



La qualité de service dans les réseaux

Étude de Differentiated Service

Baptiste MATHUS
Vincent ROBERT



Table des matières

1	Sujet.....	3
2	Introduction.....	4
3	Environnement.....	5
3.1	Architecture testée.....	5
4	La simulation.....	7
4.1	Mise en place de l'architecture commune.....	7
4.1.1	createNodes.....	7
4.1.2	createLayer4.....	9
4.1.3	createLayer5.....	10
4.1.4	scheduling.....	10
4.2	Routage sans Diffserv.....	11
4.3	Routage avec DS.....	12
4.3.1	Principe de Diffserv.....	13
4.3.2	La création des liens.....	13
4.3.3	Les deux type de files.....	14
4.3.4	Fonction de création d'un lien « Edge ».....	15
4.3.5	Fonction de création d'un lien « Core ».....	17
4.3.6	Utilisation des fonctions et mise en place de la QoS par DiffServ... ..	18
5	Analyse des résultats.....	19
5.1	Parsing.....	19
5.2	Chiffres.....	21
5.3	Graphiques.....	21
5.3.1	Les drops de VoIP.....	21
5.3.2	Les drops des autres flux.....	22
6	Conclusion.....	24



1 Sujet

Vous travaillez pour un provider qui souhaite que son cœur réseau (core network) IP puisse faire de la QoS (Quality of Service). Le provider pour lequel vous travaillez possède une infrastructure réseau de taille moyenne centrée sur la France, il offre des accès réseaux à de multiples entreprises de tailles moyennes à grandes (des T1 en règle générale, 1.544 Mbits/s). Il possède 3 accès vers le cœur du réseau Internet, un accès vers le réseau de France Télécom OpenTransit, un vers le backbone de l'opérateur Sprint et un accès vers le point d'échange français SFINX (peering à définir).

Pour cela, votre employeur vous charge de mener à bien une étude de faisabilité sur les différentes technologies permettant de faire de la QoS. Mais votre employeur qui s'est beaucoup documenté (dans O1Informatique) reste très focalisé sur la technologie DiffServ. Essayez dans votre étude de trouver des arguments techniques qui pourront affirmer ou infirmer les idées de votre employeur.

2 Introduction

Les providers¹ doivent aujourd'hui répondre à une nouvelle demande. Celle-ci consiste à pouvoir favoriser un ou plusieurs types de trafics parmi d'autres. Pouvoir gérer une priorisation des flux s'appelle de la qualité de service (QoS²).

Le problème est que le protocole IP est basé sur le principe de Best Effort : on fait de son mieux, mais il n'y aucune garantie qu'un paquet envoyé arrivera à bon port sans avoir été corrompu ou tout simplement *droppé*. Lorsqu'on souhaite qu'un paquet soit transmis de façon sûre, il faut utiliser une couche transport adéquate comme TCP. Le protocole IP n'inclut de plus aucun mécanisme de priorisation d'un type de trafic par rapport à un autre.

Lorsqu'on souhaite télécharger un document sur un serveur Web, c'est à dire un serveur HTTP, il n'est pas grave de risquer de perdre des données : comme HTTP fonctionne sur TCP, la couche transport s'occupera de gérer les retransmissions de paquets si besoin est. Par contre, lors d'une discussion sur VoIP, protocole fonctionnant sur UDP³, il paraît important de tenter de transmettre de façon plus prioritaire ce trafic par rapport aux autres flux. En effet, si des datagrammes UDP transportant du VoIP sont perdus, au mieux on aura raté un mot, au pire, on n'aura rien compris au message de l'émetteur.

Or, comme indiqué ci-dessus, IP n'intègre aucune gestion de la priorité. Il est donc impossible de l'utiliser tel quel pour envisager une quelconque priorisation des types de flux.

DiffServ⁴ est une des solutions existantes pour mettre en place un mécanisme de QoS au sein d'un réseau. Cette étude va donc tenter d'apporter des éléments de réponse à l'utilisabilité de cette technologie pour nos besoins. Cette étude sera réalisée avec le logiciel *Network Simulator*, nous simulerons notre réseau avec et sans utilisation de DiffServ afin d'observer les différences de comportement.

Dans un premier temps, nous détaillerons l'environnement et l'architecture dans lesquels nous avons effectué les tests.

Au vu de la complexité de celui-ci pour la mise en place de DiffServ, nous expliquerons ensuite le code nécessaire à Network Simulator pour cette technologie.

Nous terminerons par une analyse des résultats qui nous permettra de tirer une conclusion de l'étude.

1 Fournisseurs d'accès.

2 Quality of Service.

3 UDP est un protocole non orienté connexion : il ne gère aucunement les retransmissions. Si un paquet est perdu, c'est l'application qui doit gérer la retransmission (contrairement à TCP).

4 Differentiated Service.

3 Environnement

L'étude porte sur le réseau central de notre entreprise. Il s'agit donc de définir si la technologie de QoS DiffServ est intéressante pour notre architecture.

3.1 Architecture testée

Notre réseau est constitué de 2 points d'accès : un premier destiné aux connexions de nos clients, et un second destiné à notre connexion à Internet. Notre connexion Internet passe par 3 grands réseaux : OpenTransit, SPRINT et SFINX. Ces 3 réseaux sont utilisés équitablement par nos clients.

