



# Introduction à la Qualité de Service

*Quality of Service (QoS)*

Emmanuel Lochin



# Place de la QoS dans l'architecture TCP/IP

- La Qualité de Service implique à la fois la couche réseau et les applications de l'utilisateur final
- Elle est également influencée par la couche transport

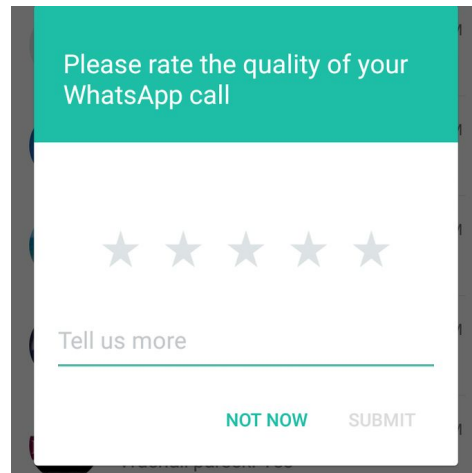
Application
Transport
Réseau
Liaison
Physique

# Définition de la QoS

- La QoS est un ensemble de métriques définissant le type de service qu'un utilisateur obtient du réseau
  - E.g., capacité, latence, taux de perte
  - Composante essentielle pour garantir le bon fonctionnement de certaines applications

# La QoS n'est pas la QoE

- QoE : Qualité d'Experience
- Mesure subjective
- Exemple pour l'audio : *Mean Opinion Score*
  - Le MOS est obtenu en sondant un groupe de personnes sur la qualité d'un appel audio avec évaluation sur une échelle de 1 à 5
- Intérêt : possibilité d'avoir  $QoE=f(QoS)$

A screenshot of a WhatsApp call quality rating interface. It has a teal header with the text "Please rate the quality of your WhatsApp call". Below the header are five grey stars for rating. Under the stars is a text input field with the placeholder "Tell us more". At the bottom right, there are two buttons: "NOT NOW" in teal and "SUBMIT" in grey.

# Le Service *Best-Effort*

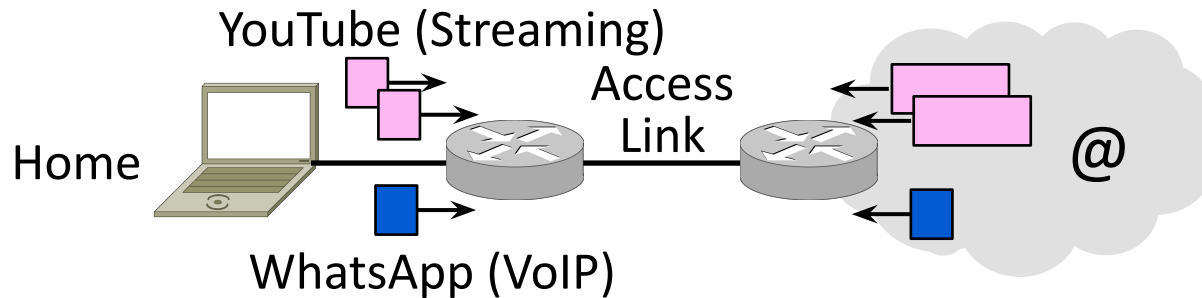
- C'est le service majoritairement délivré par l'Internet
- Les files d'attente des routeurs étant FIFO par défaut
  - Les applications se disputent la capacité disponible
  - Les files d'attente (FA) ajoutent délai et pertes
  - Aucune garantie, pas de QoS

# QoS Motivation

- Le service *Best-effort* n'est pas toujours suffisant
  - Certaines applications ont besoin de garanties
- Quel est le besoin ?
  - Garantie de capacité ou de délai borné
- Comment ?
  - Contrôler la capacité (donc délai/perte) allouée aux utilisateurs

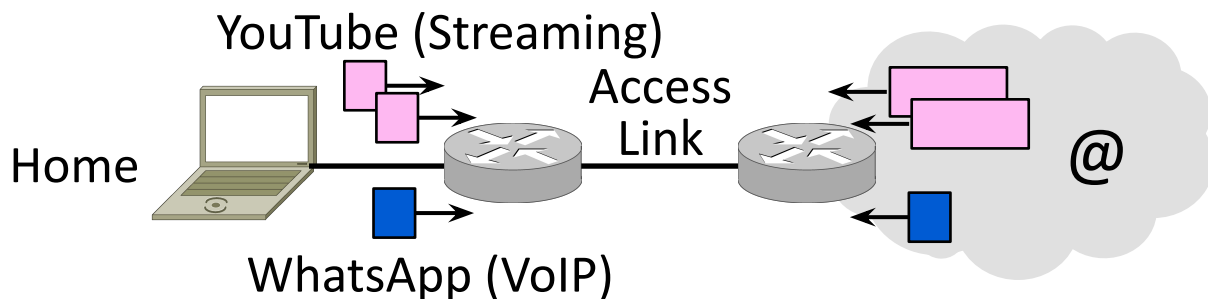
# Exemple – WhatApps et YouTube

- Un utilisateur lance WhatsApp (VoIP seulement) et YouTube en même temps
  - Il désire une faible latence pour WhatsApp (*real-time*), beaucoup de capacité pour YouTube (*bulk*)



# Exemple – WhatApps et YouTube

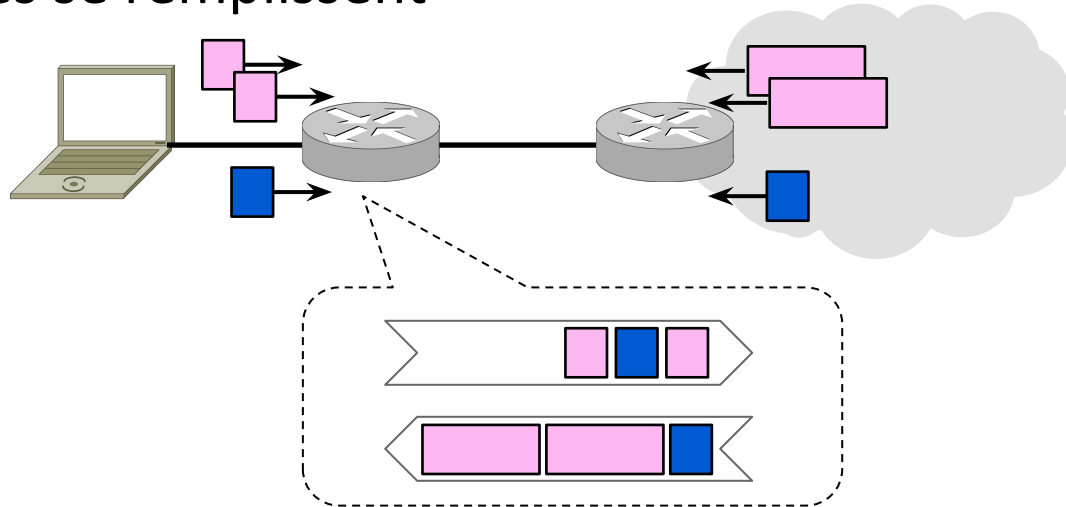
- Exemple sur lien FTTH à débit effectif  $\sim 40\text{Mbps}$ 
  - Streaming 4K/2160p à 60 FPS entre 10 et 40 Mbit/s
  - Whatsapp entre 64Kbs (voix seulement) et 384Kbs (avec vidéo)





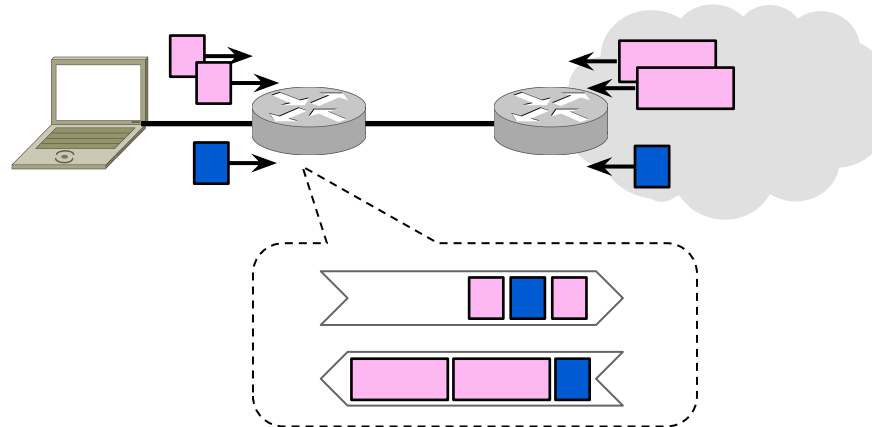
# Exemple – WhatApps et YouTube

- Qu'obtient-on avec des routeurs FIFO ?
  - Whatsapp et Youtube concourent pour la capacité disponible du lien d'accès (*bottleneck*)
  - Les files se remplissent



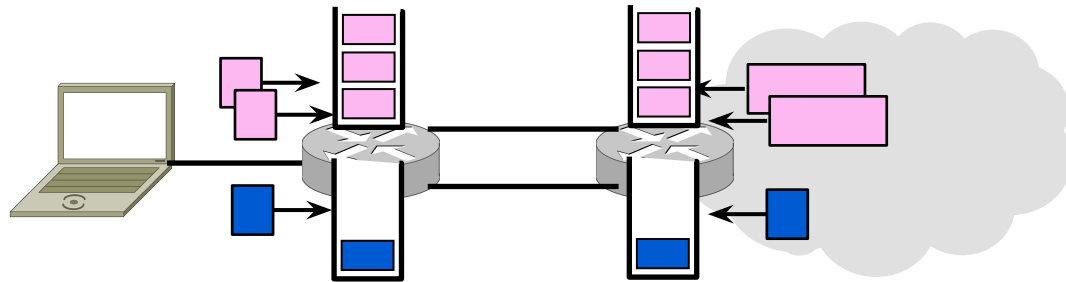
# Exemple – WhatApps et YouTube

- Qu'obtiennent les applications ?
  - Diminution de la qualité audio de WhatsApp à cause des pertes et du délai
  - Le YouTube très peu affecté par WhatsApp



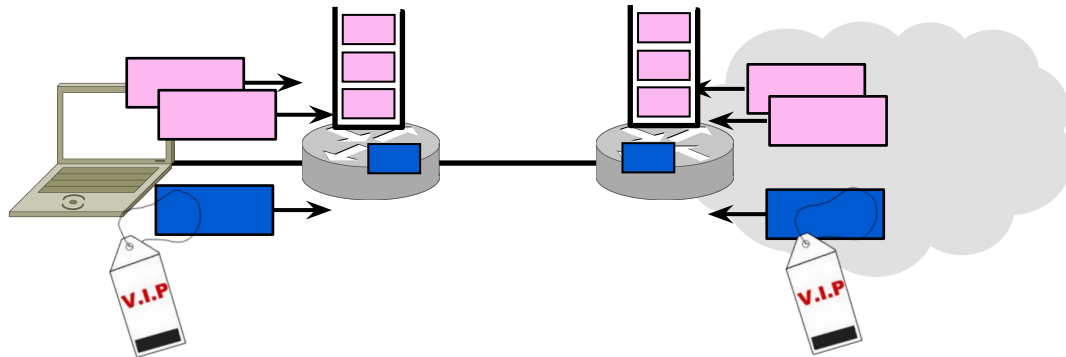
# Exemple – WhatApps et YouTube

- Que se passe-t-il si nous divisons le lien ?
  - Maintenant WhatsApp obtient une bonne qualité
  - YouTube est moins performant
  - Mais : on gâche de la capacité, solution non optimale



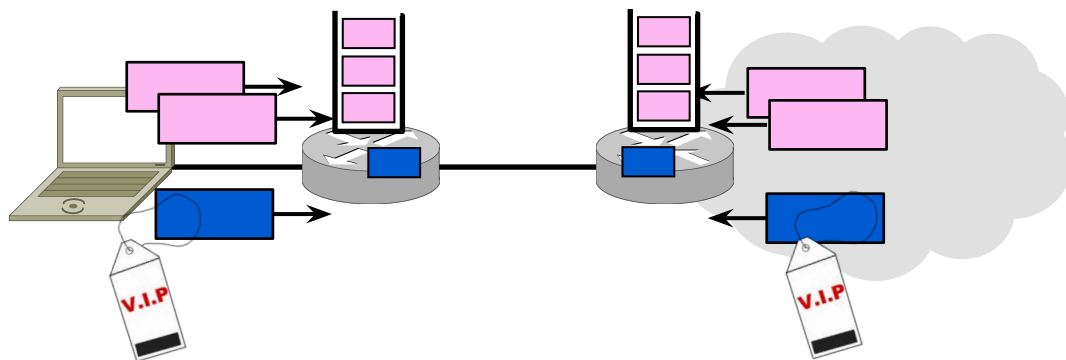
# Exemple – WhatApps et YouTube

- Une meilleure idée ?
  - Modifier la politique du routeur d'accès afin qu'il donne la priorité au trafic WhatsApp



# Exemple – WhatApps et YouTube

- Qu'obtient-on maintenant ?
  - Appel de haute qualité pour WhatsApp et débit de transfert optimal pour YouTube
  - Les deux gagnent !



