Thèse de Doctorat de l'université Paris VI
Pierre et Marie Curie

Spécialité

Systèmes Informatiques

présentée par

M. Christophe Deleuze

pour obtenir le grade de

Docteur de l'université Pierre et Marie Curie

Qualité de service dans l'Internet : problèmes liés au haut débit et au facteur d'échelle

Présentée et soutenue publiquement le 13 janvier 2000 devant le jury composé de

M. Samir Tohmé	**Président**
M. Bernard Cousin	**Rapporteur**
M. Pierre Rolin	**Rapporteur**
M. Patrick Cocquet	**Examinateur**
M. Michel Diaz	**Examinateur**
M. Serge Fdida	**Directeur de thèse**

Thèse de Doctorat de l'université Paris VI
Pierre et Marie Curie

Spécialité

Systèmes Informatiques

présentée par

M. Christophe Deleuze

pour obtenir le grade de

Docteur de l'université Pierre et Marie Curie

Qualité de service dans l'Internet : problèmes liés au haut débit et au facteur d'échelle

Présentée et soutenue publiquement le 13 janvier 2000 devant le jury composé de

M. Samir Tohmé	**Président**
M. Bernard Cousin	**Rapporteur**
M. Pierre Rolin	**Rapporteur**
M. Patrick Cocquet	**Examinateur**
M. Michel Diaz	**Examinateur**
M. Serge Fdida	**Directeur de thèse**

17/02/2000

D'après une théorie, le jour où quelqu'un découvrira exactement à quoi sert l'Univers et pourquoi il est là, ledit Univers disparaîtra sur-le-champ pour se voir remplacé par quelque chose de considérablement plus inexplicable et bizarre.

Selon une autre théorie, la chose se serait en fait déjà produite.

Le guide du routard galactique

Remerciements

Cette thèse, intitulée "Qualité de service dans l'Internet : problèmes liés au haut débit et au facteur d'échelle", a été soutenue le 13 janvier 2000 à l'université Pierre et Marie Curie (PARIS VI) devant le jury composé de M. Bernard COUSIN, M. Patrick COCQUET, M. Michel DIAZ, M. Serge FDIDA, M. Pierre ROLIN et M. Samir TOHMÉ.

Je tiens à remercier tous les membres de ce jury :

> M. Samir TOHMÉ, professeur à l'école nationale supérieure des télécommunications, qui m'a fait l'honneur de bien vouloir présider ce jury de thèse.
>
> M. Bernard COUSIN, professeur à l'Irisa, université de Rennes-1, et M. Pierre ROLIN, chef du service évaluation veille et compétences de la direction scientifique du Centre national d'étude des télécommunications, qui ont accepté d'être rapporteurs de ma thèse, pour l'intérêt qu'ils ont porté à ce travail.
>
> M. Patrick COCQUET, directeur technique chez Thomson CSF, M. Michel DIAZ, directeur de recherche du CNRS au LAAS à Toulouse ; qui ont eu l'amabilité de bien vouloir participer à ce jury.
>
> M. Serge FDIDA, professeur à l'université Pierre et Marie Curie (PARIS VI), qui a été mon directeur de thèse.

Je remercie également les membres de l'équipe réseau et performances avec qui j'ai apprécié travailler pendant ces trois années, et plus particulièrement :

Olivier Fourmaux, co-bureaute pendant de nombreuses années ; Manuel Bouyer, pour sa connaissance encyclopédique d'Unix et sa capacité à résoudre les problèmes ; et ceux qui ont quitté l'équipe depuis longtemps mais qui laissent des souvenirs de rigolade (et de travail !) : Julien Calm, José Rezende et Simon-Pietro Romano.

Je tiens à remercier aussi Thomas Dietz et Patrick Mevzek pour l'important travail qu'ils ont fourni sur l'implémentation de RSVP.

Résumé

Ce travail de thèse est centré sur la qualité de service dans les réseaux à commutation de paquets de grande dimension, tels l'Internet. Une architecture à intégration de services a été définie par l'IETF, elle utilise le protocole RSVP et est appelée IntServ/RSVP. Cependant des problèmes subsistent, en particulier dans le contexte des réseaux d'interconnexion où apparaissent des difficultés liées d'une part au haut débit, d'autre part au grand nombre de flots.

Ces dernières années, l'évolution des réseaux haut débit s'est portée en grande partie sur les solutions de type "commutation IP" qui permettent d'associer la flexibilité et la robustesse d'IP aux capacités haut débit d'un moteur de commutation ATM. Cette approche a été généralisée sous le nom de "commutation par labels". Dans une première partie, nous proposons "RSVP switching", une solution intégrant l'architecture IntServ/RSVP à un environnement de type "commutation IP". Si elle permet de combiner gestion de la qualité de service et haut débit, cette solution souffre cependant des problèmes de sensibilité au facteur d'échelle de l'architecture IntServ/RSVP qui requiert le maintien d'états par flot.

Dans une deuxième partie, nous étudions les techniques d'agrégation comme moyens de fournir des garanties de service sans gérer d'état pour chaque flot. Nous montrons dans quelles conditions la gestion d'états par agrégat de flots est suffisante et proposons une architecture mettant en œuvre ces techniques d'agrégation dans les réseaux d'interconnexion et s'interfaçant de manière transparente avec le modèle classique IntServ/RSVP utilisé dans les réseaux périphériques.

Le changement de granularité correspondant à un modèle utilisant des agrégations de flots se traduit par une perte de flexibilité dans la gestion des ressources. Dans une troisième partie, nous étudions une approche permettant de conserver une granularité au niveau flot, sans maintenir d'état par flot dans les nœuds. Pour cela le trafic est conditionné en entrée de la *zone sans état*, et des précautions sont prises pour limiter l'impact des nœuds sur l'allure du trafic.

Mots-Clés :

QoS, qualité de service, facteur d'échelle, haut débit, agrégation, commutation IP, *weighted fair queuing*, *virtual clock*

Abstract

This work focuses on quality of service in large scale packet switching networks, like the Internet. An integrated services architecture has been defined by the IETF. It uses the RSVP protocol and is thus named IntServ/RSVP. However, some problems remain, especially considering interconnection networks, namely difficulties related to high rates and to the large number of flows to be managed.

During the last few years, most of high speed network architectures have evolved toward "IP switching" solutions. They are able to associate IP's robustness and flexibility with ATM switching engines' high capacity. This approach has been generalised as "label switching". Our first contribution is the "RSVP switching" proposal, a solution that integrates the IntServ/RSVP architecture in an "IP switching" environment. Even if it's able to combine quality of service management and high rates, this solution suffers from scalability problems due to the per-flow paradigm used by IntServ/RSVP.

Our second contribution is the study of aggregation techniques as means of providing quality of service without requiring management of per flow state. We show how managing per aggregate flows can be sufficient and we propose an architecture implementing such techniques in the "aggregation zone". We also describe mechanisms to interface transparently IntServ/RSVP with the aggregation based architecture.

Aggregation is a change in granularity that implies losing some flexibility in resources management. In our third contribution, we study a different approach allowing flow level granularity without requiring per flow state management. This is achieved by conditioning traffic at "stateless zone" entry, and limiting network nodes' impact on traffic characteristics.

Key Words:

QoS, quality of service, scalability, high rates, aggregation, IP switching, weighted fair queuing, virtual clock

Table des matières

Chapitre 1

Introduction

LA QUALITÉ DE SERVICE DANS L'INTERNET

L'Internet est un réseau à commutation de paquets qui ne fournit qu'un service "au mieux" (*best-effort*), c'est à dire qu'aucune garantie d'aucune sorte n'est faite sur le service offert, autre que l'engagement du réseau à faire "du mieux qu'il peut" pour amener les paquets à destination. Ce type de service est tout à fait utilisable par les applications *élastiques*, qui n'ont pas de contrainte temporelle et peuvent donc accepter de grandes variations de délai et compenser les pertes éventuelles par des retransmissions. Ainsi le réseau ne fournit qu'un service minimal, et c'est au niveau de la couche transport qu'est gérée la fourniture d'un service de bout en bout satisfaisant. Toutes les applications initialement développées sur l'Internet se satisfont d'un tel service.

Cependant, le développement de l'Internet a suscité la création de nouveaux types d'applications, *multimédias*, qui ne peuvent se satisfaire d'un tel service minimal. Ces applications ont des contraintes du type *temps réel* et ont besoin de garanties de délai et/ou de débit pour les flots de données qu'elles génèrent. Différents types d'applications auront besoin de garanties plus ou moins strictes. Pouvoir satisfaire les besoins de ces applications, c'est leur fournir une *qualité de service* adaptée.

Depuis de nombreuses années, des travaux importants sont menés dans le domaine de la qualité de service dans les réseaux à commutation de paquets. En particulier, des techniques d'ordonnancement de paquets (comme par exemple le *weighted fair queuing*) permettent de garantir un débit et sous certaines conditions un délai à un *flot* de données. Dans le cadre de l'Internet, une architecture pour fournir la qualité de service a été développée par l'IETF. Basée sur l'*intégration de services*, l'architecture IntServ/RSVP utilise le protocole RSVP pour gérer des réservations de ressources pour chaque flot de données.

Plus récemment, une approche différente a été proposée par l'IETF : la *différenciation de services*, DiffServ, repose sur une gestion de trafic *par classe* et des méthodes de conditionnement du trafic à l'entrée dans le réseau. Cette approche résout certains des problèmes qui se posent à IntServ mais apporte ses propres difficultés. Dans cette thèse, nous nous concentrerons sur l'architecture IntServ.

Pour qu'une telle architecture soit effective, elle doit être mise en place dans l'ensemble du réseau, y compris dans les réseaux d'interconnexion (*backbones*), de façon à fournir des garanties de service de bout en bout. Ces réseaux d'interconnexion posent cependant des contraintes particulières et sont donc sources de problèmes pour l'architecture IntServ.

Les problèmes liés au haut débit

La première contrainte est celle du haut débit. Une très grande quantité de paquets devant être traités en un temps très court, il faut obtenir des temps de traitement par paquet extrêmement faibles. Ceci nécessite l'utilisation de technologies particulières pour les éléments d'interconnexion, qui sont capables de traiter les paquets à un rythme très élevé. Dans le cas d'un réseau à intégration de services, le traitement à appliquer à chaque paquet est nettement plus complexe ce qui rend le problème plus difficile à résoudre.

Une approche développée ces dernières années pour atteindre des débits élevés dans le cadre du service au mieux est représentée par les solutions de type "commutation IP" qui permettent d'associer la flexibilité et la robustesse d'IP aux capacités haut débit d'un moteur de commutation ATM. Cette approche au départ proposée par la compagnie "Ipsilon" a été reprise par de grands industriels des réseaux. Elle a été généralisée sous le nom de "commutation par labels". Le groupe de travail *Multi Protocol Label Switching* (MPLS) de l'IETF est chargé de standardiser cette technologie.

Nous nous proposons de montrer comment les mécanismes de l'intégration de services peuvent être adaptés pour ce type de technologie, résultant en une technologie de réseau haut débit capable de gérer la qualité de service.

Les problèmes liés au facteur d'échelle

Un autre problème est celui du *facteur d'échelle*. Une architecture telle que IntServ/RSVP qui requiert de maintenir des états *par flot* pose certains problèmes pour le *passage à l'échelle*. Une augmentation très importante du nombre de flots à gérer se traduit en un accroissement de la charge de traitement qui peut devenir insupportable. Cette charge se situe aussi bien dans le plan de données, où des états de classification et d'ordonnancement doivent être maintenus par flot et manipulés à chaque paquet, que dans le plan de contrôle où il faut maintenir des états de gestion par flot.

Dans le texte de cette thèse, nous utiliserons souvent les mots "scalabilité" et "scalable", dérivés des mots anglais *scalability* et *scalable*, pour éviter la lourdeur du recours systématique à des expressions du type "capacité à passer à l'échelle" ou "sensible au facteur d'échelle".

Nous explorons dans cette thèse deux types d'approches au problème du passage à l'échelle :

- les techniques basées sur l'*agrégation* correspondent à un changement de granularité dans la gestion des ressources. En regroupant plusieurs flots en une seule entité à laquelle fournir un service, on réduit le nombre d'états à gérer, en contrepartie d'une perte de flexibilité.
- une seconde approche consiste à ne plus maintenir d'état du tout, mais à transporter les informations d'état dans les paquets de données eux-mêmes. Nous proposerons une solution simple basée sur ce principe.

PLAN DU DOCUMENT

Le corps de la thèse est constitué de quatre chapitres.

Dans le chapitre deux, nous faisons un bref état de l'art de la gestion de la qualité de service dans les réseaux à commutation de paquets tels que l'Internet. Nous décrivons en particulier les deux modèles actuellement proposés, IntServ et DiffServ, leurs architectures, les services proposés, leurs avantages et les problèmes qui restent à résoudre. Nous montrons comment il est possible d'obtenir des garanties de délais déterministes dans un réseau à commutation de paquets. Enfin, nous présentons la technique d'ordonnancement *weighted fair queuing* et quelques unes de ses nombreuses variantes.

Dans le chapitre trois, nous présentons les principes de la "commutation IP", et en particulier la solution "IP switching" développée par Ipsilon. Cette technologie permet de construire des éléments d'interconnexion supportant des débits très élevés, mais n'inclut aucun support pour la qualité de service. Aussi, nous proposons une adaptation de l'architecture IntServ, et en particulier du protocole RSVP, au contexte de cette technologie. Cette proposition a été validée par la réalisation d'un prototype.

Dans le chapitre quatre, nous étudions une solution basée sur l'agrégation des flots au problème de scalabilité de l'architecture IntServ. Partant de la constatation que ce problème ne se pose réellement que dans des zones bien définies, les réseaux d'interconnexion, nous proposons d'utiliser l'architecture telle quelle partout où c'est possible et d'utiliser des techniques d'agrégation dans les réseaux d'interconnexion. Nous montrons comment des garanties par flot peuvent être assurées malgré une granularité plus grossière dans la gestion des ressources. Nous définissons ensuite des mécanismes d'interfaçage permettant une gestion entièrement transparente des mécanismes d'agrégation du point de vue des applications. Enfin, nous montrons dans quelles conditions les techniques d'agrégation permettent également une gestion plus efficace des ressources.

Dans le chapitre cinq, partant de la même architecture globale proposée au chapitre quatre, nous proposons une approche différente au problème de la scalabilité. Les techniques d'agrégation impliquent intrinsèquement une perte de flexibilité dans la gestion des ressources. La technique *Dynamic Packet State* permet de supprimer la gestion d'état par flot en transportant les informations nécessaires dans les paquets. Afin d'obtenir des garanties déterministes, elle nécessite cependant des mécanismes assez lourds. Nous proposons une approche simplifiée dans laquelle une partie des informations est transportée dans les paquets, et un conditionnement du trafic rend superflues (dans une certaine mesure) les autres informations.

Chapitre 2

La qualité de service dans l'Internet

2.1 INTRODUCTION DU CHAPITRE

L'Internet est un réseau à commutation de paquets à l'architecture et aux principes simples, qui ne fournit qu'un service dit "au mieux" (*best-effort*) c'est à dire sans aucune garantie sur le débit disponible, le délai subi ou encore les pertes de paquets. Ce service s'avère suffisant pour les applications "élastiques" de transfert de données telles que courrier électronique, groupes de discussion, transfert de fichiers etc[1].

Avec le développement considérable de l'Internet, de nouveaux usages se développent autour d'applications multimédias dont les contraintes temps réel se satisfont difficilement du service au mieux. Certaines de ces applications peuvent être conçues de manière à fonctionner plus ou moins bien avec un tel service minimal, mais ces *applications adaptatives* ont cependant besoin d'un minimum de conditions pour se comporter correctement. De plus, certaines applications ont des contraintes fortes et ne peuvent pas s'adapter.

On dira qu'un réseau peut fournir de la *qualité de service* (*QoS* pour *Quality of Service*) s'il est capable de fournir un service adapté aux besoins spécifiques des applications (en particulier temps réel) : garantie de débit ou de délai par exemple.

Dans le cadre de l'Internet, deux architectures basées sur des paradigmes différents ont été développées par l'IETF[2] pour fournir la qualité de service. Ces deux architectures, bien que très différentes, utilisent un certain nombre d'élements de base communs.

[1] La viabilité du service dépend cependant dans une très large mesure d'un contrôle de congestion géré de bout en bout et mis en œuvre par le protocole TCP.

[2] Internet Engineering Task force

Dans ce chapitre, nous décrivons les deux architectures envisagées pour fournir la qualité de service dans un réseau à commutation de paquets tel que l'Internet. Nous rappelons ensuite comment il est possible d'obtenir des garanties de délai déterministes. Puis nous décrivons l'élément fondamental sur lequel doit s'appuyer toute architecture destinée à fournir la qualité de service, l'ordonnancement de paquets en étudiant le cas des algorithmes de la famille *weighted fair queueing.*

2.2 LE MODÈLE INTSERV/RSVP

Le groupe de travail "Integrated Services" (IntServ[3]) de l'IETF a été créé en 1994 pour définir un modèle de service de l'Internet amélioré, capable de transporter aussi bien de l'audio, de la vidéo, des données temps réel que du trafic de données classique ; c'est à dire transformer l'Internet en un réseau à *intégration de services.*

Le travail du groupe (tel que spécifié dans sa charte) se décompose en trois parts :

1. définir les services ;
2. définir les interfaces aux applications, aux mécanismes des routeurs et aux technologies réseaux sous-jacentes (uniquement les interfaces générales dans ce dernier cas, les détails spécifiques à chaque technologie devant être traités par le groupe ISSLL[4]) ;
3. développer des mécanismes de validation certifiant la conformité des routeurs aux exigences du nouveau modèle (l'idée étant de ne pas imposer une implémentation particulière – e.g. un algorithme d'ordonnancement – mais de spécifier le comportement).

L'architecture globale du modèle IntServ était définie dans ses grandes lignes dès 1994 dans le RFC 1633 [BCS94] dont la partie modèle de service est très largement inspirée de [CSZ92]. Il s'agit donc d'un travail longuement mûri.

Nous décrivons ci-dessous l'architecture globale du modèle, les services proposés, la caractérisation des flots utilisée et le protocole de signalisation. Nous concluons par une discussion des limitations du modèle en terme de scalabilité.

2.2.1 Architecture

On définit un flot comme étant une suite identifiable de paquets apparentés qui résultent de l'activité d'un utilisateur et ont besoin de la même qualité de service.

La qualité du service fourni par le réseau à un flot dépend des ressources (bande passante et tampons mémoire) disponibles. Pour garantir une qualité de service à un flot, il faut s'assurer qu'une quantité de ressources suffisante est disponible dans chacun des routeurs (et liens) qu'il traverse pendant toute la durée du flot. Comme le montre la figure 2.1, dans chaque routeur on appelle "contrôle de trafic" (*traffic control*) l'ensemble des éléments qui permettent d'assurer cette condition, au nombre de trois :

[3] `http://www.ietf.org/html.charters/intserv-charter.html`

[4] Integrated Services over Specific Link Layers `http://www.ietf.org/html.charters/issll-charter.html`.

le classificateur de paquets détermine à quel flot appartient un paquet,

l'ordonnanceur de paquets sélectionne pour émission les paquets des différents flots en fonction des ressources réservées,

le contrôle d'admission vérifie que suffisamment de ressources sont disponibles au moment de l'établissement de la réservation.

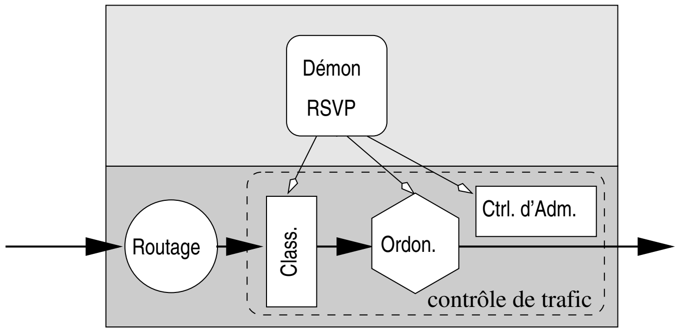


Fig. 2.1 Routeur à intégration de services

Le quatrième élement fondamental d'un réseau à intégration de services est le **protocole de mise en place de réservation** qui à partir de la requête formulée par une application fournit les informations au contrôle de trafic de chaque routeur appartenant au chemin du flot et récupère la réponse du module de contrôle d'admission. Le protocole choisi pour IntServ est RSVP (Resource reSerVation setup Protocol) que nous décrivons brièvement en 2.2.4.

2.2.2 Classes de services

Plusieurs classes de services ont été discutées par le groupe de travail IntServ depuis sa création. Seules deux propositions ont finalement été retenues et sont arrivées à maturité, mais rien n'exclut (en principe) la proposition de nouveaux services.

Controlled Load (CL) : la classe à *charge contrôlée* [Wro97a] fournit le même service qu'un réseau *best-effort* en situation non congestionnée, ce qui est un peu flou. Cette classe est destinée tout particulièrement aux applications multimédias adaptatives dont l'usage s'est développé ces dernières années. Il a en effet été constaté que ces applications se comportent remarquablement bien avec un service *best-effort*, s'adaptant aux variations et aléas du service tant que le réseau n'est pas dans une situation de congestion critique, auquel cas leur comportement se dégrade très rapidement.

Guaranteed Service (GS) : la classe à *service garanti* [SPG97] fournit une garantie stricte sur le délai maximum de chaque paquet et assure un taux de perte nul. Elle s'adresse aux applications temps réel non adaptatives. Nous décrivons en 2.4 les fondements théoriques sur lesquels se base un tel service.

2.2.3 Caractérisation des flots

Dans l'architecture IntServ, chaque flot est caractérisé par un TSPEC (*traffic specification*) [SW97] qui comprend les cinq paramètres décrits dans le tableau 2.1.

Paramètre	*Description*
b	Taille maximum de rafale
r	Débit moyen
p	Débit crête
L (ou M)	Taille maximum de paquet
m	Plus petite unité de poliçage

Tab. 2.1 Les paramètres de spécification de trafic

Les paramètres les plus importants sont les trois premiers, ils permettent de caractériser le flot selon un modèle constitué d'un **seau de jetons**[5] (*token bucket*) qui contrôle le débit moyen et la taille de rafale, suivi d'un limiteur de débit crête. Ceci est illustré par la figure 2.2.

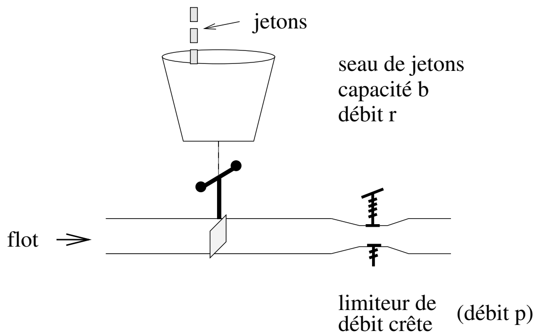


Fig. 2.2 Caractérisation d'un flot

Les deux autres paramètres sont secondaires (mais pas superflus), dans le sens où ils ne caractérisent le flot que par rapport à l'implémentation des mécanismes de qualité de service. La QoS n'est garantie qu'aux paquets dont la taille ne dépasse pas la *taille maximum de paquet* spécifiée dans le TSPEC. Tout paquet dont la taille est inférieure à la *plus petite unité de poliçage* sera "facturé" comme ayant cette taille[6]. Les ressources à réserver dans le réseau dépendent bien entendu, pour une qualité de service donnée, de cette caractérisation.

[5] On distingue le *seau de jetons*, qui est un modèle de caractérisation de trafic, du *régulateur seau percé* (*leaky bucket regulator*) qui est un régulateur de trafic constitué d'un seau de jetons et de tampons mémoire [Kes97].

[6] L'intérêt de ce paramètre est de permettre au réseau d'estimer raisonnablement les ressources *par paquet* nécessaires pour le traitement des paquets du flot et d'avoir une limite supérieure sur l'*overhead* créé par les en-têtes de niveau liaison.

Ce type de caractérisation a l'avantage d'être assez simple à comprendre et à manipuler. Il n'est cependant pas adapté à tous les types d'applications (e.g. une application qui génère un flot à débit constant avec de grandes rafales très espacées).

2.2.4 La signalisation : RSVP

Caractéristiques

Le protocole RSVP [ZDE+93] (*Ressource reSerVation setup Protocol*) a été défini par le groupe de travail RSVP[7] (sic) de l'IETF [BZB+97]. La flexibilité et la capacité à gérer des sessions entre un grand nombre de participants ont été les objectifs principaux dans la conception de RSVP, comme le montrent les caractéristiques suivantes :

Multipoint N vers M : gestion de la QoS pour des sessions multipoint à multipoint (la session point à point classique étant un cas particulier).

Hétérogénéité : différents récepteurs d'un même flot peuvent recevoir une QoS différente. Pour cela le protocole est :

Orienté récepteur : ce sont les récepteurs qui choisissent la qualité de service à fournir au flot.

Dynamicité : les réservations sont renégociables à tout moment, la composition des groupes est totalement dynamique et les changements de route sont possibles.

États relachés : l'état maintenu dans les routeurs est "relaché" (*soft state*) : il doit être rafraîchi régulièrement pour ne pas disparaître.

Réservations partagées : possibilité de partager les réservations entre plusieurs flots (émetteurs) de la même session selon différentes modalités (utile par exemple dans le cas d'une audioconférence où un seul des participants parle – émet – à tout instant).

Mécanismes

Nous décrivons maintenant brièvement les mécanismes de RSVP.

Une session RSVP est identifiée par une adresse IP destination (éventuellement multicast) et un numéro de session. Plusieurs flots peuvent exister dans la même session, un par émetteur. Une application désirant participer en tant qu'émetteur dans cette session envoie des messages PATH contenant le TSPEC du flot qui sont routés jusqu'au(x) destinataire(s). Dans le cas d'une session multicast, les récepteurs ont simplement adhéré préalablement au groupe IP multicast correspondant. Dans chacun des routeurs qu'ils traversent, ces messages PATH créent un *Path State* contenant essentiellement l'adresse du routeur précédent (*Previous Hop*), créant ainsi un **arbre de parcours inverse**. Chaque routeur ajoute également au message PATH des informations, contenues dans les objets ADSPEC (*advertising specification*), sur les qualités de service qu'il est capable de fournir, ce qui correspond à un mécanisme "une passe avec collecte" (*one pass with advertising* ou OPWA). Un appel à la fonction de routage permet ensuite de déterminer le lien de sortie à utiliser pour envoyer le message vers le nœud suivant.

[7] `http://www.ietf.org/html.charters/rsvp-charter.html`

Quand une application réceptrice reçoit le message PATH, elle connaît alors les caractéristiques du flot de données proposé et celles du chemin parcouru. Elle spécifie la qualité de service qu'elle désire obtenir en envoyant un message RESV qui "remontera" vers l'émetteur grâce aux *Path States* installés par le message PATH. Ce message RESV contient le "TSPEC du récepteur" caractérisant le trafic pour lequel le récepteur veut réserver des ressources (dans le modèle théorique un récepteur peut réserver des ressources pour un *sous-flot*) et un RSPEC (*resource specification*) spécifiant les ressources à réserver. À chaque nœud, un contrôle d'admission est réalisé pour vérifier que suffisamment de ressources sont disponibles pour fournir le service demandé. Si c'est le cas, un *Resv State* est créé, contenant les informations relatives à la réservation à effectuer et le message est envoyé au nœud amont. Les *Resv States* créés par différents récepteurs (de la même session) et associés au même lien de sortie sont accumulés dans un *Traffic Control State* contenant la réservation globale sur ce lien pour cette session. Les états de classification et d'ordonnancement peuvent alors être mis en place.

Tous les états étant relachés (*soft states*), les messages sont reémis périodiquement pour les rafraîchir. Cela permet également, en association avec les mécanismes d'IP multicast, à de nouveaux participants de se joindre à la session dynamiquement. De plus les états "orphelins" ne resteront pas indéfiniment dans le réseau. L'effacement des états se fait normalement de manière explicite avec les messages PTEAR (*path tear*) et RTEAR (*resv tear*) mais l'effacement d'un *soft state* à l'expiration d'un *temporisateur de survie* (*cleanup timeout*) permet d'assurer la robustesse du protocole sans nécessiter la fiabilité de la transmission des messages.

La spécification complète de RSVP, donnée dans [BZB+97], est assez complexe (112 pages). Elle contient également la définition des interfaces avec les applications et les routeurs. Une implémentation de l'interface avec l'application, nommée RAPI (*RSVP Application Programming Interface*) est décrite dans [BH97]. Le RFC "RSVP Message Processing Rules" [BZ97] contient une description algorithmique des règles que doit utiliser une implémentation de RSVP et est destiné à clarifier la spécification.

Utilisation avec IntServ

RSVP n'est pas lié aux classes de service de l'IntServ et peut être utilisé dans d'autres contextes. De nouvelles classes peuvent d'ailleurs théoriquement être ajoutées à l'architecture IntServ. Sa spécification ne décrit donc pas la sémantique des informations qu'il transporte. Au contraire, les objets transportés par RSVP sont "opaques". [Wro97b] ("The Use of RSVP with IETF Integrated Services") décrit le format et les règles de traitement des objets à utiliser pour les classes de service CL et GS.

Sécurité et contrôle d'accès

Dans un environnement non contrôlé, des techniques de sécurité sont nécessaires pour empêcher le "vol" de ressources et le sabotage de connexions. L'utilisation de méthodes d'authentification cryptographiques est proposée dans [BLT99].

De même, des règles de contrôle (*policy control*) doivent être définies pour vérifier qui effectue une réservation (authentification de l'utilisateur, comptabilité). Ceci est le rôle du groupe de travail RAP[8] pour *RSVP Admission Policy*, récemment renommé en *Resource Allocation Protocol*. Ce domaine est encore l'objet de recherches.

2.2.5 Scalabilité

Le reproche majeur couramment fait à IntServ, et probablement le facteur principal retardant (empêchant ?) son déploiement est la difficulté liée au passage à l'échelle. En effet le paradigme utilisé, la gestion des ressources *par flot*, implique de maintenir dans chaque routeur un triple état pour chaque flot traversant ce routeur (indépendament d'autres états globaux qui ne posent pas de problème d'échelle) :

signalisation : état de contrôle (par exemple les *resv* et *path states* de RSVP).

classification : pour l'identification des paquets d'un flot.

ordonnancement : pour l'allocation des ressources réservées.

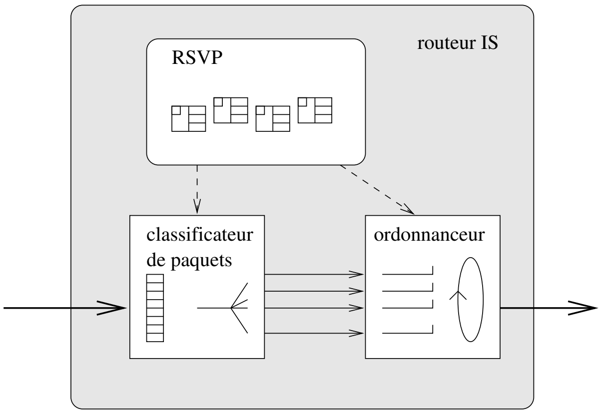


Fig. 2.3 Un état par flot dans chacun des trois modules

La gestion d'états *par flot* ne pose pas de problème dans un environnement de taille réduite où le nombre de flots est limité. Cependant IntServ a pour vocation d'être déployé à l'échelle de l'Internet, donc en particulier dans les réseaux d'interconnexion (les *backbones*) qui peuvent être traversés par plusieurs centaines de milliers de flots de données. Cette situation ne peut qu'empirer avec l'accroissement régulier de la taille de l'Internet. En ce qui concerne l'aspect signalisation, l'IETF a édité un “constat d'applicabilité” de RSVP [M$^+$97] qui incite à la prudence quant au déploiement à grande échelle de RSVP, mettant en lumière les problèmes potentiels en scalabilié mais aussi en sécurité et contrôle d'accès (que nous avons évoqués plus haut).

Il est difficile de dire dans quelle mesure et à partir de combien de flots la gestion d'état par flot devient impossible. L'opération de classification peut se réduire à la lecture d'un

[8] `http://www.ietf.org/html.charters/rap-charter.html`

champ d'en-tête et à une indirection grâce à cette valeur dans un tableau des réservations, opérations qui peuvent certainement être cablées. Beaucoup de recherches sont également menées pour cabler des algorithmes d'ordonnancement [RGB96, VS97].

Les états de signalisation maintenus par RSVP sont eux beaucoup plus complexes et ne se prêtent donc pas à une implémentation cablée. Étant cependant situés dans le plan de contrôle, ils sont utilisés beaucoup moins intensément. [Gué98] donne quelques valeurs de la charge engendrée par la gestion des états de RSVP. Au chapitre 3, nous étudions cet aspect dans le cadre de l'évaluation de notre proposition "RSVP switching", section 3.4.3, page 59.

2.3 LE MODÈLE DIFFSERV

En réaction aux limites et aux difficultés de déploiement du modèle IntServ, et face aux demandes de plus en plus pressantes des fournisseurs d'accès qui veulent pouvoir proposer différents types de service, un nouveau groupe de travail de l'IETF, *The Differentiated Services Working Group* ou *DiffServ*[9], a été chargé d'étudier une nouvelle approche, appelée la *différenciation de services*. Le modèle DiffServ suscite beaucoup de travail, mais certains documents de référence ne sont pas encore finalisés. Nous présentons ci-dessous ses principes, son architecture, et les types de services proposés.

2.3.1 La différenciation de services

"An Architecture for Differentiated Services" [B+98] définit un service comme :

> ...un ensemble de caractéristiques de transmission de paquet exprimées quantitativement ou statistiquement en termes de débit, délai, gigue et/ou taux de perte ou en terme de priorité relative d'accès aux ressources du réseau.

Comme on le voit cette définition est assez lâche et se différencie nettement du concept de qualité de service utilisé par IntServ. DiffServ fournit en fait un ensemble de briques (*building blocks*) qui peuvent être utilisées pour construire différents types de service.

Le but affiché, similaire à celui d'IntServ, est de satisfaire des besoins d'applications et des attentes d'utilisateurs hétérogènes, mais l'approche utilisée est très différente.

2.3.2 Architecture

Le principe du modèle à différenciation de services est de séparer le trafic en quelques *classes de trafic* (agrégats de flots applicatifs appelés *micro-flots*) identifiées par une valeur codée dans l'en-tête de chaque paquet IP [NBBB98]. Cette opération de classification est opérée à l'entrée du réseau (ou de la zone où la différenciation de service est mise en œuvre – *domaine DS*) par les *nœuds de bordure* (*boundary nodes*) qui gèrent des états par flot. Les *nœuds intérieurs* (*interior nodes*) ne gèrent que des états par classe et traitent les paquets en fonction de la classe codée dans l'en-tête en les traitant en accord avec le *comportement*

[9] `http://www.ietf.org/html.charters/diffserv-charter.html`

local (*per hop behavior* ou *PHB*) correspondant. Le service perçu de bout en bout pour un PHB dépend de la politique de *provisionnement du réseau* pour chaque PHB et du *conditionnement de trafic* effectué dans les nœuds de bordure.

Un *domaine DS* est une zone administrative fournissant la différenciation de services, c'est à dire fonctionnant avec un ensemble commun de politiques de provisionnement du réseau et de définitions de PHBs. Une *région DS* est un ensemble contigu de domaines DS qui peuvent offrir la différenciation de services sur des chemins traversant ces domaines. Ils ne mettent évidemment pas forcément en œuvre les mêmes politiques de provisionnement ni les mêmes PHBs.

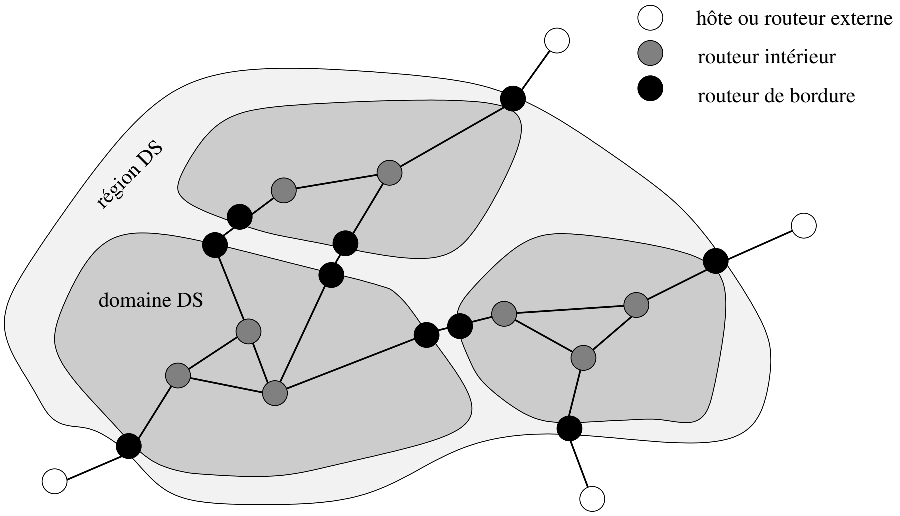


Fig. 2.4 L'architecture DiffServ

Les détails de l'architecture sont définis dans [B+98] et [BBB+99], un modèle pour les routeurs supportant DiffServ est donné dans [BSB99].