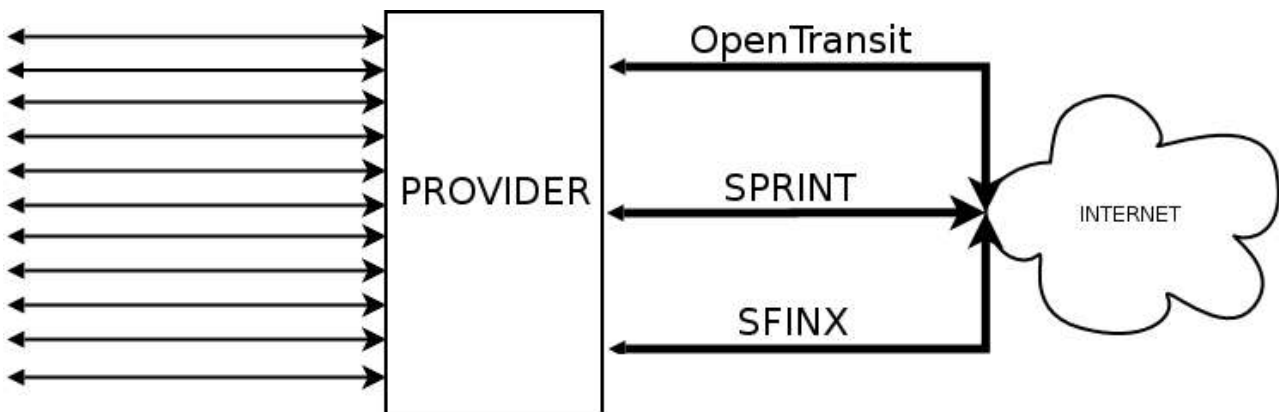


Illustration 1 - Schéma simplifié de l'architecture de notre réseau

Pour cette étude, nous avons décidé de simuler 3 types de trafic pour nos clients. Il était important pour l'étude de choisir des trafics de natures différentes dont le transport ne doit pas être traité avec la même priorité. Nous avons donc choisi des trafics FTP, HTTP et VoIP.

Nous avons symbolisé les débits de tous les clients utilisant un même protocole par un seul nœud. Ainsi, du côté de nos clients, nous avons 3 « clients », un client FTP, un client HTTP et un client VoIP. Du côté Internet, nous avons aussi 3 « serveurs » par réseau auquel nous sommes connectés, soit 9 serveurs différents, 3 par types de trafic.

Les serveurs FTP génèrent un trafic de type FTP vers les clients avec un débit d'1Mb/s chacun. Les serveurs HTTP génèrent un trafic exponentiel vers les clients avec un débit d'1Mb/s par serveur. Les serveurs et les clients VoIP génèrent pour chaque connexion client-serveur un trafic constant d'1.2Mb/s en débit montant et descendant pour chacune des 3 connexions.

On peut trouver sur la page suivante l'architecture utilisée pour l'étude.

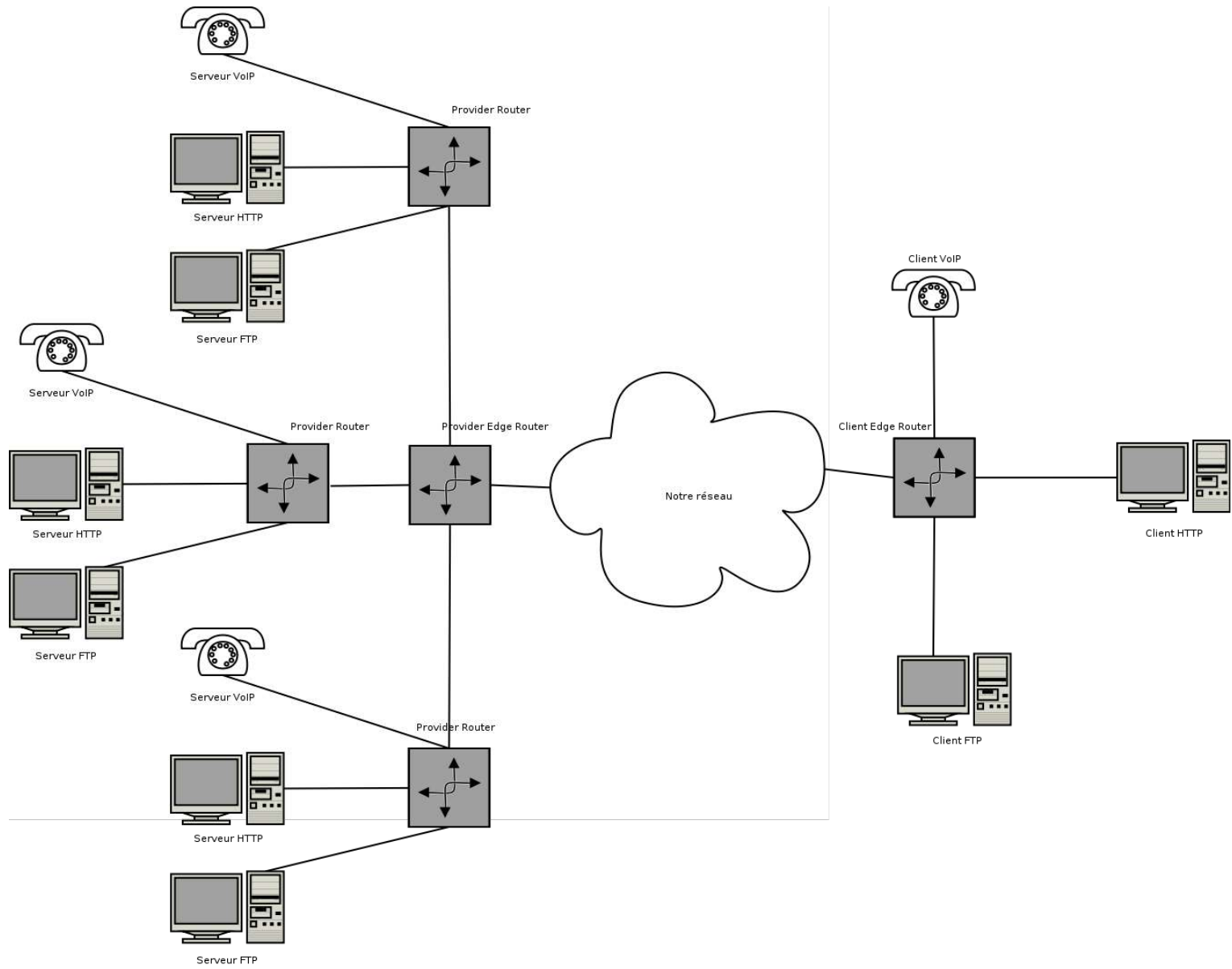


Illustration 2 - Architecture utilisée pour la simulation

4 La simulation

Nous présentons dans cette partie le code nécessaire à la simulation avec *Network Simulator*. Ceci nous semble important au vu de la complexité du code de mise en place de DiffServ.

L'objectif étant évidemment de pouvoir ensuite comparer les résultats avec et sans DiffServ, nous avons décidé de mettre en place exactement la même architecture « matérielle » dans les deux cas.

