IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TP - La QoS dans les réseaux IP

F2B407B

2018-2019

Préambule

Le concept de **Qualité de Service** ou QoS (*Quality of Service*) dans le domaine des réseaux recouvre «l'ensemble des mécanismes permettant de contrôler la gestion du trafic réseau afin d'assurer que le réseau satisfasse bien les besoins de service des applications».

Deux types de mécanismes peuvent être identifiés selon que l'on considère la QoS du point de vue global sur la totalité du réseau ou du point de vue particulier sur chaque équipement (routeur).

1. QoS du point de vue global : les mécanismes mis en œuvre dans cette catégorie assurent le service souhaité sur l'ensemble du réseau. Les applications (ou leurs utilisateurs) ou les gestionnaires du réseau définissent des politiques de QoS et celles-ci sont traduites en demandes de réservation de ressources dans les nœuds du réseau. Des protocoles (RSVP, COPS) ou des mécanismes (DiffServ) permettent d'effectuer ces réservations. Ces aspects ne seront pas développés plus avant ici car ils sont traités en cours.
2. Les techniques fines de QoS : elles permettent la mise en œuvre effective de la QoS dans les routeurs. Une requête RSVP, par exemple, se traduira effectivement dans chaque routeur par l'utilisation d'une de ces techniques. Celles-ci peuvent être utilisées sur décision des administrateurs des routeurs, sans l'aide d'un mécanisme global. Ce sont quelques-unes de ces techniques que nous étudierons dans ce TP. Nous ne pourrons pas, hélas, tout étudier et ne testerons pas les techniques avec tous les protocoles.

D'une manière générale, comme l'illustre la figure 1, une politique de QoS sur un équipement réseau se résume à trois grandes questions :

— les files d'attente : combien et quelle taille ?
— l'entrée dans ces files d'attente : lorsqu'un paquet IP arrive sur un routeur, dans quelle file le ranger ? (notions de *flux*, de *classes*, etc.)
— la sortie de ces files d'attente : combien de paquets sont pris sur quelle file et à quel moment pour être envoyés sur l'interface de sortie ?

Chacune des quelques politiques de QoS de ce TP va répondre à sa manière à ces trois questions.

Note : Pour en savoir plus, je peux vous conseiller le wiki suivant, pas trop long, pertinent et bien rédigé : `http://pteu.fr/doku.php?id=informatique:cisco:qos`

1 Mise en place du réseau et paramétrages

Le réseau de test est architecturé comme le montre la figure 2.

Les machines terminales sont des PCs munis du système d'exploitation Linux. Le mot de passe administrateur vous sera donné en début de séance.

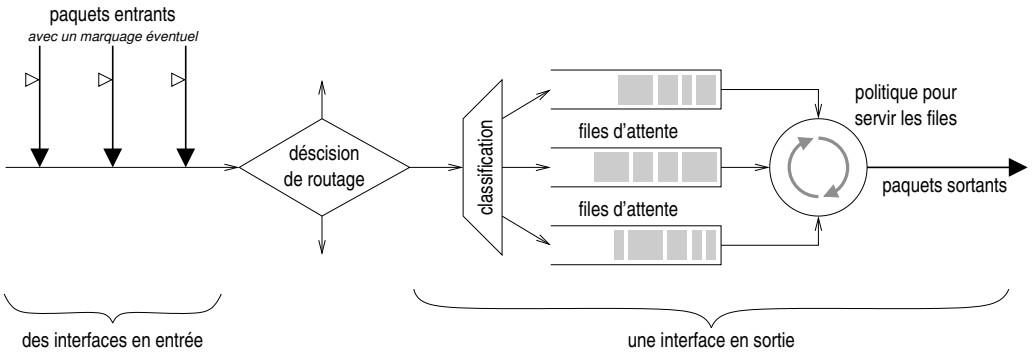


FIGURE 1 – Principe de la QoS sur un équipement réseau

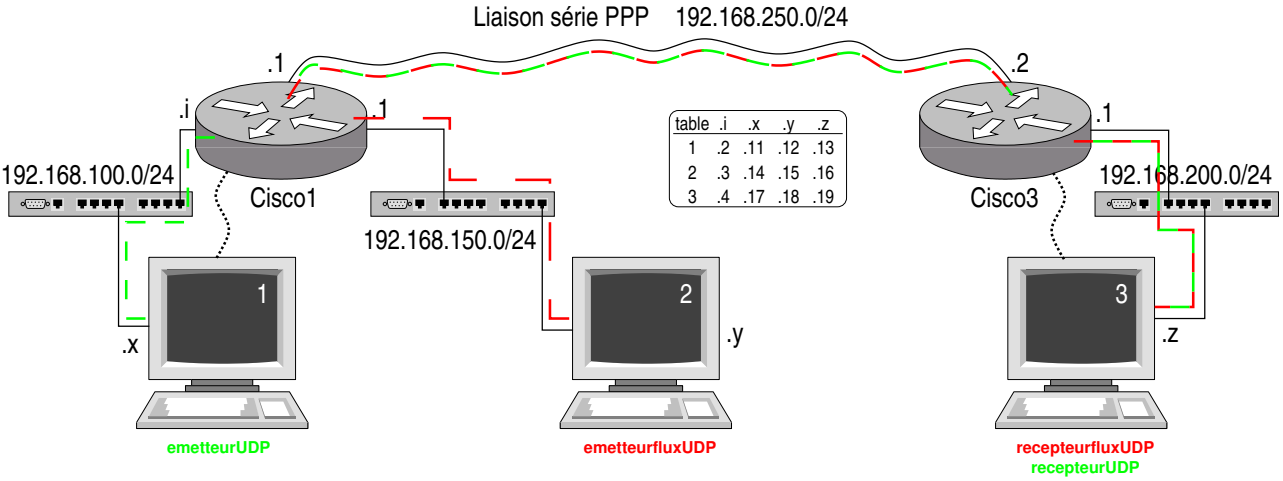


FIGURE 2 – Réseau de test

Configuration du réseau :

— Sur les machines 1 et 3 lancez `minicom` (ou `gtkterm` si vous aimez mieux), et appuyez sur la touche `Entrée`. Vous accédez ainsi aux commandes des routeurs. Vérifiez la configuration existante (`show running-config`). Ou si vous êtes pressés, tapez `sh ip int br`.

— Ouvrez une autre fenêtre terminal administrateur sur tous les PC (et donc également sur la machine 2) et positionnez-vous dans le répertoire `/opt/TP/TPQoS`. Vous trouverez notamment dans ces répertoires différents fichiers et scripts de configuration des machines et des routeurs. N'hésitez pas à faire preuve de curiosité et à regarder leur contenu. Exécutez le fichier `./configure` qui initialisera les caractéristiques réseau de la machine. (Quelques messages d'erreur peuvent apparaître, ils sont généralement sans importance.)

— Vérifiez les configurations des trois PC. (`ifconfig` ou bien `ip address show`; et `route -n` ou bien `ip route show`)

— Sur les machines 1 et 3, affichez le fichier `configrouteur` contenu dans le répertoire `TPQoS`.

— Vérifiez l'accessibilité de toutes les machines (`ping`) : 1 et 2 doivent pouvoir joindre 3.

2 Mécanisme par défaut

Lorsque aucune demande explicite de QoS n'est soumise à un routeur celui-ci utilise par défaut un mécanisme *Fair Queueing* sur ses interfaces.

Vous allez maintenant expérimenter ce mécanisme.

— Sur le routeur de gauche, vérifiez le politique associée à l'interface de «sortie» : `show interface s0/0/0`

— Placez-vous dans le répertoire `TPQoS` sur toutes les machines, dans toutes les fenêtres que vous ouvrez.

— Sur la machine 3, dans une fenêtre terminal, lancez le programme `./recepteurfluxUDP`.

— Sur la machine 2 lancez le programme `./emetteurfluxUDP` en donnant les paramètres nécessaires pour que ce flux soit envoyé vers le programme `./recepteurfluxUDP` sur la machine 3. Ce flux sera notre charge réseau contre laquelle nous tenterons d'imposer d'autres flux en les priorisant.

