Media Streaming

Lecture given by Emmanuel Lochin

ISAE-SUPAERO

Le multimédia aujourd'hui

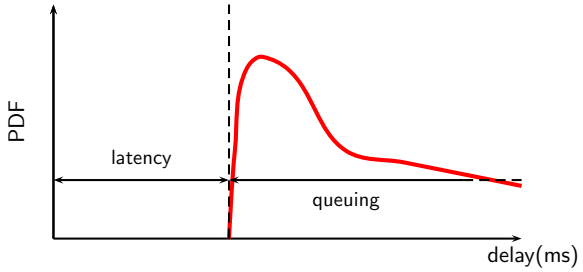
- Utilisation intensive de la diffusion de contenu vidéo sur Internet
- Ces diffusions sont réalisées sur un réseau *best-effort*
  - C'est le cas pour Youtube, Netflix, Coursera, FUN, ...
- Certaines diffusions peuvent opérer sur des réseaux particuliers offrant une garantie de service
  - C'est le cas pour la VoD que vous achetez chez votre ISP
- Ce mode est appelé *streaming*. Il en existe trois classes :
  - **stored streaming** : c'est le cas des applications présentées ci-dessus
  - **live streaming** : lorsque les données sont produites en *live* (e.g. capture depuis une caméra)
  - **interactive live streaming** : média interactif telles la VoIP (Skype), la vidéo-conférence, ...

Stored or live-streaming

- Le *stored streaming* abrégé ici en *streaming* nécessite moins de ressources :
  - Il n'y a qu'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
- Le *live streaming* et *interactive live streaming*
  - Peut opérer de façon unidirectionnelle (video *live*) ou bidirectionnelle (VoIP, vidéoconférence)
  - Traite des flots dits "temps-réel"
  - Sont sensibles à la gigue

Qu'est-ce que la gigue ?

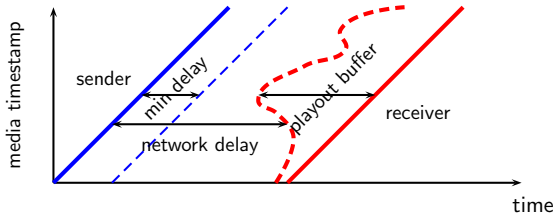
- Le délai du réseau est variable → c'est la gigue *jitter*
- Il correspond à la latence des messages + le délai de traversée des files d'attente



Comment minimiser la gigue ?
Le play-out buffer

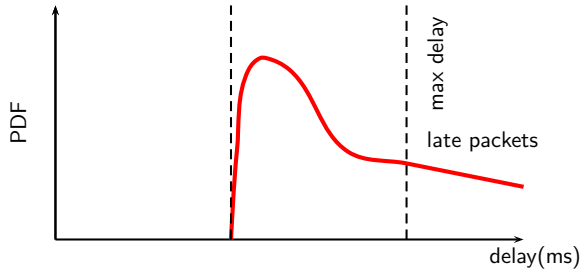
- Son objectif est de minimiser la gigue en lissant le délai du réseau
- Consiste en l'implémentation d'un buffer de stockage du côté de l'application de lecture
- Sa taille impacte évidement sur le départ de la lecture du flot
- Il permet de stocker les données du flot en attente de lecture, retransmission/correction en fonction du mécanisme de fiabilité utilisé (TCP, FEC, ...)

Principe du play-out buffer



Dimensionnement du play-out buffer

- On choisit la taille du *play-out buffer* en fonction d'une borne acceptable maximale de délai
- Au delà les paquets ne sont pas joués par l'application
- Cela résulte en un compromis entre délai et pertes
- Pour les applications interactives, la taille du *play-out buffer* est petite. On observe alors des pertes sur la vidéo ou à l'audio

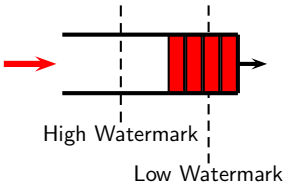


Valeurs standards pour les flots intéractif

- Pour la VOIP
  - Valeur du délai maximum tolérable 400ms
  - Une bonne qualité correspond à un délai inférieur à 150ms
    - ⋆ paquetisation applicative incluse
  - Tolérance aux pertes entre 1% et 10%
- Pour la vidéo
  - Pour garantir l'intéractivité le délai doit être inférieur à 100ms
- Le délai → plus gros problème !
- Méthodologies d'évaluation des flots multimedia
  - Pour la VoIP : norme de l'ITU - Mean Opinion Score (MOS)
    - ⋆ Score subjectif de qualité allant de 1 à 5
  - Pour la video : PSNR