# Objectifs de la QoS

- Allouer la capacité afin d'améliorer la performance des applications/utilisateurs
  - Garantir une capacité à une application
  - Satisfaire plusieurs applications simultanément
  - Nous verrons comment faire avec des mécanismes qui vont au-delà de la simple priorisation de trafic
- Fournir une QoS signifie connaître les besoins des apps.
  - Besoin en capacité, délai, perte

# Besoins applicatifs

- Une exigence élevée signifie capacité élevée, faible délai/perte

Application	Bandwidth	Delay	Jitter	Loss
Email	Low	Low	Low	Medium
File sharing	High	Low	Low	Medium
Web access	Medium	Medium	Low	Medium
Remote login	Low	Medium	Medium	Medium
Audio on demand	Low	Low	High	Low
Video on demand	High	Low	High	Low
ToIP / VoIP	Low	High	High	Low
Videoconferencing	High	High	High	Low

Variation  
in delay

# A propos du sur-provisionnement

- La QoS n'a d'importance que s'il y a un goulot d'étranglement dans le réseau (*bottleneck*)
  - Sinon aucune perte ou délai
  - Donc aucune possibilité d'amélioration
- Pourquoi ne pas sur-provisionner ?
  - Alternative simple à la QoS
  - Pas rentable et sans garantie

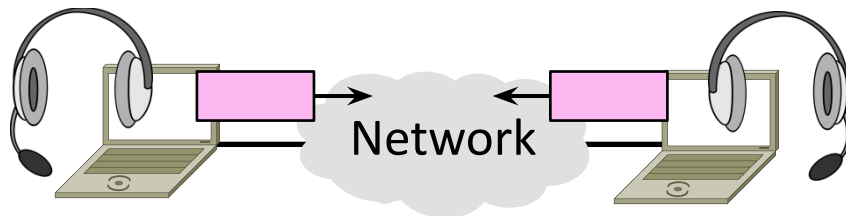


# Plan du cours

- Introduction, besoins applicatifs } ce que nous venons de voir
  - Transport temps-réel (VoIP)
  - *Streaming* (vidéo)
  - *Fair Queuing*
  - *Traffic Shaping*
  - Services différenciés
- la suite...

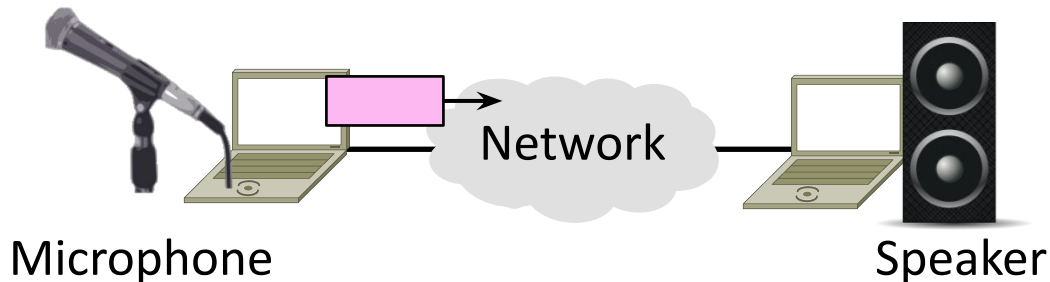
# Le Transport temps-réel

- Quelle performance pour un média interactif temps-réel, e.g., VoIP ?
  - Sachant que l'Internet est *best-effort* ?
- Solution : utilisation d'un *playout buffer*



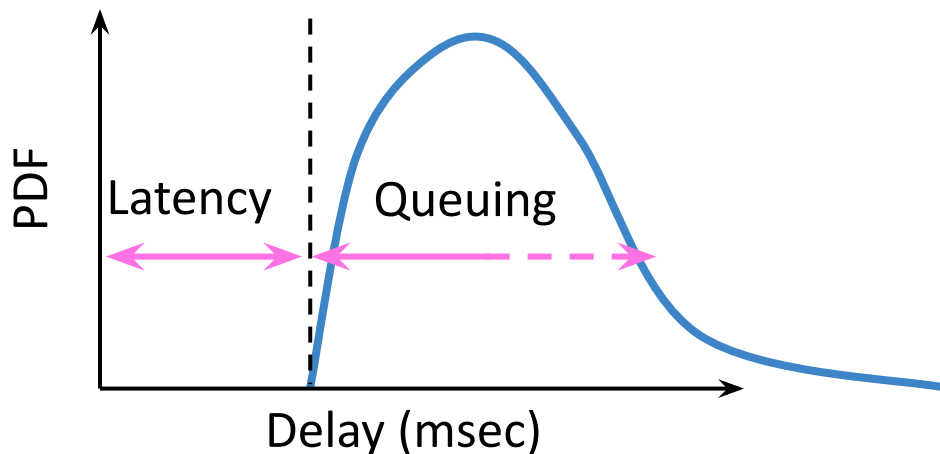
# Le challenge : le délai du réseau

- Considérons une seule direction
  - La source génère un débit constant (CBR), qui est consommé au niveau du récepteur
  - Le réseau doit avoir suffisamment de capacité, de plus, il ajoute un délai



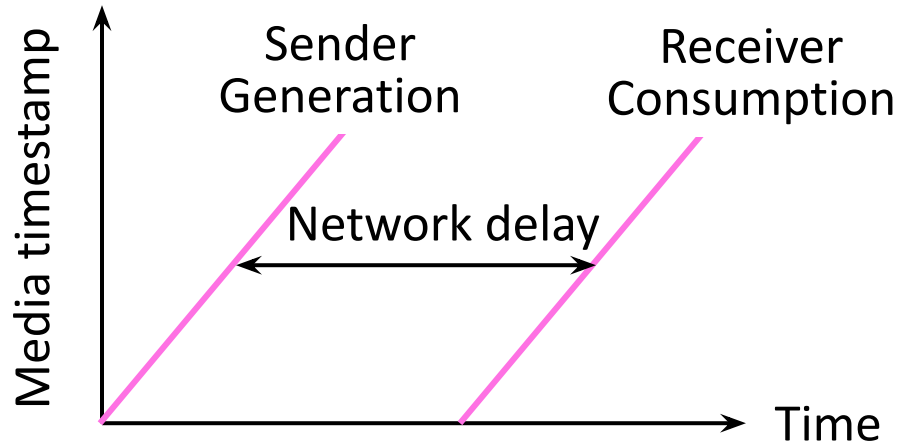
# Délai du réseau

- Le délai du réseau varie :
  - Latence des message + *queuing delay*
  - La variabilité du délai est la gigue (*jitter*)



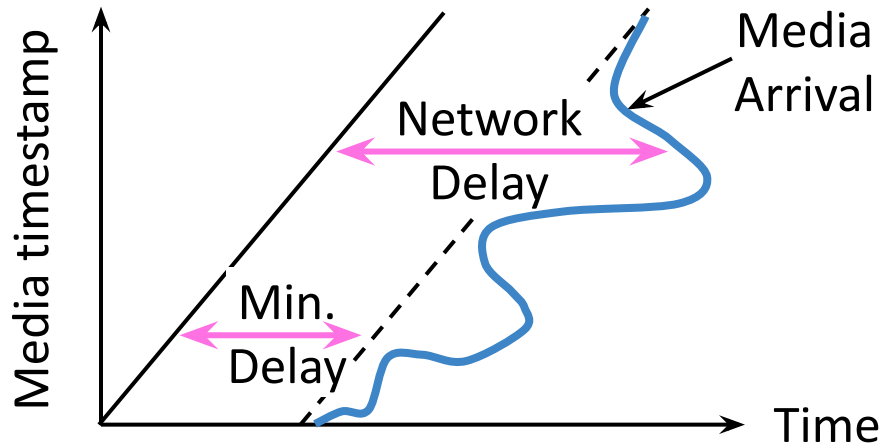
# Délai du réseau

- Idéalement ce délai est fixe et petit afin de permettre une bonne interactivité



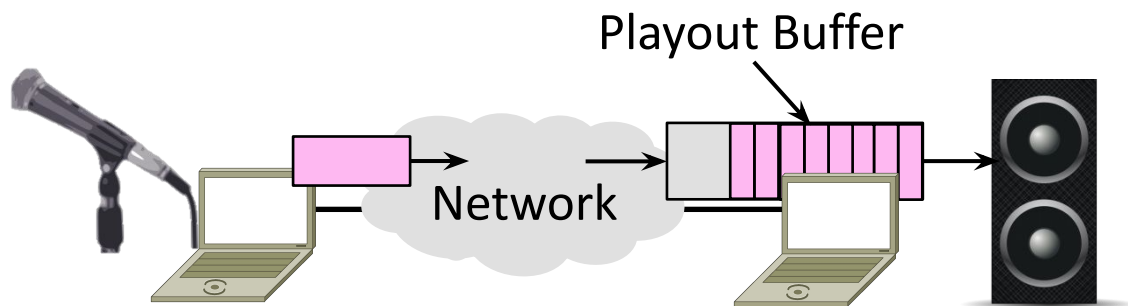
# Délai du réseau

- Les données arrivent à destination après un délai de réseau variable



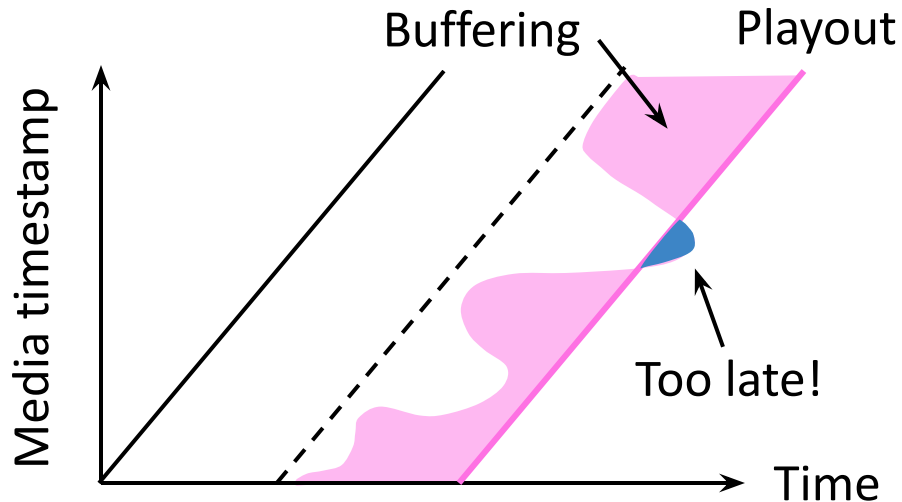
# Playout Buffer

- Ajout des données dans un *playout buffer* à réception dans l'attente de leur consommation
  - Permet de lisser le délai du réseau



# Playout Buffer

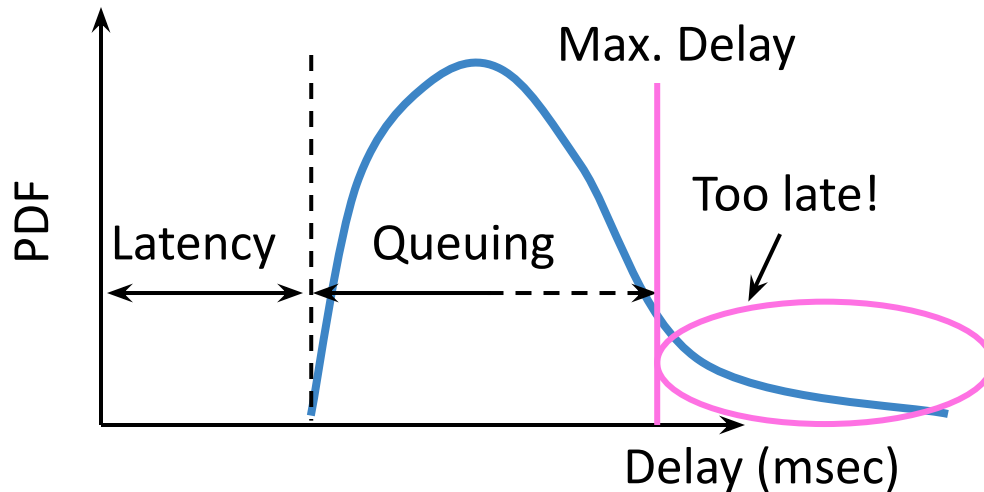
- La courbe d'arrivée des données détermine le temps de mise en *buffer* et la *deadline*





# Playout Buffer

- Sélection du délai maximum acceptable pour définir le point de lecture



# Playout Buffer

- Compromis :
  - Délai acceptable grand → grand *buffer*/délai, moins de perte
  - Délai acceptable petit → petit *buffer*/délai, plus de perte
- Difficilement utilisable dans les cas de média interactif, temps-réel
  - On fait sans (*glitch*)
  - Ou avec un très petit *playout buffer* (léger décalage)

# Media streaming

- Lecture de contenu multimédia sur le réseau
  - Toujours en *best-effort* sur l'Internet
  - Coursera, YouTube, Netflix, etc.
  - Enorme usage!