2.3.3 *Per hop behaviors*

Deux *per hop behaviors* (PHBs) ont été définis. Ce sont *expedited forwarding* (EF) et *assured forwarding* (AF).

Expedited forwarding

Le PHB *expedited forwarding* (traitement accéléré) [J+99] a pour but de fournir une garantie de bande passante avec des taux de perte, délai et gigue faibles, soit un service pouvant s'apparenter à une "ligne louée virtuelle". Aux débuts du travail sur DiffServ, ce service avait été proposé sous le nom *Premium Service*.

La spécification du PHB indique qu'à tout moment, le débit instantané (calculé sur un intervalle de temps égal à la durée d'émission, au débit configuré, d'un paquet de taille maximum) alloué à cette classe de trafic doit être supérieur au débit configuré. Ce qui implique de :

1. configurer de manière appropriée le débit alloué dans les nœuds,
2. conditionner le trafic à l'entrée du domaine de façon que le débit d'arrivée à chaque nœud soit inférieur au débit alloué.

Le rôle du PHB se limite au premier point, le deuxième dépendant de techniques de provisionnement du réseau.

Ce PHB est un bon candidat pour construire un service de même type que le service GS d'IntServ. Cependant, Stoica et Zhang ont montré dans [SZ98] qu'il n'est possible d'obtenir des délais faibles que si un pourcentage réduit du trafic utilise ce PHB. De plus aucune méthode n'est proposée pour assurer le point 2 de manière efficace.

Assured forwarding

Il s'agit en fait non d'un PHB mais d'une famille de PHBs (*PHB group*). Quatre classes de “traitement assuré” sont définies, chacune comprenant trois niveaux de priorité spatiale (*drop precedence*) [H^+99]. Dans un nœud donné, le niveau d'assurance de traitement d'un paquet dépend de la bande passante allouée à la classe, de la charge actuelle de la classe et de la priorité spatiale du paquet. La nature de la différence entre les classes n'est pas spécifiée, mais elle est souvent pensée en terme de délais [MF98].

Chaque nœud doit mettre en œuvre des mécanismes visant à :

1. réduire la congestion de long terme dans la classe,
2. accepter les rafales (congestion à court terme),
3. traiter identiquement les micro-flots ayant des débits moyens identiques (i.e. ne pas défavoriser un micro-flot *bursty*).

Ceci implique la mise en œuvre d'un algorithme de gestion active de la file, du type RED (*Random Early Detection*) [FJ93, FF97, Del00].

Le service fourni par chaque classe peut être modulé par le réglage des paramètres de cet algorithme en conjonction avec ceux du conditionnement de trafic opéré aux nœuds de bordure. Des “marqueurs à trois couleurs” sont proposés [HG99a, HG99b] pour diviser le trafic de chaque classe selon les trois priorités.

2.3.4 Avantages et problèmes

En séparant le trafic en un nombre réduit de classes et en repoussant la complexité (classification et conditionnement) aux extrémités du réseau, ce modèle atteint l'objectif de scalabilité pour le plan de données. Par contre, par rapport à IntServ il est clair que l'on perd soit en flexibilité soit en fermeté des garanties. De plus, à moins de se contenter d'une forte sous utilisation du réseau (i.e. conditionner le trafic de manière très conservatrice), le problème du plan de contrôle reste entier. Dans le cas de trafic multicast, les choses se compliquent encore et aucune approche n'est proposée sur la façon de dimensionner le réseau et de conditionner ce type de trafic. Clairement, le groupe de travail se trouve confronté sur ce point à des problèmes très difficiles à résoudre.

2.4 LES GARANTIES DE DÉLAI

Émulation de canal

La classe de service *Guaranteed Service* est conçue pour fournir aux applications une garantie stricte sur le délai subi par chaque paquet d'un flot pendant qu'il traverse les nœuds du réseau. Ce délai est composé de deux parts, le *temps de propagation* lié au support physique et incompressible, et le temps passé dans les files d'attente, que l'on peut contrôler par divers mécanismes. Nous nous intéressons ici uniquement au temps passé dans les files d'attente, le temps de propagation pouvant être facilement calculé et ajouté pour obtenir un délai total.

Pour fournir une garantie de délai, il faut isoler complètement le flot des autres flots de données en lui fournissant un **canal de communication dédié** à l'intérieur de chaque lien qu'il traverse. L'émulation parfaite de ce canal, dans laquelle le flot se voit attribuer une part de bande passante est appelée le **modèle fluide**. Dans un réseau à commutation de paquets les nœuds ne peuvent fournir qu'une approximation du service fourni par le canal dédié. Dans chaque nœud, la déviation entre le modèle fluide et cette approximation peut se caractériser par deux termes d'erreur que l'on définit de la manière suivante :

C est le terme d'erreur dépendant du débit. Il sert à calculer le délai potentiel maximum ajouté à un paquet lié aux paramètres de caractérisation du flot[10]. Il est exprimé en octets (ou bits) et est divisé par la réservation affectée au flot pour obtenir le délai correspondant.

D est le terme d'erreur indépendant du débit. Il représente la plus grande variation de délai (indépendante du débit du flot) qu'un paquet peut subir dans le nœud. Un délai constant ajouté systématiquement sera plutôt considéré comme faisant partie du délai de propagation.

Les valeurs de ces termes d'erreur dépendent de la façon dont est réalisée l'approximation du canal du modèle fluide, c'est à dire de l'ordonnanceur mis en œuvre dans le nœud. Nous donnons ces valeurs dans le cas d'un ordonnanceur de type *weighted fair queueing* (WFQ) en 2.5.

Utilisation des courbes d'arrivée et de service

La courbe d'arrivée maximale (dans un modèle abstrait fluide – non paquétisé) d'un flot caractérisé par le TSPEC (b, r, p) (défini section 2.2.3 page 22) et ayant L comme taille maximum de paquet est [PG93, G$^+$96] :

[10] Par exemple, le délai de paquetisation d'un nœud émulant le GPS (voir 2.5.2) ou le délai de réassemblage d'un paquet IP segmenté en cellules ATM.

$$A(\tau) = min\{p.\tau, b + r.\tau\}$$

ce qui s'explique de la manière suivante. À long terme (τ grand) le service maximum est obtenu quand le flot a émis une rafale de taille maximum (b) et continue d'émettre à son débit moyen (r). Ceci est exprimé par le terme $b + r.\tau$. Cependant, la rafale ne peut pas être émise instantanément, le débit étant limité par le débit crête (p). Ainsi, si à partir de $\tau = 0$ la source émet une rafale, elle pourra émettre à son débit crête jusqu'à ce que la taille maximum de rafale ait été atteinte, après quoi elle devra se limiter à son débit moyen. Donc, pour τ petit, la courbe serait $p.\tau$.

Il faut cependant tenir compte du fait que le flot n'est pas fluide mais constitué de paquets. Quel que soit le lissage mis en œuvre, un paquet est toujours émis au débit du lien, qui peut être beaucoup plus important que le débit crête du flot. N'ayant aucun moyen de le connaître en général, nous sommes forcés de le considérer comme infini. Ainsi, la courbe d'arrivée du flot pour des valeurs de τ petites devient $L + p.\tau$, ce qui correspond à une augmentation de l'effet des rafales.

$$A(\tau) = min\{L + p.\tau, b + r.\tau\}$$

Voyons maintenant la courbe de service minimale garantie par le réseau. Pour prendre en compte les termes d'erreur C_i et D_i de chaque nœud on peut considérer que l'ensemble forme un seul système caractérisé par les termes $\sum C_i$ et $\sum D_i$. Le service offert dans le pire des cas est alors l'ajout d'un délai $\sum(\frac{C_i}{R} + D_i)$, puis la fourniture d'un débit constant et égal à la réservation effectuée R.

La figure 2.5 montre comment ces deux courbes peuvent être utilisées pour calculer le délai maximum subi par un paquet (délai d'attente dans les queues, n'incluant pas le délai de propagation) et la taille maximum des tampons nécessaires dans un nœud du réseau.

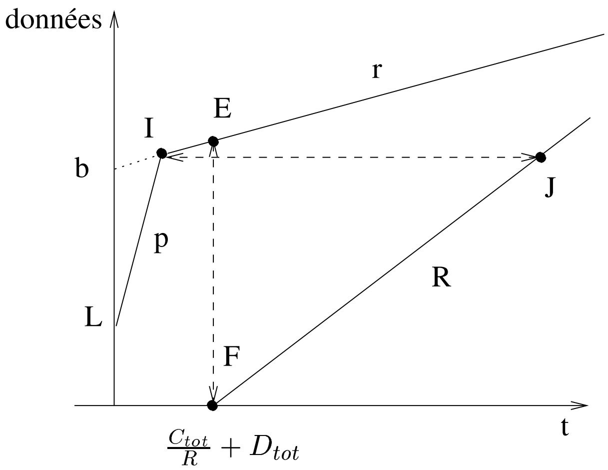


FIG. 2.5 Courbes d'arrivée et de service

Calcul du délai

Le segment IJ représente le délai maximum. Calculons sa longueur. On a (n étant le nombre de nœuds traversés) :

$$y_I = L + px_I = b + rx_I \Rightarrow x_I = \frac{b-L}{p-R}$$

et :

$$x_J = \frac{y_J}{R} + \sum_{i=1}^{n} \left(\frac{C_i}{R} + D_i\right)$$

La longueur du segment est donc :

$$D = x_J - x_I = \frac{y_J}{R} - \frac{b-L}{p-r} + \sum_{i=1}^{n} \left(\frac{C_i}{R} + D_i\right)$$

Comme y_I et y_J sont égaux, remplaçons y_J par $L + p \cdot \frac{b-L}{p-r}$:

$$D = \frac{L + p \cdot \frac{b-L}{p-r}}{R} - \frac{b-L}{p-r} + \sum_{i=1}^{n} \left(\frac{C_i}{R} + D_i\right)$$

$$D = \frac{L}{R} + \frac{p}{R} \cdot \frac{b-L}{p-r} - \frac{b-L}{p-r} + \sum_{i=1}^{n} \left(\frac{C_i}{R} + D_i\right)$$

$$D = \frac{b-L}{p-r} \cdot \left(\frac{p}{R} - 1\right) + \frac{L}{R} + \sum_{i=1}^{n} \left(\frac{C_i}{R} + D_i\right)$$

$$D = \frac{b-L}{p-r} \cdot \frac{p-R}{R} + \frac{L}{R} + \sum_{i=1}^{n} \left(\frac{C_i}{R} + D_i\right)$$

ce qui nous donne finalement :

$$D = \frac{(b-L)}{R} \cdot \frac{(p-R)}{(p-r)} + \frac{L}{R} + \sum_{i=1}^{n} \left(\frac{C_i}{R} + D_i\right) \tag{2.1}$$

Bien entendu, cette équation n'est valable que si la réservation R est supérieure au débit moyen du flot (condition de stabilité $R > r$). Nous nous sommes de plus placés dans le cas où $p > R$. Dans le cas contraire ($p < R$), les calculs sont beaucoup plus simples et la formule (2.1) devient :

$$D = \frac{L}{R} + \sum_{i=1}^{n} \left(\frac{C_i}{R} + D_i\right) \tag{2.2}$$

Qu'elle soit sous sa forme (2.1) ou (2.2), cette formule est composée de deux termes :

- un terme "d'écoulement de rafale" représentant le délai nécessaire pour écouler (c'est à dire lisser) une rafale de taille maximum au débit réservé. Dans le cas où la réservation est supérieure au débit crête ce terme se réduit à $\frac{L}{R}$, correspondant à une rafale d'un paquet émis à un débit infini.
- un terme "d'approximation du modèle fluide" constitué par la somme des délais potentiels ajoutés à chaque nœud du fait des erreurs d'approximation du canal fluide.

La réservation allouée au flot joue sur chacun des deux termes. Les autres paramètres que l'application peut utiliser sont pour le premier terme la variabilité du trafic qu'elle génère. Pour le deuxième, cela dépend de la valeur du terme C donc de la politique d'ordonnancement mise en œuvre.

Calcul de la réservation

On peut exprimer la formule (2.1) différemment de façon à obtenir la réservation nécessaire pour un délai donné :

$$D = \frac{(b-L)}{R} \cdot \frac{(p-R)}{(p-r)} + \frac{L}{R} + \sum_{i=1}^{n} \frac{C_i}{R} + \sum_{i=1}^{n} D_i$$

$$D - \sum_{i=1}^{n} D_i = \frac{(b-L)}{R} \cdot \frac{(p-R)}{(p-r)} + \frac{L}{R} + \sum_{i=1}^{n} \frac{C_i}{R}$$

$$R \cdot \left(D - \sum_{i=1}^{n} D_i \right) = (p-R) \cdot \frac{(b-L)}{(p-r)} + L + \sum_{i=1}^{n} C_i$$

$$R \cdot \left(D - \sum_{i=1}^{n} D_i \right) = p \cdot \frac{(b-L)}{(p-r)} - R \cdot \frac{(b-L)}{(p-r)} + L + \sum_{i=1}^{n} C_i$$

$$R \cdot \left(D - \sum_{i=1}^{n} D_i + \frac{b-L}{p-r} \right) = p \cdot \frac{(b-L)}{(p-r)} + L + \sum_{i=1}^{n} C_i$$

$$R = \frac{p \cdot \frac{(b-L)}{(p-r)} + L + \sum_{i=1}^{n} C_i}{\frac{(b-L)}{(p-r)} + D - \sum_{i=1}^{n} D_i} \tag{2.3}$$

Calcul des tampons

De plus, le segment EF représente la quantité maximum de données en attente de traitement, donc la taille maximum des tampons nécessaires pour éviter toute perte de paquet. Trois situations sont possibles en fonction des positions relatives des points A et B et des valeurs relatives de p et R comme le montre la figure 2.6.

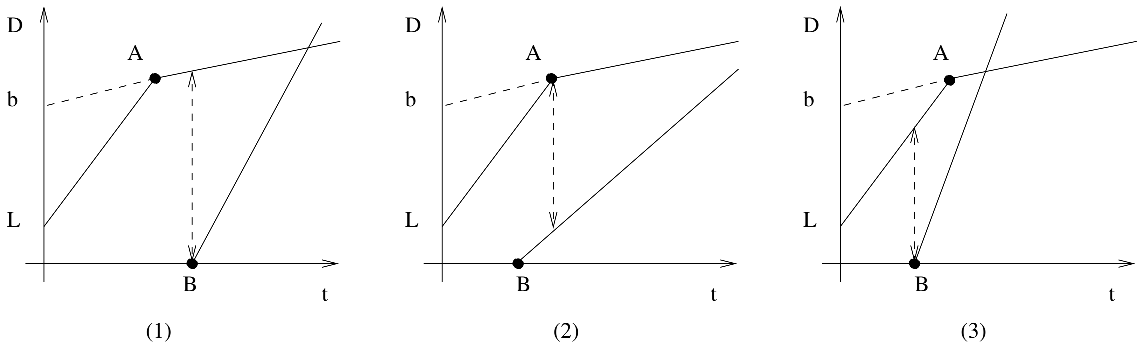


FIG. 2.6 Les trois situations pour le calcul des tampons

Tenant compte de ces trois cas, le calcul de sa longueur (que nous ne détaillerons pas ici) nous amène à :

$$B = L + \frac{(p-X)}{(p-r)} \cdot (b-L) + \sum_{i=1}^{n} \left(\frac{C_i}{R} + D_i \right) X \tag{2.4}$$

avec

$$X = \begin{cases} r \quad si \quad \dfrac{(b-L)}{(p-r)} \leq \sum_{i=1}^{n} \left(\dfrac{C_i}{R} + D_i \right) & (cas\ 1) \\ R \quad si \quad \dfrac{(b-L)}{(p-r)} > \sum_{i=1}^{n} \left(\dfrac{C_i}{R} + D_i \right) \ et \ \ p > R & (cas\ 2) \\ p \quad sinon & (cas\ 3) \end{cases}$$

Notons que ceci est valable pour un flot respectant son TSPEC. Or, les nœuds successifs traversés peuvent avoir pour effet d'augmenter la taille des rafales, auquel cas des tampons de taille supérieure seront nécessaires. Ceci est prévu par la spécification de GS [SPG97] qui propose également de re-lisser le trafic à certains nœuds (*reshaping points*) pour limiter l'accroissement de la taille des tampons.

Liaison délai-débit

Il est intéressant de noter qu'avec ce type d'approche, délai et réservation de bande passante sont couplés. C'est une propriété qui peut poser quelques problèmes car elle implique qu'on ait besoin d'une forte réservation pour garantir un délai faible, même si le débit du flot concerné est très faible. Dans le chapitre 4, section 4.5, nous proposons une approche possible, basée sur l'agrégation, pour réduire ce problème.

La technique d'ordonnancement "Delay-EDD" (*Earliest-Due-Date*) [FV90] permet de fournir des garanties de délai et de bande passante complètement découplées. Cependant elle nécessite une réservation au débit crête. Cette technique utilise l'ordonnancement EDD dans lequel les paquets sont servis dans l'ordre des *deadlines* qui leur sont assignées, et spécifie deux autres mécanismes :

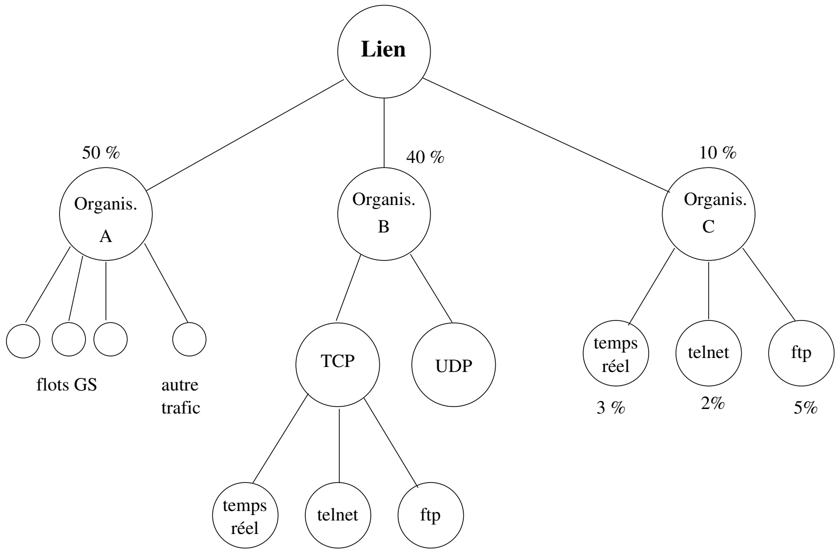


Fig. 2.7 Modèle à intégration de service hiérarchique

- le contrôle d'admission vérifie que la somme des débits crêtes de tous les flots ne dépasse pas la capacité du lien,
- la *deadline* d'un paquet est assignée à la date à laquelle il serait émis si la source respectait son contrat de trafic (débit crête). Cela revient à mettre en œuvre un lissage du trafic quand nécessaire (i.e. quand il y a contention entre les flots).

L'implémentation est comparable à celle d'un *weighted fair queuing* (voir section suivante) car il faut gérer des estampilles par flot. La complexité liée au calcul du temps virtuel est cependant absente.

2.5 LA DISCIPLINE DE SERVICE *WEIGHTED FAIR QUEUING*

La discipline de service à "partage équitable pondéré" (*weighted fair queuing* souvent noté WFQ) a suscité beaucoup de travaux depuis de nombreuses années. La raison de sa popularité tient sans aucun doute à sa flexibilité : elle peut être utilisée pour le contrôle de congestion (par partage équitable de la bande passante entre tous les flots), pour fournir des garanties de débit, de délai (comme nous le montrons en 2.4), pour faire du partage de lien (entre organisations, classes de trafic etc) et, pour finir, elle peut combiner (de façon plus ou moins efficace selon les variantes) toutes ces fonctionnalités de manière hiérarchique pour un modèle à intégration de service complet, comme illustré figure 2.7.

Dans cet exemple, un lien est partagé entre trois organisations A, B et C qui subdivisent elles-mêmes leur part. L'organisation A utilise des applications temps réel et alloue donc des parts de bande passante individuellement à ses flots GS, le reste du trafic étant traité comme un seul flot. L'organisation B traite séparément ses trafics TCP et UDP (afin d'empêcher les flots UDP d'accaparer toute la bande passante) et partage à nouveau la bande passante entre différents types d'applications utilisant TCP. L'organisation C se contente de séparer son trafic en trois classes d'applications. Ce type de modèle est proposé par exemple dans [FJ95].

2.5.1 *Fair Queuing*

La discipline à "partage équitable" (*fair queuing*) est basée sur une idée proposée par Nagle [Nag87] pour mettre en place un contrôle de congestion dans les nœuds du réseau. Considérant qu'avec les disciplines de service classiques FCFS[11], la meilleure stratégie individuelle pour chaque source (envoyer le plus de données possible pour utiliser plus de bande passante) est très mauvaise pour le réseau dans son ensemble (c'est à dire qu'elle crée de la congestion), Naggle proposait de servir les paquets de chaque source en tourniquet (*round robin*). De cette façon, la meilleure stratégie pour une source n'est plus de transmettre le plus possible mais de consommer *exactement* sa part équitable de bande passante. Si une source transmet plus que ce que l'ordonnanceur lui fournit, ses délais vont augmenter, puis elle va perdre des paquets. De plus, cela n'a aucune influence sur le service fourni aux autres sources, ce mécanisme assurant l'*isolation* entre les sources.

Un défaut de l'algorithme de Nagle est que l'équité est définie en nombre de paquets et non en quantité de données. Les sources utilisant des paquets de grande taille sont donc favorisées. C'est à partir de cette constatation que Demers et al ont proposé un mécanisme émulant un *tourniquet bit à bit*, nommé *fair queuing* [DKS89]. La version pondérée de cet algorithme (WFQ pour *weighted fair queuing*) a plus tard été réinventée sous le nom PGPS par Parekh et Gallager qui ont prouvé un certain nombre de propriétés intéressantes. Comme WFQ et PGPS sont le même algorithme, nous reportons la description de l'algorithme au paragraphe consacré à PGPS (2.5.3).

2.5.2 *Generalized Processor Sharing*

Dans [PG93], Parekh et Gallager définissent la discipline de service *Generalized Processor Sharing* (GPS), un modèle théorique de multiplexage de flots (ou sessions) de la façon suivante.

Un serveur GPS conserve le travail et opère à un débit fixé C. Il est caractérisé par des nombres réels positifs $\phi_1, \phi_2, ...\phi_N$. Soit $S_i(\tau, t)$ la quantité de service reçue par la session i dans l'intervalle de temps $[\tau, t]$. Le serveur GPS est défini tel que

$$\frac{S_i(\tau,t)}{S_j(\tau,t)} \geq \frac{\phi_i}{\phi_j}, j = 1, 2, ...N$$

pour toutes les sessions i actives (i.e. qui ont du trafic en attente) sur tout l'intervalle $[\tau, t]$. En sommant

$$S_i(\tau,t) \cdot \phi_j \geq \phi_i \cdot S_j(\tau,t)$$

pour tous les j on obtient

[11] First Come, First Served

$$S_i(\tau,t) \cdot \sum \phi_j \geq \phi_i \cdot \sum S_j(\tau,t)$$

$$S_i(\tau,t) \geq \frac{\phi_i}{\sum \phi_j} \cdot C \cdot (t-\tau)$$

En conséquence, une part de la bande passante est garantie à la session i :

$$B_i = \frac{\phi_i}{\sum_j \phi_j} \cdot C$$

Si l'on assure que $\sum_j \phi_j \leq 1$ (par exemple par un contrôle d'admission), cette garantie s'exprime alors en terme de bande passante *absolue*.

GPS est une discipline idéalisée qui suppose que le serveur peut servir plusieurs sessions simultanément. Il est important de noter que si cela permet de qualifier ce modèle de "fluide", il manipule cependant des paquets. En particulier le temps de traversé du serveur par un paquet de taille L appartenant à la session i est égal à sa taille divisée par la bande passante allouée à la session i, soit son *délai de paquetisation* L/B_i.

2.5.3 *Packet by Packet GPS* et variantes

PGPS - WFQ

Parekh et Gallager proposent une émulation de GPS pour des réseaux réels, qui ne peuvent transmettre qu'un paquet à la fois. Ils l'appellent PGPS pour *packet by packet GPS*. L'idée est de servir les paquets en les triant sur leur date de *fin de transmission* sous GPS, c'est à dire que les paquets seront émis dans l'ordre où ils finissent (et non pas commencent) leur transmission avec un serveur GPS.

Ils montrent que le retard de service d'un paquet p sous PGPS par rapport à GPS est borné par le temps de transmission d'un paquet de taille maximum :

$$F^p_{PGPS} - F^p_{GPS} \leq \frac{L_{max}}{C}$$

Il s'agit donc de trier les paquets avec une estampille correspondant à la date de leur fin de transmission *en GPS* exprimée en "temps virtuel". Ce "temps virtuel" $V(t)$ est défini de façon que le service fourni à un flot actif soit constant dans le référentiel de ce temps virtuel. Cela signifie par exemple que si tous les flots sont actifs, le temps virtuel progresse à la même vitesse que le temps réel ; si la moitié seulement des flots sont actifs, ils recoivent un service double, le temps virtuel progresse deux fois plus vite que le temps réel et le service en temps virtuel reste donc constant.

Posons que le $k^{ième}$ paquet de la session i arrive au temps a_i^k et a pour taille L_i^k. Appelons S_i^k et F_i^k les dates respectives en temps virtuel auxquelles le paquet commence et termine son service avec le serveur GPS (F_i^0 étant initialisé à zéro), on a :

$$S_i^k = max(F_i^{k-1}, V(a_i^k))$$

$$F_i^k = S_i^k + \frac{L_i^k}{\phi_i}$$

L'opération max pour le S_i^k assure que le paquet ne commence pas son service avant que le paquet précédent du même flot n'ait terminé sa transmission (auquel cas le serveur GPS traiterait simultanément deux paquets du même flot). Le calcul du temps virtuel $V(a_i^k)$ pose un problème de complexité car il doit être calculé pour *chaque* paquet et son calcul est assez complexe à cause des “effacements en cascade” (*iterated deletions*) [Kes97, page 242]. Il existe cependant d'autres méthodes de calcul (ou plutôt d'*approximation*) que nous décrivons plus bas.

Comme nous l'avons dit précédemment, il se trouve que PGPS et WFQ sont le même algorithme approximant le GPS. Les auteurs du WFQ le concevaient comme une émulation d'un tourniquet bit à bit, aussi la notion de temps virtuel est remplacée par le numéro du cycle de service (à chaque cycle un bit est servi pour chaque session active). D'autres approximations du GPS existent, nous en décrivons quelques unes dans la suite.

SCFQ

Self-Clocked Fair Queuing [Gol94] résout le problème de complexité du calcul du temps virtuel en fixant sa valeur à l'estampille (c'est à dire le *temps de fin de service* sous GPS) du paquet en cours de service. L'approximation du GPS est cependant moins bonne que WFQ, les bornes de délai étant notablement augmentées.

WF^{2}Q

Dans [BZ96b] Bennett et Zhang proposent *Worst-case Fair Weighted Fair Queuing* (WF^{2}Q), une émulation de GPS meilleure que PGPS. Ils montrent en effet qu'avec PGPS, si le retard de service par rapport à GPS est borné, l'*avance* de service, elle, ne l'est pas. Parekh et Gallager cherchaient à montrer les garanties de délais que l'on pouvait obtenir avec PGPS, ils ne se sont donc pas soucié de cet aspect.

WF^{2}Q borne aussi bien l'avance que le retard de service par rapport à GPS. Bennett et Zhang définissent un “indice de pire équité” (*worst-case-fair index*) permettant d'exprimer quantitativement le degré d'équité d'une approximation du GPS.

Une variante, appelée WF^{2}Q+ [BZ96a] est adaptée au cas des serveurs à capacité variable (voir ci-dessous).

SFQ

Dans le *start-time fair queuing* [GVC96], les paquets sont triés par temps de début et non pas de fin de service (S_i^k). Le temps virtuel est approximé par le S_i^k du paquet en cours de service. En plus d'avoir la même complexité que SCFQ, SFQ fournit des garanties de délais plus faibles pour les flots à faible débit. Il se comporte de plus correctement en cas de variation de la capacité du serveur (débit du lien en sortie) ce qui est nécessaire dans le cas d'un modèle de service hiérarchique (ou pour des liens sans fil dont la capacité est fortement susceptible de varier). Notons que ce n'est pas le cas de l'algorithme original PGPS/WFQ qui n'émule pas correctement un serveur GPS à débit variable.

VC

Virtual Clock [Zha90] n'émule pas le GPS mais le TDM (*time division multiplexing*). L'algorithme est semblable à celui des *fair queuing*, la fonction de temps virtuel étant ici remplacée par le temps réel. Si cette discipline ne fournit pas un partage équitable de la bande passante, elle présente par contre les mêmes propriétés pour les garanties de délai que WFQ [FP95], sans la complexité liée au calcul du temps virtuel.

2.5.4 Évaluation des algorithmes du type *fair queuing*

Nous avons vu dans les paragraphes précédents différents algorithmes d'approximation du GPS. L'évaluation de ces algorithmes peut se faire selon les critères suivants :

- complexité d'implémentation,
- qualité de l'approximation en terme d'équité,
- qualité de l'approximation en terme de délais,
- comportement avec un serveur à débit variable,
- adéquation à une implémentation matérielle.

Les critères décisifs seront fonction de l'utilisation recherchée.

2.6 CONCLUSION DU CHAPITRE

Depuis de nombreuses années que des recherches sont faites sur la qualité de service dans les réseaux à commutation de paquets, beaucoup de progrès ont été faits. Comme nous l'avons montré, on connaît maintenant assez bien les mécanismes à mettre en place dans le plan de données pour fournir les garanties dont ont besoin les applications de type temps réel ou multimédias ; garanties de bande passante, garanties de délai.

En ce qui concerne le plan de contrôle, RSVP semble arriver à maturité et des implémentations sont désormais disponibles. En somme, l'ensemble des éléments nécessaires à la gestion de la qualité de service dans les réseaux à commutation de paquets sont désormais assez bien connus.

Cependant, l'évolution des réseaux vers le haut débit apporte de nouveaux problèmes liés aux technologies utilisées et aux contraintes spécifiques d'un environnement haut débit. Les mécanismes et modèles que nous avons décrits ont été conçus essentiellement dans le contexte de réseaux classiques et ne prennent donc pas en compte ces contraintes (même si DiffServ tente de résoudre le problème de scalabilité).

La transposition et l'adaptation de ces mécanismes dans le contexte de ces réseaux haut débit soulève un certain nombre de problèmes. Sont à prendre en compte d'une part les technologies particulières mises en œuvre et les contraintes spécifiques au haut débit, d'autre part les problèmes liés au facteur d'échelle.

Chapitre 3

La qualité de service dans les réseaux à commutation IP

3.1 INTRODUCTION DU CHAPITRE

Comme nous l'avons vu au chapitre 2, les mécanismes nécessaires à la fourniture de la qualité de service dans les réseaux à commutation de paquets sont maintenant assez bien connus. Cependant, le développement de nouvelles technologies pour les réseaux haut débit rend nécessaire l'adaptation de ces techniques pour ces nouvelles technologies. Il est en effet nécessaire, pour obtenir une qualité de service de bout en bout, que les mécanismes de qualité de service soient mis en place également dans les réseaux d'interconnexion.

Une technologie de réseau haut débit apparue récemment est la *commutation IP* (ou plus généralement la *commutation par labels*). Nous nous proposons d'adapter le modèle défini par l'IntServ (et en particulier le protocole RSVP) au contexte de cette solution de façon à obtenir une technologie de réseau haut débit intégrant la gestion de la qualité de service.

Nous commençons par décrire le principe de la commutation IP et plus particulièrement la solution IP switching. Puis nous étudions la façon d'y intégrer la qualité de service pour obtenir notre proposition que nous appelons "RSVP switching". Pour celle-ci nous proposons des ajouts simples au protocole RSVP et une technique d'ordonnancement permettant d'optimiser l'allocation des ressources. Pour finir, nous évaluons notre proposition par le biais de simulations, de la réalisation d'un prototype et enfin d'une étude de scalabilité réalisée sur une implémentation de RSVP.

3.2 LA COMMUTATION IP

3.2.1 Motivation

Depuis plusieurs années des travaux importants sont menés pour utiliser le protocole IP sur les réseaux ATM. En effet, la flexibilité et la robustesse d'IP associées au développement spectaculaire de l'Internet l'ont imposé comme protocole de couche réseau quasi-universel. D'autre part les investissements industriels massifs dans ATM ont résulté en l'apparition de matériel très performant à (relativement) bas prix et ont ainsi fait d'ATM la technologie par excellence pour les réseaux haut débit. De plus, ATM a été largement adopté dans le milieu des télécommunications.

De nombreuses solutions existent, nous pouvons les classer en deux familles selon l'approche utilisée :

- l'approche "classique" où IP fonctionne au dessus de la signalisation ATM définie par l'ATM forum,
- l'approche "commutation IP" où IP fonctionne directement sur le matériel ATM, la signalisation de l'ATM forum étant éliminée.

L'approche "classique", chronologiquement la première, inclut aussi bien des solutions proposées par l'IETF, telles que *Classical IP over ATM* (CLIP) [Lau93, Atk94], puis [LH98] et son extension au multicast *Multicast Address Resolution Server* (MARS) [Arm96, TA97], que des solutions développées par l'ATM forum, comme *LAN Emulation* (LANE) [FM96] ou *MultiProtocol Over ATM* (MPOA) [MPO97].

La mise en œuvre et l'utilisabilité de ces solutions sont rendues difficiles pour deux raisons :

- l'inadéquation entre le modèle utilisé par IP et celui de l'ATM forum : IP fonctionne en mode non connecté alors que les services ATM forum sont en mode connecté.
- le recouvrement de fonctionnalités : il y a duplication des fonctionnalités entre la signalisation ATM forum et la signalisation IP. La superposition des deux signalisations fait qu'IP ne peut pas connaître la topologie effective du réseau.

Si ces problèmes ne sont pas insurmontables, leurs conséquences sont visibles dans toutes ces solutions par :

- une complexité protocolaire très importante,
- l'inefficacité dans l'utilisation des ressources.

L'approche "commutation IP", au contraire, élimine la signalisation ATM forum, IP ayant ainsi directement accès aux fonctionnalités de niveau liaison du matériel ATM. Nous décrivons le principe de cette approche dans la section suivante.

3.2.2 ATM sous IP

Comme le montre la figure 3.1, dans les solutions de type "commutation IP" IP est utilisé directement sur le matériel ATM (avec cependant une couche d'adaptation réduite au strict nécessaire et donc très légère) ce qui évite l'empilement des couches et des fonctionnalités des solutions “classiques”.

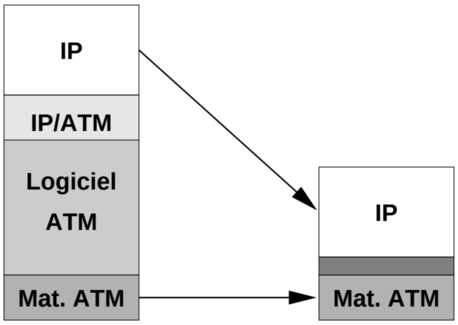


Fig. 3.1 Piles protocolaires des deux approches

Un élément d'interconnexion est un routeur IP qui contrôle un commutateur ATM. Il peut ainsi traiter le trafic classiquement au niveau IP (*forwarding*, nous parlerons de trafic *routé*) ou au niveau ATM (*switching*, nous parlerons de trafic *commuté*).

Pour être commuté, le trafic doit arriver porteur d'un label correspondant à une association (label amont, label aval, lien aval) pour le lien amont considéré. Dans ce cas, le lien et le label de sortie sont accessibles directement par simple indirection dans la table de commutation, on dit que le trafic emprunte un “raccourci” (*cut-through*). La commutation de cellules ATM peut avoir lieu directement, alors que l'absence de cette information nécessite le réassemblage du paquet IP et la consultation des tables de routage IP.

La mise en place des labels peut être :

- *orientée flot* (*flow driven*), c'est à dire basée sur les flots de données qui traversent le nœud. Les labels sont alors associés aux flots.
- *orientée topologie* (*topology driven*), c'est à dire basée sur les routes suivies par les données. Les labels sont alors mis en place en fonction des informations de routage.