Contrôle de flot du play-out buffer

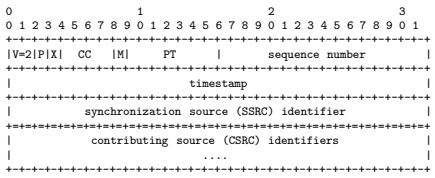
- Du côté client, pour le *stored streaming* le fichier est stocké localement (au contraire d'une données *live*)
- Ceci permet au client d'émettre les données jusqu'à la capacité d'émission du lien
- Il peut donc y avoir débordement côté récepteur
- Pour cela les applications de lecture implémentent un type de contrôle de flot basé sur un *watermarking*



Quel protocole transporte les données multimédia ?

RFC1889

- Généralement transportée dans RTP (*Real-Time Transport Protocol*)
- RTP définit un format de paquets afin de transporter l'audio et la vidéo
- Travaille uniquement au dessus d'UDP (on ne fait pas de *real-time* avec TCP !)
  - RTP est une sur-couche transport qui étend UDP
  - L'en-tête contient des informations sur le format des données multimédia, les *timestamp*, numéro de séquences, ...
- Les données multimédia suivent un format standardisé : G711, H264, ...



RTP et QoS

- RTP ne fournit aucun mécanisme d'assurance de délivrance des données dans un temps donné ou tout autre garantie de QoS
- RTP est un protocole bout-en-bout : aucune prise en charge du réseau au niveau des routeurs intermédiaires

Real-Time Control Protocol (RTCP)

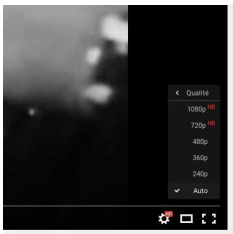
- Travaille de pair avec RTP
- RTCP correspond en quelque sorte à la voie retour
- Chaque participants transmettent périodiquement des paquets RTCP aux autres participants
- Les paquets RTCP contiennent des rapports d'émission ou réception
  - Statistiques utiles pour l'application : paquets émis, perdus, inter-arrivée, gigue, ...
- Ces statistiques sont utilisées afin d'améliorer les performances
  - L'émetteur peut adapter sa transmission en fonction de ces retours

Synchronisation des flux

- RTCP peut synchroniser différents médias au travers d'une session RTP
  - Audio/vidéo d'une vidéoconférence
- Pour cela RTCP utilise des *timestamps*
- Ces *timestamps* permettent de synchroniser des flux entre-eux

Gestion de la capacité

- Le serveur de *streaming* peut encoder différentes qualités audio, vidéo
- Meilleure est la qualité, plus la capacité nécessaire est grande
- Le serveur choisit la meilleure qualité à envoyer au client en fonction de la capacité réseau disponible



Quel protocole de transport pour le streaming ?

- UDP
  - Pour minimiser le délai des message des sessions interactives ou temps-réel
  - Besoin d'un mécanisme de compensation des pertes
- TCP
  - Lorsque le délai (de démarrage) n'est pas essentiel
  - Retransmission des paquets perdus
  - HTTP/TCP passe plus facilement les *firewalls*

Composants du media streaming

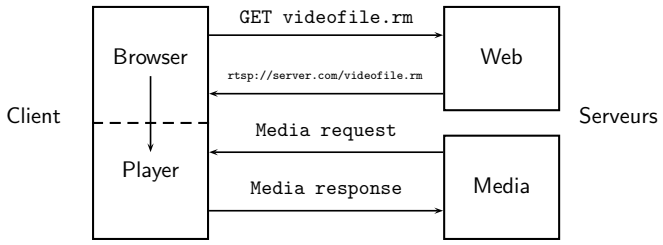
- Une session de *media streaming* se décompose en plusieurs parties
- Nous venons de voir qu'il était composé :
  - D'un mécanisme *play-out buffer* et de *watermarking*
  - D'un media de transport (e.g. HTTP) associé à un protocole de transport (e.g. TCP ou UDP)
- Mais il est également composé :
  - D'un mécanisme de signalisation (e.g. *Real-Time Streaming Protocol* - RTSP). Utiliser pour initialiser et mettre en place :
    - ★ un appel VoIP
    - ★ une vidéo-conférence
    - ★ les paramètres de transfert de la vidéo (qualité, ...)
  - Les standards RTSP et HTML évolue très vite. Maintenant HTML5 inclut un *video player*. Devenu un standard de facto
- A noter que les contenus de *media streaming* sont généralement très proches des utilisateurs et hébergés dans des CDN ou des replicats