— Vérifiez avec `wireshark` ou `tcpdump` que le flux arrive bien sur la machine 3. (Ne laissez pas wireshark capturer trop longtemps ce flux incessant, si non le PC va finir pas swapper...)

— Sur la machine 3, dans une autre fenêtre lancez le programme `./recepteurUDP`. Ce programme va recevoir des paquets UDP précédés par une annonce du nombre de paquets à recevoir. Il prendra alors l'heure système et la mémorisera. Il recevra ensuite les paquets annoncés. Ceux-ci se termineront par un paquet spécial contenant le mot «`FIN`» (acquitté par un paquet contenant le mot «`OKFIN`»). Le programme prendra alors à nouveau l'heure système. Il pourra ainsi calculer le débit du flux reçu en fonction du temps mis à recevoir les paquets. À noter qu'avec UDP, des paquets peuvent être rejetés s'il y a congestion dans le réseau (avec TCP aussi mais TCP demande leur retransmission). Le programme nous indiquera le nombre de paquets reçus. La mesure de débit ne sera à considérer que pour un nombre de paquets reçus correspondant au nombre annoncé.

— Sur la machine 1, lancer le programme `./emetteurUDP` en lui donnant comme arguments l'adresse et le port du recepteur ainsi que le nombre de messages à émettre, on choisira 20 pour cette valeur. (Il y a un dernier paramètre optionnel, *precedence*, nous l'utiliserons plus tard, ne mettez rien pour l'instant.)

— Faites les essais suivants en notant pour chacun le débit annoncé :

— Machine 1 → machine 3 (`emetteurUDP`→`recepteurUDP`) en présence du flux de fond 2 → 3. Recommencez plusieurs fois en notant le débit annoncé par le recepteur.
— Arrêtez le flux de fond 2 → 3, attendez quelques minutes, vous pouvez lancer `wireshark` sur la machine 3 pour vérifier que le flux 2 → 3 s'est bien arrêté. Relancez alors le flux 1 → 3 plusieurs fois en notant les débits.
— Relancez le flux 2 → 3 et refaites les essais 1 → 3 en notant toujours les débits.

Question : Que constatez-vous ?
Vous remarquerez que nous n'avons pas encore paramétré la QoS.

3 Flow-Based Weighted Fair Queueing

Le Fair Queueing est un peu plus riche que cela. En fait, cette stratégie (historiquement ancrée dans la philosophie *best effort* et dans le monde IP) sait accorder plus de poids à certain flux par rapport à d'autre. On l'appelle alors *Weighted Fair Queueing*.

Deux concepts dans cette expression : *fair queueing* d'une part et *weighted* d'autre part, i.e. une notion de *juste traitement de file d'attente* et une notion de *poids*. Le *fair queueing* est un mécanisme assurant un traitement équitable à tous les flux en termes de nombres d'octets émis. Ainsi, par exemple, si dans la file 1 on a un paquet de 100 octets et dans la file 2 deux paquets de 50 octets alors on émettra un paquet de la file 1 et deux paquets de la file 2.

Le mécanisme WFQ mis en œuvre de base par Cisco repose sur l'utilisation du champ TOS de l'entête IP des paquets qui circulent, plus exactement les bits dits *de précédence* qui sont les trois bits de poids fort de l'octet TOS. Notez que cette information est transportée par les paquets eux-mêmes. Ce poids peut être attribué par l'émetteur, ou modifié par n'importe quel équipement IP sur le réseau. Mais regardons ce qu'il se passe lorsque le routeur doit traiter un paquet qui comporte une indication de précédence.

— Avec le programme `emetteurUDP`, jouez avec le dernier paramètre optionnel : *precedence*. Le programme placera cette valeur dans les 3 bits de poids fort du champ TOS, la fameuse précédence transportée dans les paquets IP eux-mêmes. Variez de 0 (pas de précédence, valeur classique par défaut), jusqu'à 7.
— Observez le débit obtenu.
— Est-ce que vous arrivez à deviner l'équation qui relie la valeur de précédence et le débit observé ? (Nous y reviendrons en section 10.)
— À l'aide de `wireshark`, retrouvez dans l'entête IP des paquets la précédence que vous avez positionnée. (Préalablement, configurez `wireshark` ainsi : dans le menu *Edit / Preferences / Protocols / IPv4*, retirez l'option *Decode IPv4 TOS field as DiffServ field*).
— Avec `wireshark`, regardez dans quel ordre arrive les paquets entre nos deux flux. Voyez-vous un lien avec la valeur de la précédence ?

Note : Dans la terminologie Cisco, ce mécanisme porte le nom de *Flow-Based Weighted Fair Queueing* (par opposition au *Class-Based Weighted Fair Queueing* que l'on abordera ensuite). Généralement les termes *fair queueing*, *weighted fair*, *weighted fair queueing*, *flow based weighted fair queueing*, sont tous synonymes et désignent le mécanisme que l'on trouve par défaut sur les équipements IP (PC, routeurs, etc.).

4 Le mécanisme Fifo

Avec la commande de niveau `interface` enlevez le mode `fair-queue` sur l'interface série du routeur de gauche. Vérifier le résultat avec `show interface s0/0/0`. Pour cela, placez-vous dans la fenêtre `gtkterm` de la machine 1 :

```
cisco1> enable

Password:

cisco1# show interface serial 0/0/0

cisco1# configure terminal

cisco1(config)# interface s0/0/0
```

```
cisco1(config-if)# no fair-queue

cisco1(config-if)# exit

cisco1(config)# exit

cisco1# show interface s0/0/0
```

Maintenant, refaites les essais précédents, sans flux de fond pour commencer puis lancez le flux de fond et recommencez.

Note : la commande `show interface s0/0/0` affiche un certain nombre de compteurs que vous pouvez remettre à zéro avec `clear counter s0/0/0` (ou `clear counters`).

Question : Que constatez-vous ?
Comment expliquez-vous le phénomène observé lorsqu'il y a du flux de fond ?

Notes concernant Cisco :

- La taille du FIFO se configure au niveau de l'interface avec la commande `hold-queue` n `in /out` (n en paquets ; par défaut 75 en in, 40 en out)
- Ne pas confondre la stratégie de QoS FIFO avec le buffer de sortie de l'interface (qui est également un FIFO), mais qui est plutôt d'ordre matériel. Ce dernier se configure avec la commande `tx-ring-limit`, pour indiquer la taille du buffer en nombre de *particles* (des tranches de 512 octets).
- Mais tout ceci est bien spécifique à Cisco. Si vous avez compris l'impact d'une stratégie FIFO sur la QoS, c'est le principal.

5 Class Based Queuing

5.1 Introduction

Avant de passer à l'action prenons le temps de quelques explications spécifiques à nos équipements.

Historiquement (jusqu'à l'IOS 15.0), Cisco proposait principalement les stratégies de QoS suivantes :

- *Weigthed Fair Queuing* : le grand classique dans le monde du Best Effort, tel que vu précédemment.
- *Priority-Queuing* : 4 files de priorité stricte (*low normal medium high*) ; le routeur fait sortir les paquets de la file de priorité supérieure avant de regarder la (les) file(s) de priorité inférieure ; on ne fait aucun calcul sur la bande passante consommée / disponible.
- *Custom-Queuing* : 16 files disponibles ; pour chacune on spécifie une quantité (en octet, 1500 par défaut, comme le MTU) indiquant la quantité de données à servir à chaque tour de round-robin ; autrement dit on alloue à chaque file une portion de la bande passante disponible, sans notion de priorité.

Ces stratégies étaient assez simples à comprendre. On a une séparation claire entre la notion de *priorité* d'une part (certains flux passent plus tôt, *avant* les autres), et la notion de *bande passante* d'autre part (certains flux ont droit à *plus* de place que d'autre sans forcément passer plus tôt). Par contre ces stratégies ne sont pas forcément pratiques à utiliser en condition réelle...