Le principe de la commutation IP a été généralisé à tous types de réseaux sous l'appellation “commutation par labels” (*label switching*) qui est étudiée par le groupe de travail *Multi Protocol Label Switching* (MPLS[1]) de l'IETF [C^+99]. L'utilisation des labels est bénéfique même si le nœud ne dispose pas d'un moteur de commutation. Ils permettent d'accélérer le traitement des paquets en évitant l'opération de “recherche de plus long préfixe” (*longest prefix match*) du routage classique.

3.2.3 La solution IP switching

IP switching est chronologiquement la première solution à avoir été présentée et même commercialisée. Elle est *orientée flot*, donc initialement tout le trafic est routé, puis des labels sont associés aux flots qui peuvent alors être commutés. Comme illustré sur la figure

[1] `http://www.ietf.org/html.charters/mpls-charter.html`

3.2, chaque *IP switch* choisit des flots[2] susceptibles d'être commutés, leur attribue un label et transmet l'information (information de *redirection* : description d'un flot et label associé) à son voisin amont au moyen du protocole IFMP (Ipsilon Flow Management Protocol [N+96a]). Celui-ci enverra désormais les paquets de ce flot sur le VPI/VCI correspondant à ce label. Après réception d'une information de redirection pour ce flot depuis l'IP switch aval, deux labels amont et aval identifient le flot et il devient possible de le commuter par simple insertion d'une entrée dans la table de commutation. C'est le rôle du protocole GSMP (General Switch Management Protocol [N+96b]). Celui-ci fournit également au routeur les informations sur les caractéristiques pertinentes du commutateur. Dans sa deuxième version [N+98], il permet également de spécifier des paramètres de qualité de service associés à chaque entrée de commutation. Les informations de redirection sont stockées dans des états relachés (éphémères). Ces états doivent être rafraîchis régulièrement sous peine d'expirer, ce qui assure simplicité et robustesse du mécanisme. L'encapsulation des paquets IP avec label dans les trames ATM est définie dans le RFC 1954 [N+96c].

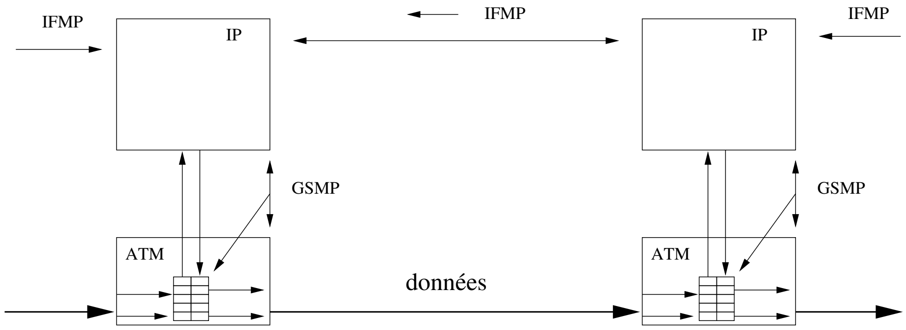


Fig. 3.2 IP switching

La décision d'attribuer un label à un flot IP est prise localement par le *module de classification de flots*. Pour obtenir des performances élevées, il est bien évident que l'IP switch doit commuter le plus de trafic possible. Mais l'établissement d'un raccourci demandant un certain *overhead* et un certain délai, il n'est rentable de commuter que les flots suffisament longs pour que le gain obtenu par la commutation excède l'*overhead* nécessaire à la mise en place des labels.

C'est le rôle du classificateur de flots de déterminer quels sont les flots qui doivent être commutés, en "devinant" quelle sera leur longueur à partir des informations telles que le protocole utilisé ou le nombre de paquets déjà routés pour ce flot [NLM96, NML]. Des simulations basées sur des traces de trafic Internet [LM97] ont montré que l'approche était efficace pour le trafic Internet tel qu'il était il y a deux ans. Mais du fait de l'évolution rapide de l'Internet, ces caractéristiques changent rapidement et il est donc difficile de prévoir la validité de cette approche à moyen terme.

[2]Plusieurs granularités sont possibles pour la définition des flots : paire d'adresses IP et ports sources/destination ou paires d'adresses IP uniquement [NMLH97].

3.3 NOTRE PROPOSITION : RSVP SWITCHING

3.3.1 Intégration de RSVP

Dans IP switching, le rôle du *classificateur de flots* est de repérer les flots de longue durée afin qu'ils soient commutés. Il se base pour cela sur diverses informations qui fournissent *implicitement* des indications sur la durée du flot. Nous considérons que le fait qu'une garantie de service soit demandée pour un flot est une indication *explicite* que ce flot sera de longue durée, la mise en place de la réservation à l'aide du protocole RSVP demandant elle même un certain délai. Nous décidons donc de commuter tous les flots pour lesquels doivent être fournies des garanties de service. Cela n'implique cependant absolument pas que tous les flots "best-effort" doivent être routés, mais plutôt que l'on ajoute un critère pour la décision de commuter un flot. Commuter *uniquement* les flots à QoS n'est souhaitable que si ces flots constituent la majeure partie du trafic, le but étant toujours d'obtenir le débit le plus élevé possible en profitant au maximum des capacités du commutateur.

Si le fait de commuter les flots à QoS est bénéfique pour la performance globale de l'IP switch en terme de bande passante totale, il l'est aussi pour l'efficacité dans la fourniture des garanties de services. Le moteur de commutation engendrant des délais bien moins élevés que le routage au niveau IP, ces flots subiront moins de délai et auront donc besoin de moins de ressources pour recevoir le même service, indépendamment des mécanismes mis en œuvre pour garantir le service demandé. Nous détaillons ce point dans la section suivante (3.3.2).

Enfin, il apparaît clairement que RSVP et IFMP utilisant tous deux des *états relachés* et transmettant des informations de nœud à nœud le long du chemin, leur intégration peut être réalisée de manière assez directe. L'information de redirection (i.e. un label) peut très facilement être ajoutée aux informations de réservation et transportée dans les paquets RESV. De cette façon, le chemin commuté s'établit en même temps que les réservations. Il est d'ailleurs à noter que dans le cas de services à garanties de délai (comme le GS) la séparation entre l'établissement de la réservation et la mise en place du chemin commuté compliquerait singulièrement les choses, les caractéristiques (déviation par rapport à un modèle fluide, voir 2.4) des chemins routés et commutés n'étant pas les mêmes.

Architecture d'un RSVP switch

La figure 3.3 montre l'architecture du RSVP switch ainsi défini. On y retrouve tous les éléments d'un routeur à intégration de services (classificateur de paquets, module RSVP, ordonnanceur) et d'un IP switch classique (module GSMP ainsi que classificateur de flots et module IFMP pour le trafic BE). Bien entendu, le commutateur ATM doit être équipé de mécanismes de contrôle de trafic par flot (ordonnanceur de cellules).

Modifications de RSVP

Les modifications de RSVP portent sur la syntaxe des messages et l'interface avec le contrôle de trafic. Elles sont décrites en détail dans [Die98], nous les décrivons succinctement ici.

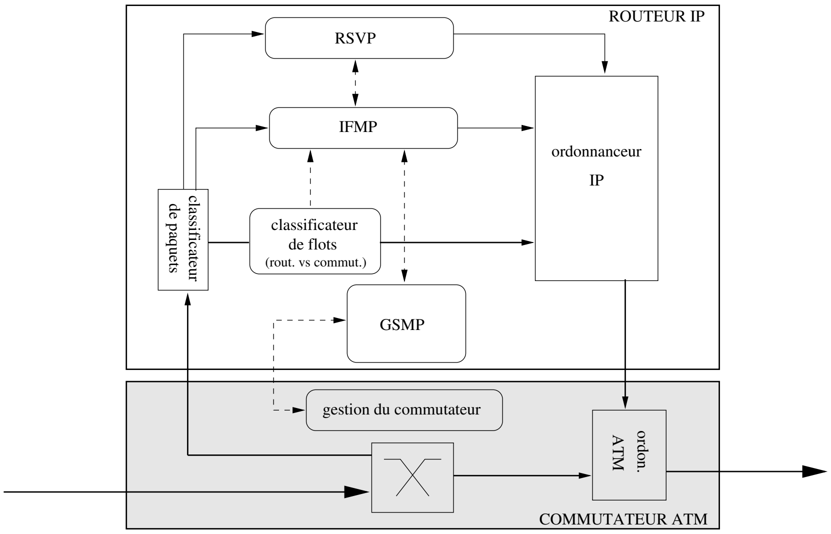


Fig. 3.3 Architecture d'un RSVP switch

Ajout de l'objet LABEL Pour que RSVP transporte les informations de redirection, nous proposons de définir un objet RSVP "LABEL" représenté figure 3.4. Une valeur non utilisée a été choisie pour l'identifieur (**Class**=40). Nous définissons le label sur 32 bits, donc **C-type**=1. L'objet complet occupe 8 octets et peut contenir un label ATM basé sur le VPI/VCI ou celui d'une autre technologie. Comme nous l'avons dit, le label sera manipulé avec les informations de réservation, donc transporté dans les messages RESV. Il identifie un flot de données et sera donc placé à côté de l'objet FILTER_SPEC. Afin de maintenir la compatiblité avec la version sans label de RSVP, cet objet est défini comme étant optionnel dans la structure du message. Ainsi un flot pourra traverser de manière transparente des zones avec et sans gestion des labels.

Length (= 8 octets) sur 16 bits	**Class** (= 40) 8 bits	**C-type** (= 1) 8 bits
Label sur 32 bits		

Fig. 3.4 Objet LABEL

Une difficulté se pose avec le modèle multipoint étendu de RSVP, qui demande des fonctionnalités de *filtrage* et d'*agrégation* (voir 2.2.4) qui peuvent difficilement être réalisées au niveau de la couche liaison. Une solution possible est de reporter ces fonctionnalités hors de la zone commutée, en effectuant des *pre-filtrage* et *post-agrégation*. Le multicast n'étant pas spécifiquement notre propos, nous ne détaillerons pas ces méthodes qui ont été décrites dans [FF99].

Modification de l'interface avec le noyau La communication avec le contrôle de trafic implémenté dans le noyau du système d'exploitation se fait par l'interface suivante, définie dans [BZB+97].

```
rhandle  TC_AddFlowspec(OIf, FLOWSPEC, SENDER_TSPEC,
                        ADSPEC, flags, FLOWSPEC);
FLOWSPEC TC_ModFlowspec(OIf, rhandle, FLOWSPEC, SENDER_TSPEC,
                        ADSPEC, flags, FLOWSPEC);
         TC_DelFlowspec(OIf, rhandle);

fhandle  TC_AddFilter(OIf, rhandle, Session, FILTER_SPEC);
         TC_DelFilter(OIf, fhandle);
```

Les fonctions `TC_AddFlowspec`, `TC_ModFlowspec` et `TC_DelFlowspec` agissent sur les paramètres de l'ordonnanceur de paquets, et les fonctions `TC_AddFilter` et `TC_DelFilter`, sur les paramètres du classificateur de paquets. Les *Flowspecs* et *Filterspecs* existant peuvent être référencés par des *handles* retournés par les fonctions.

La gestion des labels a nécessité de légers ajouts à cette interface qui devient :

```
rhandle  TCrsw_AddFlowspec(OIf, FLOWSPEC, SENDER_TSPEC,
                           ADSPEC, flags, FLOWSPEC);
FLOWSPEC TCrsw_ModFlowspec(OIf, rhandle, FLOWSPEC, SENDER_TSPEC,
                           ADSPEC, flags, FLOWSPEC);
         TCrsw_DelFlowspec(OIf, rhandle);

fhandle  TCrsw_AddFilter(OIf, rhandle, Session, FILTER_SPEC);
fhandle  TCrsw_AddFilter(OIf, rhandle, Session, FILTER_SPEC, LabelSwap);
         TCrsw_DelFilter(OIf, fhandle);

LABEL    TCrsw_GetLabel(IIf);
         TCrsw_FreeLabel(LABEL);
```

L'ajout du préfixe "`rsw`" aux noms de fonctions correspond à une interface spécialisée pour la technologie réseau RSVP switching. La nouvelle version de `TCrsw_AddFilter` permet de spécifier une paire de labels pour l'établissement du raccourci. L'espace des labels est géré par le noyau, `TCrsw_GetLabel` renvoie à RSVP un label libre et `TCrsw_FreeLabel` permet de libérer un label qui n'est plus utilisé.

Situation de l'approche

Étant partis d'une approche de la commutation IP *orientée flot* nous avons construit une approche basée sur RSVP que nous appelons *orientée session* (*session driven*) car basée sur les sessions RSVP.

Il peut paraître naturel de l'assimiler à l'approche orientée flot puisque les labels sont associés aux flots de données par l'intermédiaire de RSVP. Cependant, l'approche orientée topologie, initialement basée sur les informations de routage a été étendue pour pouvoir tenir compte de tout type de signalisation ce qui en fait une approche *orientée signalisation* (*control driven*). Notre approche orientée session peut donc être rattachée aussi bien à l'approche orientée flot (car basée sur les flots de données) qu'à l'approche orientée signalisation (car basée sur la signalisation RSVP).

3.3.2 Implémentation des classes de service

Efficacité du *Guaranteed Service*

Nous détaillons ici l'intérêt de commuter les flots demandant des garanties de délai. Nous avons vu page 31 que le délai donné dans la formule (2.1) est composé de deux parties. La première est le délai d'écoulement au débit de la réservation d'une rafale de taille maximale, la deuxième les délais ajoutés dans chaque nœud à cause des effets de paquetisation et de serialisation. Ces deux effets sont directement liés aux tailles de paquets[3] et seront donc considérablement réduits dans le cas d'un ordonnanceur de cellules. Le gain sera bien entendu particulièrement important pour les flots dont la majeure partie du délai est apporté par le deuxième terme, c'est à dire les flots à petites rafales ou traversant un grand nombre de nœuds. Nous illustrons ceci dans le cas de deux flots ayant pour caractéristiques respectives (p=25 Mb/s, r=300 kb/s, b=20 ko) et (p=25 Mb/s, r=300 kb/s, b=0 ko), traversant n=10 nœuds. Le débit des liens est C=155 Mb/s. Nous comparons la réservation nécessaire dans les cas du routage de paquets IP de 1024 octets et de la commutation de cellules ATM (53 octets) en fonction du délai désiré. Notez que dans ces courbes, nous ne tenons pas compte de la condition de stabilité $R > r$. La réservation à affecter est donc le maximum de la valeur donnée par la courbe et du débit moyen du flot.

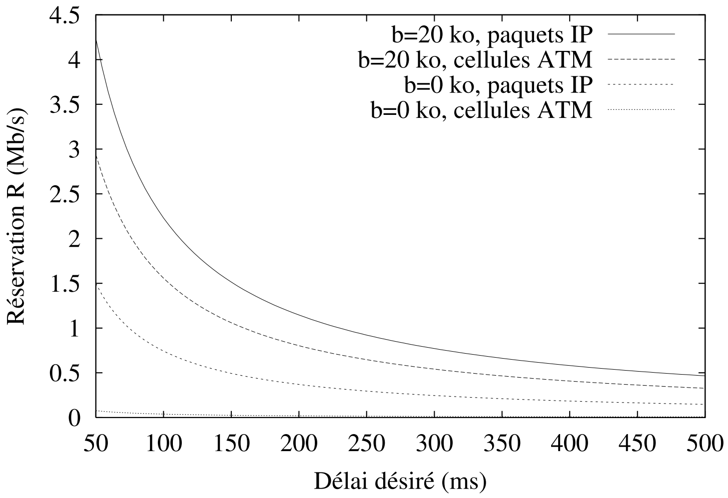


Fig. 3.5 Réservation en fonction du délai désiré

Sur la figure 3.5, les deux courbes supérieures concernent le premier flot (taille maximum de rafale de 20 ko). La majeure partie du délai est alors dû au temps d'écoulement de la rafale et la différence entre les cas IP et ATM est relativement faible. Par contre, pour le deuxième flot, dont la taille de rafale est nulle, le délai dépend uniquement des tailles de paquets et le gain est très important.

[3] Cela est vrai aussi bien dans le cas d'un WFQ que dans toutes les disciplines à contrôle de débit (RCS, *Rate Controlled Service*) [G+96].

Ceci implique que pour une garantie de délai donnée, la quantité de ressources à réserver sera (éventuellement beaucoup) plus faible dans le cas de la commutation. La *quantité de qualité de service* que le réseau peut fournir pour un dimensionnement (quantité de ressources) donné est donc plus importante.

Le service BR : *Bandwidth Recovery*

Motivation Bien que nous venions de montrer que l'ordonnancement au niveau cellule pouvait considérablement améliorer les choses, il n'en reste pas moins qu'avec un ordonnanceur de type *weighted fair queuing* la garantie de délai est directement liée à la bande passante réservée, ce qui fait que les flots à faible débit demandant des délais courts devront réserver beaucoup de ressources. Avec un ordonnanceur qui découple réservation de bande passante et garantie de délai, la réservation R est également toujours plus grande que le débit moyen du flot r, et une quantité de bande passante $R - r$ est alors réservée mais non utilisée. Bien sur cette bande passante n'est pas strictement “perdue” dans le sens où elle reste disponible pour le trafic *best-effort* (si l'ordonnanceur conserve le travail). Cependant, dans la mesure où elle a été réservée, sa disponibilité est garantie et elle devrait pouvoir être utilisée pour fournir des garanties de services à d'autres flots. Évidemment, le type de service qu'il est possible de garantir est bien plus “faible” que le GS puisque cette bande passante n'est disponible qu'en **moyenne**[4].

Beaucoup d'applications n'ont pas besoin de garanties strictes en terme de délai, mais ont plutôt besoin d'un certain niveau de garantie en bande passante. Nous pensons en particulier à toutes les applications qui ont besoin d'une “interactivité à échelle humaine”, c'est à dire qui ont besoin d'effectuer leurs transferts de données en un temps “raisonnable” (tel que perçu par l'utilisateur). Les butineurs web ou les applications de transfert de fichier en sont un bon exemple.

Aussi nous proposons une classe de service permettant d'utiliser ces sur-réservations pour fournir une garantie de bande passante “à long terme”. L'idée de ce service est similaire à ce qui est présenté dans [G$^+$96]. La différence principale réside dans la façon dont il est implémenté et dont il interagit avec le trafic *best-effort*.

Le type de QoS fournie par cette classe de service peut être globalement compatible avec ce qui est requis pour la classe de service *Controlled Load*. On peut dont la voir comme une implémentation possible du CL, conçue pour être implémentée conjointement avec la classe GS de façon à optimiser l'utilisation des ressources. Elle peut très bien cohabiter avec une autre réalisation du CL dans le cas où la bande passante “gaspillée” par les flots GS ne serait pas suffisante pour satisfaire l'ensemble des flots CL.

Réalisation Pour implémenter cette classe de service, que nous appellerons BR (pour *Bandwidth Recovery*), nous utilisons un ordonnanceur de type *Weighted Fair Queuing* avec des files séparées pour chaque flot GS ou BR. À chaque flot GS est affecté un poids qui détermine la quantité de bande passante R_i allouée à ce flot. Supposons un lien de capacité

[4]Dans le cas d'un flot sans rafale, la sur-réservation dûe aux délais ajoutés par les termes d'erreur C et D des nœuds peut être utilisée pour fournir des garanties identiques à celles fournies par le GS. La solution est d'*agréger* des flots, comme nous le décrivons au chapitre 4, section 4.5.

C_l et N flots GS chacun caractérisé par le TSPEC (b_i, r_i, p_i). Pour garantir le service aux flots GS, l'inégalité suivante doit être respectée :

$$\sum_{i=1}^{N} R_i \leq C_l - \Delta_{\text{BR}} - \Delta_{\text{SIG}} \tag{3.1}$$

Cette inégalité est illustrée sur la partie gauche de la figure 3.6. Dans l'équation précédente, Δ_{BR} et Δ_{SIG} représentent les portions de bande passante réservées respectivement aux flots BR et à la signalisation. Le poids associé au "flot signalisation" est basé sur une estimation de la bande passante nécessaire au trafic de signalisation.

Le poids associé à chaque flot BR, et donc la bande passante réservée, est choisie volontairement très petite. Son rôle est de permettre aux flots BR d'accéder à la bande passante réservée mais non utilisée par les flots GS. Avec une discipline de service de type GPS, une portion de bande passante est garantie à chaque flot, mais si certains n'utilisent pas la totalité de ce qui leur a été réservé, d'autres flots tentant d'utiliser plus que leur propre part se partageront alors cette bande passante. Ainsi dans notre cas les flots GS n'utiliseront pas la totalité de leur part, et les flots BR utiliseront plus que la leur en récupérant la bande passante "gaspillée" par les flots GS comme illustré sur la figure 3.6. La bande passante "récupérée" est répartie équitablement (au sens de l'équité "max-min") entre les flots BR en fonction de leurs poids respectifs.

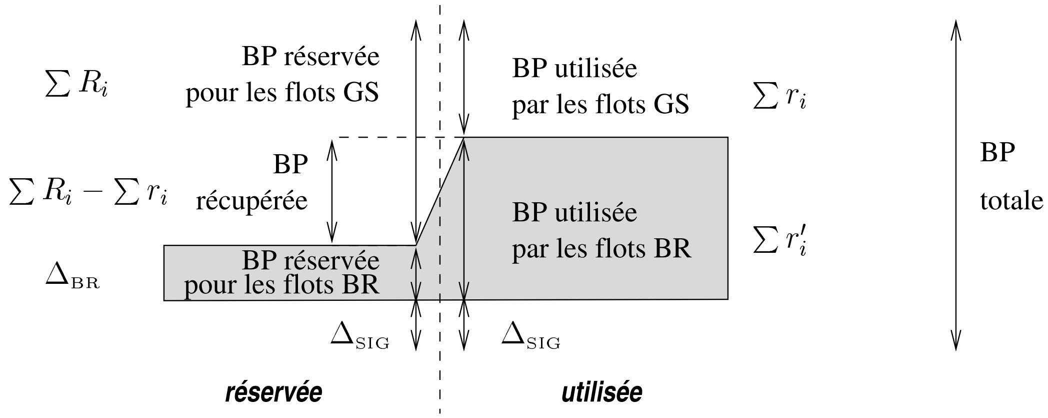


Fig. 3.6 Bande passante réservée et utilisée par les flots GS et BR

Considérons N' flots BR avec un débit moyen égal à r'_i, leur garantir un débit minimum à long terme est possible si :

$$\sum_{i=1}^{N} r_i + \sum_{i=1}^{N'} r'_i \leq C_l - \Delta_{\text{SIG}} \tag{3.2}$$

L'équation (3.2) assure que si les flots GS respectent leur contrat de trafic (ou qu'un mécanisme de polissage est utilisé), les flots BR recevront chacun un débit minimum garanti r_i' sur une échelle de temps raisonnable.

Le trafic *best-effort*

Comme nous l'avons dit, le trafic *best-effort* devra être *routé* ou *commuté* selon les critères utilisés par les IP switches classiques. Dans tous les cas, il sera servi à un niveau de priorité réduite afin de ne pas influer sur le service garanti aux autres flots. Dans toute la discussion précédente sur la réalisation des classes de service, nous avons considéré que toute la bande passante du lien pouvait être utilisée par les flots à QoS. Nous l'avons fait uniquement dans un souci de simplification, il est très fortement souhaitable de réserver une partie du lien pour le trafic *best-effort* ce qui pourra être fait en remplaçant dans les équations (3.2) et éventuellement (3.1) la capacité du lien C_l par la capacité dédiée aux flots à QoS.

L'ordonnanceur utilisé

Le schéma de l'ordonnanceur est donné en figure 3.7. Dans la partie commutation, chacun des flots GS et BR, ainsi que la signalisation sont traités individuellement alors que le trafic *best-effort* est traité dans une seule file de priorité inférieure. Au niveau routage (c'est à dire tout le trafic qui n'est pas commuté, soit une partie du trafic BE et la signalisation) on traite séparément la signalisation et le trafic BE (par exemple en utilisant des priorités) de façon à garantir le passage de la signalisation.

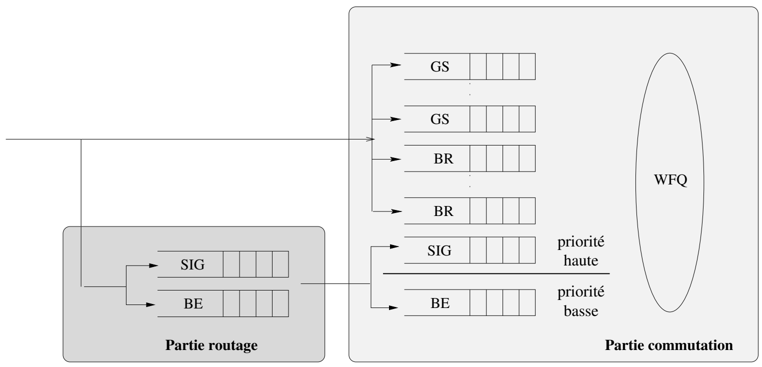


Fig. 3.7 L'ordonnanceur utilisé

3.4 ÉVALUATION DE RSVP SWITCHING

Nous avons procédé à une évaluation de la solution RSVP switching selon trois axes :

- des simulations ont permis de valider l'ordonnanceur proposé et le bon fonctionnement des classes de service GS et BR ;

– la réalisation d'un prototype nous a permis de montrer la faisabilité des mécanismes proposés, en particulier l'inclusion des fonctionnalités d'IFMP dans RSVP ;
– enfin, nous avons voulu évaluer dans quelle mesure le fait d'avoir utilisé une gestion des ressources par flot est une limitation à la scalabilité du modèle. Le plan de contrôle étant à notre avis le plus sensible à ce type de problèmes, nous avons effectué des tests sur le comportement d'une implémentation de RSVP avec un grand nombre de flots.

3.4.1 Simulations pour les classes de service

Afin de valider notre proposition concernant l'implémentation de la classe de service BR, nous avons procédé à des simulations. Le réseau simulé (figure 3.8) est constitué de cinq RSVP switches connectés par des liens à 51 Mb/s et implémentant les services GS et BR avec un ordonnanceur *weighted fair queuing*. À chaque RSVP switch est connectée une station qui génère et reçoit des flots. Un total de neuf flots GS sont répartis de façon que chaque lien inter-switch soit occupé par six flots GS. Trois flots traversent cinq nœuds, deux traversent quatre nœuds, deux traversent trois nœuds et deux traversent deux nœuds.

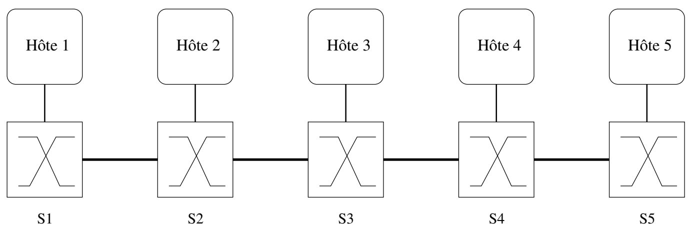


Fig. 3.8 Topologie du réseau simulé

Les sources sont du type ON/OFF filtrées par un double seau percé (*leaky bucket*) pour les contraindre aux spécifications du type (b, r, p) (figure 3.9). La génération des jetons se fait au débit r pour le premier seau et p pour le second. Le premier seau a une profondeur de b, il sert à réguler le débit moyen et les rafales, le deuxième seau a une profonfeur de L, il joue le rôle de régulateur de débit crête. Tous les flots GS sont caractérisés par (b=100 ko, r=3 Mb/s, p=25 Mb/s) et demandent un délai inférieur à 100 ms. Les réservations R nécessaires pour satisfaire ce délai sont données dans le tableau 3.1 avec le poids associé au flot dans l'ordonnanceur et la taille de tampons nécessaires. Le tableau 3.2 montre que la réservation des liens atteint presque 100 %.

longueur du chemin	*R (Mb/s)*	*poids*	*B (nb. cellules)*
5	8.4603	0.16589	1804
4	8.4562	0.16581	1804
3	8.4521	0.16573	1804
2	8.4480	0.16565	1804

Tab. 3.1 Réservations (R), poids (W), taille des tampons (B)

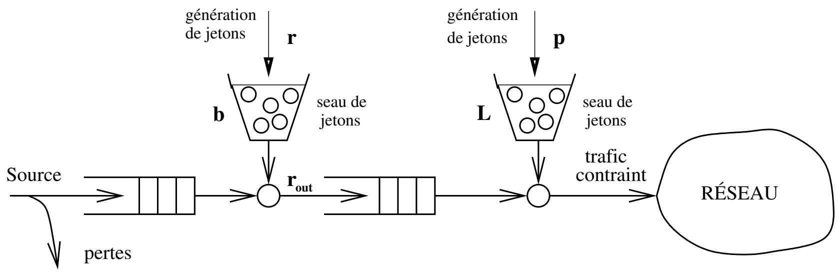


Fig. 3.9 Le double seau percé

lien	*R total (Mb/s)*	*réservation du lien (%)*
S_1-S_2	50.7372	99.49
S_2-S_3	50.7454	99.50
S_3-S_4	50.7454	99.50
S_4-S_5	50.7372	99.49

Tab. 3.2 Réservation des liens

À ces flots GS s'ajoutent cinq flots BR qui traversent la totalité du réseau et ont des débits moyens de 12 Mb/s, 9 Mb/s, 6 Mb/s et 1,5 Mb/s pour les deux derniers. Ces valeurs sont choisies de façon à pousser l'utilisation du réseau au maximum possible, tout en respectant l'équation (3.2). Enfin, du trafic *best-effort* traverse tous les nœuds.

Résultats

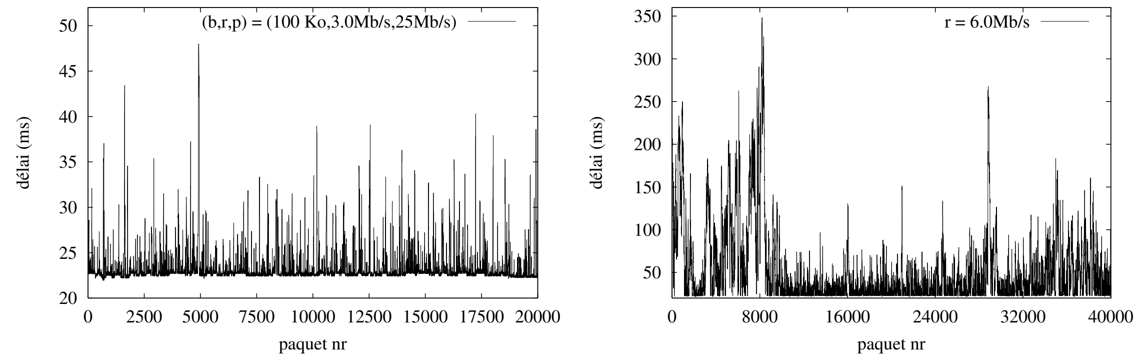
Les résultats des simulations sont montrés figures 3.10 et 3.11. La figure 3.10(a) montre les délais des paquets d'un des flots GS. On constate que la garantie de délai est bien respectée, donc que les flots BR n'ont pas eu d'impact sur la garantie des flots GS. Alors que le délai maximum visé était de 100 ms, le maximum effectif est de moins de 50 ms et la plupart des paquets ont subi un délai inférieur à 30 ms. Ceci est la conséquence de l'aspect très conservateur (*conservative*) des méthodes de calcul (décrites au chapitre 2) utilisées pour le calcul des bornes de délais.

La figure 3.10(b) montre le délai des paquets pour un des flots BR. Les délais sont ici bien plus importants, mais restent tout à fait acceptables pour les types d'applications que nous avons envisagés. Enfin, les figures 3.11(a) et (b) montrent respectivement le débit moyen reçu par tous les flots BR à court terme et à long terme. On voit que le débit converge assez rapidement vers les valeurs recherchées.

3.4.2 Réalisation d'un prototype

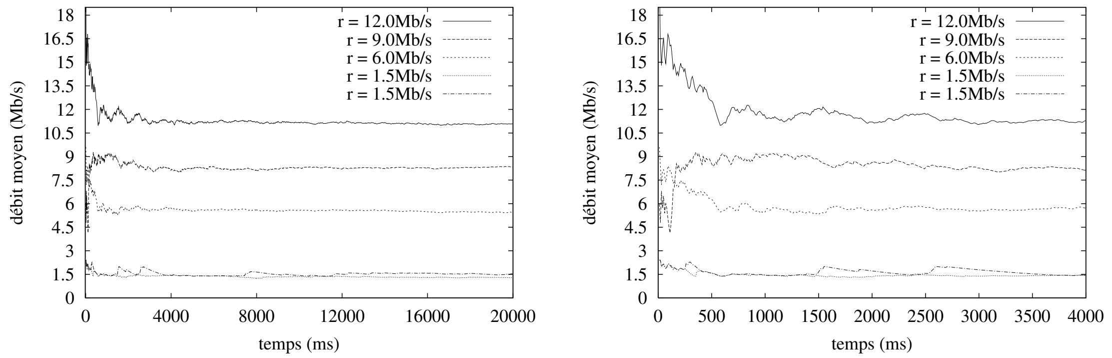
Principe de l'émulation

La réalisation d'un RSVP switch nécessite un accès de bas niveau aux capacités du commutateur ATM, de manière à manipuler la table de commutation et les paramètres de chaque canal virtuel. C'est normalement GSMP qui fournit cette fonctionnalité. Réaliser



(a) Délai flot GS (b) Délai flot BR

Fig. 3.10 Délai



(a) Débit moyen BR à long terme (b) Débit moyen BR à court terme

Fig. 3.11 Débit moyen

une implémentation de GSMP nécessite de pouvoir manipuler facilement la configuration interne du commutateur ce qui n'est évidemment pas notre cas. Au moment de ce travail, il n'existait pas à notre connaissance de matériel ATM ouvert susceptible d'être utilisé pour notre prototype.

Nous avons donc choisi de placer dans le routeur une *émulation logicielle* de la commutation. Deux chemins de traitement sont possibles dans le routeur pour les paquets IP ; ils peuvent être routés normalement ou "commutés" si un raccourci est en place pour le flot concerné. Les paquets "commutés" évitent une partie du traitement normalement appliqué, en particulier l'opération de routage est remplacée par une simple consultation de la *table de commutation* et le paquet est placé directement dans la file d'attente du périphérique de sortie.

Même si ce chemin raccourci peut effectivement améliorer quelque peu les performances (en particulier si la table de routage a un grand nombre d'entrées et que la configuration du trafic est telle que le *cache de routes* n'est pas efficace), il est clair que l'on perd une grande part du gain potentiel de performance, qui provient principalement des capacités matérielles du commutateur ATM. Ce prototype a cependant pour but de montrer la faisabilité et la viabilité des mécanismes proposés, et non pas la performance globale du système. L'approche logicielle nous apporte également plus de souplesse et de facilité de développement·

Les composants du prototype

Le prototype est basé sur du matériel de type PC standard. Il utilise le système d'exploitation GNU/Linux [Sta98] avec un noyau modifié, associé à une version modifiée du démon RSVP développé par l'ISI[5] (*Information Sciences Institute, University of Southern California*) comme indiqué dans le tableau 3.3.

Composant	*Ajouts/Modifications*
RSVP ISI 4.2a3	gestion des labels
	interface avec le noyau
Noyau Linux 2.1.23	souche IPv6 DRET/LIP6/INRIA
	"commutation"
	ordonnancement des paquets
	interface de configuration (appels systèmes)
Outil de configuration	

Tab. 3.3 Les composants de l'implémentation

Le noyau est Linux 2.1.23 avec la souche IPv6 expérimentale de la DRET développée conjointement par le LIP6 et l'INRIA[6] et un gestionnaire de carte ATM "Linux-ATM" développé à l'EPFL (École Polytechnique Fédérale de Lausane) [Alm96]. Les cartes ATM utilisées sont des *Efficient Networks ENI155* pour port PCI, pouvant supporter des débits de 155 Mb/s sur fibre optique multimode. Notons qu'il était tout à fait possible techniquement d'utiliser IPv4, le choix d'IPv6 découle de contraintes contractuelles du projet dans le cadre duquel le prototype a été expérimenté (voir le paragaphe sur l'expérimentation, pages suivantes).

L'implémentation

Au noyau Linux ont été ajoutées les deux fonctionnalités constituant l'émulation de la partie commutateur :

– le moteur de commutation,

[5] `ftp://ftp.isi.edu/rsvp/release/`

[6] Des informations sur cette souche sont disponibles à `http://www-rp.lip6.fr/IPv6`.

– l'ordonnanceur de paquets.

La figure 3.12 décrit sommairement l'organisation du code réseau dans le noyau Linux. Un paquet arrivant sur la carte réseau (1) déclenche une interruption matérielle et la fonction `netif_rx()` place le paquet dans une file de paquets (la file `backlog`) pour traitement ultérieur. Périodiquement, une interruption logicielle (fonction `net_bh()`) vient chercher un paquet dans la file `backlog` et le passe à la couche IP (fonction `ip_rcv()`). Là, le paquet est routé, puis la fonction `do_dev_queue_xmit()` le place en queue de la file associée au périphérique de sortie (`device`) d'où il sera extrait pour être transmis à la carte (2) par la même fonction (qui joue donc deux rôles).