# Media streaming

- Il en existe trois classes de *streaming* :
  - *stored-streaming* : c'est le cas des applications précédentes
  - *live-streaming* : lorsque les données sont produites en temps-réel (e.g. capture depuis une caméra)
  - *interactive live-streaming* : média interactif telles la VoIP (WhatsApp), la vidéo-conférence, ...

# Streaming vs. Interactive Media

- Le *store-streaming* (abrégé *streaming*) est le cas le plus simple :
  - Une seule direction à considérer
  - Le délai impacte uniquement le démarrage → nécessite un *playout buffer*
  - En revanche le *streaming* est sensible à la gigue (*jitter*) et à la capacité disponible



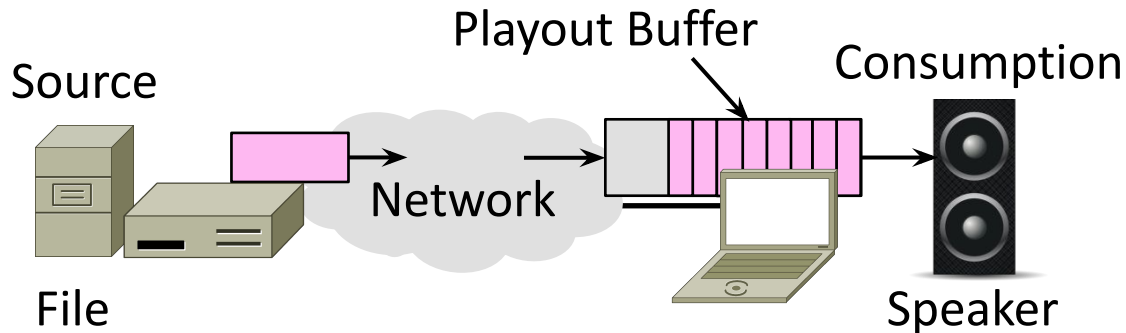
# Streaming vs. Interactive Media

- Le *live-streaming* et l'*interactive live-streaming*
  - Peuvent opérer de façon unidirectionnelle (video live) ou bidirectionnelle (VoIP, vidéoconférence)
  - Traite des flots dits "temps-réel"
  - Sont sensibles à la gigue



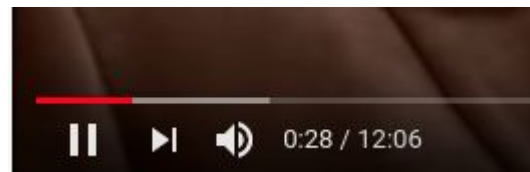
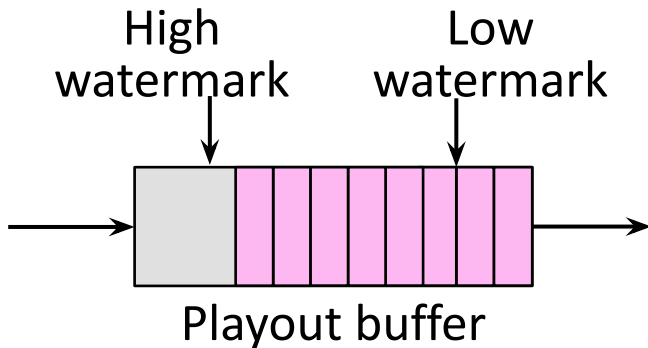
# Gestion de la gigue

- De la même façon que précédemment :
  - on bufferise le média à la réception jusqu'à ce qu'il soit joué par l'application
  - toujours la même idée : lisser la variation du délai



## Gestion de la gigue (2)

- Contrôle de flot opéré par l'application via des *watermarks*
  - Haut : arrêt de la demande de données au serveur
  - Bas : demande plus de données au serveur





# Gestion de la capacité

- Envoyer des fichiers avec différents encodage
  - Plus la qualité est haute plus il y a besoin en capacité
  - Sélection du meilleur encodage en fonction de la capacité disponible (celle du *bottleneck*)



# Streaming avec TCP ou UDP?

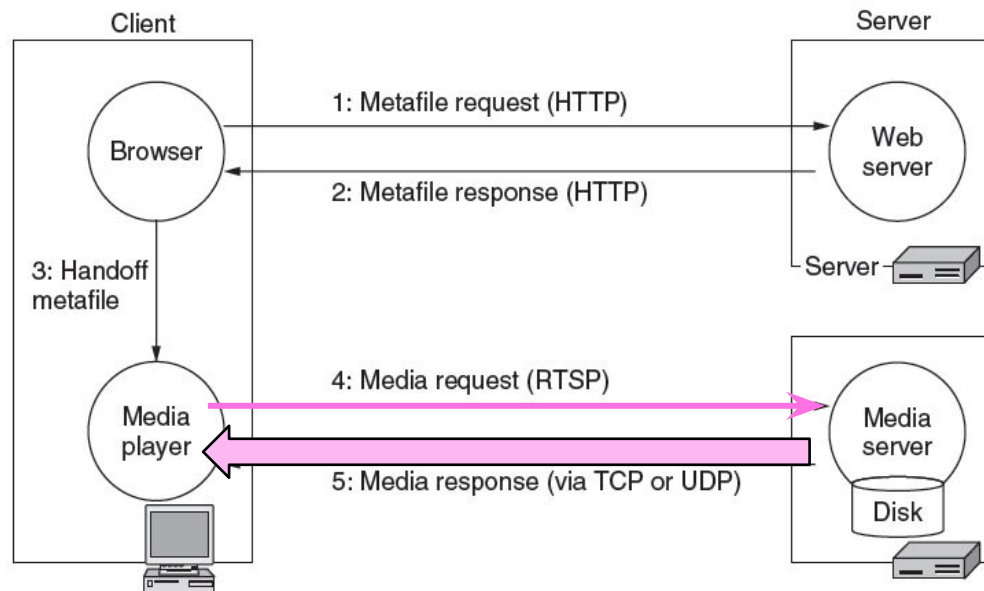
- UDP minimise le délai des messages pour les sessions interactives temps-réel
- TCP est utilisé pour le *streaming* lorsque
  - Un faible délai n'est pas essentiel; importance du démarrage  
→ visionner [RITE Latency](#)
  - La récupération des pertes simplifie la présentation
  - HTTP/TCP pour passer les pare-feu

# Composants du Multimédia

- Une session se compose :
  - D'une phase de signalisation, e.g., avec RTSP
  - Du transport, e.g., avec HTTP
  - *Playout buffer*
  - Les standards évoluent très vite, e.g., HTML5
- Usage des CDNs

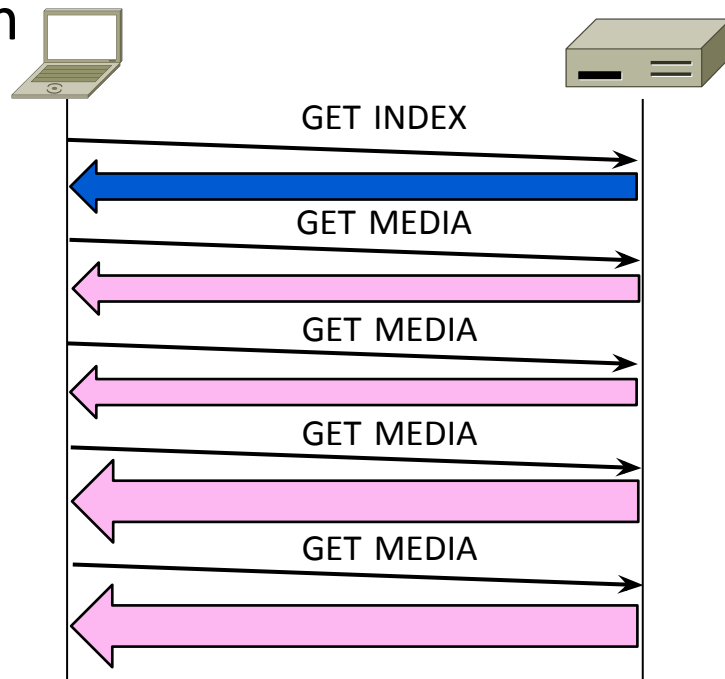
# Streaming avec RTSP

- Récupération via HTTP d'un *metafile*
- Lancement d'un *media player* → dialogue entre RTSP (Real-Time Streaming Protocol) et le serveur de contenus
- Média transmis, e.g., RTP au-dessus de TCP/UDP



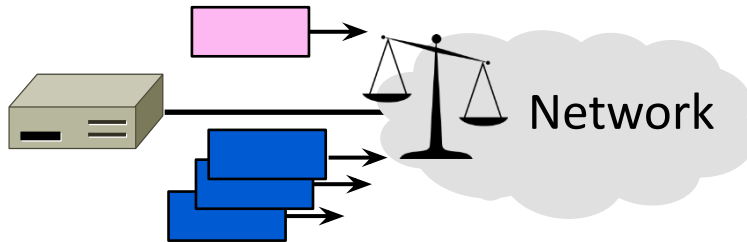
# HTTP Streaming

- Récupère les données de description des médias
  - Donne l'index des vidéos, débits
- Récupère des petits segments
  - Stockés dans le *playout buffer*
- Adapte l'encodage
  - Suivant l'occupation du *buffer*
- Evolution des standards, e.g., DASH
  - Exploite HTTP et HTML5
  - Sinon serveur sans état



# Fair Queuing

- Partager la capacité entre les flots
  - WFQ (Weighted Fair Queuing)
  - Composant essentiel de la QoS

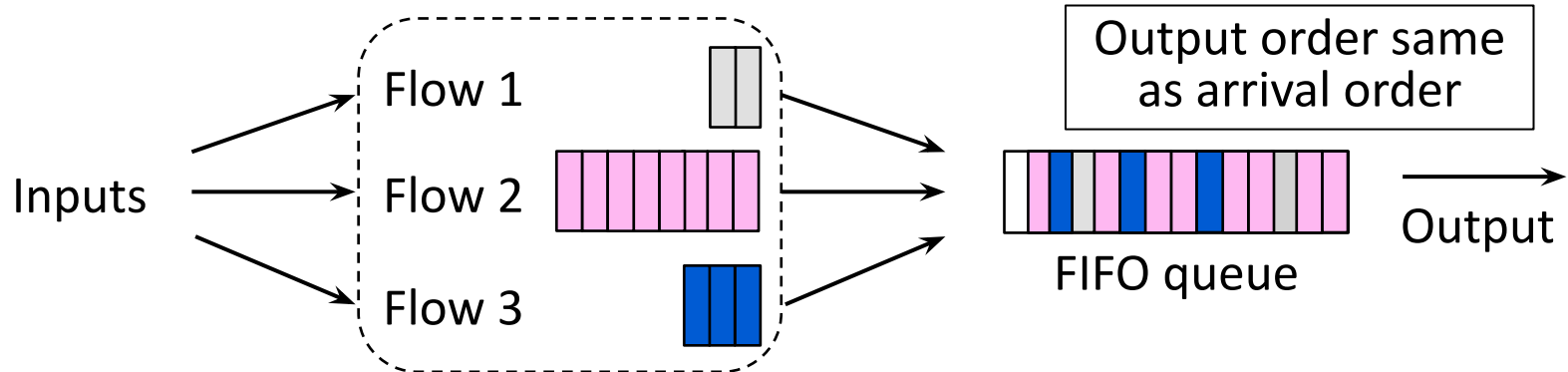


# Partage avec FIFO

- FIFO *drop-tail queue*:
  - First In First Out (FIFO)
  - Rejet des nouveaux paquets lorsque plein
  - Modèle typique d'un routeur standard
- Partage avec FIFO
  - Plusieurs utilisateurs/flots envoient des paquets vers le même lien de sortie
  - Qu'arrive-t-il?

