Une brève introduction aux mécanismes pour la Qualité de Service

Chaput Emmanuel

2011-2012

Notes :

Notes :

Introduction

Notes :

Le problème

Réseaux à commutation de paquets : multiplexage temporel asynchrone

Altération du profil temporel des flots de paquets
Inéquité lors de contention entre les taux de perte des flots
Pas de traitement spécifique en fonction de besoins différents

Comportement *best effort*

- Le réseau fait ce qu'il peut
- Sans garantir quoi que ce soit

Notes :

Objectifs

Fournir des mécanismes permettant d'assurer de la "Qualité de service" dans les réseaux de paquets.

- Distribuer "équitablement" le débit des liens
  - Quelle granularité ?
  - Quelle équité ?
  - Quelle métrique ?
- Isoler les trafics entre eux
  - Éviter que le comportement d'un dégrade le service des autres
- Accroître les performances du service fourni

Délai de bout en bout
Gigue sur un flux
Débit sur le chemin
Perte de paquets

Notes :

Différents types de métriques

- Comment définir les métriques ?
  - Ce qui a une signification applicative n'en a pas toujours au niveau réseau (ou transport)
- Difficulté d'évaluer les métriques
  - Fenêtre glissante, moyenne mobile, ...
- Divers comportements sur un chemin
  - Métrique additive
    - $M_{r_1,r_2} = M_{r_1} + M_{r_2}$
    - *pe* délai, gigue
  - Métrique concave
    - $M_{r_1,r_2} = min(M_{r_1}, M_{r_2})$
    - *pe* débit
  - Métrique multiplicative
    - $M_{r_1,r_2} = M_{r_1} \times M_{r_2}$
    - *pe* probabilité de non perte

Notes :

Mécanismes =/= architecture

Mise en œuvre de la qualité de service

- Une architecture (*IntServ*, *DiffServ*, . . .)
- Des protocoles (RSVP, DSCP, . . .)
- Des mécanismes (ordonnancement, lissage, . . .)

Les mécanismes sont la “cheville ouvrière” de la QoS

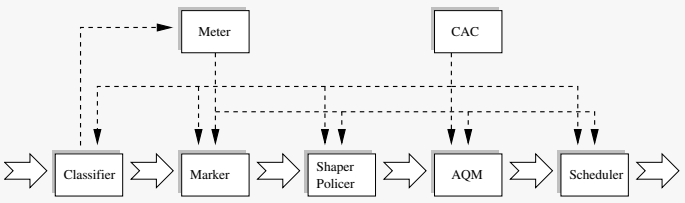
- Mécanismes non liés à une architecture bien que
- Souvent implantés pour assurer des services définis dans le cadre d’une architecture

Notes :

Schéma général

Notes :

Les fonctions



Présence et localisation dépendantes

- De l’architecture
  - *IntServ [3], DiffServ [1], . . .*
- De l’entité dans l’architecture
  - Routeur de frontière, de cœur, . . .

Notes :

Classifier

Objectif : réaliser une ségrégation des paquets

- Ségrégation par flot (à la *IntServ*)
- Ségrégation par classe (à la *DiffServ*)
- Fondée sur des informations contenues dans les paquets
- Nécessaire du fait de l'absence de connexion IP

Notes :

Meter

Objectif : mesurer les caractéristiques (temporelles) d'un ensemble de paquets

- Caractéristiques selon critères de ségrégation
- Vérification de conformité
  - À un profil de trafic négocié
- Comparaison à des seuils
  - Éventuellement négociés
  - Éventuellement dynamiques

Notes :

Marker

Objectif : ajouter des informations pertinentes au paquet

- Emplacement prévu dans les entêtes
  - Champ ToS d'IPv4, ...
- Encapsulation supplémentaire
  - Label MPLS, ...
- Structure de données associée
  - Traitement local
- Pemet un traitement plus rapide en aval

Notes :