Les ajouts effectués pour assurer les fonctionnalités du RSVP switch sont montrés figure 3.13. Lorsque qu'un paquet est extrait de la file `backlog` par `net_bh()` pour traitement, il est passé à la fonction `rsw_rcv()` qui détermine si le paquet doit être routé (il est alors passé à `ip_rcv()`) ou commuté. Dans ce dernier cas, le périphérique de sortie est déterminé à l'aide de la *table de commutation* et le paquet est passé directement à la fonction `do_dev_queue_xmit()` pour être placé dans une des files de paquets du périphérique de sortie. Les paquets best-effort sont placés dans la file de priorité basse et les paquets GS ou BR dans celle de priorité haute. Dans ce dernier cas la fonction, au lieu de placer le paquet en queue de file, calcule l'estampille du paquet en consultant la table "QoS" contenant les informations d'ordonnancement pour chaque flot et insère le paquet dans la file triée[7].

Une interface à RSVP sous forme d'appels système permet la configuration des tables de commutation et d'ordonnancement.

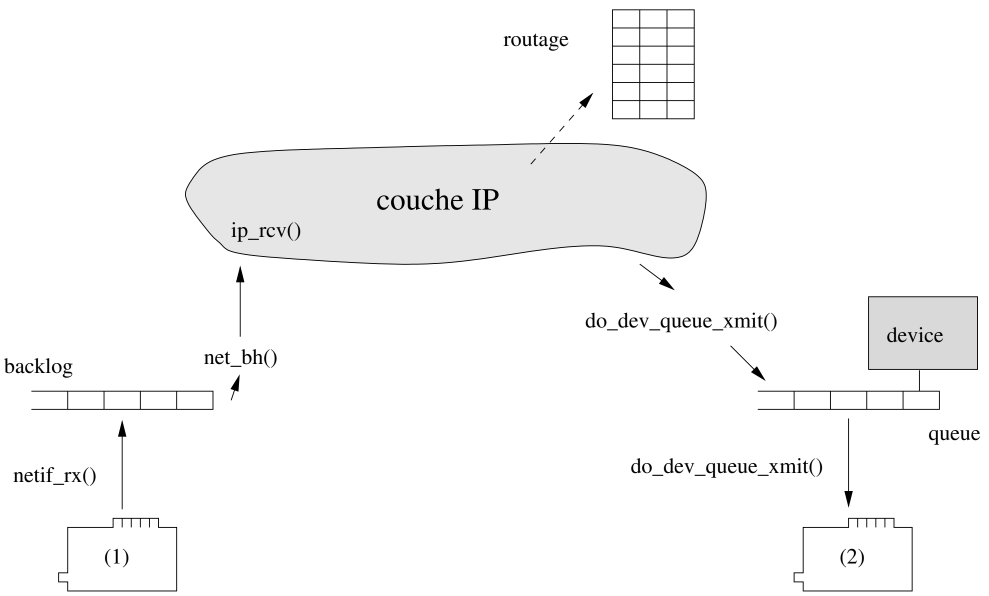


Fig. 3.12 Mécanismes réseau de Linux 2.1

[7]La gestion d'une seule file de paquets triés permet d'obtenir un comportement équivalent à des files séparées.

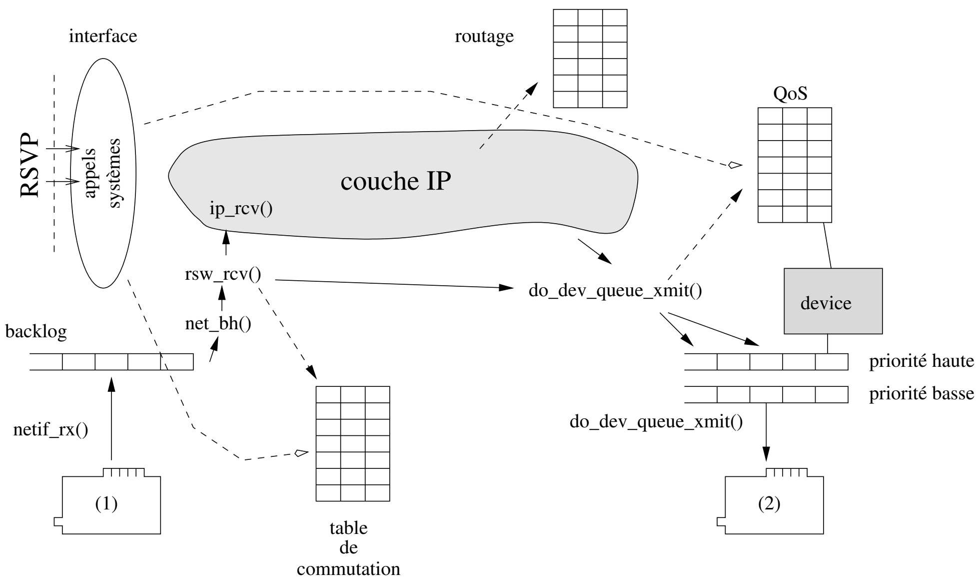


Fig. 3.13 Architecture du RSVP switch émulé

L'implémentation du moteur de commutation est décrite de manière détaillée dans [FRDF98]. Pour l'ordonnanceur de paquets, l'algorithme SCFQ (cité page 37) a été choisi pour la simplicité de calcul de la fonction de temps virtuel.

Les modifications apportées au démon RSVP concernent l'intégration de la gestion des labels (implémentation documentée dans [Die98]), le portage du démon pour la souche IPv6-Linux de la DRET et l'ajout de l'interface avec le noyau (mise en place des raccourcis et réservations).

Enfin, un outil de configuration `rsw_config` a été développé pour permettre la configuration manuelle du RSVP switch (mise en place de raccourcis et de réservations) sans passer par RSVP.

L'expérimentation

Notre prototype de RSVP switch a été utilisé dans l'expérimentation du projet DGA/MENERT appelé DIS/ATM auquel le LIP6 participait avec le LAAS, l'INRIA Sophia Antipolis et Dassault Électronique. Le projet consistait en la fourniture d'une architecture réseau à haut débit avec qualité de service pour des applications de type *simulation interactive distribuée* (DIS pour *Distributed Interactive Simulation*).

Le réseau d'expérimentation a été bâti à partir de la plate-forme Safir en interconnectant les réseaux des trois sites (LIP6, LAAS et INRIA) au moyen de RSVP switches (figure 3.14).

Les résultats de l'expérimentation ont été assez décevants au niveau de la qualité de service. Ceci est dû en partie à des erreurs de mise en place de l'expérimentation et en partie aux limitations de notre implémentation de l'ordonnanceur. L'analyse des résultats de l'expérimentation [CDD+99] a montré que des pertes avaient lieu dans des éléments dont

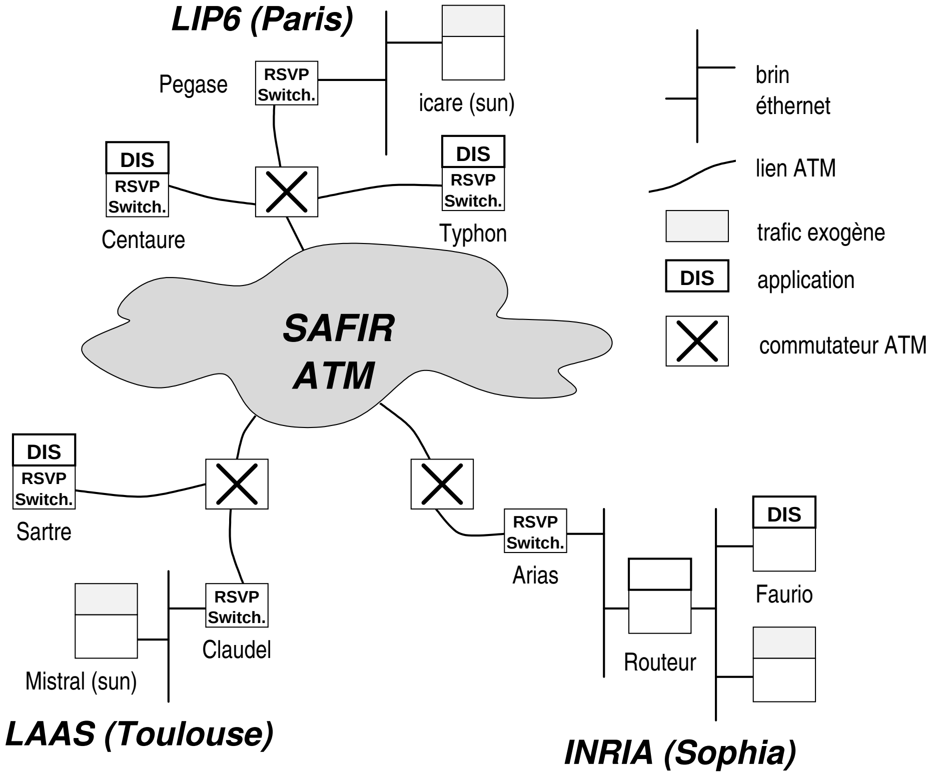


Fig. 3.14 Plateforme d'expérimentation

nous n'avions pas le contrôle (lissage réalisé par les commutateurs ATM de raccordement au réseau SAFIR). Il aurait donc fallu mettre en œuvre un conditionnement de trafic au niveau du RSVP switch de façon à limiter le débit sortant et empêcher l'entrée en action des mécanismes de contrôle de trafic des éléments hors de notre contrôle.

En ce qui concerne les limitations de l'implémentation, nous avions choisi de réduire au minimum les modifications à faire au noyau Linux, qui présente plusieurs défauts dans ce contexte :

- la gestion des fonctionnalités réseau s'effectuant par interruption logicielle (c'est à dire plus ou moins en tâche de fond), il est difficile de prévoir le temps écoulé avant le traitement d'un paquet. De plus, la capacité de traitement dépendra de la charge de la machine pour les autres tâches.
- la file d'entrée *backlog* pose un gros problème car on peut difficilement différencier les paquets à ce niveau. La fonction `netif_rx()` correspondant à une interruption matérielle doit se terminer le plus rapidement possible sous peine de perturber de manière imprévisible l'ensemble du système. De plus, comme il n'y a qu'une seule file pour toutes les interfaces, le trafic suivant des routes différentes est mélangé.

Enfin, le comportement d'un ordonnanceur WFQ se différencie d'un traitement FIFO par définition en situation de congestion (instantanée ou de longue durée), c'est à dire précisément dans une situation de surcharge de la machine où le comportement se dégrade et l'observation est rendue délicate, sinon impossible.

Toutefois, l'expérimentation a permis de valider fonctionnellement le modèle RSVP switching et de montrer la bonne interopération entre zones commutées (utilisant RSVP switching) et zones non commutées (utilisant RSVP sur un réseau classique).

3.4.3 Étude de scalabilité de RSVP

Motivation

Le troisième aspect de l'évaluation de RSVP switching concerne les limites de scalabilité. En effet, si nous avons adapté les mécanismes du modèle proposé par l'IntServ au contexte d'une technologie de réseau haut débit, nous avons conservé le principe d'une gestion de la qualité de service *par flot*. La viabilité de cette approche dans le cadre de réseaux d'interconnexion traversés par un très grand nombre de flots est sujette à discussion. Comme nous l'avons déjà évoqué au chapitre 2, le maintien d'états par flot est susceptible de poser des problèmes aussi bien dans le plan de données (classification et ordonnancement) que dans le plan de contrôle (états RSVP). Cependant dans le cadre d'une solution basée sur la commutation IP les fonctionnalités du chemin de données sont assurées par le matériel ATM. L'opération de classification consiste en une simple lecture du label, et des techniques d'ordonnancement cablées de type WFQ sont d'ores et déjà capables de traiter jusqu'à 64000 flots [Gué98].

Il nous semble donc que c'est le plan de contrôle, c'est à dire RSVP, qui est le plus susceptible de poser un problème de scalabilité pour RSVP switching.

Expérimentation

La plateforme de test utilisée est composée de trois PC standards sous GNU/Linux et est représentée figure 3.15.

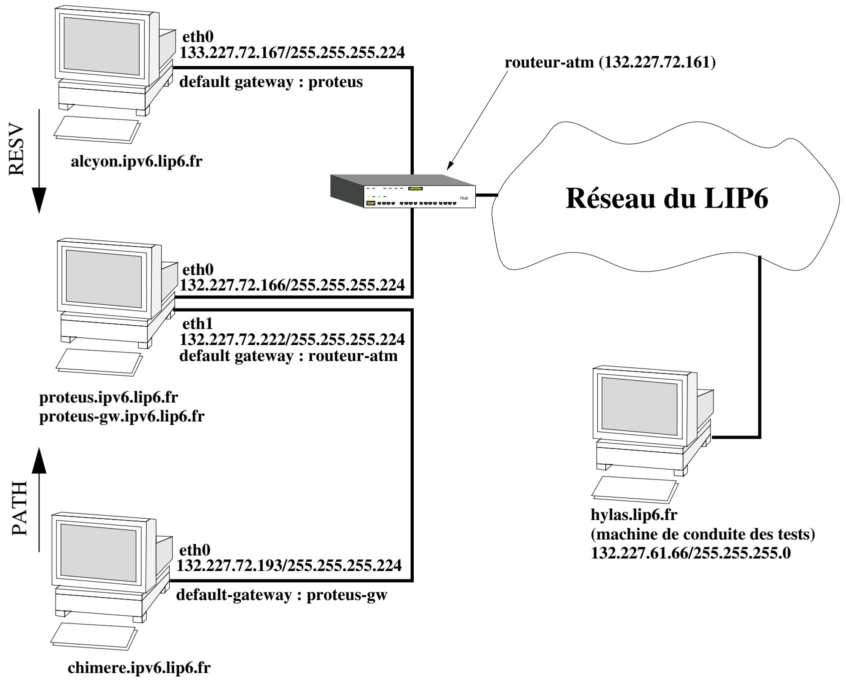


Fig. 3.15 La plateforme utilisée

L'implémentation de RSVP utilisée est celle de l'ISI originale, sans ajout de la gestion des labels. Celle-ci nous a posé de très gros problèmes pour mener cette expérimentation. Les principales difficultés ont concerné le nombre maximal de flots que l'implémentation peut gérer. Il s'est en effet avéré que tel que livré par l'ISI, le démon ne peut supporter plus de 64 sessions. Repousser ces limites a constitué un travail plus que conséquent, car de nombreuses structures de données utilisaient ces limites pour diverses "astuces" de programmation de manière non documentée. Nous avons finalement pu repousser ces limites à une valeur théorique de 524288. Limite extrêmement théorique, car le démon utilisant de manière statique (c'est à dire avant qu'une seule réservation soit effectivement réalisée) de grandes quantités de mémoire dépendant de ces limites, nous avons finalement dû utiliser une limite de 4096 flots pour obtenir un démon utilisable. Les mêmes limitations concernent l'outil de test `rtap` fourni avec RSVP.

Lors des expérimentations, nous avons mesuré la charge du processeur et la mémoire utilisée par le démon RSVP à l'aide d'un outil spécialement développé à partir des sources de `top`, célèbre outil de mesure de la charge pour Linux.

Résultats

Les deux graphes 3.16 et 3.17 montrent l'évolution de la charge en mémoire et en temps processeur (temps de calcul utilisé *depuis de début de l'expérimentation*, exprimé en *jiffies*, unité de temps du noyau égale à 10 ms sur architecture Intel) en fonction du nombre de réservations effectuées.

Dans l'expérimentation de la figure 3.16, les demandes de réservation successives étaient espacées de deux secondes. On voit que l'occupation mémoire est parfaitement linéaire à part une anomalie autour de la réservation numéro 1500. La charge processeur présente quant à elle des paliers assez étonants et son allure globale est assez peu encourageante pour la scalabilité. Nous avons tracé les fonctions en x^2 et en x^3 les plus proches de la courbe expérimentale (déterminées par le logiciel `gnuplot`). L'allure de la courbe de la charge processeur est plus proche du type x^3 (la courbe en x^2 colle moins bien aux extrémités), ce qui indique une charge quadratique en fonction du nombre de réservations (la charge à un instant donné étant la dérivée de la courbe tracée).

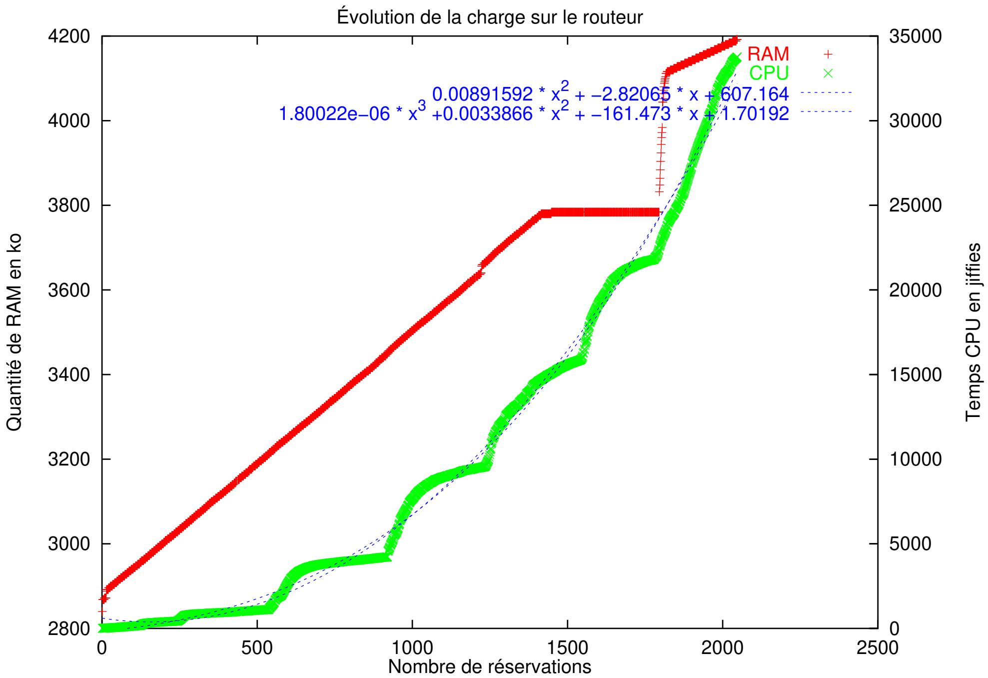


Fig. 3.16 Test 1 – Charge mémoire et processeur

Dans le cas de la figure 3.17, le délai entre les demandes de réservation successives était fixé à une seconde. L'occupation mémoire présente le même type d'anomalie mais de façon beaucoup plus importante et pas au même moment. Elle semble cependant pouvoir à nouveau être majorée par une fonction linéaire. Le phénomène de paliers se retrouve pour la charge du processeur qui n'est à nouveau pas très encourageante. Sa progression semble cependant s'infléchir quelque peu au dessus de 2500 réservations. Nous n'avons pas représenté les courbes en x^2 et x^3, mais nos tests ont montré que la courbe en x^3 est la meilleure si l'on se limite aux 2500 premières réservations, tandis que la courbe en x^2 est plus satisfaisante aux extrémités.

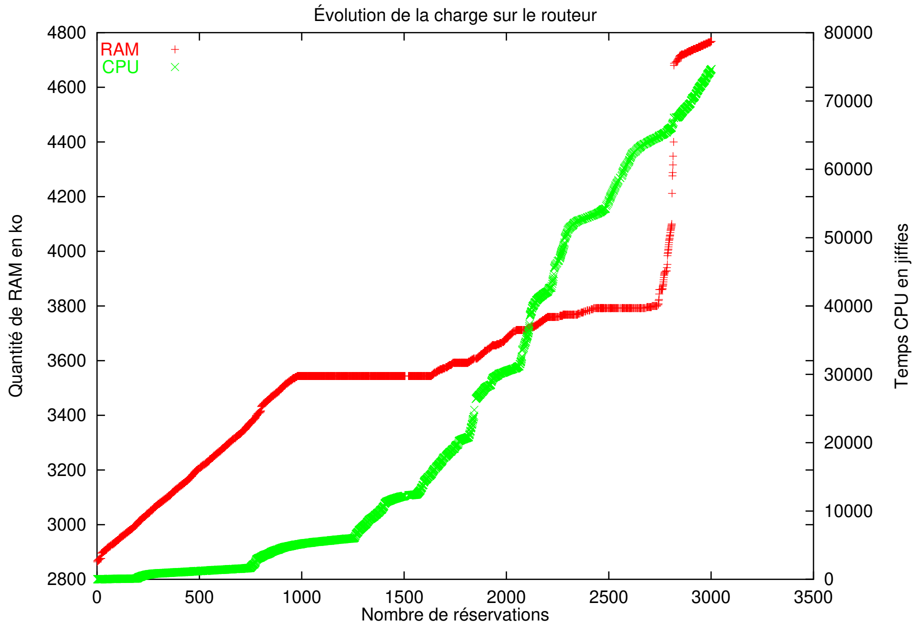


Fig. 3.17 Test 2 – Charge mémoire et processeur

Pour le test 1, l'allure du temps nécessaire pour établir une réservation en fonction du nombre de réservations déjà établies, figure 3.18, conforte les résultats suggérés par la figure 3.16 puisqu'elle est elle aussi quadratique.

Étant données nos réserves sur la qualité de l'implémentation de RSVP utilisée, il est difficile de tirer des conclusions sur le protocole lui même. Il est en effet délicat de distinguer dans nos résultats ce qui est intrinsèque à RSVP de ce qui provient des défauts de l'implémentation.

Il semble cependant raisonnable de considérer que l'occupation mémoire est linéaire par rapport au nombre de flots (le contraire aurait été assez surprenant). Ces expérimentations tendent à montrer que la charge processeur est en x^2, ce qui est nettement plus problématique. Aucun des mécanismes de RSVP ne nous semble pouvoir expliquer ce résultat, aussi est-ce peut être dû à des défauts de l'implémentation. On notera toutefois qu'il s'agit d'une courbe quadratique très peu agressive (le coefficient du terme en x^2 étant trois ordres de grandeur plus faible que celui du terme en x, voir les coefficients de x^3 et x^2 – cette courbe est l'intégrale de la charge instantanée – sur la figure 3.16) ce qui tempère quelque peu l'aspect négatif. Enfin, il faut garder à l'esprit que les tests ne portent que sur un nombre assez réduit de réservations. Pour des conclusions plus définitives, il serait nécessaire d'une part de mener des tests de façon plus poussée, d'autre part de comparer les résultats de plusieurs implémentations.

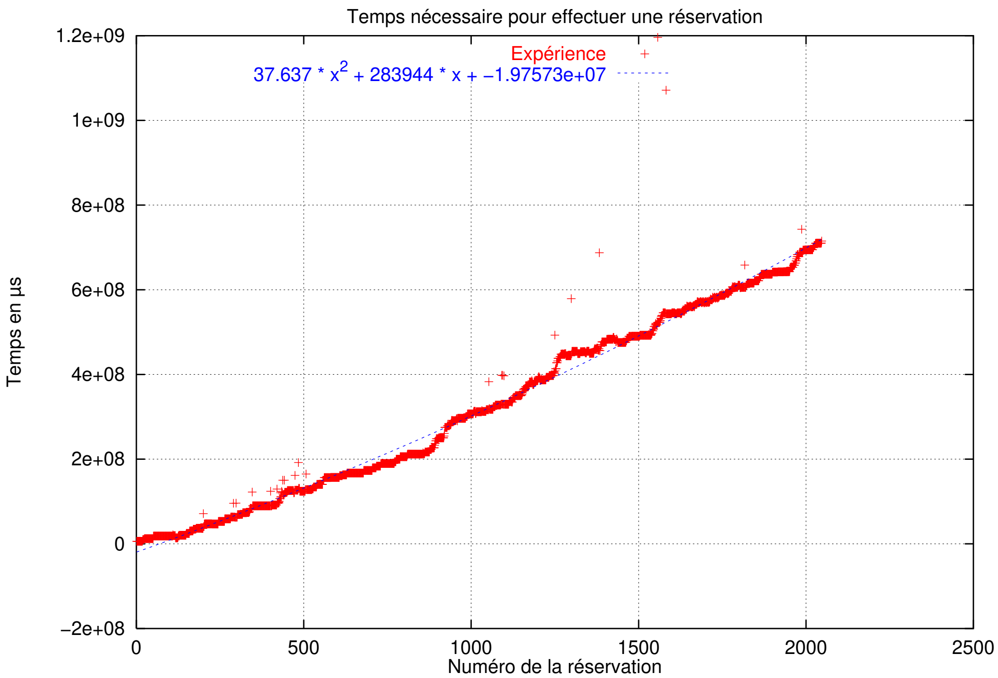


FIG. 3.18 Test 1 – Temps nécessaire pour une réservation

3.5 CONCLUSION DU CHAPITRE

Dans ce chapitre, nous avons proposé une méthode d'adaptation de l'architecture IntServ à une technologie spécifique au haut débit, la commutation IP. Nous avons pour cela intégré les fonctionnalités de mise en place de réservations et de raccourcis pour obtenir un système cohérent.

L'objectif était de conserver les bases de l'architecture IntServ (notamment sa grande flexibilité) tout en mettant à profit les capacités du commutateur en terme de débit et de qualité de service. Les caractéristiques positives d'un ordonnancement au niveau cellule associées avec le service BR permettent d'envisager une utilisation efficace des ressources du réseau.

Il n'est pas possible dans le cadre d'un travail de thèse d'évaluer la performance effective d'un tel système dont la réalisation nécessite de gros investissements. Nous avons cependant montré, au moyen d'un prototype utilisé dans une expérimentation, la viabilité fonctionnelle de la solution.

Dans ce travail, nous avons tenté de résoudre les problèmes liés au haut débit sans nous préoccuper de ceux liés au facteur d'échelle. Dans le plan de données, les limitations dépendront des capacités du commutateur dont on peut espérer qu'elles évolueront rapidement. Au niveau du plan de contrôle, nous pensons, appuyés par les résultats de notre étude, que les problèmes ne sont pas si graves que l'on peut le croire, même s'il est clair que c'est là que se trouvent les limites principales. En particulier, nous n'avons pas pu tirer de conclusion définitive sur l'évolution de la charge processeur en fonction du nombre de réservations, même si nous soupçonnons des problèmes liés à l'implémentation utilisée.

Cette solution est donc plutôt destinée à des environnements haut débit ayant besoin de qualité de service mais avec un nombre (relativement) limité de flots. Dans les chapitres suivants, nous nous intéressons au problème du passage à l'échelle d'une architecture à intégration de service.

Chapitre 4

Une approche de la scalabilité par l'agrégation de flots

4.1 INTRODUCTION DU CHAPITRE

Comme nous l'avons abordé au chapitre 2 et en fin de chapitre 3, l'architecture Int-Serv/RSVP pose un problème de scalabilité qui rend difficile son utilisation dans le cadre de réseaux de grandes dimensions. Il est encore difficile de dire dans quelle mesure et à partir de combien de flots le problème se pose réellement, mais il est clair qu'une gestion des ressources *par flot* dans le contexte de l'Internet dont la dimension ne cesse de croître peut impliquer un très fort *overhead*.

Dans ce chapitre nous étudions une approche basée sur l'agrégation de flots pour résoudre le problème de scalabilité de l'architecture IntServ. Nous proposons de conserver les mêmes mécanismes et méthodes de gestion du trafic en changeant la granularité de la gestion des ressources à l'intérieur des réseaux d'interconnexion.

Nous commençons par définir l'architecture globale de notre proposition, puis montrons comment on peut déduire une garantie de service à des flots agrégés à partir des ressources allouées à l'agrégat. Nous décrivons ensuite les mécanismes et protocoles à mettre en œuvre pour les fonctionnalités dont nous avons besoin. Puis nous étudions les conditions dans lesquelles l'agrégation de flots, en plus de réduire le nombre d'états, permet une meilleure gestion des ressources (bande passante). Enfin, nous discutons les limites de notre proposition et la comparons avec quelques autres propositions de la communauté de recherche avant de conclure.

4.2 ARCHITECTURE

Nous considérons une interconnexion de réseaux mettant en œuvre le modèle IntServ/RSVP. Ces réseaux sont interconnectés par un *backbone* utilisant des techniques d'agrégation pour fournir les services de l'IntServ avec une quantité d'état fortement réduite. Ceci est représenté sur la figure 4.1.

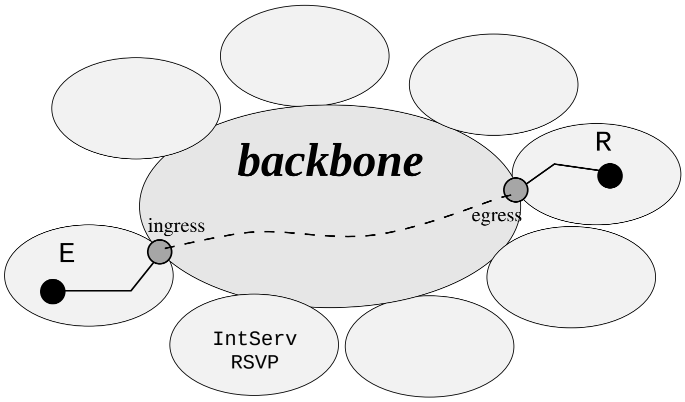


Fig. 4.1 Réseaux IntServ interconnectés par un backbone

Les deux objectifs que nous nous fixons sont :

- garantie de QoS *de bout en bout*,
- fonctionnement totalement *transparent* (une application ne doit pas se préoccuper de savoir si le flot traversera une zone *backbone* ou non).

Le premier point ne peut être atteint que si le modèle mis en œuvre au sein du backbone assure le même type de QoS que l'IntServ. Un moyen de fournir la transparence est de faire apparaître tout le backbone comme étant *un seul élément* IntServ/RSVP.

Nous proposons d'agréger les flots (dans un macro-flot que nous appellerons *b-flot* pour *backbone flow*) par couples de routeur d'entrée dans le backbone (qu'on appellera *ingress*) et de routeur de sortie (*egress*). Ces routeurs de connexion entre les réseaux IntServ et le backbone jouent évidemment un rôle primordial.

Montrons que ce modèle résout le problème de scalabilité de l'architecture IntServ/RSVP. Soit n le nombre de réseaux connectés au backbone. On a alors n^2 combinaisons {ingress, egress}. Sur chaque couple, on devra gérer un nombre fini a de b-flots (un pour chaque classe de service disponible[1]). Remarquons aussi qu'à chaque extrémité on aura au maximum $a \cdot (n-1)$ b-flots en entrée et $a \cdot (n-1)$ b-flots en sortie. Le nombre de flots RSVP n'a aucune incidence sur la scalabilité du modèle, et le nombre de réseaux connectés restera forcément (pour des raisons matérielles) à des valeurs raisonnables.

Ainsi, le nombre de b-flots à gérer sur chaque routeur d'extrémité est *linéaire* par rapport au nombre de réseaux connectés tandis que le nombre total de b-flots à gérer par un routeur

[1]Dans certains cas, plusieurs b-flots peuvent être nécessaires pour une classe de service comme nous le verrons avec le service GS page 75.

à l'intérieur du backbone est *quadratique* dans le pire des cas. Comme le nombre de réseaux connectés n'est pas susceptible d'atteindre de grandes valeurs, ce modèle parait satisfaisant du point de vue de la scalabilité.

4.3 AGRÉGATION

4.3.1 Concept de *b-flot*

Nous nous proposons d'effectuer une agrégation homogène, c'est à dire d'agréger parallèlement les trois états de signalisation, de classification et d'ordonnancement associés à un flot. Ainsi des flots sont regroupés en un seul *b-flot* auquel sont associés ses trois états, qui représentent chacun l'état agrégé des états correspondants des flots agrégés.

Cette agrégation homogène nous conduit donc à un modèle *par flot* semblable à celui de RSVP. À chaque flot sont associés un état de classification qui permet de filtrer les paquets faisant partie de ce flot, un état d'ordonnancement pour fournir la QoS allouée et un état de signalisation pour la gestion du flot.

La différence se situe au niveau de la génération du flot. Dans le modèle IntServ, un flot résulte de l'activité d'un utilisateur. Il est défini de bout en bout, par un couple d'instances d'applications (une instance d'application étant identifiée par une adresse machine et un numéro de port).

Le b-flot n'existe qu'à l'intérieur du *backbone*, ses extrémités sont les routeurs d'entrée (l'*ingress*) et de sortie (l'*egress*) du backbone. Le b-flot est donc défini non pas au niveau de l'application mais au niveau du réseau puisque c'est le réseau qui le génère par agrégation de flots applicatifs. Nous pouvons l'identifier par le couple d'adresses de "ses" routeurs *ingress* et *egress* et le type de service qu'il doit recevoir (on agrégera évidemment des flots ayant des demandes de QoS semblables).

type	*génération*	*granularité*	*étendue*
flot	utilisateur	application	hôte à hôte
b-flot	réseau	route	ingress à egress

Tab. 4.1 flots et b-flots

Ainsi, par rapport au modèle RSVP classique, on modifie la *granularité* ("à qui s'applique la réservation ?") et l'*étendue* ("d'où à où ?") des réservations, ce qui est résumé dans le tableau 4.1. Cette approche de l'utilisation de RSVP sur des agrégations de flots à l'intérieur du backbone est décrite notamment dans [GBH97].

4.3.2 Spécificités

Nous venons de voir que l'on peut se ramener à un modèle similaire à celui qu'utilise RSVP. Cependant, de par leur nature d'agrégats de flots, les b-flots auront des comportements particuliers que nous devrons prendre en compte. L'environnement de type réseau d'interconnexion dans lequel évoluent ces b-flots, possède également des caractéristiques spécifiques. Ainsi les protocoles de routage dynamique sont très rarement utilisés dans un backbone. Les changements de route sont exceptionnels et seront plutôt réalisés par un opérateur ("à la main"). Nous avons relevé les trois points suivants ; les deux premiers ne sont pas à prendre en compte dans le cas d'un routage statique.

CAC après changement de route RSVP utilise le concept d'"état relâché" qui permet de s'adapter automatiquement aux changements de route. Le *path state* est installé sur la nouvelle route par un message PATH qui aura été créé soit suite à l'expiration d'un temporisateur de rafraîchissement (*refresh timeout* – avec donc un délai potentiel lié à la fréquence de rafraîchissement), soit suite à une notification du protocole de routage (*local repair*). Quand ce message PATH arrive au premier nœud commun à l'ancienne et à la nouvelle route, le nœud, constatant que le champ PHOP diffère de celui stocké dans son *path state* envoie immédiatement un rafraîchissement RESV à ce nœud amont. La réservation se crée donc sur la nouvelle route. Il peut bien sur arriver que le contrôle d'admission (CAC pour *call admission control*) refuse alors la réservation ce qui entraînera la fermeture de la session. Ceci est parfaitement acceptable dans le cas classique, où un flot est indivisible, mais pas pour un b-flot qui est divisible en de nombreux flots. Le bon comportement serait de fermer seulement une partie des sessions RSVP de façon à ramener la réservation nécessaire au b-flot à une valeur acceptée par le CAC.

Annulation des réservations Dans le cas d'un changement de route, non seulement des réservations doivent être installées sur la nouvelle route, mais celles de l'ancienne route doivent être annulées.

Comme nous venons de le voir, dès qu'un changement de route est détecté les réservations sont intallées sur la nouvelle portion de route. Par contre les réservations qui étaient établies sur l'ancien fragment de route ne disparaîtront qu'à l'expiration de la temporisation de survie (*cleanup timeout*). Ce temps est directement proportionnel à la période des rafraîchissements dont la valeur conseillée est assez grande (30 secondes) ce qui aboutit à une valeur conseillée de 150 secondes (!) pour le *cleanup timeout*.

Ceci est sans doute acceptable avec la sémantique de flot utilisée par les concepteurs de RSVP, mais pose un réel problème pour des b-flots auxquels seront généralement réservées de grandes quantités de ressources.

Trois façons de traiter ce problème sont possibles :

1. considérer qu'il n'y a jamais (ou presque) de changement de route dans l'environnement considéré (et éventuellement les empêcher) ;
2. garder le mécanisme de temporisation de RSVP mais avec des rafraîchissements très rapprochés (et donc un délai d'expiration court), ce qui a pour inconvénient d'augmenter la charge de traitement ;
3. détecter les changements de route et annuler les réservations.

Ces points sont à prendre en compte lors de la conception du protocole de gestion intra-backbone (section 4.4.3) qui dépendra fortement de la technologie utilisée. Nous ne les discuterons donc pas plus dans notre étude.

Dynamicité Dans le modèle IntServ les caractéristiques d'un flot et la réservation qui lui est affectée peuvent changer à tout moment, mais c'est en pratique rarement le cas. Ce sont donc essentiellement les rafraîchissements qui se chargent de la maintenance des états relâchés.