La seule différence résidera en effet dans la configuration de DiffServ sur les liens concernés pour tester l'influence de la technologie.

4.1 Mise en place de l'architecture commune

Afin de clarifier le code et d'en factoriser la partie commune, nous avons beaucoup utilisé la méthode *source* qui permet d'inclure le code d'un fichier tcl dans un autre fichier tcl.

Le tableau suivant présente la structure du code tel qu'il est écrit à la fois pour la simulation sans DiffServ et celle avec. La colonne centrale définit l'action effectuée et la colonne de droite indique le nom du fichier dans lequel se trouve le code, le cas échéant.

1	Création de l'instance de ns et définition de la procédure finish()	
2	Création des nœuds	createNodes.tcl
3	Création des liens	Spécifique
4	Création de la couche 4 (TCP&UDP)	createLayer4.tcl
5	Création de la couche 5 (applications)	createLayer5.tcl
6	Règles DiffServ	Spécifique
7	Scheduling : temps de démarrage/arrêt des services	scheduling.tcl
8	\$ns run	

Les cellules grisées indiquent les parties du code qui diffèrent entre la version avec ou sans DiffServ. Celles-ci seront détaillées dans leurs parties respectives (4.2 et 4.3).

4.1.1 createNodes

Nous avons fortement utilisé les couleurs et les labels afin de différencier mieux et plus rapidement les nœuds du réseau.



Tous les noms utilisés sont volontairement très explicites pour faciliter la lecture du code.

```
#####  
# CRÉATION DES NOEUDS DU RÉSEAU #  
#####  
  
array set colors { 0 blue 1 cyan 2 red 3 magenta 4 green 5 lightgreen }  
  
#Les clients de notre réseau  
set nodeClientVoIP [$ns node]  
set nodeClientHTTP [$ns node]  
set nodeClientFTP [$ns node]  
$nodeClientVoIP label "Client VoIP"  
$nodeClientVoIP label-color $colors(0)  
$nodeClientHTTP label "Client HTTP"  
$nodeClientHTTP label-color $colors(2)  
$nodeClientFTP label "Client FTP"  
$nodeClientFTP label-color $colors(4)  
  
#Notre réseau  
set nodeClientEdgeRouter [$ns node]  
set nodeProviderEdgeRouter [$ns node]  
  
$nodeClientEdgeRouter label "ClientEdge"  
$nodeProviderEdgeRouter label "ProviderEdge"  
  
set providerNb 3  
array set providerName { 1 "SPRINT" 2 "SFINX" 3 "OPENTRANSIT" 4 "FreeIX" 5 "PariX" 6  
"OVH" 7 "MFN" }
```

On n'utilise que les 3 premiers providers pour mieux correspondre au sujet proposé. Le code a toutefois été écrit pour pouvoir simplement changer le nombre de providers.

```
for {set i 1} {$i <= $providerNb } {incr i} {  
  #  
  set nodeProviderRouter($i) [$ns node]  
  set nodeServerVoIP($i) [$ns node]  
  set nodeServerHTTP($i) [$ns node]  
  set nodeServerFTP($i) [$ns node]  
  
  $nodeProviderRouter($i) label "$providerName($i)"  
  
  $nodeServerVoIP($i) label "VoIP"
```

```
$nodeServerVoIP($i) label-color $colors(1)

$nodeServerHTTP($i) label "HTTP"
$nodeServerHTTP($i) label-color $colors(3)

$nodeServerFTP($i) label "FTP"
$nodeServerFTP($i) label-color $colors(5)
}
```

4.1.2 createLayer4

```
#On met des couleurs sur les clients
$ns color 0 $colors(0)
$ns color 1 $colors(1)
$ns color 2 $colors(2)
$ns color 3 $colors(3)
$ns color 4 $colors(4)
$ns color 5 $colors(5)

for {set i 1} {$i <= $providerNb} {incr i} {
  #VoIP Sens Client -> internet
  set udpVoIPClientToInternet($i) [new Agent/UDP]
  set udpVoIPClientToInternetClient($i) [new Agent/Null]
  $ns attach-agent $nodeClientVoIP $udpVoIPClientToInternet($i)
  $ns attach-agent $nodeServerVoIP($i) $udpVoIPClientToInternetClient($i)
  $ns connect $udpVoIPClientToInternet($i) $udpVoIPClientToInternetClient($i)
  $udpVoIPClientToInternetClient($i) set fid_ 0

  #VoIP Sens Internet -> Client
  set udpVoIPInternetToClient($i) [new Agent/UDP]
  set udpVoIPInternetToClientClient($i) [new Agent/Null]
  $ns attach-agent $nodeClientVoIP $udpVoIPInternetToClientClient($i)
  $ns attach-agent $nodeServerVoIP($i) $udpVoIPInternetToClient($i)
  $ns connect $udpVoIPInternetToClientClient($i) $udpVoIPInternetToClient($i)
  $udpVoIPInternetToClient($i) set fid_ 1

  #HTTP
  set tcpHTTPClient($i) [new Agent/TCPSink]
  set tcpHTTPServer($i) [new Agent/TCP]
  $ns attach-agent $nodeClientHTTP $tcpHTTPClient($i)
  $ns attach-agent $nodeServerHTTP($i) $tcpHTTPServer($i)
  $ns connect $tcpHTTPClient($i) $tcpHTTPServer($i)
  $tcpHTTPServer($i) set fid_ 2
}
```

```
#FTP
set tcpFTPClient($i) [new Agent/TCPSink]
set tcpFTPServer($i) [new Agent/TCP]
$ns attach-agent $nodeClientFTP $tcpFTPClient($i)
$ns attach-agent $nodeServerFTP($i) $tcpFTPServer($i)
$ns connect $tcpFTPClient($i) $tcpFTPServer($i)
$tcpFTPServer($i) set fid_ 4
}
```