Shaper

Objectif : Lisser le trafic

- Le rendre conforme à un profil temporel
- Évite les bursts en aval
- Introduction de “délai” entre paquets
- Implanté dans le scheduler

Notes :

Policer

Objectif : contraindre le trafic

- Le rendre conforme à un contrat
- Destruction/remarquage de paquets
- Pas de lissage
- Isolation des trafics

Notes :

Active Queue Management

Objectif : diminuer les risques de congestion

- Gestion active des files d’attente [2]
- Éviter les files d’attente pleines
  - Capacité à accueillir les bursts
  - Minimiser le temps de traversée
- Prévenir la congestion plutôt que la subir
  - Définition de taux de remplissage seuils
  - Actions curatives (anticipées) ou préventives
  - Actions plus équitables

Notes :

Scheduler

Objectif : réaliser le multiplexage temporel

- Utiliser “équitablement” le débit disponible
- Appliquer les choix faits lors des étapes précédentes
- Implantation intègre les fonctions de shaper/policer
- Éventuellement non work conserving (*cf* page suivante)

Notes :

Ordonnancement *Work Conserving*

- Ordonnancement *Work Conserving*
  - Un paquet en attente est émis dès que le support est disponible
  - Utiliser au mieux la bande passante
- Ordonnancement *Non Work Conserving*
  - Un paquet en attente doit de plus être éligible pour être émis
  - Attendre un paquet plus prioritaire
  - Mettre en place du lissage

Notes :

Connexion Admission Control

Objectif : Vérifier qu’un contrat peut être assuré

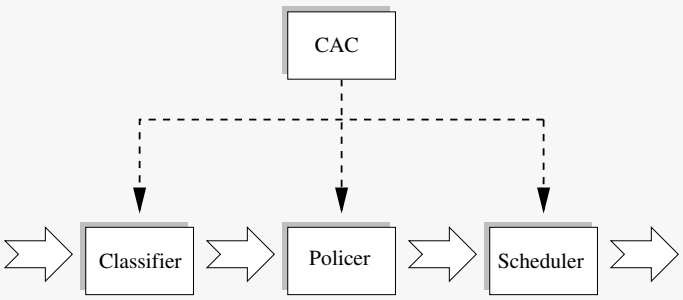
- Négociation de paramètres de QoS
- Associé à de la réservation de ressource
- Nécessite un protocole orienté connexion ou un équivalent
- Peut permettre l’isolation des trafics

Notion de SLA

- *Service Level Agreement*
- Accord entre le client et l’opérateur
- Description des paramètres de QoS

Notes :

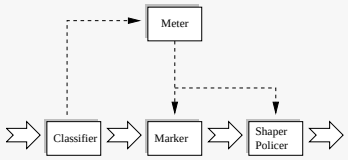
Un exemple : *IntServ*



- Ressources réservées par RSVP

Notes :

Un exemple : *DiffServ* (edge router)



- Ségrégation évoluée (MF-classifier)
- Marquage simple (DSCP) pour le domaine
- Policing (ingress router)
- Shaping (egress router)

Notes :

Un exemple : *DiffServ* (core router)

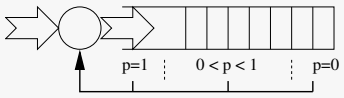
- Ségrégation simple
- Comportement conforme à un PHB

Notes :

Exemples d'AQM

Notes :

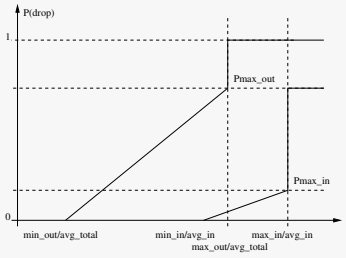
Random Early Detection



- Observation de l'état de la file
- Définition d'une probabilité en fonction de l'état
- Marquage/destruction de paquets choisis aléatoirement
- Désynchronisation des TCPs
- Première proposition en 1993 [7]

Notes :

RED with In and Out



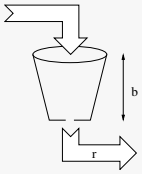
- Double RED [4]
  - Paquets marqués *in* ou *out*
- Perdre prioritairement des paquets hors-profil
- Indépendant du marquage des paquets
- Quel paramétrage ?

