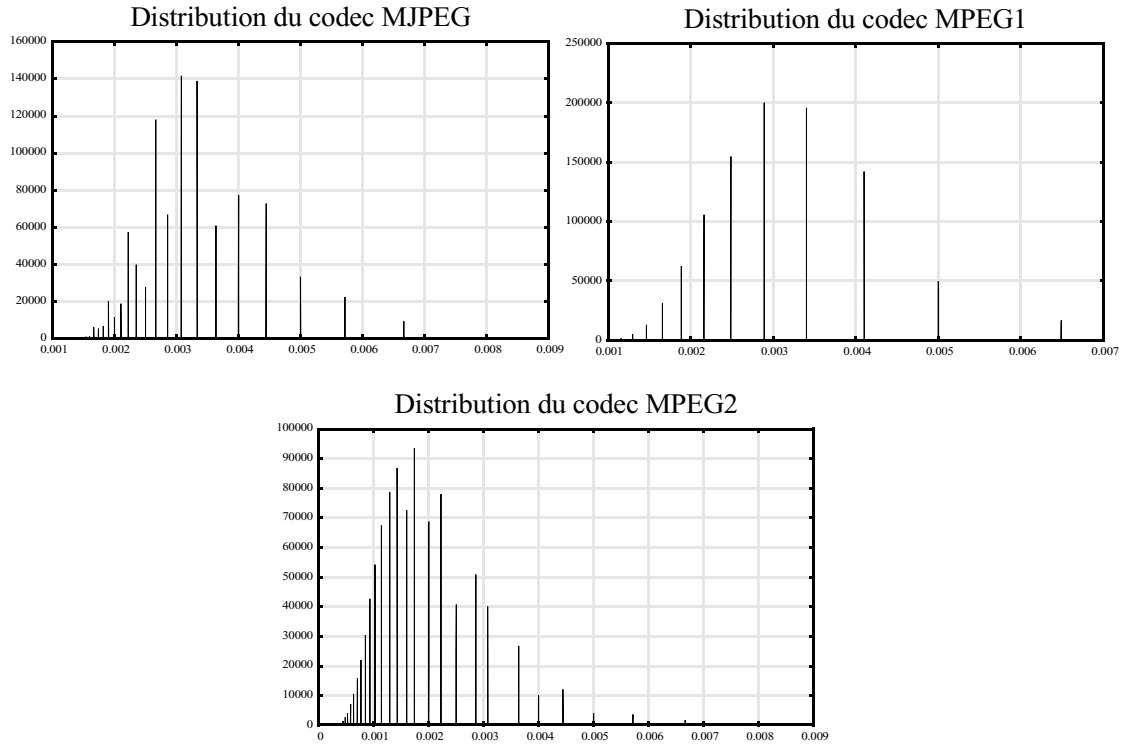
Codec Vidéo	M-JPEG	H261	H263	MPEG-1	MPEG-2
Redondances	spatiales	spatiales et éventuellement prédiction de l'image suivante		spatiales (I) I + image précédente (P) P + image suivante (B)	
Principe	Séquence d'images JPEG indépendantes	Compression spatiale uniquement, ou avec prédiction de l'image suivante		Regroupement d'image en GOP (*Group Of Images*): GOP = IBBPBBPBBPBB	
Débit typique (Mbits/s)	8 à 10	p * 0.064 ($0<p<32$)	0.008 à 1.5	1.5	4 à 10

Tableau 10: **Exemples de paramètres du générateur** $M/G/\infty$

Codec	MJPEG	MPEG2	MPEG1
Fonction de corrélation	$\rho(k) = e^{-0,03\sqrt{k}}$	$\rho(k) = e^{-0,055\sqrt{k}}$	$\rho(k) = e^{-0,35\sqrt{k}}$
Distribution de la taille des paquets (M, V)	Gamma (3.54, 3.54)	LogNormale (2.93, 0.48)	LogNormale (4.9, 0.32)
Fréquence paquets (s^{-1})	306	537	320
Débit moyen (Mbits/s)	2.39	4.20	2.50

Les distributions empiriques mettent en évidence les corrélations. Elle ont été obtenues avec un pas de comptage des échantillons de largeur 10^{-6}. Les distributions ont été construites sur un ensemble de 10^7 tirages.

Figure 45: **Quelques distributions issues du processus** $M/G/\infty$



III.4 Statistiques sur des superpositions de sources de trafic

On sait théoriquement que la superposition d'un nombre infini de sources conduit à un processus de poisson. On mesure ici, suivant le type de source, à partir de combien de sources superposées le processus de Poisson constitue une approximation satisfaisante.

Ces résultats sont importants pour la modélisation analytique du coeur de réseau dans lequel le nombre de sources agrégées est grand.

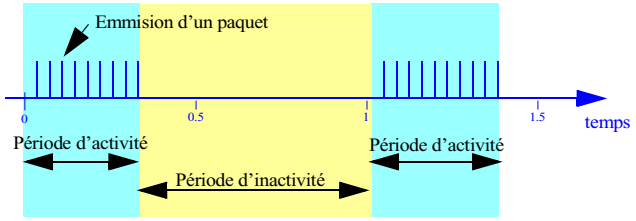
Les courbes sont obtenues par simulation événementielles effectuées avec le simulateur DHS (chapitre V).

III.4.1 Sources Audio avec le Codec Audio G729 ON-OFF

Nous disposons des modèles IPP et ON-OFF pour représenter les flux de type Voix dans l'internet. L'intérêt du modèle IPP est la possibilité d'agréger mathématiquement les processus IPP en processus MMPP-M. Le modèle ON-OFF représente cependant mieux la source réelle et son caractère déterministe pendant la période ON (la période dans laquelle les données digitalisées sont empaquetées et émises à intervalles réguliers).

La figure 46 illustre temporellement un processus ON-OFF. Les durées d'activités (ON) et d'inactivités (OFF) sont exponentielles, et les intervalles de génération de paquets pendant chaque période ON sont constants.

Figure 46: **Représentation temporelle des inter-arrivées des paquets d'un processus ON-OFF**



III.4.1.1 Flux unitaire

Les caractéristiques du codec G729 sont rappelées dans le tableau suivant (la variance est mesurée) :

Tableau 11: **Caractéristiques Codec G729**

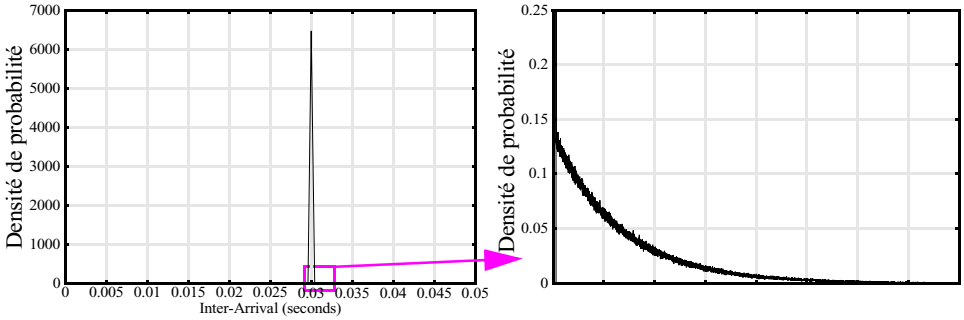
Taux moyen d'émission λ	Durée moyenne d'une inter-arrivée (1/λ)	Variance interarrivées (σ)
11.7 paquets/s	0.00853 s	0.00688 s^2

La distribution empirique de la figure 47a (pas de discrétisation de 10^{-4} secondes) est obtenue par la génération de 10^6 paquets. Pour cette distribution, il est nécessaire d'avoir un pas de discrétisation très fin afin d'obtenir une fonction de distribution correcte; en particulier, si on veut pouvoir observer le « dirac » correspondant au processus dans l'état ON.

Durant la période d'activité, l'émission de paquets est effectuée toutes les trente millisecondes (caractéristique de paquétisation du codec G729). Ceci est clairement visible sur la fonction de distribution et se manifeste par un « dirac » à cet endroit précis.

Le modèle ON-OFF permet de représenter les mécanismes VAD (*Voice Activity Detection* : détection d'activité vocale) conçu pour réduire la bande passante consommée par deux. On en observe les effets dans la figure 47b. En effet, l'introduction d'une période d'inactivité perturbe le caractère déterministe de la source en créant des temps d'inter-arrivées entre paquets de deux périodes d'activité plus grands.
On remarquera que cette partie a l'allure d'une exponentielle, ce qui semble en accord avec le modèle, puisqu'elle représente les intervalles entre le dernier paquet d'une période ON d'une période et le premier paquet de la période ON suivante.

Figure 47: **Distribution du codec G729 ON-OFF**



(a) Vue globale (b) Zoom

III.4.1.2 Superposition en nombre constant de sources G729 ON-OFF

La figure 48 représente l'évolution de la distribution d'un ensemble de sources G729 en fonction de leur nombre. Nous pouvons constater que plus ce nombre est grand, plus la courbe tend vers une distribution exponentielle.

La figure 49 donne les distributions obtenues avec un agrégat de *N* sources G729 comparées avec la distribution exponentielle de même débit.

Afin de valider l'hypothèse de l'approximation d'un agrégat de source G729 par une source de distribution exponentielle, nous avons effectué la comparaison de la variance de l'agrégat de flux G729 avec la variance d'une exponentielle de débit équivalent. Les résultats obtenus sont donnés dans le tableau 12 et illustrés sur la figure 50.

On observe très clairement une convergence de la variance de l'agrégat de flux G729 vers la variance d'un flux Poissonien de même débit. Sur la figure 50b, on peut remarquer qu'à partir de 10 sources G729, le comportement de l'agrégat est équivalent à un flux exponentiel. Cette information est primordiale du point de vue modélisation analytique.

Afin de confirmer ces hypothèses, nous avons chargé une file d'attente G/M/1 dans laquelle les arrivées (distribution générale) sont les arrivées de paquets G729 ON-OFF et le temps de service pour chaque paquet est donné par une distribution exponentielle. Cette charge est comparée à la charge d'une file M/M/1 avec une source exponentielle de même débit moyen (figure 51).

Figure 48: **Superposition en nombre constant de sources G729 ON-OFF**

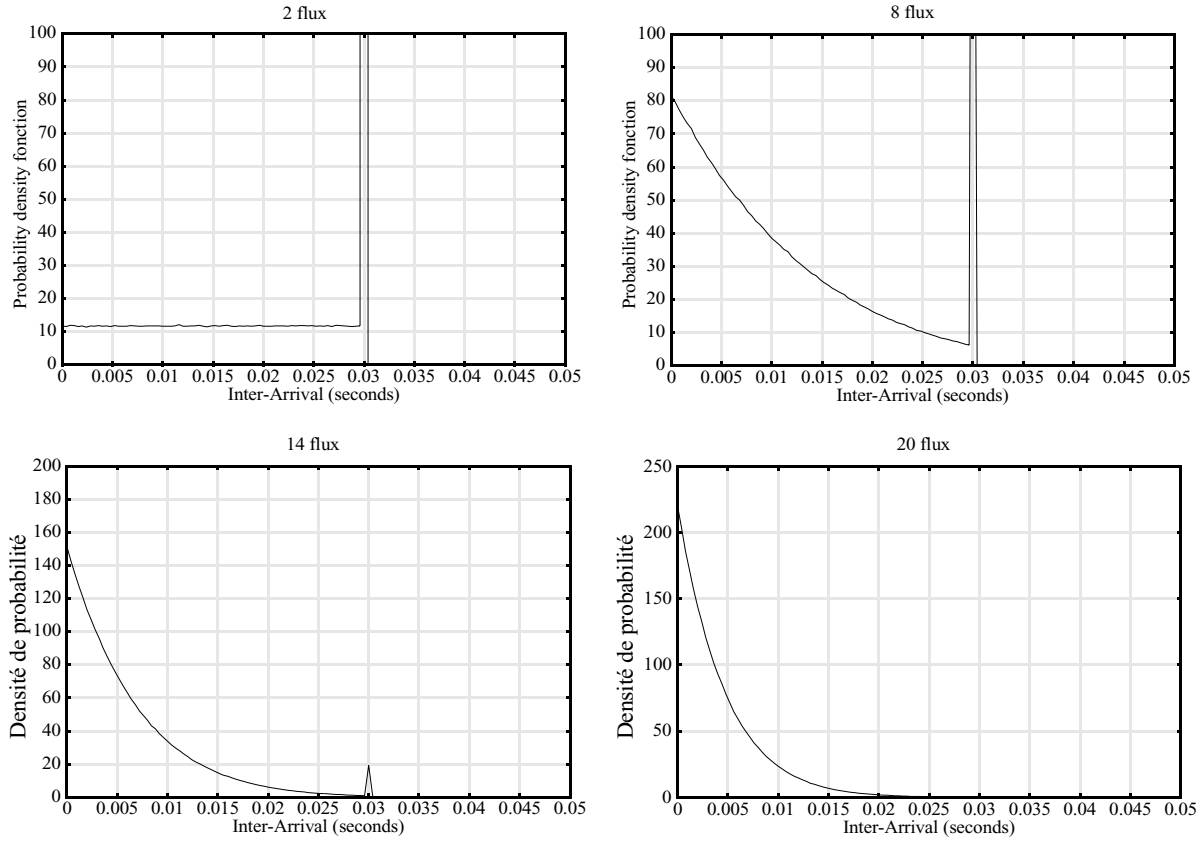


Figure 49: **Comparaison d'un agrégat G729 et d'une exponentielle de même débit**

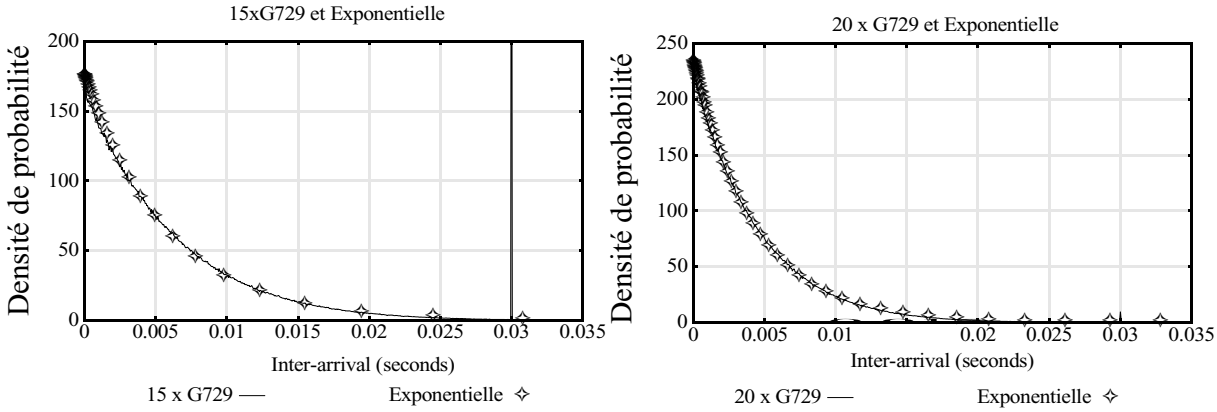
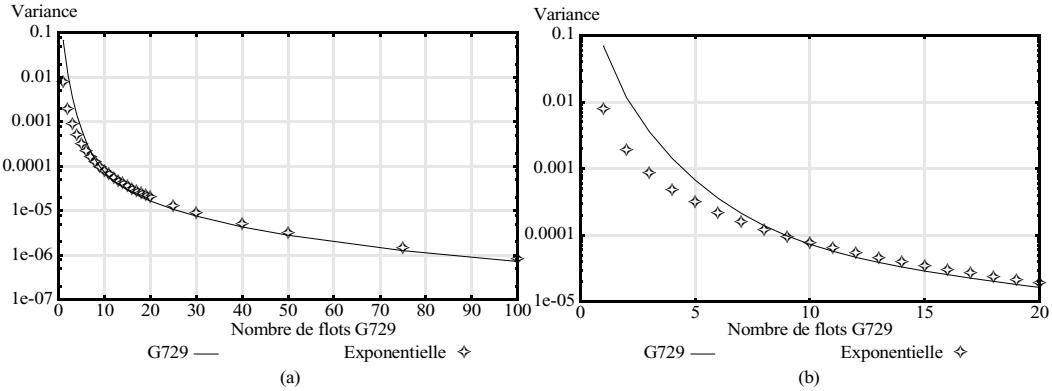


Tableau 12: **Comparaison de variance entre agrégats G729 ON-OFF et exponentielle**

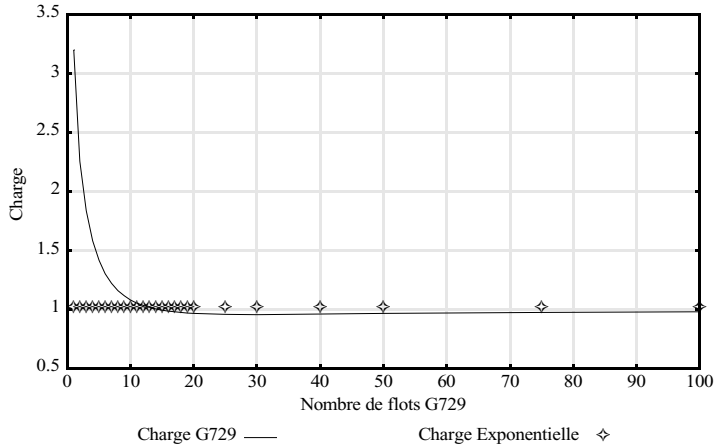
Nb. G729	$\overline{IA} = 1/\overline{\lambda}$	ν aggrégat	ν exponentielle (λ^{-2})
1	8.53922e-02	6.89240e-02	7.29184e-03
2	4.26961e-02	1.16189e-02	1.82296e-03
3	2.84641e-02	3.55594e-03	8.10204e-04
4	2.13481e-02	1.41808e-03	4.55740e-04
5	1.70784e-02	6.67922e-04	2.91673e-04
6	1.42320e-02	3.57859e-04	2.02551e-04
7	1.21989e-02	2.13108e-04	1.48813e-04
8	1.06740e-02	1.38929e-04	1.13935e-04
9	9.48803e-03	9.77790e-05	9.00227e-05
10	8.53922e-03	7.31614e-05	7.29184e-05
15	5.69282e-03	2.90450e-05	3.24082e-05
20	4.26961e-03	1.65371e-05	1.82296e-05
25	3.41569e-03	1.07710e-05	1.16669e-05
30	2.84641e-03	7.58129e-06	8.10204e-06
40	2.13481e-03	4.33500e-06	4.55740e-06
50	1.70785e-03	2.80233e-06	2.91674e-06
75	1.13856e-03	1.26211e-06	1.29633e-06
100	8.53923e-04	7.14861e-07	7.29184e-07

Figure 50: **Comparaison de variance entre agrégats G729 ON-OFF et exponentielle**



Plus précisément, sur la figure 51 sont représentées les charges des files d'attente, proportionnelles au délai de traversée du noeud G/M/1 (Loi de Little). La première courbe correspond à la charge moyenne dans l'état stationnaire de la file en fonction du nombre de sources G729 superposées, et la seconde la charge engendrée par un flux unique de distribution d'inter-arrivées exponentielle dont le débit est égal à celui de la somme des sources. Le temps de service est exponentiellement distribué et le taux moyen de service est deux foix plus grand que le débit des sources considérées (ρ=0.5).

Figure 51: **Comparaison des charges engendrées par *n* sources G729-ON-OFF et une source exponentielle de débit $n*\overline{\lambda}_{G729}$ (ρ=0.5)**



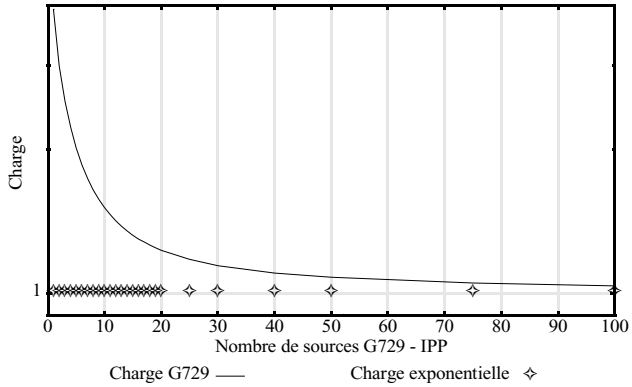
On remarque qu'à partir de 15 sources, l'agrégat des flux G729 peut être approximé par un flux unique exponentiellement distribué. En dessous de 10 sources, les délais et la charge de l'agrégat de sources G729 est nettement supérieur à celui de la source exponentielle.

III.4.1.3 Codec G729 modélisé par un processus IPP

Nous avons effectué le même test que celui de la figure 51, qui consistait en la comparaison des charges engendrées par agrégat de codecs G729-ON-OFF avec des sources exponentielles de même débit respectif, mais cette fois-ci en utilisant un processus G729-IPP, dont on le rappelle, la seule différence avec le processus ON-OFF est la distribution exponentielle des arrivées pendant l'état ON.

Les courbes en figure 52 convergent, mais de manière beaucoup plus lente que dans le cas de l'utilisation du processus ON-OFF.

Figure 52: **Charges engendrées par agrégats G729 de type IPP**



Rappelons que l'intérêt des processus IPP réside dans la possibilité de calculer un modèle théorique de leur agrégation. Les codecs Audio réels génèrent des inter-arrivées fixes pendant la période ON (temps de paquetisation) et sont donc mieux représentés par des processus ON-OFF.

III.4.2 Sources Audio avec le Codec Audio G711 ON-OFF

Nous avons mené la même série de comparaisons en utilisant le codec G711-ON-OFF (tableau 13).

Tableau 13: **Caractéristiques du codec G711**

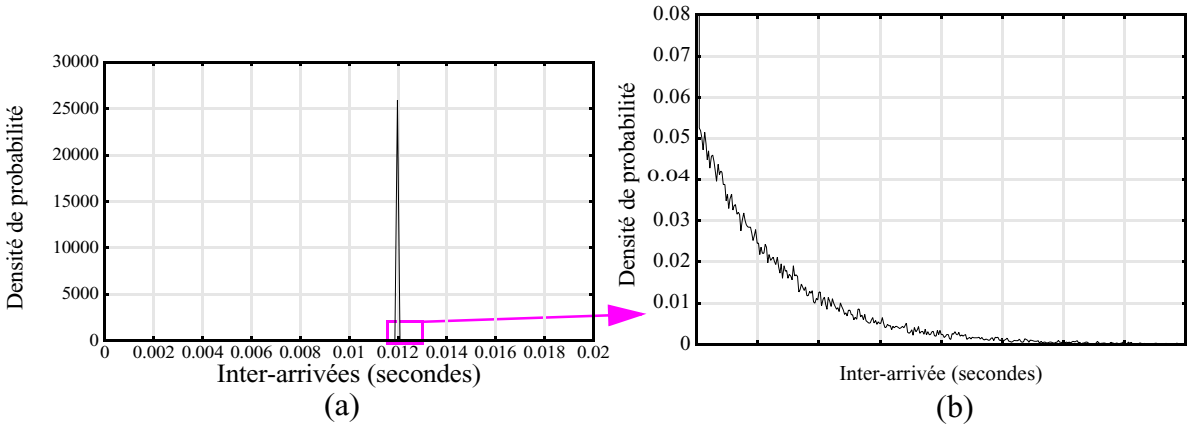
Taux moyen d'émission λ	Durée moyenne d'une inter-arrivée ($1/\lambda$)	Variance inter-arrivées σ
29.3 paquets/s	3.41608e-02	2.83169e-02

Le codec G711 est plus ancien que le codec G729 et ne possède aucun algorithme de compression ce qui conduit à un nombre de paquets moyens plus important que le codec G729. La voix est codée sur 8 bits à une fréquence de 8 kHZ, ce qui donne un débit utile de 64Kbits/s. La génération de flux audio se fait comme avec le codec G729, c'est à dire avec un processus ON-OFF ayant les même caractéristiques. Seul le taux d'émission durant l'état ON change.

III.4.2.1 Flux unitaire

Durant la période d'activité, l'émission de paquets se fait toutes les douze millisecondes (figure 53a).

Figure 53: **Distribution du codec G711 ON-OFF**



III.4.2.2 Superposition en nombre constant de sources G711 ON-OFF

La figure 54 donne les distributions obtenues avec un agrégat de plusieurs sources comparées avec une distribution exponentielle de même débit.

Comme précédemment, nous avons comparé la variance de l'agrégat de flux de codec G711 avec la variance d'une exponentielle de débit équivalent. Les résultats obtenus sont donnés dans le tableau 14 et illustrés sur la figure 55.

Enfin, la comparaison de la charge dans une file G/M/1 par le codec G711-ON-OFF et une source exponentielle de même débit est illustrée par la figure 56.

Ces résultats expérimentaux montrent que la superposition de 15 sources (en nombre constant) de type G711-ON-OFF est bien représentée par une source exponentielle de même débit, tant en terme de distribution des inter-arrivées qu'en terme de charge engendrée dans une file d'attente à service exponentiel.

Figure 54: **Comparaison d'un agrégat G711-ON-OFF et d'une exponentielle de même débit**

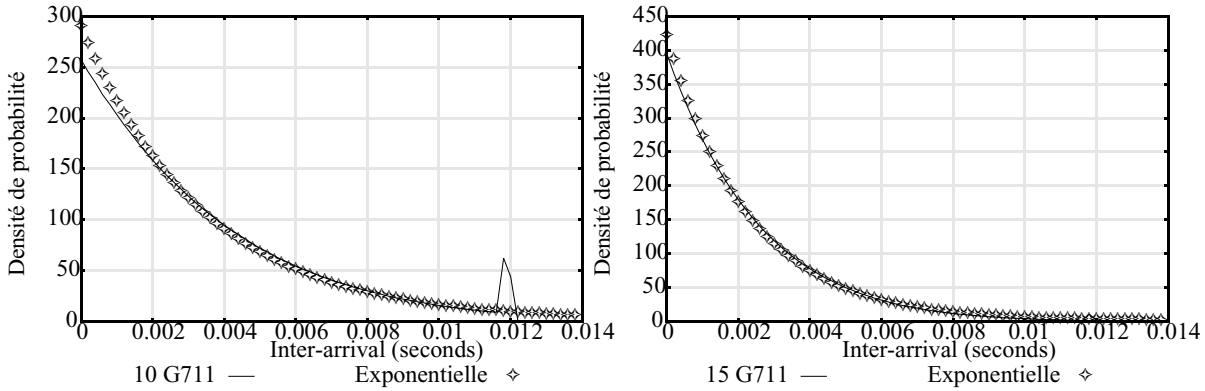


Tableau 14: **Comparaison de variance entre agrégats G711 ON-OFF et exponentielle**

Nb. G711	$1/\overline{\lambda}$	$\overline{\nu}$	$\overline{\nu}$ exponentielle $(1/\overline{\lambda}^2)$
1	3.41608e-02	2.83169e-02	1.16696e-03
2	1.70804e-02	4.66248e-03	2.91741e-04
3	1.13869e-02	1.37704e-03	1.29663e-04
4	8.54021e-03	5.22063e-04	7.29352e-05
5	6.83217e-03	2.29498e-04	4.66785e-05
10	3.41608e-03	1.52251e-05	1.16696e-05
15	2.27739e-03	4.83017e-06	5.18650e-06
20	1.70804e-03	2.65862e-06	2.91741e-06
25	1.36643e-03	1.72528e-06	1.86714e-06
30	1.13869e-03	1.21350e-06	1.29663e-06
40	8.54021e-04	6.93424e-07	7.29352e-07
50	6.83217e-04	4.48407e-07	4.66785e-07
75	4.55478e-04	2.01994e-07	2.07460e-07
100	3.41608e-04	1.14376e-07	1.16696e-07

Figure 55: **Comparaison de variance entre agrégats G711-ON-OFF et Exponentielle**

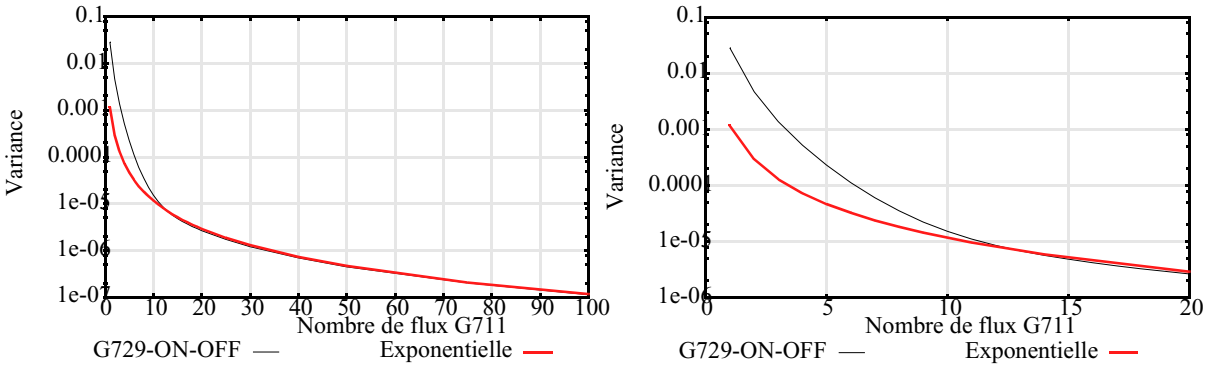


Figure 56: **Comparaison des charges engendrées par *n* sources G711-ON-OFF et une source exponentielle de débit $n*\overline{\lambda}_{G711}$ (ρ=0.5)**

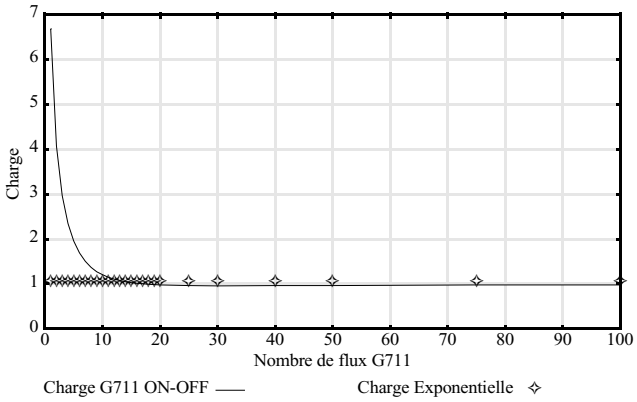
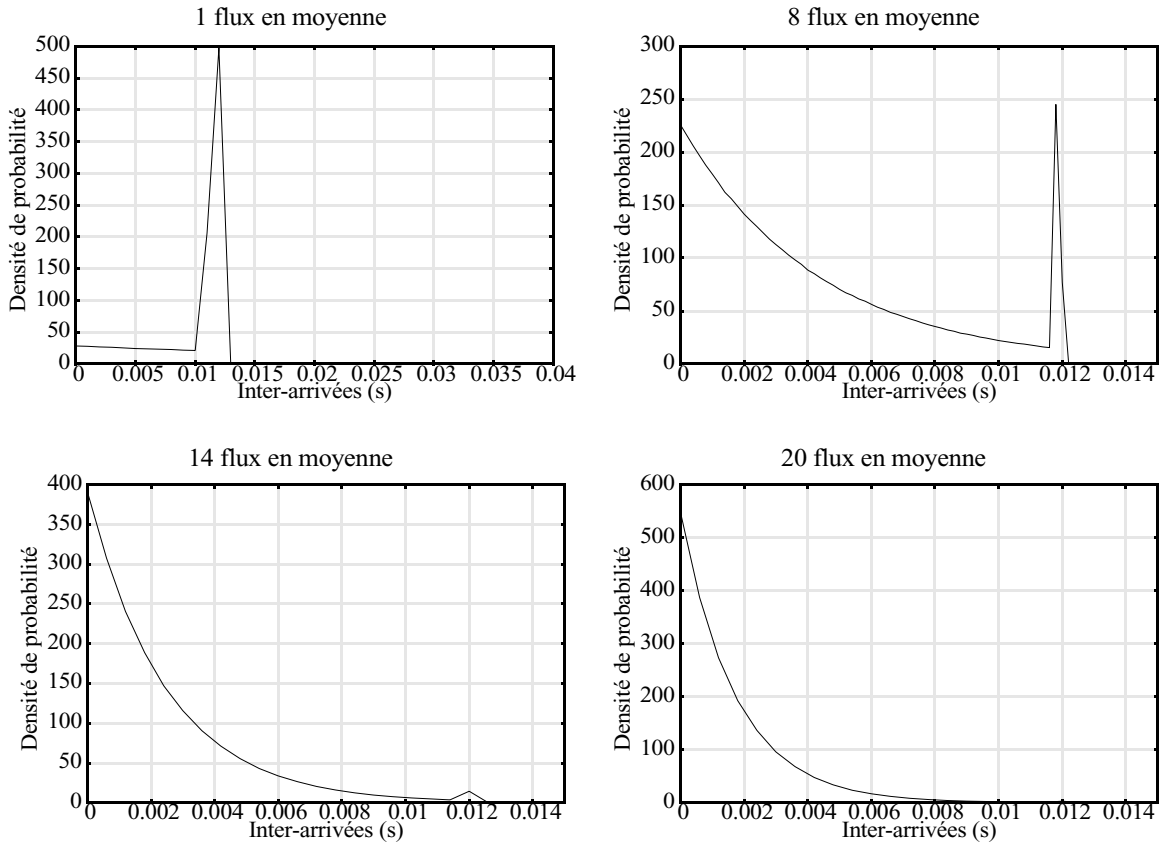


Figure 57: **Superposition en nombre aléatoire de sources G711 ON-OFF**



III.4.2.3 Superposition en nombre aléatoire de sources G711 ON-OFF

Dans les réseaux réels, le nombre de communications de type Voix n'est pas constant. Le nombre de sources simultanément actives peut être représenté par un processus de naissance et de mort ou d'appel.

Nous présentons en figure 57 les distributions des dates d'inter-arrivées de paquets provenant d'un processus de naissance et de mort de sources G711-ON-OFF. Cette figure n'est pas sans rappeler la figure 48 (nombre constant de sources G729-ON-OFF).

Les processus d'appel ou de naissance et de mort pouvant être paramétrés pour avoir en moyenne un nombre déterminé de sources actives, des statistiques sur le processus d'occupation d'une file G/M/1 ont également été effectué.

III.4.3 Sources Audio avec le Codec Audio G726-ON-OFF

III.4.3.1 Flux Unitaire

Tableau 15: **Caractéristiques Codec G726**

Taux moyen d'émission λ	Durée moyenne d'une inter-arrivée ($1/\lambda$)	Variance inter-arrivées σ
21.95 paquets/s	4.55466e-02	3.75329e-02

Le codec G726 est intermédiaire entre le codec G729 et le codec G711. Un algorithme de type ADPCM est utilisé pour la compression. Le débit résultant est de 32 kbits/sec. La génération de flux audio se fait comme avec les codecs G711 et G729, c'est à dire avec un processus ON-OFF de mêmes caractéristiques.

Les résultats statistiques d'agrégations effectuées sur ce codec sont tout à fait comparables aux résultats de G729-ON-OFF et G711-ON-OFF.

III.4.4 Sources Vidéo avec un processus $M/G/\infty$ modélisant le codec MPEG1

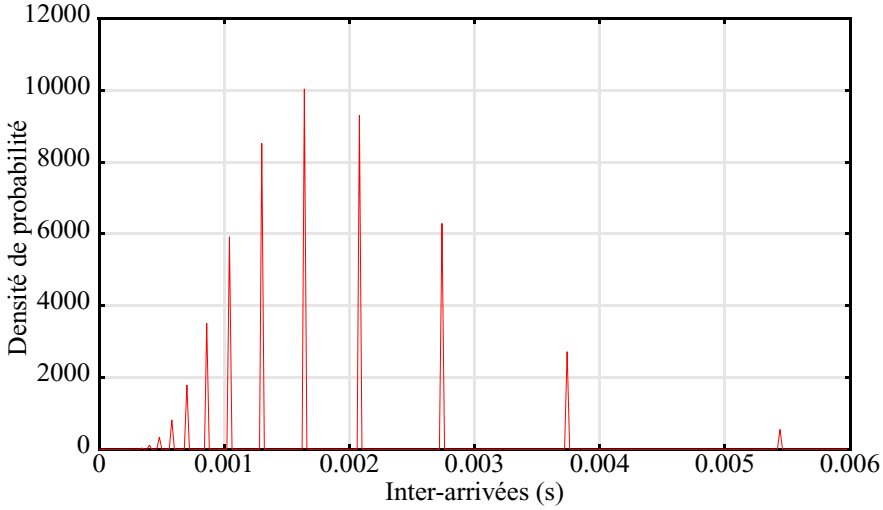
III.4.4.1 Flux unitaire

La figure 58 illustre la distribution des arrivées d'une source vidéo modélisée par un processus $M/G/\infty$ dont les paramètres sont résumés dans le tableau 16. Les données sont générées à l'échelle GOP (12/25s) et la répartition des inter-arrivées dans les *slots* est uniforme.

Tableau 16: **Source vidéo unitaire**

Codec	MPEG1
Fonction de corrélation	$\rho(k) = e^{-0,35\sqrt{k}}$
Distribution de la taille des paquets (moy., var.)	LogNormale (5.36, 0.48)
Fréquence paquets (s^{-1})	562
Débit moyen (Mbits/s)	4.29

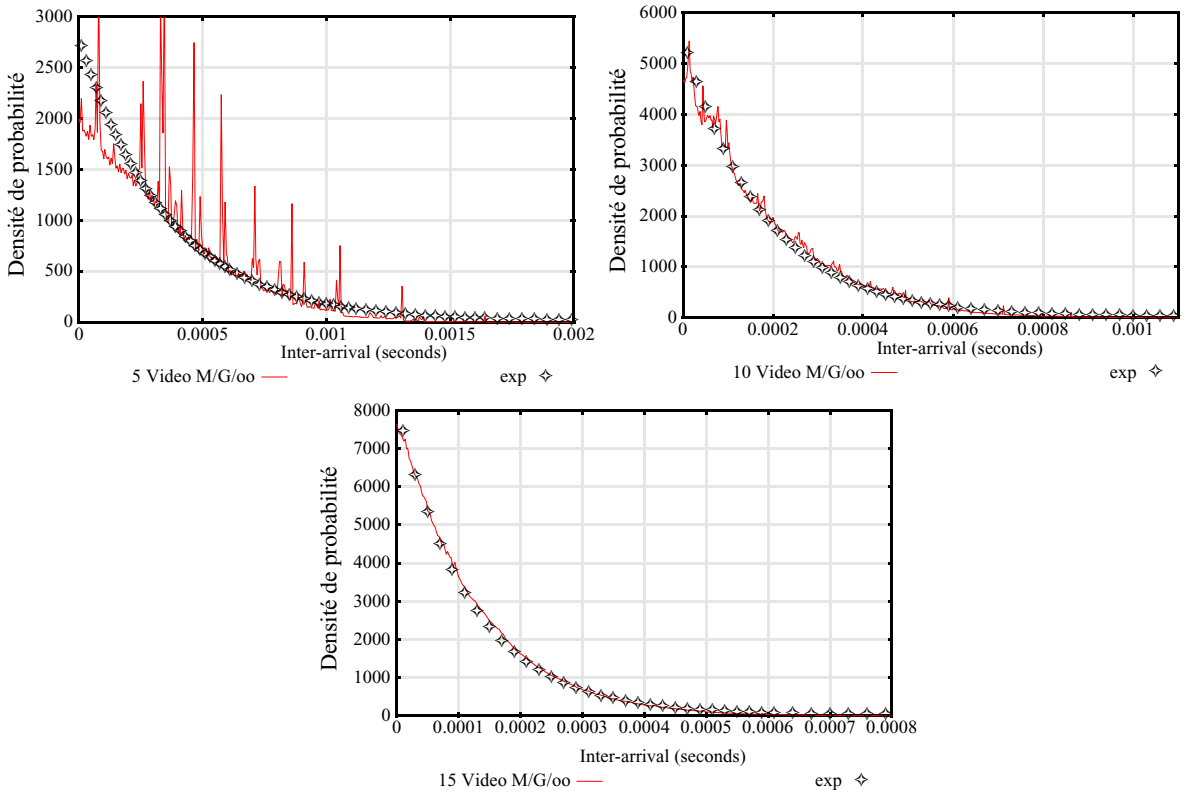
Figure 58: **Distribution des inter-arrivée d'une source Vidéo $M/G/\infty$**



III.4.4.2 Superposition en nombre constant de sources Vidéo

La figure 59 montre la distribution des inter-arrivées issues de la superposition de 5, 10 et 15 sources vidéo $M/G/\infty$, comparées avec les distributions exponentielles respectivement de même débit moyen.

Figure 59: **Comparaison d'un agrégat de sources vidéo $M/G/\infty$ et une exponentielle**



Comme dans le cas des « codecs voix » de type ON-OFF, la distribution d'inter-arrivées d'une superposition de sources $M/G/\infty$ tend rapidement (à partir de 10 sources) vers la distribution exponentielle de même moyenne.

Les variances des inter-arrivée de superpositions des flux vidéo sont également comparées aux variances de sources exponentielles de débit moyen équivalent. Sur le tableau 17 et sur la figure 60a, on peut constater que pour peu de sources vidéo, l'écart est grand (l'échelle des variances est logarithmique), mais tend à diminuer rapidement (à partir de dix sources) lorsque le nombre de sources superposées augmente (l'échelle des variances est linéaire sur la figure b).