Dans le même temps (à partir d'IOS 12.0), Cisco a commencé à introduire une façon de paramétrer la QoS légèrement différente, appelée *Class Based Queuing* (CBQ). On retrouve un peu tous les mécanismes précédents, par contre ils sont organisés autour de la notion de *classes* de flux, un mécanisme un peu plus générique.

Les stratégies disponibles sont :

- *Low Latency Queuing* (LLQ) : on retrouve le principe de flux prioritaires sur d'autres. On peut déclarer qu'une classe (ou flux) a une priorité stricte sur la class-default (i.e. tous ses paquets sortent avant). On peut également paramétrer le flux LLQ, en lui spécifiant une portion de bande passante minimale garantie pour chacun. Plus précisément, il s'agit de l'algorithme du token-bucket. Si l'on n'est pas en situation de congestion (c'est-à-dire qu'il y a de la bande passante non consommée par d'autres flux) notre flux prioritaire peut dépasser cette valeur. Par contre, en cas de congestion, le flux aura ce minimum garantit mais ne pourra pas le dépasser (des paquets sont détruits). Notons que la stratégie LLQ est ainsi nommée car elle bénéficie d'une latence faible et constante grâce au token bucket (intéressant pour de la voix sur IP par exemple).
  Notez que dans la documentation Cisco on désigne parfois cette stratégie par *priority policy*.

— *Class Based Weigthed Fair Queuing* (CBWFQ) : c'est un peu un mélange entre les principes du *Weigthed Fair Queuing* et du *Custom Queueing* historique décrit plus haut. Bref, les différents flux sont caractérisés non pas par le TOS, mais par une *classe* au sens Cisco. Et à chacune de ces classes on va allouer une portion de bande passante, selon l'algorithme du Custom-Queuing.
Notez que dans la documentation Cisco on désigne parfois cette stratégie par *bandwidth policy* (par opposition à *priority policy*).
En général, on paramètre une politique de QoS, avec une seule classe LLQ pour le flux prioritaire, et on répartit le reste de la bande passante avec plusieurs flux CBWFQ.
— Et à l'intérieur des *classes*, s'il y a en fait plusieurs flux, ils sont par défaut en mode FIFO. Mais dans le cas du CBWFQ, on peut choisir de les différencier par du *fair-queue* (i.e. sur les bits de précédence du TOS).
— À cela s'ajoute les spécificités liées au Frame Relay, à DSCP, et à la stratégie du Weighted Random Early Detection (WRED) consistant à détruire aléatoirement certains paquets dans les flux qui approchent de la limite de congestion...
— Et pour finir il y a des calculs savants pour réattribuer la bande passante non consommée aux flux qui en auraient besoin.

Vous trouvez cela compliqué ? [1] Attendez de voir quand on va le paramétrer...

La documentation Cisco (voir par exemple «*Modular Quality of Service Command Line Interface*» en annexe A) résume cela en 3 étapes :

1. Définir des classes avec la commande `class-map`
2. Créer une politique de service en associant chaque classe de trafic avec une ou plusieurs politiques de QoS (en utilisant la commande `policy-map`)
3. Attacher la politique de service à l'interface réseau avec la commande `service-policy`

5.2 Configuration

Le principe de base est de définir des classes de flux. Lorsqu'un nouveau paquet arrive, il faut que le routeur devine dans quelle classe le ranger. Historiquement, on faisait ça directement avec des Access List (ACL), en se basant sur les champs TCP-IP traditionnels (adresse IP source destination, protocole de transport, numéro de port source destination)...

Cette notion de classe apporte plus de souplesse, plus de critères de classification. On peut même encapsuler des classes dans des classes ; on parlera alors de classes *hiérarchiques*. (Ça fait rêver non ?)

Les critères relèvent typiquement :
— de la couche 2 : numéro de l'interface réseau par laquelle le paquet est arrivé, adresse MAC source ou destination, type du protocole de couche 3 transporté, classe de service du tag 802.1p, l'identifiant DLCI de Frame Relay, etc. ;
— de la couche "2.5" : autrement dit des labels et autres paramètres MPLS ;
— des couches 3&4 : typiquement et majoritairement on trouve les bonnes vieilles ACL, mais on peut également filtrer sur la longueur du paquet, le champ DSCP ou la précédence du TOS, etc. ;
— de la couche 7 : Cisco fournit un module de reconnaissance applicative (basé visiblement sur les numéros de port traditionnels et les premiers paquets) capable de reconnaître quelques types de trafics : FTP, HTTP, RTP, Citrix, Gnutella, FasTrack, etc. On peut filtrer sur ces protocoles, et parfois sur certains champs dans ces protocoles...
— et pour finir on a la possibilité de filtrer sur un peu n'importe quel octet donné dans l'entête du paquet...

5.2.1 Configurer des Access Lists

En mode `configuration` définissez deux `access-list` que vous appellerez `101` et `102`. La liste `101` concernera le flux de test ($1 \rightarrow 3$), la liste `102` concernera le flux de fond ($2 \rightarrow 3$). Repérez bien les numéros de port UDP que vous avez choisis.

La syntaxe peut être la suivante :

```
access-list  num_acc_list  permit  udp  host ip_src     any      eq   xyz
               <100-199>     deny   tcp       any   host ip_dst  gt  port_dst
                                    ip                            lt
                                   icmp                           neq
```

1. Et encore, je ne vous ai pas parlé du Generic Traffic Shaping (GTS) permettant de fixer le débit moyen et le pic toléré... Mais Cisco non plus n'est pas très loquace sur l'algorithme derrière ce truc-là... Un double token bucket ?

5.2.2 Configurer des class-maps

En vous aidant de l'annexe A, configurez les `class-map` suivantes :
— Vous pourrez appeler la première `flux13`, et configurez-là pour qu'elle `match` l'ACL 101 définie précédemment (mot clef `access-group`).
— La seconde pourrait s'appeler `flux23`, et matcher l'ACL 102.
Vérifiez votre configuration avec `show class-map`.

Notez qu'il y a une dernière classe à notre disposition, la `class-map default`, dans laquelle atterrissent les paquets qui ne rentrent pas dans les autres classes.

Voila, nos classes de flux sont prêtes, nous allons pouvoir faire de la QoS dans quelques instants...

6 Le mécanisme Low Latency Queueing

Le mécanisme LLQ assure que le trafic «important» est transmis avant le trafic «d'importance moindre». Ce mécanisme a été pensé pour du trafic de type voix sur IP, c'est-à-dire du trafic qui requiert une faible latence, mais qui n'est pas forcément très gourmand en bande passante. Par conséquent, on s'arrange d'une part pour faire passer les paquets LLQ avant les autres, d'autre part on ne les stocke pas dans des longues files d'attente (et par conséquent on s'autorise volontiers à perdre des paquets LLQ, notamment dans le cas où l'on veut limiter la bande passante accordée à LLQ).

Créez une `policy-map` (que vous pourrez appeler *`ma_qos`*), qui travaille sur la `class flux13`, en déclarant cette classe strictement prioritaire (mot clef `priority`, sans paramètre).
Ne précisez aucune autre classe.

Affectez cette politique de QoS à l'interface de sortie (i.e. l'interface série), pour traiter les paquets sortants. (En mode configuration d'interface : `service-policy output` *`ma_qos`*.

Vérifiez votre configuration avec `show policy-map` ou `show policy-map inferface s0/0/0`

— Lancez le flux de fond ($2 \rightarrow 3$) s'il ne l'est pas déjà, puis le flux de test ($1 \rightarrow 3$) et notez le débit. Vérifiez en faisant plusieurs essais.
— Modifiez votre `policy-map ma_qos`, et remplacez-la `class flux13` [2] par la `class flux23`. À chaque fois n'oubliez pas de définir votre classe comme prioritaire (`priority`). Et refaites les mesures à chaque fois, bien évidemment.
— Que constatez-vous et comment pouvez-vous l'expliquer ?

7 Le mécanisme Class Based Weigthed Fair Queuing

Ce mécanisme est aussi qualifié de politique de QoS en *bandwidth*, par opposition à LLQ qui est une politique avec *priority*.
Nous allons procéder comme précédemment.