Notes :

Exemples de shaper

Notes :

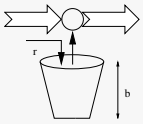
Leaky bucket



- Les paquets entrent dans le "seau"
- Sortie du seau à débit borné (`r`)
- Ici, le leaky bucket est utilisé pour faire du *shaping*
- Si capacité (`b`) finie : policing

Notes :

Token bucket



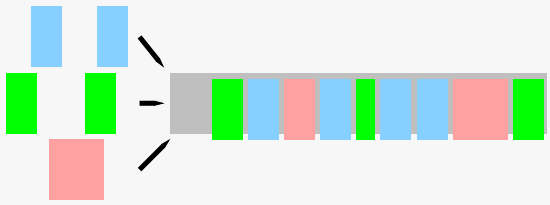
- Chaque paquet (octet) consomme un jeton
- Bursts possibles en sortie (limités par `b`)
- Paquets hors profil (pas de jeton)
  - Retardés (shaper)
  - Détruits/déclassés (policer)
- Le *leaky bucket* peut également être utilisé de la sorte

Notes :

Algorithmes d'ordonnancement

Notes :

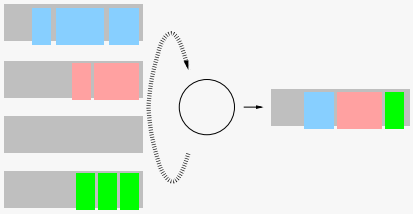
First Come First Served



- Tail drop
- Simple à implanter
- Pas de distinction entre trafics

Notes :

Round Robin



- Classification du trafic
  - Besoin d'un mécanisme qui place les paquets dans des files virtuelles
- Simple à implanter
- Certaine isolation entre les trafics
  - Un paquet est émis à chaque tour
- Équitable en paquets : en débit ?

Notes :

Deficit Round Robin

- Constat : packet round robin non équitable si taille variable
- Service pondéré : Q_i bits *max* par tour pour la file *i*
- Prise en compte du déficit (Q_i - service) au tour suivant
- Compatible avec SFQ
- [14]

Notes :

Deficit Round Robin : algorithme

Q_i nombre max de bits du flux *i* transmis par cycle

DC_i déficit du flux *i*

L_i^k nombre de bits du paquet *k* du flux *i*

Algorithme

- $DC_i = 0$
- À chaque cycle, pour chaque flux *i*
  - $DC_i \leftarrow DC_i + Q_i$
  - tant qu'il existe un paquet *k* en attente tel que $L_i^k \leq DC_i$
    - émettre le paquet
    - $DC_i \leftarrow DC_i - L_i^k$

Notes :

Fair Queuing

- Simulation d'un round-robin fluide (bit à bit)
- Réception d'un paquet : calcul de sa date de départ théorique
- Émission ordonnée selon les dates théoriques
- [11] [12] [6]

Notes :

Fair Queuing : principe

Notation

- $R(t)$ nombre de cycles du round robin à t
- $N_a(t)$ nombre de flux actifs à t
- a_i^k date d'arrivée du paquet i du flux k
- L_i^k nombre de bits du paquet i du flux k
- S_i^k et F_i^k dates de début et fin de service du paquet i du flux k *en nombre de cycles*

Propriétés

$N_a(t) = card(\{k, F^k_{max(i,a_i^k \leq t)} \geq t\})$

$F_i^k = S_i^k + L_i^k$

$S_i^k = max(F_{i-1}^k, R(a_i^k))$ (respect de l'ordre !)