Un b-flot étant constitué d'un grand nombre de flots, il sera beaucoup plus dynamique. Définissons la *dynamicité* comme étant la fréquence avec laquelle les caractéristiques d'un flot sont modifiées. Si la dynamicité moyenne d'un flot est égale à d, la dynamicité moyenne d'un b-flot composé de n flots est égale à $n \times d$.

Ainsi, si l'on groupe 100 flots de dynamicité $1/100\ Hz$ (flots ayant une période stable de 100 secondes), la dynamicité du b-flot est de $1\ Hz$ (soit une période stable de 1 seconde). Il est clair que dès que l'on agrège une quantité significative de flots, il est nécessaire de réaliser une "accumulation" des demandes de modification (incluant l'ajout ou le retrait d'un flot) avant de les appliquer au b-flot. Une possibilité est de procéder à des modifications par paliers.

4.3.3 Caractérisation du b-flot

Comme décrit en 2.2.3, dans l'architecture IntServ chaque flot est caractérisé par un TSPEC (*traffic specification*) qui comprend essentiellement trois paramètres : b la taille maximum de rafale, r le débit moyen et p le débit crête. Les ressources à réserver dans le réseau pour une qualité de service donnée dépendent de cette caractérisation. On définit le TSPEC (b_b, r_b, p_b) d'un b-flot comme étant la "somme" des TSPEC (b_i, r_i, p_i) des flots i qui le composent. De manière assez évidente, on a $b_b = \sum b_i$, $r_b = \sum r_i$ et $p_b = \sum p_i$. Le service à rendre au b-flot étant connu, le calcul de son TSPEC permet de calculer la réservation à lui affecter.

Dans le cas du *controlled load*, la connaissance des trois paramètres principaux est suffisante pour caractériser un flot. Il n'y a donc aucune difficulté pour le calcul de la réservation à affecter à un b-flot.

Par contre, dans ce cas du service GS tous les paramètres du TSPEC doivent être connus. De plus les fondements théoriques du GS (que nous avons décrits en 2.4) utilisent des outils mathématiques relativement complexes [Cru96, LB96]. Nous montrons maintenant comment caractériser une agrégation de flots devant recevoir ce service.

Agrégation de courbes d'arrivée

Soit un flot F_i caractérisé par le TSPEC (b_i, r_i, p_i) et ayant L_i comme taille maximum de paquet. Sa courbe d'arrivée dans un modèle fluide est alors :

$$A_i(\tau) = min\{p_i.\tau, b_i + r_i.\tau\}$$

Si l'on agrège n flots F_i, $1 \le i \le n$, la courbe d'arrivée du b-flot F_a constitué par l'agrégation est donnée par :

$$A_a(\tau) = \sum_{i=1}^{n} A_i(\tau)$$

Pour chaque flot F_i, on définit la valeur τ_i telle que $p_i\tau_i = b_i + r_i\tau_i$. On a alors :

- si $\tau \le \tau_i \quad A_i(\tau) = p_i\tau$
- si $\tau \ge \tau_i \quad A_i(\tau) = b_i + r_i\tau$

Calculons la valeur de la courbe d'arrivée de F_a :

- si $\tau \le min(\tau_i) \quad A_a(\tau) = \sum(p_i\tau) = \tau \sum p_i$
- si $\tau \ge max(\tau_i) \quad A_a(\tau) = \sum(b_i + r_i\tau) = \sum b_i + \tau \sum r_i$
- si $min(\tau_i) \le \tau \le max(\tau_i) \quad A_a(\tau) = \sum min(p_i\tau, b_i + r_i\tau)$

Remarquons que $A_a()$ est une somme de fonctions continues et croissantes, elle est donc elle même continue et croissante. La figure 4.2 montre qu'entre $min(\tau_i)$ et $max(\tau_i)$ la courbe d'arrivée du flot se trouve quelque part dans le triangle (ABC). Une première solution est de la majorer par les côtés CA et AB ce qui nous donne :

$$A_a(\tau) = min(\tau \sum p_i, \sum b_i + \tau \sum r_i)$$

Il est possible de calculer une courbe d'arrivée plus raide (*tight*) en utilisant la *cascade des TSPEC* comme discuté dans [SKWS98]. On dispose alors d'une caractérisation plus précise dans la zone (ABC) ce qui évite de surestimer les délais et réservations nécessaires. La caractérisation obtenue est cependant plus complexe et ne peut plus s'exprimer dans le cadre d'un simple TSPEC. Par soucis de clarté et de simplicité, nous considérerons dans la suite la première caractérisation même s'il est clair qu'en pratique une implémentation aura tout intérêt à baser ses calculs sur la caractérisation la plus "serrée" possible.

Pour tenir compte du fait que l'on n'est pas dans un modèle fluide, il faut, comme nous l'avons montré en section 2.4 ajouter l'effet du premier paquet non policé. Or ici dans le pire des cas chacun des flots agrégés fournira en même temps un paquet de taille maximum, ce qui constitue un "paquet virtuel" de taille $\sum L_i$ octets.

$$A_a(\tau) = min(\sum L_i + \tau \sum p_i, \sum b_i + \tau \sum r_i)$$

Insistons bien sur le fait que ce "paquet virtuel" n'existe qu'au moment de la création du b-flot, et n'a de sens que pour le calcul de la courbe d'arrivée du b-flot. Ce "paquet" sera immédiatement "lissé" (les paquets qui le composent seront espacés), et la taille maximum de paquet du b-flot est $max(L_i)$.

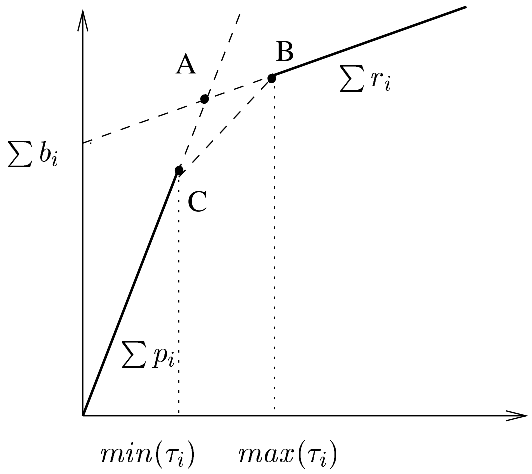


Fig. 4.2 Courbe d'arrivée (fluide) de l'agrégation

Une autre façon de considérer les choses est de caractériser le b-flot par sa courbe d'arrivée *après* que le "paquet virtuel" ait été lissé et de tenir compte du délai ajouté par ce lissage que nous appellerons "délai d'agrégation". La courbe d'arrivée est alors :

$$A_a(\tau) = min(max(L_i) + \sum p_i\tau, \sum b_i + \sum r_i\tau)$$

Et le délai d'agrégation peut se calculer comme $D_2 - D_1$ sur la figure 4.3 ce qui donne :

$$D_1 = \frac{(b_b - \sum_i L_i)}{R_b} \cdot \frac{(p_b - R_b)}{(p_b - r_b)} + \frac{\sum_i L_i}{R} + + \sum_{j=1}^{n} \left(\frac{C_j}{R_b} + D_j \right)$$

$$D_2 = \frac{b_b - L_{max}}{R_b} \cdot \frac{p_b - R_b}{p_b - r_b} + \frac{L_{max}}{R_b} + \sum_{j=1}^{n} \left(\frac{C_j}{R_b} + D_j \right)$$

et

$$D_{agr} = D_1 - D_2 = \frac{b_b - \sum_i L_i}{R_b} \cdot \frac{p_b - R_b}{p_b - r_b} - \frac{b_b - L_{max}}{R} \cdot \frac{p_b - R_b}{p_b - r_b} + \frac{\sum_i L_i}{R_b} + \frac{L_{max}}{R_b}$$

$$D_{agr} = \frac{\sum_i L_i - L_{max}}{R_b} \cdot \frac{R_b - r_b}{p_b - r_b}$$

En conclusion, le *b-flot* peut être caractérisé par les paramètres $(b_b = \sum b_i, r_b = \sum r_i, p_b = \sum p_i)$. La taille maximum de paquet est $L = max(L_i)$. Un "paquet virtuel" de taille $L = \sum L_i$ est à prendre en compte pour le calcul des délais par les courbes d'arrivée et de service.

La formule classique du délai (2.1) s'applique donc sous la forme suivante :

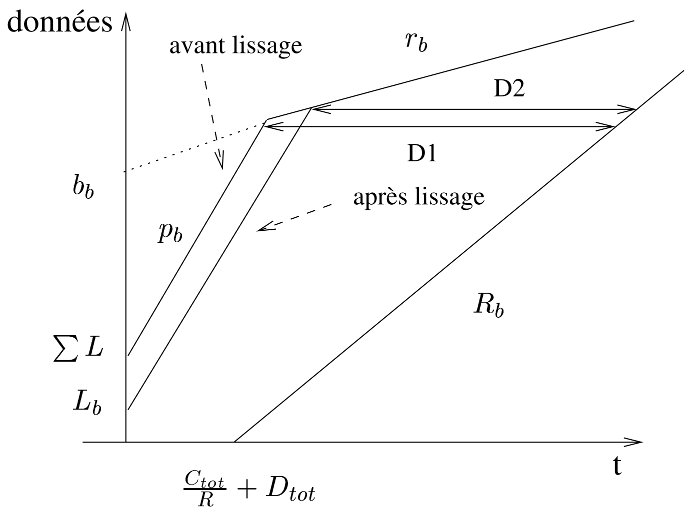


Fig. 4.3 Courbes d'arrivée du b-flot

$$D = \frac{(b_b - \sum_i L_i)}{R_b} \frac{(p_b - R_b)}{(p_b - r_b)} + \frac{\sum_i L}{R_b} + \sum_{j=1}^{n} \left(\frac{C_j}{R_b} + D_j \right) \tag{4.1}$$

La réservation à appliquer pour garantir un délai se calcule classiquement à partir de cette équation.

4.4 MÉCANISMES

4.4.1 Interfaçage

Émulation d'un lien virtuel

Pour assurer un fonctionnement transparent, le backbone doit se comporter globalement comme *un* élément de l'architecture IntServ/RSVP. On peut en fait considérer que le routeur ingress est le nœud RSVP et le chemin dans le backbone le lien (virtuel) associé à ce nœud. Le routeur egress gère quant à lui normalement "son" lien, mais doit de plus coopérer avec l'ingress pour la gestion du lien virtuel. On définit un lien virtuel comme la route entre un couple (ingress, egress). L'*état de contrôle de trafic* (TCS pour *traffic control state*) de l'ingress pour ce b-flot ne sera pas associé à un lien physique mais à un lien virtuel. Sur un même lien virtuel, plusieurs b-flots peuvent fournir des QoS différentes, de la même façon qu'un lien est partagé entre plusieurs flots.

Les différences se situent alors dans les éléments suivants :

lien virtuel Les messages RSVP doivent "sauter" les nœuds internes du backbone. Il suffit pour cela que ces nœuds se contentent de router les paquets RSVP sans leur appliquer aucun traitement particulier (au sens où ils ne doivent pas déclencher d'action autre

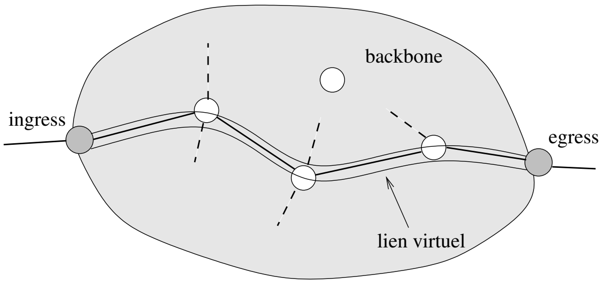


Fig. 4.4 Le chemin dans le backbone est vu comme un “lien virtuel”

que leur simple transmission ; ils peuvent très bien être traités différemment, par exemple prioritairement sur le reste du trafic).

routage Pour déterminer quel est le lien virtuel de sortie, une solution est d’envoyer un message adressé à la destination de la session. Ce message sera intercepté par un routeur egress, qui répondra à l’ingress. Celui-ci connaîtra alors le lien virtuel de sortie. Cette information sera conservée pour éviter de recommencer cette coûteuse opération de “routage sur lien virtuel” à chaque message PATH. On peut par exemple gérer un cache de routes.

contrôle d’admission Le contrôle d’admission vérifie que suffisamment de ressources sont disponibles dans le routeur et dans le lien, ce qui est habituellement une opération locale. Dans notre cas, le test devra être fait à chacun des nœuds du lien virtuel. Ceci est du ressort du protocole de gestion intra-backbone (section 4.4.3).

caractérisation du lien Chaque ingress doit connaître les caractérisations (les termes d’erreur C et D) correspondant à chacun des liens virtuels auxquels il est relié. Ces caractérisations représentent la combinaison des caractérisations des routeurs (ingress inclu, egress exclu) et liens physiques qui constituent *réellement* le lien virtuel. La découverte de cette caractérisation peut être faite par exemple en même temps que le routage sur lien virtuel décrit ci-dessus.

Avec ces modifications (qui ne portent que sur le backbone) RSVP peut être utilisé de manière classique et complètement transparente.

Mises à jour du b-flot

Quand la composition du b-flot change, la réservation qui doit lui être affectée est recalculée, et la nouvelle valeur doit être transmise à tous les nœuds du lien virtuel emprunté par le b-flot.

La nature du protocole à utiliser pour ces mises à jour peut être très variable, et dépend notamment des caractéristiques du backbone (topologie, routage dynamique ou non...). Nous discutons ce point en section 4.4.3.

Nous avons vu en section 4.3.2 que la dynamicité d’un b-flot est beaucoup plus élevée que celle d’un flot classique, ce qui interdit d’appliquer immédiatement toutes les modifications. Elles devront plutôt être “accumulées” et appliquées régulièrement (modèle synchrone) ou bien la réservation du b-flot peut être modifiée par paliers. La façon précise de réaliser cette accumulation dépend largement du type de protocole utilisé à l'intérieur du backbone et de la dynamicité qu'il offre.

Génération du b-flot

À la création à l'ingress d'un *resv state* (par réception d'un message RESV et succès du contrôle d'admission pour la modification résultante du b-flot sur le lien virtuel), l'agrégation du flot dans le b-flot doit devenir effective. Les paquets du flot concerné seront alors aiguillés vers le b-flot correspondant (sous réserve que ce flot respecte son contrat de trafic, voir le paragraphe “Isolation” ci-dessous).

L'effacement d'une réservation (par expiration ou provoquée explicitement) entraîne la modification du TCS, donc de la bande passante allouée au b-flot. À partir de ce moment, les paquets du flot concerné doivent cesser d'être aiguillés vers le b-flot pour être émis en trafic best-effort.

C'est le rôle de ce que nous appellerons le *module d'aiguillage* de générer le b-flot en sélectionnant les paquets des flots appropriés.

Isolation entre les flots

Dans le modèle classique, les réservations sont allouées séparément à chaque flot et ceux-ci sont donc *isolés* afin qu'un flot ne respectant pas son contrat de trafic n'affecte pas le service fourni aux autres flots.

La réservation affectée à un b-flot assure une garantie de service pour ce b-flot. Pour que cette garantie se traduise en des garanties individuelles pour chacun des flots composant le b-flot, il faut pouvoir contrôler l'effet que l'un des flots peut avoir sur les autres. À défaut de pouvoir isoler les flots les uns des autres dans chaque nœud, on peut contrôler le trafic offert par chaque flot au moment de la génération du b-flot, ce que nous appellerons *isolation en amont*. Selon la classe de service concernée, la façon de réaliser cette isolation peut varier légèrement. Nous décrivons ci-dessous les méthodes à utiliser pour les classes GS et CL.

Cas du GS Dans le cas du GS, l'application se doit de respecter son contrat de trafic car tout excès peut rendre complètement caduque la garantie de délai, et même provoquer des pertes de paquets. De plus, pour des raisons liées à la façon dont ce service est implémenté, le réseau peut être amené à lisser le trafic des flots GS conformément à leur TSPEC (*reshaping points* [SPG97, pages 9–12]). On lissera donc les flots GS au moment de l'agrégation.

Cas du CL Le cas de la classe de service *controlled load* est plus délicat. Un flot peut très bien excéder son contrat de trafic, auquel cas le service n'est garanti qu'à la portion conforme au TSPEC, le surplus étant traité comme trafic best-effort[2]. On devrait donc au

[2] La spécification [Wro97a] laisse en fait le choix entre cette solution et une dégradation uniforme du service pour tous les paquets du flot.

moment de l'agrégation policer chacun des flots et diriger le trafic en excès vers le service best-effort.

Cependant, dans la mesure où l'état des ressources du réseau le permet (pas de congestion), il serait préférable de ne pas "diviser" les flots de cette façon car cela peut provoquer des déséquencements inutiles. L'idéal serait donc de ne rejeter les paquets non conformes au TSPEC vers le service best-effort que lorsque cela est réellement nécessaire, c'est à dire quand (et où) le réseau n'est plus capable de rendre le service demandé *également* au trafic en excès. Pour cela, on peut au moment de l'agrégation marquer pour chaque flot les paquets qui dépassent le TSPEC, c'est à dire effectuer un *profilage* du flot (par analogie avec le lissage). Si une congestion apparaît dans un des nœuds du réseau, ils seront alors relégués prioritairement vers le service best-effort (ou jetés si la congestion le rend nécessaire, par exemple avec un mécanisme de type *push-out*) [GBH97]. Il s'agit là d'une forme d'isolation combinée *en amont/locale.*

Le tableau 4.2 résume les façons possibles de fournir l'isolation.

Type	Description
locale	- traitement séparé des flots dans chaque nœud
en amont	- lissage des flots à l'agrégation - traitement agrégé dans chaque nœud
combinée en amont/locale	- profilage des flots à l'agrégation - traitement agrégé avec différentiation IN/OUT éventuelle

Tab. 4.2 Les trois types d'isolations

Architecture d'un routeur *ingress*

La figure 4.5 montre l'architecture d'un routeur *ingress*. Dans le plan de contrôle les états de RSVP *path state* (PS) et *resv state* (RS) sont maintenus pour chaque flot. Par contre les *traffic control states* (TCS) correspondent chacun à un b-flot. Un module (CAC) est chargé de gérer le contrôle d'admission sur les liens virtuels, et un autre de gérer le cache de routes sur liens virtuels.

Dans le plan de données, les paquets sont tout d'abord *routés* normalement de façon à déterminer l'interface de sortie. Ils sont ensuite *classifiés* (par flot) et *lissés* ou *profilés* conformément aux spécifications de trafic contenues dans les états RSVP. À partir de l'information fournie par le routage sur lien virtuel associée à l'information sur la qualité de service désirée, on détermine dans quel b-flot ils doivent être *aiguillés*. Finalement, l'ordonnanceur partage les ressources entre les b-flots en fonction des TCS correspondants.

Quel délai pour les flots GS ?

Si le service fourni par la classe CL est le même pour tous les flots quels que soient leurs caractéristiques, ce n'est pas le cas de la classe GS. Les flots GS demandent tous une

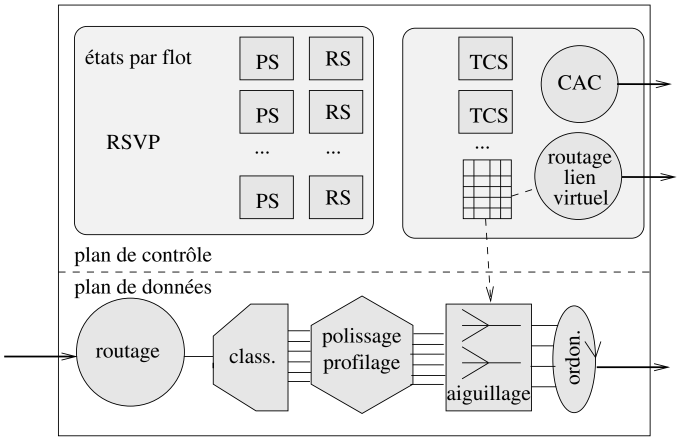


Fig. 4.5 Architecture d'un routeur ingress

garantie de délai qu'ils choisissent individuellement. Il est clair cependant qu'il serait très inefficace d'agréger des flots ayant des demandes de délai différentes, car il faudrait fournir le délai le plus faible de tous les flots à l'ensemble du b-flot, ce qui équivaut à fournir un sur-service à tous les flots qui demandaient un délai moins strict et aboutit donc à une sur-réservation des ressources. Deux approches sont possibles pour résoudre ce problème : le délai à fournir peut être choisi par le réseau ou par l'application.

Choix par le réseau Une première solution consiste à agréger dans un seul b-flot tous les flots GS d'un lien virtuel, en leur fournissant donc le même délai. Pratiquement cela peut-être réalisé en exportant la valeur du délai garanti par le backbone au moyen du paramètre de caractérisation D du routeur ingress (on met alors C à zéro). Ainsi l'application prendra ce délai en compte pour le calcul de sa réservation. Ayant calculé la courbe d'arrivée du b-flot et connaissant les caractéristiques du chemin à l'intérieur du backbone (les *vraies* valeurs $C_{backbone}$ et $D_{backbone}$ caractérisant le lien virtuel, sommes des C et D des nœuds qui le constituent) on peut calculer la réservation à appliquer au b-flot pour garantir le délai fixé. L'intérêt de cette approche est de ne devoir gérer qu'un seul b-flot GS pour chaque couple (ingress, egress).

Choix par l'application Cette solution a l'inconvénient de proposer un service standard identique pour tous. L'autre solution est de laisser l'application choisir le délai. Le routeur ingress (qui gère le lien virtuel) exporte les *vraies* valeurs $C_{backbone}$ et $D_{backbone}$ qui caractérisent le lien virtuel. À la réception de la réservation, connaissant les valeurs de $C_{backbone}$ et $D_{backbone}$ et les caractéristiques du flot, on peut calculer le délai à fournir sur le backbone (nous détaillons comment dans la section suivante). On peut alors agréger les flots demandant des délais proches et créer plusieurs b-flots GS. Évidemment, agréger des flots ayant des demandes de délai hétérogènes implique un surcoût en réservation dépendant des écarts de délais demandés.

Sachant que pour que l'agrégation soit efficace le backbone ne peut proposer qu'un nombre limité de délais, la solution la plus efficace serait de proposer à l'application de choisir entre quelques valeurs de délais susceptibles de satisfaire tous les types d'applications. Cela serait possible à réaliser avec RSVP mais nécessite des modifications de l'API (*application programming interface*), et est donc contraire à notre objectif de transparence pour l'application.

4.4.2 Calcul de délais partiels

Nous avons vu que le critère majeur pour décider quels flots agréger était le délai demandé. Or dans le cadre de l'architecture proposée par l'IntServ, le délai *de bout en bout* est décidé par l'application qui calcule en conséquence la réservation qu'elle va demander au réseau (le réseau lui ayant fourni une caractérisation du chemin suivi).

Pour mettre en œuvre l'agrégation, le réseau a besoin de connaître le délai à fournir à un flot (et non pas sa réservation), de plus c'est le délai sur le tronçon de réseau où l'agrégation a lieu qui est bien évidemment à prendre en compte, et non pas le délai de bout en bout. Nous montrons donc comment le réseau peut calculer le délai à fournir à un flot sur un tronçon de réseau à partir de la réservation demandée par l'application, des caractéristiques du flot et de celles de ce tronçon du réseau.

La formule qui permet de calculer le délai maximum d'un paquet qu'elle soit sous sa forme simplifiée (2.2) ou non (2.1) est constituée de deux parties :

- un terme de temps d'écoulement de la rafale,
- un terme lié à l'approximation du modèle fluide.

Le premier terme est lié aux caractéristiques du flot, alors que le deuxième est la somme des délais ajoutés à chacun des nœuds traversés (qui peut dépendre aussi des caractéristiques du flot — terme C).

Intuitivement on en déduit que le délai maximal sur un tronçon de réseau donné est égal à la somme du délai dû au temps d'écoulement de la rafale (qui peut se produire n'importe où dans le réseau) et des délais dûs aux termes d'erreur des nœuds k à l traversés.

$$D = \frac{(b-L)}{R} \cdot \frac{(p-R)}{(p-r)} + \frac{L}{R} + \sum_{j=k}^{l} \left(\frac{C_j}{R} + D_j \right) \tag{4.2}$$

Cette formule peut être démontrée à partir du calcul avec les courbes d'arrivée et de service comme le montre la figure 4.6. Si le délai est à calculer sur le tronçon qui va du k^{ieme} au l^{ieme} nœud, posons $C_{par} = \sum_{i=k}^{l} C_i$ et $D_{par} = \sum_{i=k}^{l} D_i$, la figure montre que le délai est matérialisé par le segment IK.

Les informations nécessaires à ce calcul sont donc les caractérisations du flot (qui est fournie par l'application) et du tronçon du réseau.

On remarquera que la somme des délais partiels est supérieure au délai total, car le délai dû aux rafales peut se produire n'importe où sur le réseau (mais une seule fois).

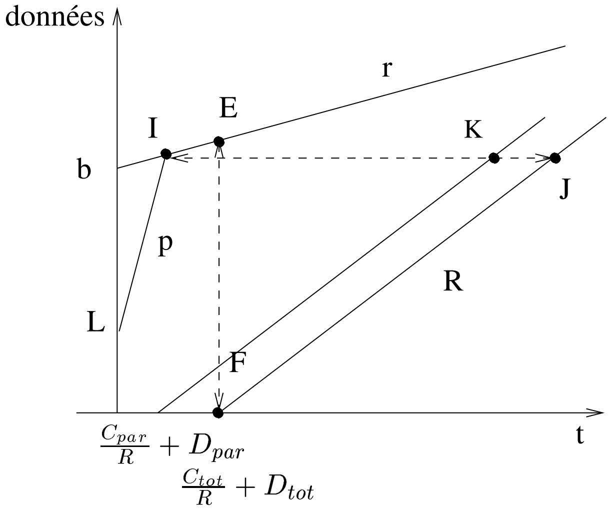


FIG. 4.6 Courbes d'arrivée du flot et de service

4.4.3 Protocole de gestion des b-flots

Le protocole utilisé à l'intérieur du backbone permet la gestion des b-flots. Ses fonctionnalités doivent être :

– mise à jour des réservations (et CAC),
– éventuellement[3] création et suppression des b-flots.

Plusieurs types de protocoles peuvent assurer ces fonctionnalités. On peut utiliser un protocole du type RSVP allégé (adapté conformément à nos remarques de la section 4.3.2) et orienté émetteur. En effet, l'orientation récepteur de RSVP se justifie par son modèle multicast étendu hétérogène. Dans notre contexte, où l'on ne traite que des b-flots unicast et homogènes, l'orientation émetteur paraît plus naturelle.

On peut également utiliser une architecture centralisée, où un serveur (*broker*) gère l'ensemble des b-flots au moyen d'un protocole propriétaire.

En fait, la façon dont le service est réalisé au sein du backbone importe peu, du moment que sont disponibles les fonctionnalités nécessaires à la gestion des b-flots. Différentes solutions sont possibles, dont les extrêmes sont le modèle réparti et à base de soft states "à la" RSVP et le protocole propriétaire centralisé utilisant des connexions dures.

4.5 GAINS DE BANDE PASSANTE DÛS À L'AGRÉGATION

Nous avons vu que l'agrégation entraîne l'ajout d'un *délai d'agrégation* correspondant à la contention des paquets des flots agrégés accédant au b-flot. Le phénomène inverse, c'est à dire une diminution du délai (et donc des ressources réservées pour garantir un délai

[3]Éventuellement car on peut aussi choisir d'avoir des b-flots installés statiquement.

donné) se produit également et est même susceptible d'être largement majoritaire. Des travaux visant à utiliser l'agrégation spécifiquement pour cet aspect (sans recherche de la scalabilité) ont d'ailleurs été menés (par exemple [RG97]).

4.5.1 Cas général

La première source de gain possible est constituée par le terme C reflétant l'erreur d'approximation dépendant du débit du flot d'un serveur émulant le GPS (voir 2.4 page 29). Le délai correspondant à ce terme d'erreur est égal à $\frac{C}{R}$. Or, dans le cas de serveurs WFQ par exemple, la valeur de C est la taille maximum d'un paquet du flot.

Lors de l'agrégation de flots, les effets des rafales s'ajoutent mais sont compensés par l'augmentation de la réservation. Les délais dûs au terme d'erreur D ne bougent pas puisque ce délai est (potentiellement) appliqué à tout paquet indépendamment du flot auquel il appartient.

Enfin les délais dûs au terme d'erreur C diminuent car les valeurs de C ne s'ajoutent pas quand on agrège des flots (le C du b-flot étant le maximum des C des flots agrégés), et les effets sont inversement proportionnels à la réservation qui elle augmente (intuitivement, elle est de l'ordre de grandeur de la somme des réservations qu'il fallait allouer aux flots individuels). Le gain ainsi procuré est d'autant plus grand que la portée de l'agrégation (le nombre de nœuds sur lesquels les flots sont agrégés) est élevé.

4.5.2 Cas des flots à débit constant

Soit un flot F à débit constant r, traversant n nœuds i émulant un canal dédié avec les termes d'erreur C_i et D_i, et ayant besoin d'une garantie de délai D. Nous retrouvons l'équation (2.2) :

$$D = \frac{L}{R} + \sum_{i=1}^{n} \left(\frac{C_i}{R} + D_i \right)$$

Le premier terme correspondant au fait que le flot n'étant pas fluide, et donc pas à débit constant aux petites échelles de temps, il émet des "rafales" de données de taille un paquet, comme illustré figure 4.7.

Ceci se traduit en réservation :

$$R = \frac{L + \sum_{i=1}^{n} C_i}{D - \sum_{i=1}^{n} D_i}$$

Cette équation n'étant bien sur valable que si $r \leq R$. Toutefois, si $r < R$, une partie de la bande passante réservée $(R - r)$ n'est pas utilisée. Dans le chapitre 3 (section 3.3.2), nous avons montré comment utiliser la bande passante réservée en excés pour un flot GS pour fournir des garanties en moyenne. Dans le cas présent, où le flot est à débit constant, cette bande passante peut être utilisée pour fournir les mêmes garanties strictes instantanées qui sont fournies au flot à débit r.

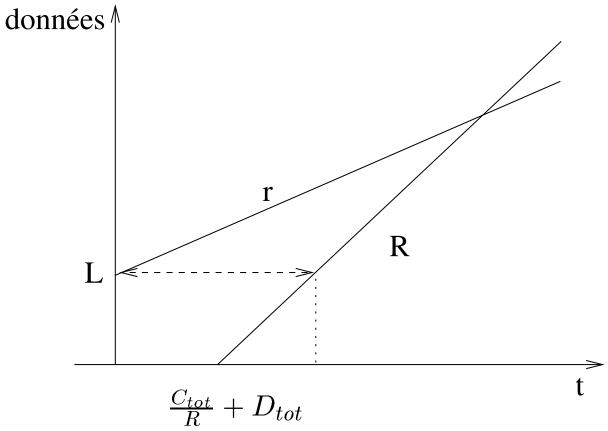


FIG. 4.7 Cas d'un flot à débit constant

En effet, la réservation R peut fournir la garantie de délai D jusqu'à la valeur maximale de $r = R$. L'utilisation optimale des ressources correspond au cas où on a $r = R$. Cependant, la réservation ne dépend pas du débit du flot (sauf dans la mesure où elle doit être supérieure à ce débit) mais uniquement du délai demandé, et l'on peut très bien se retrouver dans des situations où le *taux d'utilisation des ressources* $\frac{R}{r}$ est très faible. Ainsi en agrégeant p flots ayant la même contrainte de délai, on crée un b-flot dont le débit r_b est la somme des débits r_i des flots agrégés. En ajoutant le délai créé à l'agrégation, le délai du b-flot s'exprime comme :

$$D = n \cdot \frac{L}{R_b} + D_{tot} + (p-1) \cdot \frac{L}{R_b}$$

$$D = (n+p-1) \cdot \frac{L}{R_b} + D_{tot}$$

et la réservation nécessaire R_b pour garantir le délai est :

$$R_b = max\left(\sum r_i, \frac{(n+p-1) \cdot L}{D - D_{tot}}\right)$$

Si $R_b = \sum r_i$, la réservation moyenne ramenée par flot est égale à $\frac{R_b}{p} = \frac{\sum r_i}{p} = r_{i,moy}$. L'utilisation du réseau est donc optimale, puisqu'on garantit le délai avec une réservation égale au débit du flot, le taux d'utilisation des ressources est égal à un. Il est à noter qu'on ne peut pas réduire la réservation pour obtenir un délai garanti plus élevé. Si la réservation est inférieure au débit du flot, plus aucune garantie n'est possible, car il n'y aura plus assez de bande passante pour écouler le trafic du flot.

Sinon ($R_b > \sum r_i$) la réservation moyenne ramenée par flot est :

$$R_{b/flot} = \frac{(n+p-1) \cdot L}{p \cdot (D - D_{tot})} = \frac{L(p-1)}{p \cdot (D - D_{tot})} + \frac{n \cdot L}{p \cdot (D - D_{tot})}$$

$$= \frac{(p-1) \cdot R_{fl}}{n \cdot p} + \frac{R_{fl}}{p}$$

$$= R_{fl} \cdot \left(\frac{p-1}{n \cdot p} + \frac{1}{p}\right)$$

$$= R_{fl} \cdot \frac{p-1+n}{n \cdot p}$$

Voyons maintenant dans quels cas cette opération d'agrégation est bénéfique en terme d'utilisation des ressources, c'est à dire que la réservation du b-flot ramenée par flot est inférieure à la réservation du flot seul.

Il faut que :

$$\frac{p-1+n}{p \cdot n} \leq 1$$

ce qui est vrai pour tous $p \geq 1, n \geq 1$.

Ainsi, nous venons de montrer que l'agrégation de flots à débit constant demandant des délais semblables en plus d'être intéressante pour la scalabilité, pouvait conduire à une meilleure utilisation des ressources du réseau. Ce résultat nous semble très intéressant et mériterait un approfondissement des travaux dans cette direction mais cela sort du cadre de notre sujet.

4.5.3 Utilisation

La première méthode, qui utilise le gain sur le terme C_{tot}, est applicable dans n'importe quel contexte, mais le gain n'est pas systématique. Il faudra donc avant de l'appliquer à des flots donnés calculer le résultat de l'agrégation éventuelle. Des règles générales sur les caractéristiques des flots susceptibles de résulter en une agrégation bénéfique doivent pouvoir être dégagées. Par exemple, pour des flots aux caractéristiques identiques demandant le même délai, l'agrégation est toujours bénéfique [RG97].

L'applicabilité de la deuxième méthode peut paraître assez réduite puisque se limitant à des flots à débit constant. En fait, le débit des flots n'a pas réellement à être constant, du moment que leur débit crête p est inférieur à leur réservation R. La bande passante disponible susceptible d'être récupérée pour des garanties dures est alors égale à $R - p$ (au lieu de $R - r$). Les applications qui ont besoin de délais très courts pour des flots à débit crête très faible (par exemple séries de transactions temps réel, certains flots des simulations interactives distribuées) et qui engendrent donc une grande sur-réservation de ressources, pourraient être de bonnes candidates pour cette méthode.

4.6 LIMITATIONS ET COMPARAISONS

4.6.1 Le multicast

Le modèle que nous avons choisi d'utiliser (un b-flot par couple {ingress, egress} et classe de QoS) s'applique difficilement au multicast. En effet le nombre de b-flots multicast potentiels est extrêmement élevé. Ainsi on a n-2 flots 1 vers 2 et 2 vers 1, n-3 flots 1 vers 3 et 3 vers 1 etc...

Ce très grand nombre de b-flots à gérer peut compromettre la scalabilité du modèle en rendant l'agrégation largement inopérante. Le modèle multicast étendu de RSVP, en impliquant des arbres hétérogènes, accentuera encore ce problème.

Cependant, dans le cas où la topologie du backbone est à maillage complet (ce qui est souvent le cas) le problème du multicast se résout très simplement car un flot multicast se réduit à une série de flots unicast que l'on traite indépendamment.

Pour un flot multicast 1 vers n, le message envoyé par l'ingress sera reçu par plusieurs egress. Pour un flot multicast n vers 1, l'egress recevra plusieurs messages et transmettra donc les messages RESV à chacun d'eux. Le modèle étendu n vers m s'obtient par combinaison de ces deux mécanismes, comme le montre la figure 4.8.

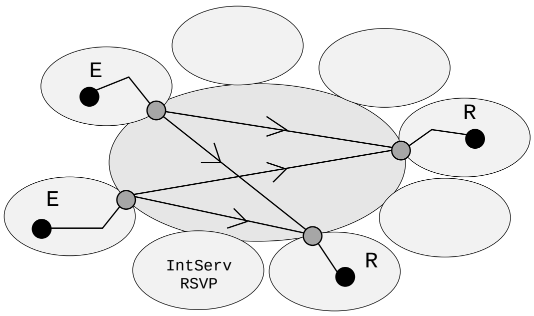


Fig. 4.8 Multicast 2 vers 2

Dans le cas où le backbone n'est pas à maillage complet, la gestion du multicast s'avère plus délicate. Si le nombre de flots multicast est faible (ce qui est le cas dans l'Internet actuel) et ne pose donc pas de problème de scalabilité on peut envisager de les traiter normalement (c'est à dire sans mettre en œuvre de techniques d'agrégation) à l'intérieur du backbone. On aura alors une architecture IntServ classique pour les flots multicast en parallèle à des techniques d'agrégation pour les flots unicast. Cela peut paraître une approche raisonnable à court terme, tant que l'utilisation des applications multicast ne se développe pas sensiblement.