Reprenez votre `policy-map ma_qos`. Incluez-la `class-map flux13` mais cette fois-ci, n'employez pas le mot-clef `priority`. (Si elle est déjà là, faites `no priority`).
Par contre, vous allez lui paramétrer une `bandwidth percent 60`.
Retirez les autres classes s'il y en a d'autres de présentes.
— Faites les mesures avec les flux.
— Modifiez votre `policy-map ma_qos`, et rajoutez-la `class flux23` avec une `bandwidth` à 15%.
— Donnez une borne maximale au `flux13` avec une règle du type `shape average percent 80`
— Vérifiez tout cela : `show policy-map inferface s0/0/0`
— Faites des mesures à chaque fois. Que constatez-vous ?
— Comment compareriez-vous LLQ et CBWFQ (les deux grandes fiertés de Cisco) ?

2. Pour retirer une `class`, il suffit de la précéder du mot clef `no`.

8 Utilisation conjointe de LLQ et CBWFQ

Ce type de configuration est, semble-t-il, assez courant dans les situations de convergence data-voip.

Tout d'abord on a *un* flux prioritaire en LLQ (typiquement les paquets de type VoIP, mais on peut y ajouter des paquets de signalisation divers, ICMP, les ACK TCP, etc., bref des paquets pas forcément très gros, plutôt sporadiques, mais qui peuvent être un peu bloquant pour les couches supérieurs s'ils tardent à arriver). Et dans le même temps on va répartir la bande passante restante entre les flux applicatifs, fichier, web, data, etc.

Concrètement, le flux prioritaire est paramétré avec une règle du type `priority percent x`, et les autres flux sont paramétrés avec des règles du type `bandwidth remaining percent y`.

Pour simplifier le travail des administrateurs réseau, Cisco propose en plus un mécanisme d'*AutoQoS*, avec un profil *VoIP* pré-configuré, et un profil *Entreprise* qui passe d'abord par une phase d'apprentissage.

Mais je vous propose de poursuivre plus avant les manipulations avec les stratégies qui suivent...

9 Policer du traffic entrant sur le routeur

Les différentes stratégies de QoS que nous avons vues précédemment consistaient à paramétrer les flux qui sortent du routeur. Or, dans notre banc de test, nous avons en entrée un flux qui est très agressif et qui sature rapidement les files d'attente de notre routeur (on l'a observé en section 4). (Pensez aux attaques de type déni de service.) Fondamentalement, ça ne sert pas à grand-chose qu'il rentre plus vite qu'il ne peut sortir.

Une façon de régler le problème consiste à borner le trafic en entrée du routeur. Cela s'appelle *policer* le trafic. (Attention, le trafic est *écrêté* et non pas *lissé* comme le fait de la QoS en bandwidth vue en 7).

Définissez une nouvelle `policy-map` que vous pourrez appeler `mon_policer`. Cette politique va travailler sur la classe appelée `class-default` (c.à.d. tous les paquets), que vous paramètrerez ainsi : `police rate 60000 bps`, puis en indiquant `conform-action transmit` et `exceed-action drop`

Ensuite, vous attacherez cette politique à l'interface par laquelle entre les paquets du PC2 (p.ex. `FastEthernet 0/1`, commande `service-policy input ...`. Notez qu'il vous faudra au préalable retirer le mode `fair-queue` qui est par défaut sur l'interface.

- Est-ce que vous observez un changement dans les débits actuellement mesurés ?
- Retirez la politique `ma_qos` actuellement en cours sur l'interface de sortie `Serial 0/0/0`. On se retrouve alors dans le cas du FIFO simple.
- Refaites les mesures de la section 4. Qu'est-ce qui a changé (notamment sur le nombre de paquets reçus/perdus) ?

Notes : Une différence notable entre le *policer* et le *shaping* est que ce dernier gère d'une file d'attente pour absorber les pics de trafic, au lieu d'écrêter. La contre-partie est que ce mécanisme ne peut pas être configurée sur les entrées, et qu'il introduit de la latence.

La documentation de Cisco [3] nous explique que ces deux services sont implémentés selon le principe du *tocken bucket*.

Pour se conformer à un débit nominal paramétré (appelé `CIR`, pour *Committed Information Rate*), l'équipement considère une période de temps (appelée `Tc`, pour *Committed Time*, généralement 125ms, paramétrable, mais Cisco dit qu'il calcule tout seul la valeur optimale), au cours de laquelle il va transmettre sur un temps court mais au débit physique de l'interface une certaine quantité de données (appelée `Bc`, pour *Committed burst*). On a alors la relation simple $\texttt{Bc} = \texttt{Tc} \times \texttt{CIR}$. De plus, dans le cas du shaping qui gère des pics de débit, on s'autorise à transmettre en fait `Bc`+`Be` données (`Be` pour *excess burst*).

Autrement dit, `Tc` est le temps pendant lequel on remplit le seau à jetons, `Bc` est le nombre de jetons, et `Bc`+`Be` la capacité du seau. [4]

10 Mécanisme Weighted Fair Queueing *avancé...*

Revenons sur le mécanisme du *Weighted Fair Queueing* (WFQ) vu à la section 3.

La notion de *poids* vient peser sur des flux et les rendre moins *fair* que d'autre (et donc les privilégier).

3. `http://www.cisco.com/en/US/tech/tk543/tk545/technologies_tech_note09186a00800a3a25.shtml`
4. `http://networklessons.com/quality-of-service/qos-traffic-shaping-explained/`

10.1 Calculs préliminaires

Nous disions donc que le mécanisme WFQ défini par Cisco repose sur l'utilisation du champ TOS de l'entête IP, plus exactement les bits dits *de précédence* qui sont les trois bits de poids fort de l'octet TOS.

Le routeur affecte un poids (*weight*) à chaque flux par assignation des bits de précédence de l'entête IP des paquets qui les portent. Les flux sont priorisés en fonction de ce poids. Le poids est calculé selon la formule : $\frac{4096 \vee 32384}{n+1}$, n étant la valeur des bits de précédence (comprise entre 0 et 7). La valeur 4096 ou 32384 dépend de la version de l'IOS Cisco. Après s'être assuré que le mécanisme de QoS de l'interface n'est pas en simple FIFO, vous pouvez consulter cette valeur avec la commande : `show queue` *`interface`*. Cette commande indique, par flux (ou *conversation*) les informations suivantes : `depth/weight/discard/tail/drop/interleave`.[5]

Avec cette méthode nous allons pouvoir affecter une partie de la bande passante aux flux que nous voulons. La méthode est la suivante : si nous affectons les précédences 1, 2, 3, 4, 5, 6, 7 respectivement à sept flux et la précédence 0 à vingt flux (arbitrairement 20 pour l'exemple) nous pourrons déterminer ce que nous pourrions appeler le *total des précédences* :

$$20 \times (0+1) + 1 \times (1+1) + 1 \times (2+1) + \cdots + 1 \times (7+1) = 55$$

Chaque flux de précédence 1 à 7 prendra $[(n+1)/total]$ parties de la bande passante, soit respectivement 2/55, 3/55, 4/55, ..., 8/55 parties. Les 20 flux de précédence 0 prendront chacun 1/55 de la bande passante.

Question : Si nous avons uniquement deux flux, dont un à la précédence 0 et un à la précédence n, quelle sera la part de bande passante pour chaque flux en fonction de n ? Faites le calcul pour chaque valeur possible de n en considérant une bande passante globale de 64 Kb/s.

Est-ce que vous retrouvez les chiffres mesurés au 3 ?

10.2 Application pratique sur Cisco

Nous reprenons nos deux flux maintenant bien connus.