$\frac{\partial R(t)}{\partial t} = \frac{1}{N_a(t)}$ si 1 bit par unité de temps

Notes :

Fair Queuing : algorithme

- Réception d'un paquet : calcul de sa date de fin de service par un round-robin bit à bit
- Support libre : émission du paquet de plus faible F_i^k
- On ne s'intéresse pas aux dates réelles puisque $r(t)$ est strictement croissante
- Durée de cycle variable (arrivée d'un paquet sur un flux inactif)
  - Dates (réelles) de départ théorique à recalculer
  - Dates en $R(t)$ inchangées !

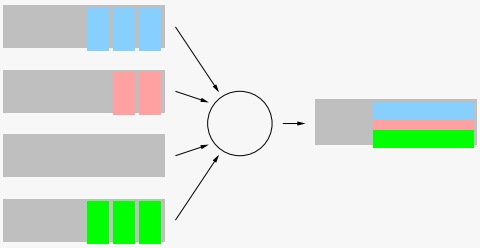
Notes :

Stochastic Fair Queuing

- Constat du Fair Queuing : lourd à implanter à haut débit (gestions des files)
- Flots répartis selon une méthode de hachage
- Hachage modifiée régulièrement
- [10] [9]

Notes :

Generalized Processor Sharing (GPS)



- Service fluide pondéré (débit de sortie ϕ_i pour flux i)
- Chacun obtient $\frac{\phi_i}{\sum \phi_i}$
- Extrèmement équitable
- Impossible à implanter

Notes :

Packet GPS

- Weighted Fair Queuing (pondéré par ϕ_i, où $\sum \phi_i = 1$)
- Pour chaque paquet : estimation de sa date de sortie sur un GPS
- Paquets émis dans l'ordre de ces dates
- [6] [13]

Notes :

Packet GPS : principes

Notations du Fair Queuing plus

$V(t)$ temps virtuel évoluant proportionnellement à $\frac{1}{\sum_{i \in \{actifs\}} \phi_i}$ si 1 bit par unité de temps

Propriétés

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

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

$\frac{\partial V(t)}{\partial t} = \frac{1}{\sum_{i \in \{actifs\}} \phi_i}$ si 1 bit par unité de temps

Notes :

Virtual Clock

- Inspiré du TDM
- Calcul d'une date de départ théorique
- Date calculée en supposant un débit constant
- Émission ordonnée selon les dates théoriques
- Isolation des flux
- [15]

Notes :

Virtual Clock : principes

Notations

r_i débit moyen associé au flot i ($i \in [1 \ldots n]$)

S_i^k et VC_i^k dates de début et fin de service du paquet i du flux k

a_i^k date d'arrivée du paquet i du flux k

L_i^k nombre de bits du paquet i du flux k

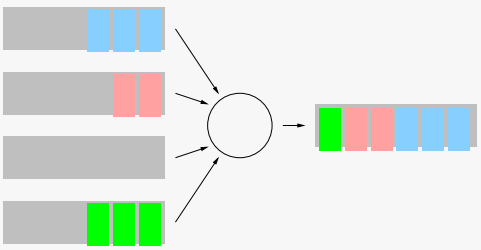
Propriétés

$VC_i^k = S_i^k + \frac{L_i^k}{r_i}$

$S_i^k = max(F_{i-1}^k, a_i^k)$ (respect de l'ordre !)

Notes :

Priority Queuing



- Chaque file est dotée d'une priorité
- File de plus faible priorité servie si prioritaire vide
- Généralement pas de préemption
- Risque de famine

Notes :

Un exemple : l'implantation dans Linux

Notes :

Architecture générale

Notes :

Objectifs de l'architecture

Lissage des trafics : supression des salves, . . .
Ordonnancement des paquets selon diverses stratégies
Policing des trafics entrants
Destruction des paquets hors-limites

Notes :

Architecture de base

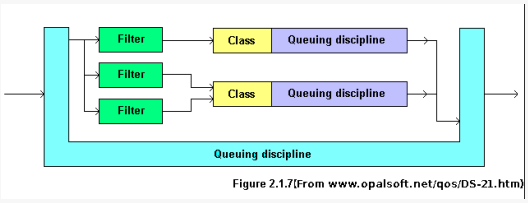


Figure 2.1.7(From www.opalsoft.net/qos/DS-21.htm)