Figure 60: **Comparaison des variances d'inter-arrivées de *n* sources Video et d'une source exponentielle de débit $n*\lambda_{vidéo}$**

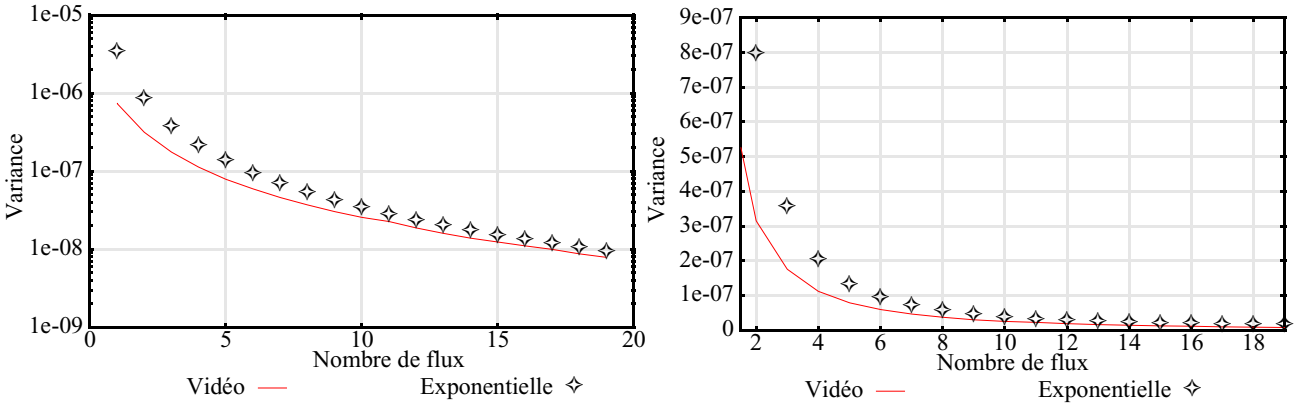


Tableau 17: **Comparaison des variances d'inter-arrivées de *n* sources Video et d'une source exponentielle de débit $n*\lambda_{vidéo}$**

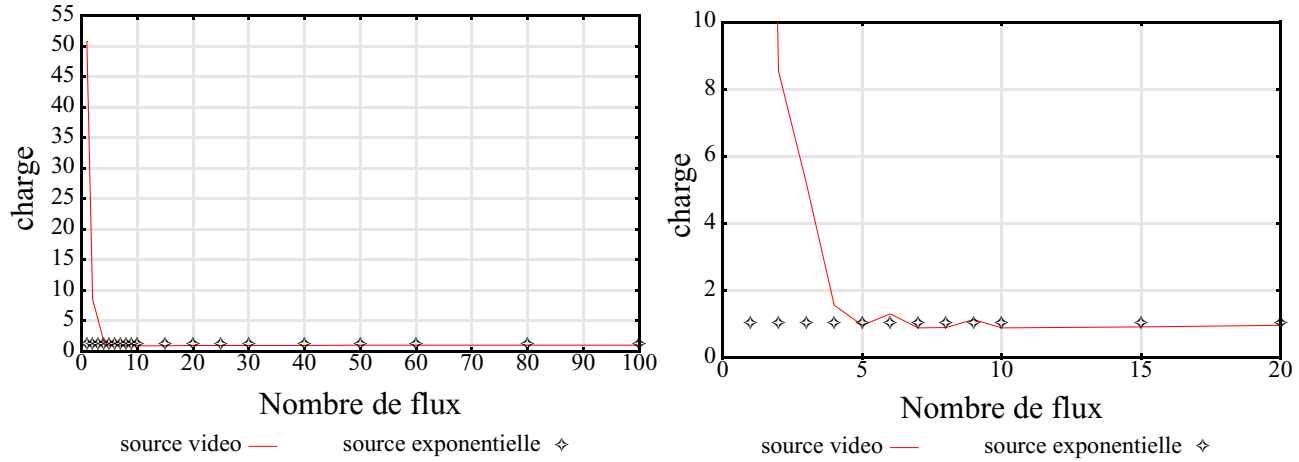
nb sources Vidéo	$1/\overline{\lambda}$	$\overline{v}$	$\overline{v}$ exponentielle $(1/\overline{\lambda}^2)$
1	1.77877e-03	7.40193e-07	3.164023e-6
2	8.89390e-04	3.14669e-07	7.91015e-07
3	5.92866e-04	1.75510e-07	3.51490e-07
4	4.44623e-04	1.12403e-07	1.97689e-07
5	3.55655e-04	7.86318e-08	1.26490e-07
6	2.96082e-04	5.90124e-08	8.76647e-08
7	2.54075e-04	4.60626e-08	6.45541e-08
8	2.22113e-04	3.69375e-08	4.93341e-08
9	1.97346e-04	3.02711e-08	3.89453e-08
10	1.77632e-04	2.54680e-08	3.15531e-08
11	1.61497e-04	2.26548e-08	2.60811e-08
12	1.48051e-04	1.90908e-08	2.19192e-08
13	1.36654e-04	1.62309e-08	1.86742e-08
14	1.26874e-04	1.38896e-08	1.60971e-08
15	1.18420e-04	1.23391e-08	1.40234e-08
16	1.11017e-04	1.10145e-08	1.23247e-08
17	1.04487e-04	9.96003e-09	1.09176e-08
18	9.86879e-05	8.74706e-09	9.73931e-09

nb sources Vidéo	$1/\overline{\lambda}$	$\overline{\nu}$	$\overline{\nu}$ exponentielle ($1/\overline{\lambda}^2$)
19	9.34969e-05	7.88229e-09	8.74167e-09

La charge engendrée dans une file d'attente de type G/M/1 par un agrégat de flux vidéo est également comparée avec celle qui est engendrée par une source exponentielle de même débit (ρ=0.5) en figure 61.

Ici aussi, la superposition d'au moins 10 sources video suffit, dans une file d'attente G/M/1 de taux de service deux fois supérieur au débit d'entrée, à engendrer un comportement identique à celui d'une source événementielle de même débit.

Figure 61: **Comparaison des charges engendrées par *n* sources Video et une source exponentielle de débit $n*\overline{\lambda}_{video}$ (ρ=0.5)**



III.5 Conclusion

Dans ce chapitre, nous avons présenté des modèles de sources de trafic. Ces sources ont été élaborées afin de bien représenter certains trafics multimedia dans les réseaux IP multiservices. Ces modèles de source concernent principalement le trafic audio et le trafic video. Les modèles que nous avons présentés sont basés sur les processus ON-OFF, les processus IPP et sur la discrétisation temporelle de systèmes $M/G/\infty$ (« processus $M/G/\infty$ »).

Les flux vidéos sont issus de codecs video (MPEG2 en général). Une des caractéristiques fondamentales des données transportées (images provenant d'une caméra en mouvement) est leur corrélation spatiale et temporelle. Les processus $M/G/\infty$ sont utilisés pour représenter les corrélations temporelles existant dans les flux vidéos en temps réel (visio-conférence, diffusion de programme de télévision, ...).

Les processus ON-OFF, quant à eux, représentent bien les instants de paroles entrecoupées de silence dans les trafics transportant de la voix. Pendant les instants de parole, les codecs échantillonnent la voix à intervalle régulier et génèrent des paquets de taille fixe à débit constant. Les durées de parole et de silence sont de durées exponentielles ce qui correspond aux comportements naturels et humains.
Les processus IPP sont utilisés pour modéliser les mêmes trafics de type voix. Cependant ils

modélisent de façon moins naturelle les codecs existant du fait de l'aspect exponentiel des inter-arrivées de paquets pendant la période ON. Ces processus restent cependant intéressants parce que l'on sait les agréger mathématiquement de façon exacte en utilisant les processus MMPP-M.

Agréger ces sources de trafic est un problème majeur pour l'évaluation de performances des réseaux. Car en effet, les utilisateurs de ces réseaux sont d'une part nombreux, et d'autre part utilisent les ressources du réseau de façon indépendante. Afin de représenter ces « appels », nous avons utilisé un processus de naissance et de mort régissant le nombre de connexions actives à chaque instant. Les taux d'arrivées et de départs sont, là encore, distribués exponentiellement pour refléter le caractère naturel des instants de départ et des durées de ces appels. Le processus d'Erlang B, que nous utilisons également, est plus précis car il permet de tenir compte du nombre maximum de connexions simultanées dans les réseaux.

L'ensemble de ces sources de trafic a été implémenté dans le simulateur dont nous parlerons au chapitre V. Ce simulateur nous a permis de réaliser et d'analyser quelques agrégations de sources. Nous savons que théoriquement, la distribution des temps d'inter-arrivées de paquets issus de l'agrégation de ces sources en nombre infini est exponentielle. En cherchant à déterminer le nombre de sources à partir duquel ce phénomène devenait visible, nous nous sommes aperçus que ce nombre était raisonnablement faible (typiquement 15) pour les processus ON-OFF et $M/G/\infty$. Cela ouvre des perspectives intéressantes en ce qui concerne la modélisation analytique des réseaux IP car le problème de la modélisation des flux est plus aisément traité lorsque les clients sont distribués exponentiellement.

Chapitre IV - TCP/IP

IV.1 Introduction

Le protocole TCP/IP joue un rôle fondamental dans le réseau Internet car il assure des connexions sans perte de bout-en-bout (livraison en séquence, retransmission des paquets perdus, exploitation efficace de la connexion). De fait, un grand nombre d'applications l'utilise (les services les plus courants comme la moindre application propriétaire), et beaucoup de protocoles indispensables au fonctionnement du réseau reposent également sur ce protocole de transport fiable.

Le protocole TCP est le fruit d'une longue évolution. Ses principes de bases ont été décrits dans [CER74] en 1974. Il fut ensuite standardisé dans plusieurs IEN (*Internet Engineering Notes*) puis repris dans différents RFC [RFC761] (1980), [RFC793] (1981). Outre la fiabilité dans le transport des données, le problème majeur auquel doit faire face ce protocole est le problème du contrôle de la congestion dans les réseaux IP. TCP intègre des mécanismes de contrôle de congestion [BAR01] propres, qui ont été le fruit d'une longue évolution au fur et à mesure du développement du réseau Internet et des problèmes engendrés par la montée en charge du trafic.

Dans ce chapitre, nous allons décrire en détail le protocole TCP/IP, principalement dans sa version *NewReno,* décrite en 1999 dans [RFC2582]. Cette description aide d'une part à mieux comprendre le fonctionnement du protocole de transport standard, et d'autre part reflète l'implémentation de TCP réalisée dans le simulateur DHS (voir chapitre V).
Pour améliorer la compréhension, nous décrirons brièvement l'évolution et les caractéristiques des mécanismes de contrôle de congestion utilisés dans les différentes déclinaisons de TCP.

Une analyse du comportement d'une source TCP en boucle fermée a également été développé. Cette analyse a permis l'élaboration d'un modèle de source TCP analytique dynamique que nous présentons ensuite.

Nous présentons enfin en annexe A.1 un modèle analytique fluide de sources TCP agrégées. Ce modèle tient compte des deux principales phases du protocole : la phase *Slow-Start* et *Congestion Avoidance*. L'aspect fluide de ce modèle de source est tout à fait adapté à la simulation analytique réalisée dans DHS.

IV.2 TCP/IP Version *NewReno*

Les principes généraux du protocole de transport TCP ont été vus au paragraphe I.3.2. Nous allons voir maintenant le détail de l'implémentation qui en a été faite dans le simulateur DHS.

IV.2.1 Notations

Les unités des grandeurs correspondant aux adresses (*offsets*) de paquets et à leur taille sont toutes en octets. Cependant, dans certains simulateurs [NS], ces unités peuvent être exprimés en paquets. L'utilisation de l'une ou l'autre des unités ne change rien à l'algorithme. Dans ce paragraphe (IV.2), qui concerne la simulation événementielle, nous parlerons plutôt en dimension octets. Dans cette dimension, la grandeur *mss* est utilisée pour représenter la taille maximum d'un segment TCP (*Maximum Segment Size*). Pour raisonner en paquet, il suffit de considérer que *mss* vaut 1.

En se plaçant sur une des deux extrémités de la connexion virtuelle, on note :

• ***mss*** (*Maximum Segment Size*)
Il s'agit de la taille maximum utile de données transportées dans un paquet de la couche réseau,

• ***seq*** (SEQuence)
seq est l'adresse de départ du prochain paquet à envoyer,

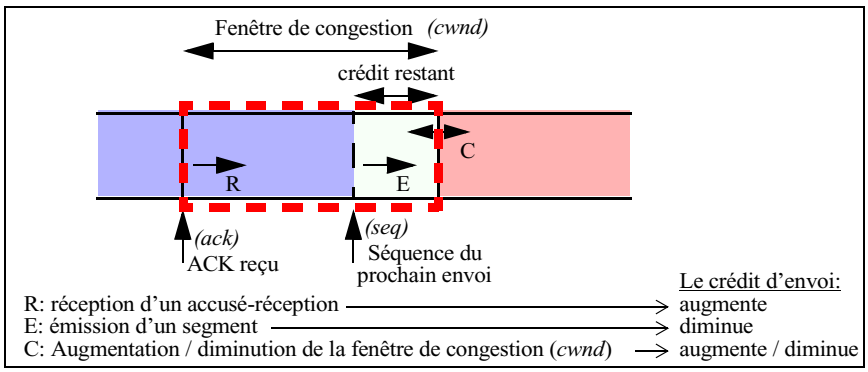
• ***ack*** (*ACKnowlegement* - accusé réception)
La valeur du dernier *ack* reçu correspond à l'adresse de départ des données utiles du prochain paquet attendu par l'autre extrémité,

• ***rtt*** (*Round Trip Time*)
rtt est la mesure du délai entre l'envoi d'un paquet et la réception de l'*ack* correspondant,

• ***rto*** (*Retransmit Time-Out*)
rto est l'estimation de délai maximum entre l'émission d'un paquet et la réception du ack correspondant,

• ***cwnd*** (*Congestion WiNDow*)
La fenêtre de congestion (figure 62), qui peut être assimilée au débit sortant par unité de temps *rtt*. Cette fenêtre représente la taille maximum des données présentes sur le réseau entre l'émetteur et le récepteur. Ce sont des paquets envoyés mais dont la source ne sait pas encore s'ils sont reçus,

Figure 62: **Fenêtre de congestion et crédit d'envoi**



• ***ssthresh*** (*Slow Start THRESHold*)
Le seuil permettant de déterminer l'algorithme à utiliser pour mettre *cwnd* à jour: «*Slow-Start*» en dessous de *ssthresh*, ou «*Congestion Avoidance*» au dessus de *ssthresh*.

• ***rwnd*** (*Receive WiNDow*)
Cela correspond à la taille maximum pouvant être reçue dans le tampon de réception,

• ***ndup*** (*Number of DUPlicate ack*)
ndup est le compteur de valeurs de *ack* identiques reçues par la destination,

• ***credit***
credit est le crédit d'envoi possible par la source. À chaque instant, le crédit d'envoi est donné par la formule:
credit = ack + min*(cwnd,rwnd) - seq + ndup*mss*
Une illustration de ce crédit est donné figure 62 (sans tenir compte de *rwnd* ni de *ndup*).

IV.2.2 Algorithme

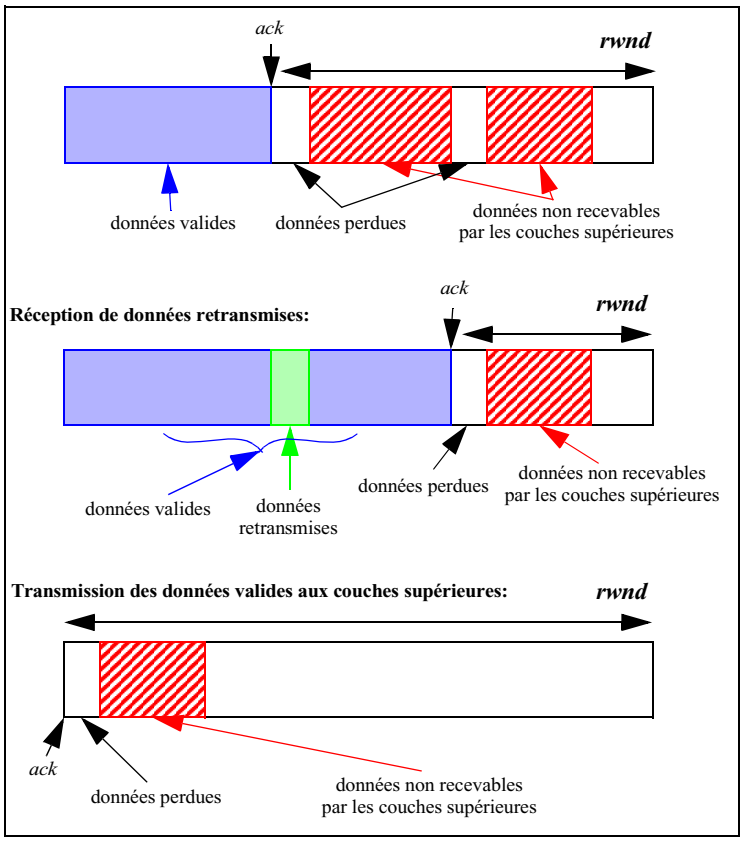
L'algorithme TCP est une succession de règles à appliquer sur les valeur définies précédemment en fonction de deux types d'événement qui sont la réception d'un paquet provenant de l'autre extrémité, et le déclenchement d'une alarme. Nous allons dans un premier temps détailler les mécanismes mis en jeu lors de la réception d'un paquet dans une extrémité, puis nous verrons les conditions d'envoi de paquets et l'action des alarmes.

IV.2.2.1 Réception d'un paquet

Chaque extrémité de la connexion suit de manière identique un ensemble d'algorithmes, et ne s'échange des informations (en dehors des données utiles) que par l'intermédiaire de l'entête des paquets échangés. Ces informations sont les champs *seq*, *ack*, l'indicateur «ACK présent», *rwnd*, et la taille des données utiles (dans le cadre d'un fonctionnement typique). Détaillons un peu plus la signification de ces données.

• *seq*
Cette valeur est maintenue du côté émetteur et est insérée dans l'entête de chaque paquet envoyé. Elle détermine l'adresse de début des données utiles qui seront contenues dans le prochain paquet à envoyer. Cette valeur est normalement incrémentée lors de chaque paquet envoyé, de la taille de ce paquet. Il arrive qu'elle soit modifiée dans deux cas précis.
Le premier cas est la détection d'une perte «*triple ack*». *seq* prend alors la valeur de l'adresse à renvoyer, le temps d'un paquet uniquement, puis reprend sa valeur normale.
Le second cas est la détection de perte sur une alarme de type «*time out*». *seq* reprend alors son évolution depuis l'adresse du paquet à retransmettre,

Figure 63: **Tampon de réception.**



• *ack*
La seule manière qu'a une extrémité d'informer son correspondant de la mauvaise réception des données (un paquet manquant) est de positionner le champ *ack* du paquet à l'adresse du premier octet non reçu (figure 63). La destination doit prendre en compte les données déséquencées (non recevables par les couches hautes) mais ne peut pas rendre compte de leur bonne réception à la source. Une fois qu'une zone perdue est comblée, la prochaine valeur de *ack* renvoyée à destination peut être repoussée jusqu'à la dernière adresse reçue,

• *rwnd*
Le tampon représenté en figure 63 représente la totalité des données pouvant être contenues dans le tampon de destination avant de transférer les données aux couches hautes. La valeur *rwnd* est initialement égale à la taille de ce tampon, mais est décrémentée au fur et à mesure que les données utiles sont stockées et non remises à l'utilisateur. Lors du fonctionnement normal des couches supérieures, on peut supposer que les données valides sont instantanément transmises et n'encombrent pas le tampon de réception.

• taille des données utiles
Elle correspond à la taille du paquet sans l'entête. Cette valeur peut être à zero, car la destination peut transmettre un paquet contenant un *ack* sans pour autant avoir de données à envoyer.

Traitement de *seq, ack* et des données utiles

Lorsque le récepteur reçoit un paquet, *seq* correspond à l'adresse des données dans le tampon de réception. *ack*, côté récepteur, est calculé selon la structure du tampon, l'adresse *seq*, et la taille des données transportées par le paquet. Le récepteur peut renvoyer un paquet transportant l'information *ack*. Cette information indique à l'émetteur l'adresse de début du prochain segment à envoyer (cette valeur de *ack* pourra être la valeur de *seq* d'un prochain paquet reçu).

Lors de la réception d'un paquet contenant des données utiles, celles-ci peuvent ne pas être prises en compte dans les cas suivants :

• L'adresse de fin des données reçues (seq + taille du paquet) est inférieure à *ack* (du récepteur en figure 63),

• L'adresse de début des données reçues est supérieure à *ack*+*rwnd* (du récepteur).

Dans les autres cas, il faut placer ces données (ou du moins celles qui rentrent dans l'intervalle [*ack* .. *ack*+*rwnd*[dans le tampon de réception et calculer la nouvelle valeur de *ack*, correspondant à l'adresse maximum des données contiguës (la nouvelle valeur du prochain *ack* à envoyer).

Rappelons que dans tous les cas, le prochain *ack* envoyé à l'autre extrémité contient l'adresse de début de la prochaine donnée devant être reçue pour garder ou reconstituer un tampon de réception connexe.

Evolution du débit : *cwnd*

La taille de la fenêtre de congestion est modifiée à chaque réception d'un *ack*. Deux modes coexistent dans TCP-NewReno : «*Slow-Start*» et «*Congestion Avoidance*». La figure 4 (page 23) et la figure 65 (page 109) illustrent l'évolution de *cwnd* à chaque *rtt*.

Le mode «*Slow-Start*» permet de découvrir la bande passante disponible, en augmentant de manière agressive la valeur du débit *cwnd* (cette appellation provient du fait que la découverte se fait à partir d'un rythme très lent : 1 seul paquet pendant le premier *rtt*). Dans ce mode, *cwnd* est augmenté d'un paquet (*mss*) lors de la réception d'un *ack*, ce qui a pour effet d'augmenter le crédit de deux paquets (un via *cwnd*, et un via l'augmentation de *ack*).

Lorsque le débit *cwnd* dépasse le seuil *ssthresh*, on sort du mode «*Slow-Start*» pour entrer dans le mode «*Congestion Avoidance*». Le principe dans ce mode est de considérer qu'un régime stationnaire est atteint, en ne faisant pas augmenter le débit de manière trop forte, mais assez pour exploiter une éventuelle libération de la bande passante de bout-en-bout. Pour ce faire, *cwnd* est

augmenté de 1/*cwnd* paquet (en réalité *mss***mss*/*cwnd* octets), ce qui offre un crédit d'un peu plus d'un paquet lors de la réception d'un *ack*.

En résumé, le crédit offert en fonctionnement normal est, à chaque réception d'un *ack* (en supposant qu'un *ack* est renvoyé à chaque paquet reçu) :

- *Slow-Start* (*cwnd* < *ssthresh*):
  cwnd (débit) est augmenté de *mss*,
  credit est donc augmenté de *2*mss* (2 paquets ajoutés),
- *Congestion Avoidance* (*cwnd* > *ssthresh*):
  cwnd (débit) est augmenté de *mss***mss*/*cwnd,*
  credit est donc augmenté de *mss* + *mss***mss*/*cwnd* (au moins 1 paquet ajouté).

Traitement de *ack*

Des pertes peuvent être détectées en analysant la valeur du *ack*, lorsqu'elle est présente dans l'entête du paquet. Chaque extrémité connaît la valeur du précédent *ack* reçu, et possède un compteur *ndup* qui indique le nombre de réception (successive) de cette même valeur. Le premier type de détection de perte dans l'algorithme TCP-NewReno est le «*triple ack detection*», c'est à dire la réception de trois *ack* identiques successifs.

Il peut arriver qu'un *ack* soit reçu plusieurs fois. En effet le protocole réseau IP sous-jacent ne garantit pas que l'ordre de réception des paquets est le même que l'ordre de leur émission. Lorsque deux paquets sont déséquencés, le premier paquet arrivant est stocké dans le tampon de réception, mais si le récepteur renvoie un *ack*, sa valeur n'aura pas changé par rapport à la précédente envoyée. Elle ne changera que lorsque le second paquet (qui devait être le premier) sera reçu (comme l'illustre la figure 63).
C'est pour cela que l'on doit considérer que des données sont perdues (congestions dans les files des noeuds intermédiaires) que lorsque le compteur *ndup* atteint une valeur critique. Cette valeur est prise égale à trois («*triple ack detection*»).

Signification de ndup

ndup est le compteur du nombre de *ack* dupliqués. Lorsque cette valeur atteint sa valeur critique, les algorithmes Fast Retransmit et Fast Recovery sont activés, et la source TCP va arrêter d'émettre de nouveaux paquets jusqu'à ce que tous les paquets envoyés avant l'activation de ces algorithmes soient reçus.

Algorithmes Fast Retransmit et Fast Recovery

Le passage à 3 de la valeur *ndup* est considéré comme une détection de perte. Le rôle de l'algorithme *Fast Retransmit* est de retransmettre le premier paquet perdu. Il se poursuit par l'algorithme *Fast Recovery* dont le rôle est d'attendre la fin de la reprise sur erreur. Comme la source ne peut connaître les données perdues que par la valeur du *ack* reçu, les paquets vont être retransmis un par un, à chaque réception d'un nouveau *ack*, et jusqu'à ce que celui-ci corresponde au plus grand paquet précédemment envoyé (c'est à dire *seq*+*mss*). Le récepteur envoie systématiquement un *ack* lorsqu'un paquet comblant des données manquantes dans le tampon de réception est reçu. Une fois l'algorithme *Fast Recovery* terminé, *ndup* est positionné à 0 et le fonctionnement normal de l'émetteur peut reprendre en mode *Congestion Avoidance*.

Plus précisément, la méthode employée est la suivante. Au début de l'algorithme *Fast-Retransmit*, le paquet perdu est retransmis puis le seuil *ssthresh* prend la valeur max(*cwnd*/2, 2**mss*) (soit une division par deux du seuil), et le débit *cwnd* prend la valeur *ssthresh* plus trois segments (pour récupérer artificiellement le crédit des trois paquets ayant provoqué les acquittements dupliqués). On entre alors dans l'algorithme *Fast Recovery*. À chaque nouvel *ack* dupliqué, *ndup* est augmenté de un pour se donner le crédit du précédent paquet retransmis. Lorsque tous les paquets perdus sont retransmis puis acquittés, l'algorithme *Fast Retransmit* est terminée, *cwnd* prend la valeur *ssthresh* : on est alors en mode *Congestion Avoidance*, avec un débit qui est la moitié de celui avant la détection de perte.

IV.2.2.2 Emission d'un paquet

Le fonctionnement de TCP nécessite l'émission de paquets depuis chaque extrémité de la connexion, et cela même si une seule a des données utiles à transmettre. En effet une source ne peut émettre que si elle acquiert un crédit d'envoi, et celui-ci ne peut augmenter que grâce à la réception de *ack* provenant du récepteur. Aucun envoi de données utiles n'est possible si le crédit est nul en fonctionnement normal (une erreur de type «*triple ack detection*» entraîne un envoi immédiat même si le crédit est nul).

Emission d'un paquet régulier

Un paquet doit être envoyé dès que le tampon d'émission est rempli par les couches supérieures. Ces données peuvent arriver de façon massive et continue (transfert de fichier *ftp*), ou au goutte à goutte, comme dans le cas d'une connexion interactive de type *telnet*.

• Algorithme de Nagle

La taille réelle des paquets à transmettre peut être très variable, de 40 octets (taille de l'entête TCP/IP uniquement) à *mss*+40 (taille maximum). Dans le cas d'une grande distance entre les extrémités, un trop grand nombre de petits paquets est un gaspillage de la bande passante. L'algorithme de Nagle consiste à retarder l'émission des paquets de petites tailles (typiquement de taille *mss*/2) jusqu'à ce qu'un paquet de taille plus conséquente puisse être envoyé, ou jusqu'à expiration d'une alarme (typiquement de 200 millisecondes).
Cet algorithme doit cependant pouvoir être désactivé pour certains types de connexion pour lesquels les temps de réponse doivent être très rapides (par exemple *telnet*).

Emission d'un *ack*

Les *ack* sont des données particulières. Un *ack* est transporté dans l'entête du paquet. Pour des raisons d'encombrement de la bande passante, un *ack* n'est pas systématiquement envoyé à chaque réception d'un paquet mais tous les *b* paquets (typiquement 2). Cela ne change pas la valeur du crédit acquis à la source puisqu'il est calculé en fonction de la valeur de *ack*, hormis pendant la phase *Slow-Start* dans laquelle un crédit d'un paquet supplémentaire est obtenu par la source à chaque ack reçu.
Le retard imposé à un *ack* (si un seul doit être renvoyé) est cependant limité dans le temps à 200ms (en général).

Si un *ack* doit être envoyé alors que des données sont retardées à cause de l'algorithme de Nagle, alors l'ensemble est envoyé. De même, si un *ack* n'a pas besoin d'être immédiatement envoyé

alors que des données (assez grandes) le doivent, le tout est également envoyé.

Calcul de *rto*

rtt est une valeur mesurée. Elle correspond à la différence entre la date d'arrivée d'un *ack* et la date d'émission du *seq* correspondant. De la valeur de *rtt* est déduite la valeur de *rto* (tableau 18, [STE94]) qui est utilisée pour la mise en place de l'alarme de retransmission.

Tableau 18: **Estimateur du délai de transmission**

$$Erreur \Leftarrow RTT - RTTmoy$$

$$RTTmoy \Leftarrow RTTmoy + g \cdot Erreur = gRTT + (1-g) \cdot RTTmoy$$

$$Ecart \Leftarrow Ecart + h \cdot (|Erreur| - Ecart) = h \cdot |Erreur| + (1-h) \cdot Ecart$$

$$rto \Leftarrow RTTmoy + 4 \cdot Ecart$$

Les grandeurs mises en jeu dans les piles TCP sont entières. Manipuler des grandeurs réelles est un facteur de ralentissement important dans les mécanismes bas-niveau. L'algorithme de calcul de *rto* tient compte de cela, et fournit un algorithme de calcul en valeur entière. La base de temps utilisée dépend du système, mais est généralement de l'ordre du centième de seconde.

RTTmoy est un estimateur à oubli exponentiel de *rtt. Ecart* est également un estimateur à oubli exponentiel de l'erreur de *RTTmoy* par rapport à *rtt*. Pour chaque paquet envoyé, l'alarme de retransmission est déclenché au bout de *RTTmoy* plus quatre fois *Ecart*, ce qui permet de prendre en compte les fluctuations de *rtt* sur de longs trajets de bout en bout. Les valeurs de *g* et *h* sont respectivement 1/8 et 1/4.

L'algorithme de Karn désactive la mise à jour de *rto* en cas d'erreur de type «*triple ack*» car sur les paquets retransmis, la valeur mesurée de *rtt* serait faussée.

Alarme de retransmission

A l'émission d'un paquet, l'alarme de retransmission «*Retransmit Timeout*» est positionnée à *rto* + la date d'émission du paquet. L'alarme sera déclenchée si *ack* à cet instant n'a pas acquitté le paquet précédemment émis. Ce test représente le second type de détection de perte. Ce type de perte est plus grave que le précédent car il indique un problème sur le réseau.
En effet, une perte de type «*triple ack detection*» indique un paquet perdu. La perte peut être due à un débit trop rapide de la connexion, ou une montée en charge du réseau provoquant la saturation des tampons dans les routeurs intermédiaires. Dans TCP-NewReno, la réaction immédiate est de réduire le débit d'émission par deux, de retransmettre, puis de continuer la transmission. Une alarme de retransmission reflète une erreur plus grave : elle représente le fait que la source ne reçoit plus de *ack*. Les causes sont plus graves qu'une simple saturation de tampons dans les routeurs intermédiaires. Pour récupérer ce type d'erreur, une redécouverte complète du chemin de bout-en-bout est réalisée.

Le débit doit être alors réduit de manière drastique et une redécouverte de la bande passante disponible doit être réalisée («*Slow Start*»). Pour ce faire, le seuil *ssthresh* est réduit à max(*cwnd*/2, 2**mss*), *cwnd* est positionné ensuite à *mss* (un seul paquet), et l'émission reprend à partir du paquet perdu (*seq* est ramené à l'adresse du paquet victime du *timeout*).

IV.2.3 Phénomène d'oscillations (*silly window syndrom*)

Ce phénomène d'oscillation se produit lorsque plusieurs connexions TCP empruntent un même organe du réseau. Tant qu'il n'y a pas congestion de cet organe, les différentes connexions augmentent leurs fenêtres et donc leurs débits ce qui finit par entraîner la saturation de cet équipement. Cette saturation va affecter l'ensemble des connexions traversant cet équipement, ce qui a pour effet de réduire brusquement tous les débits. Le débit des différentes connexions va ensuite augmenter simultanément jusqu'à ce qu'il y ait de nouveau congestion, puis rechuter, et ainsi de suite.

Une parade pour limiter ce phénomène est décrite au paragraphe I.5.2.3 avec le mécanisme RED (*Random Early Detect/Discard*). Ce mécanisme provoque la destruction aléatoire des paquets arrivant dans le tampon avant que celui-ci commence à être saturé. La probabilité de destruction est dépendante du niveau de saturation de la file. Pour une connexion TCP, cela se traduit par une perte de paquet qui engendre la réduction par deux du débit d'envoi. Ainsi, l'ensemble des connexions TCP passant par un mécanisme RED ne perdent pas toutes un paquet au même moment à cause de l'aspect aléatoire du mécanisme de destruction, donc les réductions des fenêtres de congestion ne sont pas toutes effectuées au même moment. Cela engendre une meilleure utilisation de la bande passante de l'équipement, et évite ainsi une congestion totale régulière des flux TCP le traversant.

IV.2.4 Exemple

Nous présentons ici un exemple simple de connexion TCP. Ce test a été réalisé en utilisant le simulateur DHS. Sur une topologie simple, il présente un fonctionnement réel de l'algorithme facilement interprétable. Cette topologie sera reprise ultérieurement pour la mise au point du modèle analytique dynamique.

Le même test a été réalisé par le simulateur NS et donne les mêmes résultats.

IV.2.4.1 Congestion des files d'attentes

Réseau test

Cet exemple illustre l'adaptation du débit d'une source TCP au débit maximum offert de bout-en-bout par le réseau. Le réseau de test utilisé est illustré sur la figure 64. Toutes les files d'attente ont une capacité de cinq paquets, les liens ont un délai de dix millisecondes. Le routeur 1 a des données à transférer vers le routeur 2. Celui-ci n'envoie aucune donnée utile (seulement des *acks*). Les taux de service des interfaces sont choisis de telle manière qu'un risque de forte congestion existe dans l'interface d'entrée du routeur 2.

Estimation grossière de la RTT

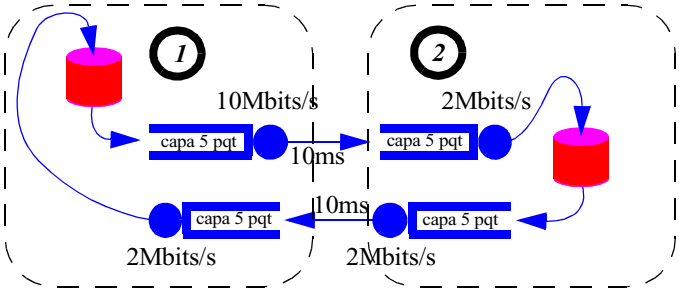
Le débit maximum de ce réseau dans le sens 1 vers 2 est de 2 Mbits/s. Le protocole TCP est connu pour être capable d'atteindre au moins 90% de la bande passante nominale, nous allons donc considérer un débit moyen de 1.8Mbits/s. L'algorithme TCP-NewReno essaie sans cesse d'augmenter le débit d'émission tout en le rajustant en fonction des pertes mesurées. On peut alors supposer que l'interface d'entrée du routeur 2 a une forte probabilité d'être saturé.

Supposons également les files du chemin de retour peu occupées : les paquets ack prennent très

peu de place (40 octets contre 1024) et sont aussi nombreux que les paquets de données (1 réception engendre un *ack* dans cet exemple).

Le temps moyen d'aller-retour peut alors être estimé en sommant un temps de service dans la première file de sortie (paquets de 1024 octets), un délai de lien, le temps d'attente dans la file d'entrée du routeur 2 grâce à la loi de Little (le débit moyen λ étant 1.8Mbits/s et le nombre moyen *N* de paquets dans l'interface approchant 5), puis un temps de service par ack (40 octets), un délai et un second temps de service.
La somme de ces temps donne une approximation de la valeur moyenne de *rtt* de 42.8ms.

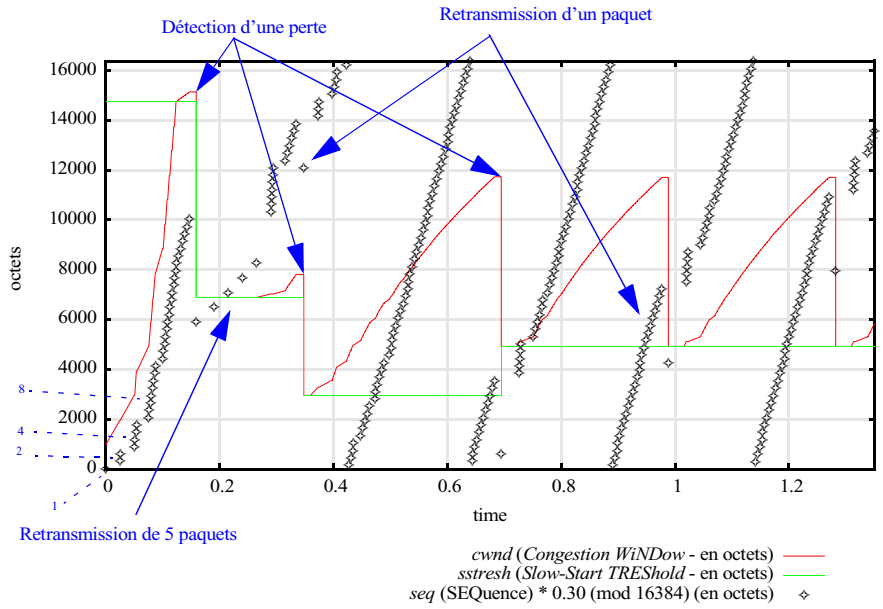
Figure 64: **Réseau test TCP**



Simulation

La figure 65 illustre l'évolution temporelle des grandeurs caractéristiques de la source TCP du routeur 1. Trois grandeurs sont représentées. Ce sont le seuil *ssthresh* (constant par palier), le débit *cwnd* (en dent de scie) et le pointeur *seq*. Celui-ci indique la position du paquet envoyé dans le flux source. Pour faciliter la présentation, cette dernière grandeur n'est pas à l'échelle de l'axe des ordonnées et est de plus tracée modulo 16384.
Sur cet exemple, un *ack* est systématiquement envoyé à chaque paquet reçu par le récepteur, pour bien illustrer l'augmentation exponentielle de la fenêtre de congestion en mode *Slow-Start*.

Figure 65: **Exemple de congestion dans TCP-NewReno**



Détaillons le fonctionnement de l'algorithme TCP-NewReno sur cet exemple.

• démarrage

Nous nous trouvons en mode *Slow-Start*. L'augmentation exponentielle du débit est visible dès le départ. À chaque *rtt*, le nombre de paquets envoyés est doublé (tant que *cwnd* reste sous *ssthresh*). Le débit des paquet devenant trop grand, une perte survient dans l'interface d'entrée du routeur 2. Cela est caractérisé par le petit plat de *cwnd* juste avant chaque chute (plus d'augmentation de *cwnd* car des *ack* multiples sont reçus et *ndup* devient non nul).

• Fast-Retransmit et Fast-Recovery

Cinq paquets sont retransmis au rythme de un par *rtt*, chacun correspondant aux *ack* reçus au fur et à mesure du «colmatage» du tampon de réception (figure 63). Une fois les paquets retransmis, l'algorithme *Fast-Recovery* permet à TCP de poursuivre en mode *Congestion Avoidance*.

• Grandeurs moyennes

Une simulation sur un temps assez long (transmission de $5x10^7$ octets, soit 47.7Mo) a permis de mesurer les grandeurs suivantes:

- Temps de transmission : 220 secondes, ce qui fait un débit de 1776Kbits/s.
- moyenne de *cwnd* mesurée : 8895 octets, moyenne de *rtt* de 39.1ms ce qui donne un débit moyen de 1777Kbits/s (*cwnd* est le débit par *rtt*).
- Pas de perte et moyenne de moins d'un paquet par file, excepté dans l'interface d'entrée du routeur 2 : 2.57 paquets en moyenne et 1.5% de perte.

IV.3 Quelques déclinaisons de TCP

Il existe plusieurs versions et évolutions différentes d'algorithmes TCP dont on peut citer quelques exemples :

- TCP-*Tahoe*,
- TCP-*Reno*,
- TCP-*NewReno*,
- TCP-SACK (*Selective ACKnowlegement* : Acquittement sélectif),

Ces versions sont également décrits dans [FAL96], qui propose une implémentation de TCP-*SACK* dans le simulateur NS et qui le compare aux autres versions de TCP. Il est intéressant de connaître les évolutions successives de TCP, qui concernent principalement le contrôle de congestion, pour l'élaboration de modèles analytiques.

Nous allons décrire ici les principales différences entre ces versions de TCP et TCP-*NewReno* que nous avons déjà largement abordé dans ce chapitre.

IV.3.1 TCP-*Pre-Taohe*

Les versions de TCP précédant TCP-*Tahoe* n'avaient que l'alarme *retransmit timeout* (délai de retransmission) pour renvoyer à destination les paquets perdus. L'absence de reprise sur perte par

« *triple ack detection* » n'offrait que de médiocres performances dans un réseau congestionné.

IV.3.2 TCP-*Tahoe*

TCP-*Tahoe* est la première implémentation de TCP utilisant des algorithmes de contrôle de congestion. Ce travail a été réalisé après avoir constaté une considérable dégradation des versions précédentes de TCP sur des réseaux congestionnés [JAC88].

Ces nouveaux algorithmes étaient *Slow-Start*, *Congestion Avoidance* et *Fast Retransmit*.

Par rapport à TCP-*NewReno*,

- un *ack* est renvoyé par le récepteur à chaque réception de paquet au lieu de tous les *b* paquets,
- la fenêtre de congestion est réinitialisée à un paquet au lieu d'être divisée par deux après l'algorithme *Fast Retransmit* ce qui implique un retour systématique en phase *Slow-Start* après chaque reprise sur erreur (pas d'algorithme *Fast Recovery*).

La lacune principale de cette implémentation s'est avérée, plus tard, être l'absence du *Fast Recovery*. En effet, après une retransmission faisant suite à une perte (3 *ack*s dupliqués), la liaison se retrouvait sous-utilisée le temps de la phase *Slow-Start*, réduisant ainsi les performances moyennes de la connexion.

IV.3.3 TCP-*Reno*

Pour faire face aux lacunes de TCP-*Tahoe*, l'algorithme de *Fast Recovery* [JAC90] y a été introduit pour constituer TCP-*Reno*.

La modification est la suivante : Après avoir reçu trois acquittements dupliqués, la source réduit de moitié son débit d'envoi (la fenêtre de congestion *cwnd*) au lieu de la positionner à un seul paquet. Le seuil *ssthresh* étant lui aussi réduit de moitié, la source passe ou reste en phase *Congestion Avoidance*, et utilise la connexion plus efficacement.

D'autre part, chaque acquittement dupliqué faisant suite à une perte est considéré comme représentant un paquet ayant quitté le réseau. L'idée est que ces paquets sont autant de crédits d'envoi valable (les paquets n'ont pas été perdus) que la source peut récupérer sans saturer le réseau. Ces crédits sont alors utilisés pour envoyer, si possible, de nouveaux paquets vers le récepteur en attendant l'acquittement terminant l'algorithme *Fast Retransmit*.

IV.3.4 TCP-*NewReno*

Le problème de TCP-*Reno* est apparu plus tard avec l'augmentation du débit et du volume de données échangées.

Bien que TCP-*Reno* soit capable de faire correctement face à une perte unique de paquet, l'algorithme de retransmission ne s'arrête que lorsque le premier paquet perdu est retransmis.

Cela implique que toutes les pertes survenant entre le moment de la première et le moment de la détection de cette perte ne sont pas récupérées par l'algorithme de retransmission.

TCP-*NewReno* [RFC2582] est une adaptation de l'algorithme *Fast Retransmit* permettant la retransmission de l'ensemble des paquets perdus une fois que la première perte est détectée. Pour pouvoir réaliser cela, l'algorithme *Fast Retransmit* ne s'arrête que lors de la réception d'un

acquittement concernant le dernier paquet envoyé avant la détection d'erreur. Un *ack* indiquant la réception d'un paquet retransmis, mais ne correspondant pas au dernier paquet émis avant la phase de retransmission est appelé *partial ack* (*ack* partiel). *Fast Retransmit* s'arrête à la réception d'un *ack* non partiel, appelé dans ce cas *recovery ack* (*ack* de récupération).

TCP-*NewReno* est meilleur que TCP-*Reno* dans le cas de plusieurs pertes consécutives. TCP-*Reno* est rapidement capable de récupérer la première, mais la source doit attendre l'alarme *retransmit timeout* pour traiter une seconde perte dans la même fenêtre. TCP-*NewReno* retransmet tous les paquets perdus un par un tous les RTT, jusqu'à réception du *recovery ack*.

Un tel exemple de retransmission est visible sur la figure 65 (page 109).

IV.3.5 TCP-*SACK*

Comparé à TCP-*NewReno*, TCP-*SACK* ([RFC2018] et [RFC2883]) réalise un autre type de reprise sur de multiples pertes de paquets. TCP-*SACK* a un fonctionnement de base identique à celui de TCP-*Reno*, à ceci près que la section 'options' (figure 3, page 22) de l'entête TCP est utilisée pour transporter des informations supplémentaires.

Ces nouvelles informations permettent au récepteur d'indiquer à la source les paquets reçus de manière plus informative que l'habituel acquittement (celui-ci n'indique en effet que le dernier paquet reçu de façon contiguë). Ces informations représentent les adresses de bordure des blocs de données présents dans la mémoire du récepteur (figure 63).

TCP-*SACK* est meilleur que TCP-*NewReno* en cas de pertes multiples. En effet, l'algorithme *Fast Retransmit* est informé de l'ensemble des paquets à retransmettre sans pour autant attendre une durée RTT à chaque retransmission pour connaître la perte suivante. Ceci est bien illustré dans [FAL96].

IV.4 Modèle différentiel par phase de TCP-*NewReno*

L'objectif de ce paragraphe est de décrire un modèle différentiel représentant analytiquement le comportement réel de la source TCP.

Les noeuds du réseau sont modélisés par des files d'attentes D/D/1 analytiques de façon à évaluer les pertes et les délais, nécessaires au fonctionnement de TCP.