- Affectation de précédence au flux 1 → 3. Une manière d'effectuer cela est de créer une `route-map` au niveau de configuration de base. Cette route-map portera un nom, par exemple `rmap1`, elle aura pour numéro de séquence 0. On lui fera correspondre l'`access-list` caractérisant le flux 1 → 3 (`match ip address` *`num`*). On affectera à cette route-map la précédence 5 (`set ip precedence 5`).
  Notez que l'on peut également faire cela avec une `policy-map` (voir en annexe A), mais on va changer un peu...
- Affectation de la route-map à une interface entrante par configuration de niveau interface avec la commande `ip policy route-map` *`nom`*. L'interface configurée sera bien évidemment celle recevant le flux à *marquer*.
  Bien évidemment, si vous avez choisi de procéder avec une `policy-map`, il faudra au contraire l'accrocher à l'interface avec `service-policy input`.
- Vérification que l'interface de sortie est bien en mode *weighted fair*. Si ce n'est pas le cas, soit retirer le `service-policy` en cours sur l'interface, et repositionner le WFQ traditionnel avec la commande d'interface `fair-queue` (le routeur va râler un peu que c'est *deprecated*, pas grave), ou autrement refaire une `policy-map` qui ne comporte rien d'autre que la `class-default` sur laquelle on aura positionné le `fair-queue`.
- Lancer le flux de fond sur la machine 2 si ce n'est déjà fait. Ce flux n'étant pas marqué spécialement aura la précédence 0. Lancer le flux de test sur la machine 1, vérifier à l'aide d'un analyseur sur la machine 3 qu'il est bien marqué dans le champ TOS (vérifier dans les options de l'analyseur que celui-ci n'est pas positionné pour interpréter *DSCP-DiffServ* sur le champ TOS : menu `Editer Préférences Protocoles IP`). Relever le débit et vérifier par le calcul si celui-ci ne diffère pas trop. La valeur affichée doit être du même ordre de grandeur que ce que nous donne le calcul.

11 Notes

11.1 QoS Hiérarchique

Les solutions modernes pour faire de la QoS offrent la possibilité de faire de la QoS hiérarchique. Il ne s'agit pas d'un nouvel algorithme, mais plutôt une façon d'utiliser les algorithmes connus, en les encapsulant les uns dans les autres.

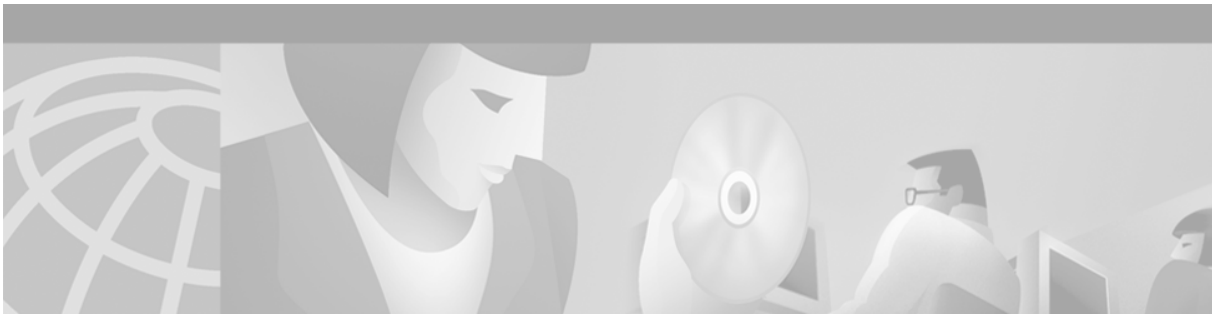
5. Prenez la précaution de générer des flux pour que cette commande ait justement quelque chose à raconter ! Par exemple, laissez tourner le `fluxUDP(emetteur/recepteur)`.

L'idée de base est la suivante : à un niveau donné, on a une politique de QoS qui gères des paquets, et qui décide d'un faire sortir un à un instant donné. Mais ce paquet qui sort, il ne part pas sur le réseau, mais dans une autre stratégie de QoS qui englobe la première, et qui à son tour va décider du paquet qu'elle va faire sortir. Et ainsi de suite jusqu'à ce que le paquet sorte effectivement sur le réseau.

11.2 Et sur Linux...

Là pour le coup on peut dire que le monde du logiciel libre ne s'est pas contenté de réimplémenter quelque chose qui existait dans le domaine commercial, mais a bien été ici un précurseur dans le domaine. Linux proposait historiquement une première implémentation de la QoS qui faisait uniquement du shaping et qui était (d'après les experts) plutôt moche. La ré-implémentation de la QoS dans Linux a le mérite d'être claire, propre, et plutôt efficace (c'est tout en logiciel, même si l'on commence à voir des choses sur carte NetFPGA). Disons-le, au jour d'aujourd'hui les équippementiers "pro" sont encore un peu à la traîne... (imho)

Les quelques stratégies de QoS vu ici sont des *classiques* (elles existent depuis longtemps, et sont implémentées Cisco...). De nombreuses stratégies de QoS ont été implémentées sous Linux (voir le `man tc`, pour *traffic control*). Certaines ont un intérêt surtout académique, d'autre ont un intérêt plus pratique (voir le script `wondershaper`).



Configuring the Modular Quality of Service Command-Line Interface

This section contains the tasks for configuring QoS functionality using the Modular Quality of Service (QoS) Command-Line Interface (CLI) (MQC).

For complete conceptual information, see the chapter "Modular Quality of Service Command-Line Interface Overview" in this book.

For a complete description of the QoS commands in this chapter, refer to the *Cisco IOS Quality of Service Solutions Command Reference*, Release 12.4T. To locate documentation of other commands that appear in this chapter, use the command reference master index or search online.

To identify the hardware platform or software image information associated with a feature, use the Feature Navigator on Cisco.com to search for information about the feature or refer to the software release notes for a specific release. For more information, see the "Identifying Supported Platforms" section in the "Using Cisco IOS Software" chapter in this book.

Modular QoS CLI Configuration Task List

To configure and enable class-based QoS features, perform the tasks described in the following sections. The tasks in the first three sections are required; the task in the fourth section is optional.

- Creating a Traffic Class (Required)
- Creating a Traffic Policy (Required)
- Attaching a Traffic Policy to an Interface (Required)
- Verifying the Traffic Class and Traffic Policy Information (Optional)

See the end of this chapter for the section "Modular QoS CLI Configuration Examples."

Creating a Traffic Class

To create a traffic class, use the **class-map** command. The syntax of the **class-map** command is as follows:

class-map [**match-any** | **match-all**] *class-name*
no class-map [**match-any** | **match-all**] *class-name*

The match-all and match-any Keywords

The **match-all** and **match-any** keywords need to be specified only if more than one match criterion is configured in the traffic class.

The **match-all** keyword is used when *all* of the match criteria in the traffic class must be met in order for a packet to be placed in the specified traffic class.

The **match-any** keyword is used when only *one* of the match criterion in the traffic class must be met in order for a packet to be placed in the specified traffic class.

If neither the **match-all** nor **match-any** keyword is specified, the traffic class will behave in a manner consistent with **match-all** keyword.

About The match not Command

The **match not** command, rather than identifying the specific match parameter to use as a match criterion, is used to specify a match criterion that prevents a packet from being classified as a member of the class. For instance, if the **match not qos-group 6** command is issued while you configure the traffic class, QoS group 6 becomes the only QoS group value that is not considered a successful match criterion. All other QoS group values would be successful match criteria.

Procedure

To create a traffic class containing match criteria, use the **class-map** command to specify the traffic class name. Then use one or more **match** commands to specify the appropriate match criteria. Packets matching the criteria you specify are placed in the traffic class.

Note In the following steps, a number of **match** commands are listed. The specific **match** commands available vary by platform and Cisco IOS release. For the **match** commands available, see the Cisco IOS command reference for the platform and Cisco IOS release you are using.

	Command or Action	Purpose
Step 1	`Router> enable`	Enables privileged EXEC mode.
Step 2	`Router# configure terminal`	Enters global configuration mode.
Step 3	`Router(config)# class-map [match-all | match-any] class-name`	Creates a class to be used with a class map, and enters class-map configuration mode. The class map is used for matching packets to the specified class. **Note** The **match-all** keyword specifies that all match criteria must be met. The **match-any** keyword specifies that one of the match criterion must be met.
	Use one or more of the following **match** commands, as applicable.	
Step 4	`Router(config-cmap)# match access-group {access-group | name access-group-name}`	(Optional) Configures the match criteria for a class map on the basis of the specified access control list (ACL). **Note** Access lists configured with the optional **log** keyword of the **access-list** command are not supported when configuring a traffic class. For more information about the **access-list** command, see the *Cisco IOS IP Application Services Command Reference,* Release 12.4 T.