4.1.3 createLayer5

```
for {set i 1} {$i <=$providerNb} {incr i} {
  #VoIP : CBR
  #sens client->internet
  set appVoIPClientToInternet($i) [new Application/Traffic/CBR]
  $appVoIPClientToInternet($i) attach-agent $udpVoIPClientToInternet($i)
  $appVoIPClientToInternet($i) set rate_ 1.2Mb

  #sens Internet->client
  set appVoIPInternetToClient($i) [new Application/Traffic/CBR]
  $appVoIPInternetToClient($i) attach-agent $udpVoIPInternetToClient($i)
  $appVoIPInternetToClient($i) set rate_ 1.2Mb

  #HTTP
  set appHTTPServer($i) [new Application/Traffic/Exponential]
  $appHTTPServer($i) attach-agent $tcpHTTPServer($i)
  $appHTTPServer($i) set rate_ 1Mb

  #FTP
  set appFTPServer($i) [new Application/FTP]
  $appFTPServer($i) attach-agent $tcpFTPServer($i)
  $appFTPServer($i) set rate_ 1Mb
}
```

4.1.4 scheduling

```
for {set i 1} {$i<=$providerNb} {incr i} {
  #démarrages
  $ns at 0.5 "$appVoIPClientToInternet($i) start"
  $ns at 0.5 "$appVoIPInternetToClient($i) start"
```

```

$ns at 0.5 "$appHTTPServer($i) start"
$ns at 0.5 "$appFTPServer($i) start"

#Terminaisons
$ns at 4.5 "$appVoIPClientToInternet($i) stop"
$ns at 4.5 "$appVoIPInternetToClient($i) stop"
$ns at 4.5 "$appHTTPServer($i) stop"
$ns at 4.5 "$appFTPServer($i) stop"
}
$ns at 5 "finish"

```

4.2 Routage sans DiffServ

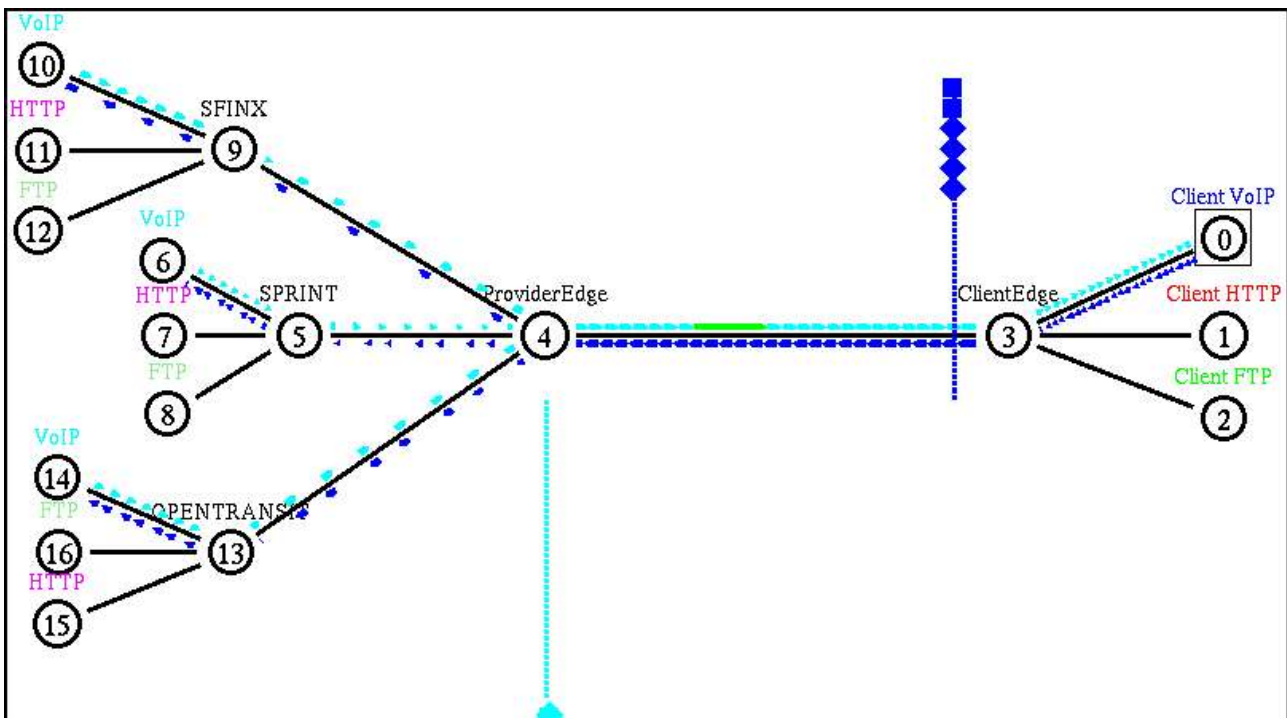


Illustration 3 - Simulation sans DiffServ dans nam (network animator)

La création des liens est spécifique parce que sans DiffServ, on peut mettre des duplex-link alors qu'il faudra mettre en place des simplex-link dans les 2 sens avec DiffServ.

```

#Client
$ns duplex-link $nodeClientVoIP $nodeClientEdgeRouter 4Mb 10ms DropTail
$ns duplex-link $nodeClientHTTP $nodeClientEdgeRouter 2Mb 10ms DropTail
$ns duplex-link $nodeClientFTP $nodeClientEdgeRouter 2Mb 10ms DropTail

```

```

#notre core network = entre nodeClientEdgeRouter et nodeProviderEdgeRouter
$ns duplex-link $nodeClientEdgeRouter $nodeProviderEdgeRouter 3Mb 15ms DropTail

#taille de la file ds les 2 sens
$ns duplex-link-op $nodeProviderEdgeRouter $nodeClientEdgeRouter queuePos 0.5
$ns duplex-link-op $nodeClientEdgeRouter $nodeProviderEdgeRouter queuePos 0.5

#Serveur
for {set i 1} {$i <= $providerNb } {incr i} {
    $ns duplex-link $nodeProviderEdgeRouter $nodeProviderRouter($i) 4Mb 10ms DropTail
    $ns duplex-link $nodeProviderRouter($i) $nodeServerVoIP($i) 2Mb 10ms DropTail
    $ns duplex-link $nodeProviderRouter($i) $nodeServerHTTP($i) 2Mb 10ms DropTail
    $ns duplex-link $nodeProviderRouter($i) $nodeServerFTP($i) 2Mb 10ms DropTail

    #taille de la file pour les liens providerEdge vers providerRouter
    # $ns duplex-link-op $nodeProviderEdgeRouter $nodeProviderRouter($i) queuePos 0.5
}