# Partage avec FIFO

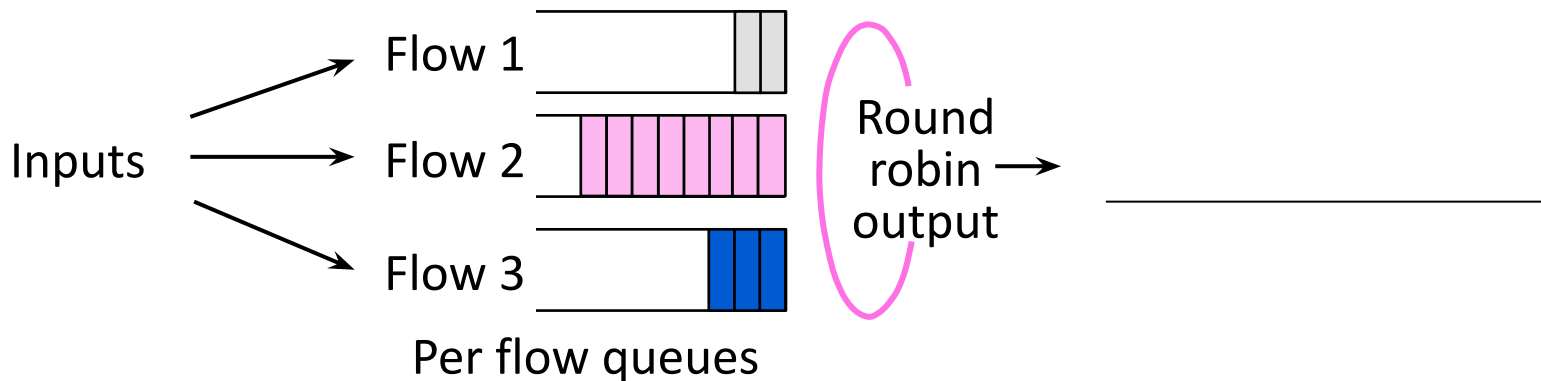
- L'allocation dépend du comportement des autres flots
  - TCP fait du partage à long-terme, suivant le couple (délai, perte) et un biais suivant le RTT
  - Un utilisateur/flot agressif peut évincer les autres





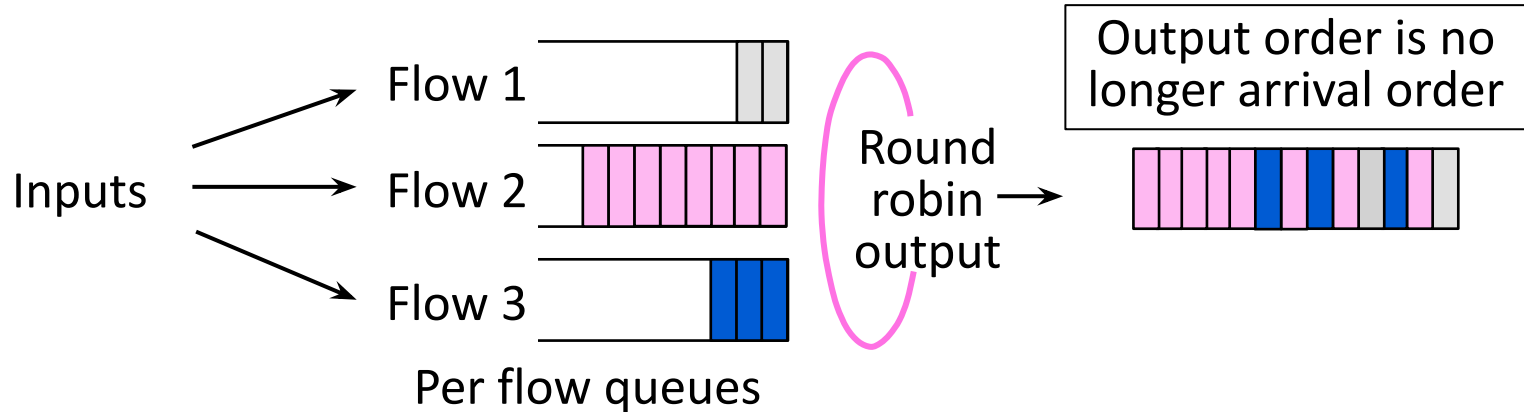
# Round-Robin

- Idée pour améliorer l'équité :
  - Mettre en file d'attente chaque flot séparément
  - Appliquer l'algorithme du tourniquet



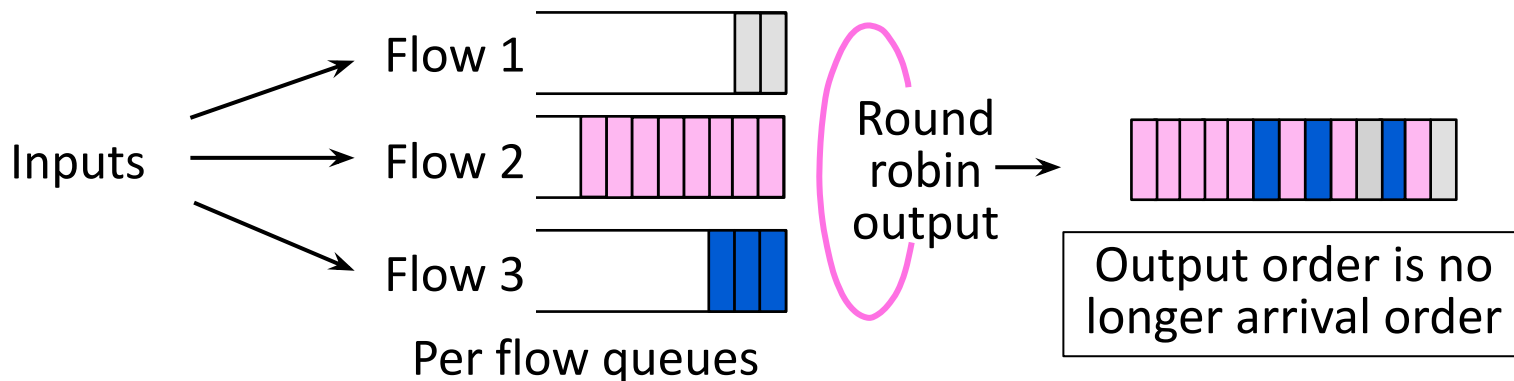
# Round-Robin

- Idée pour améliorer l'équité :
  - Mettre en file d'attente chaque flot séparément
  - Appliquer l'algorithme du tourniquet



# Round-Robin

- Isolation entre les flots
- La taille des paquets entraîne un déséquilibre du partage
  - qui peut être important, e.g., 40B vs 1500B



# Fair Queuing

- Round robin avec équité approximative au niveau bit:
  - Approximation en calculant un *virtual finish time*
  - L'horloge virtuelle s'incrémente une fois pour chaque bit envoyé par tous les flots
  - Envoi les paquets dans l'ordre de leurs heures de fin virtuelles,  $\text{Finish}(j)_F$
  - Imparfait - ne préempte pas le paquet transmis

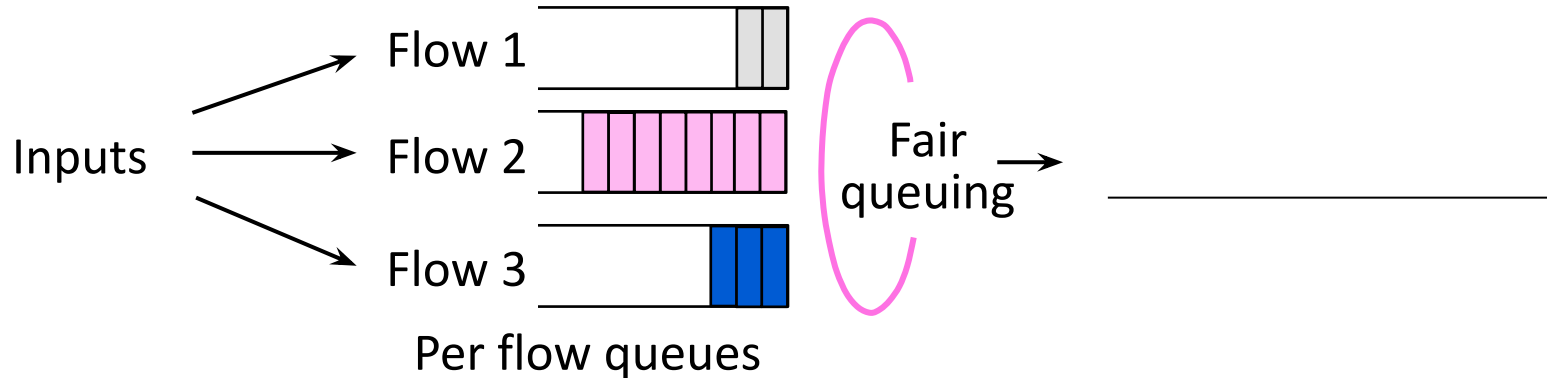
$\text{Arrive}(j)_F$  = temps d'arrivée du paquet  $j$  du flot  $F$

$\text{Length}(j)_F$  = taille du paquet  $j$  du flot  $F$

$\text{Finish}(j)_F = \max (\text{Arrive}(j)_F , \text{Finish}(j-1)_F) + \text{Length}(j)_F$

# Fair Queuing

- Exemple, supposons :
  - Flots 1 et 3 ont des tailles de paquets de 1000B, et 2 de 300B
  - Qu'obtient-on avec cet algorithme?



# Fair Queuing

- Exemple, supposons :
  - Flots 1 et 3 ont des tailles de paquets de 1000B, et 2 de 300B
  - Qu'obtient-on avec cet algorithme?

Hypothèses :  $\text{Finish}(0)_F = 0$ , les files sont toujours pleines [ $\text{Arrive}(j)_F < \text{Finish}(j-1)_F$ ] i.e., il y a toujours quelque-chose à émettre

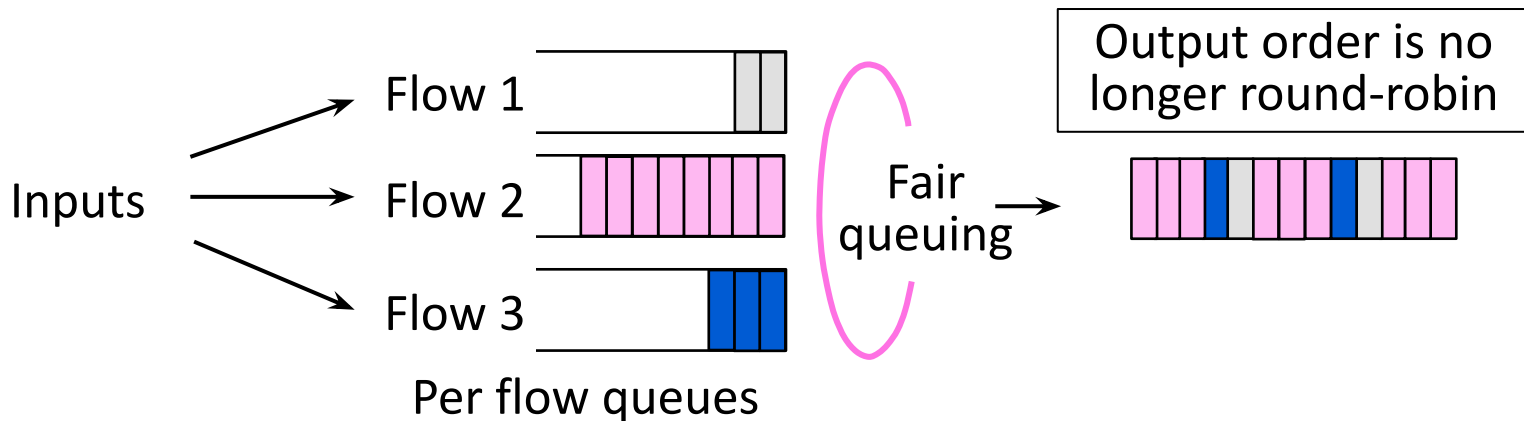
$\text{Finish}(1)_{F1} = 1000$ ,  $\text{Finish}(2)_{F1} = 2000$ , ...

$\text{Finish}(1)_{F2} = 300$ ,  $\text{Finish}(2)_{F2} = 600$ ,  $\text{Finish}(3)_{F2} = 900$ , 1200, 1500, ...

$\text{Finish}(1)_{F3} = 1000$ ,  $\text{Finish}(2)_{F3} = 2000$ , ...

# Fair Queuing

- Exemple, supposons :
  - Flots 1 et 3 ont des tailles de paquets de 1000B, et 2 de 300B
  - Qu'obtient-on avec cet algorithme?