	Command or Action	Purpose
Step 5	`Router(config-cmap)# match any`	(Optional) Configures the match criteria for a class map to be successful match criteria for all packets.
Step 6	`Router config-cmap)# match class-map class-name`	(Optional) Specifies the name of a traffic class to be used as a matching criterion (for nesting traffic class [nested class maps] within one another).
Step 7	`Router(config-cmap)# match cos cos-number`	(Optional) Matches a packet based on a Layer 2 class of service (CoS) marking.
Step 8	`Router(config-cmap)# match destination-address mac address`	(Optional) Uses the destination Media Access Control (MAC) address as a match criterion.
Step 9	`Router(config-cmap)# match discard-class class-number`	(Optional) Matches packets of a certain discard class.
Step 10	`Router(config-cmap)# match [ip] dscp dscp-value [dscp-value dscp-value dscp-value dscp-value dscp-value dscp-value]`	(Optional) Identifies a specific IP differentiated service code point (DSCP) value as a match criterion. Up to eight DSCP values can be included in one match statement.
Step 11	`Router(config-cmap)# match field protocol protocol-field {eq [mask] | neq [mask] | gt | lt | range range | regex string} value [next next-protocol]`	(Optional) Configures the match criteria for a class map on the basis of the fields defined in the protocol header description files (PHDFs).
Step 12	`Router(config-cmap)# match fr-dlci dlci-number`	(Optional) Specifies the Frame Relay data-link connection identifier (DLCI) number as a match criterion in a class map.
Step 13	`Router(config-cmap)# match input-interface interface-name`	(Optional) Configures a class map to use the specified input interface as a match criterion.
Step 14	`Router(config-cmap)# match ip rtp starting-port-number port-range`	(Optional) Configures a class map to use the Real-Time Protocol (RTP) protocol port as the match criterion.
Step 15	`Router(config-cmap)# match mpls experimental mpls-values`	(Optional) Configure a class map to use the specified value of the Multiprotocol Label Switching (MPLS) experimental (EXP) field as a match criterion.
Step 16	`Router(config-cmap)# match mpls experimental topmost values`	(Optional) Matches the MPLS EXP value in the topmost label.
Step 17	`Router(config-cmap)# match not match-criteria`	(Optional) Specifies the single match criterion value to use as an unsuccessful match criterion.
Step 18	`Router(config-cmap)# match packet length {max maximum-length-value [min minimum-length-value] | min minimum-length-value [max maximum-length-value]}`	(Optional) Specifies the Layer 3 packet length in the IP header as a match criterion in a class map.
Step 19	`Router(config-cmap)# match port-type {routed | switched}`	(Optional) Matches traffic on the basis of the port type for a class map.
Step 20	`Router(config-cmap)# match [ip] precedence precedence-value [precedence-value precedence-value precedence-value]`	(Optional) Identifies IP precedence values as match criteria.

	Command or Action	Purpose
Step 21	`Router(config-cmap)# match protocol protocol-name`	(Optional) Configures the match criteria for a class map on the basis of the specified protocol. **Note** There is a separate **match protocol** (NBAR) command used to configure network-based application recognition (NBAR) to match traffic by a protocol type known to NBAR.
Step 22	`Router(config-cmap)# match protocol citrix [app application-name-string] [ica-tag ica-tag-value]`	(Optional) Configures NBAR to match Citrix traffic.
Step 23	`Router(config-cmap)# match protocol fasttrack file-transfer "regular-expression"`	(Optional) Configures NBAR to match FastTrack peer-to-peer traffic.
Step 24	`Router(config-cmap)# match protocol gnutella file-transfer "regular-expression"`	(Optional) Configures NBAR to match Gnutella peer-to-peer traffic.
Step 25	`Router(config-cmap)# match protocol http [url url-string | host hostname-string | mime MIME-type | c-header-field c-header-field-string | s-header-field s-header-field-string]`	(Optional) Configures NBAR to match Hypertext Transfer Protocol (HTTP) traffic by URL, host, Multipurpose Internet Mail Extension (MIME) type, or fields in HTTP packet headers.
Step 26	`Router(config-cmap)# match protocol rtp [audio | video | payload-type payload-string]`	(Optional) Configures NBAR to match Real-Time Transfer Protocol (RTP) traffic.
Step 27	`Router(config-cmap)# match qos-group qos-group-value`	(Optional) Identifies a specific QoS group value as a match criterion.
Step 28	`Router(config-cmap)# match source-address mac address-destination`	(Optional) Uses the source MAC address as a match criterion.
Step 29	`Router(config-cmap)# match start {l2-start | l3-start} offset number size number {eq | neq | gt | lt | range range | regex string} {value [value2] | [string]}`	(Optional) Configures the match criteria for a class map on the basis of the datagram header (Layer 2) or the network header (Layer 3).
Step 30	`Router(config-cmap)# match tag {tag-name}`	(Optional) Specifies tag type as a match criterion.
Step 31	`Route(config-cmap)# exit`	(Optional) Exits class-map configuration mode.

Creating a Traffic Policy

To configure a traffic policy (sometimes also referred to as a policy map), use the **policy-map** command. The **policy-map** command allows you to specify the traffic policy name and also allows you to enter policy-map configuration mode (a prerequisite for enabling QoS features such as traffic policing or traffic shaping).

Associate the Traffic Policy with the Traffic Class

After using the **policy-map** command, use the **class** command to associate the traffic class (created in the "Creating a Traffic Class" section on page 405) with the traffic policy.

The syntax of the **class** command is as follows:

class *class-name*
no class *class-name*

For the *class-name* argument, use the name of the class you created when you used the **class-map** command to create the traffic class (Step 3 of the "Creating a Traffic Class" section on page 405).

After entering the **class** command, you are automatically in policy-map class configuration mode. The policy-map class configuration mode is the mode used for enabling the specific QoS features.

Procedure

To create a traffic policy (or policy map) and enable one or more QoS features, perform the following steps.

Note This procedure lists many of the commands you can use to enable one or more QoS features. For example, to enable Class-Based Weighted Fair Queuing (CBWFQ), you would use the **bandwidth** command. Not all QoS features are available on all platforms or in all Cisco IOS releases. For the features and commands available to you, see the Cisco IOS documentation for your platform and version of Cisco IOS software you are using.

	Command	Purpose
Step 1	`Router> enable`	Enables privileged EXEC mode.
Step 2	`Router# configure terminal`	Enters global configuration mode.
Step 3	`Router(config)# policy-map` *policy-name*	Creates or specifies the name of the traffic policy and enters policy-map configuration mode.
Step 4	`Router(config-pmap)# class` {*class-name* | **class-default**}	Specifies the name of a traffic class (previously created in the "Creating a Traffic Class" section on page 405) and enters policy-map class configuration mode.
	Use one or more of the following commands to enable the specific QoS feature you want to use.	
Step 5	`Router(config-pmap-c)# bandwidth` {*bandwidth-kbps* | **percent** *percent*}	(Optional) Specifies a minimum bandwidth guarantee to a traffic class in periods of congestion. A minimum bandwidth guarantee can be specified in kbps or by a percentage of the overall available bandwidth.
Step 6	`Router(config-pmap-c)# fair-queue` *number-of-queues*	(Optional) Specifies the number of queues to be reserved for a traffic class.
Step 7	`Router (config-pmap-c)# police` *bps* [*burst-normal*] [*burst-max*] **conform-action** *action* **exceed-action** *action* [**violate-action** *action*]	(Optional) Configures traffic policing.
Step 8	`Router(config-pmap-c)# priority` {*bandwidth-kbps* | **percent** *percentage*} [*burst*]	(Optional) Gives priority to a class of traffic belonging to a policy map.
Step 9	`Router(config-pmap-c)# queue-limit` number-of-*packets*	(Optional) Specifies or modifies the maximum number of packets the queue can hold for a class configured in a policy map.
Step 10	`Router(config-pmap-c)# random-detect` [**dscp-based** | **prec-based**]	(Optional) Enables Weighted Random Early Detection (WRED) or distributed WRED (DWRED).
Step 11	`Router(config-pmap-c)# set atm-clp`	(Optional) Sets the cell loss priority (CLP) bit when a policy map is configured.