```

4.3 Routage avec DiffServ

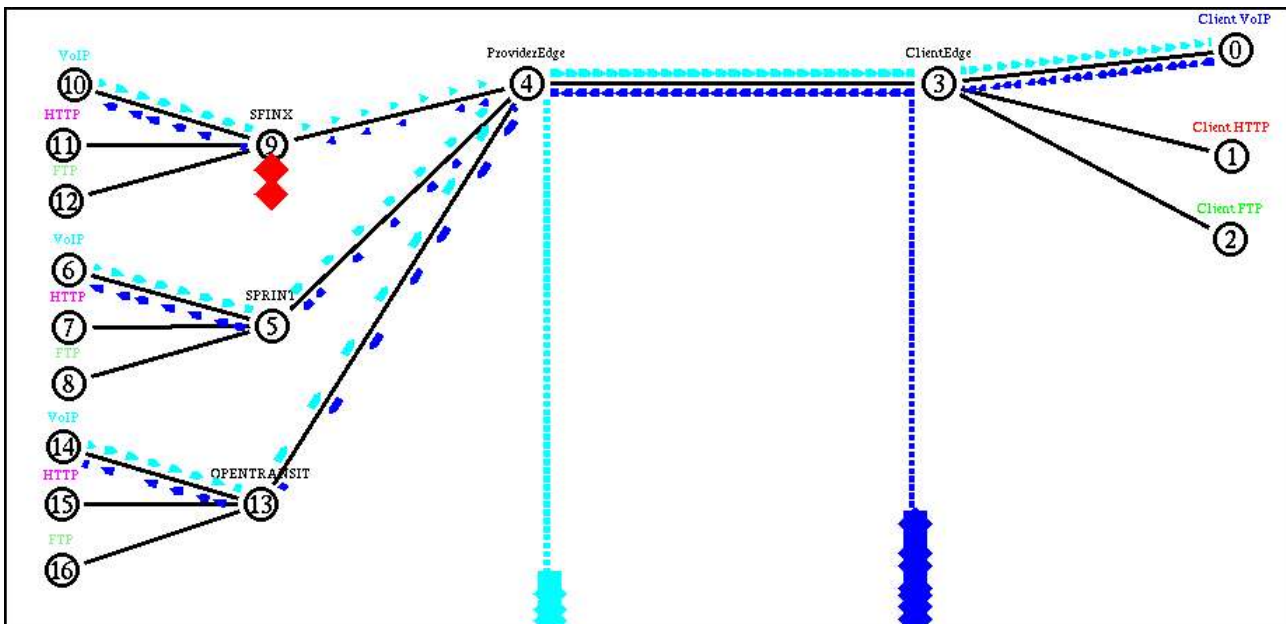


Illustration 4 - Simulation avec DiffServ dans nam (network animator)

4.3.1 Principe de DiffServ

DiffServ étant un protocole de QoS, il permet de différencier les paquets et de leur assigner des priorités. Le protocole DiffServ s'applique sur un réseau, et cette différenciation se fait en 2 étapes.

Lorsqu'un paquet entre dans le réseau, celui-ci est « marqué » d'un identifiant (DSCP: DiffServ Code Point) calculé à partir de son protocole, de sa source et de sa destination. Les routeurs effectuant le marquage à l'entrée du réseau sont appelés routeurs « Edge ».

À l'intérieur du réseau, à chaque fois que le paquet sera routé, l'algorithme DiffServ regardera son type, et par un système de probabilité et de files d'attente virtuelles décidera de router ou de détruire le paquet. Les routeurs effectuant le routage sont appelés routeurs « Core ». En effet, un paquet arrivant est placé dans la file correspondant à son marqueur. Ensuite, le paquet sera routé ou supprimé selon les paramètres de la file.

Dans notre cas, nous souhaitons privilégier les paquets VoIP par rapport aux autres paquets afin de garantir à nos clients une certaine qualité dans l'utilisation de cette technologie. Les autres paquets tels que FTP ou HTTP pouvant être retransmis car utilisant TCP.

4.3.2 La création des liens

Dans NS, l'algorithme DiffServ se met en place sur les liens, et pas sur les routeurs. Il faut donc passer par des liens simplex afin de paramétrer les envois et les réceptions différemment.

Nous avons donc défini des liens « Edge » de nos clients vers notre point d'accès client, le lien retour utilisant toujours l'algorithme « DropTail ».

```
# Création des liens simplex depuis les clients vers le edge routeur
$ns simplex-link $nodeClientVoIP $nodeClientEdgeRouter 4Mb 10ms dsRED/edge
$ns simplex-link $nodeClientHTTP $nodeClientEdgeRouter 2Mb 10ms dsRED/edge
$ns simplex-link $nodeClientFTP $nodeClientEdgeRouter 2Mb 10ms dsRED/edge

# les liens de retour vers les clients sont des droptail classiques
$ns simplex-link $nodeClientEdgeRouter $nodeClientVoIP 4Mb 10ms DropTail
$ns simplex-link $nodeClientEdgeRouter $nodeClientHTTP 2Mb 10ms DropTail
$ns simplex-link $nodeClientEdgeRouter $nodeClientFTP 2Mb 10ms DropTail
```

Notre réseau simulé par le lien entre les deux points d'accès est configuré en lien « Core » afin que l'algorithme de routage de DiffServ soit appliqué. Nous avons ici aussi défini 2 liens simplex pour appliquer DiffServ.



```
# Notre réseau
# Liens simplex d'un routeur vers l'autre susceptible de générer des drops,
# on active donc une file dsRED/core (car les paquets sont déjà taggés)
$ns simplex-link $nodeClientEdgeRouter $nodeProviderEdgeRouter 3Mb 15ms dsRED/core
$ns simplex-link $nodeProviderEdgeRouter $nodeClientEdgeRouter 3Mb 15ms dsRED/core

#taille de la file
$ns simplex-link-op $nodeClientEdgeRouter $nodeProviderEdgeRouter queuePos -0.5
$ns simplex-link-op $nodeProviderEdgeRouter $nodeClientEdgeRouter queuePos 0.5
```