De même, si la majorité des sessions multicast a lieu entre les mêmes réseaux l'agrégation entre ces réseaux peut se révéler efficace, on peut alors installer un b-flot multicast. Le protocole intra backbone devra alors évidemment être capable de gérer le multicast. L'efficacité de cette approche dépend de la distribution (au sens statistique) des flots multicast dans le réseau...

Enfin, on peut choisir de ne pas supporter le multicast, donc de décomposer les flots multicast en séries de flots unicast en dupliquant les paquets au niveau de l'ingress. On préserve ainsi la scalabilité en sacrifiant la bande passante.

Il est clair que notre solution d'*agrégation par flots* n'est pas applicable au cas du multicast. Peut être une approche basée sur l'*agrégation par classe* (à la DiffServ) pourrait elle se révéler plus applicable au multicast, mais cela induirait très probablement un relâchement des garanties. Indépendamment des problèmes de scalabilité liés aux garanties de service, le multicast apporte ses propres problèmes qui ne sont pas encore tous résolus. Il nous semble donc que la fourniture de garanties de service dures de manière scalable à des flots multicast est un problème extrêmement difficile que nous n'avons pas la prétention de résoudre dans le cadre de ce travail de thèse.

4.6.2 Autres approches

Interopération IntServ/DiffServ

Dans [BYF+99], il est proposé une architecture globalement similaire à ce que nous venons de présenter mais en utilisant DiffServ à l'intérieur du *backbone*. Nous avons brièvement discuté le modèle DiffServ en section 2.3. Indépendamment des problèmes qui ne sont pas encore résolus, il nous semble que le modèle de service proposé est inférieur à celui de l'IntServ, soit en fermeté des garanties, soit en flexibilité. Que l'on ait réellement besoin d'un modèle aussi complet et flexible que celui proposé par l'IntServ est bien sur discutable. Cependant, deux options sont possibles :

- soit le modèle DiffServ est suffisant et on a tout intérêt à l'utiliser partout, y compris dans les réseaux de petites dimensions où IntServ est utilisable,
- soit il ne l'est pas et alors l'utiliser dans le *backbone* ne résout pas le problème.

Nous pensons donc que cette approche n'est pas très satisfaisante.

SRP

Une approche originale est *Scalable resource Reservation Protocol* (SRP) proposée dans [FALB98]. L'idée est de s'inspirer du modèle de contrôle de congestion de l'Internet, c'est à dire de mettre le moins possible d'intelligence dans les routeurs en reportant la complexité dans les hôtes (et donc en leur faisant confiance). Sur chaque lien, tous les flots sont agrégés et aucune connaissance des flots individuels n'est conservée dans le réseau. Chaque routeur conserve la réservation totale qu'il a accordée et surveille le trafic. Le protocole utilise trois types de paquets : *best-effort*, *request* et *reserved*. Brièvement, une application désirant obtenir une garantie de débit émet des paquets *request* au débit désiré. Quand un routeur reçoit des paquets de ce type, si il a assez de ressources il augmente la réservation et transmet les paquets, sinon il change les paquets en *best-effort*. Le récepteur mesure le débit arrivant en paquets *request* et en déduit la réservation acceptée sur tout le chemin. Il l'envoie par un protocole de réaction (*feedback*) à l'émetteur qui peut alors émettre des paquets *reserved* au débit accepté. Dans les routeurs, les paquets *request* (non dégradés en *best-effort*) et *reserved* sont traités prioritairement.

Il est possible de gérer le multicast mais le modèle se complique sérieusement si l'on veut permettre l'hétérogénéité des récepteurs. Les principes utilisés font que ce modèle ne peut convenir que pour des garanties relativement lâches, du type de celles requises pour le service *Controlled Load*. Il supposent de plus la coopération honnête des hôtes.

Dynamic Packet State

Enfin, Stoica et Zhang proposent dans [SZ98, SZ99] un modèle adapté aux garanties de délai dures. L'originalité de leur approche est qu'elle n'est pas basée sur l'agrégation, mais consiste à supprimer le besoin de maintenir des états par flot dans chaque nœud en transportant les états dans les paquets, selon la technique qu'ils appellent DPS (*Dynamic Packet State*). Nous reportons la description de ce modèle au chapitre suivant où nous proposons une approche similaire.

4.7 CONCLUSION DU CHAPITRE

Dans ce chapitre, nous avons proposé une solution possible au problème de scalabilité de l'architecture IntServ. Basée sur des techniques d'agrégation, cette approche reproduit le même modèle que l'IntServ à une échelle différente et permet donc de conserver la sémantique des garanties IntServ, ce qui est son atout majeur par rapport aux autres approches proposées (à l'exception de DPS).

Nous avons de plus montré comment pouvaient être mis en œuvre des mécanismes assurant la transparence totale de l'interopération entre les zones avec et sans agrégation, ce qui permet de conserver une interface unique au niveau des applications.

La contrepartie inhérente à l'agrégation est une certaine perte de flexibilité. En particulier notre approche n'est malheureusement pas applicable au multicast en général. Le multicast n'étant cependant pas lui-même utilisable à grande échelle, cela ne nous paraît pas un inconvénient majeur, au moins pour les quelques années à venir. Il nous semble que fournir à la fois la qualité de service de manière scalable *et* le multicast est un problème extrêmement difficile, et qu'il faudra probablement que les problèmes propres au multicast aient eux-mêmes été résolus pour pouvoir y apporter des solutions.

Notre objectif dans l'étude des techniques d'agrégation était la réduction du nombre d'états pour fournir un modèle scalable. Il se trouve que l'agrégation peut apporter d'autres bénéfices en terme d'efficacité d'utilisation des ressources, ce que nous avons montré en section 4.5. Ceci, combiné au service BR proposé dans le chapitre 3, devrait contribuer à permettre d'atteindre une utilisation optimale des ressources dans un réseau à intégration de services.

Chapitre 5

Garanties de délai sans état par flot

5.1 INTRODUCTION DU CHAPITRE

Dans le chapitre précédent, nous avons utilisé une approche basée sur un changement d'échelle du modèle classique de gestion par flot de la qualité de service. Comme toutes les solutions basées sur le principe de l'agrégation, elle entraîne intrinsèquement une perte de flexibilité puisqu'on ne peut plus faire de gestion *par flot*. L'exemple le plus évident de ce manque de flexibilité est l'impossibilité de traiter le multicast. De même, dans le cas des flots GS nous étions obligés de fournir le même délai à tous ou de gérer un b-flot pour chaque délai.

L'approche que nous nous proposons d'étudier dans ce chapitre est similaire dans ses grandes lignes à celle présentée dans le chapitre 4 : des réseaux mettant en œuvre le modèle IntServ/RSVP sont interconnectés par un *backbone* utilisant un modèle différent mais capable de fournir le même type de garanties.

Nous proposons une méthode permettant de résoudre les problèmes de scalabilité sans pour autant diminuer la flexibilité. Le but est de conserver une granularité minimale dans la gestion des flots, mais en réduisant la nécessité de maintenir des états pour chaque flot à l'intérieur du backbone qui devient une "zone sans état".

Nous commençons par situer notre travail dans le cadre des garanties de délai dures, et décrivons les différences entre les disciplines de service *weighted fair queuing* et *virtual clock*. Puis nous décrivons rapidement une proposition pour le plan de contrôle, c'est à dire une méthode pour faire un contrôle d'admission par flot sans maintenir d'état pour chaque flot. Nous discutons ensuite le plan de données et montrons que le trafic étant contraint en entrée, on peut supprimer les états par flot et prévoir les effets de cette suppression sur les garanties. Cependant, les outils analytiques ne pouvant nous donner des bornes sur ces effets que dans des cas *pires*, nous utilisons des simulations pour obtenir des estimations plus réalistes.

5.2 PRÉLIMINAIRES

5.2.1 Gestion des garanties dures

Plusieurs types de services doivent être proposés dans un réseau à intégration de services. Certaines applications ont besoin de garanties dures alors que d'autres peuvent se contenter de garanties statistiques. Un autre "service" hautement désirable pour le trafic *best-effort* est le partage équitable de la bande passante. Tous ces services doivent pouvoir être proposés dans l'ensemble du réseau.

Nous plaçons notre travail dans le contexte de garanties de délai "dures", par exemple le service GS de l'IntServ. Nous pensons en effet que c'est le cas le plus difficile, et celui pour lequel on a le plus besoin de flexibilité. Pour des services du type *controlled load*, des solutions comme SRP peuvent être suffisantes. Pour l'aspect partage équitable de bande passante, des mécanismes du type RED (*Random Early Detection*) nous semblent bien plus intéressants car ils peuvent fournir une équité globale par opposition aux techniques d'ordonnancement qui ne peuvent fournir qu'une équité locale à chaque nœud.

5.2.2 *Virtual Clock* et *Fair Queuing*

Nous avons déjà dit au chapitre 2 que *Virtual Clock* et *Weighted Fair Queuing* avaient des caractéristiques communes du point de vue de l'implémentation, et fournissaient les mêmes garanties de délai.

Le *Weighted Fair Queuing* émule la discipline de service idéale GPS. Le *Virtual Clock* (VC) a lui éte pensé comme émulation du TDM (*Time Division Multiplexing*), une discipline bien réelle mais utilisable uniquement dans les réseaux à commutation de circuit. Dans [FP95], Figueira et Pasquale proposent une autre interprétation de la discipline VC et montrent les propriétés de délais identiques à celles de PGPS (démonstration d'ailleurs très similaire à celle faite dans [PG93] pour PGPS). Leur interprétation consiste à considérer que VC émule une discipline dans laquelle chacun des flots est servi par son propre serveur FCFS à un débit fixé. Par convention, nous appellerons TDM cette discipline.

Un serveur GPS partage la bande passante équitablement entre tous les flots de manière dynamique, en fonction de leurs *poids* ϕ_i. Pour obtenir une garantie de débit pour un flot, il faut limiter le nombre de flots. Chaque flot i reçoit alors un débit garanti égal à $(\phi_i / \sum \phi_j) \cdot C$, C étant la capacité du serveur. C'est à partir de cette garantie de débit que l'on déduit des garanties de délais pour les flots dont le trafic est contraint par un *seau percé*.

Dans la discipline TDM, chaque flot se voit également garanti une bande passante égale à $(\phi_i / \sum \phi_j) \cdot C$ et les mêmes garanties de délai s'appliquent dans les mêmes conditions. Dans le cas du WFQ cependant, le débit effectivement alloué à chaque flot est au minimum égal à $(\phi_i / \sum \phi_j) \cdot C$. Dès qu'un ou plusieurs flots ne sont pas actifs, la bande passante qui leur est allouée est immédiatement partagée équitablement entre les autres flots, qui recoivent alors un débit plus élevé et, en conséquence, des délais plus faibles.

Dans le cas du VC, lorsque certains flots sont inactifs le débit en surplus est alloué aux autres flots (la discipline conserve le travail – *work conserving*) mais de manière arbitraire ; il n'est donc pas possible de garantir un accroissement du débit disponible pour un flot donné.

Néanmoins, comme les délais maxima sont basés sur le cas où le flot dispose du minimum garanti de bande passante, les bornes de délais sont les mêmes pour les deux disciplines. En fait, dans le cas où l'on n'est intéressé que par les garanties de délai, VC se révèle un meilleur choix car le calcul réalisé pour l'estampillage des paquets est plus simple, et surtout, il ne dépend pas de l'état des autres flots.

Dans la suite, nous appellerons donc TDM la discipline de service fluide équivalente au GPS quand tous les flots sont actifs. En ce sens, *virtual clock* est une émulation de TDM de la même façon que *weighted fair queuing* est une émulation de GPS.

5.3 PLAN DE CONTRÔLE

Le problème du plan de contrôle se ramène à la question de comment réaliser un contrôle d'admission sans maintenir d'état par flot. Nous ne détaillerons pas le problème mais décrirons seulement les grands principes d'une solution acceptable.

5.3.1 CAC local

On conserve dans chaque nœud la somme R_{som} des réservations allouées pour tous les flots. Localement, le contrôle d'admission pour un flot i demandant une réservation R_i teste si $R_{som} + R_i \leq C$ (C étant la capacité du lien). Si le test réussit, R_{som} est mis à jour en $R_{som} + R_i$. Sinon, la réservation est refusée. Quand un flot se termine, sa réservation doit être soustraite du total $R_{som} -= R_i$.

5.3.2 CAC global

De la même façon que ce que nous avons proposée au chapitre 4, l'ensemble de la zone sans état doit être perçu comme un seul élément par les réseaux périphériques. L'interfaçage au niveau du contrôle d'admission sera donc similaire. La réalisation du contrôle d'admission dans la zone sans état sera par contre très differénte. Deux approches sont possibles, utiliser une architecture répartie ou un gestionnaire de contrôle d'admission centralisé (*broker*). Dans les deux cas, il est impératif d'assurer qu'aucun changement de route ne se produise.

Architecture répartie

Si on utilise une architecture répartie, chaque nœud gérant sa valeur de R_{som} il est alors primordial d'assurer la fiabilité de transmission des messages, aussi bien d'ajout (message ADD) que d'effacement (message DEL) de réservation. Il faut également gérer la "remontée d'un échec", c'est à dire annuler les réservations faites en amont du nœud où le CAC échoue. Pour cela, le message ADD peut accumuler la liste des nœuds parcourus. Un message CANCEL pourrait alors remonter jusqu'à l'ingress en annulant la réservation. Ceci est décrit plus formellement dans le tableau 5.1. Les notations CAR, CDR et CONS représentent les fonctions classiques de manipulation de liste du langage LISP.

Message reçu	**Traitement**	
ADD(R_i, liste)	si $R_{som} + R_i \leq C$	$R_{som} += R_i$
	envoie	ADD(R_i,CONS(nœud,liste)) $\rightarrow$ suivant
	sinon envoie	CANCEL(R_i,CDR(liste)) $\rightarrow$ CAR(liste)
CANCEL(R_i,liste)		$R_{som} -= R_i$
	envoie	CANCEL(R_i,CDR(liste)) $\rightarrow$ CAR(liste)
DEL(R_i)		$R_{som} -= R_i$
	envoie	DEL(R_i) $\rightarrow$ suivant

TAB. 5.1 Traitement des messages pour le contrôle d'admission

Comme nous l'avons dit, il faut assurer une fiabilité *absolue* de la transmision de ces messages, sans quoi les valeurs R_{som} stockées dans les nœuds finiraient par diverger. Même si des procédures de contrôle et retransmission des messages sont mises en place, on ne peut pas éviter des pertes de paquets dans des cas extrêmes, par exemple la chute d'un lien. Il est donc impératif de disposer d'un moyen de rétablir les valeurs des R_{som} de manière cohérente. Nous proposons de permettre aux nœuds de mesurer la bande passante *réservée* (et non pas utilisée) par observation du trafic, selon une idée proposée dans [SZ99].

Pour cela, un bit est utilisé dans l'en-tête des paquets. Au niveau des ingresses, ce bit est mis à 1 pour un paquet de chaque flot sur un intervalle de temps fixé T. Sur l'intervalle de temps, chaque nœud ajoute à son compteur R_{mes} la valeur R_i stockée dans tout paquet marqué (ayant le bit à 1) qu'il traite. À la fin de l'intervalle de temps, R_{mes} vaudra la somme des réservations faites dans ce nœud divisée par la longueur de l'intervalle[1]. Pour éviter que les variations de délai fassent qu'un paquet soit compté dans le mauvais intervalle, on peut synchroniser tous les nœuds (en utilisant par exemple NTP [Mil92]) et marquer les paquets à peu prés au milieu de l'intervalle.

[1] Sous réserve que tous les flots aient émis au moins un paquet. Cela sera généralement le cas si l'intervalle de temps choisi est suffisament grand. Les routeurs de bordure (voir la section 5.4.6) peuvent éventuellement émettre un paquet pour les flots silencieux.

Architecture centralisée

Une solution centralisée consiste à utiliser un *broker* connaissant la topologie du réseau et gérant la valeur R_{som} pour chaque nœud. Le contrôle d'admission n'étant plus réparti, les problèmes de fiabilité de transmission ne se posent plus. La structure de données à maintenir ne pose pas à priori de problème de scalabilité, sa taille ne dépendant que de la taille de la zone sans état. La limite sera plutôt liée à la puissance de calcul nécessaire pour traiter tous les contrôles d'admission, qui dépendra de la dynamicité du trafic.

Le multicast

Le problème du multicast est particulier (et difficile) car la signalisation et les états liés au routage multicast peuvent eux mêmes poser des problèmes de scalabilité. La gestion de la qualité de service pour le multicast dépendra fortement, et pourrait éventuellement être associée avec, la signalisation liée au routage multicast. Cela sort du cadre de notre travail et nous ne discuterons donc pas ce problème.

5.4 PLAN DE DONNÉES

5.4.1 L'implémentation du WFQ/VC

Comme nous l'avons vu en 2.5 l'implémentation d'un ordonnanceur du type *weighted fair queuing* ou *virtual clock* consiste typiquement à maintenir une file de paquets triés selon une estampille calculée par la formule suivante (F_i^k est l'estampille de P_i^k, le $k^{ième}$ paquet du flot i) :

$$F_i^k = max(A_i^k, F_i^{k-1}) + \frac{L_i^k}{R_i} \tag{5.1}$$

où A_i^k est la date d'arrivée du paquet (exprimée en temps virtuel pour le WFQ et temps réel pour le VC), L_i^k sa taille et R_i la réservation allouée au flot i.

De cette formule il apparaît que pour chaque flot on a besoin de connaître les valeurs de F_i^{k-1} et R_i. L'état à maintenir par flot dans chaque nœud se réduit donc au couple (F_i^{k-1}, R_i). Nous ne prenons pas en compte la gestion de tampons mémoire réservés au flot pour assurer un taux de perte nul. Ceci sera justifié en 5.4.3.

Il est facile de se débarrasser de l'état R_i : celui-ci étant constant dans l'espace (c'est à dire qu'il a la même valeur sur chacun des nœuds du chemin), on peut le transporter simplement dans les paquets, dans un champ d'en-tête réservé à cet effet. Si à un moment donné la réservation associée à un flot change, il suffit de le refléter dans la valeur de R_i codée dans les prochains paquets du flot (sous réserve que le contrôle d'admission ait précédement accepté le changement de réservation).

Éliminer l'état F_i^{k-1} est plus difficile ; d'une part il a une valeur purement locale (il représente une date exprimée dans le référentiel temporel de l'interface considérée), d'autre part il est hautement dynamique : par définition, sa valeur est mise à jour pour chaque paquet du flot qui traverse le nœud. Afin de voir de quelle façon et dans quelle mesure on peut éventuellement éliminer cet état, nous allons maintenant examiner en détail sa raison d'être.

5.4.2 Le rôle du terme F_i^{k-1}

Comme nous l'avons dit en 2.5, le besoin du terme F_i^{k-1} tient au fait qu'en GPS le paquet P_i^k ne peut pas commencer son service avant que le paquet P_i^{k-1} n'ait terminé le sien, et permet donc éventuellement d'augmenter l'estampille du paquet à la valeur qu'elle aurait avec le serveur GPS.

Intuitivement, on peut dire que cela correspond à retarder[2] un paquet qui arriverait “en avance”. Nous distinguons trois causes pour lesquelles un paquet est susceptible d'arriver en avance :

1. la source émet à un débit supérieur à sa réservation ;
2. le paquet est plus court que le paquet précédent ;
3. le nœud précédent a créé de la gigue.

Le premier point est évident. Le deuxième correspond au fait que le délai de paquetisation étant proportionnel à la taille du paquet, un paquet plus court subira des (aura des garanties de) délais plus faibles et aura donc tendance à rattraper un paquet plus long le précédant. Pour le troisième point, il est nécessaire d'examiner d'un peu plus près que nous ne l'avons fait jusqu'à présent le fonctionnement de l'algorithme VC.

Annulation de la gigue

Celui-ci garantit qu'un paquet P_i^k ne finira pas sa transmission avec un retard plus grand que $\frac{L_{max}}{C}$ par rapport au service fourni par le TDM. Il peut cependant très bien le finir beaucoup plus tôt. Sur la figure 5.1 on voit que la transmission du paquet P_i^k dure $\frac{L_i^k}{R_i}$ en TDM et $\frac{L_i^k}{C}$ en VC, où sa transmission peut avoir lieu (se terminer[3]) n'importe où dans l'intervalle $[A_i^k + \frac{L_i^k}{C}, A_i^k + \frac{L_i^k}{R_i} + \frac{L_{max}}{C}]$. Il y a donc une incertitude égale à $\frac{L_i^k}{R_i} + \frac{L_{max}}{C} - \frac{L_i^k}{C}$ sur le moment de transmission du paquet. Pour simplifier, nous considérerons dans la suite que l'instant de transmission du paquet est situé dans l'intervalle $[0, \frac{L_i^k}{R_i}]$, A_i^k étant égal à 0.

Ceci implique qu'en sortie d'un nœud implémentant la politique VC, l'espacement entre les paquets ne sera pas nécessairement (et très probablement pas) constant et égal à $\frac{L}{R}$ même si le flot était parfaitement lissé à l'entrée dans le nœud. Au contraire, les paquets auront tendance à “s'agglomérer” par paires pour former ce que nous appellerons des “grumeaux” de paquets (paquets 1 et 2 sur la figure 5.1).

[2]Le retard n'est pas nécessairement effectif puisqu'il s'agit d'une estampille destinée à arbitrer la contention.

[3]Nous considérons qu'un paquet a été transmis/reçu quand son *dernier bit* a été transmis/reçu.

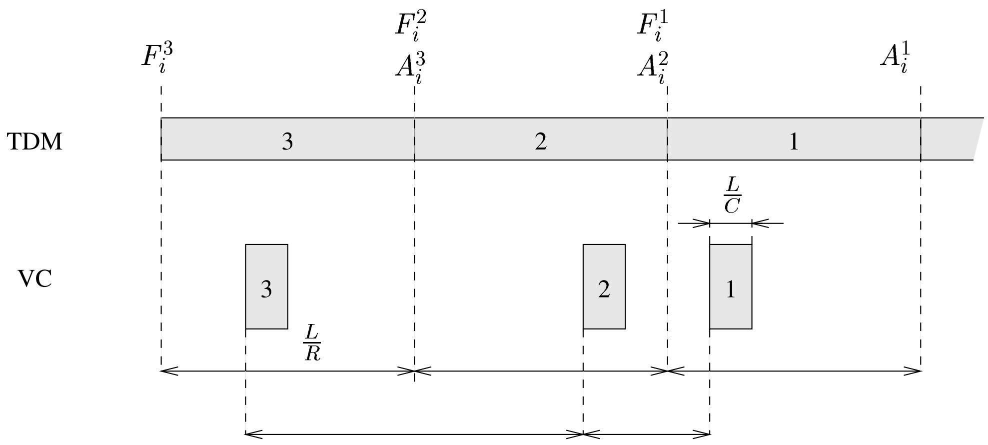


Fig. 5.1 Transmission des paquets en TDM et VC

Dans le pire des cas, le paquet P_i^{k-1} aura subi un délai maximum égal à $\frac{L_i}{R_i}$ et le paquet P_i^k un délai nul. Les deux paquets se retrouvent alors "collés". Notons que la simplification que nous avons opérée sur l'intervalle d'émission possible du paquet se justifie dans la mesure où un intervalle plus grand ne peut aggraver encore l'effet d'agglomération (l'ordre des paquets ne peut pas être modifié).

5.4.3 Élimination du terme F_i^{k-1}

À partir de notre analyse du rôle joué par le terme F_i^{k-1} dans l'algorithme du VC nous pouvons maintenant envisager comment limiter les conséquences de sa suppression. Nous avons précédement relevé trois aspects de ce rôle.

Débit trop élevé

Le premier consiste à contraindre au débit réservé un flot qui émettrait à un débit plus élevé. Ce rôle peut être assuré par un lisseur de trafic à l'entrée de la région sans état. Ce lissage systématique n'a pas d'influence sur le délai maximal des paquets, car dans le modèle classique le lissage peut de toute façon potentiellement être réalisé dans n'importe lequel des nœuds (au premier nœud congestionné, le flot ne pourra pas utiliser plus que la bande passante réservée et sera donc lissé au débit correspondant). Évidemment, il est probable que le délai moyen soit augmenté mais cela ne nous semble pas être un problème. Le corollaire est d'ailleurs une diminution de la gigue au récepteur.

De plus, le flot étant lissé à l'entrée de la zone sans état, aucun tampon autre que ceux nécessaires à l'arbitrage de la contention (donc de taille réduite) n'est requis à l'intérieur de la zone. Celle-ci étant typiquement un réseau à haut débit devant utiliser des mémoires très rapides et donc coûteuses, cela permettra des économies substantielles. Cette économie se reflète également en gestion puiqu'il n'est plus nécessaire de réserver et gérer l'occupation de tampons *par flot*. Il suffira au contraire de les dimensionner suffisamment pour ne pas risquer de pertes dues à la contention.

Changement de la taille de paquet

Il s'agit à notre avis d'un problème secondaire. On peut simplement se ramener au cas d'une taille de paquet constante en "bourrant" les paquets trop courts au niveau de l'ingress, de façon à n'avoir que des paquets de taille L_i constante pour chaque flot i dans la zone sans état. Il faut alors faire l'opération inverse à la sortie. On peut néanmoins agir plus subtilement, c'est à dire faire simplement en sorte que les paquets soit traités comme s'ils avaient une taille L_i fixée. Par exemple, au lieu de transporter la valeur de R_i dans le paquet, on peut transporter directement L_i/R_i (la valeur de R_i n'étant utilisée dans le calcul de l'estampille que comme diviseur de L_i).

Il faut cependant bien noter que dans ce cas une application ne peut pas supposer qu'elle obtiendra des délais plus faibles si à un moment donné elle réduit la taille de ses paquets.

Correction de la gigue

Une solution déterministe a été apportée à cet aspect dans [SZ99]. Brièvement, l'idée est d'utiliser une variante de *Virtual Clock* qui contrôle la gigue (c'est une discipline qui ne conserve pas le travail). Au moment où un paquet est émis, la différence entre les instants de départ effectif et théorique est placée dans un champ de l'en-tête du paquet. Au nœud suivant, cette valeur sera utilisée pour retarder le paquet. Ainsi la gigue ne peut s'ajouter de nœud en nœud, elle est limitée à L/R.

Nous discutons plus en détail cette solution pour la comparer avec notre proposition en section 5.6 page 108. Plutôt que d'assurer un contrôle strict de la gigue, nous prenons comme optique de prévoir et prendre en compte les effets de la gigue sur les garanties.

5.4.4 Les effets de la gigue

Si la gigue créée par le nœud précédent n'est pas corrigée par le terme F_i^{k-1}, l'estampille du paquet P_i^k arrivé "en avance" va être plus petite que ce qu'elle devrait être pour émuler correctement le TDM. Cela se traduira éventuellement par un gain de placement pour la résolution de la contention au détriment des paquets d'autres flots en contention. L'ordonnanceur n'émule plus correctement le modèle de référence puisqu'il alloue à cet instant au flot i plus de bande passante que ce qui lui est dû. Il émule alors un serveur fluide qui servirait deux paquets du flot i simultanément. Par exemple, sur la figure 5.2 le paquet 2 arrive en avance, un ordonnanceur VC classique repousse son estampille pour émuler correctement le TDM alors qu'un ordonnanceur qui ne connaît pas la valeur de F_i^1 (appelons le *stateless VC*) commet une erreur sur le calcul de F_i^2. Cette sur-consommation des ressources par le flot n'a pas d'effet sur le flot lui même (à part une augmentation de la gigue) mais peut par contre résulter en une violation des garanties dûes aux autres flots.

Plus grave, les effets de la gigue peuvent s'accumuler lors des traversées de nœuds successifs. Si dans le pire des cas, on peut comme nous l'avons montré obtenir un grumeau de deux paquets complètement collés après un nœud, la taille de grumeau maximum théorique après n nœuds sera de $n+1$ paquets collés (figure 5.3). Évidemment, il s'agit d'un cas que

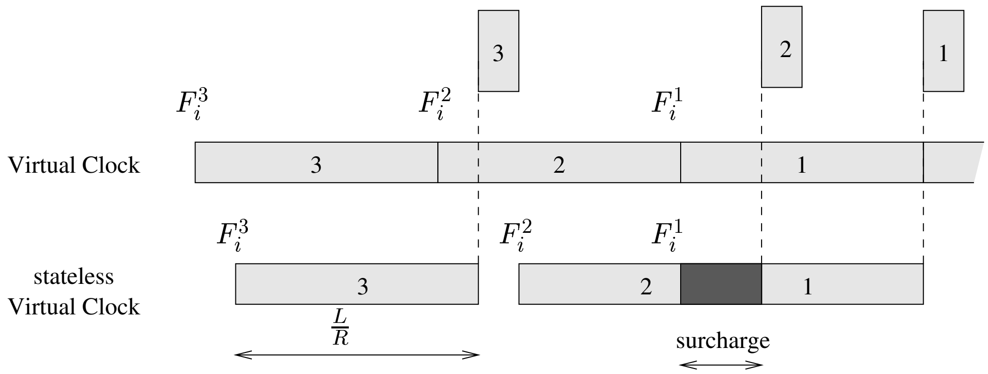


Fig. 5.2 Service de deux paquets du même flot en parallèle

nous qualifierons de *pire* car hautement improbable mais qu'il n'est pas possible d'exclure formellement.

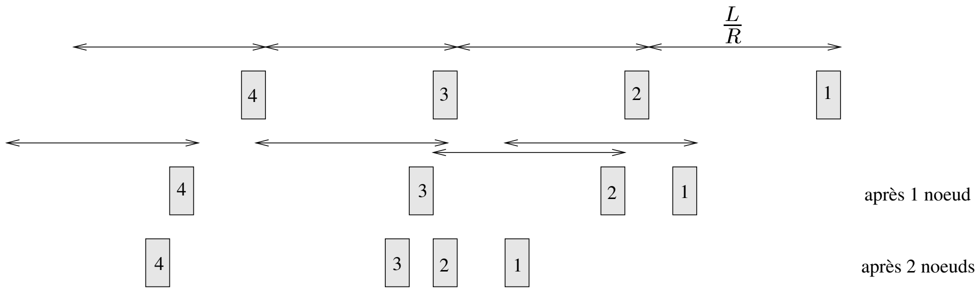


Fig. 5.3 Accumulation d'un grumeau

Influence de la charge

Ce phénomène d'agglomération dépend essentiellement de la charge. Soit l la charge instantanée à un moment donné sur un nœud. Le délai maximal subi par un paquet et correspondant au délai de paquetisation dans le modèle fluide est égal à $l \times \frac{L}{R}$. Cela s'explique par le fait que dans l'ordonnanceur discret, ce délai correspond à la résolution de la contention entre les paquets. Ainsi, quand la charge à un instant donnée est maximale, il faut au maximum un temps égal à L/R pour “écouler” tout le trafic. Si la charge est réduite de x %, moins de paquets sont en contention, le temps nécessaire pour écouler toutes les données est lui aussi réduit de x %.

Cependant, la charge instantanée maximale possible sur un nœud croit avec la déformation du trafic. Des grumeaux de taille plus élevée signifient une charge instantanée potentielle plus élevée. Ainsi, la charge influe sur la déformation du trafic, et la déformation du trafic induit une charge maximale plus importante au nœud suivant. La déformation sera alors plus importante, etc...

Acceptabilité de la gigue

À nouveau la charge Appelons *charge statique* le rapport de la somme des réservations allouées sur la capacité du lien :

$$l_s = \frac{\sum_j R_j}{C}$$

Si l_s est inférieure à un, le réseau est en fait *sous provisionné*. Ainsi, un flot i auquel est réservée une bande passante R_i peut en fait consommer une bande passante jusqu'à $\frac{R_i}{l_s}$ (puisque $\sum_j \frac{R_j}{l_s} = C$).

Dans une telle situation, une surcharge pour chaque flot (c'est à dire des grumeaux) est acceptable sans risquer de violer aucune des garanties faites aux autres flots. Pour prendre un exemple très simple, une charge de 50 % signifie que deux fois plus de bande passante que ce qui a été réservé est disponible instantanément. Des grumeaux de taille deux n'auront alors aucun effet sur les garanties ; dans le cas *pire* où un grumeau de taille deux pour chaque flot arrivent en même temps sur l'ordonnanceur, l'ensemble de la bande passante disponible est consommée à cet instant sans qu'aucun flot ne reçoive moins de bande passante que ce qui lui avait été réservé.

Cas des flots à faible débit Pour les flots qui utilisent moins que leur débit réservé (flots à faible débit demandant un délai faible par exemple), les effets de l'agglomération sont retardés. Si le flot a un débit r et une réservation R, l'incertitude sur l'instant d'émission est égale à $L/R < L/r$.

Il s'agit en fait du même phénomène que nous venons de décrire avec la charge, mais agissant au niveau d'un flot et non pas globalement. Le flot disposant de plus de ressources qu'il n'en consomme, il dispose d'une certaine “marge” de déformation acceptable avant de sur-consommer des ressources, éventuellement au détriment d'autres flots.

Influence globale de la charge

En limitant la charge, nous limitons la proportion de la capacité du lien allouée aux flots GS. Le trafic *best-effort* (ou d'une autre classe de service moins exigeante) traité à une priorité plus faible (c'est à dire uniquement quand aucun des flots GS n'utilise les ressources) n'entre pas en compte dans cette charge. On peut donc obtenir les effets d'une charge réduite sans pour autant limiter l'utilisation du réseau, mais en limitant la charge statique (c'est à dire la somme de la bande passante réservée divisée par la bande passante du lien) en flots GS.

Ainsi, si l'on limite la charge statique en flots GS à 50 % de la capacité du réseau, on réduit le rythme de création des grumeaux tout en rendant possible le traitement de grumeaux de taille deux paquets sans risquer aucune violation des garanties.

Au premier nœud, les flots ayant été lissés à l'entrée de la zone sans état, nous considérerons qu'il n'y a aucun grumeau. La charge instantanée l_1 est alors limitée par la charge statique l_s. Une déformation du trafic est introduite, et dans le pire des cas des grumeaux de taille 2 sont créés. Ainsi, pour le prochain nœud la charge instantanée potentielle est doublée.

L'intervalle d'incertitude sur le moment d'émission des paquets au prochain nœud double donc, et des grumeaux de quatre paquets peuvent apparaître, qui doublent la charge au nœud suivant et ainsi de suite. En fin de compte, la charge instantanée maximale après n nœuds est :

$$l_n = 2^n \times l_s$$

Et l'on veut assurer que la charge instantanée ne dépasse jamais la charge maximale acceptable pour chaque flot compte tenu du sous provisionnement du réseau, soit :

$$l_s \times R_i \leq \frac{R_i}{l_s} \Longleftrightarrow l_n \leq \frac{1}{l_s}$$

Donc :

$$2^n \times l_s \leq \frac{1}{l_s} \Longleftrightarrow 2^n \leq \frac{1}{l_s^2} \Longleftrightarrow n \leq log_2 \frac{1}{l_s^2}$$

ce qui est représenté sur la figure 5.4.

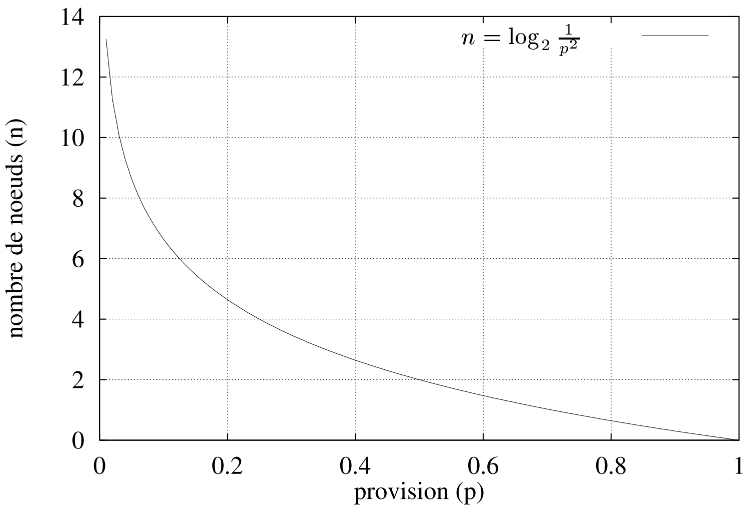


Fig. 5.4 Longueur maximum en fonction de la charge

On voit très bien qu'une faible réduction de la charge a très peu d'effet ; par contre des limitations plus importantes changent significativement la situation : jusqu'à cinq nœuds peuvent être traversés avec une charge proche de 20 %. Ce résultat n'est cependant valable que si la déformation des flots rencontrés dans les nœuds augmente progressivement le long du chemin, ce qui dépend de la topologie du réseau.