	Command	Purpose
Step 12	`Router(config-pmap-c)# set cos {cos-value | from-field [table table-map-name]}`	(Optional) Sets the Layer 2 class of service (CoS) value of an outgoing packet.
Step 13	`Router(config-pmap-c)# set discard-class value`	(Optional) Marks a packet with a discard-class value.
Step 14	`Router(config-pmap-c)# set [ip] dscp {dscp-value | from-field [table table-map-name]}`	(Optional) Marks a packet by setting the differentiated services code point (DSCP) value in the type of service (ToS) byte.
Step 15	`Router(config-pmap-c)# set fr-de`	(Optional) Changes the discard eligible (DE) bit setting in the address field of a Frame Relay frame to 1 for all traffic leaving an interface.
Step 16	`Router(config-pmap-c)# set precedence {precedence-value | from-field [table table-map-name]}`	(Optional) Sets the precedence value in the packet header.
Step 17	`Route(config-pmap-c)# set mpls experimental value`	(Optional) Designates the value to which the MPLS bits are set if the packets match the specified policy map.
Step 18	`Router (config-pmap-c)# set qos-group {group-id | from-field [table table-map-name]}`	(Optional) Sets a QoS group identifier (ID) that can be used later to classify packets.
Step 19	`Router(config-pmap-c)# service-policy policy-map-name`	(Optional) Specifies the name of a traffic policy used as a matching criterion (for nesting traffic policies [hierarchical traffic policies] within one another).
Step 20	`Router(config-pmap-c)# shape {average | peak} mean-rate [burst-size [excess-burst-size]]`	(Optional) Shapes traffic to the indicated bit rate according to the algorithm specified.
Step 21	`Router(config-pmap-c)# exit`	(Optional) Exits policy-map class configuration mode.

Attaching a Traffic Policy to an Interface

To attach a traffic policy to an interface, use the **service-policy** command. The **service-policy** command also allows you to specify the direction in which the traffic policy should be applied (either on packets coming into the interface or packets leaving the interface).

The **service-policy** command syntax is as follows:

service-policy {**input** | **output**} *policy-map-name*
no service-policy {**input** | **output**} *policy-map-name*

Procedure

To attach a traffic policy to an interface, perform the following steps.

Note Depending on the platform and Cisco IOS release you are using, a traffic policy can be attached to an ATM permanent virtual circuit (PVC) subinterface, a Frame Relay data-link connection identifier (DLCI), or another type of interface.

	Command	Purpose
Step 1	`Router> enable`	Enables privileged EXEC mode.
Step 2	`Router# configure terminal`	Enters global configuration mode.
Step 3	`Router(config)# interface serial0`	Configures an interface type and enters interface configuration mode.
Step 4	`Router(config-if)# service-policy output [type access-control] {input | output} policy-map-name`	Attaches a policy map to an interface.
Step 5	`Router (config-if)# exit`	(Optional) Exits interface configuration mode.

Note Multiple traffic policies on tunnel interfaces and physical interfaces are not supported if the interfaces are associated with each other. For instance, if a traffic policy is attached to a tunnel interface while another traffic policy is attached to a physical interface with which the tunnel interface is associated, only the traffic policy on the tunnel interface works properly.

Verifying the Traffic Class and Traffic Policy Information

To display and verify the information about a traffic class or traffic policy, perform the following steps.

	Command	Purpose
Step 1	`Router> enable`	Enables privileged EXEC mode.
Step 2	`Router# show class-map [type {stack | access-control}] [class-map-name]`	(Optional) Displays all class maps and their matching criteria.
Step 3	`Router# show policy-map policy-map class class-name`	(Optional) Displays the configuration for the specified class of the specified policy map.
Step 4	`Router# show policy-map policy-map`	(Optional) Displays the configuration of all classes for a specified policy map or all classes for all existing policy maps.
Step 5	`Router# show policy-map interface [type access-control] type number [vc [vpi/] vci] [dlci dlci] [input | output]`	(Optional) Displays the packet statistics of all classes that are configured for all service policies either on the specified interface or subinterface or on a specific permanent virtual circuit (PVC) on the interface.
Step 6	`Router# exit`	(Optional) Exits privileged EXEC mode.

Modular QoS CLI Configuration Examples

This section provides the Modular QoS CLI configuration examples:

- Traffic Classes Defined Example
- Traffic Policy Created Example
- Traffic Policy Attached to an Interface Example
- match not Command Example

- Default Traffic Class Configuration Example
- class-map match-any and class-map match-all Commands Example
- Traffic Class as a Match Criterion (Nested Class Maps) Example
- Traffic Policy as a QoS Policy (Hierarchical Traffic Policies) Example

For information on how to configure the QoS functionality with the Modular QoS CLI, see the section "Modular QoS CLI Configuration Task List" in this chapter.

Traffic Classes Defined Example

In the following example, two traffic classes are created and their match criteria are defined. For the first traffic class called class1, access control list (ACL) 101 is used as the match criterion. For the second traffic class called class2, ACL 102 is used as the match criterion. Packets are checked against the contents of these ACLs to determine if they belong to the class.

```
Router(config)# class-map class1
Router(config-cmap)# match access-group 101
Router(config-cmap)# exit

Router(config)# class-map class2
Router(config-cmap)# match access-group 102
Router(config-cmap)# exit
```

Traffic Policy Created Example

In the following example, a traffic policy called policy1 is defined to contain policy specifications for the two classes—class1 and class2. The match criteria for these classes were defined in the traffic classes (see the section "Creating a Traffic Class" in this chapter).

For class1, the policy includes a bandwidth allocation request and a maximum packet count limit for the queue reserved for the class. For class2, the policy specifies only a bandwidth allocation request.

```
Router(config)# policy-map policy1
Router(config-pmap)# class class1
Router(config-pmap-c)# bandwidth 3000
Router(config-pmap-c)# queue-limit 30
Router(config-pmap)# exit

Router(config-pmap)# class class2
Router(config-pmap-c)# bandwidth 2000
Router(config-pmap)# exit
```

Traffic Policy Attached to an Interface Example

The following example shows how to attach an existing traffic policy (which was created in the preceding "Traffic Policy Created Example" section) to an interface. After you define a traffic policy with the **policy-map** command, you can attach it to one or more interfaces by using the **service-policy** command in interface configuration mode. Although you can assign the same traffic policy to multiple interfaces, each interface can have only one traffic policy attached at the input and only one traffic policy attached at the output.

```
Router(config)# interface e1/1
Router(config-if)# service-policy output policy1
Router(config-if)# exit

Router(config)# interface fa1/0/0
Router(config-if)# service-policy output policy1
Router(config-if)# exit
```

match not Command Example

The **match not** command is used to specify a specific QoS policy value that is not used as a match criterion. When using the **match not** command, all other values of that QoS policy become successful match criteria.

For instance, if the **match not qos-group 4** command is issued in class-map configuration mode, the specified class will accept all QoS group values except 4 as successful match criteria.

In the following traffic class, all protocols except IP are considered successful match criteria:

```
Router(config)# class-map noip
Router(config-cmap)# match not protocol ip
Router(config-cmap)# exit
```

Default Traffic Class Configuration Example

Unclassified traffic (traffic that does not meet the match criteria specified in the traffic classes) is treated as belonging to the default traffic class.

If the user does not configure a default class, packets are still treated as members of the default class. However, by default, the default class has no QoS features enabled. Therefore, packets belonging to a default class have no QoS functionality. These packets are placed into a first-in, first-out (FIFO) queue managed by tail drop. (Tail drop is a means of avoiding congestion that treats all traffic equally and does not differentiate between classes of service. Queues fill during periods of congestion. When the output queue is full and tail drop is in effect, packets are dropped until the congestion is eliminated and the queue is no longer full).