Du côté de nos connexions Internet, nous avons configuré le lien arrivant d'un grand réseau vers notre point d'accès en lien « Edge », les autres liens étant des liens DropTail standard.

```
# Réseaux Internet
for {set i 1} {$i <= $providerNb } {incr i} {
    $ns simplex-link $nodeProviderEdgeRouter $nodeProviderRouter($i) 4Mb 10ms DropTail
    $ns simplex-link $nodeProviderRouter($i) $nodeProviderEdgeRouter 4Mb 10ms
    dsRED/edge

    $ns duplex-link $nodeProviderRouter($i) $nodeServerVoIP($i) 2Mb 10ms DropTail
    $ns duplex-link $nodeProviderRouter($i) $nodeServerHTTP($i) 2Mb 10ms DropTail
    $ns duplex-link $nodeProviderRouter($i) $nodeServerFTP($i) 2Mb 10ms DropTail
}
```

4.3.3 Les deux type de files

DiffServ fonctionnant sur un principe d'utilisation de files d'attente virtuelles au sein des files physiques. On définit des priorités, des caractéristiques différentes pour ces files afin de les traiter de façon plus ou moins privilégiée. On peut notamment varier la probabilité de supprimer un paquet.

Pour nos simulations, nous avons décidé de définir deux types de paquet puisqu'il s'agissait de privilégier les flux VoIP par rapport aux autres.

Nous avons donc définir 2 types de flux pour notre simulation : BestEffort pour les paquets les plus prioritaires et ExpediteForward pour les autres :

```
set ExpediteForward(cp) 10
set ExpediteForward(qw) 6
set ExpediteForward(in_min) 4.5
set ExpediteForward(in_max) 8.0
set ExpediteForward(in_prob) 0.05
```

```
set ExpediteForward(out_min) 0.0 ;#on jette tout paquet hors du profil
set ExpediteForward(out_max) 0.0
set ExpediteForward(out_prob) 1.00
set ExpediteForward(qlimit) 10

set BestEffort(cp) 20
set BestEffort(qw) 1
set BestEffort(in_min) 15
set BestEffort(in_max) 60
set BestEffort(in_prob) 0.05
set BestEffort(out_min) 15
set BestEffort(out_max) 60
set BestEffort(out_prob) 0.05
set BestEffort(qlimit) 100
```

4.3.4 Fonction de création d'un lien « Edge »

Afin de nous simplifier la configuration des liens « Edge », nous avons écrit une fonction TCL qui effectuait la configuration du lien. Rappelons que les liens « Edge » ont la responsabilité de marquer les paquets afin de permettre le travail de tri des liens « Core ».

Cette fonction configure un lien « Edge » en définissant les 2 types de flux, et la politique de marquage que l'on va leur appliquer.

Dans notre cas, nous avons pu identifier le type de trafic selon la source puisque nous avons pour chaque source un type de trafic différent. Dans la réalité, nous aurions dû regarder le protocole de chaque paquet en plus de regarder la source. Il est aussi possible avec l'algorithme DiffServ de regarder la destination du paquet, mais nous n'avons pas utilisé cette fonctionnalité.

Nous utiliserons simplement 2 files virtuelles, une pour les paquets correspondant au type de trafic à identifier, et une 2^e pour les autres.

```
#on passe src et host parce qu'ils peuvent être différents.
#On n'a pas besoin du protocole (app) parce qu'on indique
#la source et que ds notre cas, chaque source émet un
#protocole différent et connu.
proc createEdgeLink {src host edgeRouter cir phbName} {
    global ns queues ExpediteForward BestEffort

    set queues($host$edgeRouter) [[ $ns link $host $edgeRouter ] queue]
```



```
if { $phbName=="BestEffort" } {
  set cp_ $BestEffort(cp)
  set in_min_ $BestEffort(in_min)
  set in_max_ $BestEffort(in_max)
  set in_prob_ $BestEffort(in_prob)
  set out_min_ $BestEffort(out_min)
  set out_max_ $BestEffort(out_max)
  set out_prob_ $BestEffort(out_prob)
  set qlimit_ $BestEffort(qlimit)
} else {
  set cp_ $ExpediteForward(cp)
  set in_min_ $ExpediteForward(in_min)
  set in_max_ $ExpediteForward(in_max)
  set in_prob_ $ExpediteForward(in_prob)
  set out_min_ $ExpediteForward(out_min)
  set out_max_ $ExpediteForward(out_max)
  set out_prob_ $ExpediteForward(out_prob)
  set qlimit_ $ExpediteForward(qlimit)
}
$queues($host$edgeRouter) set numQueues_ 1
$queues($host$edgeRouter) setNumPrec 2
$queues($host$edgeRouter) meanPktSize 300

#$queues($host$edgeRouter) setPhysQueueSize 0 0 $qlimit_

#Les politiques : on utilise l'id de la source
#(rappel : c ça qui nous permet
#de ne pas avoir besoin du proto
$queues($host$edgeRouter) addPolicyEntry [$src id] -1 TSW2CM $cp_ $cir

#On facilite la vie le plus possible au best effort
if { $phbName == "BestEffort" } {
  $queues($host$edgeRouter) addPolicerEntry TSW2CM $cp_ $cp_
} else {
  $queues($host$edgeRouter) addPolicerEntry TSW2CM $cp_ [expr $cp_+1]
}

#Dans quelle file on met quel code point
$queues($host$edgeRouter) addPHBEntry $cp_ 0 0
$queues($host$edgeRouter) addPHBEntry [expr $cp_+1] 0 1
$queues($host$edgeRouter) configQ 0 0 $in_min_ $in_max_ $in_prob_
$queues($host$edgeRouter) configQ 0 1 $out_min_ $out_max_ $out_prob_
}
```

4.3.5 Fonction de création d'un lien « Core »

La configuration d'un lien « Core » est plus complexe que celle d'un lien « Edge ». Il faut dans un premier temps remettre la même configuration que le lien « Edge » afin qu'il connaisse les différents types de flux.