# WFQ (Weighted Fair Queuing)

- WFQ est une généralisation du partage équitable (*Fair Queuing*)
  - Un poids est assigné à chaque flot :  $\text{Weight}_F$
  - Un poids de 2 donne 2 fois plus de capacité

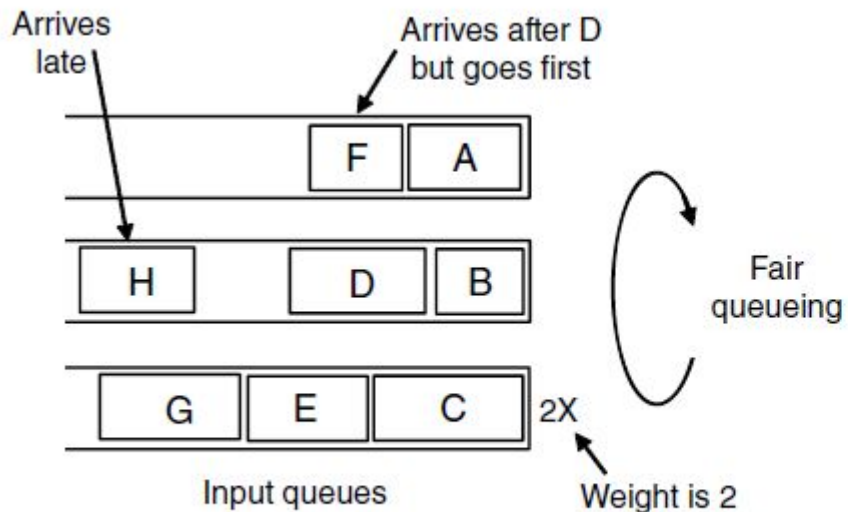
$\text{Arrive}(j)_F$  = temps d'arrivée du paquet  $j$  du flot  $F$

$\text{Length}(j)_F$  = taille du paquet  $j$  du flot  $F$  /

$\text{Finish}(j)_F = \max (\text{Arrive}(j)_F , \text{Finish}(j-1)_F) + \text{Length}(j)_F / \text{Weight}_F$



# Exemple avec WFQ



Packet	Arrival time	Length	Finish time	Output order
A	0	8	8	1
B	5	6	11	3
C	5	10	10	2
D	8	9	20	7
E	8	8	14	4
F	10	6	16	5
G	11	10	19	6
H	20	8	28	8

# Utilisation de WFQ

- Enorme potentiel
  - Peut prioriser et protéger les flots
  - Composant essentiel pour la mise en oeuvre de la QoS
- Ce n'est pas une solution complète
  - Identification d'un flot (utilisateur, application, connexion TCP ?)
  - Limite de traitement à très haute vitesse
  - Besoin de mettre en place un poids

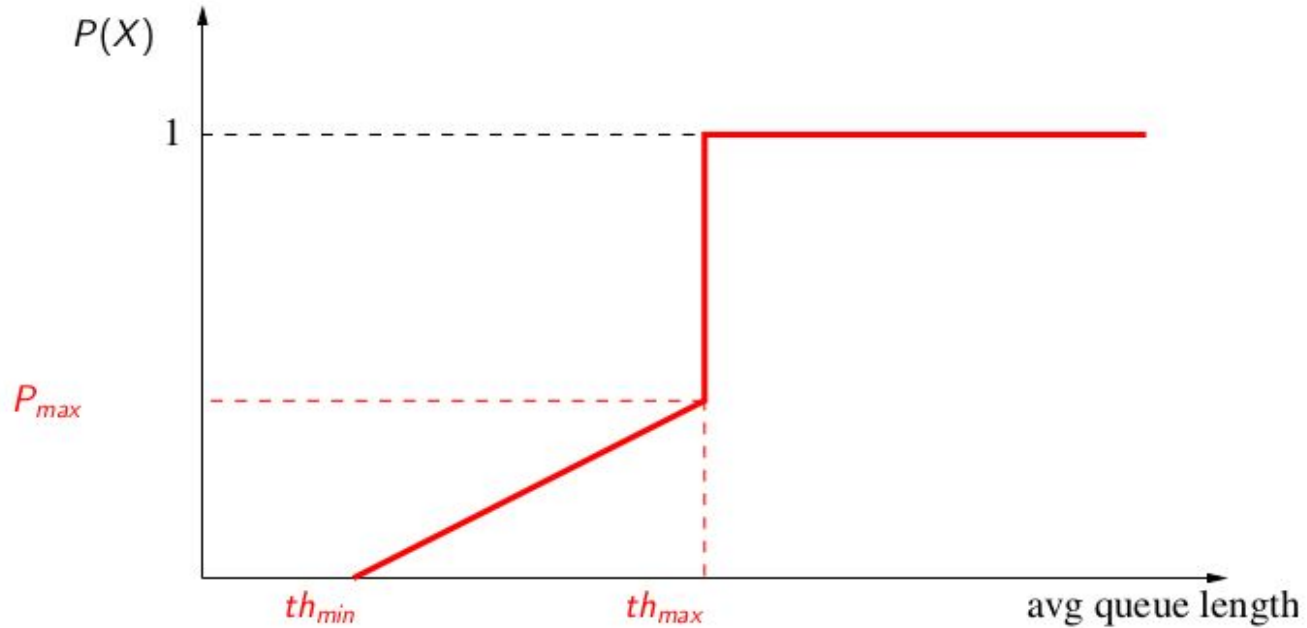
# Active Queue Management (AQM)

- Au-delà de FIFO
  - Gestion active des paquets de la file
  - Algorithmes CHOKe, **RED**, PIE, Codel, ...
  - Rejet de paquets
    - En fonction d'un seuil, temps de mise en attente
  - Ou marquer des paquets
    - Explicit Congestion Notification

# Random Early Detection (RED)

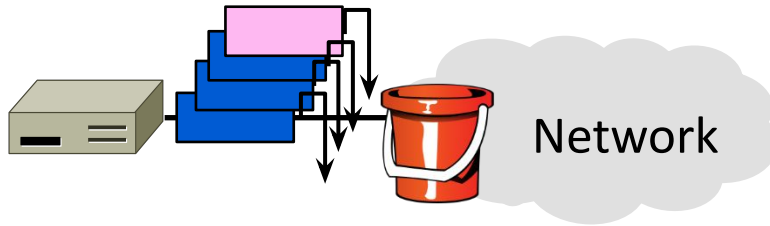
- RED résout certains problèmes TCP avec FIFO
  - Réduit les oscillations
  - Réduit la longueur moyenne
  - Evite la synchronisation
    - en FIFO lorsque la file est pleine, plusieurs connexions perdent des paquets en même temps
  - Permet aux rafales courtes de passer sans que la taille de la file augmente beaucoup
  - Contrôle la taille moyenne de la file afin de diminuer le délai moyen

# Algorithme RED



# Traffic Shaping

- Le lissage du trafic (*traffic shaping*) permet de limiter/contraindre les rafales de trafic (*traffic bursts*)
  - *Token buckets*
  - Encore une composante essentielle de la QoS

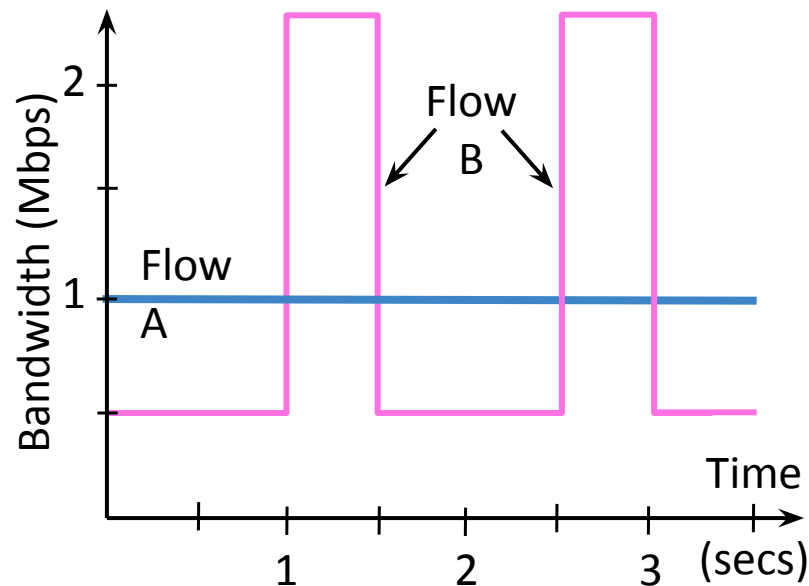


# Motivation

- Lisser le trafic permet de limiter la charge du réseau
  1. La limitation du trafic total permet des garanties de capacité
  2. Limiter les rafales permet d'éviter des délais et pertes inutiles
- Comment lisser le trafic?
  - Les applications génèrent un trafic variable - irréaliste de le lisser en sortie
  - Besoin de le lisser avant l'entrée dans le réseau

# Motivation

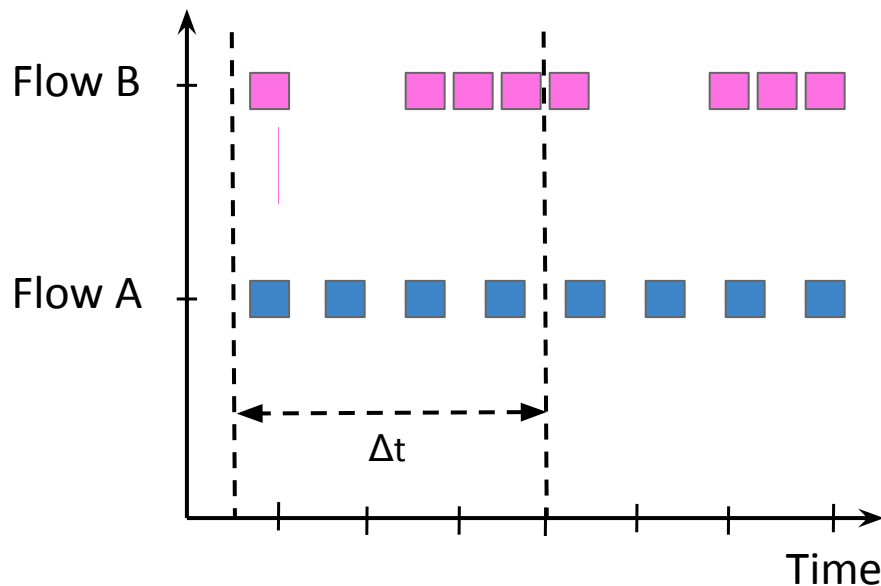
- Les flots A et B ont le même débit moyen
  - 1 Mbps sur 3.5 sec
  - Mais ils ont un comportement différent
- Le débit moyen seul n'est pas un descriptif suffisant
  - besoin du *burst* moyen/max





# Motivation

- Autre représentation discrète
- Le sporadicité du trafic est différente
- Mais le débit moyen est identique  $\Delta t = 4\text{pkt}$
- Que se passe-t-il si le flot B traverse une FA de routeur de capacité  $B=1\text{pkt}$  et un débit de sortie de  $4\text{pkt}/\Delta t$  ?