Nous allons dans un premier temps décrire plus qualitativement le fonctionnement de TCP-NewReno. Cette analyse exclue tout type de perte. En effet, une détection de perte dans le réseau provoquera un changement ou un redémarrage de phase (*Slow-Start* ou *Congestion-Avoidance*) avec de nouveaux paramètres initiaux.

Dans un second temps, nous décrirons un modèle de comportement de réseau avec des files d'attente analytiques *D(t)/D/*1 (où *D(t)* signifie une arrivée déterministe transitoire). L'analyse du comportement du réseau permettra d'évaluer à chaque instant les pertes dans les files d'attente des routeurs et les délais de bout en bout ainsi que *RTT(t)*. Ces valeurs transitoires constituent la boucle de retour sur la source TCP. Elles permettent de calculer les paramètres des phases *Slow-Start* et *Congestion Avoidance*.

IV.4.1 Analyse qualitative

Nous allons nous intéresser à la transmission au niveau de l'émetteur pour mieux comprendre le fonctionnement de TCP en phase *Slow-Start* et en phase *Congestion Avoidance*.

L'établissement de la connexion commence tout d'abord avec un SYN+ACK, ce qui offre un crédit de deux paquets au commencement de la transmission des données effectives.

On note *b* le nombre de paquets devant être reçus par le récepteur avant l'envoi d'un ACK. On suppose qu'il n'y a pas de perte due à un *timeout*. Notre raisonnement par étape consistera à décrire tout ce qu'il se passe entre deux pertes dues à une congestion des files d'attente entre la source et le récepteur.

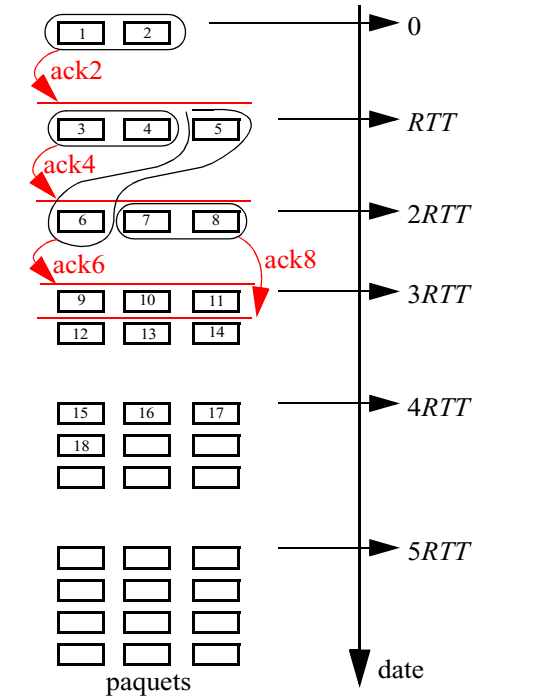
IV.4.1.1 La phase *Slow-Start*

Augmentation exponentielle de débit et limitation

La figure 65 illustre les phases *Slow-Start* et *Congestion-Avoidance* avec des pertes de type *triple-ack* dues à une saturation des tampons. Cette figure illustre bien l'augmentation exponentielle de la fenêtre de congestion (*cwnd*) mais il s'agit d'un cas particulier où *b* vaut 1.

Dans la plupart des implémentations de TCP-*NewReno*, cette valeur vaut 2. La figure 66 illustre l'évolution de *cwnd* dans ce cas.

Figure 66: **Evolution de *cwnd* avec *b*=2**



Un crédit de deux paquets est alloué au début de la communication effective.

À la réception du second paquet, un *ack* est renvoyé par le récepteur vers la source, ce qui offre un crédit de trois paquets. Lorsque les deux premiers sont reçus, un *ack* est renvoyé, et reçu par la source à la date 2RTT.

Trois paquets sont alors envoyés, un *ack* est généré à la réception du premier et du troisième, ce qui permet à la source d'envoyer 6 paquets (date 3RTT).

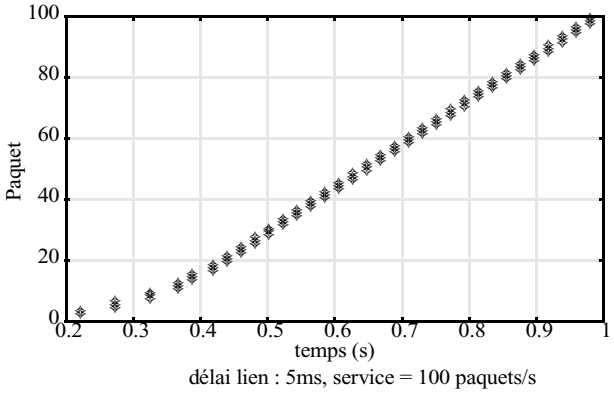
Le processus continue jusqu'à ce que le temps d'envoi d'une rafale atteigne RTT. L'envoi de paquets se fait alors en continu et les tampons finissent par être saturés et engendrer des pertes.

Le nombre de paquets de chaque rafale (r_n) suit approximativement une série géométrique de terme *(b+1)/b*. C'est cette suite qui donne le caractère exponentiel de l'augmentation du débit lors de la phase *slow-start* du protocole TCP.

Il est souvent admis que l'augmentation de *cwnd* est exponentielle pendant la phase de *Slow-Start*, mais il s'agit là d'une approximation. En réalité, les temps de service des paquets ne sont pas nuls, et les paquets arriveront à destination à une cadence maximale liée au routeur le plus lent (cette cadence sera notée μ_{min} dans la suite). Ainsi les acquittements reviennent à la source eux aussi à ce débit maximum (ces paquets sont petits). C'est bien cette limitation du taux de transmission des *acks*

qui limite la bande passante maximum de la source TCP. Donc, le mécanisme d'augmentation de fenêtre TCP en mode *slow-start* a beau être exponentiel, le débit est forcément limité à sa valeur maximale décrite précédemment. Sur la figure 67, on remarque bien ce phénomène avec une partie ayant une allure exponentielle puis devenant linéaire.

Figure 67: **Augmentations exponentielle et linéaire de *cwnd* en mode *Slow-Start***



Influence de RTT

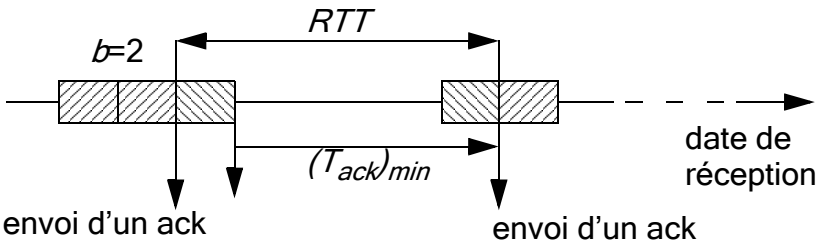
Le récepteur n'envoie un *ack* que tous les *b* paquets reçus. Cependant si un seul paquet est reçu, la source doit en être notifiée tôt ou tard. Les piles TCP possèdent une alarme pour cela, réglée habituellement à 200ms maximum. Il est possible de trouver une valeur minimum pour le temps de transmission des paquets de l'émetteur jusqu'au récepteur pour que cette alarme ne se déclenche pas.

Soit T_{ack} la durée de l'alarme. On suppose que les arrivées se font avec $1/\mu_{min}$ secondes d'écart au minimum, que les files ne sont pas congestionnées. Pour pouvoir envoyer un *ack* pour tous les b=2 paquets sans déclenchement de l'alarme, il faut que :

$$RTT < \frac{1}{\mu_{min}} + T_{ack} \qquad (61)$$

Pour justifier (61), utilisons un diagramme temporel (figure 68). Le récepteur envoie un ack après la réception de deux paquets. Le troisième est reçu $1/\mu_{min}$ plus tard. Un paquet est reçu RTT après l'envoi du ack. (61) provient du fait que l'alarme ne doit pas se déclencher avant la réception de ce paquet.

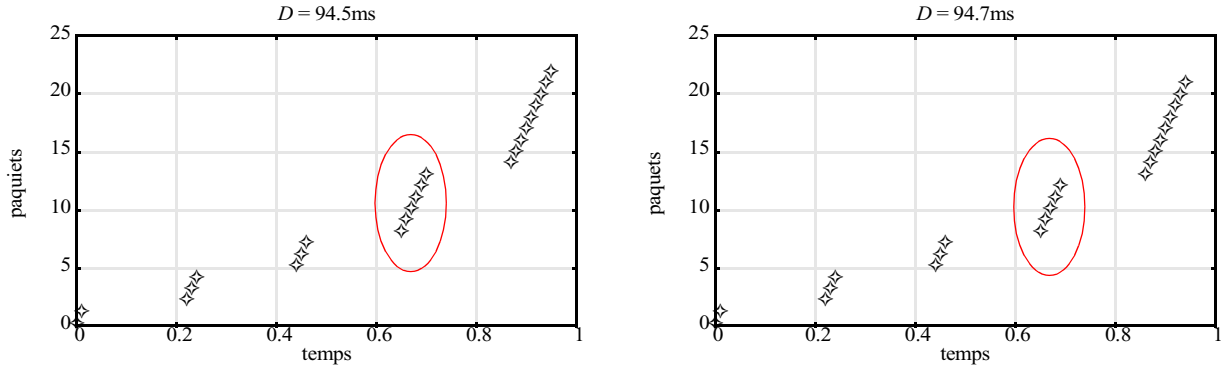
Figure 68: **Délai maximum d'envoi**



Exemple: En reprenant le réseau de la figure 64, pour b=2, μ_{min}=100 paquets/s ou 2560 *ack*s/s (réseau à 10Mbits/s, paquets de 1024 octets, *ack*s de 40 octets), T_{ack}=200ms, il faut que *RTT* soit au maximum de 210ms. Pour vérifier cela, calculons le délai des liens pour être faiblement en deçà ou au delà de cette valeur.
Selon la topologie du réseau, on a $RTT = (D + 2/100)_{(\text{aller paquet})} + (D + 2/2560)_{(\text{retour ack})}$ donc $D <$ (0.2 + 1/100 - 2/100 - 2/2560) / 2 = 0.0946s.

La figure 69 présente la phase *Slow-Start* avec 0.0945 et 0.0947 pour la valeur du délai *D*.

Figure 69: **Influence de RTT sur le fonctionnement de TCP**



Nous pouvons constater que pour D = 94.5ms, le fonctionnement est identique à celui qui est décrit dans par la figure 66. Pour D = 94.7ms, il y a un paquet de moins dans la quatrième rafale.

En pratique, il faut veiller à ce que T_{ack} soit largement supérieur à RTT pour assurer un fonctionnement normal de TCP.

IV.4.1.2 La phase *Congestion Avoidance*

Dans ce mode, chaque *ack* rétribue la source d'un crédit égal au nombre de paquets acquittés plus une petite contribution de l'ordre de *1/cwnd*. Pour chaque *ack* reçu, ce ne sont pas *b+1* paquets qui sont renvoyés systématiquement, mais *b+1/cwnd* en moyenne, ce qui correspond en réalité à *b* le plus souvent, et *b+1* de temps en temps.
Ce mode est donc bien moins agressif que le mode *Slow-Start*.

IV.4.2 Modèle analytique différentiel de TCP-*NewReno*

A partir de cette analyse, nous avons construit un modèle analytique pour le protocole TCP-*NewReno* avec les équations différentielles permettant de décrire en moyenne le comportement de la source.

Nous allons étudier le modèle analytique de chaque phase de la source TCP de façon indépendante et sans considérer les pertes.

IV.4.2.1 Phase *Slow-Start* sans perte

Débit maximum d'émission

Le récepteur reçoit les paquets au rythme maximum $\mu_{\min}$ paquets/seconde. Il envoie un ack tous les *b* paquets reçus. Pour chaque *ack* reçu, la source envoie *b+1* paquets. (*b*+1) paquets sont envoyés tous les $\mu_{\min}/b$ secondes. Le débit maximum d'une rafale en phase *Slow-Start* ($D_{r\text{-}ss}$) à la source est donc :

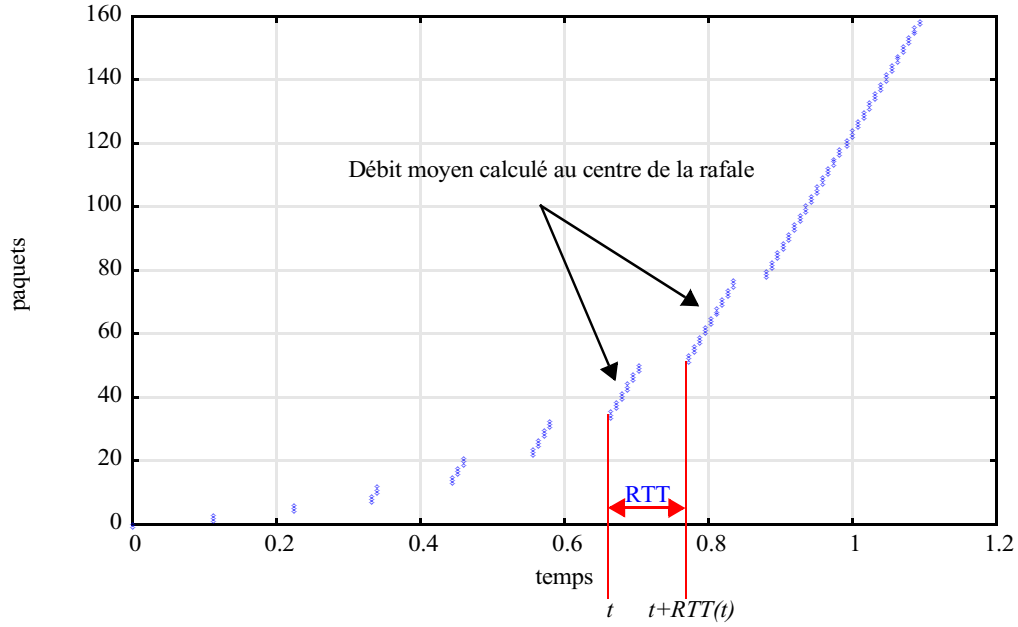
$$D_{r-ss} = \frac{b+1}{b}\mu_{min} \qquad (62)$$

Rafales

La figure suivante illustre les rafales de paquets émises en mode *Slow-Start*. Soit *RTT(t)* le *Round Trip Time* à l'instant *t*. Soit tr_i l'instant de démarrage de la rafale *i*, et soit tr_{i+1} l'instant de démarrage de la rafale *i+1*, on aura :

$$tr_{i+1} = tr_i + RTT(tr_i) \tag{63}$$

Figure 70: **Modèle analytique de la phase *Slow-Start***



Débit de la source

Notons $\lambda(t)$ le débit en paquets par seconde de la source TCP et *cwnd(t)* sa fenêtre de congestion à l'instant *t*. En moyenne, le nombre de paquets envoyés pendant *RTT(t)* est $\lambda(t)RTT(t)$. La durée $d_{r\text{-}ss}(t)$ d'une rafale démarrant à l'instant *t* est donc, d'après (62), de :

$$d_{r-ss}(t) = \frac{b \cdot RTT(t) \cdot \lambda(t)}{(b+1) \cdot \mu_{min}} \tag{64}$$

En mode *Slow-Start*, le débit $\lambda(t)$ augmente sans cesse jusqu'à détection d'une perte de paquet ou jusqu'à atteindre le débit maximum. L'équation suivante est vraie jusqu'à ce qu'une perte soit détectée :

$$\frac{d}{dt}\lambda(t) = \begin{cases} \dfrac{\lambda(t)}{b \cdot RTT(t) \cdot \left[1 + \dfrac{\lambda(t)}{2 \cdot \mu_{min}}\right]} & \text{si } \lambda(t) < D_{r-ss} \\ 0 & \text{sinon} \end{cases} \tag{65}$$

Pour démontrer cela, plaçons nous dans le cas où $\lambda(t)<D_{r\text{-}ss}$ (dans le cas contraire, la valeur 0 est imposée). Le débit de la rafale dans une période RTT est le mieux représenté par $\lambda(t)$ à la date *t* correspondant au milieu de la rafale (figure 70). Dans la période RTT suivante, la durée de la rafale (commençant à *t+RTT(t)*) est donnée par :

$$d_{r-ss}(t + RTT(t)) = d_{r-ss}(t) \cdot \frac{(b+1)}{b} \tag{66}$$

Le débit moyen entre rafales successives suivant également une loi géométrique de terme *(b+1)/b*, on a (figure 70, (66)) :

$$\lambda\left(t + RTT(t) + \frac{d_{r-ss}(t + RTT(t))}{2}\right) = \frac{(b+1)}{b} \cdot \lambda(t) \tag{67}$$

Rappelons le principe d'intégration d'Euler des équations différentielles de premier ordre :

$$\lambda(t + \Delta t) = \lambda(t) + \Delta t \cdot \frac{d}{dt}\lambda(t) \tag{68}$$

Le premier terme de l'équation (67) peut alors s'écrire, en considérant *RTT(t)* suffisamment petit,

$$\lambda\left(t + RTT(t) + \frac{d_{r-ss}(t + RTT(t))}{2}\right) = \lambda(t) + \left(RTT(t) + \frac{d_{r-ss}(t + RTT(t))}{2}\right) \cdot \frac{d}{dt}\lambda(t) \tag{69}$$

l'équation (67) devient alors, en utilisant (66) et (69) :

$$\lambda(t) + \left(RTT(t) + d_{r-ss}(t) \cdot \frac{(b+1)}{2 \cdot b}\right) \cdot \frac{d}{dt}\lambda(t) = \frac{(b+1)}{b} \cdot \lambda(t) \tag{70}$$

et (64) permet de déduire le résultat (65).

IV.4.2.2 Phase *Congestion-Avoidance* sans pertes

Comme dans la phase *Slow–Start*, la phase *Congestion Avoidance* est également dépendante de la valeur de *b*. La source obtient un crédit de *b+1/cwnd* à chaque *ack* reçu. Le temps entre la réception de deux acks est déterminé par le débit de réception des paquets sur le noeud récepteur soit μ_{min}/b, en supposant qu'ils subissent une gigue négligeable sur le chemin du retour. Le débit maximum d'une rafale est donné par :

$$D_{r-ca} = \frac{\text{crédit supplémentaire par } ack}{\text{temps entre deux } acks} = \frac{b + \frac{1}{cwnd(t)}}{\frac{b}{\mu_{min}}} = \left(1 + \frac{1}{b \cdot cwnd(t)}\right) \cdot \mu_{min} \tag{71}$$

La variation du débit est alors donné par l'équation suivante :

$$\frac{d}{dt}\lambda(t) = \begin{cases} \dfrac{1}{b \cdot RTT^2(t)} & \text{si } \lambda(t) < D_{r-ca} \\ 0 & \text{sinon} \end{cases} \tag{72}$$

Pour démontrer cela, plaçons nous dans le cas où $\lambda(t) < D_{r-ca}$ (dans le cas contraire, la valeur 0 est imposée). L'algorithme TCP-*NewReno* en phase *Congestion Avoidance* précise qu'à chaque *ack* reçu, *cwnd(t)* est incrémenté de 1*/cwnd(t)*. Sachant que *cwnd(t)* paquets par RTT sont reçus à la destination, *cwnd(t)/(bRTT(t)) ack*s sont reçus à la source par seconde. Ainsi :

$$\frac{d}{dt}cwnd(t) = \frac{1}{b \cdot RTT(t)} \tag{73}$$

cwnd(t) étant le débit de la source par RTT, on la relation :

$$cwnd(t) = \lambda(t) \cdot RTT(t) \tag{74}$$

En la dérivant, on obtient :

$$\frac{d}{dt}cwnd(t) = \frac{d}{dt}\lambda(t) \cdot RTT(t) + \lambda(t) \cdot \frac{d}{dt}RTT(t) \tag{75}$$

RTT(t) variant peu dans le temps, on peut négliger sa variation par rapport aux autres grandeurs. Le résultat (72) est alors obtenu en utilisant (73).

IV.4.3 Modélisation du *Round-Trip-Time* et des pertes dans le réseau

La source TCP nécessite certaines informations du réseau pour adapter son comportement. Ces informations sont le temps d'aller-retour (RTT) et la probabilité de perte de paquets. Les paquets sont générés par une source TCP sous la forme de rafales de débit constant. De plus, le temps de service étant constant dans les routeurs, ceux-ci sont modélisés par des files d'attente de type *D(t)/D/*1 où *D(t)* signifie une arrivée déterministe transitoire.

Nous nous limitons pour l'instant à la propagation d'une seule source TCP dans le réseau. La simulation de la source et du réseau consistera alors à intégrer pas à pas l'ensemble des équations différentielles (débit de la source, débit de sortie et charge des noeuds). Cela permettra de déduire les paramètres nécessaires au fonctionnement de la source en boucle fermée.

IV.4.3.1 Calcul de *RTT(t)* et de sa dérivée en fonction de la charge

Il est clair qu'avec un réseau ayant beaucoup de trafic, la charge des files d'attente des routeurs va varier aléatoirement et influencer le temps d'aller-retour d'un flux considéré. Dans notre cas, on cherche à établir l'équation de charge du système traversé par un seul flux TCP.

Notons $n_i(t)$ le nombre de paquets dans le tampon de l'interface i, à l'instant t. Notons C_i la capacité en nombre de paquets de l'interface i (capacité de la file plus un) et μ_i le taux de service. Le service est proportionnel à la longueur des paquets, mais comme les paquets sont de longueur fixe dans notre cas, on choisira μ_i déterministe, et son unité est en paquets par seconde.

On supposera en outre que les arrivées se font de manière déterministe avec un taux $\lambda(t)$ donné par TCP ou par le noeud amont. Notons, λ_i le taux d'arrivée des paquets au routeur i, T_i le délai de traversée du nœud i par un paquet à l'instant t et $P_i(t)$ le taux de perte à l'entrée de la file i à l'instant t. L'équation de charge simplifiée du noeud i aura la forme suivante :

$$\frac{d}{dt}n_i(t) = a_i(t) - d_i(t) \tag{76}$$

où

$$a_i(t) = \begin{cases} \lambda_i(t) & \text{si } n_i(t) < C_i \\ 0 & \text{sinon} \end{cases} \tag{77}$$

et

$$d_i(t) = \begin{cases} \lambda_i(t) & \text{si } n_i(t) < 1 \\ \mu_i & \text{sinon} \end{cases} \tag{78}$$

Remarque :

$$\lambda_{i = \text{noeud émetteur}}(t) = \lambda(t)$$
$$\lambda_{j = \text{noeud récepteur}}(t) = \frac{d_{i=n}(t)}{b} \quad (79)$$

Le taux de pertes à l'instant *t* sera donné par :

$$P_i(t) = \begin{cases} 0 & \text{si } n_i(t) < C_i \\ \lambda_i(t) & \text{sinon} \end{cases} \quad (80)$$

Le délai de traversée d'un paquet au routeur *i* sera donné par :

$$T_i(t) = \frac{n_i(t) + 1}{\mu_i} \quad (81)$$

Au départ tous les tampons étant vides, le temps de traversée d'un paquet est le temps (déterministe) de service du paquet. Si on note i=1..n les indices des n routeurs traversés sur le chemin aller entre les deux extrémités de la connexion TCP, et *TA(t)* le temps de transit de bout en bout du paquet, on aura :

$$TA(t) = \sum_{i=1}^{n} \left[\frac{n_i(t) + 1}{\mu_i} + Delai_i \right] \quad (82)$$

Notons maintenant *TR(t)* le temps que mettent les paquets d'acquittement (*ack*s) pour revenir du nœud récepteur au nœud émetteur. Les *ack*s étant de taille beaucoup plus petite (car contenant peu de données), on note α le rapport de taille par rapport aux paquets TCP envoyés par la source :

$$\alpha = \frac{\text{taille paquet TCP regulier}}{\text{taille paquet } ack} \quad (83)$$

L'ensemble des routeurs sur le chemin de retour ont donc un taux de service $\alpha\mu_k$. Notons par des indices j=1..m les routeurs traversés par les *ack*s entre le nœud récepteur et le nœud émetteur, on aura :

$$TR(t) = \sum_{j=1}^{m} \left[\frac{n_j(t) + 1}{\alpha \cdot \mu_j} + Delai_j \right] \quad (84)$$

$Delai_k$ est le délai imposé sur les liens en sortie du noeuds k. On a donc :

$$RTT(t) = TA(t) + TR(t) \quad (85)$$

On peut calculer la variation de *RTT(t)* par rapport au temps :

$$\frac{d}{dt}RTT(t) = \sum_{i=1}^{n} \frac{1}{\mu_i}\frac{d}{dt}n_i(t) + \sum_{j=1}^{m} \frac{1}{\alpha \cdot \mu_j}\frac{d}{dt}n_j(t) \quad (86)$$

IV.4.3.2 Calcul des pertes

Dans notre problème très simple à un seul flux, les pertes proviennent d'un seul noeud, c'est-à-dire de celui qui a le plus petit taux de service, ou de celui qui a le plus petit tampon à taux de service égaux, ou s'ils sont tous identiques, du premier noeud connecté à la source (car la source émet à un débit plus grand que le débit minimum (62), (71)). Les pertes de type *time-out* ne sont pas prises en compte dans notre modèle.

Dans les réseaux réels, les noeuds qui engendrent des pertes sont les plus congestionnés. On peut considérer qu'il n'y en a qu'un seul à un instant donné, et qu'il peut varier au cours du temps.

Les pertes dans notre modèle vont être occasionnées par l'augmentation de débit de TCP. Cette augmentation de débit ne va s'arrêter que lorsque la source sera avertie, par un *triple ack detection*, d'un paquet perdu. En fait quand une file est en état saturé, elle le reste jusqu'à ce que la source TCP s'arrête d'émettre ($\lambda(t)$=0).

Soit i^* le routeur ayant sa file saturée. On a $\lambda_{i^*}(t) > \mu_{i^*}$ et $n_{i^*}(t) > C_{i^*}$. Tant que λ_{i^*} n'est pas plus petit que μ_{i^*}, il y aura des pertes sur le nœud i^* ; ces pertes notées P (en nombre de paquets) sont égales à :

$$P = \int_{\text{instant de debut de perte}}^{\text{instant de fin de perte}} (\lambda_{i^*}(t) - \mu_{i^*})dt \qquad (87)$$

Si l'on suppose que $\lambda_{i^*}(t)$ varie peu entre les instants de début et de fin de blocage, l'équation précédente peut s'écrire :

$$P = (\lambda_{i^*}(t) - \mu_{i^*}) \cdot \Delta t \qquad (88)$$

Cependant pendant cette période de blocage, certains paquets sont quand même passés. Notons X ce nombre de paquets :

$$X = \mu_{i^*} \cdot \Delta t \qquad (89)$$

IV.4.3.3 Analyse et calcul du retard pour arrêter l'émission

La détection de pertes de type *triple ack* nécessite qu'au moins trois paquets suivant celui qui est perdu arrivent à destination. Chacun de ces paquets engendrent un *ack* indiquant un ordre incorrect. Lorsque la source reçoit le troisième *ack* incorrect, elle considère qu'un paquet est perdu et commence la transmission.

Notons Δt_1 le temps (dans la période de congestion) au bout duquel est passé le troisième paquet :

$$\Delta t_1 = \frac{3}{\mu_i} \qquad (90)$$

Notons maintenant Δt_2 le temps au bout duquel la source reçoit l'acquittement de ce troisième paquet. C'est au bout de ce temps là que la source va s'arrêter d'émettre pour retransmettre les paquets perdus et réajuster sa fenêtre. On a donc :

$$\Delta t_2 = \sum_{i=i^*}^{n} \left[\frac{n_i(t) + 1}{\mu_i} + Delai_i \right] + \sum_{j=1}^{m} \left[\frac{n_j(t) + 1}{\alpha \cdot \mu_j} + Delai_j \right] \qquad (91)$$

La somme de ces deux temps est donc le temps entre la première perte et le moment où la source s'en rend compte. On peut évaluer le nombre de perte dans cet intervalle comme suit :

$$P(t_0, t_0 + \Delta t_1 + \Delta t_2) = \int_{t_0}^{t_0 + \Delta t_1 + \Delta t_2} (\lambda_{i^*}(t) - \mu_{i^*})dt \qquad (92)$$

Que l'on peut écrire plus simplement avec notre approximation précédente :

$$P(t_0, t_0 + \Delta t_1 + \Delta t_2) = (\lambda_{i*}(t_0) - \mu_{i*}) \cdot (\Delta t_1 + \Delta t_2) \qquad (93)$$

IV.4.4 Validations

La figure 71 illustre deux simulations d'une source TCP, sur le réseau décrit en figure 64 avec les paramètres suivants :

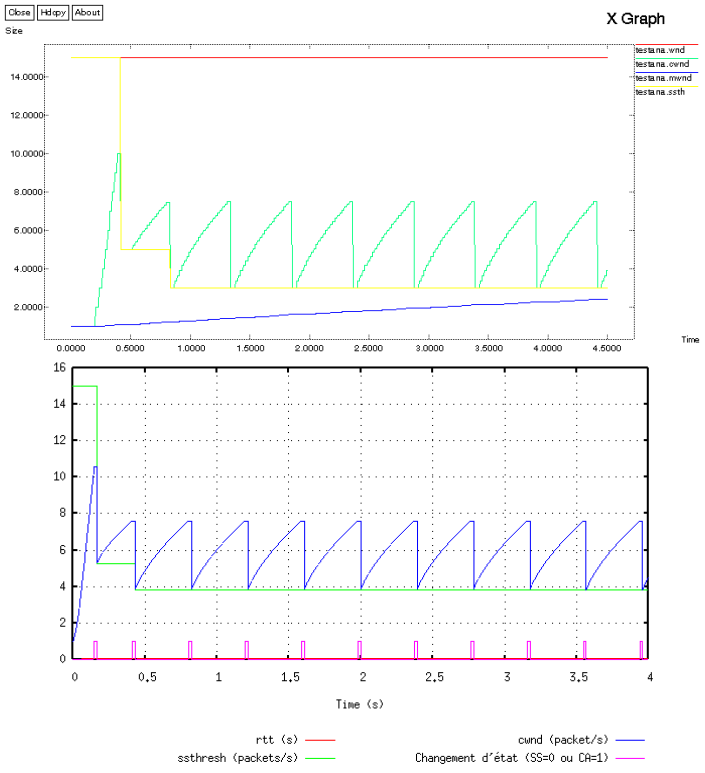
- débit de service : 100 paquets/s (sens aller) et 2560 paquets/s (*ack*s, sens retour),
- capacité de chaque file : 5 paquets,
- seuil initial de *congestion avoidance* : 15 paquets,
- délai des liens : 10ms,
- valeur de b : 2.

La figure 71 et le tableau 19 illustrent le fonctionnement quasi-identique des deux simulations.

Tableau 19: **Comparaison du modèle TCP analytique et d'une simulation événementielle**

	Débit moyen (Ko/s)	Débit maximum (Ko/s)	Paquets perdus
Modèle analytique	95.15	200	317.54
Simulation événementielle	95.81	200	319

Figure 71: **Comparaison du modèle TCP analytique et d'une simulation événementielle**



IV.4.5 Conclusion

L'analyse des performances dans un réseau IP-DiffServ-MPLS, traversé par des flux de toute sorte est devenue possible avec l'ensemble des générateurs de trafic et les modèles de routeurs IP-MPLS qui ont été présentés dans les chapitres précédents.

Cependant, l'ensemble de connexions et des flux générés par TCP devient vite énorme dans un réseau réel. L'approche hybride permet de diminuer singulièrement cette complexité. Toutefois les flux générés par TCP ne sont pas stationnaires, et l'approche différentielle développée ici préserve l'aspect essentiel du protocole TCP qui fait que la source fait varier sans cesse son débit, en boucle fermée en fonction de la réponse du réseau, par une fenêtre de congestion qui augmente puis réduit brutalement. Modèles analytiques fluides de TCP.

Chapitre V - Simulateur hybride

V.1 La difficulté des approches classiques

Une modélisation de bout en bout complète d'un réseau de télécommunication, incluant toute la chaîne des traitements effectués (commutation, transmission, propagation) ainsi que les différents protocoles utilisés, est une tâche très complexe qui ne peut être accomplie qu'en utilisant un environnement de simulation puissant. Cependant, pour des réseaux réels, les temps de calcul de ces simulations événementielles deviennent extrêmement importants. En général, cette complexité calculatoire provient de la taille des réseaux, des matrices de trafic, de la complexité des traitements réalisés dans le réseau ou encore des intervalles de confiance nécessaires pour une estimation correcte.

Au niveau de l'ingénierie du réseau, que ce soit dans le cadre de la conception d'un nouveau réseau ou pour une tâche de planification périodique (pour adapter le réseau à une modification de la demande, introduire une nouvelle technologie, etc.), un algorithme d'optimisation va être utilisé pour optimiser itérativement certains paramètres tels que le routage des flux ou le dimensionnement des équipements. Cette procédure itérative va nécessiter un nombre important d'exécution du modèle de simulation. Evidemment, une telle utilisation d'un modèle de simulation va engendrer des temps de calcul énormes. Pour réduire ces temps, différentes techniques, comme par exemple celles de réduction de variance, peuvent être employées. En particulier, l'utilisation de machines parallèles est un moyen très intéressant pour accélérer les simulations. La difficulté est alors de concevoir des algorithmes parallèles permettant de résoudre efficacement le problème sur une machine multi-processeurs cible. Dans certains cas particuliers, la parallélisation est triviale: le modèle est constitué de composants opérant en parallèle, ou bien on doit en exécuter plusieurs instances indépendantes, une pour chaque valeur des paramètres. Néanmoins, en général, la parallélisation des simulations événementielles ne permet pas des gains de temps importants [MIS86].

Une voie prometteuse pour réduire significativement les temps de calcul est de concevoir des modèles de simulation hybride combinant des modèles de simulation à événements discrets et de simulation continue.

L'objectif du projet ESPRIT $STAR_1$ [STR99] était d'utiliser des technologies de calcul haute performance (HPCN: *High Performance and Computing Network*) pour réduire le temps de calcul nécessaire à la simulation de grands réseaux de télécommunication. Une approche spécifique de Simulation Hybride (analytique et stochastique) a été conçue et validée pour obtenir un gain de temps supplémentaire. Les méthodes analytiques (équations différentielles) sont numériquement très efficaces mais ne peuvent modéliser avec précision que quelques types très simples de réseaux. Les méthodes de simulation à événements discrets sont nécessaires pour analyser les éléments complexes d'un réseau mais sont beaucoup trop lentes pour de grands réseaux.

Le cadre global défini par la simulation hybride consiste à modéliser complètement le réseau par les équations différentielles associées à chaque flux de trafic. Les paramètres de ces équations différentielles sont soit obtenus *via* des relations analytiques non-linéaires (dont quelques unes ont été présentées précédemment) soit *via* des statistiques sur des simulations à événements discrets. Dans le prototype de simulateur hybride $STAR_1$ que nous avons conçu et développé, les noeuds du réseau sont simulés analytiquement lorsque c'est possible tandis que des noeuds plus complexes sont simulés par des méthodes à événements discrets. Ces simulations sont de plus aisément

parallélisables. Ainsi, les temps de simulation sont réduits à la fois par l'utilisation de modèles analytiques et par la possibilité de paralléliser certains calculs.

Dans ce chapitre, nous présentons au paragraphe V.2 le concept de la simulation hybride, conçue dans le cadre de cette thèse et utilisée dans le simulateur hybride. Le fonctionnement du simulateur hybride est détaillé dans le paragraphe V.3. Ce simulateur est capable de réaliser la simulation analytique complète d'un réseau de files d'attente en utilisant les modèles décrits au chapitre II, la simulation événementielle complète des réseaux décrits au chapitre I en utilisant des modèles de source de trafic vus dans le chapitre III. L'originalité de ce simulateur est qu'il utilise les deux concepts de simulation hybride décrites dans ce chapitre. Le paragraphe V.4 précise la manière dont sont réalisés les rapports d'évaluation de performances, et enfin dans le paragraphe V.5 nous montrons comment sont modélisés les réseaux IP-DiffServ-MPLS pour pouvoir être évalué par le simulateur.

V.2 La Simulation Hybride

Pour bien comprendre le concept de simulation hybride, nous allons détailler plus précisément le concept de simulation analytique. Nous décrirons ensuite les deux concepts de simulation hybride que nous avons développés et nous les illustrerons ensuite sur des exemples simples.

V.2.1 La simulation analytique

La simulation analytique consiste à modéliser un réseau de files d'attente par :

- un ensemble de source $\lambda_k(t)$ allant du noeud origine $o(k)$ au noeud destination $d(k)$,
- un ensemble de noeuds modélisés par des équations analytiques issus de la théorie du trafic différentiel dont l'équation générale est rappelée (94),

$$\begin{aligned}\frac{dX_i^k(t)}{dt} &= TE_i^k(t, k, \lambda_i^{s(1)}(t), \ldots, \lambda_i^{s(i)}(t), X_i^{s(1)}(t), \ldots, X_i^{s(i)}(t), \mu) \\ &\quad - TS_i^k(t, k, \lambda_i^{s(1)}(t), \ldots, \lambda_i^{s(i)}(t), X_i^{s(1)}(t), \ldots, X_i^{s(i)}(t), \mu)\end{aligned} \qquad k = s(1)\ldots s(K_i) \qquad (94)$$

où $s_i(1)..s_i(K_i)$ représente les K_i flots arrivant au noeud i. $\lambda_i^k(t)$ et $X_i^k(t)$ (k=$s_i(1)..s_i(K_i)$) sont les charges induites par chaque source, μ le taux de service et TE_i^k et TS_i^k les taux d'entrée et de sortie dont les expressions sont données par le modèle analytique différentiel considéré (chapitre II),

- des coefficients $\alpha^k_{ij}(t)$ permettant de router une proportion du trafic k du noeud i vers le noeud j.

Deux types de simulations peuvent être réalisés :

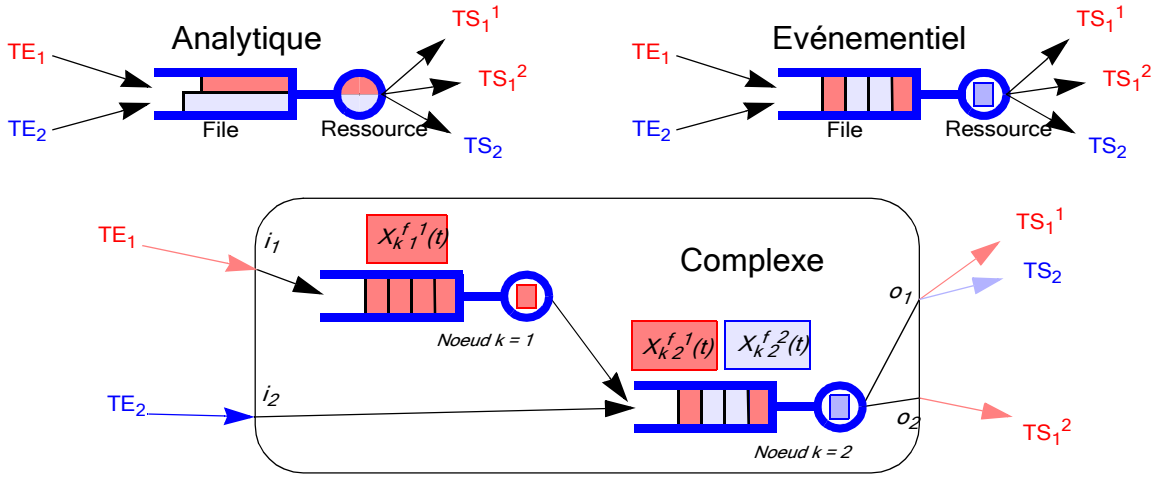
- La simulation analytique transitoire
  L'intervalle de simulation est discrétisé par pas de durée Δt. À chaque pas, et pour chaque noeud i et flot k du réseau, chaque paramètre $TE_i^k(t)$ est la somme des $\alpha^k_{ji}(t)TS_j^k(t)$ (pour tous les α^k_{ji} non nuls), et les équations sont intégrées pour mettre à jour $X_i^k(t)$ et les taux de sortie $TS_i^k(t)$.

• La simulation analytique stationnaire
Dans l'état stationnaire du réseau, toutes les dérivées par rapport au temps sont nulles. La recherche de l'état stationnaire consiste à exécuter un algorithme de point fixe sur l'ensemble des équations $TS_i^k = TE_i^k$.

V.2.2 Le concept de Simulation Hybride

Dans le modèle de simulation hybride, certains noeuds sont simulés avec des équations différentielles analytiques (on les appelle alors les *noeuds analytiques*) tandis que les autres noeuds, en général plus complexes, sont simulés à l'aide d'une modélisation stochastique à événements discrets (on les appelle *noeuds simulés*). La figure 72 présente ces deux types de noeuds, ainsi qu'un *noeud complexe* qui est un groupe de noeuds simulés.

Figure 72: **Les modèles de noeuds analytique, simulé, et complexe**



Il est inenvisageable d'obtenir en un temps acceptable l'évaluation de performance de la totalité d'un réseau complexe grâce à la seule simulation événementielle. Cependant l'analyse fine d'un élément particulier du réseau (un noeud, un flux) obtenue par simulation à événements discrets de cet élément, le reste du réseau étant simulé de manière analytique par des équations différentielles est tout à fait envisageable.

L'idée fondamentale de la *simulation hybride* est qu'un noeud simulé ou qu'un ensemble de noeuds simulés interconnectés (noeud complexe) vont évaluer le comportement de leur propre équation différentielle à l'aide d'une simulation stochastique.

La figure 73 représente un réseau constitué de quatre files d'attente dans lequel un noeud dont la file contient des paquets est simulé, tandis que les 2 autres noeuds sont des noeuds analytiques, et le quatrième un *noeud hybride*. Un noeud hybride est un noeud analytique qu'un flux de paquets peut traverser. Les paquets sont retardés lorsqu'ils traversent le noeud analytique, en fonction de l'état de ce dernier.

V.2.3 Deux principes de simulation hybride

Nous avons donc à disposition deux niveaux de simulation hybride permettant une analyse fine des composants du réseau :

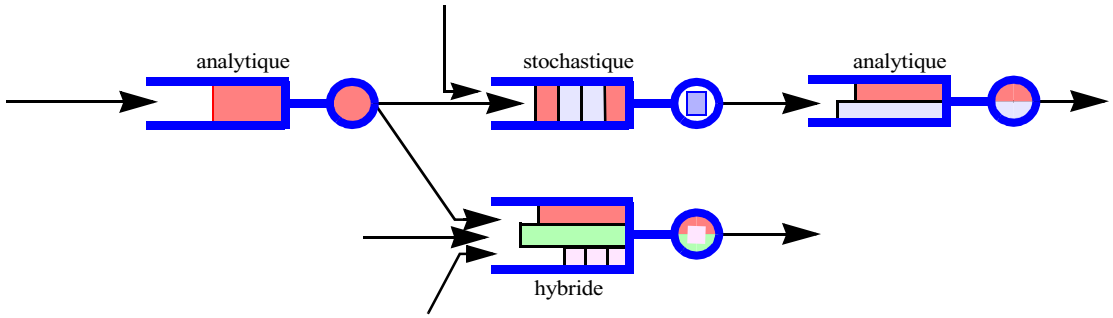
- Interconnexion de noeud analytiques et de noeuds complexes (principe 1) :
  Il s'agit du concept de la simulation hybride. Il permet une analyse fine des noeuds complexes, dont le comportement est déterminé par une simulation exacte,
- Superposition de flux événementiels et analytiques sur un noeud analytique (principe 2) :
  Cela est réalisé par une simulation de noeuds analytiques, avec, de préférence, une majorité de flux « analytiques » et une minorité de flux « paquets », dont on veut connaître plus précisément les caractéristiques dans le réseau.

V.2.4 Le simulateur hybride distribué DHS *(Distributed Hybrid Simulator)*

Le simulateur hybride distribué est une application de simulation de réseaux de files d'attente conçue et développée dans le cadre de cette thèse. Cette application fait suite au projet Esprit $STAR_1$. DHS est un simulateur général de réseaux de files d'attente interconnectant des noeuds simulés analytiquement et des noeuds stochastiques. Les modèles différentiels de files d'attente décrits au chapitre II sont utilisés pour évaluer le comportement de chaque noeud analytique, et des simulations de Monte-Carlo sont utilisées pour évaluer certains noeuds dont le modèle différentiel n'est pas connu. Enfin, le concept de la simulation hybride y est intégré de façon à rigoureusement combiner ces deux types de simulation.