Notes :

Trois grandes familles

Les architectures fournissent un cadre
Les ordonnanceurs . . . ordonnancent
Les outils annexes offrent d'autres fonctionnalités

Notes :

Les architectures

Notes :

Les architectures

- CBQ ancètre historique proposé par S. Floyd et Van Jacobson [8]
- CSZ partiellement implanté [5]
- HTB autre implantation des concepts de CBQ

Notes :

Clark Shenker Zhang

- Description de trafic et contrôle d'admission
  - Token bucket ou débit max
- Classification des flux
  - Temps réel avec contraintes strictes
  - Temps réel "mou"
  - Best effort
- Ordonnancement
  - Isolation pour temps réel (WFQ)
  - Technique non optimale pour le reste (FIFO, FIFO+)
- Quelle garantie de bout en bout ?

Notes :

Clark Shenker Zhang (implantation)

- Version simplifiée
  - Pas de contrôle d'admission
  - FIFO + priorités pour non temps réel
  - WFQ version paquet simplifiée

Notes :

Hierarchical Token Bucket, CBQ

- Issu des travaux de Floyd et V. Jacobson
- Hiérarchisation des trafics
- Cadre générique pour partager les liens :
  - Débit par classe
  - Ordonnancement avec/sans congestion
  - Partage du débit non utilisé

Notes :

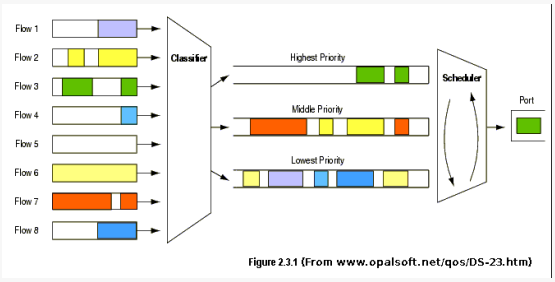
Hierarchical Token Bucket, CBQ (implantation)

Bien plus simple que CSZ

- Estimation des ressources utlisées
- Détermination des classes à contrôler
- Utilisation du débit résiduel

Notes :

Prio



Notes :

Prio

- Plusieurs niveaux de priorités
- Classifieurs
- Correspondance avec le champ `TOS`

Notes :

Les ordonnanceurs

Notes :

First In, First Out

- Premier arrivé premier servi
- Limite sur longueur de file
- Version paquet et version octet

Notes :

FIFO rapide

- FIFO à trois priorités
- Implantation du TOS IPv4
- Plus rapide que PRIO + FIFO

Notes :

Token Bucket Filter

Classique. Paramètres

- Taille du seau, débit
- Taille maximale de la file
- Burst maximal

Notes :

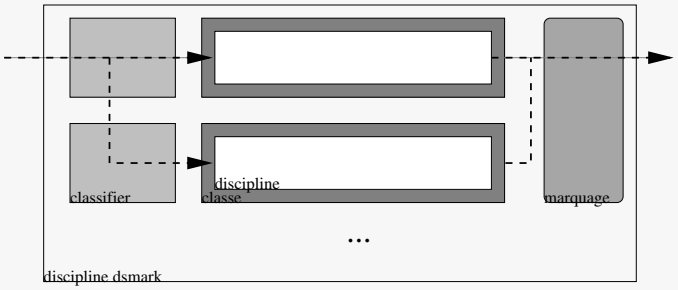
Stochastic Fairness Queuing

- Distribution des flux sur des files
- Tourniquet sur les files
- Hashage pour répartition

Notes :

La mécanique *DiffServ*

Notes :



Notes :

Le plus haut niveau

Outil `dsmark`
- Cadre général
- (Re)marquage des paquets
- Consultation du `DSCP`

Notes :