Il faut ensuite configurer le lien « Core » en lui spécifiant tous les paramètres pour les files virtuelles qui vont servir au tri des paquets. C'est ici que nous utiliserons la majorité des paramètres des files définis au 4.3.3 (page 14).

```
proc createCoreLink {node1 node2} {
    global ns queues ExpediteForward BestEffort

    #On définit la file pour le lien de $node1 vers node2
    set queues($node1$node2) [[${ns link $node1 $node2} queue]

    $queues($node1$node2) set numQueues_ 2
    $queues($node1$node2) setNumPrec 2
    $queues($node1$node2) meanPktSize 300

    $queues($node1$node2) setSchedulerMode WRR

    #On indique le poids de chaque file pour le WRR
    $queues($node1$node2) addQueueWeights 0 $ExpediteForward(qw)
    $queues($node1$node2) addQueueWeights 1 $BestEffort(qw)

    #pour chacune des files virtuelles, on configure le phb (code point)
    #File physique 0 : ExpediteForward
    $queues($node1$node2) addPHBEntry $ExpediteForward(cp) 0 0 ;
    $queues($node1$node2) addPHBEntry [expr $ExpediteForward(cp)+1] 0 1
    $queues($node1$node2) configQ 0 0 $ExpediteForward(in_min) $ExpediteForward
(in_max) $ExpediteForward(in_prob)
    $queues($node1$node2) configQ 0 1 $ExpediteForward(out_min) $ExpediteForward
(out_max) $ExpediteForward(out_prob)

    #File physique 1 : BestEffort
    $queues($node1$node2) addPHBEntry $BestEffort(cp) 1 0 ;
    $queues($node1$node2) addPHBEntry [expr $BestEffort(cp)+1] 1 1
    $queues($node1$node2) configQ 1 0 $BestEffort(in_min) $BestEffort(in_max)
$BestEffort(in_prob)
    $queues($node1$node2) configQ 1 1 $BestEffort(out_min) $BestEffort(out_max)
$BestEffort(out_prob)
}
```

4.3.6 Utilisation des fonctions et mise en place de la QoS par DiffServ

Nous pouvons ici voir la configuration finale de l'algorithme DiffServ pour notre architecture.

Nous mettons d'abord en place la configuration de notre réseau avec les liens « Core », puis nous configurons les liens « Edge » de nos clients vers leur point d'accès à notre réseau, et enfin les liens « Edge » des serveurs de chaque grand réseau vers le point d'accès de ce réseau.

```
#Les 2 liens core (même relation dans les 2 sens)
createCoreLink $nodeClientEdgeRouter $nodeProviderEdgeRouter
createCoreLink $nodeProviderEdgeRouter $nodeClientEdgeRouter

set cir_ftp 200
set cir_http 200
set cir_voip 200

#les edges
createEdgeLink $nodeClientVoIP $nodeClientVoIP $nodeClientEdgeRouter $cir_voip
"BestEffort"
createEdgeLink $nodeClientHTTP $nodeClientHTTP $nodeClientEdgeRouter $cir_http
"ExpediteForward"
createEdgeLink $nodeClientFTP $nodeClientFTP $nodeClientEdgeRouter $cir_ftp
"ExpediteForward"

for {set i 1} {$i<=$providerNb} {incr i} {
    createEdgeLink $nodeServerVoIP($i) $nodeProviderRouter($i) $nodeProviderEdgeRouter
    $cir_voip "BestEffort"
    createEdgeLink $nodeServerHTTP($i) $nodeProviderRouter($i) $nodeProviderEdgeRouter
    $cir_http "ExpediteForward"
    createEdgeLink $nodeServerFTP($i) $nodeProviderRouter($i) $nodeProviderEdgeRouter
    $cir_ftp "ExpediteForward"
}
```

5 Analyse des résultats

La comparaison effectuée entre les deux simulations avec et sans DiffServ a consisté à mesurer le taux de paquets VoIP droppés dans toute l'architecture. Pour cela, nous avons analysé le fichier de sortie créé par *ns* et généré des diagrammes en fonction des résultats.

Nous souhaitons mettre en place une certaine qualité de service sur le trafic VoIP, et cette qualité de service se traduit par le nombre de paquets VoIP qui arrivent effectivement à leur destinataire.

Une analyse intéressante est donc de suivre au fur et à mesure du temps le nombre de nombre de paquets VoIP perdus avec et sans DiffServ. Nous pourrions ainsi calculer le gain de qualité sur notre simulation avec DiffServ.

5.1 Parsing

Il s'agissait de récupérer les lignes concernant les drops de paquets provenant de flux CBR⁵ (VoIP) et les autres. L'objectif était de pouvoir tracer un graphe cumulatif par centième de seconde du nombre de drops CBR/Non CBR avec ou sans DiffServ.

La première étape a donc consisté à récupérer lesdits drops de paquets CBR et non CBR.

- Les drops cbr :

```
$cat fichier.nam |grep "^d.*cbr"
```

- Les autres drops :

```
$cat fichier.nam | grep "^d" | grep -v cbr
```

Les lignes ainsi récupérées ressemblent à cela :

```
d -t 1.0028599999999999 -s 4 -d 3 -p cbr -e 210 -c 1 -i 2101 -a 1 -x {6.1 0.1 344  
----- null}
```

Pour savoir combien de drops il y a eu à chaque centième de seconde, seule la troisième colonne indiquant le moment auquel le drop est survenu nous intéresse. On la récupère donc en « pipant » de nouveau le résultat précédent sur un *awk* '{print \$3}'.

En reproduisant ces commandes pour les fichiers avec et sans DiffServ, on obtient chaque fois la liste des temps auquel un drop est survenu :

5 Constant Bit Rate



```
...  
0.6360599999999996  
0.6388599999999996  
0.6416599999999996  
...
```

Nous avons ensuite écrit un script perl pour compter combien de drops étaient effectués par centième de seconde.

treatDrop.pl :

```
#!/usr/bin/perl  
use strict;  
  
my $prevTime = 0;  
my $count = 0;  
  
while(<>)  
{  
    $_ =~ s/\n//;  
    my $timeRaw = $_;  
  
    my $time = ($timeRaw * 100);  
    $time =~ s/\..+//g;  
  
    while($prevTime < $time)  
    {  
        print "$prevTime $count\n";  
        $prevTime++;  
    }  
  
    $prevTime = $time;  
    $count++;  
}  
  
print "$prevTime $count\n";
```

Ce fichier génère des lignes directement exploitables par un logiciel de génération de graphiques. Il affiche le numéro du centième de seconde (de 0 à 500 maximum, puisque la simulation est configurée pour durer 5 secondes) suivi du nombre de drops apparus durant celle-ci.