Nous insistons sur le fait que ce résultat correspond à une situation *pire* et hautement improbable. En pratique, on devrait donc obtenir des comportements bien meilleurs.

Nous appelons *stateless VC* la discipline de service inspirée de *Virtual clock* et ne maintenant pas l'état F_i^{k-1}. Elle utilise le calcul suivant pour le calcul de l'estampille d'un paquet :

$$F_i^k = A_i^k + \frac{L_i^k}{R_i} \tag{5.2}$$

Évidemment, deux situations sont possibles, dans lesquelles l'algorithme émule correctement ou non le TDM.

(1)	$A_i^k \geq F_i^{k-1}$	correct
(2)	$A_i^k < F_i^{k-1}$	erreur

Dans le cas (1), le paquet arrive après le F_i du paquet précédent, $A_i^k > F_i^{k-1}$ et donc le fonctionnement est identique à la version avec états.

Dans le cas (2), le paquet arrive avant le F_i du paquet précédent. Il y a donc là erreur et mauvaise émulation du TDM. Néanmoins, si certaines précautions sont prises, cette erreur devrait se produire assez peu et surtout ses conséquences peuvent être contrôlées.

5.4.5 Suppression partielle du terme F_i^{k-1}

Nous venons de voir qu'il est possible dans certaines conditions de ne pas contrôler à chaque nœud la gigue prise par les paquets. Cependant on peut aussi dans une certaine mesure contrôler cette gigue sans pour autant créer un état dédié. L'idée est d'utiliser l'information déjà présente dans la file de paquets en attente.

Rappelons que les paquets en contention sont triés sur leur estampille F_i^k dans une file d'attente. Ils doivent être stockés avec leur estampille pour conserver une file triée lors de l'insertion de nouveaux paquets. Ainsi quand un paquet arrive, si on sait qu'un paquet du même flot est en attente, on connaît la valeur à utiliser pour le F_i^{k-1}. Évidemment, le fait qu'aucun paquet du même flot ne soit présent dans la file n'assure pas que l'on ait $A_i^k \geq F_i^{k-1}$. f_i^k étant la date de départ effectif du $k^{ième}$ paquet du flot i, le calcul à appliquer est alors :

$$\begin{cases} F_i^k = A_i^k + \frac{L_i^k}{R_i} & si \quad A_i^k \geq f_i^{k-1} \\ F_i^k = F_i^{k-1} + \frac{L_i^k}{R_i} & si \quad A_i^k < f_i^{k-1} \end{cases} \tag{5.3}$$

Trois situations sont possibles :

(1)	$A_i^k \geq F_i^{k-1}$	correct
(2)	$A_i^k \leq f_i^{k-1} \leq F_i^{k-1}$	correct
(3)	$A_i^k < F_i^{k-1} \leq f_i^{k-1}$	erreur

Dans le cas (1), le paquet arrive après le F_i du paquet précédent, $A_i^k > F_i^{k-1}$ et donc le fonctionnement est identique à la version avec états.

Dans le cas (2), le paquet arrive avant le F_i du paquet précédent et *avant* le départ effectif de ce paquet. La valeur F_i^{k-1} est récupérée et le fonctionnement est identique à la version avec états.

Dans le cas (3), le paquet arrive avant le F_i du paquet précédent et *après* le départ effectif de ce paquet. La valeur F_i^{k-1} n'est plus disponible et l'algorithme suppose que l'on se trouve dans le cas (1). Il y a donc là erreur et mauvaise émulation du TDM.

L'utilisation de cet algorithme suppose deux choses :

- que l'on puisse identifier un flot,
- que l'on puisse retrouver un paquet d'un flot donné (s'il existe) dans la file.

L'identification d'un flot peut se faire de manière classique (paire d'adresses et ports source/destination) ou en utilisant un label attribué par le routeur ingress, ce qui permettrait un traitement plus simple. Ce label pourrait par exemple être constitué d'un identificateur de l'ingress et d'un numéro de flot unique pour cet ingress. Une liste des estampilles des derniers paquets de chaque flot présents dans la file peut être conservée dans une table hachée utilisant l'identificateur de flot comme clé. Ces opérations demandent un traitement supplémentaire par paquet. Leur complexité ne dépend pas du nombre total de flots mais du nombre de flots dont au moins un paquet est présent dans la file qui devrait être beaucoup plus faible à tout instant.

Nous obtenons une discipline de service située entre le *statefull VC* et le *stateless VC*, que nous appellerons *reduced state VC* puisqu'elle n'utilise qu'une quantité réduite d'état déjà disponible. La motivation derrière cette proposition est que cela devrait être efficace en cas de grumeaux sévères, donc limiter les effets dans les cas les plus graves. Ainsi, si des violations de garanties ont lieu du fait d'une gigue excessive, c'est qu'il y a une augmentation significative de la contention. Intuitivement, il est clair que dans cette situation la probabilité du cas (2) augmente alors que celle du cas (3) diminue.

5.4.6 L'architecture proposée

Comme au chapitre 4, nous considérons un ensemble de réseaux dans lesquels le modèle IntServ/RSVP est déployé, interconnectés par un backbone utilisant des ordonnanceurs sans état par flot. Nous nous intéressons uniquement au cas des flots GS, les flots CL étant par exemple traités par des mécanismes du type SRP [FALB98]. Les flots GS sont donc lissés à leur débit réservé R à l'entrée du backbone, par les nœuds *ingress* qui placent également la valeur de R dans un champ de l'en-tête du paquet. L'opération inverse est réalisée à la sortie par le routeur *egress* (figure 5.5). Un contrôle d'admission par flot est réalisé de façon à limiter la charge en flots GS à une valeur fixée, par exemple 80 % de la capacité sur chaque lien.

Dans les nœuds internes, tous les flots autres que GS sont traités en priorité inférieure (plusieurs niveaux de priorité ou tout autre mécanisme peuvent être utilisés, par exemple pour séparer trafic *best-effort* et flots CL).

Nous résumons les différences des deux variantes *scalable* de l'algorithme *virtual clock* avec la version complète (*statefull*) dans le tableau 5.2.

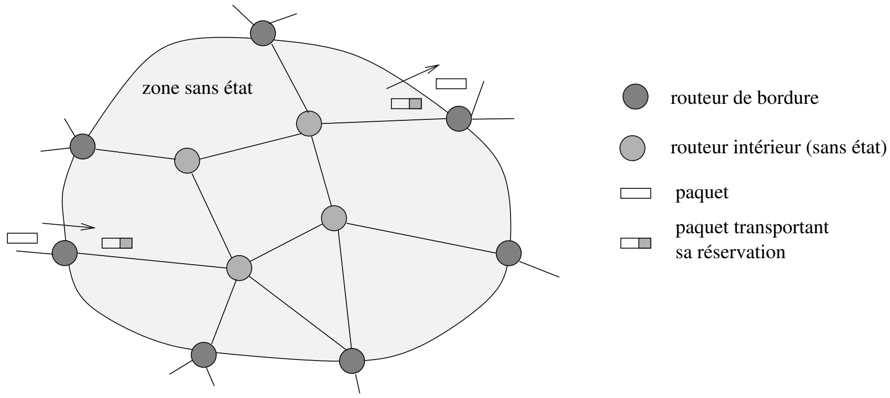


Fig. 5.5 Architecture de la solution

	Statefull VC	Reduced state VC	Stateless VC
1	$R_i \leftarrow$ état du flot $F_i^{k-1} \leftarrow$ état du flot	$R_i \leftarrow$ paquet $F_i^{k-1} \leftarrow$ file si $A_i^k < f_i^{k-1}$	$R_i \leftarrow$ paquet
2	$F_i^k = max(A_i^k, F_i^{k-1}) + \frac{L_i^k}{R_i}$	$F_i^k = A_i^k + \frac{L_i^k}{R_i}$ si $A_i^k \geq f_i^{k-1}$ $F_i^k = F_i^{k-1} + \frac{L_i^k}{R_i}$ si $A_i^k < f_i^{k-1}$	$F_i^k = A_i^k + \frac{L_i^k}{R_i}$
3	insertion du paquet dans la file		
4	état du flot $\leftarrow F_i^k$		

Tab. 5.2 Les trois algorithmes VC

5.5 SIMULATIONS

5.5.1 Scénario

Le scénario de nos simulations est représenté sur la figure 5.6. Nous considérons un ensemble de flots de référence (sur lesquels nous effectuerons des mesures) traversant un certain nombre de nœuds dans un backbone et demandant des garanties de délai. L'ordonnanceur utilisé dans les nœuds est une des trois variantes *stateful/stateless/reduced state VC*. À chaque nœud, des flots *parasites* viennent partager la bande passante du lien aval avec les flots de référence. Ces flots parasites sont supposés avoir déjà traversé un certain nombre de nœuds implémentant le *stateless/reduced state VC* et présentent donc des grumeaux susceptibles de perturber le service dû aux flots de référence.

Tous les flots (sauf un de référence et un parasite à chaque nœud) sont à débit constant (on suppose qu'ils ont été lissés à l'entrée du backbone) et ont des ressources réservées dans les ordonnanceurs. Les flots restant (un flot de référence et un parasite à chaque nœud) sont des flots *best-effort* et émettent à un débit élevé, déterminé de façon à toujours obtenir une utilisation maximale du réseau.

Dans nos simulations, nous faisons varier le nombre de nœuds – `nbcore` –, les caractéristiques des flots parasites (taille – `grum` – et type des grumeaux) et la charge en flots demandant des garanties de délai. Plus exactement, c'est la capacité des liens que nous faisons varier.

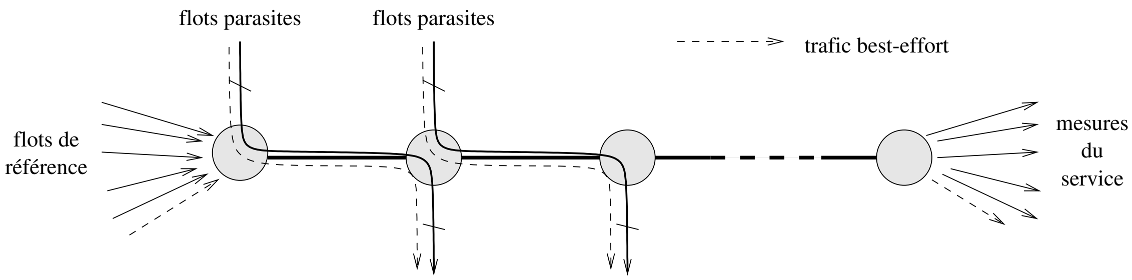


Fig. 5.6 Topologie simulée

Pour chaque valeur des paramètres, nous comparons les résultats obtenus avec les trois variantes de l'ordonnanceur. Pour toutes les simulations il y a 10 flots parasites à chaque nœud et 10 flots de référence. Nous avons choisi un nombre réduit de flots pour des raisons pratiques de mise en place et de temps de simulation. Cela n'est pas un problème dans la mesure où nous ne cherchons pas à montrer la scalabilité de notre solution (elle l'est car elle ne nécessite pas la gestion d'états par flot), mais plutôt à évaluer la qualité du service fourni par rapport à un modèle utilisant des états par flot.

Tous les flots ont une taille de paquet identique et égale à 1000 octets, sauf un des flots de référence qui émet des paquets de 500 octets (ce qui implique une garantie de délai plus faible). Nous effectuerons les mesures sur deux des flots de référence, le flot 1 (paquets de 1000 octets) et le flot 3 (paquets de 500 octets).

5.5.2 Réalisation

Nous utilisons le simulateur `ns-2` [FE99] développé au *Lawrence Berkeley National Laboratory* de l'Université de Californie à Berkeley. Celui-ci n'offre pas d'ordonnanceur *Virtual Clock* en standard, mais une implémentation en a été réalisée pour une ancienne version de `ns` par Markus A. Wischy à l'Université de l'Orégon [Wis98]. Nous avons porté cette implémentation sur la dernière version de `ns` et l'avons modifée de façon à obtenir un comportement soit en VC classique soit l'une de nos deux variantes sans état.

Il nous fallait ensuite générer du trafic présentant des grumeaux de façon paramétrable pour les flots parasites. Nous avons utilisé deux méthodes. Pour la première, nous avons modifié le générateur de trafic à débit constant de `ns` (`Application/Traffic/CBR`) pour obtenir de grandes variations du temps inter paquets. Ainsi modifiée la source CBR génère aléatoirement des grumeaux de taille variable. Le trafic obtenu est ensuite conditionné par un régulateur utilisant un seau de jetons (*token bucket filter*, `TBF`) de façon à limiter la taille maximum de grumeaux à la valeur désirée.

Nous avons aussi voulu observer le comportement du système dans le cas *pire*, c'est à dire avec des flots parasites exhibant systématiquement des grumeaux de taille fixée. Pour cela, un altérateur de trafic `Clumper` a été défini pour modifier le trafic généré par les sources

CBR. Son fonctionnement est très simple, il retient simplement les paquets qu'il reçoit dans une file d'attente jusqu'à ce que la file atteigne une taille configurée, il émet alors tous les paquets en attente dans la file.

Nous appellerons les grumeaux générés de cette façon des *grumeaux lourds* (car très sévères) et ceux générés avec la première méthode des *grumeaux légers*. Ceux-ci correspondent à des situations plus réalistes, alors que les grumeaux lourds peuvent nous fournir des indications sur le comportement de l'algorithme en situation extrême et donc nous aider à comprendre son fonctionnement. Les caractéristiques de ces deux types de trafics parasites sont décrits dans la section suivante.

5.5.3 Caractérisation des flots parasites

Nous proposons d'utiliser comme méthode de caractérisation d'un flot parasite la représentation graphique de la distribution des temps inter paquets de ce flot. Pour cela nous prenons un groupe de deux cents paquets consécutifs du flot, calculons les temps inter paquets que nous trions et divisons par le temps nominal afin de normaliser les valeurs. Nous traçons la courbe obtenue.

Ainsi, un flot à débit constant parfait aura une caractérisation correspondant à une ligne horizontale d'ordonnée un (équation $y = 1$). La figure 5.7 montre les courbes obtenues dans le cas des grumeaux légers pour différentes configurations du régulateur.

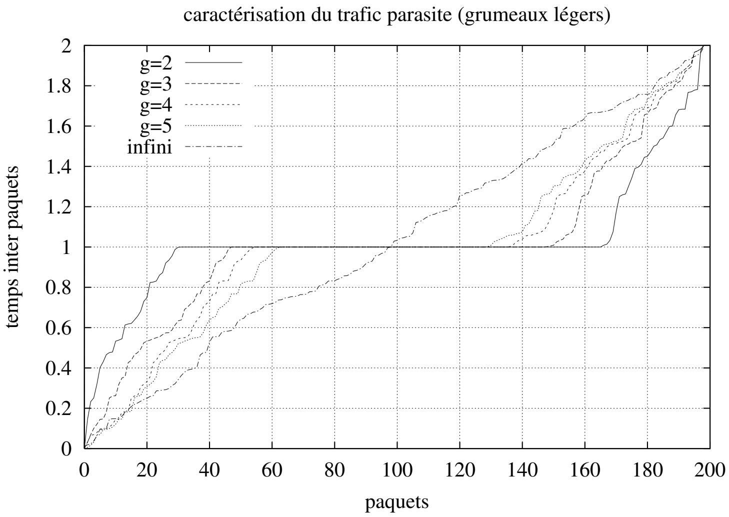
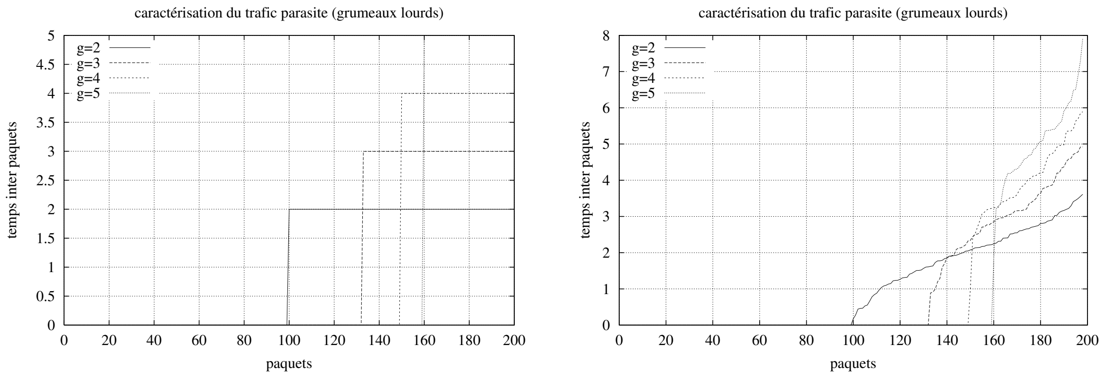


FIG. 5.7 Caractérisation du trafic parasite (grumeaux légers)

Si aucun seau de jetons ne conditionne le trafic généré par la source CBR modifiée, le temps inter paquets est réparti uniformément entre 0 secondes et deux fois le temps nominal. On obtient donc un segment de droite variant entre $y = 0$ et $y = 2$. Quand un seau de jetons vient limiter la taille de rafale (donc la taille des grumeaux), une partie des paquets est contrainte au débit nominal (lissage) ce qui apparaît graphiquement comme un segment de droite central sur $(y = 1)$, dont la longueur est inversement liée à la profondeur du seau de jetons utilisé.



(a) source CBR “pure” (b) source CBR modifiée

FIG. 5.8 Caractérisation du trafic parasite (grumeaux lourds)

La figure 5.8 montre les caractérisations obtenues pour les grumeaux lourds. La sous figure 5.8(a) représente le cas où le *clumper* est appliqué au trafic issu d'une source CBR parfaite. On voit, ce qui correspond tout à fait à l'intuition, que si les grumeaux formés sont de taille 2, la moitié des paquets a un temps inter paquets nul, et l'autre moitié un temps double de la normale. Le même raisonnement s'applique pour toutes les tailles de grumeaux.

La sous figure de droite 5.8(b) correspond au cas où le *clumper* est appliqué au trafic issu d'une source CBR modifiée, dont la répartition des temps inter paquets est uniforme dans l'intervalle compris entre zéro et deux fois le temps nominal. C'est le cas que nous avons utilisé dans nos simulations.

Dans tous les cas, il nous paraît clair que l'effet du trafic parasite sur le comportement de l'ordonnanceur sans état dépendra de son *agressivité* qui correspond graphiquement à la partie de la courbe située sous la droite $(y = 1)$.

5.5.4 Mesures

Le but des simulations est de déterminer dans quelles conditions nos deux variantes du *Virtual Clock*, *stateless VC* et *reduced state VC* ont un comportement (et fournissent des garanties) identique ou très proche de celui de l'ordonnanceur de référence, le *statefull VC*.

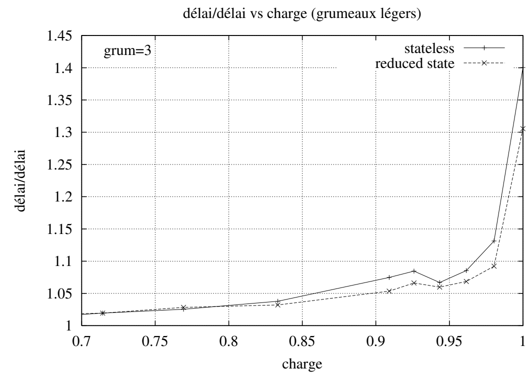
Pour chaque configuration de simulation, nous mesurons :

1. les délais maximaux,
2. la caractérisation du trafic en sortie, telle que définie en 5.5.3.

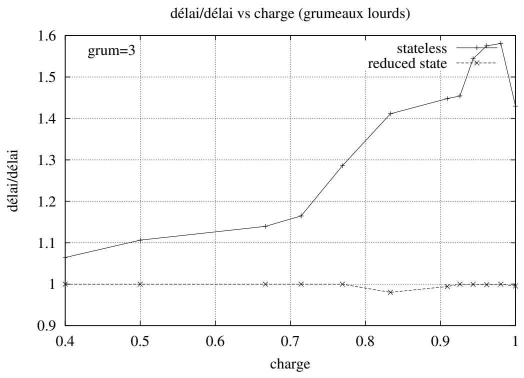
Les délais permettront d'évaluer le comportement de l'ordonnanceur tandis que la caractérisation du trafic permettra de vérifier la validité du modèle de trafic utilisé pour les flots parasites.

Déviation de délai

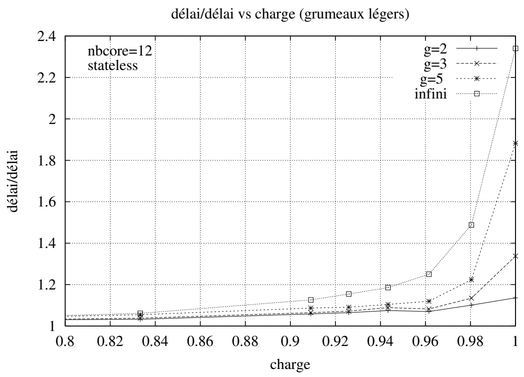
Nous ne voulons pas mesurer le délai fourni par chacune des variantes mais plutôt la différence par rapport au délai de référence. Nous définissons donc le *sur-délai* dans un contexte donné comme étant le rapport du délai fourni par l'ordonnanceur sans ou avec peu d'états sur le délai fourni par l'ordonnanceur de référence. Toutes les courbes de sur-délai présentées dans la suite sont celles du flot 1, les courbes sont globalement similaires pour le flot 3.



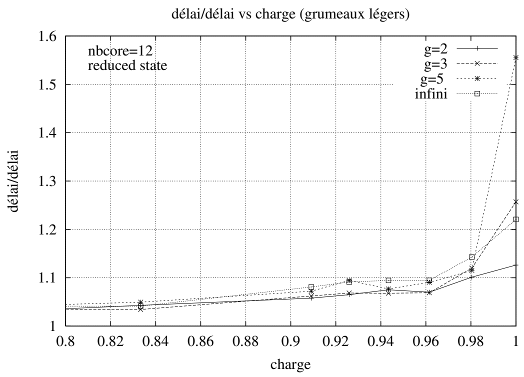
(a) Grumeaux légers



(b) Grumeaux lourds



(c) Grumeaux légers – stateless



(d) Grumeaux légers – reduced state

Fig. 5.9 Sur-délai vs charge

Les figures 5.9(a) à (d) montrent le sur-délai de chacune des deux variantes en fonction de la charge pour les deux types de grumeaux pour différentes configurations. Dans les figures

(a) et (b), le nombre de nœuds traversés est fixé à 6. La première montre très nettement que le sur-délai reste très faible tant que la charge est inférieure à 95 %, ce qui nous paraît un résultat extrêmement positif. Dans la deuxième, l'effet des grumeaux lourds se fait plus sentir et le sur-délai augmente progressivement avec la charge. Nous expliquons l'aspect de la courbe du *reduced state VC* plus bas, en commentant la figure 5.11. Les figures (a) et (b) correspondant à des tailles de grumeaux limitées à trois, nous montrons avec les figures (c) et (d) l'influence de cette limitation. Celle-ci ne devient sensible que quand la charge approche de très près les 100 %. On note tout particulièrement que pour le cas pire où les grumeaux ne sont pas contraints (noté "infini") les courbes restent dans le même ordre de grandeur.

Les figures 5.10 et 5.11 montrent le sur-délai en fonction de la taille des grumeaux des flots parasites pour différentes valeurs de la charge (le nombre de nœuds traversés est toujours fixé à 6). On voit sur la figure 5.10 qu'avec une charge de 100 %, le sur-délai augmente nettement avec la taille des grumeaux. Deux résultats intéressants sont visibles sur cette figure. Premièrement, avec une charge maximale, l'ordonnanceur *reduced state VC* se comporte nettement mieux que le *stateless VC*, ce qui s'explique par le fait que la charge maximale résulte en une contention assez forte et que la situation (2) (décrite page 96) est assez fréquente. Deuxièmement, l'effet d'une réduction même minime de la charge est très important comme le montre la courbe correspondant à une charge de 98 %. Si la charge est réduite à 90 %, le sur-délai est extrêmement faible (moins de 5 et 7 % de délai ajouté) quelle que soit la taille des grumeaux. La différence entre les deux variantes est alors très légère mais visible.

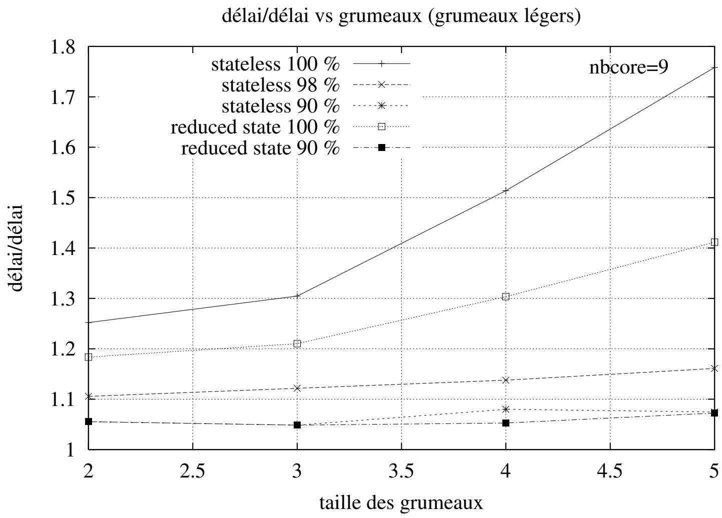


Fig. 5.10 Sur-délai vs taille des grumeaux

La figure 5.11 montre les mêmes courbes dans le cas de grumeaux lourds. Nous constatons que l'impact des grumeaux est beaucoup plus important et que l'effet d'une diminution de charge est nettement moins visible.

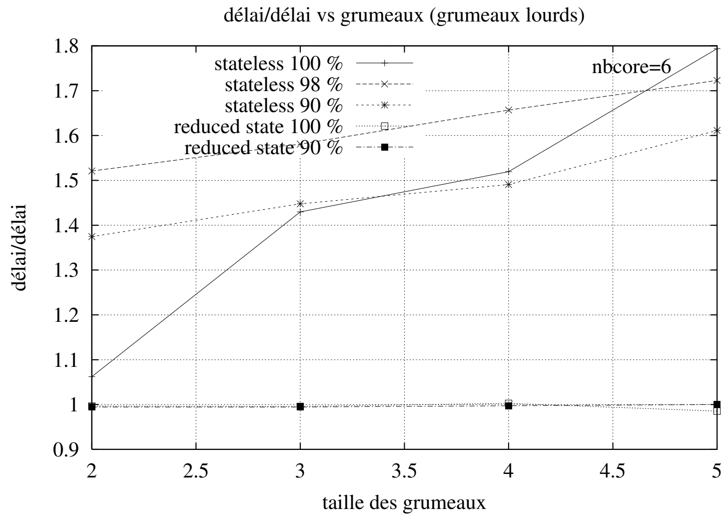


Fig. 5.11 Sur-délai vs taille des grumeaux

Le *reduced state VC* fournit alors exactement le même délai que la référence. Cela s'explique par le fait que les paquets collés dans les grumeaux lourds créent systématiquement la situation (2).

La troisième paire de figures montre le sur-délai en fonction du nombre de nœuds traversés sous diverses charges. Dans le cas de grumeaux légers (figure 5.12), on retrouve à nouveau des sur-délais très faibles tant que la charge ne dépasse pas 90 % et un avantage significatif du *reduced state VC* quand la charge approche 100 %, toujours pour les mêmes raisons. L'effet d'une très légère diminution de charge est à nouveau très net. On constate également que le sur-délai est globalement indépendant du nombre de nœuds, c'est à dire que le délai absolu ajouté augmente proportionnellement au nombre de nœuds traversés. La figure 5.13 montre les mêmes résultats pour des grumeaux légers non contraints, avec simplement des valeurs de surdélai un peu plus élevées.

Dans le cas de grumeaux lourds (figure 5.14), on voit à nouveau que le *reduced state VC* devient équivalent à un *statefull VC*. Avec le *stateless VC*, le sur-délai reste significatif tant que la charge ne descend pas au dessous de 70 %.

Caractérisation du trafic en sortie

Nous comparons maintenant la caractérisation des flots de référence en sortie du réseau avec celle des flots parasites dans le cas des grumeaux légers. Nous considérons que les résultats des simulations sont valables quand les flots de référence en sortie du réseau ne sont pas plus agressifs que les flots parasites.

Une première remarque que l'on peut faire à partir des figures 5.15 et 5.16 est que les caractérisations des flots de référence en sortie et des flots parasites ont une allure assez proche, bien qu'un peu différente, les flots de référence montrant une caractérisation plus “lisse”. Il ne sera donc pas toujours possible de juger de manière catégorique de l'agressivité comparée des deux caractérisations.

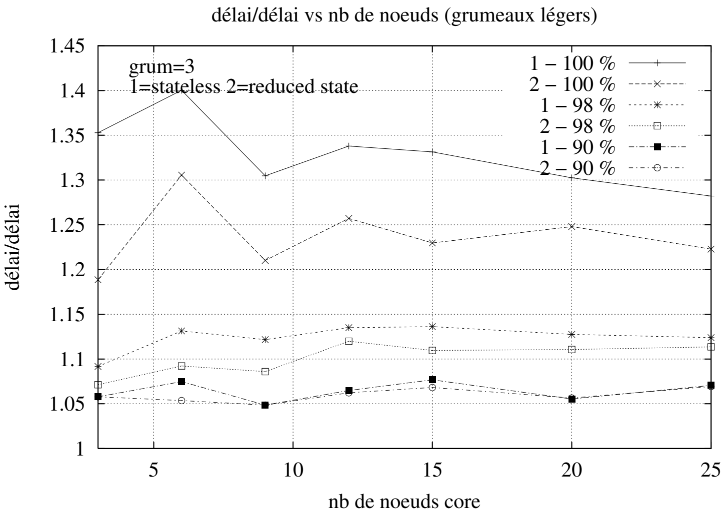


Fig. 5.12 Sur-délai vs nbre de nœuds (grumeaux légers)

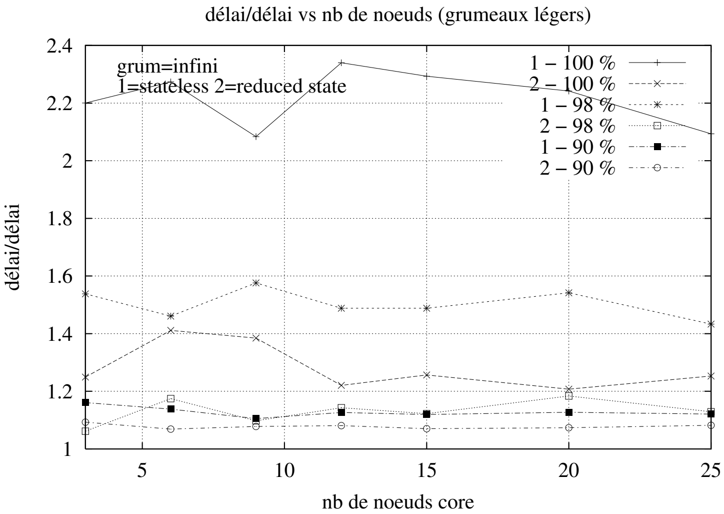


Fig. 5.13 Sur-délai vs nbre de nœuds (grumeaux légers) – cas pire

La figure 5.15 montre le cas du flot 1. On constate que dans tous les cas, et à charge maximale, le flot est assez peu agressif puisque le temps inter paquets minimal ne descend pas au dessous de la moitié du temps nominal. On voit très bien qu'il est nettement moins agressif qu'un flot parasite contraint à une taille de grumeaux de trois. On peut donc considérer que les résultats obtenus avec des grumeaux de taille trois ou supérieure sont valables. La comparaison avec la courbe parasite pour des grumeaux de taille deux est plus difficile.

La figure 5.16 montre le cas du flot 3. L'impact que la traversée du réseau a eu sur sa caractérisation est nettement plus élevé que pour le flot 1 (le temps inter paquets minimal approche la valeur 0), ce que nous attribuons au fait que ses paquets sont de taille nettement

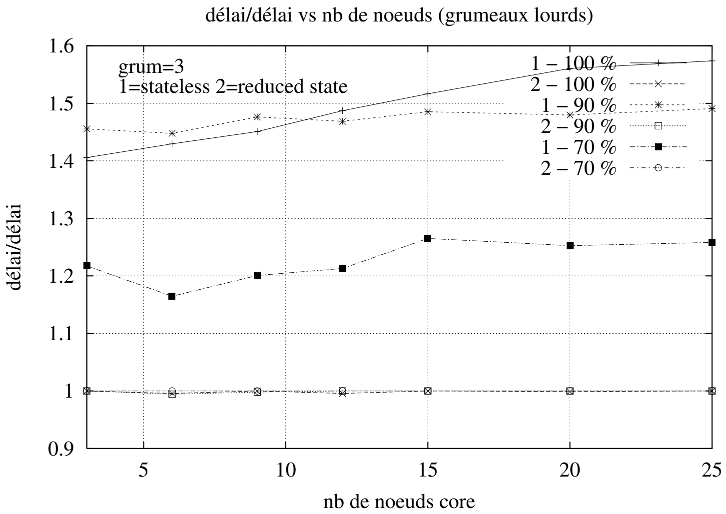


Fig. 5.14 Sur-délai vs nbre de nœuds (grumeaux lourds)

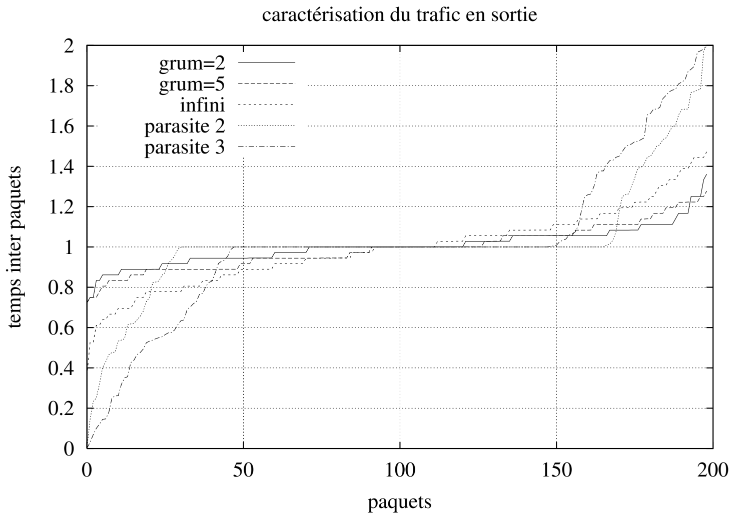
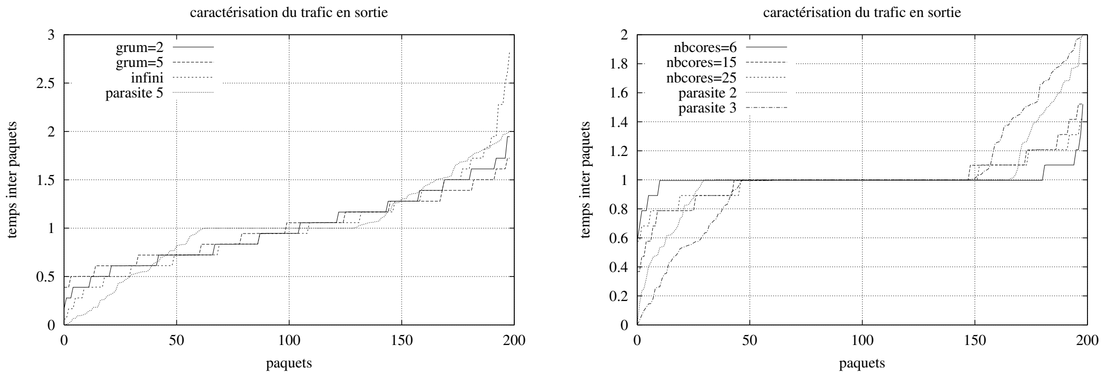


Fig. 5.15 Caractérisation du flot 1

inférieure (moitié) à celle des flots parasites. L'aspect plus "saccadé" peut s'expliquer de la même façon.

La figure 5.16(b) montre l'influence d'une diminution de la charge sur la caractérisation du flot 3. Avec une charge de 94 %, les flots parasites n'étant pas contraints, le flot est très clairement moins agressif qu'un trafic parasite avec des grumeaux de taille 3.

Ainsi, les résultats obtenus en terme de délai semblent être valables dans pratiquement n'importe quelles conditions si les tailles de paquets des flots parasites et de référence sont identiques. Dans le cas de tailles de paquets différentes, on obtient des résultats similaires avec une légère diminution de la charge.