Le filtrage en entrée

Classifieur `tcindex`

- Choix d'une classe en fonction du marquage
- Définition d'un nouveau champ
- Utilisation délicate

Notes :

Le cas du champ `tc_index`

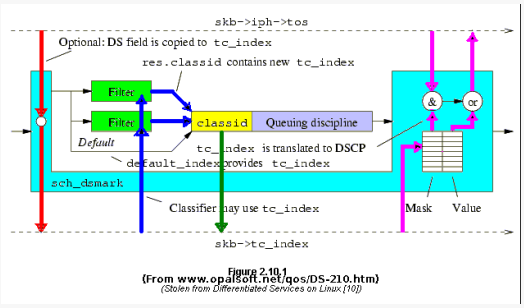


Figure 2.10.1
(From www.opalsoft.net/qos/DS-210.htm)
(Stolen from Differentiated Services on Linux [10])

Notes :

Les outils annexes

Notes :

Random Early Detection

- Destruction (marquage) probabiliste
- Interaction avec ECN possible
- [2]

Notes :

Generalized RED

- Files d'attentes virtuelles
- Caractéristiques à la RED distinctes
- Possibilité de priorité
- Pour les BAs de type AF de *DiffServ*

Notes :

Policing de trafic

Notes :

Policing de trafic

- Discipline `ingress`
- Policing par des filtres
- Peu évolutif

Notes :

Intermediate Queuing device

- Périphérique virtuel
- Reçoit le trafic de périphériques réels
- Ré-injecte le trafic dans le système

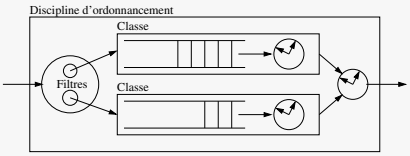
Notes :

Un exemple simple

Notes :

Structure visée

- Différencier deux types de trafic
- Attribuer une proportion de débit à chacun
- Permettre à l'un d'utiliser ce que l'autre ne veut pas



Notes :

Mise en place d'un ordonnanceur

- Utilisation, par exemple de l'architecture HTB
- Configuration sur l'interface de sortie
- Définition d'une file par défaut

```
# tc qdisc add dev eth1 root handle 1: htb default
11
```


Notes :

Classification des trafics

- Par exemple en fonction de l'adresse destination
- Paquets correspondants aiguillés dans une file spécifique

```
# tc filter add dev eth1 protocol ip parent 1:0 prio
1 u32 match  ip dst 192.168.16.0/24 flowid 1:10
```


Notes :

Paramétrage d'une classe de trafic

- Définition d'un débit maximale (en cas de contention)
- Définition d'un débit crête (si ressources disponibles)

```
# tc class change dev eth1 classid 1:11 htb rate
40kbps ceil 100kbps
```


Notes :

Un cas d'étude

Notes :

Notes :

Architecture de qualité de service dynamique

Définition d'une architecture de qualité de service

- Utilisation de l'implantation des mécanismes DiffServ dans le noyau Linux ;
- Couplage avec les couches hautes et basses
  - QoS server ;
  - Couche MAC (allocation) ;
- Évaluation des algorithmes disponibles (tâche 2).

Notes :

Exemple de mise en place

Notes :

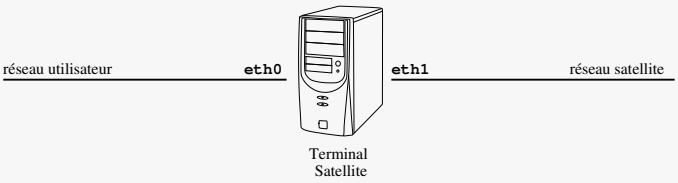
Simulation du lien

- Outil de simulation de réseau `netem` pour pertes et retard
- Ordonnancement pour débit

```
# tc qdisc add dev eth0 root handle 1: netem delay
240ms 1ms loss 0.00001% 90%
# tc qdisc add dev eth0 parent 1: handle 2: tbf rate
512kbit
```