Real-Time Streaming Protocol
RFC 2326

- Agit comme un *network remote controller*
- Supporte les opérations suivantes :
  - Récupérer un média depuis un serveur
  - Gérer les invitations d'un média, par exemple, à une vidéo-conférence
  - Gère l'enregistrement d'une conférence
- Indépendant du protocole de transport sous-jacent
- Supporte tous les formats de description de session (SDP, XML, ...)
- Design similaire à HTTP avec différences mineures (e.g. le serveur maintient un état de session)
- Unicast et multicast

Media streaming avec RTSP

- Pour obtenir une vidéo : requête `GET` depuis votre navigateur web
- On obtient alors une URL du *metafile* correspondant à la vidéo demandée : `rtsp://server.com/videofile.rm`
- Le navigateur invoque alors le *media player* via RTSP
- La vidéo est alors transmise avec (par exemple) RTP au dessus de TCP ou UDP



Media streaming avec HTTP

- La méthode est différente avec HTTP. C'est la méthode couramment utilisée aujourd'hui
- On récupère tout d'abord un fichier de description des données (index des clips, débits, différents encodage...) via une requête `GET`
- Le navigateur va alors lancer plusieurs requêtes HTTP et récupérer des segments qu'il stockera dans le *playout buffer*
- En fonction du taux d'occupation du *buffer*, le *media player* va sélectionner l'encodage le plus adéquat
- Les nouveaux standards tels DASH (*Dynamic Adaptive Streaming over HTTP*) ou HTML5 implémentent ce système
- Ils existent des versions propriétaires chez Adobe, Apple, Netflix, ...
- Problème : il n'y a aucune adaptation **dynamique** de l'encodage en fonction de la capacité disponible
- Solution : DASH s'adapte **dynamiquement** à la capacité disponible

Dynamic Adaptive Streaming over HTTP

- Du côté serveur
  - Divise le fichier vidéo en multiples *chunks*
  - Chaque *chunks* est stocké et encodé à différents taux
  - Un fichier spécial, le *manifest file*, fournit les URLs des différents *chunks*
- Du côté client
  - Mesure périodiquement la capacité entre le client et le serveur
  - Consulte le *manifest file* et demande un *chunk* à la fois
  - Choisit le bon taux d'encodage en fonction de la capacité disponible
  - Peut choisir différent taux d'encodage à différent moment de la diffusion (suivant la capacité disponible)
  - Demande le *chunk* au bon moment pour éviter famine ou saturation du *buffer*
  - Choisit la bonne URL localisant le *chunk* le plus proche du client

La VoIP
Session Initiation Protocol - SIP

- Protocole d'initialisation de session standardisé par l'IETF → RFC3261
- SIP est une vision sur le long terme de ce que devrait être la téléphonie sur IP
  - Toutes les appels téléphoniques et vidéo-conférences se feront via Internet
  - Les personnes sont identifiés par des identifiants ou des adresses emails plutôt que des numéros de téléphones
  - Vous pouvez atteindre l'appelant quelque-soit sa localisation géographique et l'IP qu'il utilise à l'instant de l'appel
- Attention ! SIP n'est pas Skype

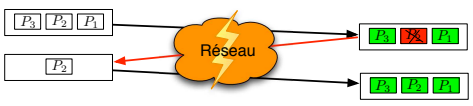
La VoIP
Fonctions proposées par SIP

- Initialise l'appel, l'ouverture et sa fermeture et négocie les paramètres d'encodage audio
- Détermine l'IP courant de l'appelé
  - Correspondance effectuée entre l'identifiant de l'utilisateur SIP et son adresse IP
- Gestion de l'appel
  - Ajout/retrait d'utilisateurs durant l'appel
  - Changement d'encodage
  - Transfert, mise en attente

La VoIP
Skype

- Application propriétaire → spécifications exactes non disponibles
- Skype est un réseau dit en *overlay* → réseau P2P
- Les clients s'enregistrent auprès d'un *supernode* puis la connexion des clients s'effectue
- Une communication entre deux clients Skype se trouvant sur le même réseau dialogue directement
- Lorsqu'au moins un des deux clients se trouve derrière un NAT, une connexion est maintenue avec le *supernode* pour que la communication fonctionne
  - Quel problème de sécurité voyez-vous ?