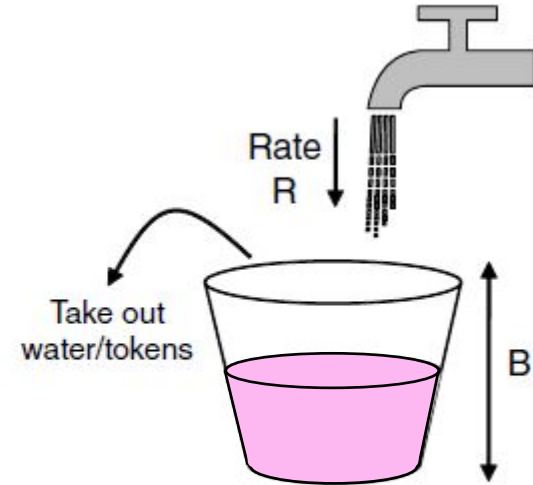


# Motivation

- Comment caractériser le trafic réseau ?
  - Le débit moyen est une caractérisation de la capacité sur le long-terme
  - La sporadicité du trafic (*burstiness*) est une caractérisation de la capacité sur le court-terme
- Deux caractéristiques très utiles
  - La combinaison des deux donne plus d'information
  - Utile dans plein de contexte

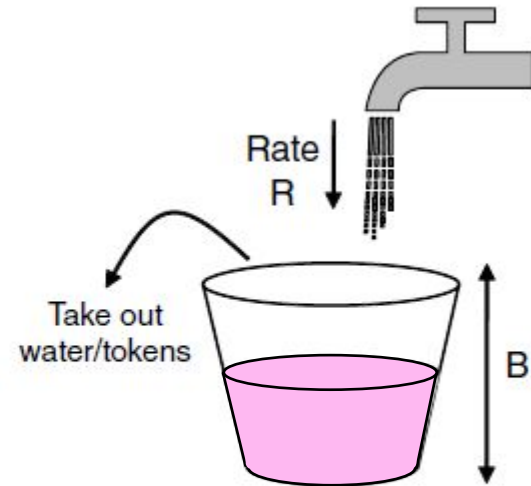
# Token Bucket

- Un seau à jeton de paramètre  $(R, B)$  se caractérise par :
  - Un débit moyen  $\leq R$  bits/sec
  - Une rafale (sur  $R$  sec)  $\leq B$  bits



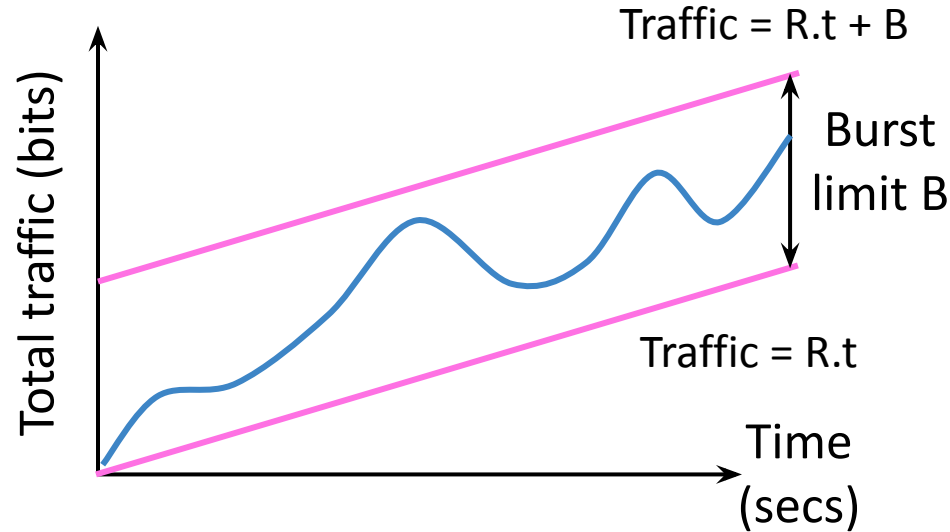
## Token Bucket (2)

- Des jetons (crédits) sont retirés du seau lors de l'émission de paquets
- Pas de jetons pas d'émission
  - Le seau se remplit continuellement au débit  $R$  bits/sec
  - Débordement à  $B$  bits



# Token Bucket (3)

- Objectif : contraindre le trafic



# Shaping vs. Policing

- Le *shaping* lisse le trafic de la source pour se conformer à un profil défini par  $(R, B)$ 
  - Le trafic source est accepté si il y a des jetons
  - Il est **décalé** dans l'attente de jetons
- Permet à **l'utilisateur** de conditionner son trafic pour respecter le contrat réseau

## Shaping vs. Policing (2)

- Le *policing* lisse le trafic de la source pour se conformer à un profil défini par (R, B)
  - Le trafic source est accepté si il y a des jetons
  - Il est **jeté** lorsqu'il n'y a plus de jetons
- Permet au **réseau** de vérifier le trafic pour vérifier qu'il respecte le contrat de l'utilisateur

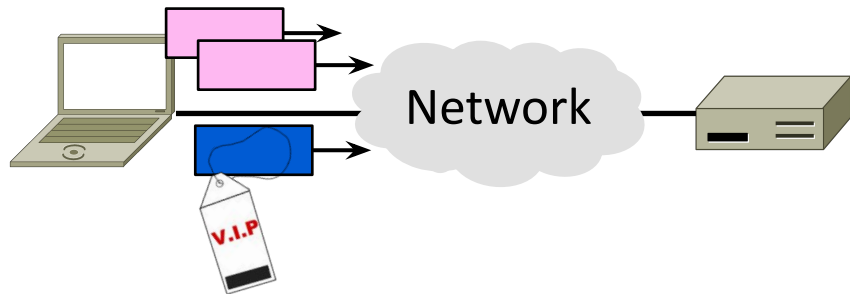
# Utilisation pour la QoS

- Les seaux de jetons permettent à l'utilisateur et au réseau de réguler le trafic
  - Le réseau peut limiter le trafic pour un traitement préférentiel
  - L'utilisateur peut sélectionner ce trafic de manière flexible
- Un traitement spécifique peut-être mis en oeuvre avec d'autres moyens tels que WFQ



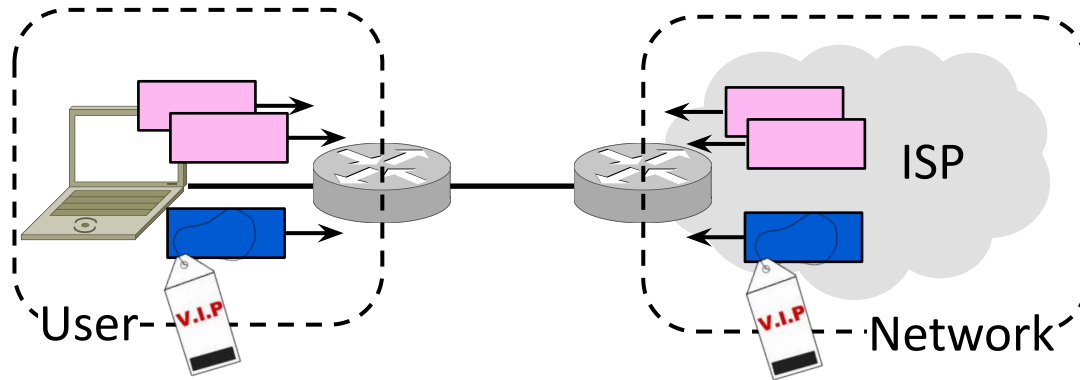
# DiffServ - Differentiated Services

- Traiter les différents flux de trafic différemment dans le réseau
  - Standard IETF RFC 2475
  - On parle de services différenciés
  - Déployé chez les FAI, architectures SATCOM



# Motivation

- Un utilisateur tourne WhatsApp et YouTube
  - Ou un accès distant, jeu en ligne, web, ...
  - Comment donner un traitement préférentiel à ces flots ?

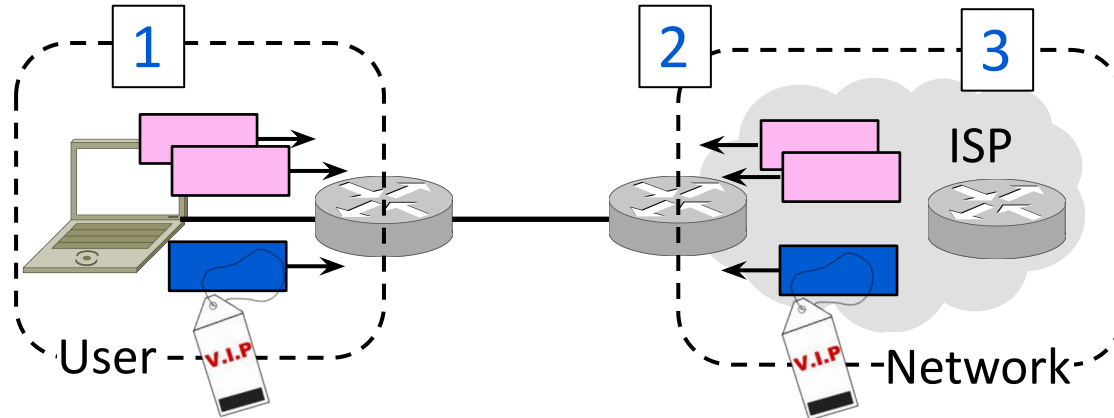


# Differentiated Services

- L'idée est de traiter différemment différents types de trafic sur le réseau
  - Avoir quelques types de service réseau (or, argent, bronze)
  - Différents types de trafic obtiennent un traitement meilleur ou pire dans le réseau
  - Mapper les applications sur le bon type de service

# Differentiated Services (2)

- Principe architectural :
  1. Les paquets sont marqués avec le service désiré côté utilisateur/application (e.g., WhatsApp=gold)
  2. Le réseau de bordure vérifie la conformité du trafic (*shaping/policing*)
  3. Le réseau de coeur se charge de l'acheminement/ordonnancement (WFQ)



# 1. Marquer le trafic

- Utilise le champs DSCP de l'en-tête IPv4/IPv6 pour marquer le type de service
  - 6-bit DSCP (Differentiated Services Code Point)

IPv4 Header

Version	IHL	Differentiated Services	Total length			
Identification				D F	M F	Fragment offset
Time to live		Protocol		Header checksum		
Source address						
Destination address						

# Marquer le trafic

- Plusieurs marques DSCP possibles en fonction du service et des applications
  - Les services supportés dépendent de la configuration du réseau

Service Name / Meaning	DSCP Value	Traffic Need (App example)
Default forwarding / Best effort	0	Elastic (Bulk Data Transfer)
Assured forwarding / Enhanced effort	10-38	Average rate (streaming video)
Expedited forwarding / Real-time	46	Low loss/delay (VoIP, gaming)
Precedence / e.g., Network control	48	High priority (Routing protocol)

# Marquer le trafic

- Le trafic est marqué par l'utilisateur
  - Dépend de la politique locale e.g., *gaming* = EF?
- Peut-être réalisé du côté de l'hôte
  - L'OS ou l'application classe son trafic
- Peut-être réalisé dans le réseau
  - Avec des heuristiques, numéros ports, ...

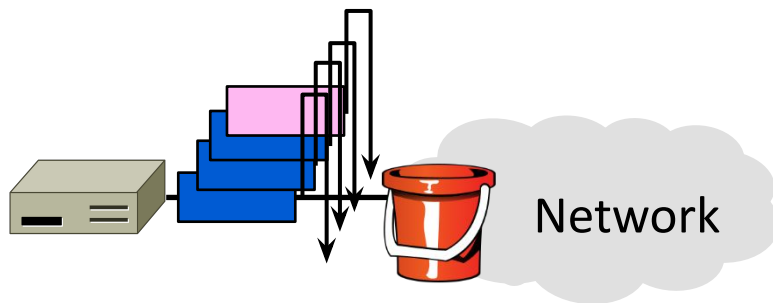
## 2. Contrôler le trafic (*policing*)

- Les FAI vérifient que le trafic entrant respecte le contrat de trafic
  - Aucun trafic supplémentaire que celui négocié (ou celui acheté)
  - Uniquement les marquages autorisés, e.g., les utilisateurs ne contrôlent pas le réseau



# Contrôler le trafic (*policing*)

- Ce contrôle est réalisé par des *token buckets*
  - Peut rétrograder le trafic «hors profil» en re-marquant (par exemple, par DE/*best-effort*) ou en priorisant la perte

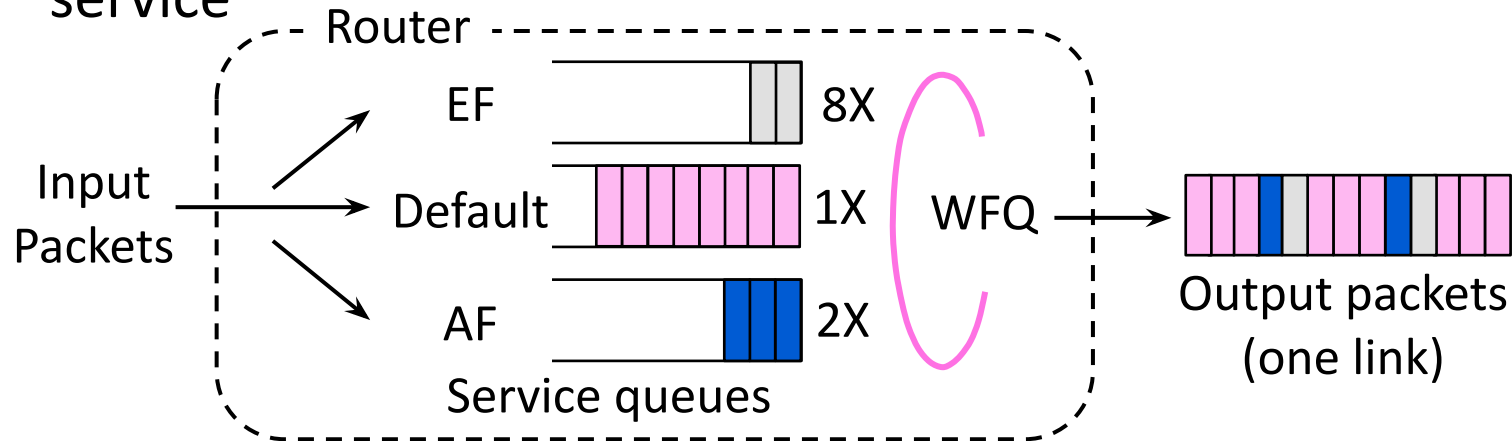


### 3. Forwarding Packets

- Les routeurs de réseau (ceux du FAI) utilisent WFQ (voire plus) au lieu de FIFO
- Les différents types de services sont les différents flots/AQM
- Les valeurs DSCP sont utilisées pour mapper le paquet vers le bon flux / file d'attente