Notes :

Configuration du ST



Notes :

Classification des trafics

Possibilités MFC très vastes

- Tous les champs de l'en-tête paquet
- Nombreux autres champs
- Résultat routage
- Résultat policing

Notes :

Classification des trafics

Sur un routeur d'entrée.
Création de l'outil de marquage

```
# tc qdisc add eth0 root handle 1: dsmark indices 16
```

Marquage en fonction de la classe (*pe* classe 2)

```
# tc class change dev eth0 classid 1:2 dsmask 0x03
value 0xb8
```

Aiguillage des paquets vers les classes

```
# tc filter add dev eth0 parent 1: u32 match ip dst
1.2.3.0/24 flowid 1:2
```


Notes :

Policing

- `ingress` et filtres

```
# tc qdisc add dev eth0 handle ffff: ingress
# tc filter add dev eth0 parent ffff: protocol
ip u32 match ip src 1.2.3.0/24 police rate
128kbit burst 10k drop flowid :1
```

- IMQ si nécessaire

```
# ip link set imq0 up
# qdisc add dev imq0 root handle 1: dlb
# iptables -t mangle -A PREROUTING -i eth1 -p
udp -j IMQ --todev 0
```


Notes :

Ordonnancement : stratégie

```
# tc qdisc add dev eth1 handle 1: root dsmark
indices 64 set_tc_index
# tc filter add dev eth1 parent 1: prio 1 tcindex
mask 0xfc shift 3
```

Classe	*dscp*	*indice*
*af*11	001010*xx*	0*x*05
...	...	...
*af*13	001110*xx*	0*x*07
...	...	...
ef	101110*xx*	0*x*17
...	...	...

Notes :

Ordonnancement : organisation

Utilisation d'une hiérarchie de classes

```
# tc qdisc add dev eth1 parent 1: handle 2: htb rate
2mbit burst 16k
# tc filter add dev eth1 parent 2: protocol ip prio
1 tcindex mask 0x1c shift 3 pass_on
```

Trafic	*Classe*
*af*1*n*	0*x*1*n*
*af*2*n*	0*x*2*n*
*af*3*n*	0*x*3*n*
*af*4*n*	0*x*4*n*
ef	0*x*50
be	0*x*60

Notes :

Ordonnancement : répartition

```
# tc filter add dev eth1 parent 2: protocol ip prio
1 handle 1 tcindex classid 2:10
# tc filter add dev eth1 parent 2: protocol ip prio
1 handle 2 tcindex classid 2:20
# tc filter add dev eth1 parent 2: protocol ip prio
1 handle 3 tcindex classid 2:30
# tc filter add dev eth1 parent 2: protocol ip prio
1 handle 4 tcindex classid 2:40
```


Notes :

Ordonnancement : gestion des files

- Expedited Forwarding
  ```
  # tc class add dev eth1 parent 2: classid 2:50
  rate 512Kbit ceil 1Mbit
  # tc qdisc add dev eth1 parent 2:50 red limit
  256000 min 16000 max 32000 avpkt 1000 burst 10
  probability 0.02 bandwith 512 ecn
  ```
- Assured Forwarding . . . patience !
- Best Effort
  ```
  # tc class add dev eth1 parent 2: classid 2:50
  rate 512Kbit ceil 1Mbit
  ```


Notes :

Assured Forwarding

Les AFxy (x constant) ont des taux de pertes différents

```
# tc class add dev eth1 parent 2: classid 2:10 htb
rate 64kbit ceil 128kbit
# tc qdisc add dev eth1 parent 2:10 gred setup DPs 3
default 3 grio
```


Notes :