La simulation de Monte-Carlo consiste à générer, en tirant des événements aléatoirement, un ensemble de trajectoires possibles du système considéré. Le comportement transitoire de ce système peut alors être déduit de statistiques sur les trajectoires générées. Ce type de simulation est très lourd à mener, tant au niveau occupation mémoire que temps de calcul. Il est cependant intéressant de noter que l'éventail des modèles simulables et leur précision sont sans limite (au prix d'un temps de calcul souvent déraisonnable). En particulier tous les mécanismes des interfaces des réseaux IP décrits au chapitre I sont modélisables et simulables.

Figure 73: **Le concept de simulation hybride**



Pour faciliter la construction du réseau à simuler, les objets manipulés par DHS sont interconnectables, c'est à dire que les échanges entre noeuds (analytique vers événementiel, événementiel vers analytique) au sens de l'interconnexion hybride sont possibles. Il est également possible de faire circuler un flux événementiel (des arrivées de paquets) dans un noeud analytique : les trafics sont superposées et interagissent sur le service offert par le noeud.

V.2.5 La simulation hybride

V.2.5.1 Algorithme général

Le modèle global peut être résolu par l'intégration numérique d'équations différentielles. A chaque pas d'intégration les calculs suivant sont réalisés:

- Pour les noeuds analytiques, les équations différentielles sont intégrées, ou l'état stationnaire est recherché,
- Les simulations stochastiques à événements discrets sur les noeuds simulés sont réalisées pendant l'intervalle de temps correspondant,
- Les informations d'état (débit, variance des temps d'inter-arrivée, …) sont échangées de noeud en noeud en fonction des paramètres d'interconnexion (coefficient de routage).

De cette façon, la décomposition du réseau en noeuds analytiques et en noeuds simulés permet de réduire les temps de calcul tout en préservant le caractère général de l'approche de simulation.

Décrivons maintenant la façon dont la simulation d'un noeud complexe est réalisée sur un intervalle de temps Δt. Pour cela, notons:

- S l'ensemble des trajectoires aléatoires (réalisations indépendantes) générées pour ce noeud complexe. Il convient de noter que ces trajectoires ne sont pas réinitialisées au début de chaque pas de temps Δt (*i.e. à chaque itération*),
- K l'ensemble des noeuds constituant le noeud complexe. En effet, un noeud simulé peut en fait être un sous-réseau regroupant plusieurs noeuds et plusieurs liens et à l'intérieur duquel les paquets sont traités et routés en fonction de politiques données,
- I l'ensemble des liens d'entrée du noeud complexe. Ces liens d'entrée sont soit des liens venant d'autres noeuds, soit des liens rattachés à des sources de trafic,
- O l'ensemble des liens de sortie du noeud complexe. Ces liens de sortie transportent tous les flux traités à l'intérieur du noeud complexe,
- F l'ensemble des flots entrant du noeud complexe.

A l'itération t, l'objectif est d'estimer les quantités suivantes:

- $d_o^f(t)$: débit du trafic du flot f sur le lien de sortie o à l'instant t,
- $v_o^f(t)$: variance du processus de départ du flot f sur le lien de sortie o à l'instant t,
- $X_k^f(t)$: nombre moyen de paquets du flux f dans le noeud k à l'instant t.

Pour cela, la simulation évalue pour chaque trajectoire aléatoire s :

- $D_{o,s}^{f}(t-\Delta t,t)$: nombre de paquets du flot f transmis sur le lien de sortie o entre t-Δt et t pour la trajectoire s,
- $X_{k,s}^{f}(t)$: nombre de paquets du flux f au noeud k à l'instant t pour la trajectoire s.

L'itération t consiste alors à réaliser les étapes suivantes :

- **Etape 1.** Réception des paramètres de trafic (taux et variance des arrivées) de tous les noeuds voisins connectés au noeud complexe par un lien d'entrée.
- **Etape 2.** Pour chaque trajectoire aléatoire s :

- **Etape 2.a.** Générer de nouveaux événements d'arrivée de paquets durant Δt pour chaque flot *f* en fonction de la distribution des inter-arrivées de ce flux. Selon la distribution (loi de Poisson, loi déterministe, ou Gamma), on réalise une approximation à un ou deux moments.
- **Etape 2.b.** Traiter tous les événements du noeud complexe en partant de l'état $X_{k,s}^{f}(t-\Delta t)$, donné par l'itération précédente, jusqu'à l'instant *t*. Pendant l'intervalle de temps $(t-\Delta t,t)$, les paquets sont comptés sur les liens sortants pour évaluer les quantités $D_{o,s}^{f}(t-\Delta t,t)$.
- **Etape 2.c.** Mémoriser $X_{k,s}^{f}(t)$.

- **Etape 3.** Estimer les deux premiers moments de $d_o^f(t)$ (permettant de connaître $v_o^f(t)$) pour chaque lien de sortie *o* du noeud complexe ainsi que l'état moyen $X_k^f(t)$ de chaque élément *k* à l'intérieur du noeud complexe.
- **Etape 4.** Envoyer les valeurs estimées de $d_o^f(t)$ et $v_o^f(t)$ aux noeuds voisins connectés au noeud complexe par un lien de sortie.

Dans une simulation du domaine transitoire, l'intervalle de temps Δt dépend fortement du plus petit temps de traitement dans le réseau. En même temps, Δt doit être suffisamment petit pour pouvoir correctement capturer le comportement transitoire. Une analyse hors ligne des composants du réseau et des intensités des trafics des flux permet de choisir une valeur appropriée. En général, les équations différentielles ne sont pas très sensibles à ce paramètre et la principale difficulté provient des noeuds complexes simulés.
De la même façon, le nombre de trajectoires aléatoires dépend des intensités et des variances des trafics d'entrée. Nous travaillons actuellement sur de nouvelles techniques d'estimation permettant de réduire ce nombre de trajectoires.

En revanche, lors de la recherche de l'état stationnaire, une seule trajectoire (simulation de type *long-run*) est nécessaire pour les noeuds événementiels, et Δt n'a aucune influence sur les équations d'état des noeuds analytiques.

V.2.5.2 Propagation des flots événementiels dans des noeuds analytiques

Besoin

Les caractéristiques précises des flux (délais et gigue de bout en bout, variance des inter-arrivées en sortie de réseau) peuvent être calculées par la simulation événementielle. Les modèles analytiques permettent, dans une certaine mesure, d'obtenir ce type d'information. Certains modèles peuvent cependant ne pas donner de résultats plus détaillés que le premier moment des valeurs manipulés (la moyenne).

La simulation événementielle complète d'un réseau n'étant pas souhaitable à cause de sa lourdeur, et le niveau de détail des résultats d'une simulation analytique ou hybride pouvant ne pas être suffisant à cause des modèles analytiques employés, nous proposons ici une manière de simuler de façon événementielle uniquement les flots dont on désire évaluer précisément la qualité de service, et utiliser les modèles analytiques, plus rapides, pour évaluer le reste de la charge du réseau. Ce principe de simulation est appelé, dans la suite, le principe de superposition hybride (principe 2).

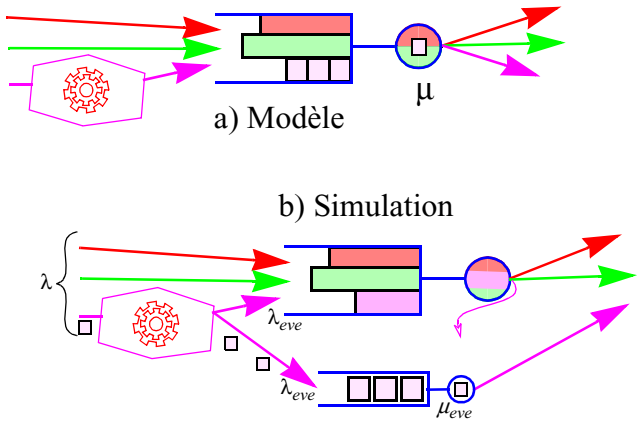
Principe

Le simulateur DHS permet de propager certains flots de façon complètement événementielle à travers des noeuds analytiques, permettant ainsi l'utilisation de tous les mécanismes nécessaires pour le traitement de ces flots (figure 74a) : en entrée de file, les flots événementiels peuvent subir les traitements des mécanismes de régulation habituels dans les réseaux à qualité de service (décrits au chapitre I et au paragraphe V.5).

Lorsque les paquets d'un flot événementiel traversent une file, ils subissent un temps d'attente qui dépend de la charge induite par les autres flots (analytiques ou événementiels). Ces flots subissent également l'influence du flot considéré.

Approximation de l'influence mutuelle

Figure 74: **Propagation de flots événementiels**



La figure 74a illustre deux flots analytiques et un flot événementiel en entrée d'une file d'attente analytique. L'approche actuelle permettant de calculer l'influence mutuelle de l'ensemble des flots est la suivante.

Le noeud lui-même est géré exactement comme décrit dans l'algorithme général au paragraphe V.2.5.1, tout se passe comme si ce flot événementiel était transformé en un flot analytique à l'entrée du noeud. Les premiers moments (débit, variance des arrivées) du flot entrant sont estimés pendant Δt, et le flux est traité dans le noeud comme un flux analytique.

Cependant, à la différence de l'interconnexion hybride 1 (figure 74b),

- Les moments analytiques du « débit » sortant ne sont pas envoyés aux noeuds voisins,
- Les paquets ne sont pas détruits, mais stockés dans une file de type PAPS (*FIFO*). La distribution de service des paquets de la file est la même que celle du noeud analytique. Ses paramètres sont recalculés à chaque pas d'intégration analytique.

V.2.5.3 Calcul du taux de service moyen du service hybride

L'idée est que le flot événementiel soit gêné par les autres flots. La statistique du flot événementiel entrant permet de calculer la charge et le débit sortant moyen du flot analytique équivalent. Une méthode possible consiste à donner au flot événementiel le débit de sortie moyen du flot analytique équivalent calculé par le modèle analytique. Le flot événementiel est alors servi au même rythme, en moyenne, que son équivalent analytique.

En première approximation, nous considérerons que le système est une file M/M/1. Le taux de service exponentiel μ_{eve} (figure 74b) est donné par (95) :

$$\mu_{eve} = \mu - (\lambda_{total} - \lambda_{eve}) \qquad (95)$$

μ_{eve} représente la proportion de service restant dans la file M/M/1 en tenant compte du service des autres flots. En effet, appelons X_{eve} la charge dûe à la source événementielle (transformée en

analytique) et X_{ana} la charge dûe aux autres sources analytiques.

Nous savons que

$$X_{eve} = \frac{\rho_{eve}}{1-\rho} \qquad \text{avec} \qquad \rho = \frac{\lambda}{\mu} = \frac{\lambda_{ana}+\lambda_{eve}}{\mu} \qquad \rho_{eve} = \frac{\lambda_{eve}}{\mu} \tag{96}$$

Cela peut s'écrire aussi :

$$X_{eve} = \frac{\rho_{eq}}{1-\rho_{eq}} \qquad \text{avec} \qquad \rho_{eq} = \frac{\lambda_{eve}}{\mu-\lambda_{ana}} \tag{97}$$

Cela revient à dire que X_{eve} est la charge d'une file M/M/1 ayant une source de débit λ_{eve} et un taux de service $\mu-\lambda_{ana}$.

Ce principe général peut être adapté en fonction du modèle du noeud analytique utilisé.

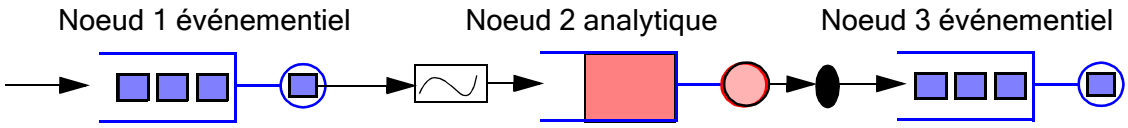
V.2.6 Exemples

Dans la suite, nous présentons deux exemples simples illustrant les deux principes de simulation hybride. Ces exemples indiquent en particulier comment sont transformés les flots (analytique vers événementiels et événementiel vers analytique) et ils sont illustrés par des courbes correspondant aux régimes transitoires et stationnaires.

V.2.6.1 Interconnexion hybride

La figure 75 représente le réseau sur lequel nous nous appuyons. Il est constitué de trois noeuds et d'une source. Les premier et troisième noeud sont événementiels et le second est analytique. Cet exemple explicite la conversion du flot sortant événementiel en un débit analytique entrant, et la transformation du débit analytique sortant en un flux de clients.

Figure 75: **Réseau de test pour l'interconnexion hybride**



Régime transitoire

En régime transitoire, le temps est discrétisé en pas de durées Δt. Cette discrétisation est sans objet en ce qui concerne la simulation événementielle des premier et troisième noeuds. Ce pas correspondant au pas d'intégration des équations différentielles. A chaque pas est également réalisé une statistique sur les clients sortant du premier noeud, permettant d'évaluer la moyenne et les moments d'ordre supérieur des inter-arrivées. Soit N le nombre de clients arrivant dans le pas délimité par les instants t et $t+\Delta t$. La figure 76 illustre les dates d'arrivées d_i des clients dans l'intervalle de temps. Le débit $\lambda(t...t+\Delta t)$ (noté λ dans la suite) offert au noeud analytique est constant pendant la durée Δt est donné par l'équation (98) et correspond à la statistique du pas précédent.

$$\frac{1}{\lambda} = m_1 = \frac{1}{N}\sum_{i=1}^{N} d_i - d_{i-1} = \frac{1}{N}(d_N - d_0) \tag{98}$$

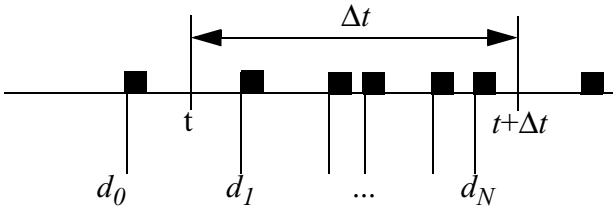
Les moments d'ordres supérieurs peuvent être également calculés (99) :

$$m_n = \frac{1}{N}\sum_{i=1}^{N}(d_i - d_{i-1})^n \qquad (99)$$

et en particulier la variance des inter-arrivées est donnée par (100) :

$$\sigma^2 = m_2 - m_1^2 \qquad (100)$$

Figure 76: **Arrivées de clients dans un pas de durée** Δt



La largeur du pas Δt dépend fortement des caractéristiques de la source. Un pas trop petit ne permet pas de faire une statistique correcte des arrivées de clients (trop peu de paquets par intervalle). Un pas trop grand ne serait pas adapté à l'intégration numérique des équations différentielle. En pratique, il doit être du même ordre de grandeur que celui de la moyenne des temps d'inter-arrivée.

Selon le modèle analytique employé pour le noeud 2, les paramètres d'entrée sont λ seul ou λ et σ^2. Le modèle employé dans notre exemple est M/M/1. La distribution d'entrée est exponentielle et ne nécessite que le débit moyen d'entrée λ. L'équation différentielle de charge du noeud 2 est rappelée (101) :

$$\begin{aligned} \frac{d}{dt}X(t) &= TE(t) - TS(t) \qquad X(t_0) \text{ donné} \\ TE(t) &= \lambda \\ TS(t) &\approx \mu\frac{X(t)}{1+X(t)} \end{aligned} \qquad (101)$$

La distribution des temps inter-clients générés en entrée du noeud 3 doit suivre la distribution des temps d'inter-sortie du noeud 2. Ces temps sont distribués exponentiellement conformément aux caractéristiques de sortie du modèle M/M/1. La valeur moyenne de ces temps est l'inverse du débit moyen de sortie $TS(t)$ du noeud 2. Ainsi sur chacune des trajectoires, un générateur aléatoire de distribution exponentielle et de paramètre $TS(t)$ réévalué tous les Δt génère des clients pour le noeud 3.

Régime stationnaire

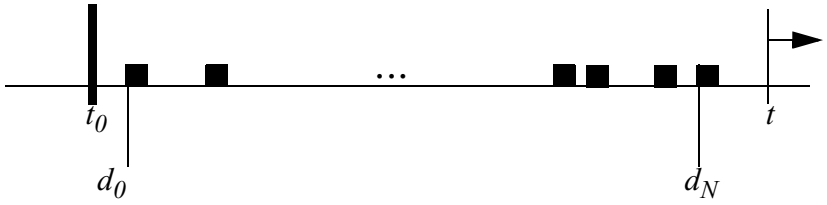
Le régime stationnaire pour les modèles événementiels est obtenu par une seule simulation (une seule trajectoire) de type «long-run». Contrairement au régime transitoire pour lequel les statistiques sont réévaluées à chaque pas Δt sur l'ensemble des trajectoires, les statistiques sont calculées sur toute la période d'une unique trajectoire de la grandeur évaluée (débit entrant, sortant, et charge). La figure 77 illustre la fenêtre de mesure (t_0 est constant, N et t sont croissants) et les formules (98) à (100) restent valides.

Le calcul du régime stationnaire pour un modèle analytique se résume à résoudre l'équation $TE(\infty)=TS(\infty)$. De cette résolution peut être calculé $X(\infty)$. La résolution de cette équation donne immédiatement le régime stationnaire du noeud si l'entrée $TE(\infty)$ était vraiment stationnaire. En

réalité nous faisons à chaque évaluation l'approximation (102).

$$TE(\infty) \approx \overline{TE(t)} = \frac{1}{t - t_0} \int_{t_0}^{t} TE(u)du \qquad (102)$$

Figure 77: **Fenêtre de mesure en régime stationnaire**



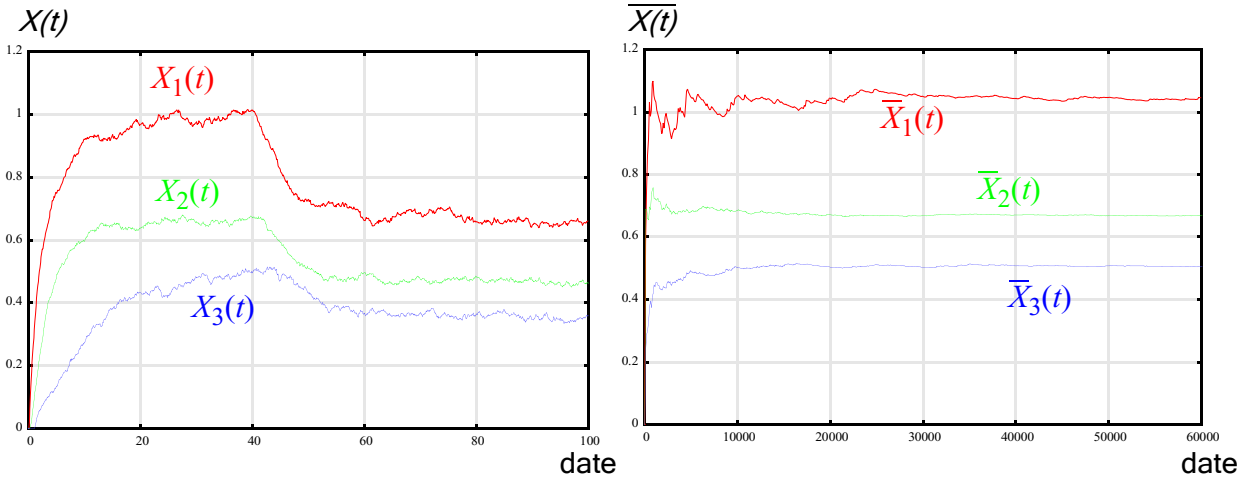
Exemples numériques

Le réseau de la figure 75 est constitué de trois noeuds M/M/1. Le débit d'entrée dans le premier noeud est 0.5 clients/s. Les taux de services sont respectivement 1, 1.25 et 1.5 clients/s (noeuds 1, 2 et 3).

La figure 78a représente le régime transitoire (les charges sont nulles au départ). Les courbes de charge des noeuds 1 et 3 sont la statistique de 5000 trajectoires réalisées en parallèle. Les paramètres d'entrée du noeud 2 sont évalués à chaque pas Δt de 0.1s et résultent de la statistique des 5000 trajectoires également. L'équation différentielle de charge du noeud 2 est intégrée également tous les 0.1s en utilisant la méthode d'Euler. Les paramètres des générateurs de chaque trajectoire en entrée du noeud 3 sont mis à jour tous les 0.1s.

Le calcul du régime transitoire permet d'observer le comportement du réseau face à des fluctuations de la source. Dans cet exemple, le débit moyen de la source (génération de paquets de dates d'inter-arrivées de distribution exponentielle) est 0.5 client/s entre 0 et 40s, et 0.4 clients/s à partir de 40s.

Figure 78: **Régimes stationnaires et transitoire (interconnexion hybride)**



(a) Régime transitoire (5000 trajectoires) (b) Régime stationnaire

La figure 78b représente le calcul du régime stationnaire (génération de 0.5 client/s en moyenne, distribution d'inter-arrivées exponentielle). Pour des dates petites, les statistiques du premier noeud n'ont pas convergées, ce qui donne une statistique fausse en entrée du noeud

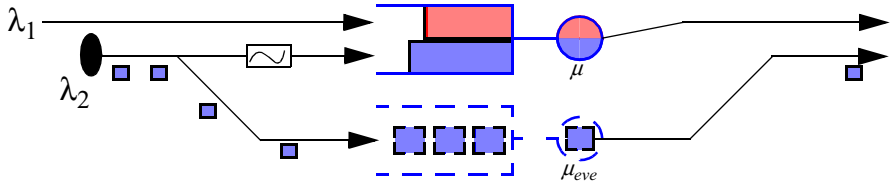
analytique, ainsi qu'au générateur de paquets en entrée du noeud 3. Le contrôle de convergence peut être réalisé à l'aide d'un algorithme de point fixe sur les débits d'entrée et les charges en chaque noeud.

V.2.6.2 Superposition hybride

Exemple 1

Dans cet exemple, nous ne simulons qu'un noeud traversé par deux sources. La première source est analytique (Poissonienne, représentée par un débit λ_1) et la seconde est événementielle (clients de distribution d'inter-arrivée exponentielle et de paramètre λ_2). Le principe de superposition hybride permet le traitement de ces deux types de flots dans la même ressource. Le système simulé est représenté par la figure 79.

Figure 79: **Réseau de test pour le principe de superposition hybride**

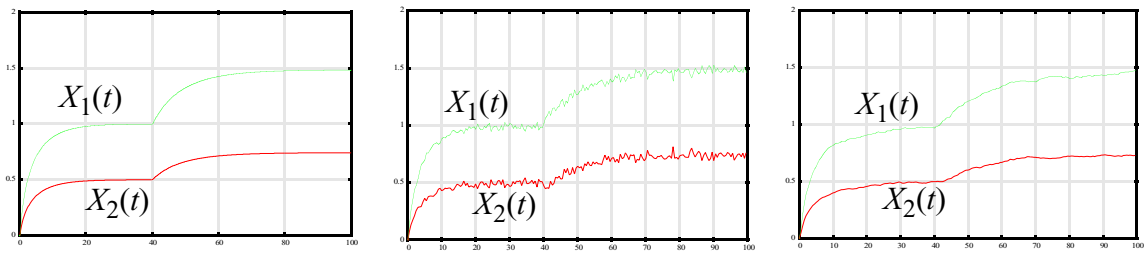


Les principes généraux de simulation analytique (intégration des équations différentielles pour le régime transitoire ou résolution analytique du régime stationnaire) et de simulation événementielle sont identiques que dans l'exemple précédent. Nous illustrerons ici l'influence mutuelle des deux flots.

Afin que la présence du flot λ_2 soit prise en compte dans le noeud analytique, les paramètres de débit du flux événementiel sont évalués à chaque pas d'intégration (régime transitoire) ou à chaque réévaluation du point fixe (régime stationnaire). La charge induite par le flux λ_1 est évaluée par les équations qui tiennent compte de λ_2 dans les calculs.

Parallèlement, les clients sont envoyés dans une file d'attente de mêmes caractéristiques que celles que le modèle analytique représente. Afin de tenir compte fictivement de la présence du flot analytique λ_1 dans le noeud événementiel, nous faisons varier le paramètre de service μ_{eve}. Nous avons proposé au paragraphe V.2.5.3 une méthode de calcul de ce taux de service dans le cas d'un modèle de file d'attente M/M/1.

Figure 80: **Simulations analytiques, événementielles et superposition hybride**



a) Simulation analytique
Flot 1 analytique
Flot 2 analytique

b) Simulation hybride (2)
Flot 1 analytique
Flot 2 événementiel

c) Simulation événementielle
Flot 1 événementiel
Flot 2 événementiel

En sortie du noeud analytique, le second flot analytique n'est pas propagé. Sont propagés le

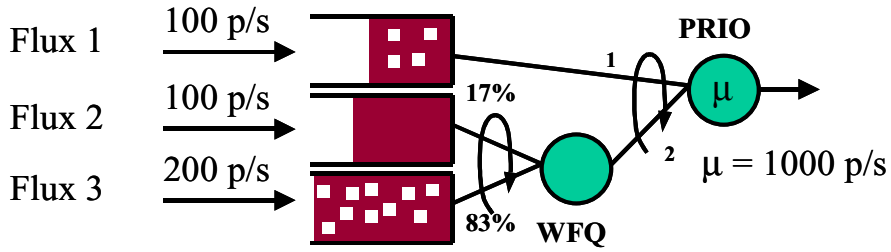
premier flot analytique (issus de la source λ_1) et les clients du second flot (issus de la source discrète).

Les courbes présentées en figure 80 représentent les charges induites par les deux flots dans le système dans le domaine transitoire. Le modèle employé est M/M/1, et à la date initiale le système est vide. Le taux de service global est 1 client/s et le taux de service μ_{eve} de la file événementielle est donné par l'équation (95). λ_1 et λ_2 valent respectivement 0.4 et 0.2 client/s. A la date t=40s, le débit moyen des deux sources est augmenté de 15%.

Exemple 2

Le système illustré en figure 81 représente trois flots injectés dans trois files d'attentes de débits respectifs 100, 100 et 200 paquets par seconde. Le taux de service global du système est de 1000 paquets par secondes. La première file est servie prioritairement, tandis que les deux files suivantes se partagent le temps de service restant, à hauteur de 17% pour la seconde et 83% pour la troisième.

Figure 81: **3 sources Poisonniennes, combinaison de priorités et GPS**



Quatre simulations de ce système ont été réalisées. La première dans laquelle les trois flots sont simulés par des événements, et dont les résultats sont représentés dans la première colonne du tableau 20. La seconde est analytique : tous les flots sont représentés par des équations. Le deux dernières sont hybrides : respectivement le flot 1 puis le flot 3 est simulé par des événements tandis que le reste est calculé numériquement par des équations. Les colonnes de résultats du tableau montrent la justesse des méthodes employées. La simulation hybride par superposition permet ainsi de simuler un ou quelques flots en utilisant des simulations événementielles de bout en bout.

Tableau 20: **3 sources poisonniennes, combinaison de priorités et GPS**

	Tout Evénementiel	Tout Analytique	Erreur relative	Hybride (flux 1)	Erreur relative	Hybride (flux 3)	Erreur relative
Charge File 1	0.144377	0.144444	**0.05%**	0.144319	**0.04%**	0.144434	**0.04%**
Charge File 2	0.181013	0.188624	**4.0%**	0.188572	**4.0%**	0.188579	**4.0%**
Charge File 3	0.342502	0.333598	**2.6%**	0.333525	**2.6%**	0.337970	**1.3%**

V.3 Structure générale du simulateur DHS

Nous allons dans un premier temps décrire les éléments principaux utilisés dans le modèle de simulation, puis nous détaillerons le fonctionnement global du simulateur.

V.3.1 Les noeuds, les flux et les flux-états

Afin de permettre la simulation hybride de tout type de réseau, le simulateur DHS manipule différents objets qui sont le noeud et les interconnexions de noeuds, le flux, et les paramètres de routage. La simulation de Monte-Carlo peut-être utilisée pour tout type de noeuds, tandis que certains d'entre eux seulement peuvent être analytiquement évalués (le nombre de modèles analytiques est limité).

• Le noeud

Le modèle de noeud utilisé dans le simulateur est un modèle général permettant de représenter tout type de files d'attentes et tout type d'ordonnancement (*scheduling*). Un intérêt majeur de ce simulateur pour les réseaux IP multiservices est d'avoir un modèle de noeud se rapprochant le plus de l'interface utilisé dans ce type de réseau. Les mécanismes de *shaping*, *policing*, et *buffer management* concernant ces réseaux sont également introduits dans le modèle. Nous verrons comment cela est réalisé dans le simulateur DHS.

• Le flux

Les caractéristiques d'un flux sont le noeud origine, le noeud destination, les paramètres de la source et la classe de service à laquelle il appartient.

Une source est soit analytique, soit un générateur de paquets. Dans chaque cas, elle est caractérisée par la distribution des temps inter-paquets et des tailles de paquets. Dans le cas analytique, ce sont les paramètres des distributions qui sont échangés de la source au premier noeud dans lequel elle est acheminée. Dans le cas événementiel, des générateurs aléatoires de distribution voulue (chapitre III) sont utilisés pour produire des dates d'arrivées et des tailles de paquets pour la simulation événementielle.

• Le *flowstate* (l'état de flux)

Pour pouvoir mesurer la charge induite par les flux dans une file d'attente, une représentation de l'état de chaque flux en chaque noeud est nécessaire. Cela permet de fournir la charge, le taux de perte, les délais et le débit de sortie de chaque flux en chaque noeud. Cette représentation est appelée « état de flux ».

Un *flowstate* est associé à un flux et un noeud traversé par ce flux. Celle-ci est mise à jour directement par les équations à chaque pas d'intégration des noeuds analytiques, ou peut être calculée par la statistique des trajectoires événementielles des noeuds simulés.

Cette statistique sur les flux événementiels est faite dans les cas suivants :

- Lorsque la sortie d'un noeud complexe (stochastique) est connectée à l'entrée d'un noeud analytique, la statistique doit être effectuée à chaque pas d'intégration du noeud analytique (intervalle Δt),
- Lorsque la recherche de l'état stationnaire d'un noeud complexe simulé est nécessaire (intervalle Δt),
- Lorsqu'une trace de l'évolution dynamique de la statistique doit être enregistrée.

V.3.2 Le routage

V.3.2.1 Structure du routage

Le routage est modélisé par un poids $r^f_{i,j}$ affecté au flux *f*, dans le noeud *i*, sur le lien le connectant au noeud *j*. Pour un flux donné, la somme des poids de tous les liens de sortie vaut 1. Le partage de charge sur les chemins est ainsi possible.
Ce type de représentation est suffisamment général pour permettre :

- Le routage statique par flux :
  Il s'agit du routage à la fois le plus simple et le plus général. Il est calculé avant, et peut être mis à jour pendant la simulation. Il s'agit de positionner les coefficients $r^f_{i,j}$ pour que chaque flux *f* suive un chemin prédéterminé. Ce routage est le plus fin que l'on puisse avoir dans le simulateur (chaque flux peut être partagé sur plusieurs routes différentes),
- Le routage par destination (utilisé dans Internet) :
  En utilisant le routage statique par flux, il suffit d'affecter des coefficients égaux à 1 aux liens de sortie choisis, et 0 aux autres. Ainsi le routage IP peut être modélisé.

Le routage dynamique est possible, car les coefficients de partage peuvent être modifiés par un algorithme extérieur (précalculé ou non).

Sur un réseau avec beaucoup de trafic, le routage dynamique par flux peut engendrer une structure de données assez lourde. Une structure de données de type routage IP par classe de service est également possible. L'utilisation de ce type de routage a le même effet sur le simulateur que dans le réseau Internet : les tailles des tables de routage en chaque noeud diminuent considérablement par rapport aux tailles des structures de données nécessaires pour le routage par flux.

V.3.2.2 Routage de flux analytiques et de flux événementiels

Le coefficient de routage $r^f_{i,j}$ correspond à la proportion du flux *f* partant du noeud *i* vers le noeud *j*. Ces coefficients sont utilisés de la manière suivante:

- Si le type de représentation du flux est analytique, le débit de sortie de ce flux est multiplié par le coefficient $r^f_{i,j}$ puis est injecté dans le noeud *j*.
- Si le type représentation du flux utilisé dans le noeud *i* est événementiel (paquet), il faut choisir de façon aléatoire le prochain noeud à atteindre en fonction des paramètres connus $r^f_{i,k}$ (*k* étant pris dans l'ensemble des noeuds connectés à *i*).
  Cela se fait de la manière suivante. Soit $S_i=\{j_0, j_1, \ldots j_{s(i)-1}\}$ les *s(i)* noeuds connectés en sortie du noeud *i*, et les coefficients de routage r^f_{i,j_k} (*k* dans [*0..s(i)-1*]) du flux *f* pour l'ensemble des noeud j_k appartenant à S_i. Si un paquet doit être transmis, une variable aléatoire uniforme *u* entre 0 et 1 est tirée, et le noeud j_p est choisi pour router le paquet, *p* vérifiant :

$$p = 0 \quad et \quad 0 \le u < r_{ij_0}$$

ou

$$0 < p < s(i) \quad et \quad \sum_{k=0}^{p-1} r_{ij_k} \le u < \sum_{k=0}^{p} r_{ij_k}$$

V.3.2.3 Routage dynamique

De manière général, le routage dynamique permet d'optimiser les performances d'un réseau. Il permet également la modification des tables de routage à n'importe quel moment afin de contourner les pannes. Dans DHS, cela peut se faire de plusieurs manières.

- Les tables de routage peuvent être précalculées à l'avance. Une table de routage initiale au démarrage de la simulation, et éventuellement une ou plusieurs tables supplémentaires correspondant à des scenarii de panne prévues à des dates déterminées.
- L'algorithme de routage permettant de construire les tables peut être lui aussi simulé par DHS. Les tables seront alors automatiquement recalculées en cas de «mise en panne programmée» d'un composant réseau.
  Un tel algorithme peut être exécuté de façon centralisée (extérieur au réseau), ou de façon complètement intégrée à la simulation : des paquets spéciaux seront alors utilisés pour permettre aux algorithmes résidant dans les routeurs de s'échanger des informations.

DHS permet ainsi l'étude du temps de rétablissement de la connectivité en cas de panne, en fonction du temps de propagation des informations de contrôle d'un algorithme de routage particulier dans un réseau congestionné. DHS peut également servir de plate-forme d'expérimentation sur la mise au point d'un algorithme de routage devant dynamiquement modifier les tables de routage pour optimiser la qualité de service des flots (délai minimum pour les uns, perte minimum pour les autres par exemple).

V.3.3 Les événements

Les événements sont au coeur de la simulation, que les éléments du réseaux soient analytiques ou événementiels. Chaque action menée par le simulateur est représentée par un événement particulier.

La boucle principale du simulateur consiste à gérer un échéancier. Cet échéancier est une liste d'événements classés par date d'action. Voici une liste non exhaustive comprenant les principaux événements :

- *Intégration des équations différentielles*
  Cet événement est décrit au paragraphe V.3.5.
- *Recherche de l'état stationnaire*
  Voir le paragraphe V.3.5.
- *Appel à une fonction utilisateur* (*Callback*)
  Cet événement récurrent est utile si l'on veut garder une trace de la réponse dynamique du réseau. Il provoque l'appel à une fonction utilisateur qui a accès à l'état du réseau.
- *Création de paquet*
  Les paquets issus des générateurs ont besoin d'être créés à des dates bien déterminées. Cet événement crée un paquet selon les caractéristiques de la source, l'insère dans le noeud et calcule la date d'arrivée du prochain paquet en utilisant le générateur de la source.
  Les paquets sont également créés lors d'une transition hybride d'un noeud analytique vers un noeud événementiel. C'est là aussi un générateur correspondant à la distribution de sortie du noeud analytique qui créé les inter-arrivées des paquets. Les paramètres d'entrée du générateur sont donnés par les paramètres de sortie du noeud analytique (débit moyen, variance si nécessaire).

- *Service du paquet*
  Lorsque qu'un paquet entre en service, une date de sortie est calculée (en fonction de plusieurs paramètres du noeud, du paquet, et du flux d'appartenance). Lorsque cet événement survient, le paquet est extrait du noeud, routé sur le lien de sortie, et si nécessaire, un autre paquet entre en service.
- *Arrêt de la simulation*.

Les événements présentés ici suffisent pour faire une simulation simple, afin d'observer le comportement dynamique ou pour rechercher le régime stationnaire du réseau.
Un nouveau type d'événement peut aisément être créé à des fins particulières. Par exemple le module de gestion des processus de naissance et de mort régissant le nombre de source actives (III.2) d'un trafic est géré par un événement particulier servant à la création d'une nouvelle source.

Grâce à cette échéancier d'événements, les simulations des noeuds analytiques et des noeuds complexes événementiels sont menées indépendamment. Les opérations d'échange des paramètres d'entrée et de sortie aux interfaces hybrides sont effectuées lors de chaque pas d'intégration ou d'itération du point fixe pour la recherche de l'état stationnaire des équations.

V.3.4 La simulation événementielle

V.3.4.1 Principe

La simulation événementielle est une simulation de Monte-Carlo. Chaque élément du réseau est dupliqué un certain nombre de fois afin de réaliser des simulations indépendantes, appelée aussi trajectoires. L'évaluation quantitative des grandeurs (nombre de paquets présents dans le système, délais, probabilité de perte) est effectué en calculant la moyenne (et la variance si nécessaire) sur l'ensemble des trajectoires. Pratiquement, cela implique que toutes les trajectoires sont menées en parallèle. Leur différence se situe uniquement lors de la génération des variables aléatoires. Les générateurs associés à chaque trajectoire ont des caractéristiques identiques, mais fournissent des réalisations différentes pour chacune d'elles afin d'obtenir une statistique correcte de l'ensemble.

- Lorsqu'un flot ne traverse que des noeuds simulés événementiellement, ou lorsqu'il est marqué comme devant rester événementiel même sur les noeuds analytiques (superposition hybride), les trajectoires du flot existent depuis le noeud d'entrée jusqu'au noeud de sortie.
- Lors d'une transition hybride de type passage d'un noeud à événements discrets vers un noeud analytique, la trajectoire s'arrête sur le dernier noeud à événements discrets, puis la statistique du débit de sortie sur ce noeud fournit un résultat pour les modèles fluides de noeuds sur la suite du chemin.
- Lors d'une transition hybride de type passage d'un noeud analytique vers un noeud à événements discrets, de nouvelles trajectoires sont créées à partir du premier noeud événementiel, chacune avec des générateurs de paquets ayant des caractéristiques identiques (mais générant des réalisations différentes) utilisant les paramètres de sortie du noeud analytique mis à jour à chaque pas d'intégration Δt (voir V.2.5).

Bien que l'ensemble des événements soit classé et stocké dans un unique échéancier global, la simulation événementielle peut être vue comme un ensemble d'algorithmes indépendants et décentralisés, ayant une cohérence globale grâce à leur interconnexion réalisée par le routage.

V.3.4.2 Le paquet

Le paquet est à la base de la simulation événementielle. Les paquets du simulateur DHS correspondent aux paquets générés par une interface réelle de niveau 2 (couche réseau), et également aux clients des files d'attente utilisées dans la théorie différentielle de trafic.

Les paquets basiques

Au minimum, les informations transportées par un paquet sont :

• Le flux d'appartenance,

• Sa classe de service,

• Sa destination,

• La taille totale des données transportées (entête comprise),

• Le numéro de la trajectoire à laquelle appartient le paquet,

• La date de création du paquet,

• Le noeud et la file dans lesquels il se trouve.

Un paquet de ce type peut être créé par la plus simple des sources (un générateur simple), par exemple de distribution d'inter-arrivées issue d'un processus de poisson. Avant d'arriver à destination, ces paquets peuvent être perdus sur un noeud dont les files sont saturées ou parce qu'un algorithme particulier l'a décidé (*RED, Token Bucket*). La classe de service d'un paquet sert à décider la file dans laquelle le paquet doit être inséré (classification) en entrée de chaque noeud.

Les paquets spéciaux

Les paquets peuvent également être issus d'une source spéciale, comme un protocole de transport (par exemple TCP), ou un protocole de routage (par exemple OSPF). Dans ce cas des informations spécifiques lui sont ajoutées pour le fonctionnement des algorithmes associés au protocole.
Par exemple le protocole OSPF, qui a été en partie intégré dans le simulateur, a besoin d'échanger des informations sur les *link-states* entre routeurs. Ces informations sont transportées dans ces paquets spéciaux. De même, les paquets issus du protocole TCP contiennent le même type d'information que celles qui sont contenues dans l'entête TCP (les champs *seq*, *ack*, *rwnd*...).

Pour des raisons évidentes, ces paquets ne peuvent pas être transformés en un flot analytique lors d'une simulation hybride, car ils perdraient toute leur information utile pour le protocole associé.

Les paquets spéciaux subissent les même traitements que les paquets classiques, mais sont transférés à une fonction associée au protocole avant d'être détruits une fois qu'ils sont arrivés à destination. Le protocole OSPF pourra exploiter les données transportées, tandis que le protocole TCP pourra renvoyer d'autres paquets spéciaux, par exemple de type *TCP-ack*, vers la source. La fonction de traitement associée à chaque type de paquet spécial est intégrée dans la définition du paquet lui-même, ce qui laisse une grande latitude sur l'étude de nouveaux protocoles et de leurs effets sur les performances d'un réseau.

V.3.5 La simulation analytique

Chaque modèle de noeud analytique possède deux modes de calculs. Le calcul de l'état stationnaire d'une part, et le calcul de l'évolution dynamique d'autre part. Ces calculs sont ré-effectués à la date de chaque événement analytique.

- *Stationnaire analytique*
  Lors de chaque événement de type « recherche de l'état stationnaire », le simulateur itère l'algorithme du point fixe : il calcule le nouvel état des noeuds analytiques (charge, perte, délai) en fonction de leurs paramètres d'entrée. Ceci se fait au moyen d'équations différentielles associées aux différents modèles des noeuds.
  La date de l'événement importe peu dans une simulation stationnaire qui comporte uniquement des noeuds et des flux analytiques.
- *Transitoire analytique*
  Cet événement est régulier, à chaque pas de temps Δt, et provoque l'intégration de toutes les équations entre les dates t-Δt et t. Une fois que tous les noeuds sont intégrés, les nouvelles valeurs (charge et débit sortant) sont mises à jour pour toute la durée du pas Δt suivant. Ces valeurs permettent à n'importe quel instant de calculer le délai de bout en bout (en utilisant la formule de Little) et de paramétrer les générateurs de paquets pour les flux ou les noeuds événementiels connectés en sortie d'un noeud analytique.

V.4 Autres considérations sur la simulation

V.4.1 Statistiques

Le simulateur étant un outil d'évaluation de performances d'un réseau à commutation de paquets, il convient de générer des statistiques à la fin de la simulation. Selon le type de simulation effectué, deux types de rapports peuvent être générés : le rapport stationnaire et le rapport transitoire.

De plus, l'analyse globale du réseau, en cours ou en fin de simulation permet de calculer le délai, la gigue et la probabilité de perte de bout-en-bout pour chaque flot. Ce rapport permet de vérifier à posteriori si la qualité de service requise pour certains flots est atteinte.

D'autres grandeurs peuvent être calculées comme l'écart-type des dates de départ et des dates d'arrivée des paquets pour un flot particulier. L'évaluation empirique d'une distribution d'arrivées est également utile pour l'étude de nouveaux modèles analytiques. Cette dernière fonctionnalité est décrite au paragraphe III.3.1.

V.4.1.1 Rapport Stationnaire

Le rapport stationnaire peut être généré en fin de simulation. Il contient les caractéristiques de chaque élément :

- Caractéristiques des flots (voir ci dessous en V.4.1.3),
- Charge moyenne par interface, par file de l'interface, ou pour chaque flot, en paquets et en octets,

• Pour les flots événementiels, éventuellement des distributions (ou fonction de répartition / CDF) sous forme de données d'une courbe. Pour ces données en particulier, il convient d'indiquer quelle grandeur mesurer avant le début de la simulation, car cela peut consommer beaucoup d'espace mémoire.