5.2 Chiffres

	Sans Diffserv	Avec DiffServ
Nombre de CBR	2995	2761
Nombre de !CBR	36	18
% drop CBR	1,2	0,65
Tous les drops	3031	2779
% drops en moins		8,31

Ce tableau récapitule le nombre de drops rencontrés dans chacun des modes :

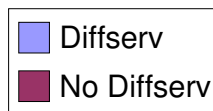
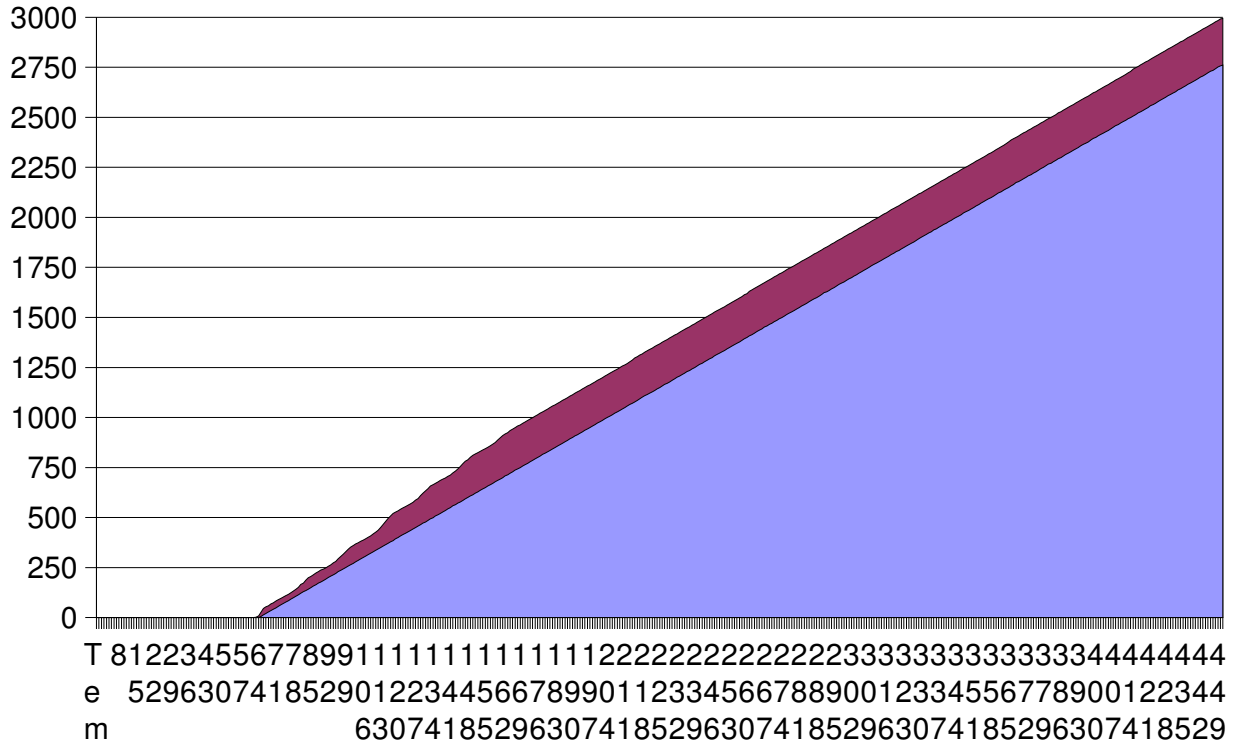
On constate qu'on a une diminution de 8,3% du nombre de drops lorsqu'on active DiffServ. On a aussi globalement moins de drops lorsqu'on active DiffServ.

5.3 Graphiques

Les graphiques suivants sont issus des mêmes données que celles qui ont permis de donner les chiffres ci-dessus. Ils représentent le cumul des drops au fur et à mesure de la simulation.

5.3.1 Les drops de VoIP

CBR



5.3.2 Les drops des autres flux

La légende du graphique précédent s'applique ici aussi.



6 Conclusion

Suite à l'analyse de la simulation de l'algorithme DiffServ que nous avons réalisée, nous pouvons voir qu'il permet une amélioration de la qualité du transport du trafic VoIP sur notre réseau. Il permet donc à nos clients d'être plus satisfaits de nos prestations VoIP tout en étant faiblement dérangés lors de leurs transferts HTTP ou FTP.

Cependant, on peut évidemment se poser la question de la difficulté de mise en place de cet algorithme. Il faut en effet bien connaître son fonctionnement pour l'implémenter efficacement. Les concepts liés aux files virtuelles sont par exemple très compliqués à comprendre mais permettent une très grande souplesse dans le traitement des paquets selon les très nombreux critères utilisables.

Cette souplesse est tout de même ce qui fait la force de DiffServ. À notre connaissance, c'est le seul protocole de QoS qui soit configurable aussi finement. De plus, l'un des énormes avantages de ce protocole réside dans son fonctionnement purement logiciel : il est donc facilement modifiable sans impacter l'organisation matérielle du réseau. Devoir modifier la structure de notre réseau serait un investissement et un risque autrement plus conséquent. DiffServ permet en effet une configuration dynamique de la QoS et une modification en temps réel des règles, facilement centralisable sur une seule machine par le biais d'une console d'administration SNMP⁶ par exemple, afin de répondre rapidement à tout problème ponctuel pouvant survenir sur le réseau.

Il deviendrait donc possible pour nos solutions d'hébergements de garantir des offres VoIP assurant un débit garanti à nos clients. De même, nous serions à même de conserver un niveau minimum de fonctionnement pour nos clients les plus importants malgré d'éventuels problèmes provoquant une réduction de la capacité de notre réseau.

Toutes ces raisons font que DiffServ, malgré sa difficulté d'apprentissage, représenterait certainement un investissement intéressant pour nous aider dans beaucoup de problèmes de gestion du trafic auxquels nous devrions faire face.

6 Simple Network Management Protocol