Assured Forwarding

Paramètrage des taux de perte

```
# tc qdisc change dev eth1 parent 2:10 gred limit
2048 min 512 max 1024 burst 10 avpkt 512 bandwidth
64kbit DP 1 probability 0.05 prio 1
# tc qdisc change dev eth1 parent 2:10 gred limit
2048 min 512 max 1024 burst 10 avpkt 512 bandwidth
64kbit DP 2 probability 0.10 prio 2
# tc qdisc change dev eth1 parent 2:10 gred limit
2048 min 512 max 1024 burst 10 avpkt 512 bandwidth
64kbit DP 3 probability 0.15 prio 3
```


Notes :

Les difficultés techniques

Notes :

Algorithmes de traitement de la QoS

- Lissage, marquage, ”poliçage” ;
- Réécriture ?
  - Déjà fait
  - Plus adapté !
- Remplacement ?
  - Choix des remplaçants ?
  - Plus pérenne.

Notes :

Dialogue émulation/prototypage

- Flux de données / informations de contrôle ;
- Efficacité/mise en œuvre ;
- Impact des deux côtés ;
- TUN/TAP, Netfilter/libipq, netlink, appels système, . . .

Notes :

Quelle implantation ?

- Actuellement : module applicatif dialogant avec le module de mise en œuvre de la QoS ;
- Transitoirement : le même module contrôlant l'ordonnanceur ;
- Idéalement : module proche de l'ordonnanceur.

Notes :

[1] S. Blake, D. Black, M. Carlson, E. Davies, and Z. Wang January.
RFC 2475 - an architecture for differentiated service.
Informational, IETF, December 1998.

[2] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang.
RFC 2309 : Recommendations on queue management and congestion avoidance in the internet.
Technical report, IETF, April 1998.
Category : Informational.

[3] R. Braden, D. Clark, and S. Shenker.
Integrated services in the internet architecture : an overview.
Technical report, Internet Engineering Task Force, United States, 1994.

[4] David D. Clark and Wenjia Fang.
Explicit allocation of best-effort packet delivery service.

Notes :

IEEE/ACM Trans. Netw., 6(4) :362–373, 1998.

[5] David D. Clark, Scott Shenker, and Lixia Zhang.
Supporting real-time applications in an integrated services packet network : Architecture and mechanism.
In *SIGCOMM*, pages 14–26, 1992.

[6] A. Demers, S. Keshav, and S. Shenker.
Analysis and simulation of a fair queueing algorithm.
In *SIGCOMM '89 : Symposium proceedings on Communications architectures & protocols*, pages 1–12, New York, NY, USA, 1989. ACM Press.

[7] Sally Floyd and Van Jacobson.
Random early detection gateways for congestion avoidance.
IEEE/ACM Transactions on Networking, 1(4) :397–413, 1993.

[8] Sally Floyd and Van Jacobson.
Link-sharing and resource management models for packet networks.

Notes :

IEEE /ACM *Transactions on Networking*, 3(4) :365–386, 1995.

[9] Paul E. McKenney.
High-speed event counting and classification using a dictionary hash technique.
Proceedings of the International Conference on Parallel Processing, 3 :71–75, 1989.

[10] P.E. McKenney.
Stochastic fairness queuing.
In IEEE, editor, INFOCOM *'90 proceedings*, volume 2, pages 733–740, June 1990.

[11] J. Nagle.
On packet switches with infinite storage.
Technical report, IETF, United States, 1985.

[12] J. B. Nagle.
On packet switches with infinite storage.
Innovations in Internetworking, pages 136–139, 1988.

Notes :

[13] Abhay K. Parekh and Robert G. Gallager.
A generalized processor sharing approach to flow control in integrated services networks : the single-node case.
IEEE/ACM Trans. Netw., 1(3) :344–357, 1993.

[14] M. Shreedhar and George Varghese.
Efficient fair queueing using deficit round-robin.
IEEE/ACM Trans. Netw., 4(3) :375–385, 1996.

[15] L. Zhang.
Virtual clock : a new traffic control algorithm for packet switching networks.
In *SIGCOMM '90 : Proceedings of the ACM symposium on Communications architectures & protocols*, pages 19–29, New York, NY, USA, 1990. ACM Press.

Notes :