V.4.1.2 Rapport Transitoire

Le rapport transitoire contient le même type de données que le rapport stationnaire. Il faut cependant indiquer les éléments du réseau devant être tracés (ainsi que le pas de mesure) pour ne pas obtenir une surcharge de données.

V.4.1.3 Rapport de qualité de service sur les flots

Une information primordiale dans l'évaluation des performances d'un réseau est le rapport de performances des flots de bout-en-bout. Nous décrivons ici comment sont évalués les délais, les gigues et les probabilités de perte des flots sur le réseau. En cas de routage par partage de charge, les grandeurs sur le chemin aval (à chaque bifurcation) sont pondérées par le coefficient de partage.

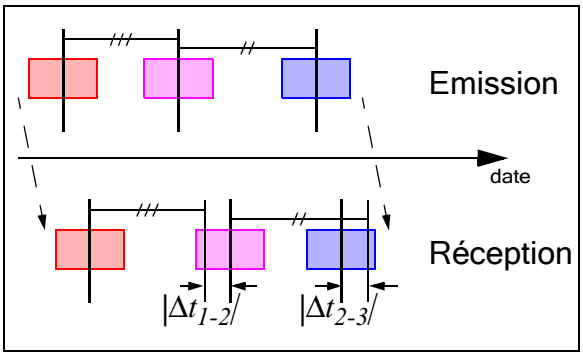
Délai

Le délai moyen de bout-en-bout d'un flot peut se calculer de plusieurs manières, selon le type de simulation du flot.

Si le flot est événementiel de bout en bout, un calcul statistique sur la différence de la date d'arrivée et de la date de départ des paquets est effectué. Cela permet d'obtenir le délai moyen de transit, ainsi que l'écart type des temps d'inter-départs et d'inter-arrivées.

Pour obtenir la gigue introduite par le réseau dans la distribution des paquets injectés par la source, la différence des dates de générations entre deux paquets successifs est insérée dans chaque paquet. Cette différence est réévaluée dans le noeud destinataire du flot permettant ainsi d'obtenir la gigue en mesurant la différence en valeur absolue de ces deux valeurs. Ces différences sont représentées par $|\Delta t_{1\text{-}2}|$ et $|\Delta t_{2\text{-}3}|$ sur la figure 82.

Figure 82: **Calcul de la gigue**



Si le flot n'est pas événementiel, le calcul du délai de bout-en-bout du flot est effectué en sommant les différents délais introduits dans le flot par les différents mécanismes. Dans chaque noeud, la formule de Little est utilisée pour évaluer le temps de séjour dans la file et le serveur, puis sont ajoutés d'autres paramètres comme le temps moyen pris par la fonction de routage, et le délai éventuel introduit par la liaison entre deux noeuds successifs.

Probabilité de perte

La probabilité de perte est aisément calculable pour un flot complètement événementiel. Concernant les flots analytiques de bout-en-bout ou sur des parties du chemin, la probabilité de passage est évaluée en chaque noeud du réseau. En faisant l'hypothèse classique de l'indépendance

des probabilités de blocage, les probabilités de passage sont multipliées afin d'obtenir la probabilité de passage globale.

V.5 Modélisation des réseaux IP Multi-Services et MPLS

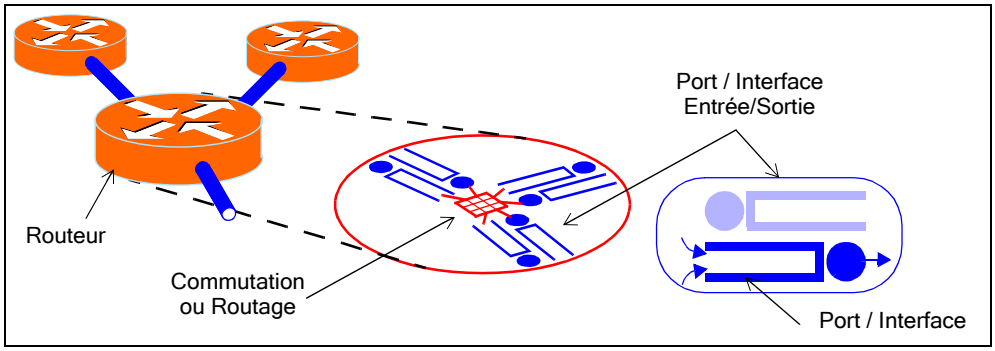
V.5.1 Analogie avec les réseaux de files d'attente

Les composant d'un réseau IP sont des machines (stations de travail, répéteurs multiports (*hubs*), commutateurs (*switches*), routeurs (*routers*)) et un ensemble de protocoles permettant la création de flux de données et leur routage à travers le réseau ainsi créé. D'un point de vue réseau, les composants principaux sont les interfaces. Une « machine » avec plusieurs interfaces peut être assimilée à une forme plus ou moins complexe de routeur (figure 83).

Le routeur lui-même introduit principalement une fonction de routage ou de commutation et un délai de traitement. Le délai de traitement peut être une constante ou une fonction complexe dépendant de plusieurs paramètres (taille des tables de routage, nombre de flux traversant l'interface, débit moyen de l'interface). Cette fonction varie de noeud en noeud, et est déterminée grâce à la métrologie. Elle permet d'ajuster l'imprécision d'un modèle à la complexité d'un routeur réel dont les spécifications ne sont pas publiées.

Une interface est un port de connexion bi-directionnel permettant d'envoyer et de recevoir des données sous forme de paquets. Un routeur est un ensemble d'interfaces muni d'une fonction de routage (IP) ou de commutation (MPLS).

Figure 83: **File d'attente dans les réseau IP.**



Un port d'entrée-sortie d'un routeur IP peut alors être modélisé par deux « files d'attente » (pouvant être complexes, figure 84), une en entrée et une en sortie. La notion de lien dans les réseaux de communication à commutation de paquets est associée à l'interface de sortie. Celle-ci est essentiellement composée d'une ou de plusieurs files d'attentes, de mécanismes de mises en forme ou de police (*shaping* et *policing* par des *token* et *leacky buckets*), de gestion de tampon (de type *WRED*), d'ordonnancement (*scheduling* de type GPS ou Round-Robin et / ou priorité, voir I.5.2), et d'un serveur avec ses paramètres. Les paramètres de service sont en général une bande passante déterministe exprimée en bits/s ou octets/s.

V.5.2 Le modèle d'interface

Le modèle d'interface unidirectionnel présenté en figure 84 est général et permet de modéliser la plupart des mécanismes que l'on rencontre sur les routeurs réels. Il peut être simulé

événementiellement ou analytiquement par DHS. Les mécanismes de *bucket* et *RED* n'ont cependant pas encore de modèle analytique exploitable par DHS.

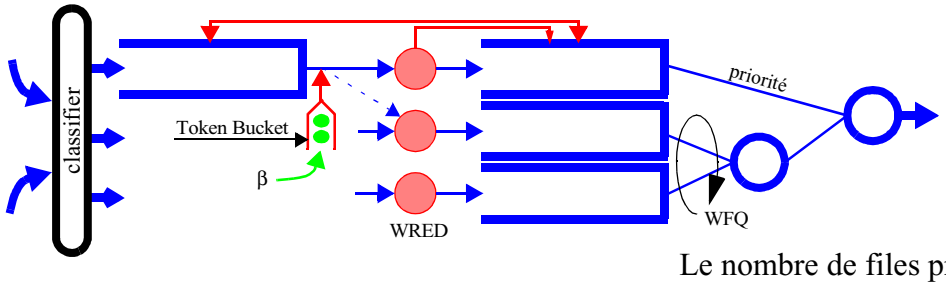
Détaillons le fonctionnement de ces mécanismes dans le simulateur DHS :

• *Classifier*
Un certains nombre de files d'attente existent dans l'interface. Les flux entrant sont regroupés par groupe de classes de service et sont dirigés dans la file d'attente correspondante. Dans notre exemple, la première file, prioritaire, recevra les flux qui requièrent un faible délai (une forte qualité de service).

• *Bucket (*I.5.2.1, I.5.2.2*)*
Les *token bucket* offrent une fonction de lissage (*shaping*) et de police (*policing*). Pour le *shaping*, la mémoire de la file d'attente des mécanismes de *bucket* est partagée avec la mémoire de la file d'attente correspondante.
Les *leacky bucket* sont un cas particulier des *tocken bucket* dans lesquels le nombre maximum de jeton est limité à 1 paquet.

• *WRED (Weighted Random Early Detect* - I.5.2.3*)*
Le mécanisme de suppression aléatoire de paquets en fonction de la charge de la file fonctionne avec des paramètres différents pour chaque file, ce qui permet de faire du *WRED* dans les réseaux à qualité de service (si les files se partagent la même mémoire).
Le modèle *WRED* est fortement recommandé dans les réseaux IP (en particulier dans les files recevant le trafic TCP) dans [RFC2309] et l'implémentation dans DHS est celle définie dans [FLO93].

Figure 84: **Détail d'une interface IP**



Le nombre de files prioritaires et de files GPS est configurable.

V.5.2.1 Ordonnancement

Priorité

La gestion des files prioritaires est triviale : les files prioritaires ayant des paquets en attente sont servies d'abord, dans l'ordre de priorité. Si aucun paquet n'est en attente dans ces files, ce sont celles qui sont gérées par l'ordonnanceur de type GPS qui seront servies.

GPS et autres algorithmes de partage équitable

GPS a été décrit au paragraphe I.5.2.4. Les mécanismes implémentés dans DHS sont :

• GPS théorique

L'ordonnanceur GPS théorique a été implémenté dans le simulateur, afin de pouvoir comparer le séquencement des paquets par rapport à un ordonnanceur réel. Ce mécanisme ne peut pas être utilisé dans les routeurs réels car les paquets réels sont indivisibles.

Plus précisément, dans un ordonnanceur GPS, soit T_i le temps de traitement total devant être alloué au premier paquet de la file i et $S_i(t)$ le temps de service déjà alloué au premier paquet de la file i à la date t. L'ordonnanceur GPS sert toutes les files contenant au moins un paquet selon leur poids ϕ_i. La date de sortie D_i du premier paquet de la file i est donné par :

$$D_i = t + (T_i - S_i(t)) \cdot \frac{\phi_i}{\sum_{\text{file } j \text{ non vide}} \phi_j} \qquad (103)$$

L'algorithme est le suivant : à chaque événement d'entrée ou de sortie d'un paquet dans le système, les dates de sortie pour chaque file i sont recalculées, et le paquet ayant la date de sortie la plus petite sort le premier.

• «Weighted Random» (hasard pondéré)

Ce mécanisme est utilisé dans certains routeurs Internet. La file servie est choisie au hasard par tirage aléatoire pondéré par son poids par rapport à la somme des poids des files non vides, sans tenir compte de la taille des paquets.

• CBWFQ : une Approximation discrète de GPS

Plusieurs approximations discrètes du paradigme GPS (II.2.1.1) ont été étudiées [TOU98]. Nous présentons ici l'algorithme WFQ (*Weighted Fair Queueing*) appelé aussi PGPS (*Packetized GPS*) qui est utilisé dans le simulateur.

Le principe de WFQ est de servir les paquets dans l'ordre de leurs dates virtuelles de départ. Le temps virtuel, noté $V(t)$, fournit une estimation de la progression du service normalisé des sessions actives. Selon (47) (page 67), on a :

$$\frac{W_i(\tau_1, \tau_2)}{\phi_i} = \frac{C(\tau_2 - \tau_1)}{\sum_{j \in B(\tau_1, \tau_2)} \phi_j} \qquad \forall i \in B(\tau_1, \tau_2) \qquad (104)$$

Dans une période active $[\tau_1,\tau_2]$, le temps virtuel $V(t)$ est alors défini comme suit :

$$V(t) = V(\tau_1) + \frac{C(t - \tau_1)}{\sum_{i \in B(\tau_1, \tau_2)} \phi_i} \qquad 0 \le t - \tau_1 \le \tau_2 \qquad (105)$$

Remarque :

$$V(t) - V(\tau_1) = \frac{W_i(\tau_1, t)}{\phi_i} \qquad 0 \le t - \tau_1 \le \tau_2 \qquad (106)$$

Au début d'une période d'activité de date réelle τ_1 (lorsque le système passe de l'état vide à l'état non vide par l'arrivée d'un client), on pose $V(\tau_1)=0$. Pour chaque client k de la session i active durant la période d'activité, on note a_i^k la date réelle de son arrivée, L_i^k la quantité de travail requise

et F_i^k son étiquette. Les étiquettes sont calculées comme suit :

$$F_i^k = \frac{L_i^k}{\phi_i} + max(F_i^{k-1}, V(a_i^k)) \qquad et \qquad F_i^0 = 0 \tag{107}$$

À chaque fin de service d'un paquet, la session (la file) *i* élue est celle dont le client à servir possède la plus petite étiquette.

Bien que de manière globale l'algorithme WFQ approxime bien GPS, sa dynamique peut s'en écarter considérablement. En effet, le traitement de certains messages peut être fini sous WFQ alors qu'ils n'auraient même pas commencé leurs services en GPS. Pour pallier à ce problème, d'autres algorithmes ont été proposés [TOU98].

Cette approximation efficace de GPS dans les réseaux à commutation de paquets est implémentée dans DHS. Elle est également utilisée dans les routeurs commercialisés par certains grands équipementiers.

V.5.3 Coeurs de réseaux MPLS

Les réseaux MPLS introduisent de nouveaux éléments dans les structures de données du simulateur, dont il faut tenir compte pour la simulation et la génération des résultats.

Concernant la simulation, un coeur de réseau MPLS implique la prise en compte de plusieurs choses :

- L'existence des LSP, avec leurs chemins et la liste des trafics commutés par ces LSP,
- Le temps de commutation des paquets MPLS dans les routeurs MPLS,
- Pour chaque trafic encapsulé dans un LSP, et globalement pour les LSP : le calcul du délai, de la gigue, et les probabilités de pertes de bout-en-bout,
- La prise en compte des LSP de secours (re-routage) en cas de panne des LSP primaires,

Les calculs statistiques des délais, liens et pertes sont décrits au paragraphe V.4.

V.6 Validations

Nous présentons ici des tests de validité et de performance du simulateur DHS.

Dans une première partie, les tests seront réalisés sur un réseau académique (réseau « Double X »). Après avoir montré la validité des simulations complètement analytiques et complètement événementielles, les deux principes de simulation hybrides seront vérifiés et nous verrons que les temps de calcul sont plus faibles par rapport à la simulation événementielle.

Dans une seconde partie, nous agrégerons des sources de trafic non Poissonien. Nous montrerons que lorsque ces sources sont agrégées en nombre suffisant de façon à permettre l'utilisation de modèles analytiques simples, les techniques analytiques et hybrides permettent d'obtenir des résultats précis en un temps réduit.

La troisième partie illustre l'utilisation d'un modèle analytique à service déterministe. Là encore, les simulations hybrides seront confrontées à la simulation événementielle.

Enfin dans la dernière partie, nous simulerons un réseau IP de taille réelle. L'ordonnancement dans des routeurs d'entrée (composés de plusieurs files d'attente) seront de type GPS. Les autres

routeurs auront des files d'attente PAPS.

V.6.1 Réseau académique

Le réseau académique utilisé dans ce paragraphe est présenté en figure 85. Dans toute la suite de ce paragraphe, les flux entrent dans les files d'attente 0, 1, 2 et 3, et sortent après avoir été servis par les serveurs 4 et 5. La liste des flux et leur routage seront explicités pour chaque test.

Figure 85: **Réseau académique**

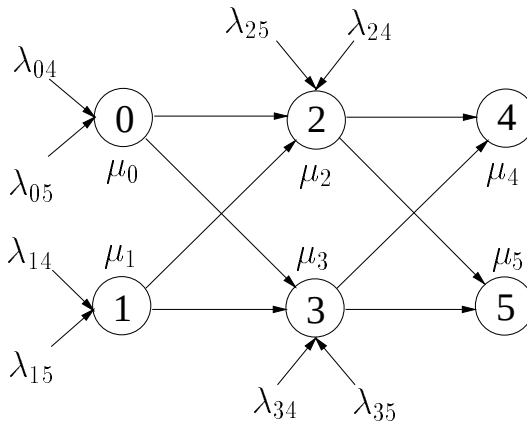


Tableau 21: **Valeurs des taux d'arrivées λ_{ij}**

λ_{04}	λ_{05}	λ_{14}	λ_{15}	λ_{24}	λ_{25}	λ_{34}	λ_{35}
2	3	3	2	2	1	1	2

Tableau 22: **Valeurs des taux de service μ_i**

μ_4	μ_1	μ_2	μ_3	μ_4	μ_5
10	10	15	15	10	10

V.6.1.1 Description des trafics

Dans ce paragraphe, les inter-arrivées de clients sont issues d'une loi de Poisson et les services sont exponentiels. Les trafics directs entrant aux noeuds 2 et 3 et destinés aux noeuds 4 et 5 n'ont qu'une seule route : le lien direct. Mais les trafics entrant aux noeuds 0 et 1 et destinés aux 4 et 5 peuvent être routés suivant deux chemins différents : 50% du trafic est envoyé sur chacun de ces chemins (routage par partage de charge).

Les tests sont réalisés avec les flux de débit $\lambda_{source,destination}$ du tableau 21, et les taux de services μ_i en chaque noeud i sont listés dans le tableau 22.

Le tableau 23 donne la charge moyenne théorique de chaque noeud (file d'attente et serveur), ainsi que le nombre d'itérations (point fixe en analytique ou temps de simulation événementielle) et le temps d'obtention des résultats.

Pour chaque simulation, le simulateur exhibe les délais de propagation de bout en bout pour chaque flot. La loi de Little permet de prévoir ces délais (analytique ou événementiels) en sommant les temps de traversé de chaque noeud. La simulation événementielle permet de calculer les délais et la gigue en sortie du dernier système traversé par chaque flot. Ces résultats sont présentés dans le tableau 24.

V.6.1.2 Interconnexion hybride

Le même réseau est maintenant simulé de la façon suivante : les flux sont générés de façon événementielle, les noeuds 0 et 1 sont événementiels et les noeuds 2, 3, 4 et 5 sont analytiques. Les flots sortants des noeuds 0 et 1 sont alors convertis en débits analytiques avant d'être traités sur la suite de leurs chemins respectifs.

Les tableaux 23 et 24 rapportent les résultats dans les colonnes « Hybride 1 ».

V.6.1.3 Superposition hybride

Le second principe de simulation hybride permet de superposer quelques flots événementiels sur un réseau pouvant être complètement analytique. Dans chaque noeud, les flots peuvent être de types différents. Le test présenté dans les tableaux 23 et 24 correspond à une simulation dans laquelle seulement le flot λ_{05} est événementiel. Les résultats sont dans les colonnes « Hybride 2 ».

V.6.1.4 Résultats

Tableau 23: **Nombre moyens de clients à l'équilibre dans chaque noeud**

	Exact	Analytique	Evénementielle	Hybride 1	Hybride 2
X_0	1,000	1,000	1.018	0.971687	0.986354
X_1	1,000	1,000	1,000	0.982165	1.00007
X_2	1,143	1,143	1,142	1.14538	1.13243
X_3	1,143	1,143	1,139	1.13961	1.14285
X_4	4,000	4,000	3.890	3.92533	4.00267
X_5	4,000	4.000	4,003	4.07152	3.9157
Itérations		Etapes du point fixe: 1000	Temps simulé : 10000s	Temps simulé : 10000s	Temps simulé : 10000s
Temps de calcul		0.71s	3.81	2.00s	1.31s
Erreur rel. max.		0%	2.75%	2.83%	2.10%

Tableau 24: **Délais et gigues**

Flux	Délai (ms) analytiques	Délai (ms) événementiels	Délais (ms) Hybride 1	Délais (ms) Hybride 2	Gigue (ms) Hybride 2	Gigue (ms) événementielle
Flot 0-4	842.863	837.424	830.864	841.418	N.C.	244.183
Flot 0-5	842.863	847.774	845.484	840.377	243.03	205.807
Flot 1-4	842.863	828.571	832.444	842.79	N.C.	202.978
Flot 1-5	842.863	844.116	847.064	834.093	N.C.	243.776
Flot 2-4	642.861	631.209	635.562	642.433	N.C.	202.089
Flot 2-5	642.861	633.516	650.182	633.736	N.C.	259.465
Flot 3-4	642.861	621.518	635.177	643.128	N.C.	261.755
Flot 3-5	642.861	641.34	649.797	634.431	N.C.	201.969

Flux	Délai (ms) analytiques	Délai (ms) événementiels	Délais (ms) Hybride 1	Délais (ms) Hybride 2	Gigue (ms) Hybride 2	Gigue (ms) événementielle
Err. rel. max.		3.3% /ana	1.42% /ana	1.41% /ana	18% /eve	

Ces deux tableaux permettent de constater sur un réseau test que les quatre types de simulations présentés donnent des résultats très proches. La marge d'erreur sur les charges est faible. Elles sont comparées aux résultats analytiques qui sont ici exacts.

Nous pouvons également remarquer que les temps de calcul nécessaires pour réaliser les simulations hybrides sont inférieurs au temps nécessaire pour la simulation événementielle. Par rapport à la simulation événementielle, le gain est de 47% pour l'interconnexion hybride et 65% pour la superposition hybride.

V.6.2 Agrégation de sources de trafics

Sur un réseau très simple, nous illustrons une agrégation de sources de trafics. Le réseau utilisé est celui du paragraphe V.6.1. Deux sources empruntent le même chemin : les noeuds 0, 2 et 4. L'une est une agrégation de trafics voix (codec G711) et l'autre constitue un seul trafic voix.

Dans un premier cas de figure, la source agrégée est constituée de 20 trafics voix (tableaux 25 et 26). Dans le second cas, cette source est constituée d'un seul trafic voix (les deux trafics sont alors symétriques) (tableau 27 et 28).

Pour chacun des cas considérés, quatre simulations sont réalisées. Une est totalement événementielle, une est totalement analytique, la troisième et la quatrième sont hybrides (interconnexion et superposition).

Pour pouvoir comparer la simulation analytique et la simulation événementielle, les paquets générés par le codec G711 ont des tailles de traitement distribuées exponentiellement (la distribution des inter-arrivées reste celle d'un codec G711). Le flux analytique équivalent est considéré comme un flux Poissonien de débit égal à celui d'une source G711.

L'interconnexion hybride est également utilisé ici. Les noeuds 0 et 4 sont événementiels tandis que le noeud 2 est analytique. Les flux sont générés événementiellement, et sont convertis en débits analytiques en sortie du noeud 0. Les débits de sortie du noeud 2 sont utilisés pour paramétrer les générateurs d'inter-arrivées exponentiels associés à chaque flot afin de réinjecter des clients dans le noeud 4.

Dans le dernier cas (superposition hybride), le réseau est complètement analytique, la source agrégée est analytique et la source unitaire est événementielle de bout en bout.

V.6.2.1 20 + 1 flux G711

Les tableaux 25 et 26 présentent une agrégation de 21 sources Audio G711 sur le même réseau. Le tableau 25 indique les charges pour les quatre types de simulation, et le tableau 26 les délais de bout en bout. Nous pouvons constater que les résultats de la simulation analytique et des simulations hybrides sont proches de la simulation événementielle.

Tableau 25: **charges induites par 21 flux G711**

	Analytique	Evénementiel	Hybride 1	Hybride 2
X_0	0.924059	0.894195	0.892085	0.924276
X_2	0.316019	0.312789	0.316293	0.316064
X_4	0.924059	0.932215	0.919062	0.924227
Itérations	Etapes du point fixe: 1000	Temps simulé : 10000s	Temps simulé : 10000s	Temps simulé : 10000s
Temps de calcul :	0.26s	122s	67s	4.57s

Tableau 26: **Délais et gigues (ms)**

Flux	Délai ana.	Délai évé.	Délai hyb. 1	Délai hyb. 2	Jigue hyb. 2	Jigue évé
20xG711	3.52641	3.47261	3.47496	3.52541	N.C.	1.27271
1xG711	3.52641	3.49033	3.47496	3.37306	2.22938	1.80387

La superposition hybride est relativement précise concernant la charge globale et les délais de bout en bout. Les 20% d'erreur sur la gigue peuvent être considérés comme acceptables compte tenu du fait que le modèle analytique associé n'est pas adapté à la source de trafic sous-variante utilisée.

Ce test fait apparaître l'intérêt du concept de simulation hybride. En effet, le tableau 25 montre clairement que les temps de calcul nécessaires pour les simulations hybrides sont bien inférieurs que le temps de calcul de la simulation événementiel pour le même temps simulé.

Grâce à ces résultats, nous pouvons prévoir d'utiliser l'interconnexion hybride de la manière suivante : Les noeuds d'accès au réseau sont simulés par des événements pour permettre l'agrégation de flux non Poissoniens, et les noeuds de coeur de réseau sont simulés par des équations.

V.6.2.2 Deux flux G711

Nous avons choisi de présenter ce test pour montrer que le choix du modèle analytique utilisé dans la simulation hybride est primordial. En effet, nous avons repris le même exemple en remplaçant l'agrégation de 20 sources G711 par une seule. Le tableau 27 montre de façon claire une divergence des résultats analytiques par rapport à la simulation événementielle.

Tableau 27: **charges induites par 2 flux G711**

	Analytique	Evénementiel	Hybride 1
X_0	0.842963	2.73231	2.83946
X_2	0.296509	0.350833	0.298117
X_4	0.842963	2.01841	0.850154

	Analytique	Evénementiel	Hybride 1
Itérations	Etapes du point fixe: 1000	Temps simulé : 10000s	Temps simulé : 10000s
Temps de calcul	0.27s	6.99s	11.85s

Tableau 28: **Délais et gigues (ms)**

Flux	Délai ana.	Délai évé.	Délai hyb. 1
1xG711	33.8668	87.8652	67.8188
1xG711	33.8668	87.8655	67.8188

Les résultats d'interconnexion hybride sont cependant discutables. La charge du noeud 0 est bonne, puisque il est événementiel. Le noeud 2 est analytique, et donne les même résultats que la simulation analytique (le débit moyen entrant est le même). Le noeud 4 est événementiel mais sa charge n'est pas bonne. Cela est dû au fait que la distribution des inter-arrivées en entrée est purement exponentielle (pour l'interconnexion hybride, le générateur à l'entrée du noeud 4 a la même distribution que celle du noeud 2 en sortie).

Les erreurs constatées sont donc très liées aux modèles analytiques employés dans la simulation hybride. Des modèles qui seraient capables de tenir compte de plusieurs moments des inter-arrivées en entrée (au moins les deux premiers) permettraient de mieux paramétrer les générateurs de paquets en sortie des noeuds analytiques.

V.6.3 Service D

Nous proposons ici une simulation utilisant une file d'attente à service déterministe et à capacité limitée. Le modèle utilisé dans la simulation analytique est M/D/1/*N* (distribution exponentielle en entrée, service déterministe, *N*=2).

Le réseau et le routage sont les mêmes que précédemment (la route suit le chemin 0, 2 et 4). Le trafic est une agrégation de 20 sources voix G711. Le rapport des débits moyens d'entrée sur débits moyens de service est $\rho=1/3$.

- L'interconnexion hybride consiste ici à utiliser la simulation événementielle pour le noeud 0 et le modèle analytique pour les noeuds 2 et 4.
- Dans la simulation analytique, le modèle du noeud 0 est M/D/1/2. Les noeuds 2 et 4 sont M/M/1.
- Dans la simulation événementielle, le service du noeud 0 et D, la capacité du noeud 0 et 2, et le service des noeuds 2 et 4 est M.

Nous pouvons constater dans le tableau 29 que le processus de perte calculé dans la simulation hybride correspond à celui de la simulation événementielle. La valeur calculé dans le modèle analytique est une bonne approximation compte tenu que le flux entrant n'est pas tout à fait Poissonien. La charge calculée par le modèle analytique dans le noeud 0 est très proche du résultat événementiel. Dans les noeuds 2 et 4, l'erreur sur les charges varie entre 10% et 15%. Ceci est du au

caractère non exponentiel de la distribution des temps d'inter-arrivées dans le noeud 2.

Tableau 29: **Service D en entrée**

	délai (ms)	X_0	X_2	X_4	Pertes sur le noeud 0 (%)	Temps de calcul (10000s simulées)
Evénementiel	2.56424	0.41923	0.480901	0.517235	5.32852	91.23s
Hybride	2.80102	0.419807	0.565384	0.565298	5.34825	41.73s
Analytique	2.79542	0.418512	0.558017	0.558017	6.03531	0.20s

Encore une fois, nous pouvons constater que le temps de simulation hybride est très inférieur au temps de simulation événementiel (54% de gain).

V.6.4 Réseau réel et Interconnexion Hybride

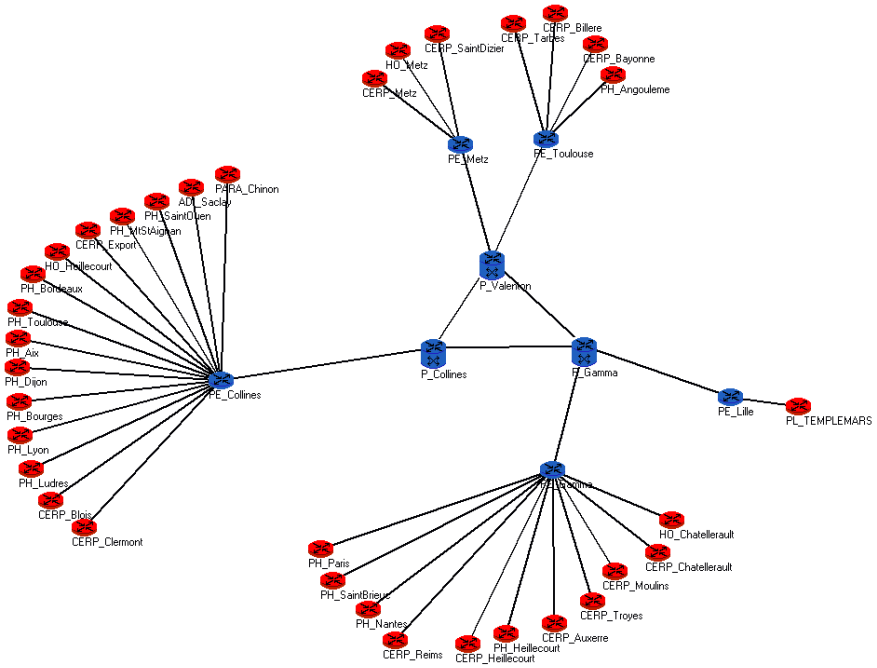
V.6.4.1 Structure du réseau

Nous présentons ici la simulation d'un réseau de grandeur réelle (figure 86). Le réseau est composé de 39 routeurs IP, 37 liens, et 74 interfaces de sorties. 142 sources, réparties en 5 classes (EF, AF1, AF2, AF3, BE) chargent ce réseau. Pour pouvoir comparer simulation événementielle, simulation analytique et simulation hybride, les mécanismes de contrôle d'admission et de congestion ont été désactivés (*shaper, RED*). La distribution des inter-arrivées de paquets et de services est exponentielle. En reprenant la structure des réseaux Internet DiffServ, l'ordonnancement de service est WFQ (noeuds événementiels) ou GPS (noeuds analytiques) avec gestion de la priorité dans les routeurs de bordures , et FIFO dans les routeurs de coeur de réseau. Les trafics sont divisés en cinq classes: EF (prioritaire), et pour GPS/WFQ : AF1 (poids 20), AF2 (poids 10), AF3 (poids 5) et BE (poids 1).

Le routage ne sera pas explicité dans ce document. Il est issu du protocole de routage OSPF. La métrique utilisée par tous les liens est 1.

Les modèles DiffServ Analytiques dont nous disposons à l'heure actuelle ne permettent pas de tenir compte de la capacité limitée des files d'attente dans les interfaces. La matrice de trafic utilisée est donc ajustée de façon à ne pas provoquer de congestion.

Figure 86: **Réseau à qualité de service**



V.6.4.2 La simulation

interfaces surchargées

L'interconnexion hybride est réalisée comme suit : L'ensemble des routeurs source et destination de trafic sont simulés événementiellement. Leur nombre est de 31. Les autres noeuds sont simulés analytiquement.

Le tableau 30 illustre les caractéristiques des trois simulations. Le tableau 31 indique les erreurs relatives des charges de files DiffServ. Le tableau 32 indique les erreurs relatives des délais de bout en bout pour chaque flot. Ces deux tableaux sont représentés graphiquement en figure 87. Les détails de chaque file (GPS/WFQ et priorité compris) et chaque flot sont présentés respectivement dans les tableaux 33 et 34 situés en annexe (A.2, page 192).

Nous pouvons constater en examinant les résultats que les différents types de simulation proposent des résultats assez proches, mais avec des temps de calcul très différents (38% de gain de l'événementiel sur l'hybride).

Tableau 30: **Caractéristiques des simulations**

	Analytique		Hybride		Evénementielle	
Temps de simulation	1.93s		116s		190s	
Temps simulé	100 points fixes		10000 s		10000 s	
Type de simulation	Ana.	Eve.	Ana.	Eve.	Ana.	Eve.
Nombre de routeurs	39	0	8	31	0	39

***Tableau 31:* Statistiques des erreurs de charge (pourcentage)**

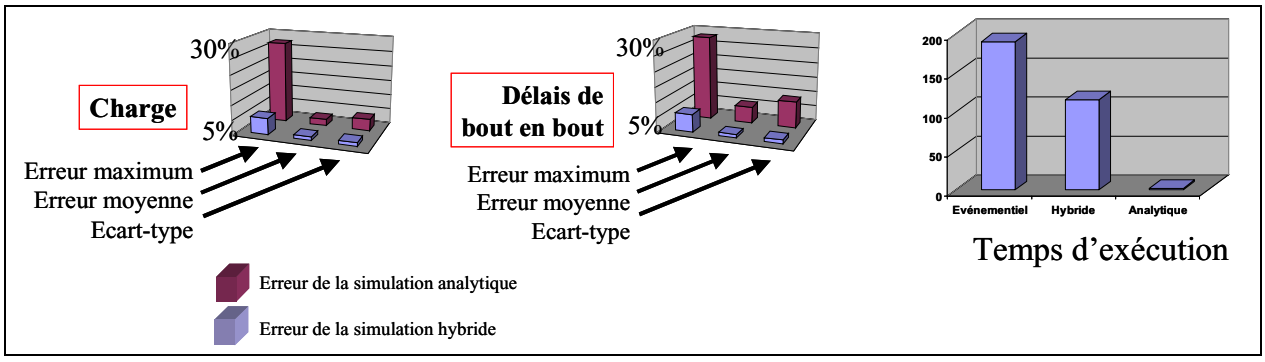
Comparaison	Analytique / Evénementiel	Hybride / Evénementiel
Erreur Max.	27.3%	5.55%
Erreur Moy.	2.20%	1.18%
Ecart-type	3.91%	1.23%

***Tableau 32:* Statistiques des erreurs de délais (pourcentage)**

Comparaison	Analytique / Evénementiel	Hybride / Evénementiel
Erreur Max.	28.1%	5.85%
Erreur Moy.	5.50%	1.17%
Ecart-type	8.86%	1.26%

Prenons la simulation événementielle comme référence. L'erreur maximum de la simulation analytique est de 28% (charge et délai) mais l'écart type reste faible (9% pour les délais, 4% pour les charges). La simulation hybride donne de meilleurs résultats (6% d'erreur au maximum, écarts types de 1.3%). Cela est dû aux modèles analytiques à ordonnancement GPS que nous utilisons : ces modèles sont une approximation de GPS, alors que les algorithmes événementiels de partage équitable sont WFQ (la granularité des paquets est prise en compte). Ces erreurs sont visibles dans l'interface surchargée n°52 du tableau 33.

***Figure 87:* Statistique des erreurs (environnement surchargé)**

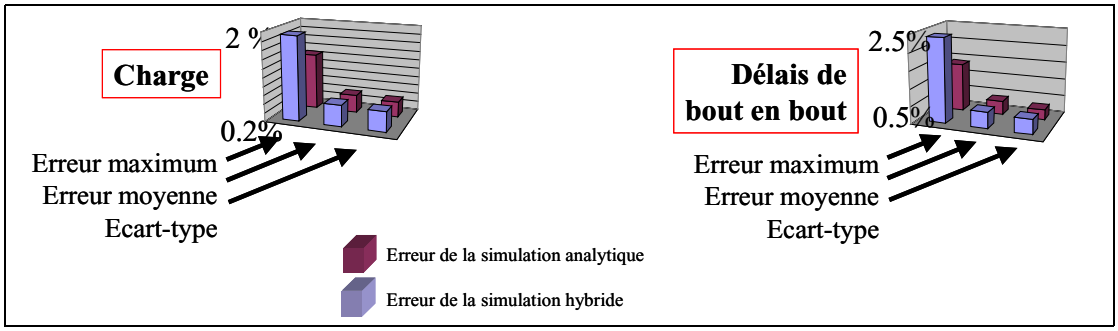


A la précision des résultats de la simulation hybride s'ajoute un gain de temps considérable : près de trois quarts (75%) des noeuds sont événementiels, et le temps de simulation est seulement de trois cinquièmes (60%) de celui de la simulation complètement événementielle. Cette réduction du temps d'exécution s'explique par le fait que le nombre d'événements traités par le simulateur est réduit. En effet, tous les trafics transitent par le coeur de réseau. Cela engendre dans la simulation événementielle un très grand nombre d'événements, et celui-ci est très largement réduit par l'utilisation des modèles analytiques.

interfaces non-surchargées

Nous avons réalisé les mêmes simulations en réduisant le débit des sources de façon à ne pas surcharger les interfaces citées dans le paragraphe précédent. La figure 88 représente un résumé des statistiques globales des erreurs. Ces résultats permettent de mettre en valeur la justesse et la cohérence des modèles employés sur un réseau de taille réelle. Les temps de simulation sont identiques que précédemment.

Figure 88: **Statistique des erreurs (environnement non-surchargé)**



V.7 Conclusion

Dans ce chapitre, nous avons présenté l'outil DHS qui est un simulateur général de files d'attente. Cet outil permet de modéliser et de simuler les réseaux de télécommunication à commutation de paquet. Deux types de simulation sont possibles : la simulation événementielle et la simulation analytique.

La simulation événementielle est une simulation stochastique à événements discrets. Elle nécessite un temps en général très long pour permettre d'obtenir des bonnes statistiques de l'ensemble des composants simulés (états transitoires ou régime stationnaire, moyennes et variances des charges des interfaces, des charges des files, des charges induites par chaque flot, délai des files et des flots de bout en bout, gigues, probabilités de perte par file, par flot). Le régime stationnaire peut être obtenu par une simulation à long-terme, tandis que le régime transitoire est obtenu en réalisant des simulations parallèles.

La simulation analytique repose sur la théorie du trafic différentiel. Le principe général de cette théorie est d'exprimer l'équation différentielle associée à la moyenne de chaque flux dans chaque ressource du réseau. Ces équations peuvent être explicites (équations différentielles autonomes) ou implicites. Les paramètres des équations implicites peuvent être résolus par simulation événementielle, ce qui est l'objet de la simulation hybride. La simulation analytique a l'avantage d'être extrêmement rapide. Là encore, il est possible d'obtenir le régime stationnaire du système en résolvant un algorithme de point fixe, ou le régime transitoire en intégrant pas à pas les équations différentielles.

Nous avons montré dans ce chapitre que la combinaison des techniques analytiques et des techniques événementielles permettent d'obtenir des résultats précis avec des temps de calculs réduits. Deux modèles de simulation hybride ont été présentés.

L'interconnexion hybride est bien adaptée à la simulation des grands réseaux. Des modèles analytiques simples peuvent être utilisés pour la simulation du coeur de réseau tandis que la simulation événementielle est utilisée pour agréger les trafics en entrée de réseau. La simulation événementielle permet également l'utilisation de mécanismes dont on ne possède pas forcément un

modèle analytique précis. Nous avons vu en particulier, sur un réseau de taille réelle que l'interconnexion hybride apporte un gain en temps de calcul considérable sur la simulation événementielle pure pour une précision tout à fait correcte. La superposition hybride est intéressante pour l'étude de certains flots. Il permet l'étude précise de trafics particuliers sur une simulation analytique complète du réseau. Le choix des modèles analytiques employés est alors plus sensible.

De manière générale, nous avons pu constater que l'utilisation des formules analytiques M/M/1 dans la simulation hybride était très bien adaptée pour l'étude précise des flots Poissoniens. Bien sûr, lorsque l'agrégation n'est pas suffisante pour retrouver le caractère Poissonien, le principe général hybride reste valide, mais il est cependant nécessaire d'utiliser des formules analytiques plus complexes ou approchées permettant de tenir compte des caractéristiques de l'agrégat.

Nous avons également montré comment nous nous servions du simulateur général de file d'attente qu'est DHS pour modéliser les réseaux IP-DiffServ-MPLS. Le modèle général de routeur et les mécanismes de gestion de la congestion (*policing*, *shaping*, *random early detect*) ont été validés par les études de métrologie réalisées pour le projet RNRT Esquimaux. Ces mécanismes peuvent être utilisés dans la simulation hybride sur les noeuds événementiels ou pour les flots événementiels. Enfin, nous avons montré la précision de la simulation hybride et les gains en temps de calcul qu'elle apportait pour de tels réseaux.

Chapitre VI - Environnement d'exécution distribuée

La simulation événementielle de réseaux de tailles petites à moyennes induit des temps de calculs très longs et nécessite beaucoup de mémoire.

Une possibilité pour réduire les temps de calcul est, lorsqu'on désire des intervalles de confiance assez réduit, la modélisation analytique des organes réseaux. Pour des raisons théoriques, certains modèles de composant de réseau ne peuvent actuellement pas être analytiquement modélisés et doivent être simulés de manière événementielle.

Une autre possibilité consiste à utiliser des processeurs plus puissants et des mémoires plus grandes. Outre le problème du prix des machines utilisées qui se pose, un seul processeur n'est pas, et ne sera jamais assez puissant pour simuler tous les composants événementiels d'un réseau de taille réelle.

Du fait du développement de plusieurs nouvelles technologies de réseaux à haut débit et de la démocratisation des ordinateurs personnels, il est possible de se procurer à bas prix des clusters de processeurs pour le calcul intensif. La parallélisation de l'algorithme de simulation événementielle permettrait ainsi de réduire les temps de simulation d'un facteur égal, dans le meilleur des cas, au nombre de processeurs utilisés.

Dans le cadre de cette thèse, une étude sur les environnements de programmation parallèle a été menée, et une nouvelle bibliothèque de communication sur réseau hétérogène haut débit a été développée, venant s'insérer dans l'environnement d'exécution distribuée LANDA (*Local Area Network for Distributed Applications*) [MON95].

L'objectif que nous nous sommes fixé concernant cette nouvelle bibliothèque est multiple :

- la bibliothèque doit être multi-réseau : Elle doit permettre l'utilisation simultanée de plusieurs media de communication au sein de la même application parallèle. Cela permet d'une part d'utiliser l'ensemble des calculateurs répartis dans plusieurs grappes (ou *clusters*), et d'autre part, dans une grappe, d'utiliser le meilleur réseau d'interconnexion (en général propriétaire, haut débit et de latence faible) pour optimiser les communications,
- la bibliothèque doit être évolutive : La possibilité d'utiliser de nouveaux media de communication doit être facilité. De nouvelles cartes d'interface de communication à architecture propriétaire sont en permanence mises sur le marché. La bibliothèque doit être conçue de manière à faciliter l'adaptation de ces nouveaux media,
- la bibliothèque doit être multi-API : De nombreuses applications distribuées existent déjà et utilisent une interface de programmation (API) déjà existante. Notre bibliothèque doit être compatible avec ces interfaces afin d'éviter de réécrire les applications parallèles,
- le noyau de communication doit être intégré à l'ensemble de la bibliothèque LANDA déjà existante, et mettre à profit la gestion de la machine virtuelle et l'analyseur réseau Network-Analyser [MON96].