The following example configures a traffic policy for the default class of the traffic policy called policy1. The default class (which is always called class-default) has these characteristics: 10 queues for traffic that does not meet the match criteria of other classes whose policy is defined by the traffic policy policy1, and a maximum of 20 packets per queue before tail drop is enacted to handle additional queued packets.

```
Router(config)# policy-map policy1
Router(config-pmap)# class class-default
Router(config-pmap-c)# fair-queue 10
Router(config-pmap-c)# queue-limit 20
```

For moredetailed information on the preceding commands, refer to the *Cisco IOS Quality of Service Solutions Command Reference*, Release 12.4T.

class-map match-any and class-map match-all Commands Example

This section illustrates the difference between the **class-map match-any** command and the **class-map match-all** command. The **match-any** and **match-all** options determine how packets are evaluated when multiple match criteria exist. Packets must either meet all of the match criteria (**match-all**) or one of the match criterion (**match-any**) in order to be considered a member of the traffic class.

The following example shows a traffic class configured with the **class-map match-all** command:

```
Router(config)# class-map match-all cisco1
Router(config-cmap)# match protocol ip
Router(config-cmap)# match qos-group 4
Router(config-cmap)# match access-group 101
```

If a packet arrives on a router with traffic class called cisco1 configured on the interface, the packet is evaluated to determine if it matches the IP protocol, QoS group 4, *and* access group 101. If all three of these match criteria are met, the packet matches traffic class cisco1.

The following example shows a traffic class configured with the **class-map match-any** command:

```
Router(config)# class-map match-any cisco2
Router(config-cmap)# match protocol ip
Router(config-cmap)# match qos-group 4
Router(config-cmap)# match access-group 101
```

In traffic class called cisco2, the match criteria are evaluated consecutively until a successful match criterion is located. The packet is first evaluated to the determine whether IP protocol can be used as a match criterion. If IP protocol can be used as a match criterion, the packet is matched to traffic class cisco2. If IP protocol is not a successful match criterion, then QoS group 4 is evaluated as a match criterion. Each criterion is evaluated to see if the packet matches that criterion. Once a successful match occurs, the packet is classified as a member of traffic class cisco2. If the packet matches none of the specified criteria, the packet is classified as a member of the traffic class.

Note that the **class-map match-all** command requires that all of the match criteria must be met in order for the packet to be considered a member of the specified traffic class (a logical AND operator). In the example, protocol IP AND QoS group 4 AND access group 101 have to be successful match criteria. However, only one match criterion must be met for the packet in the **class-map match-any** command to be classified as a member of the traffic class (a logical OR operator). In the example, protocol IP OR QoS group 4 OR access group 101 have to be successful match criterion.

Traffic Class as a Match Criterion (Nested Class Maps) Example

There are two reasons to use the **match class-map** command. One reason is maintenance; if a large traffic class currently exists, using the traffic class match criterion is simply easier than retyping the same traffic class configuration.

The more common reason for the **match class-map** command is to allow users to use match-any and match-all statements in the same traffic class. If you want to combine match-all and match-any characteristics in a traffic policy, create a traffic class using one match criterion evaluation instruction (either match any or match all) and then use this traffic class as a match criterion in a traffic class that uses a different match criterion type.

Here is a possible scenario: Suppose A, B, C, and D were all separate match criterion, and you wanted traffic matching A, B, or C and D (A or B or [C and D]) to be classified as belonging to the traffic class. Without the nested traffic class, traffic would either have to match all 4 of the match criterion (A and B and C and D) or match any of the match criterion (A or B or C or D) to be considered part of the traffic class. You would not be able to combine "and" (match-all) and "or" (match-any) statements within the traffic class, and you would therefore be unable to configure the desired configuration.

The solution: Create one traffic class using match-all for C and D (which we will call criterion E), and then create a new match-any traffic class using A, B, and E. The new traffic class would have the correct evaluation sequence (A or B or E, which would also be A or B or [C and D]). The desired traffic class configuration has been achieved.

The only method of mixing match-all and match-any statements in a traffic class is through the use of the traffic class match criterion.

Nested Traffic Class for Maintenance Example

In the following example, the traffic class called class1 has the same characteristics as traffic class called class2, with the exception that traffic class class1 has added a destination address as a match criterion. Rather than configuring traffic class class1 line by line, you can enter the **match class-map class2** command. This command allows all of the characteristics in the traffic class called class2 to be included in the traffic class called class1, and you can simply add the new destination address match criterion without reconfiguring the entire traffic class.

```
Router(config)# class-map match-any class2
Router(config-cmap)# match protocol ip
Router(config-cmap)# match qos-group 3
Router(config-cmap)# match access-group 2
Router(config-cmap)# exit

Router(config)# class-map match-all class1
Router(config-cmap)# match class-map class2
Router(config-cmap)# match destination-address mac 00.00.00.00.00.00
Router(config-cmap)# exit
```

Nested Traffic Class to Combine match-any and match-all Characteristics in One Traffic Class Example

The only method of including both match-any and match-all characteristics in a single traffic class is to use the **match class-map** command. To combine match-any and match-all characteristics into a single class, a traffic class created with the match-any instruction must use a class configured with the match-all instruction as a match criterion (through the **match class-map** command), or vice versa.

The following example shows how to combine the characteristics of two traffic classes, one with match-any and one with match-all characteristics, into one traffic class with the **match class-map** command. The result of traffic class class3 requires a packet to match one of the following three match criteria to be considered a member of traffic class class4: IP protocol *and* QoS group 4, destination MAC address 00.00.00.00.00.00, or access group 2.

In this example, only the traffic class called class4 is used with the traffic policy called policy1.

```
Router(config)# class-map match-all class3
Router(config-cmap)# match protocol ip
Router(config-cmap)# match qos-group 4
Router(config-cmap)# exit

Router(config)# class-map match-any class4
Router(config-cmap)# match class-map class3
```

```
Router(config-cmap)# match destination-address mac 00.00.00.00.00.00
Router(config-cmap)# match access-group 2
Router(config-cmap)# exit

Router(config)# policy-map policy1
Router(config-pmap)# class class4
Router(config-pmap-c)# police 8100 1500 2504 conform-action transmit exceed-action
set-qos-transmit 4
Router(config-pmap-c)# exit
```

Traffic Policy as a QoS Policy (Hierarchical Traffic Policies) Example

A traffic policy can be nested within a QoS policy when the **service-policy** command is used in policy-map class configuration mode. A traffic policy that contains a nested traffic policy is called a hierarchical traffic policy.

A hierarchical traffic policy contains a child and a parent policy. The child policy is the previously defined traffic policy that is being associated with the new traffic policy through the use of the **service-policy** command. The new traffic policy using the preexisting traffic policy is the parent policy. In the example in this section, traffic policy called child is the child policy and traffic policy called parent is the parent policy.

Hierarchical traffic policies can be attached to subinterfaces, Frame Relay PVCs, and ATM PVCs. A hierarchical traffic policy is particularly beneficial when configuring VIP-based distributed FRF.12 (and higher) PVCs. When hierarchical traffic policies are used, a single traffic policy (with a child and a parent policy) can be used to shape and prioritize PVC traffic. In the following example, the child policy is responsible for prioritizing traffic and the parent policy is responsible for shaping traffic. In this configuration, the parent policy allows packets to be sent from the interface, and the child policy determines the order in which the packets are sent.

```
Router(config)# policy-map child
Router(config-pmap)# class voice
Router(config-pmap-c)# priority 50

Router(config)# policy-map parent
Router(config-pmap)# class class-default
Router(config-pmap-c)# shape average 10000000
Router(config-pmap-c)# service-policy child
```

With the exception that the values associated with the **priority** and **shape** commands can be modified, the example is the required configuration for PVCs using FRF.12 (or higher). The value used with the **shape** command is provisioned from the committed information rate (CIR) value from the service provider. For more information about FRF.12 (or higher) PVCs, see the *Cisco IOS Wide-Area Networking Configuration Guide*, Release 12.4.

For more information about the **service-policy** command, see the *Cisco IOS Quality of Service Solutions Command Reference*, Release 12.4T.