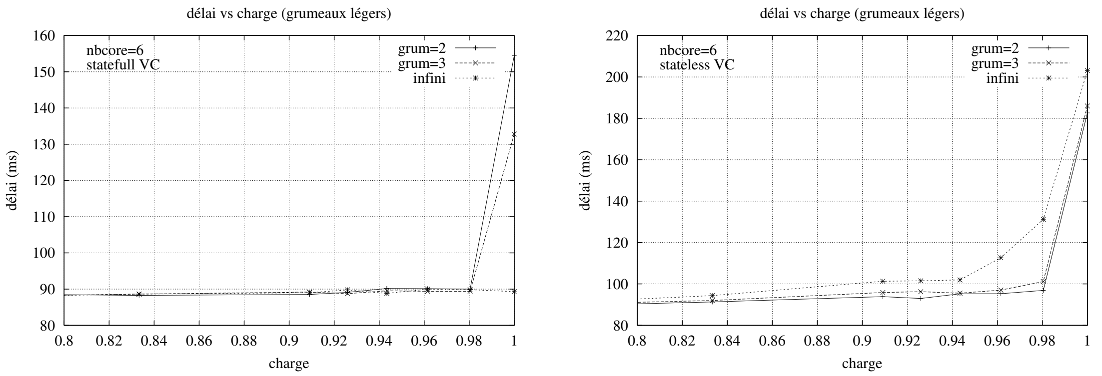


(a) Charge de 100 % (b) Charge de 94 %

Fig. 5.16 Caractérisation du flot 3

Délais absolus

Il nous paraît intéressant de montrer que même dans les cas où l'ordonnanceur sans état fournit un délai sensiblement plus important que l'ordonnanceur de référence, cela n'entraîne pas forcément une violation des délais garantis aux flots.



(a) Statefull VC (b) Stateless VC

Fig. 5.17 Délais absolus

Ainsi, la figure 5.17 montre les délais en fonction de la charge obtenus avec les ordonnanceurs statefull et stateless VC. Le délai garanti de manière déterministe est de 175 ms, et l'on constate que la plupart du temps les délais maximaux effectifs sont bien inférieurs, ce qui est un résultat connu.

Conclusion

Les résultats de nos simulations montrent que notre solution semble fonctionner correctement dès que l'on réduit légèrement la charge en flots à garanties de délai. Même avec des tailles de grumeaux élevées, le délai ajouté par rapport au modèle de référence est très faible dès que la charge ne dépasse pas 90 % pour des grumeaux légers, les plus réalistes.

On peut aussi prendre en compte le fait que le modèle de référence fournit toujours des délais nettement moins élevés que ceux qui sont garantis comme montré dans les courbes de la figure 5.17. Il peut donc être parfois tout à fait acceptable d'obtenir un sur-délai significatif.

La contrainte de devoir limiter la proportion de flots à garantie de délais ne nous semble pas gênante. La plupart des applications peuvent se satisfaire de garanties moins strictes et constitueront donc une part significative du trafic transporté par le réseau.

Enfin, même dans le cas d'un réseau qui ne transporterait que ce type de flots, le sur-dimensionnement nécessaire pour limiter la charge pourrait être très raisonnable. Ainsi, la figure 5.9(a) montre des courbes pratiquement plates jusqu'à 90 % de charge.

5.6 COMPARAISON AVEC DPS

Dans [SZ99], Stoica et Zhang obtiennent des délais *déterministes* en utilisant la technique DPS (*Dynamic Packet State*) ce qui rend leur solution supérieure à la notre sur le plan théorique. Cependant, la technique qu'ils utilisent pour obtenir ce résultat est beaucoup plus complexe et nécessite un supplément de traitement par paquet important.

Ils proposent d'utiliser un ordonnanceur CJVC (*Core Jitter Virtual Clock*). Il s'agit d'une version sans état de *Jitter Virtual Clock*, une variante non *work-conserving* de VC qui conserve la gigue entre les paquets. Comme il est impossible de réduire les variations sur le moment d'émission d'un paquet dans un nœud ($L/R + L_{max}/C$), l'écart entre le moment effectif (f_i^k) et le moment théorique d'émission (F_i^k) est stocké dans le paquet au moment de l'émission. Cette valeur qui représente "l'avance" prise par le paquet dans ce nœud est ajoutée dans le calcul du F_i^k au nœud suivant. La gigue est ainsi compensée à chaque nœud.

Le fait d'utiliser un algorithme qui conserve la gigue leur permet aussi de traiter le cas des paquets de tailles différentes. Il suffit pour cela de compenser l'avance éventuelle prise par un paquet sur son prédécesseur en ajoutant à chaque nœud un délai adéquat. Ce délai est calculé par le nœud *ingress* qui le place dans un champ d'en-tête du paquet.

Au niveau du plan de contrôle, ils proposent de n'utiliser des messages (non fiables) que pour ajouter des réservations, l'annulation étant prise en compte par une technique de mesure du débit réservée semblable à notre proposition, mais nettement plus complexe.

Notre solution est résolument plus simple et légère. Nous utilisons un ordonnanceur qui conserve le travail, simple à implémenter. Aucune modification particulière du contenu des champs d'en-tête des paquets n'est nécessaire à l'intérieur de la zone sans état. Enfin nous n'avons besoin que d'un champ d'en-tête pour stocker la valeur de R_i.

En conclusion, notre proposition est plus simple à implémenter, demande moins de puissance de traitement et nécessite moins de supplément d'en-tête dans les paquets. Par contre, les garanties ne sont pas assurées de manière déterministe. Notre approche est plus basée sur des hypothèses raisonnables et vérifiables en pratique sur la déformation du trafic (création de grumeaux) et son effet sur les garanties.

5.7 CONCLUSION DU CHAPITRE

Dans ce chapitre, nous avons proposé une architecture permettant de fournir des garanties de délai individuelles aux flots (avec donc une flexibilité maximale) sans nécessiter le maintien d'état pour chaque flot. Les contraintes de cette solution sont de lisser les flots à l'entrée du *backbone* et de limiter (de façon peu drastique) la capacité allouée aux flots demandant des garanties de délai.

Les outils analytiques ne pouvant nous donner que des résultats non significatifs, nous avons eu recours à la simulation pour obtenir des données plus utilisables. L'utilisation réelle de ce modèle nécessiterait sans doute une analyse plus poussée des phénomènes entrant en jeu, en particulier :

- caractériser la création des grumeaux dans des situations réalistes,
- caractériser l'effet cumulé des grumeaux au long d'un chemin (en particulier l'influence de tailles de paquets différentes).

De plus, dans le cas de la variante *reduced state VC* de l'ordonnanceur, il serait nécessaire d'évaluer la complexité liée à la gestion des états, et en particulier la taille maximale de la file de paquets.

Nous pensons néanmoins que nos simulations ont montré, comme c'était leur but, que l'approche était utilisable dans des conditions bien moins drastiques que celles données par les résultats théoriques. Elle peut donc être présentée comme une solution possible pour la fourniture de garanties de délai strictes dans des environnements à très grand nombre de flots.

Chapitre 6

Conclusion

L'objectif de cette thèse était l'étude des problèmes rencontrés pour la gestion de la qualité de service dans l'Internet, et en particulier dans les réseaux d'interconnexion. Nous avons tout d'abord étudié les mécanismes de qualité de service qui sont actuellement proposés, que ce soit au niveau global (architectures conçues par les groupes de travail IntServ et DiffServ) ou au niveau des mécanismes élémentaires utilisés par ces architectures (garanties de délai, ordonnancement de paquets).

Pour que la qualité de service devienne réellement disponible, ces architectures doivent être déployées dans l'ensemble du réseau. Or, les réseaux d'interconnexion ont des caractéristiques particulières qui rendent difficile l'utilisation des mécanimes de qualité de service tels qu'ils ont été conçus jusqu'à présent. On peut classer ces problèmes en deux catégories :

- les problèmes liés au haut débit,
- les problèmes liés au facteur d'échelle.

Dans le troisième chapitre, nous avons étudié une des technologies majeures proposées pour les réseaux haut débit, la commutation IP. Les points forts de cette approche sont de pouvoir profiter des capacités de traitement haut débit de moteurs de commutation matériels (typiquement ATM) tout en conservant la flexibilité et la robustesse des mécanismes réseaux implémentés par le protocole IP. Notre contribution a consisté à proposer une intégration des mécanismes de qualité de service de l'architecture IntServ à cette technologie. Cette intégration passe par l'ajout des fonctionnalités de distribution de labels au protocole RSVP chargé de mettre en place les réservations. Nous avons aussi montré les avantages liés à la fourniture de la qualité de service au niveau cellule et proposé une implémentation des services permettant d'optimiser l'utilisation des ressources. Cette implémentation a été validée par des simulations, et la réalisation d'un prototype a montré la viabilité des mécanismes proposés pour le plan de contrôle.

Le fait d'utiliser un modèle de gestion par flot rend cependant cette solution sensible au facteur d'échelle. Les progrès réalisés au niveau des moteurs de commutation capables de fournir des mécanismes de qualité de service à un très grand nombre de flots nous portent à penser que les limitations se situent plutôt au niveau du plan de contrôle, en l'occurence le protocole RSVP. Une étude réalisée sur la seule implémentation librement disponible actuellement a confirmé l'existence probable du problème même si les résultats ne sont pas aussi mauvais qu'on pourrait le croire.

Concernant ces problèmes liés au facteur d'échelle, nous considérons qu'ils ne se posent réellement que dans une partie limitée du réseau (les *backbones*) que nous appelons *zone centrale*. Nous proposons donc d'utiliser l'architecture IntServ/RSVP telle quelle à l'extérieur de cette zone et de mettre en œuvre des mécanismes différents, adaptés aux problèmes d'échelle, dans la zone centrale. La viabilité de cette approche est liée à deux contraintes :

- les mécanismes utilisés dans la zone centrale doivent fournir le même type de garanties que l'IntServ,
- l'interfaçage des deux zones doit se faire de manière totalement transparente pour les applications.

Nous avons proposé deux types d'approches pour la fourniture de la qualité de service de manière scalable à intérieur de la zone centrale.

La première, décrite dans le quatrième chapitre, est basée sur un changement de granularité dans la gestion des ressources. Les flots applicatifs sont agrégés en *b-flots* correspondant à des couples de points d'entrée et sortie de la zone centrale pour une qualité de service donnée. À partir de la caractérisation du b-flot, les flots étant contraints à leurs contrats de trafic de manière appropriée, l'allocation de ressources déterminées au b-flot permet de fournir des garanties sur le service reçu par chacun des flots individuels. Nous montrons comment calculer ces ressources et réaliser l'interfaçage des deux zones. Nous étudions également brièvement les gains potentiels en efficacité de l'utilisation des ressources apportés par les techniques d'agrégation. Cette approche est satisfaisante dans la mesure où elle permet de fournir les garanties de services définies par l'IntServ de bout en bout, en éliminant les problèmes de scalabilité apportés par les réseaux d'interconnexion. Cependant l'agrégation implique une perte de flexibilité dans la gestion des ressources ce qui se ressent par l'impossibilité de gérer le multicast (bien que les problèmes du multicast ne soient pas tous liés au fait d'utiliser l'agrégation) et l'obligation de gérer un b-flot par délai pour fournir des délais hétérogènes aux flots GS.

Dans le cinquième chapitre, nous proposons une approche différente pour fournir la qualité de service dans la zone centrale. L'idée est de gérer les ressources par flot, mais sans maintenir d'état pour chaque flot dans les nœuds. Le trafic ayant été conditionné en entrée de la *zone sans état* et chaque paquet transportant sa réservation, un ordonnancement approximant le *Virtual Clock*, que nous appelons *stateless VC*, peut être mis en œuvre sans maintenir d'état par flot. La déviation du service fourni par *stateless VC* par rapport à VC dépend de la déformation du trafic introduite par les nœuds. Nous montrons l'existence de bornes théoriques sur cette déformation et sur la déviation du service qui en résulte. Ces bornes correspondent cependant à des cas pires et nous montrons par des simulations que l'effet réel que l'on peut observer en pratique est très inférieur, ce qui étend considérablement l'usage possible de la solution. Nous proposons également une variante du *stateless VC*, appelée *reduced state VC*, qui utilise les états disponibles dans les paquets présents dans la file d'attente pour améliorer l'approximation réalisée dans certaines situations.

Perspectives

Le groupe de travail RSVP de l'IETF a récemment annoncé que son travail était terminé, la spécification du protocole étant achevée. Quelques travaux continuent cependant à être menés, notamment (mais pas uniquement) sur des améliorations de la scalabilité du protocole [GDEB99, MH99]. Nous envisageons d'approfondir l'étude menée sur la scalabilité de RSVP et notamment l'effet des améliorations proposées.

Dans le cinquième chapitre, nous pensons avoir montré que l'approche proposée est utilisable. Nous n'avons cependant pas réalisé d'étude approfondie des phénomènes entrant en jeu dans la déformation du trafic et sur les effets que cette déformation peut avoir sur les délais garantis aux flots. Une telle étude permettrait de caractériser plus systématiquement les conditions dans lesquelles la solution est valable. Dans la même optique, un autre travail intéressant serait de déterminer les performances d'un ordonnanceur VC qui ne maintiendrait d'états que pour une petite portion des flots. Nous avons montré que les flots pouvaient être plus ou moins sensibles à l'effet d'agglomération, de la même manière, il est probable que l'effet des flots déformés sur la déviation du service dans l'ordonnanceur dépende des caractéristiques de ces flots. Maintenir un état, et donc empêcher la formation de grumeaux, pour une petite quantité de flots les plus “malfaisants” pourrait permettre une très bonne approximation du VC.

Aucune des deux solutions que nous avons proposées pour le problème de scalabilité ne peut traiter facilement le cas du multicast. C'est pourtant un problème important car les applications multicast sont appelées à se développer. Les techniques liées au multicast n'étant cependant pas encore matures, la gestion de la qualité de service devrait être prise en compte dans leur définition.

Publications

[1] O. Fourmaux & C. Deleuze. "Qualité de service hétérogène et dynamique pour liaisons ATM point-à-multipoint". *Actes de la Conférence Francophone sur l'Ingénierie des Protocoles (CFIP'97)*, Liège, Belgique, septembre 1997.

[2] Simon-Pietro Romano, Christophe Deleuze, José Rezende & Serge Fdida. "Integrated QoS architecture for IP Switching". 3^{rd} *European Conference on Multimedia Applications, Services and Techniques*, Berlin, mai 1998. Publié dans *Lecture Notes in Computer Science* n. 1425, pp. 312–325.

[3] O. Fourmaux, V. Roca, C. Deleuze & S. Fdida. "Architecture réseau pour le projet DIS/ATM : RSVP Switching, une solution pour un service multipoint avec QoS en environement IPv6 sur ATM". *Rapport pour le projet DIS/ATM* avec le LAAS, l'INRIA et Dassault Électronique, septembre 1998.

[4] C. Deleuze. "Scheduling". À paraître dans le *Rapport COST 237*, A. Danthine éditeur, Springer-Verlag, 2000.

[5] C. Deleuze. "Managing best-effort traffic: RED". À paraître dans le *Rapport COST 237*, A. Danthine éditeur, Springer-Verlag, 2000.

[6] C. Deleuze & S. Fdida. "Agrégation de flots RSVP pour une architecture IntServ extensible". *Actes du Congrès "De Nouvelles Architectures pour les Communications" (DNAC'99)*, Paris, France, décembre 1999.

[7] C. Deleuze & S. Fdida. "A scalable IntServ architecture through RSVP aggregation". Accepté pour publication dans *Networking and Information Systems Journal*.

Bibliographie

[Alm96] Werner Almesberger. *ATM on Linux.* Rapport technique, CH-1015 Lausanne, Switzerland, Laboratoire de Réseaux de Communication (LRC), École Polytechnique Fédérale de Lausanne (EPFL), mars 1996.

[Arm96] G. Armitage. *Support for Multicast over UNI 3.0/3.1 based ATM Networks.* RFC 2022, IETF, novembre 1996.

[Atk94] R. Atkinson. *Default IP MTU for use over ATM AAL5.* RFC 1626, IETF, mai 1994.

[B+98] S. Blake et al. *An Architecture for Differentiated Services.* RFC 2475, IETF, décembre 1998.

[BBB+99] Yoram Bernet, James Binder, Steven Blake, Mark Carlson, Brian E. Carpenter, Srinivasan Keshav, Elwyn Davies, Borje Ohlman, Dinesh Verma, Zheng Wang et Walter Weiss. *A Framework for Differentiated Services.* Internet draft, IETF, février 1999. `draft-ietf-diffserv-framework-02.txt`.

[BCS94] R. Braden, D. Clark et D. Shenker. *Integrated Services in the Internet Architecture: an Overview.* RFC 1633, IETF, juillet 1994.

[BH97] R. Braden et D. Hoffman. *RAPI Version 5.* Internet Draft, IETF, septembre 1997. `draft-ietf-rsvp-rapi-01.txt`.

[BLT99] Fred Baker, Bob Lindell et Mohit Talwar. *RSVP Cryptographic Authentication.* Internet Draft, IETF, février 1999. `draft-ietf-rsvp-md5-08.txt`.

[BSB99] Yoram Bernet, A. Smith et Steven Blake. *A Conceptual Model for Diffserv Routers.* Internet draft, IETF, juin 1999. `draft-ietf-diffserv-model-00.txt`.

[BYF+99] Y. Bernet, R. Yavatkar, P. Ford, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, John Wroclawski et E. Felstaine. *A Framework For Integrated Services Operation Over DiffServ Networks.* Internet draft, IETF, septembre 1999. `draft-ietf-issll-diffserv-rsvp-03.txt`.

[BZ96a] Jon C.R. Bennett et Hui Zhang. Hierarchical packet fair queueing algorithms. In *Proceedings of SIGCOMM'96* [SIG96], pp. 143–156. Aussi dans *IEEE/ACM Transactions on Networking* oct 97. v5n5.

[BZ96b] Jon C.R. Bennett et Hui Zhang. WF^2Q: Worst-case fair weighted fair queueing. In *Proceedings of INFOCOM'96* [INF96], pp. 1d.3.1–1d.3.9.

[BZ97] R. Braden et L. Zhang. *Resource ReServation Protocol (RSVP) – Version 1 Message Processing Rules*. RFC 2209, IETF, septembre 1997.

[BZB+97] R. Braden, L. Zhang, S. Berson, S. Herzog et S. Jasmin. *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*. RFC 2205, IETF, septembre 1997.

[C+99] R. Callon et al. *A Framework for Multiprotocol Label Switching*. Internet draft, IETF, juin 1999. `draft-ietf-mpls-framework-03.txt`.

[CDD+99] Christophe Chassot, Laurent Dairaine, Michel Diaz, Fabien Garcia et André Lozes. *Étude et évaluation des mécanismes de contrôle de ressources en réseau hétérogène avec application au DIS*. Rapport d'expérimentations, LAAS/CNRS, avril 1999.

[Cru96] R. L. Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE Journal on Selected Areas in Communications*, vol. 13, n 6, août 1996, pp. 1048–1056.

[CSZ92] David Clark, Scott Shenker et Lixia Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanisms. *In : Proceedings of SIGCOMM'92*. Baltimore, Maryland, USA, août 1992.

[Del00] Christophe Deleuze. *Rapport final COST237*, chap. Managing best-effort traffic: RED. À paraitre chez Springer-Verlag, 2000.

[Die98] Thomas Dietz. *Label Swapping with RSVP in an IP over ATM Environnement*. Diplomarbeit am Institut für Telematik, Fakultät für Informatik - Universität Karlsruhe, Allemagne, mai 1998.

[DKS89] Alan Demers, Srinivasan Keshav et Scott Shenker. Analysis and simulation of a fair queueing algorithm. *In : Proceedings of SIGCOMM'89*, pp. 1–12. Austin, Texas, septembre 1989.

[FALB98] Tiziana Ferrari, Werner Almesberger et Jean-Yves Le Boudec. SRP: A scalable resource reservation protocol for the Internet. *In : Proceedings of the sixth IEEE/IFIP International Workshop on Quality of Service (IWQoS'98)*, pp. 107–116. Napa, CA, USA, mai 1998.

[FE99] Kevin Fall et Kannan Varadhan Editors. *ns Notes and Documents*. Rapport technique, LBNL, UC Berkeley, septembre 1999. `http://www-mash.cs.berkeley.edu/ns/ns-documentation.html`.

[FF97] Sally Floyd et Kevin Fall. *Router Mechanisms to Support End-to-End Congestion Control*. Rapport technique, Berkeley, CA, Lawrence Berkeley Laboratory, février 1997.

[FF99] Olivier Fourmaux et Serge Fdida. Multicast for RSVP switching – an extended multicast model with QoS for label swapping in an IP over ATM environment. *Telecommunication Systems Journal*, no11, 1999, pp. 85–104.

[FJ93] Sally Floyd et Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, août 1993.

[FJ95] Sally Floyd et Van Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, vol. 3, n4, août 1995, pp. 365–386.

[FM96] Norman Finn et Tony Mason. ATM LAN Emulation. *IEEE Communications Magazine*, vol. 34, n6, juin 1996, pp. 96–100.

[FP95] Norival R. Figueira et Joseph Pasquale. An upper bound on delay for the VirtualClock service discipline. *IEEE/ACM Transactions on Networking*, vol. 3, n 4, août 1995, pp. 399–408.

[FRDF98] Olivier Fourmaux, Vincent Roca, Christophe Deleuze et Serge Fdida. *Architecture réseau pour le projet DIS/ATM : RSVP Switching, une solution pour un service multipoint avec QoS en environnement IPv6 sur ATM.* Rapport technique, LIP6, septembre 1998.

[FV90] Domenico Ferrari et Dinesh Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, vol. 8, n3, avril 1990, pp. 368–379.

[G+96] L. Georgiadis et al. Efficient support of delay and rate guarantees in an internet. In *Proceedings of SIGCOMM'96* [SIG96].

[GBH97] R. Guérin, S. Blake et S. Herzog. *Aggregating RSVP-based QoS Requests.* Internet draft, IETF, novembre 1997. `draft-guerin-aggreg-rsvp-00.txt`.

[GDEB99] Silvano Gai, Dinesh Dutt, Nitsan Elfassy et Yoram Bernet. *RSVP+: An Extension to RSVP.* Rapport technique, IETF, juin 1999.

[Gol94] S. Jamaloddin Golestani. A self-clocked fair queueing scheme for broadband applications. *In : Proceedings of INFOCOM'94*, pp. 636–646. Toronto, Canada, juin 1994.

[Gué98] Roch Guérin. Quality of service: Too little or too much? how much optimization do we really need? *In : Proceedings of the sixth IEEE/IFIP International Workshop on Quality of Service (IWQoS'98).* Napa, CA, USA, mai 1998.

[GVC96] Pawan Goyal, Harrick M. Vin et Haichen Cheng. Start-time Fair Queueing: A scheduling algorithm for integrated services packet switching networks. In *Proceedings of SIGCOMM'96* [SIG96], pp. 157–168. Aussi dans *IEEE/ACM Transactions on Networking* oct 97 v5n5.

[H+99] Juha Heinanen et al. *Assured Forwarding PHB Group.* RFC 2597, IETF, juin 1999.

[HG99a] Juha Heinanen et Roch Guérin. *A Single Rate Three Color Marker*. RFC 2697, IETF, septembre 1999.

[HG99b] Juha Heinanen et Roch Guérin. *A Two Rate Three Color Marker*. RFC 2698, IETF, septembre 1999.

[INF96] *IEEE INFOCOM'96*. San Francisco, California, mars 1996.

[J$^+$99] V. Jacobson et al. *An Expedited Forwarding PHB*. RFC 2598, IETF, juin 1999.

[Kes97] Srinivasan Keshav. *An Engineering Approach to Computer Networking*. Addison-Wesley, 1997.

[Lau93] M. Laubach. *Classical IP and ARP over ATM*. RFC 1577, IETF, décembre 1993.

[LB96] Jean-Yves Le Boudec. *Network Calculus Made Easy*. Rapport technique EPFL-DI 96/128, Laboratoire de Réseaux de Communication, École Polytechnique Fédérale de Lausanne, décembre 1996.

[LH98] M. Laubach et J. Halpern. *Classical IP and ARP over ATM*. RFC 2225, IETF, avril 1998. Remplace les RFC 1577 et 1626.

[LM97] Steven Lin et Nick McKeown. A simulation study of IP switching. *In : Proceedings of SIGCOMM'97*.

[M$^+$97] A. Mankin et al. *Resource ReServation Protocol (RSVP) Version 1 Applicability Statement Some Guidelines on Deployment*. RFC 2208, IETF, septembre 1997.

[MF98] Yan Moret et Serge Fdida. A proportional queue control mechanism to provide differentiated services. *In : International Symposium on Computer System*. Belek, Turkey, octobre 1998.

[MH99] Laurent Mathy et David Hutchison. Étude de l'établissement des flux dans RSVP. *In : Actes du 7e Colloque Francophone sur l'Ingénierie des Protocoles (CFIP'99)*, éd. par A. Schaff, F. Lepage et J.P. Thomesse. Nancy, France, avril 1999.

[Mil92] David L. Mills. *Netwok Time Protocol (Version 3) : Specification, Implementation and Analysis*. RFC 1305, IETF, mars 1992.

[MPO97] *MultiProtocol Over ATM – Version 1.0*. Rapport technique af-mpoa-00087.000, ATM Forum, juillet 1997.

[N$^+$96a] Peter Newman et al. *Ipsilon Flow Management Protocol Specification for IPv4*. RFC 1953, IETF, mai 1996.

[N$^+$96b] Peter Newman et al. *Ipsilon's General Switch Management Protocol Specification – Version 1.1*. RFC 1987, IETF, août 1996.

[N$^+$96c] Peter Newman et al. *Transmission of Flow Labelled IPv4 on ATM Data Links*. RFC 1954, IETF, mai 1996.

[N+98] Peter Newman et al. *Ipsilon's General Switch Management Protocol Specification – Version 2.* RFC 2297, IETF, mars 1998.

[Nag87] John B. Nagle. On packet switches with infinite storage. *IEEE Transactions on Communications*, vol. COM-35, n4, avril 1987, pp. 435–438.

[NBBB98] K. Nichols, S. Blake, F. Baker et D. Black. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers.* RFC 2474, IETF, décembre 1998.

[NLM96] Peter Newman, Tom Lyon et Greg Minshall. Flow labelled IP: A connectionless approach to ATM. In *Proceedings of INFOCOM'96* [INF96].

[NML] Peter Newman, Greg Minshall et Tom Lyon. IP switching: ATM under IP. *IEEE/ACM Transactions on Networking.*

[NMLH97] Peter Newman, Greg Minshall, Tom Lyon et Larry Huston. IP switching and gigabit routers. *IEEE Communications Magazine*, janvier 1997.

[PG93] A. K. Parekh et R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks – the single node case. *IEEE/ACM Transactions on Networking*, vol. 1, n3, juin 1993, pp. 344–357.

[RG97] Sanjeev Rampal et Roch Guérin. *Flow Grouping for Reducing Reservervation Requirements for Guaranteed Delay Service.* Internet draft, IETF, juillet 1997. `draft-rampal-flow-delay-service-01.txt`.

[RGB96] Jennifer L. Rexford, Albert G. Greeberg et Flavio G. Bonomi. Hardware-efficient fair queuing architectures for high-speed networks. In *Proceedings of INFOCOM'96* [INF96], pp. 5d.2.1–5d.2.9.

[SIG96] *SIGCOMM '96 SYMPOSIUM Communications Architectures & Protocols.* 1996.

[SKWS98] Jens Schmitt, Martin Karsten, Lars Wolf et Ralf Steinmetz. *Aggregation of Guaranteed Service Flows.* Rapport technique TR-KOM-1998-06, Darmstadt University of Technology, novembre 1998.

[SPG97] S. Shenker, C. Partridge et R. Guérin. *Specification of Guaranteed Quality of Service.* RFC 2212, IETF, septembre 1997.

[Sta98] Richard M. Stallman. *Linux and the GNU Project.* Texte, Free Software Foundation, 1997,1998. `http://www.gnu.org/gnu/linux-and-gnu.html`.

[SW97] S. Shenker et J. Wroclawski. *General Characterization Parameters for Integrated Services Network Elements.* RFC 2215, IETF, septembre 1997.

[SZ98] Ion Stoica et Hui Zhang. *Providing Guaranteed Services Without Per Flow Management.* Rapport technique CMU-CS-99-133, School of Computer Science, Carnegie Mellon University, mai 1998.

[SZ99] Ion Stoica et Hui Zhang. Providing guaranteed services without per flow management. *In: Proceedings of SIGCOMM'99*, pp. 81–94. Boston, MA, septembre 1999.

[TA97] R. Talpade et M. Ammar. *Multicast Server Architectures for MARS-based ATM multicasting.* RFC 2149, IETF, mai 1997.

[VS97] Anujan Varma et Dimitrios Stiliadis. Hardware implementations of fair queuing algorithms for asynchronous transfer mode networks. *IEEE Communications Magazine*, vol. 35, n12, décembre 1997, pp. 54–68.

[Wis98] Markus A. Wischy. *Virtual Clock Implementation for NS.* Rapport technique, University of Oregon, 1998. `http://www.cs.uoregon.edu/~wischy/projects/virtualclock/index.html`.

[Wro97a] J. Wroclawski. *Specification of the Controlled-Load Network Element Service.* RFC 2211, IETF, septembre 1997.

[Wro97b] J. Wroclawski. *The Use of RSVP with IETF Integrated Services.* RFC 2210, IETF, septembre 1997.

[ZDE+93] Lixia Zhang, Stephen Deering, Deborah Estrin, Scott Shenker et Daniel Zappala. RSVP: A new Resource reSerVation Protocol. *IEEE Network Magazine*, septembre 1993, pp. 8–18.

[Zha90] Lixia Zhang. Virtual Clock: A new traffic control algorithm for packet switching networks. *In: Proceedings of ACM SIGCOMM'90*, pp. 19–29. Philadelphia, Pennsylvania, septembre 1990.

Liste des acronymes

AF	*Assured Forwarding*
API	*Application Programming Interface*
ATM	*Asynchronous Transfer Mode*
BE	*Best Effort*
BR	*Bandwidth Recovery*
CAC	*Call Admission Control*
CBR	*Constant Bit Rate*
CL	*Controlled-Load*
CLIP	*CLassical IP over ATM*
CNRS	Centre National de la Recherche Scientifique
CPU	*Central Processing Unit*
DPS	*Dynamic Packet State*
DS	*Differentiated Services*
DIS	*Distributed Interactive Simulation*
EDD	*Earliest Due Date*
EF	*Expedited Forwarding*
EPFL	École Polytechnique Fédérale de Lausane
FCFS	*First Come, First Served*
FQ	*Fair Queuing*
GNU	*GNU's Not Unix*
GPL	*GNU General Public Licence*
GPS	*Generalized Processor Sharing*
GS	*Guaranteed Service*
GSMP	*General Switch Management Protocol*
ID	*Internet Draft*
IETF	*Internet Enginneering Task Force*
IFMP	*Ipsilon Flow Management Protocol*
IIS	*Internet Integrated Services*

IISSL	*Internet Integrated Services over Specific Layer*
INRIA	Institut National de Recherche en Informatique et en Automatique
IP	*Internet Protocol*
IRTF	*Internet Research Task Force*
IS	*Integrated Services*
ISDN	*Integrated Services Digital Network*
ISI	*Information Sciences Institute*
LAAS	*Laboratoire d'Analyse et d'Architecture des Systèmes*
LAN	*Local Area Network*
LANE	*LAN Emulation*
LIP6	Laboratoire d'Informatique de Paris VI
MAN	*Metropolitan Area Network*
MARS	*Multicast Address Resolution Server*
MPOA	*MultiProtocol Over ATM*
OPWA	*One Pass With Advertising*
PGPS	*Packet-by-packet Generalized Processor Sharing*
PHB	*Per Hop Behavior*
PHOP	*Previous HOP*
PS	*Path State*
PSB	*Path State Block*
QoS	*Quality of Service*
RAP	*Ressource Allocation Protocol* (Groupe de travail de l'IETF, anciennement *RSVP Admission Policy*)
RAPI	*RSVP Application Programming Interface*
RCS	*Rate Controlled Service*
RED	*Random Early Detection*
RFC	*Request For Comment*
RS	*Resv State*
RSB	*Resv State Block*
RSVP	*Resource ReserVation setup Protocol*
SRP	*Scalable resource Reservation Protocol*
SCFQ	*Self-Clocked Fair Queueing*
SFQ	*Start-time Fair Queueing*
TCP	*Transmission Control Protocol*
TCS	*Traffic Control State*
TCSB	*Traffic Control State Block*
TDM	*Time Division Multiplexing*

TOS	*Type Of Service*
TSPEC	*Traffic SPECification*
UDP	*User Datagram Protocol*
VC	*Virtual Clock*
VCI	*Virtual Channel Identifier*
VPI	*Virtual Path Identifier*
WAN	*Wide Area Network*
WFQ	*Weighted Fair Queuing*
WF^2Q	*Worst-case-Fair Weighted Fair Queuing*
WRR	*Weighted Round Robin*
YAAAAA	*Yet Another Annoying And Ambiguous Acronym*

Liste des figures

Liste des tableaux

Ce document a été rédigé et composé à l'aide de logiciels libres sur un système d'exploitation libre *GNU/Linux*. L'éditeur de texte utilisé est *GNU Emacs* et le système de formattage est LaTeX (classe de document *book* modifiée). Les schémas ont été réalisés avec le logiciel de dessin vectoriel *xfig* puis convertis au format *postscript encapsulé*. Les courbes ont été tracées à l'aide du logiciel *gnuplot*.
Tous ces logiciels utilisent des formats de données publics qui garantissent la pérennité et la convertibilité des documents, au contraire des formats secrets *propriétaires* qui rendent l'utilisateur dépendant d'un logiciel et d'un éditeur spécifique.

Pour plus d'information à ce sujet, voir par exemple les documents suivants :

- Richard M. Stallman, *What's GNU? Gnu's not Unix! – The GNU Manifesto*, Free Software Foundation, 1985, 1993, `http://www.gnu.org/gnu/manifesto.html`.
- Richard M. Stallman, The Right to Read, *Communications of the ACM*, volume 40 n. 2, février 1997.
- Roberto Di Cosmo, *Piège dans le cyberespace*, Multimédium, 1997, `http://www.mmedium.com/dossiers/piege/`.

ainsi que les sites de la *Free Software Foundation* (`http://www.fsf.org`) et de l'APRIL (Association pour la promotion et la recherche en informatique libre, `http://www.april.org`).

Résumé Ce travail de thèse est centré sur la qualité de service dans les réseaux à commutation de paquets de grande dimension, tels l'Internet. Une architecture à intégration de services a été définie par l'IETF, elle utilise le protocole RSVP et est appelée IntServ/RSVP. Cependant des problèmes subsistent, en particulier dans le contexte des réseaux d'interconnexion où apparaissent des difficultés liées d'une part au haut débit, d'autre part au grand nombre de flots.
Ces dernières années, l'évolution des réseaux haut débit s'est portée en grande partie sur les solutions de type "commutation IP" qui permettent d'associer la flexibilité et la robustesse d'IP aux capacités haut débit d'un moteur de commutation ATM. Cette approche a été généralisée sous le nom de "commutation par labels". Dans une première partie, nous proposons "RSVP switching", une solution intégrant l'architecture IntServ/RSVP à un environnement de type "commutation IP". Si elle permet de combiner gestion de la qualité de service et haut débit, cette solution souffre cependant des problèmes de sensibilité au facteur d'échelle de l'architecture IntServ/RSVP qui requiert le maintien d'états par flot.
Dans une deuxième partie, nous étudions les techniques d'agrégation comme moyens de fournir des garanties de service sans gérer d'état pour chaque flot. Nous montrons dans quelles conditions la gestion d'états par agrégat de flots est suffisante et proposons une architecture mettant en œuvre ces techniques d'agrégation dans les réseaux d'interconnexion et s'interfaçant de manière transparente avec le modèle classique IntServ/RSVP utilisé dans les réseaux périphériques.
Le changement de granularité correspondant à un modèle utilisant des agrégations de flots se traduit par une perte de flexibilité dans la gestion des ressources. Dans une troisième partie, nous étudions une approche permettant de conserver une granularité au niveau flot, sans maintenir d'état par flot dans les nœuds. Pour cela le trafic est conditionné en entrée de la *zone sans état*, et des précautions sont prises pour limiter l'impact des nœuds sur l'allure du trafic.

Mots clés QoS, qualité de service, facteur d'échelle, haut débit, agrégation, commutation IP, *weighted fair queuing*, *virtual clock*

Quality of Service in the Internet:
Issues Related to High Bandwidth and Scalability

Abstract is on page 9

Keywords QoS, quality of service, scalability, high rates, aggregation, IP switching, weighted fair queuing, virtual clock