L'environnement LANDA-HSN (*High-Speed Networks*) ainsi constitué a été conçu pour une utilisation efficace des ressources disponibles (réseaux, processeurs, mémoires) dans les grappes de machines. Nous allons en décrire les principaux mécanismes ainsi que la mise en oeuvre du noyau de communication sur des machines SMP et à travers des réseaux d'interconnexions haut-débit.

Le plan est organisé comme suit : dans le paragraphe VI.1 nous introduirons les environnements

d'exécution distribués. Le paragraphe VI.1 présente d'autres environnements parallèle existant. L'architecture générale de l'environnement LANDA-HSN est détaillée au paragraphe VI.3. Le paragraphe VI.4 développe la conception et la réalisation de la nouvelle architecture de communication utilisée par LANDA. Cette architecture est basée sur une couche de communication nommée Media et dont les principes d'utilisation sont détaillés au paragraphe VI.5. LANDA permet l'utilisation d'interface logicielle standard (API) telle que l'interface MPI. Cette implémentation est décrite dans le paragraphe VI.6. Enfin le paragraphe VI.7 illustre les performances de la simulation hybride distribuée en environnement d'exécution parallèle.

VI.1 Introduction

Le très intéressant rapport prix / performance des machines de type PC grand public permet la constitution à bas coût de grappes (ou clusters) de PC afin de les utiliser comme une machine parallèle. Ce type de machine a une puissance potentielle énorme [TOP].
Le type de réseau d'interconnexion est déterminant dans le choix du type d'algorithme pouvant être exécuté sur les grappes. Un réseau lent n'autorisera que des algorithmes parallèles de type « gros grain (*coarse grain*) » (c'est à dire un temps de calcul très supérieur aux temps de communication), tandis que les réseaux haut-débit à faible latence autorisent l'exécution d'algorithmes de type « grain fin (*fine grain*) ».

Les développements technologiques récents offrent des cartes d'interfaces pour réseau haut-débit (atteignant et dépassant le Gigabit/s) à un prix abordable. On peut citer SCI[SCI93], ATM[ATM95], MPC[MPC], Myrinet[MYR] et les cartes Giga-ethernet de plus en plus disponibles sur le marché grand-public à bas prix.

Le projet LANDA[MON95] a débuté en 1989 au LAAS-CNRS. C'est un environnement parallèle général pour les clusters de stations de travail, disposant d'un environnement complet avec des interfaces graphiques (trace, charge, état dynamique) et permettant d'exécuter et configurer des applications parallèles. A cet environnement s'ajoutent un ensemble de processus de surveillance de l'état du réseau et des calculateurs sur le réseau.
Cette organisation permet une vue globale de la machine et particulièrement de sa charge et de sa disponibilité. La structure hétérogène du réseau ainsi que son architecture sont transparentes pour l'utilisateur LANDA. Les tâches d'une application parallèle communiquent en utilisant le paradigme de passage de message, l'emplacement précis de la tâche ne devant pas être nécessairement connu de l'utilisateur.
LANDA est associé au système Network-Analyser [MON96]. Celui-ci permet d'analyser et de prévoir les charges du réseau et des machines. C'est un outil d'observation (CPU, charge, états des processus, entrée/sortie, trafic global et communication point à point). À partir des informations significatives récoltées, cet outil de décision calcule le placement optimal des tâches parallèles sur les calculateurs permettant d'obtenir un temps d'exécution minimal de l'application [GAR99]. Network-Analyser fournit une bibliothèque qui peut être utilisée par tout type d'environnement d'exécution parallèle (LANDA [MON95], PVM [SUN94], MPICH[MPI]) pour obtenir des informations.

Le système de communication mis en oeuvre dans la version 3 de LANDA, appelé LANDA-HSN (*High Speed Network*), est fondé sur une utilisation efficace de l'architecture du calculateur et des ressources réseau. Il est principalement basé sur la séparation en deux parties des messages, qui sont l'entête et les données. L'objectif de ce nouveau système de communication est de distribuer et d'optimiser le système de boîte aux lettres dans le but de rendre les opérations de communication plus efficaces.

Dans la suite, nous porterons notre attention sur le modèle de communication sur lequel nous nous appuyons dans LANDA-HSN. Nous décrirons tout d'abord l'architecture de la machine virtuelle définie dans LANDA, puis nous détaillerons les schémas de communication entre les tâches sur les machines SMP et à travers les interconnexions de réseaux.

VI.2 Projets similaires

De nombreux projets sur les environnements d'exécution parallèle sur grappes de stations ou de PC se sont développés ces dernières années. Ils ont d'abord été conçus pour des réseaux locaux de stations de travail ou des machines parallèles, avant de considérer les grilles (grand nombre de stations dédiées au calcul intensif) d'une part, puis les interconnexions de grilles d'autre part.

L'un des projets bien connu est Beowulf (1994) utilisant des PC sous le système d'exploitation Linux. Il existe de nombreux autres projets : LOKI (Los Alamos National Laboratory), Hyglac (Caltech/JPL), PopC (MATRA Système et Information), NOW (University of California, Berkeley).

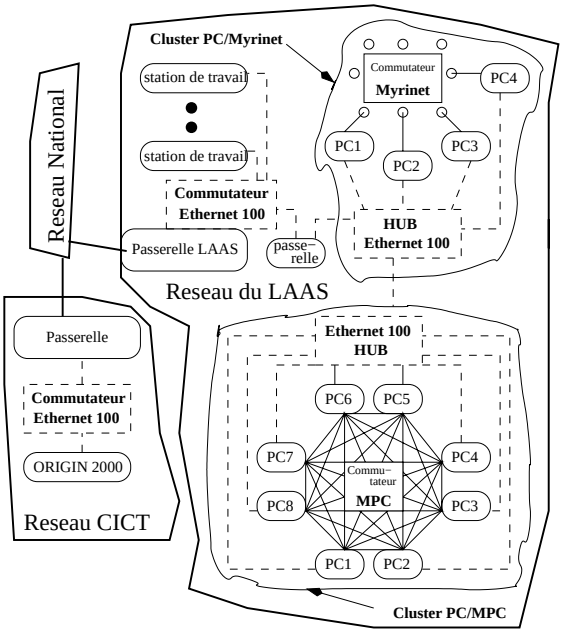
Plusieurs paradigmes de programmation sont disponibles : parallélisation automatique (HPF [HPF], passage de message (PVM [SUN94], MPI [MPI]), mémoire partagée (OpenMP [OMP]). Il est alors naturel d'utiliser ces environnements sur des grilles bien qu'ils n'aient pas été conçus pour ce type d'architecture. En effet, ces bibliothèques proposent en général plusieurs implémentations de la couche communication, correspondant aux cartes d'interface disponibles sur les machines.

Cependant, ces implémentations sont souvent incapables d'utiliser plusieurs cartes d'interface simultanément, limitant ainsi l'utilisation de la grille à un seul media de communication et réduisant les possibilités d'interconnexions. De plus, les grilles sont souvent construites avec des cartes d'interface et des protocoles de communications spécifiques ce qui implique que les couches de communication et les bibliothèques utilisateurs doivent être adaptées pour obtenir la meilleure efficacité.

Pour chaque protocole, une couche de communication particulière est par conséquent nécessaire, mais ces couches doivent être suffisamment génériques pour être facilement intégrées dans un environnement hétérogène, et suffisamment flexibles de façon à éviter la récriture de toutes les bibliothèques à chaque fois qu'un nouveau type de réseau apparaît. LANDA-HSN propose d'intégrer de telles caractéristiques. D'autres environnements comme MP_MPICH[MPM], ou PM2/ Madeleine[PM2] font de même.

VI.3 Architecture de LANDA-HSN

Figure 89: **Exemple d'architecture hétérogène distribuée et interconnectée par des réseaux haut-débit hétérogènes**



L'architecture de LANDA-HSN fait apparaître le concept de machine virtuelle. La machine virtuelle sert à abstraire une architecture sous-jacente complexe. La complexité liée à l'utilisation des services de cette architecture est cachée à l'utilisateur.

La machine virtuelle peut être vue comme un seul calculateur multi-processeur à mémoire distribuée offrant un accès unifié aux services de l'architecture réelle.

VI.3.1 Modèle de la machine virtuelle

Une machine virtuelle peut avoir une architecture complexe (figure 89), par l'hétérogénéité du matériel (PC, stations), des systèmes d'exploitation et des interconnexions réseau (topologie, protocole). Pour utiliser efficacement une telle machine, il est nécessaire d'avoir une bonne connaissance de l'architecture. LANDA permet aujourd'hui de prendre en compte trois niveaux d'interconnexion réseau à l'intérieur d'une machine virtuelle parallèle.

- Le niveau topologie globale (interconnexion de grappes)
- Les points d'accès de chaque grappe
  les correspondances entre liens et interfaces,
- La topologie locale de chaque grappe
  l'architecture physique (bus, répéteurs, commutateurs) et les ports de connexion,

Cette description fine permet au noyau LANDA de gérer les communications efficacement. Le niveau physique fournit les informations pour créer les tables de routages des messages, et les deux autres niveaux fournissent les informations pour permettre la distribution des tâches au travers des réseaux, de façon à ce que les ressources soient utilisées aussi efficacement que possible. De plus, le niveau réseau permet aux tâches parallèles d'être exécutées sur différents sites.

VI.3.2 Le modèle de communication

Le modèle de communication est basé sur le paradigme de passage de messages. Les messages sont découpés en deux parties différentes : l'entête et les données. Les entêtes (de faible taille) sont toujours stockées dans la « boîte aux lettres » de la machine réceptrice. Les données associées ne suivent pas forcément l'entête sur la machine réceptrice; leur placement optimal permettra de réduire le temps de communication sur le réseau par rapport aux capacités des liens en terme de latence et de débit. La séparation des entêtes et des données permet également de choisir parmi plusieurs modèles de communication selon la politique de stockage des messages.

La mémoire distribuée ainsi constituée par la mémoire propre de chaque calculateur permet de concevoir un système de communication partiellement ou totalement distribué. Dans ce cas, plusieurs politiques de routage sont possibles. Les trois principales politiques de routage sont les suivantes :

• les données sont toujours envoyées au noeud récepteur,

• les données restent toujours à la source,

• les données sont stockées sur un autre nœud (de préférence moins chargé et très accessible).

Ces différentes possibilités de routage permettent une meilleure utilisation de la bande passante du réseau pour les communications distantes dont les performances dépendent fortement de la charge des réseaux et des calculateurs. La politique de routage permet également d'optimiser l'utilisation de la mémoire au sein d'un sous réseau très haut débit.

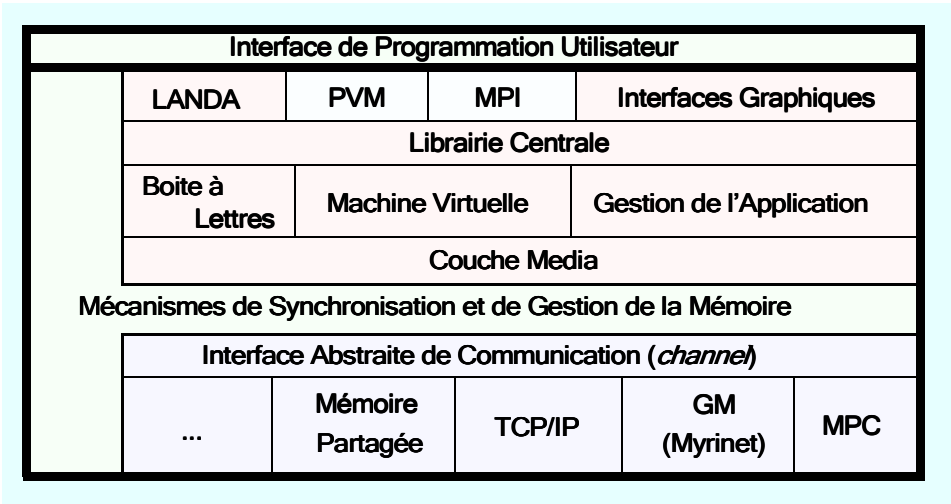
Des stratégies de placement de tâches et de routage dynamique, qui prennent en compte la charge de l'ensemble des machines et des réseaux en temps-réel ont été étudiées dans [MON96] et [GAU98]. Elles sont basées sur un modèle de prédiction dynamique de charge en fonction des mesures instantanées et passées. Le système Network Analyser (NA) réalise en permanence ces mesures.

Le modèle de prédiction dynamique de charge (cpu et réseau) est basé sur un modèle de l'application faisant intervenir les phases de calcul et de communication pour chaque tâche. Un algorithme de type glouton cherche le meilleur placement des tâches en fonction de leur profil.

Au lancement d'une application distribuée, LANDA interroge NA sur le meilleur placement des tâches. Une fois que l'application est lancée, le routage des messages entre les tâches est également calculé par NA et les tables de routage des messages entre les tâche LANDA sont mises à jour régulièrement.

Les communications entre les tâches LANDA sont réalisées par la bibliothèque centrale de communication (figure 90). Son rôle est de gérer tous les mécanismes de synchronisation et de transfert de données point-à-point et multi-points. Elle offre les services de communication aux niveaux que sont PVM[SUN94], MPI [MPI][LAU97].

Figure 90: **Architecture de LANDA**



Cette bibliothèque offre au développeur d'application une API (*Application Programming Interface* ou interface de programmation pour les applications) de haut niveau, telle que les standards PVM, MPI, ou l'API native de LANDA. Une interface graphique permet de piloter les applications (lancement, arrêt, observation des événements) et de dialoguer avec le Network Analyser.

L'API utilise la Bibliothèque Centrale. Celle-ci a pour fonction de gérer l'ensemble de l'application (lancement effectif et terminaison des tâches, service de nommage et de gestion des boites à lettres, abstraction de l'infrastructure et vision logique des calculateurs). Elle s'appuie sur la couche Media.

La couche Media est une interface permettant de transférer des données en point-à-point ou en multi-point. Les moyens utilisés pour réaliser ces transferts dépendent fortement des media de communication (par exemple une carte réseau) utilisés dans les couches inférieures. La couche media repose sur la couche *channel* (canal). Le *channel* est l'entité qui réalise effectivement la communication entre les tâches. Il existe autant de *channels* que de types d'interface réseau physique devant être utilisés par LANDA dans l'infrastructure matérielle. Un *channel* peut être assimilé à un *driver* (ou pilote) réseau.

L'ensemble de la bibliothèque repose sur le concept de mémoire partagée. Ceci permet d'optimiser les performances de communication entre les tâches exécutées sur les processeurs se partageant la même mémoire.

VI.4 Architecture de communication

VI.4.1 Introduction

Les architectures classiques, telles que les piles TCP/IP (TCP/IP *stack*) des systèmes d'exploitations permettent un usage très général des liens de communication : multiplexage, généricité de la pile de communication vis-à-vis de la couche physique sous-jacente (*ethernet, token-ring, myrinet, ...*). Cette généralité engendre une architecture logicielle qui impose souvent des manipulations complexes des données entre le moment de leur réception dans la mémoire de l'interface physique et celui de leur réception dans la mémoire du programme utilisateur. Cette complexité engendre des latences de communication non négligeables.

Un effort a été fait pour réduire les temps de latence de la couche Media dans le but d'optimiser les performances des communications faites par les *channels*. En effet, ce sont les *channels* qui réalisent effectivement le travail de communication. Chaque carte d'interface nécessite une implémentation propre (un pilote) venant s'insérer sous la couche Media.

Le mode de communication idéal entre deux tâches est l'écriture directe en mémoire : une tâche (émetteur ou récepteur) alloue une zone de mémoire, l'émetteur copie ou construit les données à transmettre puis passe l'adresse de la zone aux tâches réceptrices. Ceci suppose une sémantique particulière de communication entre tâches :

- dans le cas d'une ou plusieurs tâches réceptrices : « réception en lecture seule », ou
- dans le cas d'une seule tâche réceptrice, « émission et libération simultanées » pour la tâche émettrice.

Ainsi avec une contrainte supplémentaire comme l'allocation de la mémoire par le récepteur, une réception de type *gather* (réception multiple) peut se faire sans aucune copie : la tâche émettrice place son message dans la mémoire de la tâche réceptrice (figure 91). Ce principe peut être étendu à la communication distante, car les interfaces physiques sont de plus en plus souvent capables de transférer les messages de mémoire à mémoire (*DMA Transfert* : *Direct Memory Access*, Transfert par accès direct à la mémoire).

Pour être efficace, il faut donc que les tâches puissent se partager la mémoire sur une machine

multi-processeurs, ou être capables d'atteindre directement la mémoire des tâches distantes. Dans le second cas, les contraintes sont imposées par la carte d'interface et son interface de programmation.

Les cartes d'interfaces peuvent également poser des problèmes techniques, comme par exemple ne pouvoir être utilisées que par un seul processus dans une machine. Ce cas apparaît avec les cartes d'interfaces expérimentales (comme ce fut le cas avec certains équipements [MPC]). Il faut dans ce cas centraliser les communications dans un seul processus et distribuer les messages aux autres processus, sans copie si possible, via la mémoire partagée.

VI.4.2 La communication locale

VI.4.2.1 Partage de mémoire

Le système d'exploitation offre plusieurs possibilités pour partager la mémoire entre plusieurs processus. Une solution classique et efficace est d'utiliser le concept de *thread*, un *thread* étant un processus léger partageant sa mémoire avec d'autres *threads*. Cette solution ne peut pas être retenue car elle engendre trop de contraintes liées au système d'exploitation, ce qui rend l'implémentation de la librairie MPI impossible au dessus de la librairie centrale LANDA.

Dans les systèmes de type Unix, d'autres mécanismes sont fournis pour permettre le partage de mémoire : les IPC-SHM (*Inter-Process Communication - SHared Memory*) ou la fonction *mmap (memory map)*. Ces mécanismes permettent à plusieurs processus normaux (également appelés processus lourds) d'utiliser le même segment de mémoire partagée.

La couche media de LANDA et ses couches sous-jacentes, et plus généralement l'ensemble de la bibliothèque sont basés sur la communication locale en mémoire partagée.

VI.4.2.2 Allocation en mémoire partagée

L'allocation dynamique en mémoire partagée est réalisée par une fonction protégée contre les accès concurrents. Cette fonction est basée sur l'allocateur *malloc* (*Memory ALLOCator* : allocateur dynamique de mémoire) proposé par Doug Lea [LEA96] (appelé aussi *dlmalloc* ou *Doug Lea malloc*) et utilisé mondialement dans la librairie standard des environnements GNU [GNU].

L'allocation dynamique d'une zone en mémoire partagée peut être effectuée à tout moment par l'ensemble des tâches partageant cette mémoire. Pour chaque zone allouée, un compteur du nombre global de références à cette zone et utilisé pour gérer correctement les libérations.

VI.4.2.3 Synchronisation

Les mécanismes de synchronisation entre tâche constituent une partie critique de la communication si l'on veut obtenir des temps de latence minimaux. La synchronisation peut être réalisée de plusieurs manières différentes. Nous avons retenu les mécanismes utilisant la notion de sémaphore, à la fois performants, assez génériques pour l'allocation de ressources limitées et permettant de réaliser des zones d'exclusion mutuelle très simplement.

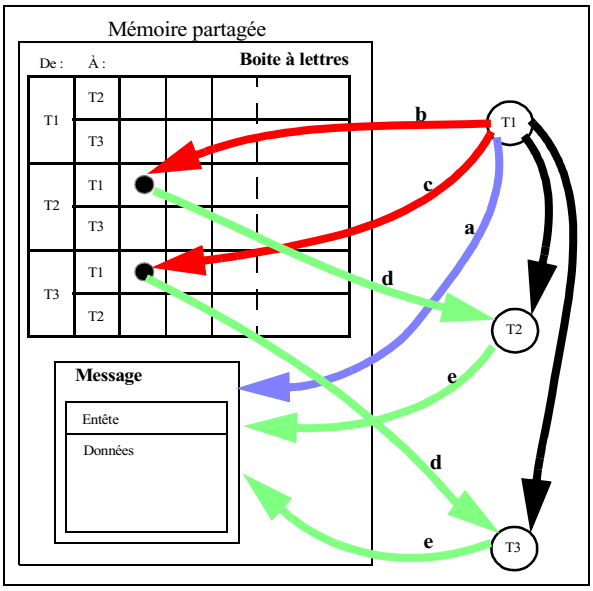
Pour réaliser des opérations synchronisées, les calculateurs offrent toujours un mécanisme de synchronisation, dont la sémantique assure l'exécution atomique d'une instruction faisant un accès mémoire en lecture suivi d'un accès en écriture. L'utilisation de ce mécanisme est la plus efficace possible, parce qu'elle offre un temps de latence nul (le calculateur est débloqué au moment même

où l'information est disponible). La contrepartie est que le processeur occupe tout son temps à ne faire que cette attente. Ces sémaphores, appelés également sémaphores à attente active (ou *spin-lock*), sont donc à utiliser dans le cas où le nombre de tâche à exécuter sur une machine n'excède pas le nombre de processeurs.

D'autres types de synchronisation sont réalisés par le système d'exploitation lui-même. Une tâche devant attendre un point de synchronisation particulier est retirée de la file d'attente des processus actifs jusqu'à ce qu'elle soit débloquée. Ce mécanisme permet de ne pas gaspiller la puissance du calculateur pendant le temps d'attente (celui-ci s'occupe uniquement des tâche actives), mais engendre un temps de latence (réveil de la tâche) plus long. C'est ce mécanisme qui est utilisé dans le standard IPC-SEM (*IPC SEMaphores*).

VI.4.2.4 Mécanisme de transfert de message en mémoire partagée

Figure 91: **Transfert de message en mémoire partagée**



La figure 91 illustre le mécanisme utilisé dans le noyau de LANDA pour le transfert de messages via la mémoire partagée.

La mémoire partagée est allouée par pages de taille fixe à la demande. Une zone spéciale de cette mémoire est utilisée pour la boite à lettre. Celle ci contient pour chaque couple (tâche source, tâche destination) une liste de pointeurs sur les messages en attente de réception.

Dans cet exemple, l'entête et les données sont regroupées dans le même message. La figure illustre le cas d'une tâche T1 qui envoie un message à deux tâches locales T2 et T3 en lecture seule (ce qui permet le transfert du message sans copie) : T1 crée le message (a) puis signale l'existence de ce message à T2 (b) et T3 (c). T2 et T3, qui étaient alors en attente de réception sont débloquées et peuvent lire, dans la boite à lettre (d), la référence vers le message créé par T1 (e).

Cet exemple illustre un schéma d'échange très simple et très performant : seuls des pointeurs sont échangés, et aucune recopie n'est effectuée. La sémantique nécessaire est cependant la plus contraignante : le message ne peut plus être modifié par la source après l'envoi, et ne peut pas être modifié par les tâches réceptrices après réception.

VI.4.2.5 Performances

Dans certains cas de figure (dépendants de l'application), il peut être nécessaire de copier le message. La figure 92 illustre un test de performance en mémoire partagée, en fonction de la taille des messages échangés.

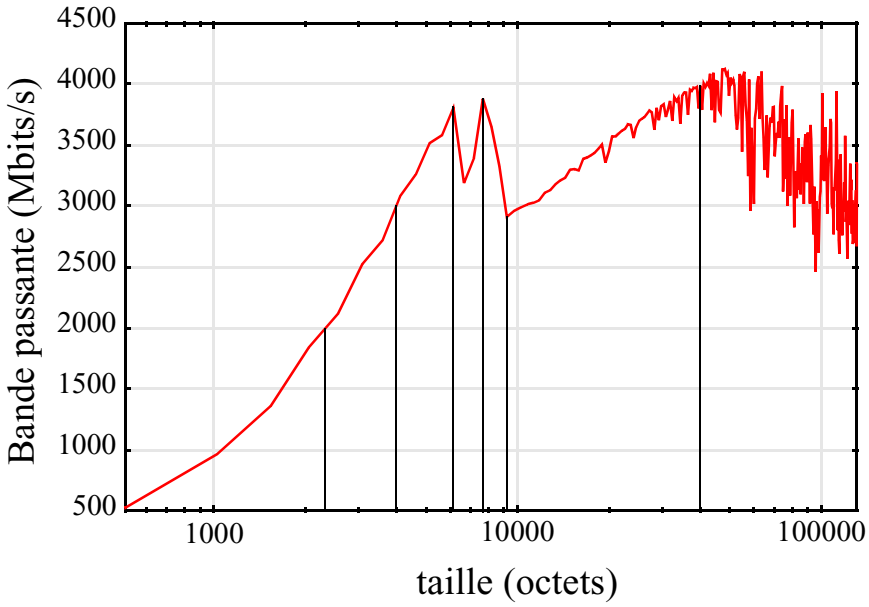
Ce test de performance a été effectué sur une PC bi-processeur (deux processeurs Intel P3 à 450Mhz). Il consiste en un échange de message de type « ping-pong » entre deux tâches, avec allocation (utilisant l'allocateur distribué), une copie, et libération d'un message en mémoire partagée lors de chaque échange.

La bande passante maximum est de 4GBits/s pour des messages de taille plus grande que 40Ko, de 3Gbits/s à partir de 4Ko, et la demi-bande passante est atteinte avec des messages de taille 2Ko environ.

La latence du mécanisme (allocation et libération comprise) est inférieure à 4µs (mesurée pour des messages de 1 octet).

Nous n'illustrerons pas de test de performances utilisant les mécanismes de transfert sans copie parce que la courbe ne représenterait pas les performances de la bibliothèque. En effet, il faut une latence maximum de 4µs pour transférer une adresse mémoire d'une tâche à une autre. Ce temps ne dépend pas de la taille des données, et la courbe n'aurait aucun sens. Le courbe serait approximativement une droite de pente 1/4µs.

Figure 92: **Bande passante d'un Ping-pong 1-copie (SHM + spinlock)**



VI.4.2.6 Implémentation

Une interface logicielle unique (API) pour les couches supérieures concernant les mécanismes d'allocation et de libération de la mémoire partagée, ainsi que la création, la suppression, et l'utilisation de sémaphores est nécessaire pour le noyau de communication LANDA. Une telle interface permet d'utiliser de façon transparente les mécanismes sous-jacents, et permet de choisir au moment de l'exécution les implémentations les plus efficaces (sémaphores à attente active ou passive, mémoire partagée IPC ou *mmap*...) en fonction des possibilités du système et de l'environnement matériel.

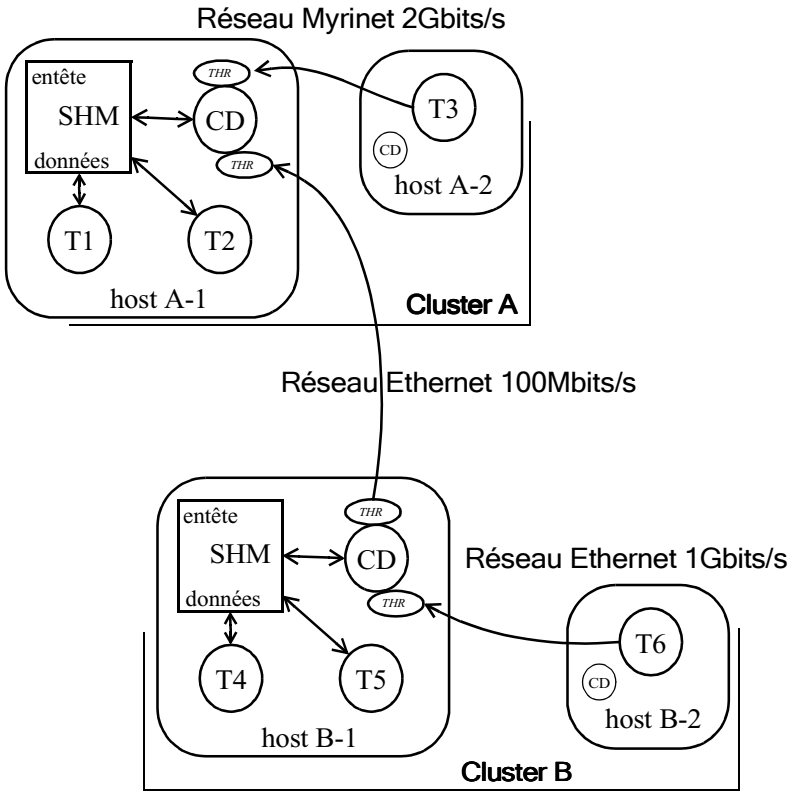
VI.4.3 Principes pour la communication distante

Pour faciliter la diffusion d'un message (une tâche vers plusieurs) et l'utilisation des pilotes logiciels pour des interfaces de communication spécifiques, nous utilisons un ou plusieurs CD (*Communication Daemon* : serveur de communication) sur chaque noeud du réseau. Le CD est une tâche particulière (que l'on peut appeler tâche système)

Le rôle principal du CD est de centraliser les réceptions de messages via le ou les réseaux. On peut distinguer plusieurs cas (figure 93) :

- Entête et données sont stockées sur le calculateur hébergeant la tâche réceptrice. Ce calculateur est directement atteignable, ou ne l'est pas. Dans ce dernier cas, les CD intermédiaires sont alors utilisés comme des routeurs pour faire suivre le message jusqu'au CD de la tâche réceptrice,
- L'entête est envoyée sur le calculateur de la tâche réceptrice, mais les données sont stockées ailleurs. Les données sont rapatriées chez le récepteur lorsque celui-ci effectue sa requête de réception. Là encore, si le CD du récepteur n'est pas directement atteignable par le calculateur envoyant l'entête ou les données, il faut utiliser les CD intermédiaires pour les acheminer.

Figure 93: **Exemple de communication distante avec LANDA**



Une tâche peut utiliser plusieurs *channel* simultanément pour émettre et recevoir des messages. Afin d'améliorer les performances en communication et compte tenu des contraintes imposées par le système, il est beaucoup plus efficace, pour une tâche donnée, d'attendre les messages sur un seul canal de communication.

La raison principale de cette restriction est qu'attendre un message sur plusieurs canaux en même temps implique que

- Soit le système offre un moyen de le faire efficacement. Le seul moyen efficace d'utiliser un système d'exploitation de type Unix pour attendre un message distant sur de multiples interfaces est d'utiliser la couche de communication standard (les «*socket*»). Mais c'est justement pour des raisons de performances que nous ne le faisons pas.
- soit les différents canaux doivent être interrogés tour-à-tour (*polling*), et cela ne permet pas de garantir une latence suffisamment basse.

L'attente de messages sur un seul canal de communication est un moyen efficace de contourner les limitations imposées par le système. On peut distinguer deux cas :

- Sur une machine mono-processeur, une tâche émet et reçoit les messages en utilisant le channel adapté à l'interface réseau utilisé.
- Sur une machine multi-processeur, la communication par mémoire partagée est privilégiée parce qu'elle est la plus efficace entre les tâches locales. Les tâches utiliseront donc la mémoire partagée pour recevoir des messages (figure 93). Pour envoyer un message, elles peuvent, au choix, utiliser un channel adapté à l'interface réseau, ou utiliser le CD local (via la mémoire partagée) pour envoyer un message à une tâche distante.

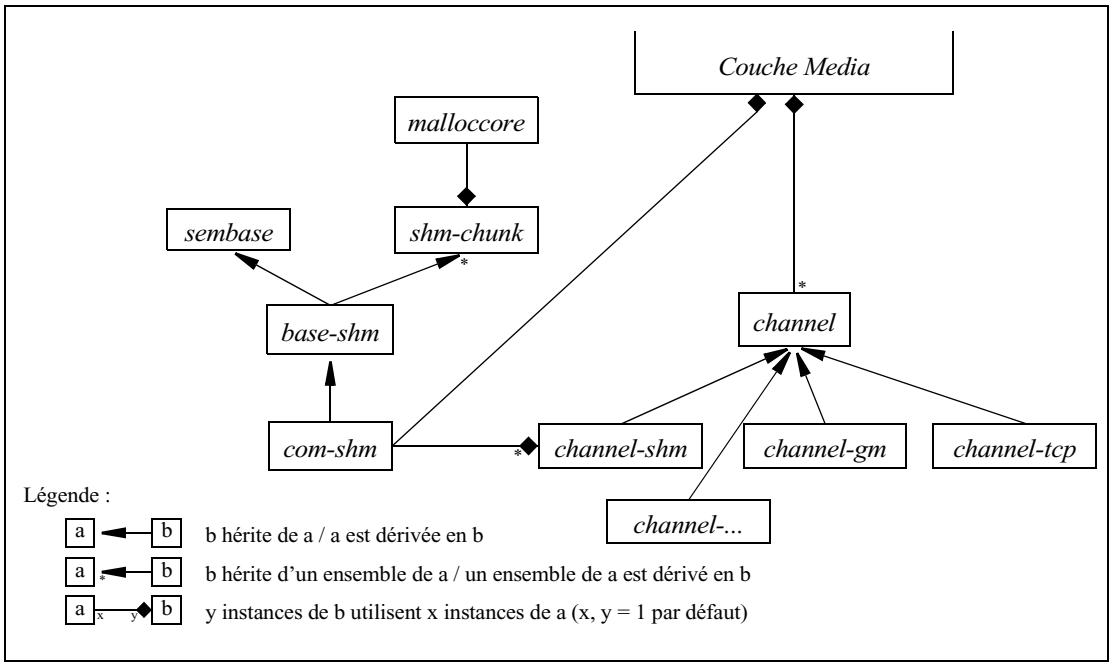
Sur une machine multi-processeur, ce sera donc le CD qui se chargera de recevoir les communications distantes. Or nous avons vu qu'il est peut efficace d'attendre sur de multiples interfaces. C'est pour cette raison que nous utilisons des *threads* dans les CD. Les *thread*s ne sont pas utilisables dans les tâches, mais cette restriction n'est pas étendue aux CD. Un *thread* sera donc créé dans le CD pour chaque nouveau canal de communication distant à utiliser. Les *threads* sont un moyen usuel pour contourner les contraintes d'attente multiples imposées par le système.

VI.4.4 Structure logicielle

Le noyau de communication de LANDA est écrit en langage C++. La hiérarchie des classes de communication est présenté en figure 94 :

- *mallocore* : Cette classe correspond à l'allocateur distribué. Elle est protégée contre les accès concurrents (dans un environnement SMP),
- *shmchunk* : Cette classe utilise la classe mallocore et réalise l'interface entre elle et le noyau de communication. Elle permet d'allouer et libérer les zones de mémoires contenant les messages,
- *sembase* : Cette classe implémente les primitives génériques de gestion des sémaphores pour la synchronisation,
- *base_shm* : Cette classe regroupe les fonctions de synchronisation et une liste de segments de mémoire partagée, et permet de les utiliser via une interface simplifiée et adaptée au noyau de communication,
- *comm_shm* : Cette classe étend la précédente en ajoutant la gestion de la boîte à lettre (décrite en figure 91),
- *channel* : Cette classe n'est qu'une spécification de fonctions virtuelles pour les transferts de données point-à-point. C'est le point d'entrée pour la couche Media du noyau de communication,
- *channel_shm*, *channel_tcp*, *channel_gm*, ... sont des classes spécifiques à une interface de communication. Elles implémentent les fonctions définies dans la classe *channel*.

Figure 94: **Hiérarchie des classes du noyau de communication LANDA**



VI.5 Utilisation de la couche Media

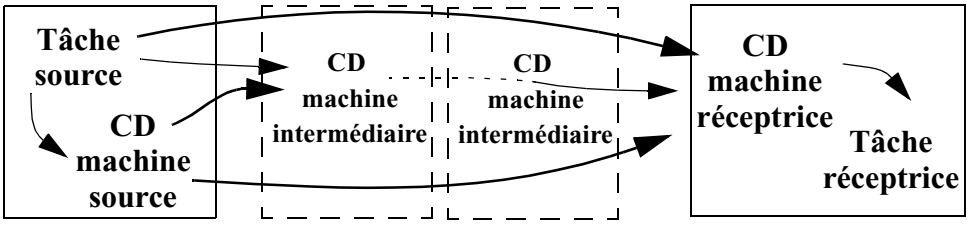
VI.5.1 Mode de communication

La couche channel (canal) est utilisée pour faire des transferts de message point-à-point. La couche Media se place à un niveau au dessus. Elle permet, en utilisant les canaux, la communication multi-point et le transfert (*forward*) de messages.

Le principe général adopté pour la communication entre tâches est que le message soit envoyé de la tâche source directement au CD de la tâche destinataire, ou en passant par le CD local (figure 93). Pour des raisons de performances, une tâche ne sera en attente de message que sur le channel-shm (la mémoire partagée). C'est donc son CD qui recevra le message si celui-ci arrive par un canal différent. L'envoi d'un message peut cependant utiliser n'importe quel canal utilisable par une tâche (selon les contraintes logicielles et matérielles). Le cheminement du message peut donc être de l'un des trois types suivant (figure 95) :

- La tâche source envoie son message directement au CD de la tâche destination (il peut être atteint par le canal utilisé),
- La tâche source envoie le message à un CD intermédiaire (un relais) permettant d'atteindre de proche en proche le CD de la tâche destinataire,
- La tâche source envoie le message à son CD (par la mémoire partagée) qui se charge de le faire suivre au CD distant, soit directement soit par des CD intermédiaires.

Figure 95: **Routage des messages dans la machine virtuelle LANDA**



VI.5.2 Les canaux de communications (channel)

VI.5.2.1 La mémoire partagée (channel-shm)

La mémoire partagée est liée au noyau de communication. Les mécanismes de transferts de messages entre tâches au sein d'un calculateur ont déjà été vus.

VI.5.2.2 TCP (channel-tcp)

Description

Le protocole de transport TCP est tellement répandu qu'il constitue un moyen standard pour atteindre une tâche distante. Il s'appuie sur le protocole IP (le protocole Internet, niveau réseau) universellement utilisé.

Pour cette raison, la plupart des constructeurs de cartes d'interfaces haute-performance fournissent un pilote d'adaptation du protocole IP sur leur architecture. Ainsi le protocole IP (donc TCP) est utilisable par exemple sur les réseaux Myrinet. Il faut cependant noter que toute émulation du protocole IP est possible au prix d'une certaine perte de performances. C'est pour cette raison que les environnements d'exécution distribuée n'utilisent de préférence pas les couches IP pour le transfert de messages à haut débit.

Dans le noyau de communication LANDA, un canal TCP est cependant nécessaire pour plusieurs raisons :

- Etablir un réseau d'administration et de contrôle. En effet, dans les grappes de PC ou de stations de travail, un réseau secondaire généralement de type ethernet / IP est installé afin de permettre l'administration des calculateurs. Ainsi, ce trafic de contrôle ne gêne pas le trafic utile circulant sur le réseau haut-débit,
- Permettre l'utilisation des calculateurs n'ayant que le protocole IP disponible pour la communication. En effet, souvent les calculateurs ne sont connectés au réseau que par des cartes d'interface de type ethernet et le protocole IP.

Mécanismes de transfert de messages avec TCP

Un thread serveur TCP est installé sur le CD. Les tâches utilisateur n'ont qu'un seul canal de réception, celui-ci étant généralement la mémoire partagée. Pour émettre un message, une tâche peut soit ouvrir une liaison TCP directe avec le CD de la tâche distante, ou utiliser son propre CD via la mémoire partagée. Le CD local effectuera alors lui-même une connexion TCP avec le CD de la tâche

distante.

Cependant, dans le cas où seule une tâche s'exécute sur un calculateur (la mémoire partagée devenant inutile), cette tâche peut recevoir des données en utilisant directement le canal channel-tcp.

VI.5.2.3 Myrinet (channel-gm)

Description

GM (*Glenn's Messages*) est un pilote et une bibliothèque (API) pour le passage de message sur réseau Myrinet [MYR][BOD95][PRY98]. En utilisant des cartes Myrinet de type Lanai4, le système fourni une bande-passante de 560 Mb/s et une latence de 38µs entre deux calculateurs bi-PentiumII-233Mhz. Les caractéristiques de GM comprennent :

- accès concurrents et protégés au niveau utilisateur pour l'interface Myrinet,
- livraison des messages de façon fiable et ordonnée (FIFO),
- possibilité d'utilisation à large échelle (des milliers de nœuds),
- utilisation très réduite des processeurs,
- deux niveaux de priorité de messages : normale et haute (pour les messages de contrôle)

L'émission et la réception sont soumises à la possession de jetons d'autorisation. L'envoi d'un message avec une priorité donnée peut être réalisé seulement si l'expéditeur possède un jeton pour le port et la priorité en question. Lorsqu'un message est envoyé, le jeton correspondant est libéré. Comme les envois ne sont pas bloquants, l'émetteur reçoit un événement signifiant que le message a été effectivement envoyé et que la zone mémoire contenant le message à envoyer peut être libérée. La réception d'un message d'une certaine taille et d'une priorité donnée peut être réalisée si le destinataire possède un jeton adapté.

Mécanisme de transfert de messages dans GM

L'utilisation de la librairie GM pour Myrinet nécessite que la zone de mémoire utilisée soit validée par le système d'exploitation comme de la mémoire non *swappable* (ne pouvant pas être stockée temporairement sur un disque dur) et utilisable par le transfert de type DMA (*Direct Memory Access* : Accès Direct à la Mémoire). Nous avons opéré en ce sens une modification dans le pilote GM pour linux afin que les segments de mémoire partagée (mémoire utilisée pour les transferts dans LANDA) soit acceptés par GM. Cette modification a été reprise dans la version officielle du pilote.

L'API GM impose que des zones de mémoire libres soient réservées à l'avance pour la réception de messages. Chaque zone de mémoire de «taille» n à réserver permet d'envoyer et recevoir des messages de taille comprise dans $]2^{n-1}..2^{n}]$ octets. La documentation conseille de réserver au moins deux zones pour chaque valeur de n.

Nous avons choisi, pour la librairie LANDA, de pré-réserver des zones pour $n < 16 = n_{grand}$. Cette valeur n'est pas figée. Elle doit être la plus grande possible, mais pas trop grande pour ne pas encombrer toute la mémoire partagée. La quantité de mémoire pré-réservée est $2^{n_{grand}+1}$ octets, ou $2^{n_{grand}-19}$ Mo. L'envoi d'un message est alors effectué comme suit :

- Si le message à envoyer est de «taille» inférieure à n_{grand}, alors le message est directement envoyé au récepteur. Lorsque celui-ci est notifié par GM de la réception d'un tel message, une nouvelle zone libre de même taille est allouée pour une réception future.

• Si le message à envoyer est de «taille» au moins égale à n_{grand}, alors un premier message de contrôle est envoyé au récepteur afin de le notifier de l'arrivée imminente d'un message de taille m, suivi du message lui-même. À la réception du message de contrôle, le récepteur effectue une allocation de taille correspondante, et peut alors recevoir le message de taille m.

Cette méthode de transfert est très efficace pour les petits messages (de «taille» inférieure à n_{grand}), un peu moins pour les plus gros messages (pas de transfert initial pour les messages «courts»; un transfert initial très court est nécessaire pour les messages plus «longs»). La valeur n_{grand} peut-être ajustée en fonction de la mémoire partagée disponible sur les calculateurs.

Cette méthode pose cependant un problème si le récepteur a besoin de recevoir le message à une adresse particulière de son choix. En effet, GM reçoit les messages dans une des zones préallouées, et n'en allouer qu'une seule n'est d'une part pas conseillé par les concepteurs de GM, et d'autre part pas efficace en cas de réception multiple ou en rafale (une seule réception à la fois ne pourrait se faire).

Pour régler ce problème, GM propose une fonction d'envoi direct en mémoire distante. Cette méthode de transfert ne peut être initiée qu'à partir du récepteur : Lorsque l'utilisateur fait une demande de réception sur une adresse précise, le récepteur déclare cette adresse à GM, puis l'envoie à l'émetteur, le message est alors transféré et réceptionné directement à l'adresse voulue. Ce mécanisme implique donc un envoi d'un message très court avant le transfert, mais ne nécessite pas de recopie dans la mémoire du récepteur. Une limite de taille peut être déterminée empiriquement pour déterminer la taille de message qui donne un temps de recopie (en utilisant la première méthode de transfert) égal au temps de latence introduit par le message court précédant la réception à une adresse prédéfinie. Cette taille dépend de l'ensemble du système (matériel, logiciel système).