# Forwarding Packets

- Services definis par des PHB (*per-hop behaviors*)
  - Aucune garantie de service de bout en bout
  - Besoin de limiter le trafic hautement prioritaire pour un bon service

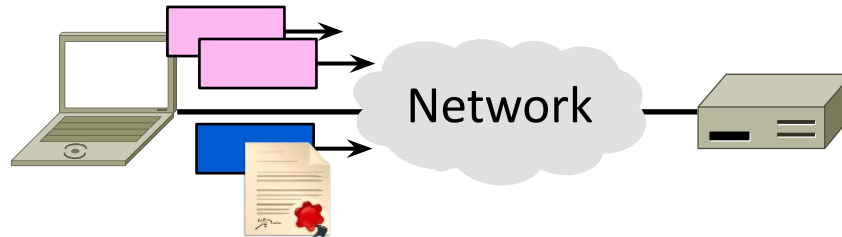


# Deploiement

- La QoS offre un avantage si déployée sur tout le réseau
  - Généralement uniquement chez les FAI
- La QoS est étroitement liée à la tarification
- Deploiement lent/difficile

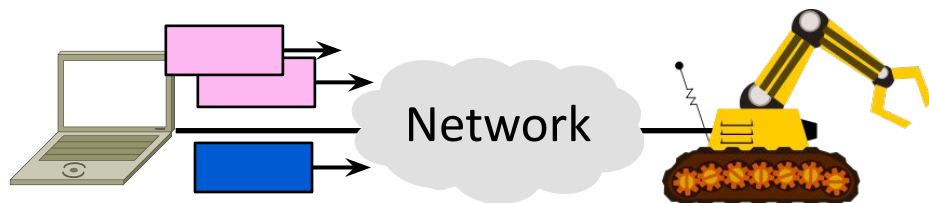
# Garantir un débit et borner un délai

- Garantir la performance du trafic sur tout le chemin
  - On parle de *hard* QoS avec des garanties strictes



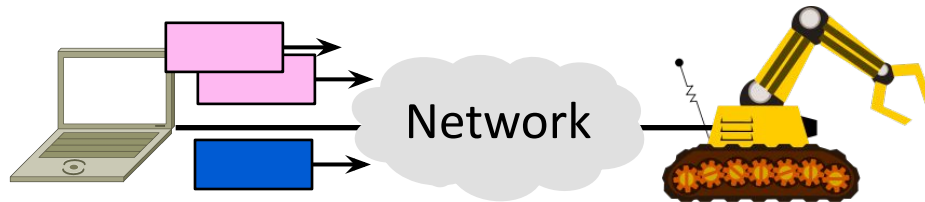
# Motivation

- Nécessaire, comme par exemple pour la téléphonie
  - Bornes de débit min et délai max quelque soit le trafic
  - e.g., contrôle/commande, *tactile Internet*



# Motivation

- Comment ?
  - Provisionner un circuit dédié (ou construire un réseau), mais cher
- Pouvons-nous avoir un multiplexage statistique avec des garanties fermes?



# Contrôle d'admission

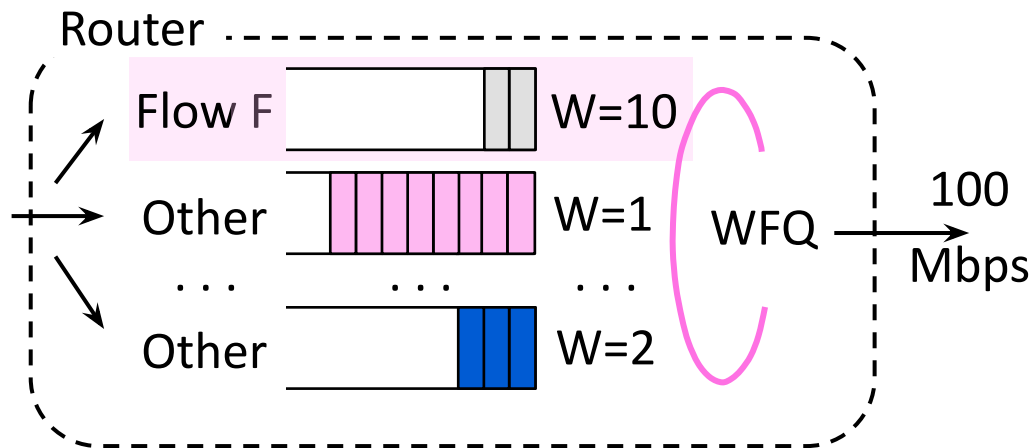
- Supposons qu'un flot  $F$  ait besoin d'un débit  $\geq R$  Mbps et d'un délai  $\leq D$  secs
- Doit-on l'accepter ou le rejeter ?
  - C'est du **contrôle d'admission**
  - Le rejet devrait être peu fréquent
- Le point clé est que nous avons besoin de contrôler la charge pour obtenir des garanties



# Garantie de débit

Au niveau du routeur

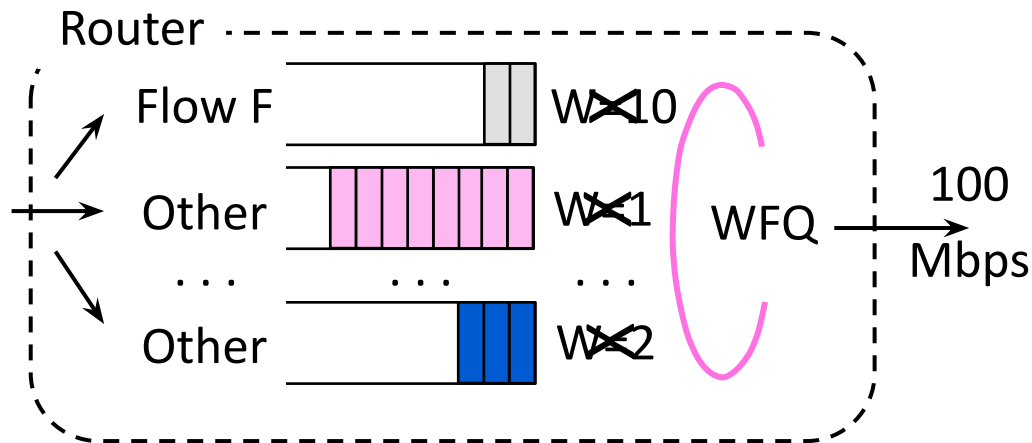
- WFQ peut garantir un débit en sortie
  - Quelle débit le flot F obtient exactement ?



# Garantie de débit

Au niveau du routeur

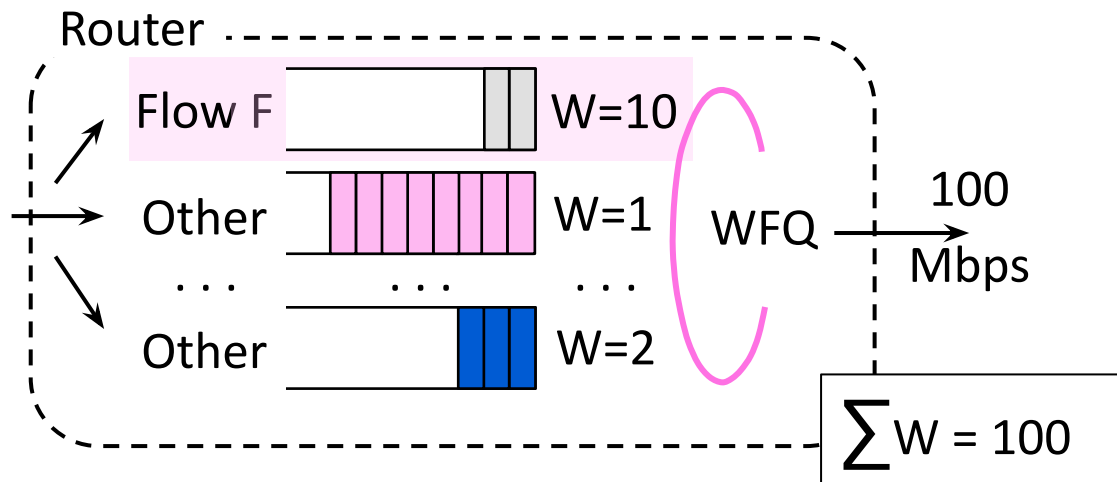
- Soit  $n$  flots avec un poids,  **$weight=1$** 
  - Chaque flot obtient  $1/n$  à saturation
  - Ou au moins  $100/n$  Mbps



# Garantie de débit

Au niveau du routeur

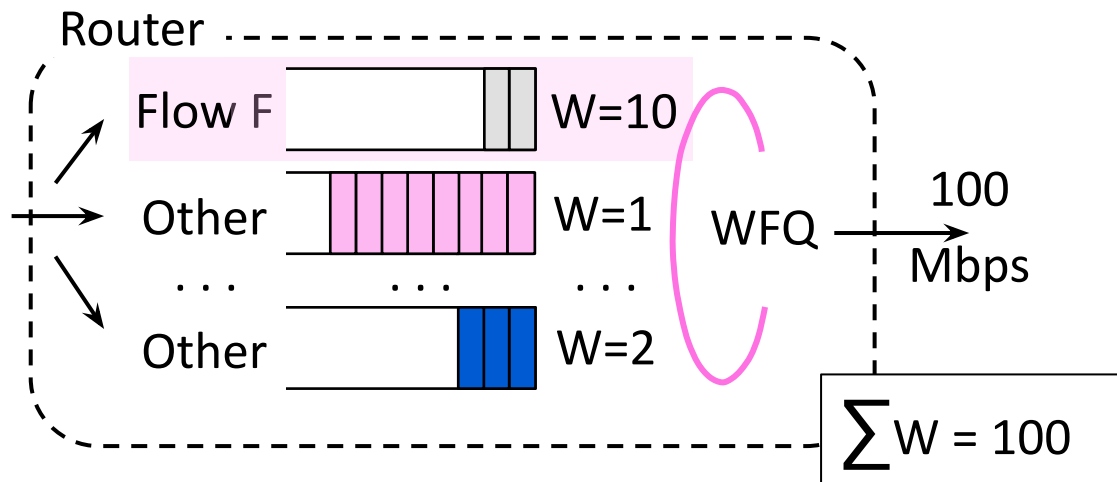
- Soit F avec un poids 10
  - Supposons que la somme des poids est 100



# Garantie de débit

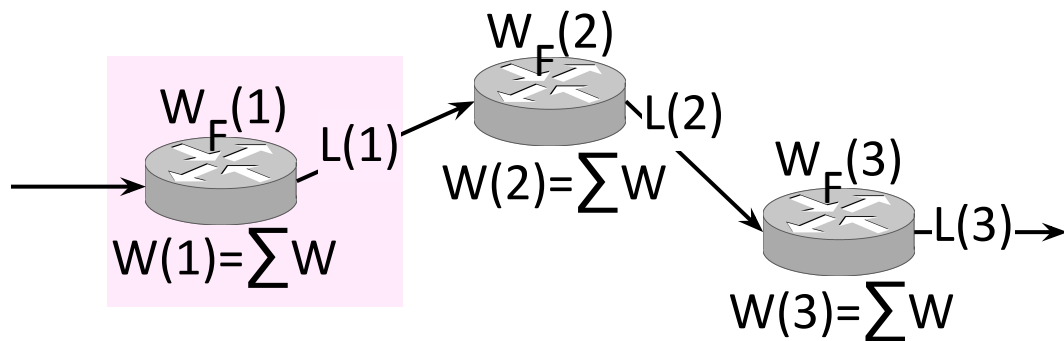
Au niveau du routeur

- Soit F avec un poids 10
  - $F \geq (10/100) \cdot 100 = 10 \text{ Mbps}$



# Garantir le débit réseau

- Possible en obtenant des garanties sur chaque routeur

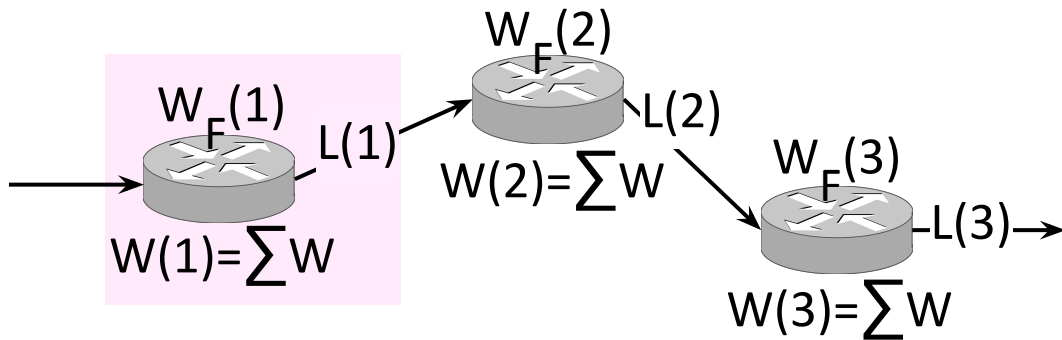


# Garantir le débit réseau

- Condition pour chaque routeur:

Pour tout routeur  $i$ :

$$W_F(i) / W(i) * L(i) \geq R \text{ Mbps}$$



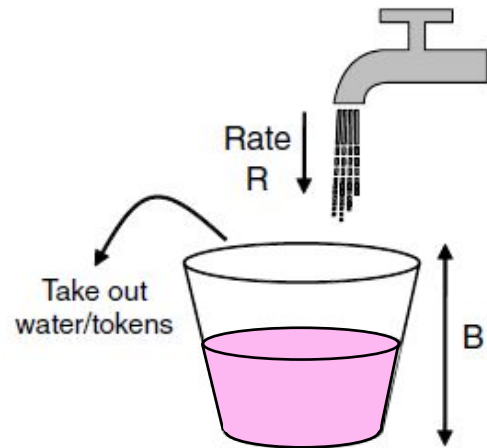
# Garantie de délai

- Qu'en est-il du *queuing delay* ?
  - A quel point le délai peut-il être supérieur à la latence, compte tenu du taux garanti ?
- Tout dépend du trafic
  - Si il dépasse  $R$ , alors les files augmentent et donc le délai de traversée
- Besoin de lisser le trafic pour garantir le délai
  - On utilise les *token buckets*

# Garantie de délai

Au niveau du routeur

- On suppose qu'un flot  $F$  est conditionné par un *token bucket*  $(R, B)$ 
  - Débit sur le long-terme  $\leq R$  Mbps
  - Rafale à court-terme  $\leq B$  bits

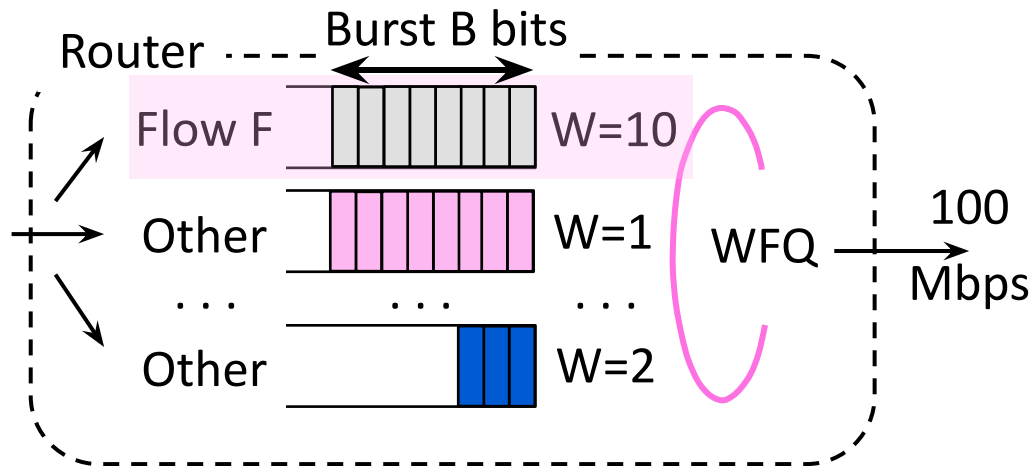




# Garantie de délai

Au niveau du routeur

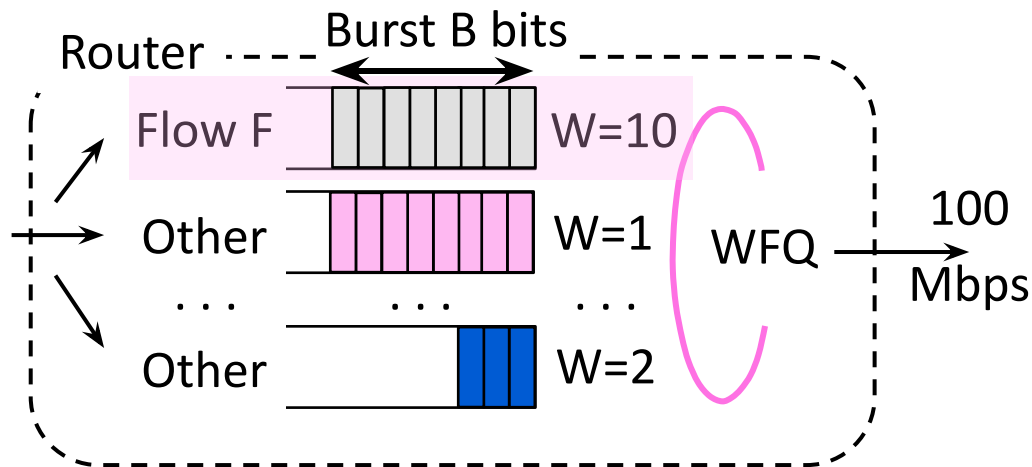
- Quel est le délai du flot F au niveau du routeur ?
  - Trafic régulé par un *token bucket* (R, B)
  - WFQ avec des poids pour un débit  $\geq R$  Mbps



# Garantie de délai

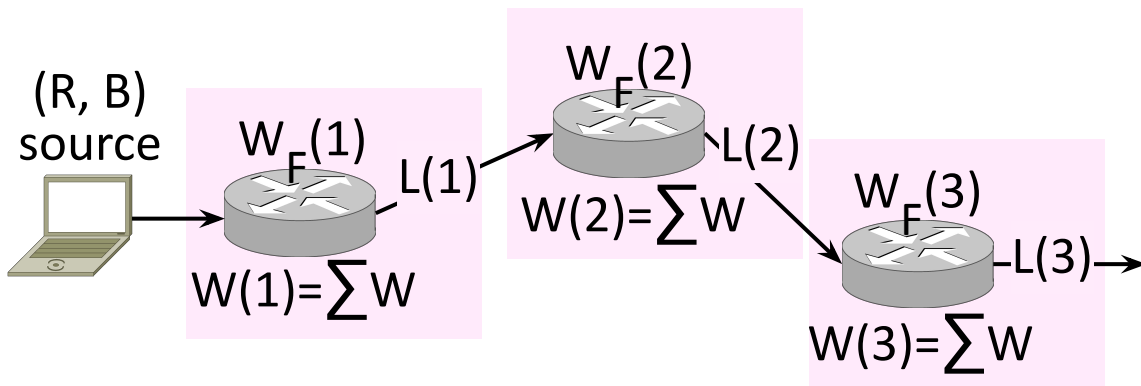
Au niveau du routeur

- Dans le pire cas, une rafale de taille B arrive en une fois
  - *queuing delay*  $\leq B/R$  secs



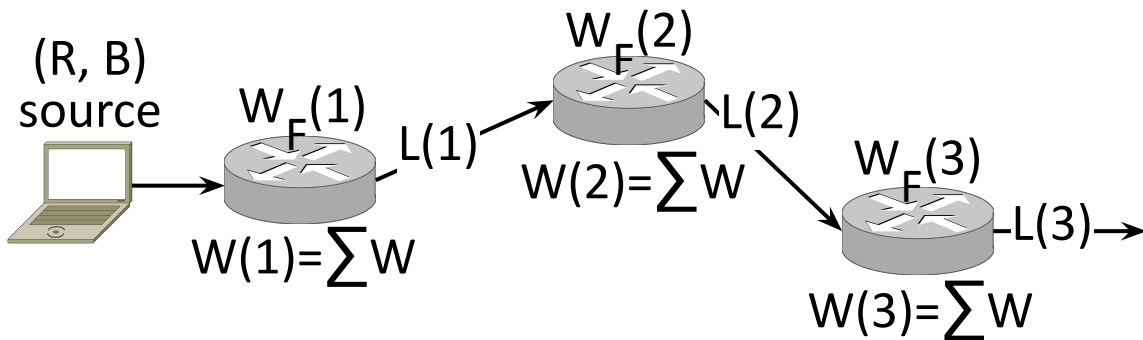
# Garantir le délai réseau

- Quel est le délai sur un chemin de  $n$  routeurs ?
  - Problème difficile, chaque routeur ajoute du délai
  - Une borne  $n*B/R$  est trop approximative
  - Argument intuitif à suivre...



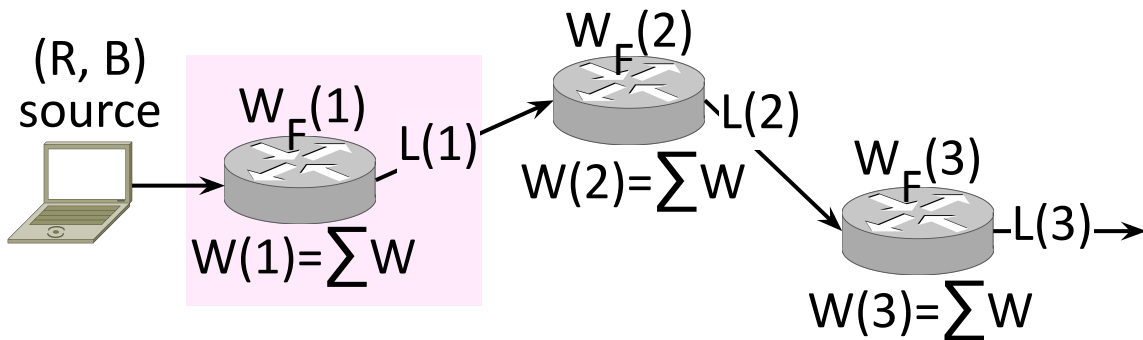
# Garantir le délai réseau

- Si le trafic est parfaitement lissé à un débit  $R$  (pas de rafale) alors *queuing delay*=0
  - Un paquet entre et est servi de suite
  - Le délai devient la latence de propagation et de transmission



# Garantir le délai réseau

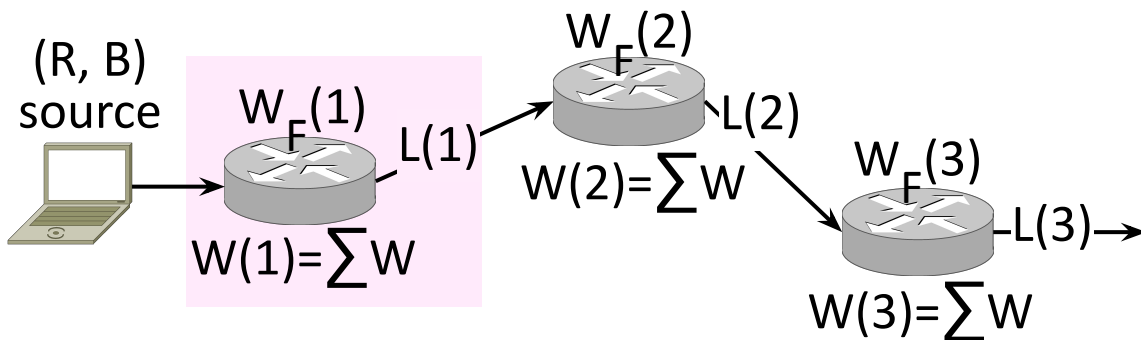
- On observe que si le trafic “paye” une rafale B sur un routeur, il est lissé sur ceux qui suivent (i.e., *pay at once*)
  - On dit qu’une rafale ne paie qu’une fois



# Garantir le délai réseau

- Ainsi le délai du chemin correspond à :

$$\text{Delay} \leq \text{Latency terms} + B/R$$



# Conclusion sur la garantie de débit/délai

- Etant donné un réseau avec :
  - Un trafic lissé de paramètres (R, B)
  - Des routeurs ordonnancés par WFQ avec des poids corrects
  - Un partage via multiplexage statistique
- Nous pouvons garantir à un flot un débit minimum et un délai maximum
  - Débit  $\geq R$  Mbps
  - Délai  $\leq \text{latency} + B/R$  secs
  - Indépendamment du comportement des autres flots

# Crédits

Cette présentation s'inspire des slides réalisés par David WETHERALL illustrant le livre d'Andrew TANENBAUM : COMPUTER NETWORKS, 5th Edition, © 2011, Pearson Education.

Merci à son auteur.

La traduction en français des slides originaux, les modifications adaptations et mises à jour, ainsi que l'ajout de slides et concepts complémentaires, ont été effectués par Emmanuel LOCHIN pour les cours ENAC.

<http://elochin.github.io/>