Boucle de réception d'événement

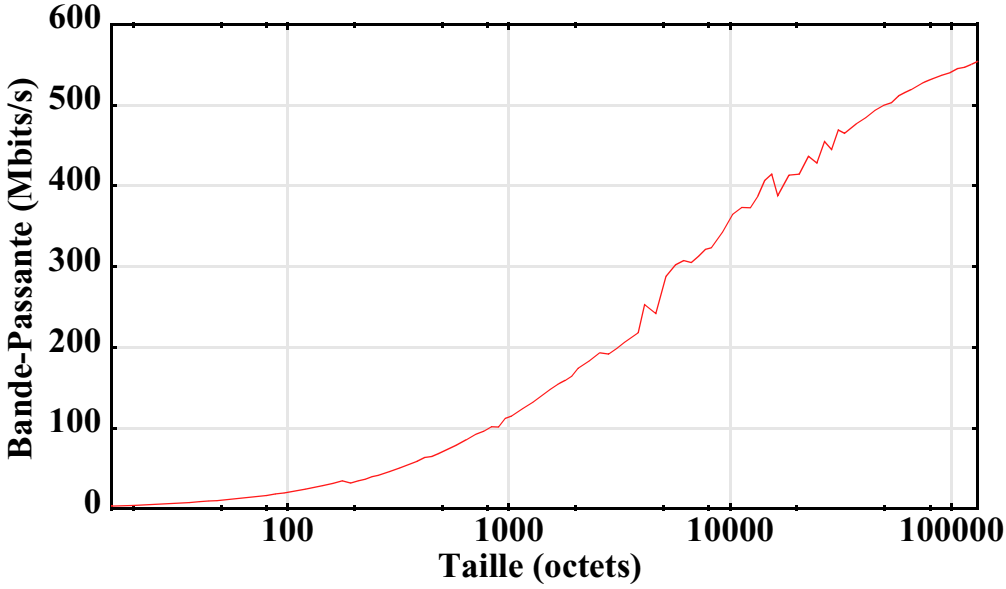
Un récepteur GM doit en permanence traiter des événements provenant du pilote de matériel. Une des fonctions de LANDA étant d'être capable de retransmettre des messages (*forwarding*), la réception ne peut se faire au niveau de l'application utilisateur. En effet, celle-ci n'effectue que des appels sporadiques à la librairie de communication et retarde ainsi les retransmissions. Cette boucle de réception a donc été implémentée dans le CD sous la forme d'un *thread* (les *threads* ne sont pas autorisés dans les applications clientes mais ne posent pas de problèmes dans un CD, figure 93).

Performances

Ce test est un ping-pong entre deux tâches utilisant le réseau Myrinet et le channel-gm. Les tests ont étés effectués en utilisant des cartes Lanai4.3 PCI 32bits/33Mhz sur deux machines Intel bi-PIII@450Mhz. Un test équivalent utilisant GM de façon native donne les performances suivantes : 610Mbits/s de débit maximum, 22µs de latence (mesurée pour des messages de 1 octet).

Le test utilisant channel-gm donne les résultats suivants : 570Mbits/s de débit maximum, 285Mbits/s atteints avec des paquets de 5Ko, et 25µs de latence pour des paquets inférieurs à 16Ko, 50µs pour des paquets de plus de 16Ko (ce seuil est paramétrable). Ce test inclus allocation et libération de messages en mémoire partagée à chaque envoi et réception.

Figure 96: **Ping-pong sur *channel-gm***

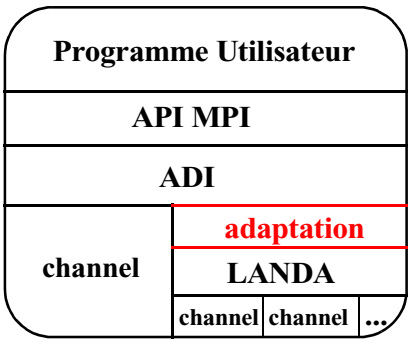


VI.6 Adaptation de MPICH sur l'environnement LANDA

VI.6.1 Introduction

MPI (*Message Passing Interface* : Interface de passage de message) est la spécification d'une interface de programmation portable permettant le transfert de messages entre tâche en utilisant un réseau d'interconnexion. L'API MPI est un standard largement utilisée, de la plus simple des applications distribuées aux applications massivement parallèles sur calculateurs dédiés. Plusieurs implémentations de cette norme existent, certaines étant payantes.

Figure 97: **MPICH sur le noyau de communication LANDA**



Afin de pouvoir utiliser LANDA pour effectuer les communications nécessaires aux applications utilisant la norme MPI, nous nous sommes intéressés à l'implémentation MPICH (*MPI-CHannel*) de cette norme. En effet, le code source de cette implémentation est librement accessible et modifiable, et propose également une architecture logicielle intéressante puisque l'implémentation des fonctions utilisateurs (la norme) et la partie communication sont clairement séparées. Il est ainsi possible de ré-écrire le programme de communication de façon presque indépendante du reste de la librairie.

La figure 97 illustre l'architecture logicielle de MPICH ainsi que le principe d'adaptation de LANDA dans la partie communication. Le programme utilisateur repose sur la norme MPI. La couche d'implémentation de cette norme utilise la couche ADI (*Abstract Device Interface* : Interface abstraite du matériel) définie par MPICH pour effectuer les transferts de messages entre les tâches d'une application. La couche ADI repose ensuite sur une couche channel propre à MPICH. L'implémentation de cette couche est fortement liée au type d'interface de communication devant être utilisé. La plus utilisée est appelée *ch_p4* et permet l'utilisation d'un réseau IP.

MPICH est capable d'utiliser simultanément la mémoire partagée avec des canaux de communication externes (*ch_p4*, utilisant TCP/IP ou *ch_gm* utilisant un réseau Myrinet). Cependant, il n'est pas possible d'utiliser ces implémentations sur deux ou plusieurs réseaux d'architectures différentes simultanément. LANDA n'a pas cette limitation.

Couche ADI : mécanismes d'envoi et de réception

La couche ADI de MPICH propose plusieurs protocoles de communication que le canal peut implémenter ou non. Ils sont au nombre de trois :

- *short* : Le message est assez court pour être contenu dans une requête (un message de contrôle). Il est envoyé en une seule fois, puis copié dans la zone mémoire allouée par l'utilisateur. C'est le protocole le plus efficace (les messages sont très courts) qui ne nécessite qu'une seule communication sur le réseau,
- *eager* : Le message peut être envoyé par l'émetteur alors que le récepteur (le programme utilisateur) n'a pas encore fait la requête de réception. Le canal peut alors recevoir le message malgré tout, et le faire remonter à l'utilisateur lorsque la demande en est faite C'est un mécanisme en deux temps : l'émetteur effectue la requête d'émission, puis l'émission elle-même. Le récepteur reçoit tout d'abord la requête, alloue la mémoire de réception, puis reçoit le message. Ce message sera copié plus tard dans la mémoire allouée par l'utilisateur lorsqu'il la requête a été effectuée.
- *rendezvous* : Les deux extrémités doivent se mettre d'accord sur le moment de l'envoi effectif d'un message. La mise en place du rendez-vous s'effectue en deux temps : une requête de réception doit être envoyée vers l'émetteur, et une requête d'émission doit être envoyée vers le récepteur. Une fois que les deux extrémités se sont mises d'accord (*handshake*), la transaction est faite : le message est reçu dans la zone de mémoire préallouée ce qui évite une recopie inutile. C'est un mécanisme en trois temps.

Adaptation de LANDA en dessous d'ADI de MPICH

L'adaptation de LANDA sous le niveau de la couche ADI de MPICH est en partie inspirée de l'adaptation du pilote GM de Myrinet pour MPICH [MYR]. Celle-ci est basée sur l'adaptation générique CH2 fournie avec le code source de MPICH.

L'adaptation de LANDA est basée sur les protocoles *short* et *rendezvous*. Le protocole *eager* n'est pas implémenté pour une raison assez simple : son fonctionnement peut être émulé par le protocole *rendezvous*, cela permet de réduire la complexité d'adaptation sans affecter les performances. Les protocoles implémentés sont :

short (bloquant): Ce protocole est très efficace car il n'effectue qu'un seul transfert, et ne nécessite pas de préparation du côté récepteur. De plus, c'est à peu de chose près le mécanisme utilisé pour transférer les requêtes de contrôle pour le fonctionnement de MPICH.

rendezvous (non bloquant): C'est le plus général et le plus complexe des trois protocoles. Il doit donc être implémenté pour permettre à toutes les primitives MPI de fonctionner correctement. Cependant sa réalisation dans les canaux LANDA n'est pas contrainte par le protocole général en trois temps décrit plus haut. La tâche à réaliser étant le transfert d'un message d'une adresse source à une adresse destination distante, peu importe la manière dont elle est réalisée. Par exemple, en mémoire partagée, une simple copie mémoire sera effectuée (la sémantique « envoi+libération de message », ou « réception en lecture seule » n'existe pas dans MPI ce qui ne permet pas le transfert

sans copie).
Par exemple dans channel-gm (Myrinet), un envoi de type *rendezvous* est réalisé à la manière d'un envoi de type *eager* (envoi du message de contrôle suivi immédiatement du message), et la réception est effectuée directement dans la zone allouée par l'utilisateur s'il a déjà effectué la requête de réception, ou dans une zone temporaire (allouée à la réception du message de contrôle) si la requête n'est pas encore faite; une copie sera alors nécessaire lorsque la requête de réception sera effectuée. Des optimisations plus poussées, selon les possibilités des cartes d'interfaces, peuvent conduire à utiliser des mécanismes d'écriture ou de réception directe en mémoire distante.

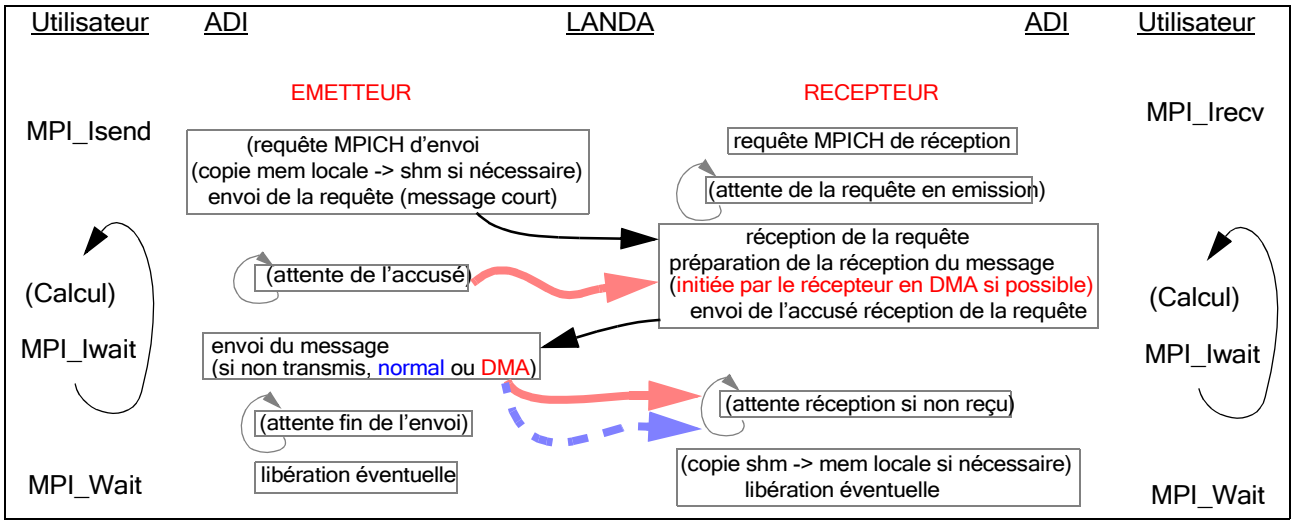
Algorithme général

La figure 98 présente l'algorithme général d'envoi et réception d'un message avec le protocole *rendezvous* non bloquant dans l'adaptation de l'ADI. Le cheminement général des données est détaillé.

Les messages courts sont représentés par une flèche simple, les données par une flèche épaisse en traits interrompus, et les données en accès direct à la mémoire par une flèche épaisse en traits pleins.

Plusieurs schémas sont possibles selon les capacités du channel de LANDA (qui dépendent directement du pilote bas-niveau de la carte d'interface). Ces capacités peuvent avoir un impact sur l'efficacité, notamment si l'algorithme utilisateur repose sur le recouvrement calcul / communication pour optimiser les performances du programme parallèle.

***Figure 98:* Algorithme général de l'adaptation de la couche ADI à LANDA**



Plus précisément, les fonctions d'envoi dans MPI sont bloquantes (*MPI_Send*) ou non-bloquantes (*MPI_Isend*). De même pour les fonctions de réception (*MPI_Recv*, *MPI_Irecv*). Pour tester si une requête est réalisée, la fonction utilisée est *MPI_Iwait* (retournant un code indiquant l'état de la requête). La fonction *MPI_Wait* bloque l'appelant jusqu'à ce que la requête soit réalisée. L'appel à *MPI_Send* est équivalent aux deux appels successifs *MPI_Isend*, *MPI_Wait*. De même, l'appel à *MPI_Recv* est équivalent à *MPI_Irecv*, *MPI_Wait*. La figure présente le fonctionnement de *MPI_Isend* et *MPI_Irecv* avec le protocole *rendezvous* non-bloquant.

Dans les couches LANDA, le transfert du message de l'émetteur vers le récepteur peut se faire de trois manières différentes. De la plus efficace à la moins efficace, si l'interface réseau le permet :

• Réception directe en mémoire à l'initiative du récepteur (première flèche épaisse, trait plein) :
Cette réception peut être effectuée par le récepteur dès que la requête de réception utilisateur est connue (*MPI_Irecv* transmet l'adresse de réception aux couches basses) et que la requête d'envoi, indiquant que l'émetteur tient le message prêt à être émis, est reçue.
Le temps d'exécution donné par le processeur aux différentes couches est minimal : il est critique, pour cet envoi/réception jusqu'à l'initiation de la réception en mémoire directe. Le transfert est ensuite de la responsabilité des cartes d'interface, et le processeur peut continuer d'exécuter le programme utilisateur sans intervention sur la communication. La latence de transfert (en temps d'occupation du processeur) est donc limitée à un seul message court (le second message court n'est pas nécessaire).

• Envoi direct en mémoire à l'initiative de l'émetteur (seconde flèche épaisse, trait plein) :
Certaines cartes d'interface ne peuvent pas initier une réception directe, mais peuvent effectuer un envoi direct. Dans ce cas, le contrôle du côté émetteur doit être régulièrement donné aux fonctions de communication (via un appel à *MPI_Wait* ou un appel régulier à *MPI_Iwait*) afin d'attendre ce message pour pouvoir initier le transfert direct en mémoire.
Le transfert initié, la communication n'est plus de la responsabilité du processeur. La latence de transfert (en temps d'occupation du processeur) est un envoi, un traitement distant, et une réception.

• Pas de transfert direct de mémoire (troisième flèche épaisse, trait interrompu) :
C'est le cas le plus général dans lequel le processeur s'occupe de tous les transferts (par exemple l'utilisation de la pile TCP/IP du système d'exploitation et de cartes ethernet standards). Le contrôle doit alors être très souvent donné aux couches de communication (via un appel à *MPI_Wait* ou un appel régulier à *MPI_Iwait*) afin que toutes les séquences décrites dans la figure soit complétées. Ce mécanisme implique généralement des recopies mémoires effectuées par les différentes couches du noyau du système traversées. Ceci se fait au détriment du programme utilisateur.

Performances

Le test présenté en figure 99 illustre un ping-pong entre deux tâches MPI, utilisant divers canaux :

• *ch_p4* : tcp/ip natif de MPICH

• *ch_shmem* : mémoire partagée native dans MPICH

• *ch_landa* : LANDA et mémoire partagée

Une zone mémoire est habituellement allouée par la fonction *malloc*. Si l'on utilise LANDA, il est préférable d'utiliser la fonction *landa_malloc* pour allouer les zones mémoire qui contiennent les messages MPI. Cette fonction retourne un pointeur dans la mémoire partagée, et permet ainsi d'éviter une recopie inutile pour le transfert des messages.

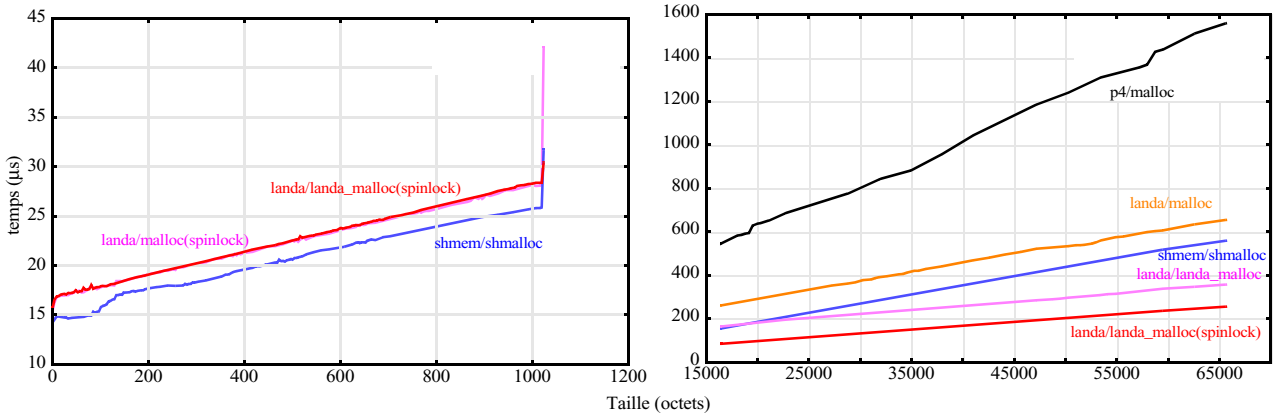
Les tests présentés sont nommés de la manière suivante : «adi»/«alloc». «adi» représente l'implémentation de la couche ADI qu'utilise le test : *landa* pour *ch_landa*, *p4* pour *ch_p4* (TCP/IP) et *shmem* pour *ch_shmem* (mémoire partagée de MPICH). «alloc» représente la fonction utilisée pour allouer dynamiquement la mémoire : *malloc* est la fonction standard d'allocation en mémoire locale, *shmalloc* est la fonction d'allocation associée à *ch_shmem* et *landa_malloc* est la fonction d'allocation de LANDA en mémoire partagée.

La synchronisation utilisée dans LANDA est réalisée par sémaphores systèmes (sémaphores

portables). Lorsque «spinlock» est indiqué, la synchronisation utilisée dans LANDA est réalisée par spin-lock (les sémaphores à attente active).

Les tests ont été réalisés sur un calculateur bi-processeur PIII à 450Mhz.

Figure 99: **Ping-pong sur MPICH-ADI-LANDA**



Tout d'abord, on peut remarquer que *ch_p4* est hors course. Bien que les deux tâches soient locales, passer par la pile TCP/IP du système est une perte de performance conséquente.
Pour les messages de petites tailles (messages courts utilisant l'algorithme *short* de MPICH), nous pouvons constater la latence introduite par les couches LANDA (sur la courbe de gauche) par rapport à la couche native de MPICH, qui est de 2 à 3 µs.
Pour les messages plus longs (protocole *rendezvous* non-bloquant), la couche LANDA semble être plus performante que le canal natif *shmem* de MPICH. Nous constatons également que l'utilisation des sémaphores à attente active joue un rôle très important sur la latence de communication, par rapport aux mécanismes plus conventionnels. L'impact de l'allocation en mémoire locale (landa) plutôt qu'en mémoire partagée (*landa_malloc*) est également important (environ 50µs de latence).

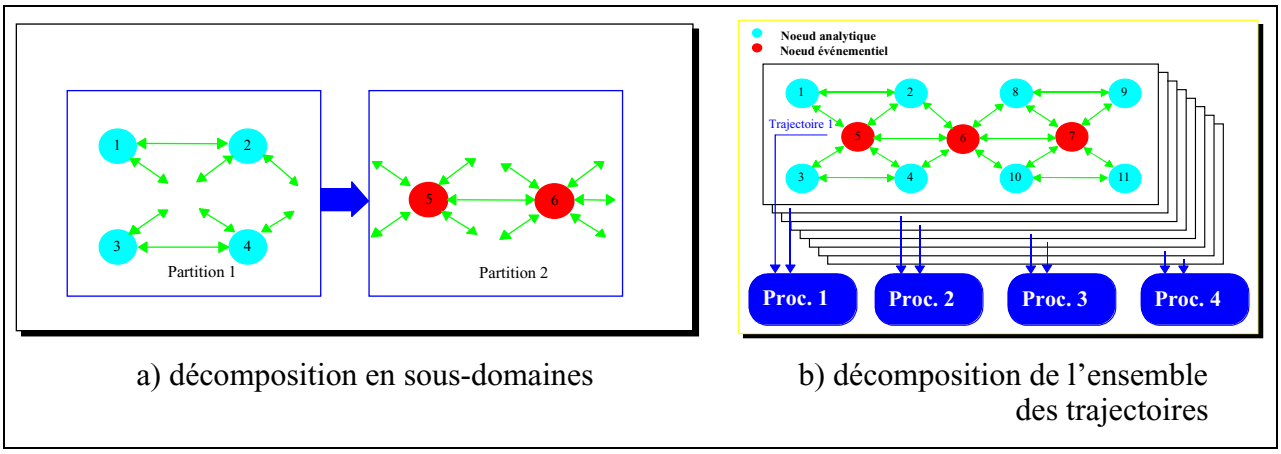
VI.7 Simulation distribuée

VI.7.1 La parallélisation

Nous avons montré dans les chapitre précédents que la simulation hybride pouvait apporter un gain considérable sur les temps de calcul par rapport à la simulation événementielle. Le parallélisme est une approche complémentaire pour réduire encore les temps de simulation. Nous avons parallélisé le noyau de simulation de manière à répartir la charge de calcul entre plusieurs processeurs avec des échanges d'informations statistiques entre les processeurs. Deux stratégies de parallélisation peuvent être utilisées simultanément (figure 100) : la décomposition du réseau en sous-domaines, et la décomposition de l'ensemble des trajectoires aléatoires en sous-ensembles.

La décomposition de l'ensemble des trajectoires (figure 100b) est adaptée au calcul du régime transitoire : Les simulations événementielles et hybrides réalisent des simulations indépendantes qui peuvent être totalement découplées les unes des autres pendant toutes la durée de simulation (simulation événementielle pure) ou entre deux (ou quelques) pas d'intégration des équations analytiques (simulation hybride). La répartition de charge de calcul sur les processeurs, mêmes s'ils n'ont pas tous la même puissance, est alors très facilement réalisable. En effet le rapport nombre de trajectoires sur puissance du processeur doit être le même sur tous les processeurs.

Figure 100: **Deux techniques de parallélisation**



a) décomposition en sous-domaines

b) décomposition de l'ensemble des trajectoires

La décomposition en sous-domaines (figure 100a) est plus adaptée aux

• calcul du régime stationnaire (simulation hybride),

• calcul du régime stationnaire ou transitoire (simulation analytique sur de très grand réseaux).

En effet, le calcul du régime stationnaire peut être réalisé sur chaque partition du réseau, indépendamment les unes des autres en utilisant les données d'entrées reçues de l'étape précédente. À chaque itération, les interfaces de chaque partition échangent leurs paramètres (moyenne, variance) d'entrée et de sortie. L'algorithme de convergence est alors un point fixe global. Il faut noter qu'une simulation événementielle complète pour chaque flot de bout en bout n'est pas réalisable dans ce cas, car il n'est pas question d'échanger les événements entre les tâches lors des étapes de communication sur le réseau d'exécution (les temps de communication seraient beaucoup trop longs). Il est cependant toujours possible de réaliser des statistiques (moyenne, variance des inter-arrivées) en sortie des interfaces des sous-réseaux et de les utiliser à l'entrée d'un autre sous-réseau avec des générateurs appropriés (c'est le principe de simulation hybride). Cette méthode de parallélisation est une approche de type gros grain.

En raison de sa rapidité, la parallélisation de la simulation analytique pure n'est intéressante que dans le cas de très grands réseaux. La méthode de parallélisation par décomposition en sous domaines consiste alors à échanger les paramètres de débit et de variance entre les interfaces à chaque pas d'intégration (régime transitoire, approche parallèle grain fin) où à chaque étape de l'algorithme du point fixe (régime stationnaire, approche parallèle gros grain).

En pratique, il est nécessaire d'avoir un modèle de réseau suffisamment complexe (nombre de noeuds, d'équations, d'événements) de façon à obtenir un programme parallèle efficace.

VI.7.2 Equilibrage de charge

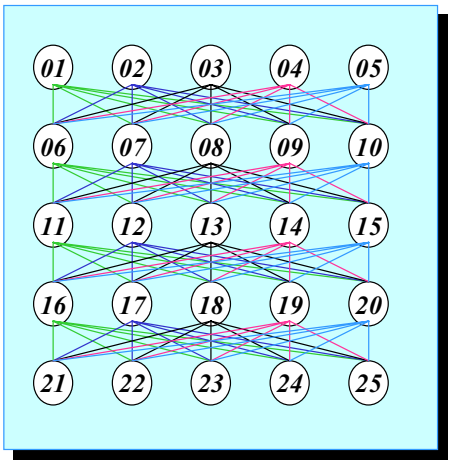
Nous avons vu que l'équilibrage de charge est une chose très facilement réalisable si l'on utilise la technique de décomposition des trajectoires.

La technique de décomposition en sous domaines est également facilement équilibrable pour le régime stationnaire. En effet chaque tâche peut calculer indépendamment le régime stationnaire de la partition qu'elle calcule, et prévenir ses « voisines » (au sens de l'interconnexion des partitions) une fois qu'elle considère avoir une bonne statistique. Le calcul du régime stationnaire peut être poursuivi (afin d'améliorer la statistique) jusqu'à ce qu'une tâche voisine ait également terminé son étape de calcul et l'échange des données est ainsi réalisé.

En revanche la technique de décomposition pose plus de problèmes en ce qui concerne le régime transitoire. En effet, l'échange des données doit intervenir à intervalles réguliers à cause de la structure même de la simulation. Pour que la parallélisation soit efficace, le synchronisme de cet algorithme impose que les charges de calcul de toutes les tâches à chaque pas d'intégration soient sensiblement égales.

VI.7.3 Résultats préliminaires

Figure 101: **Structure du réseau**

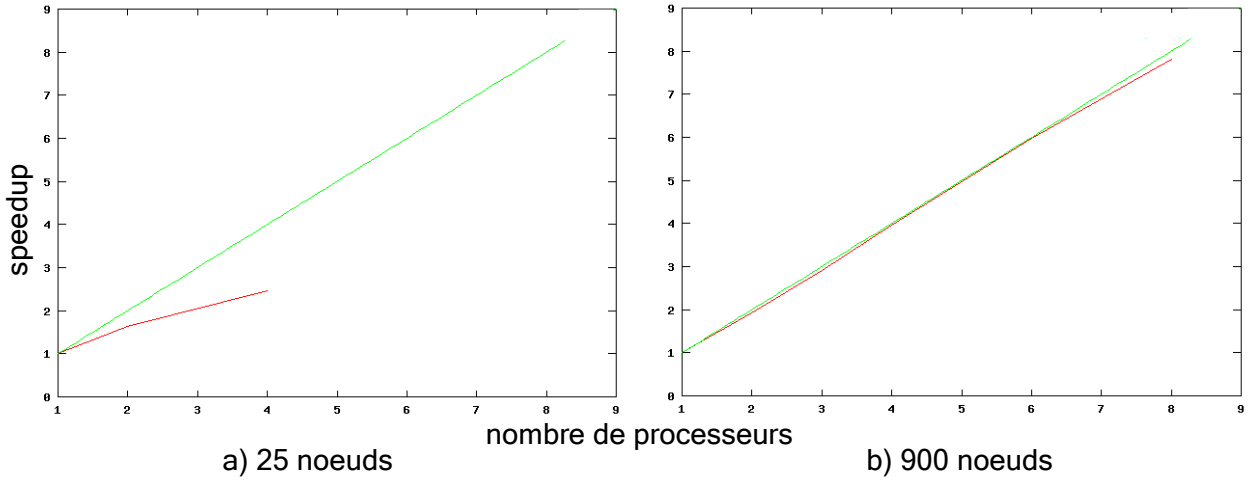


La première version du noyau de simulation développée dans le cadre du projet STAR1 est un simulateur général de réseaux de files d'attente. Il a été parallélisé, et les communications entre les tâche sont réalisées par passage de message. La bibliothèque de communication utilisée est MPI.

La figure 101 présente un réseau de 25 noeuds sur une grille de 5 par 5. Chaque noeud dans une ligne est connecté à tous les noeuds dans la ligne inférieure. En chaque noeud des quatre premières lignes, du trafic destiné à chacun des noeuds dans la ligne du bas est généré. Ce trafic est routé équitablement sur tous les chemins joignant la source à la destination. Il y a en tout 100 flots différents. Les flots suivent la loi de Poisson et les noeuds sont $M/M/1/\infty$.

Nous présentons ici le résultat de speedup (rapport temps de calcul sur nombre de processeurs) pour le calcul du régime transitoire événementiel. Le réseau est simulé pendant 50000 secondes, et le temps de simulation correspondant utilisant une seule tâche de calcul est de 550 secondes (processeur Sparc5 à 166Mhz). La figure 102a est la courbe de speedup (la courbe $y=x$ est également tracée sur le graphique). Le même test a été réalisé en utilisant la même structure de réseau mais cette fois-ci sur une grille de 30 par 30. La figure 102b correspond à la courbe de speedup. Ces simulations ont été réalisées en utilisant MPICH sur un réseau TCP/IP à 10Mbits/s.

Figure 102: **Speedup de la parallélisation**



La figure 102a montre que le temps de communication réseau peut ne pas être négligeable sur une petite topologie (le grain est trop fin). A l'inverse la figure 102b montre que la technique de parallélisation des trajectoires est très bonne quand la taille du problème augmente (grain plus gros).

VI.8 Conclusion

Les techniques de parallélisation représentent une alternative souvent utilisée pour réduire les temps de calculs des applications lourdes. Au moment où a débuté le projet LANDA-HSN, le concept de *cluster* puis de grille commençait à se développer. Les réseaux locaux haut-débits devenaient abordables pour de petites structures. L'objectif du projet LANDA-HSN fut de se donner la capacité d'utiliser le maximum des ressources, et permettre l'utilisation de l'ensemble du réseau hétérogène et haut-débit disponible. La tendance actuelle est à l'utilisation de grilles de calcul (clusters de clusters) de structures plus homogènes, constituées de calculateurs dédiés, réduisant ainsi les coûts de maintenance et facilitant le développement de grandes architectures. Malgré tout, l'objectif premier du projet LANDA-HSN reste d'actualité. Les réseaux locaux ne cessent pas d'évoluer de sorte que des systèmes d'architectures hétérogènes cohabitent en permanence.

La bibliothèque de communication présentée dans ce chapitre et destinée à être utilisée sur des réseaux locaux haut-débit propose un moyen efficace de fédérer l'ensemble des ressources disponibles. La possibilité d'utiliser plusieurs API et en particulier MPI, sa conception modulaire permettant d'intégrer efficacement tout nouveau media de communication en font un outil efficace pour les applications parallèles. Couplée à l'environnement LANDA et au système Network-Analyser, elle permet un rendement optimal des réseaux et des processeurs de bureau inutilisés. Dans une architecture de type grille, elle permet au sein d'une même application utilisateur d'utiliser l'ensemble des sous-réseaux haut débit disponibles.

Nous avons présenté l'architecture en couche de cette bibliothèque. Au niveau le plus haut nous retrouvons l'aspect multi-API utilisateur. La couche inférieure concerne la gestion de la machine virtuelle dont l'architecture a été détaillée. Cet ensemble repose sur la couche Media qui constitue le point d'entrée pour la communication logique entre les tâches de l'application parallèle. Cette couche est basée sur l'utilisation intensive de la mémoire partagée, et utilise des mécanismes systèmes efficaces permettant aux tâches exécutées dans la même mémoire d'un calculateur de communiquer de manière très performante avec des temps de latence très faible, permettant ainsi l'utilisation d'algorithmes parallèles à grain fin. Enfin, nous avons détaillé les mécanismes de communication distante que nous avons conçu pour deux media de communication (à notre portée!) « haut-débit » qui sont GM (Myrinet) et TCP.

Les mesures présentées ont montré qu'il est possible d'approcher les performances « brutes » des cartes d'interface ou des mécanismes de communication par mémoire partagée, tout en les utilisant dans un environnement complexe et en donnant une vision abstraite de l'architecture sous-jacente à l'utilisateur.

Enfin, nous avons, dans une dernière partie, explicité comment cet environnement d'exécution parallèle était utilisé pour réduire les temps de calculs du simulateur hybride, et présenté les techniques de parallélisation associées.

Conclusion

Le développement du réseau Internet et la diversification des services offerts en font un réseau extrêmement complexe. Le développement des connexions domestiques haut-débits, la vision du « tout IP » des opérateurs pour les réseaux de demain entraînera de grands changements dont on devine aujourd'hui à peine toute l'étendue. Un des aspects les plus complexes sera l'utilisation du réseau Internet pour transporter en masse des flux à qualité de service critiques tels que les communications téléphoniques ou la diffusion de films vidéo haute définition. La conception de tels réseaux nécessite le développement de méthodologies et d'outils de modélisation permettant d'évaluer de manière précise les performances de l'ensemble des ressources et des mécanismes utilisés. Etant donné la taille et la complexité des systèmes considérés, ces outils se doivent d'être eux-même très rapides pour offrir des temps de réponses acceptables.

Dans cette thèse, nous nous sommes concentrés sur l'étude de tels outils. Nous nous sommes tout d'abord attachés à la simulation des réseaux de files d'attente. La méthode généralement utilisée dans les logiciels d'évaluation de performance est la simulation événementielle. Cette méthode est efficace, car elle est générale et permet théoriquement une précision arbitraire. Cependant elle est très lourde et très lente pour de grands réseaux. Nous avons présenté la théorie du trafic différentiel qui permet d'évaluer le comportement moyen des éléments du réseau. Cette méthode est à la fois très précise est très rapide. Elle permet, au prix d'une faible complexité de calcul, de déterminer le comportement dynamique d'un réseau par intégration d'équations différentielles, ou de d'en évaluer le régime stationnaire par un algorithme de point fixe. Cependant le nombre des modèles analytiques disponibles est limité et ne permet pas de représenter l'ensemble des mécanismes réseaux que l'on veut simuler.

A partir de ce constat, nous avons élaboré le concept de simulation hybride. Nous proposons avec ce concept d'utiliser les deux modèles de simulation conjointement et de tirer parti de leurs avantages respectifs. La simulation hybride repose elle aussi sur la théorie du trafic différentiel, dans laquelle le comportement moyen des flot est déterminé à l'aide d'équations différentielles associées à chaque ressource. Ces équations expriment le fait que la variation du nombre moyen de clients (ou paquets) dans le système est la différence entre le débit entrant et le débit sortant. Les modèles analytiques fournissent des équations différentielles explicites. A chaque ressource ne disposant pas de modèle analytique assez précis est associé une équation différentielle implicite. Les débits instantanés d'entrée et de sortie intervenant dans cette équation sont obtenus par des simulations à événements discrets.

La simulation hybride consiste alors en l'interconnexion des équations implicites et des équations explicites : à chaque étape de simulation, les paramètres moyens de débit sont échangés entre les sous réseaux complexes (simulés par événements discrets) et les sous réseaux simulés par des équations. Ce principe de simulation donne de bons résultats et permet de réduire les temps de calcul par rapport à la simulation exclusivement événementielle. Un travail restant cependant à faire dans ce domaine est l'extension du type des paramètres échangés entre les différentes équations : un seul paramètre d'entrée, le débit, est utilisé par les files analytiques et correspond à un trafic Poissonien. Lorsque le trafic est sous- ou sur-variant, ce qui peut être le cas dans les réseaux multiservices, la simulation hybride permet également l'échange de la variance des inter-arrivées de paquets. Il faudra alors disposer de formules approchées tenant compte de cette variance.

Pour la simulation de réseaux multiservices, nous nous sommes intéressés aux sources de trafic multimedia et aux protocoles de transmissions utilisés dans le réseau Internet. A partir de modèles de

sources issus d'études de métrologie, nous avons pu construire des générateurs de paquets dont la distribution correspond à celle de sources réelles (non Poisonniennes) générées par des codecs (trafic voix, trafic video). Nous avons en particulier pu montrer que l'agrégation d'un relativement faible nombre de telles sources (typiquement une vingtaine) était quasiment équivalent à une source Poissonienne de même débit. Ceci est d'un intérêt considérable pour une modélisation hybride des réseaux : le réseau d'accès est simulé par des événements discrets et le coeur de réseau peut être simulé analytiquement réduisant ainsi la lourdeur des calculs. Des sources issus de protocoles de transport ont également été étudiées. A ce titre, le protocole de transport TCP, fondamental pour le fonctionnement du réseau Internet, a été implémenté sous forme événementielle dans le simulateur. Nous avons également étudié un nouveau modèle analytique dynamique de ce type de source. Ce modèle a été conçu pour suivre précisément les fluctuations de débit de sources TCP. Deux modèles analytiques fluides de sources TCP issus de travaux récents sont également utilisés dans le simulateur. Ils permettent de représenter le débit moyen de sources TCP de courte ou de longue durée à chaque instant.

Le simulateur présenté dans cette thèse est un simulateur général de réseaux de files d'attente à commutation de paquets. En utilisant les spécifications des réseaux IP-DiffServ-MPLS, cet outil a pu être spécialisé à ce qui constitue le réseau Internet d'aujourd'hui. Il permet, à l'instar d'autres logiciels du même domaine, de réaliser des simulations purement événementielles en utilisant des modèles de source de trafic précis et réalistes, offrant ainsi des statistiques détaillées sur l'état de chaque ressource et de chaque trafic du réseau, en régime transitoire ou stationnaire. Il permet également de réaliser ces simulations en utilisant exclusivement des modèles analytiques, et enfin, il permet la combinaison de ces deux types de simulation afin d'obtenir des résultats plus précis que les simulations exclusivement analytiques, et en des temps très inférieurs à la simulation exclusivement événementielle.

Afin de réduire les temps de calcul dans une plus large mesure, un travail parallèle dans un tout autre domaine a été réalisé dans le cadre de cette thèse. Il concerne les environnements d'exécution parallèle et a pour but de fournir une plate-forme d'exécution performante pour le simulateur DHS. En effet, les principes de simulation que nous utilisons se parallélisent facilement, et la simulation hybride peut ainsi tirer partie de la puissance cumulée des stations de travail, des PC de bureau (grappes), ou même des grilles de calcul dont nous disposons afin d'accélérer les temps de traitement. Ce travail a été motivé par le développement des cartes d'interface de haut-débit pour les réseaux locaux. Il a consisté en la conception et la réalisation d'un noyau de communication pour des applications parallèles s'exécutant sur des calculateurs interconnectés par des réseaux haut-débit hétérogènes. Cette bibliothèque de communication est conçue de façon à être utilisable par les environnements de développement classique, et permet l'intégration rapide et aisée de nouveaux media de communication. Outre la possibilité de fonctionner en environnement hétérogène, un objectif primordial de cette bibliothèque a été de réduire au maximum les temps de latence de communication entre tâches parallèles, permettant ainsi l'utilisation d'algorithmes de communication à grain fin, comme ceux utilisés dans la simulation hybride.

Références

[SAA60] T. L. Saaty « Time-dependent solution of the many-server Poisson queue », Operation Research 8, 1960

[ANC61] C. J. Ancker and A. V. Gafarian, « Queueing with Multiple Poisson inputs and Exponential Service Times », Operations Research 9, 321-328 (1961)

[TAK62] L. Takács « Introduction to the Theory of Queues », Oxford University Press, 1962

[COH69] J.W. Cohen « The Single Server Queue », North Holland, 1969.

[CER74] V. Cerf, R. Kahn, « A Protocol for Packet Network Intercommunication », IEEE Transactions on Communications, Vol. COM-22, No. 5, pp 637-648, Mai 1974

[KOB74] H. Kobayashi « Application of the Diffusion Approximation to Queuing Theory Networks II : Non-equilibrium Distributions and Applications to Computer Modeling », Jour. of the Ass. for Comp. Machinery, Vol 21, N^{o} 3, 1974

[KLE75] L. Kleinrock. Queueing Systems, « Volume I: Theory » A Wiley-Interscience Publication, 1975

[KLE76] L. Kleinrock. Queueing Systems, « Volume II: Computer Applications » A Wiley-Interscience Publication, 1976

[PHI76] Philips, Ravindran et Solberg « Operations Research, Principles and Practice », Wiley, 1976

[FAY79] G. Fayolle, R. Iasnogorodski, et V. Malyshev « Random walks in the quarter-plane », Springer, Berlin, 1979.

[GAR80] J.M. Garcia « Problèmes liés à la modélisation du traffic et à l'acheminement des appels dans un réseau téléphonique », Thèse de doctorat, université Paul sabatier, Toulouse, 1980.

[KON81] A. G. Konheim, I. Meilijson, A. Melkman « Processor-sharing of two parallel lines », Journal of Applied Probability, 18:952-956, 1981.

[COH83] J. W. Cohen, O. J. Boxma « Boundary value problems in queueing systems analysis », North-Holland, Amsterdam, 1983.

[COH84] J. W. Cohen « On a functional relation in three complex variables », Report 359, Mathematical Institute, University of Utrech, 1984

[LEG84] F. Le Gall, J.M. Garcia, J. Bernussou « A one moment model for telephone traffic. Application to blocking estimation and resource allocation », Rapport LAAS N. 3014, second International Symposium on the Performance of Computer Communications Systems, Zurich (Suisse), March 21-23 1984. Performance of Computer-Communication Systems, eds. H.Rudin, W.Bux, North Holland, 1984, pp.159-171

[COH85] J. W. Cohen. « On the analysis of parallel, independant processors », Report 374, Mathematical Institute, University of Utrech, 1985.

[KEL85] W. D. Kelton, A. M. Law « The transient behaviour of the M/M/s queue with implications for the steady state simulations », Operations Research 33(2), 1985

[LEG85] F. Le Gall, J. Bernussou, J.M. Garcia « A state dependent one moment model for grade of service and traffic evaluation in circuit-switched networks », Rapport LAAS N. 85230, 11th International Teletraffic Congress, pp.5.2B/2.1-5.2B/2.6, Kyoto (Japan), september 4-11 1985.

[GAR86] J.M. Garcia, F. Legall, J. Bernussou « A Model for Telephone Networks and its Use for Routing Optimization Purposes », IEEE Jour. on Select. Areas in Comm., Special issue on communication network performance evaluation, Vol. 4, 1986.

[MAR86] R. Marie, G. Rubino « M/M/1 FIFO-Multiclass Queues : Exact Results and Generalizations », IRISA Research Report N°333, 1986.

[MIS86] J. Misra « Distributed Discrete-Event Simulation », ACM Computing Surveys, Vol 18, N°1, 1986

[ABA87] J. Abate, W. Whitt « Transient Behaviour of the M/M/1 queue: starting at the origin », Queuing Systems, 2, 1987

[JAC88] V. Jacobson « Congestion Avoidance and Control », Computer Communication Review, vol. 18, no 4, p 314-329 1988 (ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z)

[MUR88] J. R. Murray, W. D. Kelton « The transient behaviour of the $M/E_k/2$ queue and steady-state simulation » Comput. Operat. Res. 15, 1988

[GRA90] A. Gravey, J.R. Louvion, P. Boyer « On the *Geo/D/1* and *Geo/D/1/N* Queues », Performance Evaluation 11(2), 1990.

[JAC90] V. Jacobson « Modified TCP Congestion Avoidance Algorithm », Technical Report, 1990 (ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt)

[BAI91] A. Baiocchi et al « Loss Performance Analysis of an ATM multiplexer Loaded with High-Speed ON-OFF Sources », IEEE Journal on Selected Areas in Communications, Avril 1991, p. 388-393

[DSL93] D. S. Lee, S. Q. Li « Transient Analysis of a switched Poisson arrival under overload control » Performance Evaluation 17(1), 1993

[FLO93] S. Floyd, V. Jacobson « Random Early Detection Gateways for Congestion Avoidance », IEEE/ACM Transactions on Networking, V.1 N.4, Août 1993, pp. 397-413. (voir également http://www.icir.org/floyd/papers/red/red.html)

[LEE93] I. J. Lee, E. Roth « A heuristic for the transient expected queue length of Markovian queueing systems » Operations Research Letters 14(1), 1993

[SCI93] « IEEE Standard for Scalable Coherent Interface », IEEE Std 1596-1992 I. C. Society, Inc 345 East 47th Street, New York, 1993

[STE94] W. Stevens « TCP/IP Illustrated, Volume 1: The protocols » (Chapter 17, 19, 20, 21) Addison Wesley, 1994

[SUN94] S. Sunderam, G. Geist, J. Dongarra, R. Mancheck « The PVM Concurrent Computing System: Evolution, Experience, and Trends. » Parallel Computing, (20):531-545, 1994

[ROS95] O. Rose « Statistical Properties of MPEG Video Traffic and their Impact on Taraffic Modeling in ATM Systems », PhD thesis, Institute of Computer Science, University of Wurzburg, Germany, 1995

[ATM95] B. Weiss « ATM », Hermes, 1995

[BOD95] N. Boden, D. Cohen, R. Feldermann, A. Kulawik, C. Seitz, W. Su « Myrinet: A Gigabit per Second Local Area Network », IEEE-Micro (15-1):29-36, Fevrier 1995

[MON95] T. Monteil, J-M. Garcia, P. Guyaux « LANDA: Une machine virtuelle parallèle », Calculateurs Parallèles, 7(2): 119-137, 1995

[LEA96] D. Lea, « A Memory Allocator », unix/mail, December 1996. Voir aussi http://g.oswego.edu/dl/html/malloc.html.

[MON96] T. Monteil « Etude de Nouvelles Approches pour les Communications, l'Observation et le Placement de Tâches dans l'Environnement de Programmation Parallèle LANDA », Thèse de Doctorat, LAAS-CNRS, 1996

[FAL96] K. Fall et S. Floyd « Simulation-based Comparison of Tahoe, Reno and SACK TCP », Computer Communication Review, July 1996 (ftp://ftp.ee.lbl.gov/papers/sacks.ps.Z)

[ROB96] Broadband Network Teletraffic « Final Report of Action Cost 242 », J. Roberts & AL, Eds Springer Berlin, 1996.

[VIC96] N. Vicari and P. Tran-Gia « A Numerical Analysis of the *Geo/D/N* Queueing System » Research Report N°151, Institute of Computer Science, University of Wurzburg, September 1996.

[LAU97] M. Lauria, A. Chien « MPI-fm: High Performance MPI on Workstation Clusters », Journal on Parallel and Distributed Computing, 40(1):4-18, 1997

[GAR98] J.M. Garcia « A new approach for analytical modelling of packet switched telecommunication networks », LAAS-CNRS Research Report N°98443, 1998

[GAU98] D. Gauchard « Etude et simulation d'algorithmes de placement des tâches d'une applicationdistribuée par prédiction de la charge cpu et du trafic réseau. », DEA, LAAS-CNRS, 1998

[HEL98] P. Hellekalek « Good random number generators are (not so) easy to find », Mathematics and Computers in Simulation 46 (1998) p. 485-505

[LEN98] T. Le-Ngoc et S. Shah-Heydari « MMPP Modeling of Aggregatred ATM Traffic », IEEE 1998, p 129-132

[KRU98] M. Krunz et M. Makowski « Modeling Video Traffic using M/G/oo Input Processes: A compromise Between Markovian and LRD Models », IEEE Journal of Selected Areas in Communications, June 1998, p 733-748.

[PRY98] L. Prylli, B. Tourancheau « BIP: A new protocol designed designed for high-performance networking on Myrinet », Proc. of PC-NOW IPPS-SPDP'98, Orlande, USA, Mars 1998

[ROB98] J. Roberts, U. Mocci, ... Eds. « Broadband Network Teletraffic », Cost 142 Final Report, 1998

[TOU98] F. Toutain « Ordonnancement de paquets équitable par les disciplines GPS et DGPS » Ann. Telecommun., 53, n° 9-10, 1998

[CIS99] Sportack, Mark A. « IP Routing Fundamentals » Indianapolis: Cisco Press, 1999

[GAR99] J-M. Garcia, D. Gauchard, T. Monteil, O. Brun « Process Mapping Given by Processor and Network Dynamic Load Prediction », European Conference on Parallel Computing Europar'99, 291-294, 1999

[STR99] « PUBLIC final report of Esprit HPCN PST activity $STAR_1$ - Simulation of Telecommunication Architectures » http://www.icetact.tcd.ie/icetact/examples/star1.html, 1999

[WAN99] L. C. Wang « On the transient delays of M/G/1 queues », J. Appl. Prob. 36(3), 1999

[BRU00a] O. Brun « Modélisation et optimisation de la Gestion des Ressources dans les Architectures Distribuées - Parallélisation Massive de la technique de Filtrage Particulaire », Thèse de Doctorat, LAAS-CNRS, 2000

[BRU00b] O. Brun, J.M. Garcia « Analytical Solution of Finite Capacity M/D/1 Queues », J. Appl. Prob., Vol 34, N°4, 2000.

[GAR00] J.M. Garcia, O. Brun « Transient Analytical Solution of M/D/1/N Queues », J.Appl. Prob. et LAAS-CNRS Research Report N°00393, 2000.

[ADA01] rI.J.B.F. Adan, O.J. Boxma, J.A.C. Resing « Queueing models with multiple waiting lines », Queueing Systems, vol. 37, pages 65-98, 2001.

[BAR01] C. Barakat « Evaluation des performances du contrôle de congestion dans l'Internet » Thèse de doctorat, Université de Nice - Sophia Antipolis - UFR Sciences, 2001

[THI01] E. Thibault « Etat de l'Art de la modélisation des Flux Multimédia », INRIA Sophia Antipolis, Rapport du Rrojet RNRT Esquimaux, 2001

[SIK01] B. Sikdar, S. Kalyanaraman, K. S. Vastola « An Integrated Model for the Latency and Steady-State Throughput of TCP Connections », Performance Evaluation, Vol.46, n°.2-3, pp.139-154, Septembre 2001

[THI02] E. Thibault « Modélisation des Flux TCP en Boucle Fermée », INRIA Sophia Antipolis, Rapport du projet RNRT Esquimaux, 2002

[RAN02] « Linear congruential generator: LCG » http://random.mat.sbg.ac.at/~charly/server/node3.html

[BOC03] C. Bockstal « Modélisation de réseaux IP Multiservices », rapport LAAS-CNRS, 2003 (à paraître)

[RFC761] Information Sciences Institute « Transmission Control Protocol », University of Southern California, janvier 1980, http://www.faqs.org/rfcs/rfc761.html

[RFC791] Information Sciences Institute « Internet Protocol (*version 4*) » University of Southern California, 1981 http://www.faqs.org/rfcs/rfc791.html

[RFC793] Information Sciences Institute « Transmission Control Protocol » University of Southern California, 1981 http://www.faqs.org/rfcs/rfc793.html

[RFC826] D. C. Plummer « An Ethernet Address Resolution Protocol, or, Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware », 1982, http://www.faqs.org/rfcs/rfc896.html

[RFC896] J. Nagle, « Congestion Control in IP/TCP Internetworks », 1984 http://www.faqs.org/rfcs/rfc896.html

[RFC970] J. Nagle « On Packet Switches With Infinite Storage », 1985 http://www.faqs.org/rfcs/rfc970.html

[RFC1058] C. Hedrick « Routing Information Protocol (RIP) » Rutgers University June 1988 http://www.faqs.org/rfcs/rfc1058.html

[RFC1247] J. Moy, « OSPF Version 2 », 1991 http://www.faqs.org/rfcs/rfc1247.html

[RFC1388] G. Malkin « RIP Version 2, Carrying Additional Information » Xylogics, Inc., 1993 http://www.faqs.org/rfcs/rfc1388.html

[RFC2001] W. Stevens « TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms », 1997 http://www.faqs.org/rfcs/rfc2001.html

[RFC2018] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow « TCP Selective Acknowledgement Options (SACK) », 1996 http://www.faqs.org/rfcs/rfc2018.html

[RFC2205] R. Braden, E. L. Zhang, S. Berson, S. Herzog, S. Jamin, « Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification », 1997, http://www.faqs.org/rfcs/rfc2205.html

[RFC2210] J. Wroclawski « The Use of RSVP with IETF Integrated Services », 1997, http://www.faqs.org/rfcs/rfc2210.html

[RFC2211] J. Wroclawski « Specification of the Controlled-Load Network Element Service », 1997, http://www.faqs.org/rfcs/rfc2211.html

[RFC2212] S. Shenker, C. Partridge, R. Guerin « Specification of Guaranteed Quality of Service », 1997, http://www.faqs.org/rfcs/rfc2212.html

[RFC2215] S. Shenker, J. Wroclawski « General Characterization Parameters for Integrated Service Network Elements », 1997, http://www.faqs.org/rfcs/rfc2215.html

[RFC2309] B. Braden D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang « Recommendations on Queue Management and Congestion Avoidance in the Internet », 1998, http://www.faqs.org/rfcs/rfc2309.html

[RFC2581] M. Allman, V. Paxson, W. Stevens, « TCP Congestion Control », 1999 http://www.faqs.org/rfcs/rfc2581.html (maj de [RFC2001])

[RFC2582] S. Floyd, T. Henderson « The NewReno Modification to TCP's Fast Recovery Algorithm », 1999 http://www.faqs.org/rfcs/rfc2582.html

[RFC2883] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky « An Extension to the Selective Acknowledgement (SACK) Option for TCP », 2000, http://www.faqs.org/rfcs/rfc2883.html

[GNU] GNU (Gnu is Not Unix) http://www.gnu.org

[HPF] High Performance Fortran, http://www.crpc.rice.edu

[MPC] Multi-PC, http://mpc.lip6.fr

[MPI] Message Passing Interface, http://www.mpi-forum.org

[MPM] Multi-Platform MPICH, http://www.lfbs.rwth-aachen.de/mp-mpich

[MYR] Myricom, http://www.myri.com

[NS] Network Simulator, http://www.isi.edu/nsnam/ns

[OMP] OpenMP (Open Multi Processing), http://www.openmp.org

[PM2] PM2 : Parallel Multithreaded Machine, http://www.pm2.org

[TOP] top500, http://www.top500.org

A. Annexes

A.1 Modèles analytiques fluides de TCP

A.1.1 Introduction

Nous présentons ici deux modèles analytique fluides de sources TCP. Ces modèles sont réactifs par rapport à l'état de congestion du réseau. Ils sont qualifiés de modèles en boucle fermée, par opposition aux modèles en boucle ouvertes qui sont indépendants de l'état du réseau et dans lesquels les pertes et les délais sont générés par des processus indépendants.

Le modèle analytique présenté au paragraphe IV.4 est construit pour représenter le fonctionnement réel d'une source TCP. Les deux modèles présentés ici proposent une approximation du débit d'une source TCP-*Reno* variant au cours du temps. Ils sont basés sur la probabilité de perte sur le chemin d'aller et de retour, ainsi que l'estimation du temps d'aller-retour (RTT). Etant basés sur des probabilités, ils ne nécessitent pas de modèle de routeur de type *D(t)/D/*1 pour être intégrés dans le simulateur DHS. Nous nous bornerons ici à rapporter des résultats issus de travaux présentés dans le cadre du projet RNRT ESQUIMAUX [THI02].

Le premier modèle est adapté aux connexions TCP persistantes (de longues durées). Il s'agit d'une approximation du débit moyen associé aux flux persistants de type *General Additive Increase and Multiplicative Decrease* GAIMD(α,β) (Augmentation Additive α et Diminution Multiplicative β Généralisées), qui est une extension très naturelle du contrôle de fenêtre de congestion associé à TCP (GAIMD(1, 1/2)). En effet, sur des connexions de longues durées, TCP-*Reno* est principalement en phase *Congestion Avoidance*, dans laquelle la fenêtre de congestion augmente d'un paquet tous les RTT, et est divisée par deux lors d'une perte de type *triple-ack.*

Le second modèle est une approximation du débit moyen associés aux flux TCP de courtes durées basée sur l'estimation du temps de transfert de données. Ces connexions représentent 80 à 90% des connexions sur le réseau Internet. Elles sont principalement dues aux connexions HTTP qui effectuent généralement des transferts de petits fichiers (10Ko pour HTTPv1 et 50Ko pour HTTPv1.1 en moyenne [THI02]). Ce modèle ne néglige donc pas la phase *Slow-Start* qui est prépondérante dans les très petites connexions.

L'intérêt d'une telle approche mathématique est de pouvoir modéliser des milliers de connexions TCP sans utiliser de simulation événementielle qui écroulerait les temps calcul pour de grands réseaux.

A.1.2 Connexions de longues durées

Le modèle analytique général pour les connexions TCP de longues durées est donné dans [THI02]. Nous présentons ici deux cas particuliers de ce type de connexion, pour un processus de perte déterministe (sans corrélation temporelles) plus adapté aux MAN et aux WAN (*Medium/Wide Area Network*), et pour un processus de perte Poissonien (plus corrélé) mieux adapté aux LAN (*Local Area Network*).
Les deux types de pertes de TCP-*Reno* sont pris en compte (*triple ack dectection*, *retransmit timeout*) dans ce modèle. Sur une période Δt considérée, on a :

• Processus de perte déterministe (MAN, WAN)
Le débit moyen de la source TCP est donné par l'équation (108).

$$\overline{D}_{\alpha,\beta} = \overline{D}_{1,\frac{1}{2}} = \frac{\frac{1}{RTT \cdot \sqrt{p \cdot b}} \cdot \sqrt{\frac{3}{2}}}{1 + p \cdot Q(p) \cdot T(p) \cdot \frac{1}{RTT \cdot \sqrt{p \cdot b}} \cdot \sqrt{\frac{3}{2}}} \quad (108)$$

• Processus de perte poissonien (LAN)
Le débit moyen de la source TCP est donné par l'équation (109).

$$\overline{D}_{\alpha,\beta} = \overline{D}_{1,\frac{1}{2}} = \frac{\frac{1}{RTT \cdot \sqrt{p \cdot b}} \cdot \sqrt{2}}{1 + p \cdot Q(p) \cdot T(p) \cdot \frac{1}{RTT \cdot \sqrt{p \cdot b}} \cdot \sqrt{2}} \quad (109)$$

avec les notations suivantes:

• b représente le nombre de paquet que le récepteur doit recevoir avant d'envoyer un acquittement en fonctionnement normal (b vaut 2 en général),

• $p = p_{\Delta t}$ représente la probabilité de perte d'un paquet utile de la source vers le récepteur sur la période Δt considérée,

• $RTT = RTT_{\Delta t}$ est le temps moyen d'aller-retour entre les deux extrémités de la connexion sur la période Δt considérée,

• $T(p_{\Delta t})$ représente le délai de retransmission moyen des paquets après une perte de type *TimeOut*, et est donné par :

$$T(p) = RTO \cdot \frac{1 + p + 2p^2 + 4p^3 + 8p^4 + 6p^5 + 32p^6}{1 - p} \quad (110)$$

• $Q(p_{\Delta t})$ représentant la probabilité de perte de type *rto* (*Retransmit TimeOut*) est donné par :

$$Q(p) = min\left(1, \frac{(1-(1-p)^3) \cdot \left(1+(1-p)^3 \cdot \left(1-(1-p)^{\omega_{1,\frac{1}{2}}-3}\right)\right)}{1-(1-p)^{\omega_{1,\frac{1}{2}}}}\right) \quad (111)$$

avec :

$$\omega_{1,\frac{1}{2}} = \frac{2+b}{3 \cdot b} + \sqrt{\frac{8 \cdot (1-p)}{3 \cdot b \cdot p} + \left(\frac{2+b}{3 \cdot b}\right)^2} \quad (112)$$

A.1.3 Connexions de courtes durées

Ce modèle fournit une approximation du temps de connexions des trafics TCP. L'estimation des temps de connexions sont séparés comme suit :

• $T_b^1(N)$: Le temps de transfert d'un document moyen,

• $T_b2(N)$: L'augmentation du temps de transfert due aux pertes sans tenir compte de la limitation de la taille de la fenêtre de congestion,

• $T_b^3(N)$: L'augmentation due à la limitation de la taille de la fenêtre de congestion.

De plus, deux cas sont distingués pour tenir compte de la valeur de b ($b = 1$ et $b = 2$).

$$T_b(N) = RTT(T_b^1(N) + T_b^2(N) + T_b^3(N)) \quad (113)$$

• Cas $b = 1$:

$$T_1^1(N) = \frac{\log N}{\log 2} \quad (114)$$

$$T_1^2(N) = \frac{4,5}{1 + RTT^2} \cdot p \cdot N \quad (115)$$

$$T_1^3(N) = \frac{1+p}{W_{max}^{\frac{3}{2}}} \quad (116)$$

• Cas $b = 2$:

$$T_2^1(N) = \frac{\log N}{\log 1,57} \quad (117)$$

$$T_2^2(N) = N \cdot f(p, RTT) + 4 \cdot p \cdot T_2^1(N) + 20 \cdot p \quad (118)$$

$$T_2^3(N) = \frac{N \cdot (10 + 3 \cdot RTT)}{4 \cdot (1-p) \cdot W_{max}^{\frac{3}{2}}} \quad (119)$$

avec

$$f(p, RTT) = \frac{2,32 \cdot (2 \cdot p + 4 \cdot p^2 + 16 \cdot p^3)}{(1 + RTT)^3} + \frac{1+p}{10^3 \cdot RTT} \quad (120)$$

Comme précédemment,

• $p = p_{\Delta t}$ représente la probabilité de perte d'un paquet utile de la source vers le récepteur sur la période Δt considérée,

• $RTT = RTT_{\Delta t}$ est le temps moyen d'aller-retour entre les deux extrémités de la connexion sur la période Δt considérée.

Le débit moyen de la source TCP est alors donné par l'équation suivante :

$$\overline{D} = \int_0^{\infty} \frac{N \cdot MSS}{T(N)} dF \quad (121)$$

Où $F(x)$ représente la fonction de répartition associée à la taille des données échangées.

A.1.4 Tests

A.2 Charge et délais dans un réseau réel

Les paquets événementiels ont une taille de 1024 octets.

Tableau 33: **Liste des Charges, délais et erreurs (par file dans les interfaces)**

NºIF	NºFile	Classes prises en charge	λ (p/s)	*X* (Ana.)	*X* (Hyb.)	*X* (Eve.)	Erreur (%) Ana./Eve.	Erreur (%) Ana./Hyb.	Erreur (%) Hyb./Eve..
1	1	AF1+AF2+AF3+BE+EF	0.635742	0.000186035	0.000185897	0.000186408	0.200499906	0.07417959	0.274883403
2	1	AF1+AF2+AF3+BE+EF	0.242188	7.08622e-05	7.07661e-05	7.13733e-05	0.721259007	0.135615321	0.858037959
3	1	AF1+AF2+AF3+BE+EF	0.666016	0.00459204	0.00459748	0.00458835	0.080356443	0.118465867	0.198587052
4	1	AF1+AF2+AF3+BE+EF	0.454102	0.00312637	0.00313119	0.00311895	0.237335952	0.154172411	0.390905694
5	1	AF1+AF2+AF3+BE+EF	0.605469	0.000177174	0.000177451	0.000177138	0.020319008	0.156343482	0.176386721
6	1	AF1+AF2+AF3+BE+EF	0.393555	0.000115156	0.000114997	0.000115053	0.089443885	0.13807357	0.048696923
7	1	AF1+AF2+AF3+BE+EF	0.938477	0.00648277	0.00648072	0.00648272	0.000771275	0.031622285	0.030860769
8	1	AF1+AF2+AF3+BE+EF	0.0605469	0.000415722	0.000415986	0.000414674	0.252091542	0.063503976	0.31539523
9	1	AF1+AF2+AF3+BE+EF	0.272461	7.97206e-05	7.98193e-05	7.93049e-05	0.521446151	0.123807397	0.644455664
10	1	AF1+AF2+AF3+BE+EF	0.363281	0.000106297	0.000106394	0.000106128	0.158988495	0.091253751	0.250014099
11	1	AF1+AF2+AF3+BE+EF	0.272461	0.00187348	0.00187514	0.00187522	0.092875291	0.088605163	0.004266348
12	1	AF1+AF2+AF3+BE+EF	0.333008	0.00229076	0.00228348	0.00228971	0.045836316	0.31779846	0.272829191
13	1	AF1+AF2+AF3+BE+EF	0.333008	0.00229076	0.00229464	0.00228268	0.352721368	0.169376102	0.521214657
18	1	AF1+AF2+AF3+BE+EF	0.272461	0.00187348	0.00187447	0.00187014	0.178277857	0.052842838	0.230998629
21	1	EF	0.0302734	0.0169407	0.016977	0.0169457	0.029514719	0.21427686	0.184367085
21	3	AF2	0.0605469	0.0339624	0.0339465	0.0341721	0.61744753	0.046816479	0.66457514
22	1	AF1+AF2+AF3+BE+EF	0.0605469	0.000415722	0.000417608	0.000412339	0.813764968	0.453668557	1.261709546
24	1	AF1+AF2+AF3+BE+EF	0.938477	0.00648277	0.00648353	0.00647792	0.074813698	0.011723384	0.086526938
34	1	AF1+AF2+AF3+BE+EF	0.666016	0.00459204	0.00458926	0.00459681	0.103875402	0.060539542	0.16451454
48	1	AF1+AF2+AF3+BE+EF	0.0908203	2.58333	2.54453	2.58658	0.125806614	1.501937422	1.652564521
49	1	AF1+AF2+AF3+BE+EF	0.0908203	2.58333	2.66107	2.51331	2.71045511	3.009294206	5.552653632
50	1	AF1+AF2+AF3+BE+EF	0.0605469	0.925373	0.933219	0.924043	0.143725827	0.847874317	0.983263307
51	1	AF1+AF2+AF3+BE+EF	0.0908203	2.58333	2.55315	2.52281	2.3427127	1.168259572	1.188335977
51	3	AF2	0.0302734	0.448005	0.437284	0.43425	3.070278234	2.393053649	0.693828267
51	5	BE	0.0302734	0.477368	0.493433	0.48561	1.726550586	3.365328216	1.585422945
52	1	EF	0.0908203	0.181543	0.180614	0.18178	0.130547584	0.511724495	0.645575648
52	2	AF1	0.393555	3.8201	2.74956	2.77661	27.31577707	28.02387372	0.983793771
52	3	AF2	0.423828	5.89198	6.87145	7.01516	19.06286172	16.62378352	2.091407199
53	1	EF	0.0302734	0.468359	0.469741	0.46414	0.900804725	0.295072797	1.192359194
53	2	AF1	0.0605469	2.11497	2.10001	2.09859	0.774479071	0.707338638	0.067618726
54	1	EF	0.0302734	0.468359	0.472432	0.476031	1.638059694	0.869632056	0.761802757
54	2	AF1	0.0605469	2.11497	2.15841	2.1795	3.051107108	2.053929843	0.977108149
55	1	EF	0.0302734	0.468359	0.464571	0.472807	0.949698842	0.808781298	1.772818364
55	2	AF1	0.0605469	2.11497	2.12576	2.11834	0.159340322	0.510172721	0.349051633
55	3	AF2	0.0302734	0.448005	0.439696	0.441401	1.474090691	1.854666801	0.387767912
55	5	BE	0.0302734	0.477368	0.501177	0.503798	5.536609073	4.98755677	0.522968931
56	3	AF2	0.0302734	0.448005	0.427468	0.435774	2.730103459	4.584100624	1.943069423
56	5	BE	0.0302734	0.477368	0.488548	0.490678	2.788205326	2.342008681	0.435985819
57	1	EF	0.0302734	0.468359	0.460855	0.469599	0.264754174	1.602189773	1.897342982
57	2	AF1	0.0605469	2.11497	2.08195	2.17018	2.610438919	1.56125146	4.237853935
58	1	EF	0.0302734	0.468359	0.475208	0.469303	0.20155479	1.462339786	1.242613761
58	2	AF1	0.0605469	2.11497	2.1843	2.11051	0.210877696	3.278060682	3.378198965

N°IF	N°File	Classes prises en charge	λ (p/s)	*X* (Ana.)	*X* (Hyb.)	*X* (Eve.)	Erreur (%) Ana./Eve.	Erreur (%) Ana./Hyb.	Erreur (%) Hyb./Eve..
59	1	EF	0.0302734	0.468359	0.46925	0.464921	0.734052297	0.190238684	0.922535962
59	2	AF1	0.0605469	2.11497	2.16541	2.04569	3.275696582	2.384903805	5.528745134
60	1	EF	0.0302734	0.468359	0.473098	0.478481	2.161162698	1.011830668	1.137819226
60	2	AF1	0.0605469	2.11497	2.14479	2.18171	3.155600316	1.409949077	1.721380648
61	1	EF	0.0302734	0.468359	0.466186	0.455876	2.665263185	0.463960338	2.211563625
61	2	AF1	0.0605469	2.11497	2.14709	2.03468	3.796271342	1.518697665	5.235458225
62	2	AF1	0.0605469	0.0339226	0.034141	0.0337327	0.559803789	0.643818575	1.195922791
62	5	BE	0.0302734	0.0169806	0.0171426	0.0171828	1.190770644	0.954029893	0.234503518
63	3	AF2	0.0302734	0.448005	0.439314	0.440373	1.703552416	1.939933706	0.241057649
63	5	BE	0.0302734	0.477368	0.495656	0.505608	5.91577148	3.831006687	2.00784415
64	4	AF3	0.787109	0.723519	0.725139	0.7235	0.002626054	0.223905661	0.226025631
65	3	AF2	0.0302734	0.448005	0.438671	0.444313	0.824097945	2.08345889	1.286157508
65	5	BE	0.0302734	0.477368	0.501915	0.505987	5.995165156	5.14215448	0.811292749
66	3	AF2	0.0302734	0.448005	0.447549	0.434621	2.987466658	0.101784578	2.888622251
66	5	BE	0.0302734	0.477368	0.502534	0.49053	2.757201991	5.271823834	2.388694098
67	3	AF2	0.0302734	0.448005	0.431784	0.434109	3.101751097	3.620718519	0.538463676
67	5	BE	0.0302734	0.477368	0.485758	0.480585	0.673903571	1.757553921	1.064933568
68	3	AF2	0.0302734	0.448005	0.434332	0.423252	5.52516155	3.051974866	2.551043902
68	5	BE	0.0302734	0.477368	0.489302	0.481952	0.960265456	2.499958104	1.502139783
69	1	EF	0.0302734	0.468359	0.467503	0.469151	0.169101053	0.182765784	0.352511107
69	2	AF1	0.0605469	2.11497	2.13423	2.13409	0.904031736	0.910651215	0.006559743
69	3	AF2	0.0302734	0.448005	0.434406	0.42868	4.313567929	3.035457194	1.318121757
69	5	BE	0.0302734	0.477368	0.494665	0.494718	3.634512577	3.623410032	0.010714322
70	3	AF2	0.0302734	0.448005	0.428402	0.436419	2.586131851	4.375620808	1.87137315
70	5	BE	0.0302734	0.477368	0.486842	0.489606	2.563640629	1.984632401	0.567740663
71	3	AF2	0.0302734	0.448005	0.430584	0.426064	4.897489983	3.888572672	1.049737101
71	5	BE	0.0302734	0.477368	0.497246	0.486536	1.920530911	4.164083055	2.15386348
72	1	EF	0.0302734	0.474388	0.471322	0.48186	1.575082	0.646306399	2.235838768
72	2	AF1	0.0605469	2.18275	2.2002	2.23949	2.599473142	0.799450235	1.78574675
73	1	AF1+AF2+AF3+BE+EF	0.0908203	2.58333	2.54605	2.5617	0.837291403	1.443098636	0.614677638
73	3	AF2	0.0302734	0.448005	0.425268	0.428733	4.301737704	5.075166572	0.814780327
73	5	BE	0.0302734	0.477368	0.484152	0.488315	2.293199377	1.42112584	0.85985393
74	1	AF1+AF2+AF3+BE+EF	0.454102	0.00312637	0.00312569	0.00312586	0.016312848	0.021750465	0.005438799

Tableau 34: Liste des Flots, délais et erreurs

N° flot	Classe	Lambda	Délai Ana. (ms)	Délai Hyb. (ms)	Délai Eve. (ms)	Erreur (%) Ana./Eve.	Erreur (%) Ana./Hyb.	Erreur (%) Hyb./Eve.
1	EF	0.0302734	222.337	218.153	224.64	1.035815002	1.881828036	2.973601096
2	EF	0.0302734	222.337	228.299	216.833	2.475521393	2.681514998	5.022361027
3	EF	0.0302734	222.337	219.763	217.791	2.044643941	1.157702047	0.897330306
4	EF	0.0302734	120.869	121.424	119.981	0.734679695	0.459174809	1.188397681
5	EF	0.0302734	120.869	121.227	121.663	0.656909547	0.296188435	0.359655852
6	EF	0.0302734	120.869	119.601	121.447	0.478203675	1.049069654	1.543465356
7	EF	0.0302734	120.869	119.93	120.911	0.034748364	0.776874136	0.817977153
8	EF	0.0302734	120.982	122.912	121.204	0.183498372	1.595278636	1.38961208
9	EF	0.0302734	120.869	120.749	120.741	0.105899776	0.09928104	0.006625314
10	EF	0.0302734	120.982	121.842	122.238	1.038170968	0.710849548	0.32501108
11	EF	0.0302734	120.982	120.909	117.811	2.621051065	0.060339555	2.562257566

N° flot	Classe	Lambda	Délai Ana. (ms)	Délai Hyb. (ms)	Délai Eve. (ms)	Erreur (%) Ana./Eve.	Erreur (%) Ana./Hyb.	Erreur (%) Hyb./Eve.
12	EF	0.0302734	120.982	121.54	121.724	0.613314377	0.461225637	0.151390489
13	EF	0.0302734	122.538	121.615	124.263	1.407726583	0.753235731	2.17736299
14	AF1	0.0302734	75.9482	54.6858	54.9729	27.61790273	27.99592354	0.524999177
15	AF1	0.0302734	75.9482	54.6858	55.1914	27.33020664	27.99592354	0.924554455
16	AF1	0.0302734	75.9482	54.6858	55.3589	27.10966159	27.99592354	1.23084969
17	AF1	0.0302734	75.8352	54.5728	54.6863	27.88797287	28.03763951	0.207979066
18	AF1	0.0302734	75.8352	54.5728	55.0561	27.40033652	28.03763951	0.885606016
19	AF1	0.0302734	75.8352	54.5728	55.2679	27.12104669	28.03763951	1.273711446
20	AF1	0.0302734	75.8352	54.5728	55.1053	27.335459	28.03763951	0.975760819
21	AF1	0.0302734	75.9516	54.6892	55.1333	27.40995581	27.99467029	0.812043328
22	AF1	0.0302734	75.8352	54.5728	55.3705	26.98575332	28.03763951	1.461717192
23	AF1	0.0302734	75.9481	54.6858	55.4946	26.93089096	27.99582873	1.478994547
24	AF1	0.0302734	75.9481	54.6858	54.6284	28.07140666	27.99582873	0.104963263
25	AF1	0.0302734	75.9483	54.6859	54.9257	27.68014557	27.99588668	0.438504258
26	AF1	0.0302734	75.9483	54.6859	55.3529	27.11765767	27.99588668	1.219692828
27	AF2	0.0302734	119.518	119.931	120.083	0.472732141	0.345554644	0.126739542
28	AF2	0.0302734	115.729	112.739	112.251	3.005296857	2.583622083	0.432858195
29	AF2	0.0302734	115.729	112.932	114.124	1.386860683	2.416853166	1.055502426
30	AF2	0.0302734	115.729	111.959	113.317	2.084179419	3.257610452	1.212944024
31	AF2	0.0302734	115.616	112.823	113.365	1.946962358	2.415755605	0.4803985
32	AF2	0.0302734	115.616	113.195	114.223	1.20485054	2.09400083	0.908167322
33	AF2	0.0302734	115.616	114.636	112.736	2.491004705	0.847633546	1.657420008
34	AF2	0.0302734	115.616	111.5	110.828	4.141295322	3.560060891	0.602690583
35	AF2	0.0302734	115.616	111.682	109.546	5.250138389	3.402643233	1.912573199
36	AF2	0.0302734	115.616	111.613	110.792	4.172432881	3.462323554	0.735577397
37	AF2	0.0302734	115.616	111.336	112.355	2.820543869	3.70190977	0.915247539
38	AF2	0.0302734	115.616	111.721	110.172	4.708690839	3.368910877	1.386489559
39	AF2	0.0302734	115.729	110.165	110.873	4.196009643	4.807783702	0.642672355
40	AF3	0.0302734	7.29612	7.30554	7.25265	0.595796122	0.129109719	0.723971123
41	AF3	0.0302734	7.29645	7.30586	7.24722	0.674711675	0.128966826	0.802643358
42	AF3	0.0302734	7.29645	7.30586	7.30095	0.061673828	0.128966826	0.067206325
43	AF3	0.0302734	7.29645	7.30586	7.32424	0.380870149	0.128966826	0.25157887
44	AF3	0.0302734	7.18333	7.19274	7.15499	0.394524545	0.130997741	0.524834764
45	AF3	0.0302734	7.18333	7.19274	7.186	0.037169391	0.130997741	0.093705598
46	AF3	0.0302734	7.18333	7.19274	7.13643	0.652900535	0.130997741	0.78287273
47	AF3	0.0302734	7.18333	7.19274	7.15933	0.334106884	0.130997741	0.464496145
48	AF3	0.0302734	7.18333	7.19274	7.20141	0.251693852	0.130997741	0.120538209
49	AF3	0.0302734	7.18333	7.19274	7.18708	0.052204201	0.130997741	0.078690457
50	AF3	0.0302734	7.18333	7.19274	7.17775	0.07767985	0.130997741	0.208404586
51	AF3	0.0302734	7.18333	7.19274	7.19719	0.192946725	0.130997741	0.061867939
52	AF3	0.0302734	7.29622	7.30564	7.29062	0.076752072	0.129107949	0.205594582
53	BE	0.0302734	4.49705	4.5318	4.53543	0.853448372	0.772728789	0.080100622
54	BE	0.0302734	222.337	220.587	220.251	0.938215412	0.787093466	0.152320853
55	EF	0.0302734	20.1034	20.0119	20.1061	0.013430564	0.455146891	0.470719922
56	EF	0.0302734	15.7317	15.6402	15.7845	0.335628063	0.581628178	0.922622473
57	AF1	0.0302734	222.337	218.153	221.796	0.243324323	1.881828036	1.669928903
58	AF1	0.0302734	222.337	228.299	217.086	2.361730166	2.681514998	4.911541443
59	AF1	0.0302734	222.337	219.763	216.828	2.477770232	1.157702047	1.335529639
60	AF1	0.0302734	272.902	271.4	268.853	1.483682787	0.550380723	0.938467207
61	AF1	0.0302734	272.902	277.613	281.414	3.119068384	1.726260709	1.369172193

N° flot	Classe	Lambda	Délai Ana. (ms)	Délai Hyb. (ms)	Délai Eve. (ms)	Erreur (%) Ana./Eve.	Erreur (%) Ana./Hyb.	Erreur (%) Hyb./Eve.
62	AF1	0.0302734	272.902	273.725	272.897	0.00183216	0.301573459	0.302493378
63	AF1	0.0302734	272.902	268.624	279.508	2.420649171	1.567595694	4.051760081
64	AF1	0.0302734	273.015	279.817	269.487	1.29223669	2.491438199	3.691698503
65	AF1	0.0302734	272.902	279.238	266.777	2.244395424	2.321712556	4.462501522
66	AF1	0.0302734	273.015	276.78	282.175	3.355127008	1.379045107	1.949201532
67	AF1	0.0302734	273.015	276.701	263.458	3.500540263	1.350108968	4.786032577
68	AF1	0.0302734	4.49208	4.51784	4.50108	0.200352621	0.573453723	0.37097374
69	AF1	0.0302734	273.015	275.73	273.85	0.305844001	0.994450854	0.681826424
70	AF1	0.0302734	281.761	284.449	289.539	2.760495597	0.954000021	1.789424466
71	AF1	0.0302734	222.336	218.151	223.673	0.601342113	1.882286269	2.531274209
72	AF1	0.0302734	222.336	228.298	217.261	2.282581318	2.681527058	4.834470736
73	AF1	0.0302734	222.336	219.762	217.835	2.0244135	1.157707254	0.876857692
74	AF1	0.0302734	273.014	271.512	273.72	0.258594797	0.550154937	0.813223725
75	AF1	0.0302734	273.014	277.725	280.459	2.726966383	1.725552536	0.984427041
76	AF1	0.0302734	273.014	273.837	274.39	0.504003458	0.301449743	0.201944953
77	AF1	0.0302734	273.014	268.736	279.443	2.354824295	1.56695261	3.984207549
78	AF1	0.0302734	273.011	279.812	271.936	0.393757028	2.491108417	2.814747044
79	AF1	0.0302734	273.014	279.35	263.002	3.667211205	2.320760108	5.852156793
80	AF1	0.0302734	273.014	276.779	283.327	3.777461962	1.379050159	2.365786422
81	AF1	0.0302734	273.014	276.7	261.361	4.268279282	1.350113914	5.54354897
82	AF1	0.0302734	4.4909	4.51666	4.51884	0.622147008	0.5736044	0.048265754
83	AF1	0.0302734	273.011	275.726	275.294	0.836230042	0.994465424	0.156677281
84	AF1	0.0302734	281.756	284.444	289.988	2.921676912	0.954016951	1.949065545
85	AF2	0.0302734	108.723	126.872	129.15	18.78811291	16.69288007	1.795510436
86	AF2	0.0302734	0.113815	0.113832	0.113875	0.052717129	0.01493652	0.037774967
87	AF2	0.0302734	108.723	126.872	129.948	19.52208824	16.69288007	2.424490825
88	AF2	0.0302734	0.113815	0.113832	0.114133	0.279400782	0.01493652	0.264424766
89	AF2	0.0302734	108.723	126.872	129.306	18.93159681	16.69288007	1.918469008
90	AF2	0.0302734	0.113815	0.113832	0.114525	0.623819356	0.01493652	0.608791904
91	AF2	0.0302734	108.61	126.759	129.669	19.38955897	16.71024768	2.29569498
92	AF2	0.0302734	0.11404	0.114057	0.113877	0.142932304	0.01490705	0.157815829
93	AF2	0.0302734	108.61	126.759	128.615	18.41911426	16.71024768	1.464195836
94	AF2	0.0302734	0.11404	0.114057	0.113603	0.383198878	0.01490705	0.398046591
95	AF2	0.0302734	108.61	126.759	129.368	19.11242059	16.71024768	2.058236496
96	AF2	0.0302734	0.11404	0.114057	0.11428	0.210452473	0.01490705	0.195516277
97	AF2	0.0302734	108.61	126.759	130.085	19.77258079	16.71024768	2.623876806
98	AF2	0.0302734	0.11404	0.114057	0.113978	0.054366889	0.01490705	0.069263614
99	AF2	0.0302734	108.723	126.872	130.426	19.96173763	16.69288007	2.801248502
100	AF2	0.0302734	0.110652	0.11067	0.110779	0.114774247	0.016267216	0.098491009
101	AF2	0.0302734	108.61	126.759	129.982	19.67774606	16.71024768	2.542620248
102	AF2	0.0302734	0.11404	0.114057	0.113774	0.233251491	0.01490705	0.248121553
103	AF2	0.0302734	108.723	126.872	129.083	18.72648842	16.69288007	1.742701305
104	AF2	0.0302734	0.113793	0.11381	0.113762	0.027242449	0.014939408	0.042175556
105	AF2	0.0302734	108.723	126.872	130.083	19.646257	16.69288007	2.530897282
106	AF2	0.0302734	0.113793	0.11381	0.113366	0.375242765	0.014939408	0.390123891
107	AF2	0.0302734	113.105	131.254	132.361	17.02488838	16.04615181	0.843402868
108	AF2	0.0302734	4.49604	4.49615	4.4838	0.272239571	0.002446597	0.274679448
109	AF2	0.0302734	108.723	126.872	130.947	20.44093706	16.69288007	3.211898606
110	AF2	0.0302734	0.110652	0.11067	0.110341	0.281061345	0.016267216	0.297280202
111	AF2	0.0302734	108.723	126.872	129.731	19.32249846	16.69288007	2.253452298

N° flot	Classe	Lambda	Délai Ana. (ms)	Délai Hyb. (ms)	Délai Eve. (ms)	Erreur (%) Ana./Eve.	Erreur (%) Ana./Hyb.	Erreur (%) Hyb./Eve.
112	AF2	0.0302734	0.110652	0.11067	0.110919	0.241297039	0.016267216	0.224993223
113	AF3	0.0302734	222.337	220.587	220.592	0.784844628	0.787093466	0.002266679
114	AF3	0.0302734	222.333	220.583	220.873	0.656672649	0.787107627	0.131469787
115	EF	0.0302734	15.7317	15.6402	15.7774	0.290496259	0.581628178	0.877226634
116	EF	0.0302734	0.110652	0.11067	0.110503	0.1346564	0.016267216	0.150899069
117	BE	0.0302734	119.518	119.931	120.007	0.409143393	0.345554644	0.063369771
118	BE	0.0302734	123.307	126.898	125.882	2.088283715	2.912243425	0.800643036
119	BE	0.0302734	123.307	129.064	129.352	4.90239808	4.668834697	0.223145106
120	BE	0.0302734	123.307	126.098	126.627	2.69246677	2.263456251	0.41951498
121	BE	0.0302734	123.194	127.371	129.908	5.449940744	3.390587204	1.991819174
122	BE	0.0302734	123.194	127.768	129.767	5.335487118	3.712843158	1.564554505
123	BE	0.0302734	123.194	130.01	127.69	3.649528386	5.532736984	1.784478117
124	BE	0.0302734	123.194	125.157	125.963	2.2476744	1.593421758	0.643991147
125	BE	0.0302734	123.194	125.989	124.11	0.743542705	2.268779324	1.491400043
126	BE	0.0302734	123.194	127.173	126.365	2.573988993	3.229865091	0.635354989
127	BE	0.0302734	123.194	126.471	126.64	2.79721415	2.660032144	0.133627472
128	BE	0.0302734	123.194	127.641	125.197	1.625890871	3.609753722	1.914745262
129	BE	0.0302734	123.307	124.806	125.242	1.569253976	1.215664966	0.349342179
130	AF3	0.0302734	7.29612	7.30554	7.31108	0.205040487	0.129109719	0.075832861
131	AF3	0.0302734	7.29645	7.30586	7.30238	0.0812724	0.128966826	0.047632996
132	AF3	0.0302734	7.29645	7.30586	7.21267	1.148229619	0.128966826	1.275551407
133	AF3	0.0302734	7.29645	7.30586	7.28088	0.213391444	0.128966826	0.34191731
134	AF3	0.0302734	7.18333	7.19274	7.17561	0.107471048	0.130997741	0.238156808
135	AF3	0.0302734	7.18333	7.19274	7.18647	0.043712317	0.130997741	0.087171231
136	AF3	0.0302734	7.18333	7.19274	7.20972	0.367378361	0.130997741	0.236071372
137	AF3	0.0302734	7.18333	7.19274	7.19644	0.182505885	0.130997741	0.051440758
138	AF3	0.0302734	7.18333	7.19274	7.19923	0.22134581	0.130997741	0.090229871
139	AF3	0.0302734	7.18333	7.19274	7.17378	0.132946697	0.130997741	0.263599129
140	AF3	0.0302734	7.18333	7.19274	7.14793	0.492807653	0.130997741	0.622989292
141	AF3	0.0302734	7.18333	7.19274	7.13337	0.695499163	0.130997741	0.825415627
142	AF3	0.0302734	7.29622	7.30564	7.25291	0.59359504	0.129107949	0.721771125