Mécanismes réactifs de récupération des pertes
ARQ



- Mécanismes basés sur la retransmission, suite à la détection d'une perte par :
  - Acquittement, acquittement négatif : ACK, SNACK, RTO, ...
- $RTO > RTT$: Délai de récupération d'une perte $\geqslant \frac{3}{2} \times RTT$

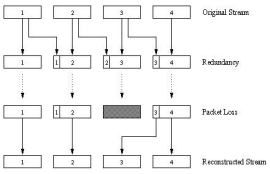
- *Forward Error Coding* (FEC) → l'émetteur envoi n paquets :
  - k paquets sources
  - $n-k$ suivis des $n-k$ paquets redondants
- Code "idéal" : x redondances suffisent pour x pertes
  - Reed Solomon
  - Complexité fonction de la taille du bloc
- LDPC, LT code, Raptor, ...
  - Faible complexité
  - Inefficace pour de petites tailles de blocs
- Taux de redondance : $R = (n-k)/n$
- Taux de codage : k/n

Mécanismes pro-actifs de récupération des pertes
Code à effacement

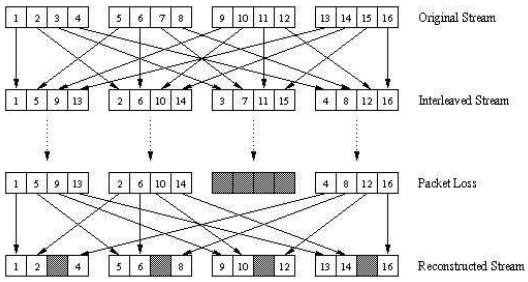
- *Forward Error Coding* (FEC) → l'émetteur envoi n paquets :
  - k paquets sources
  - $n-k$ suivis des $n-k$ paquets redondants
- Code "idéal" : x redondances suffisent pour x pertes
  - Reed Solomon
  - Complexité fonction de la taille du bloc
- LDPC, LT code, Raptor, ...
  - Faible complexité
  - Inefficace pour de petites tailles de blocs
- Taux de redondance : $R = (n-k)/n$
- Taux de codage : k/n

Autre schéma FEC pour la VoIP : le piggybacking

- Principe : on *piggyback* un flot de qualité inférieure
- L'information redondante devient un flot audio de moindre résolution
  - Par exemple : flot nominal en PCM 64kbps → flot de redondance GSM 13kbps



Principe de l'entrelacement



En résumé

- Les flots interactifs ou "temps-réel" (i.e. sensibles au délai) utilisent UDP pour ne pas être contraint par le contrôle de congestion et les retransmissions TCP
- Peuvent utiliser un "très petit" *playout buffer*
- Du côté serveur : adapte l'encodage en fonction de la capacité disponible
- Mécanisme de récupération de pertes au dessus d'UDP
  - code à effacement, entrelacement, ...
  - retransmission si le temps le permet

Mécanismes pro-actifs de récupération des pertes

- Code en bloc
  - la fiabilité augmente avec la taille des blocs
  - la complexité augmente avec la taille de blocs
  - le délai augmente avec la taille du bloc
  - efficacité dépend de la configuration (n,k)
- ARQ
  - fiabilité totale
  - délai de reconstruction fonction du RTT
  - Efficacité
    - ⋆ optimale si forte tolérance au délai
    - ⋆ décroit fortement autrement

H-ARQ : code en block + ARQ

- **Fiabilité totale et délai moyen moindre comparé à l'ARQ**
- **Dans les pires cas, le délai est toujours fonction du *RTT***

Mécanisme de fiabilité

- Objectif : disposer d'un mécanisme générique
  - Proche de la capacité du canal
  - Complexité abordable
- Problème : deux classes d'application → contrainte de délai et fiabilité totale
  - Code à effacement : compromis entre délai, robustesse et complexité
  - Avec ARQ/H-ARQ : le délai est un multiple du RTT dans le pire cas

Solution

- Codage à fenêtre élastique
- Fiabilité totale + délai de récupération des pertes paramétrable et indépendant du RTT

On-the-fly coding

- Le principe de base connu sous le nom de "*on-the-fly coding*" consiste à utiliser une fenêtre d'encodage incluant tout les paquets sources émis mais non accusés
- Nous présentons par la suite un mécanisme de ce type développé à l'ISAE nommé Tetrys
- Il existe des alternatives : RLNC (MIT)
- Ces codes peuvent s'intégrer à des niveaux applicatifs ou noyau
  - C'est le cas de RLNC avec Coded-TCP

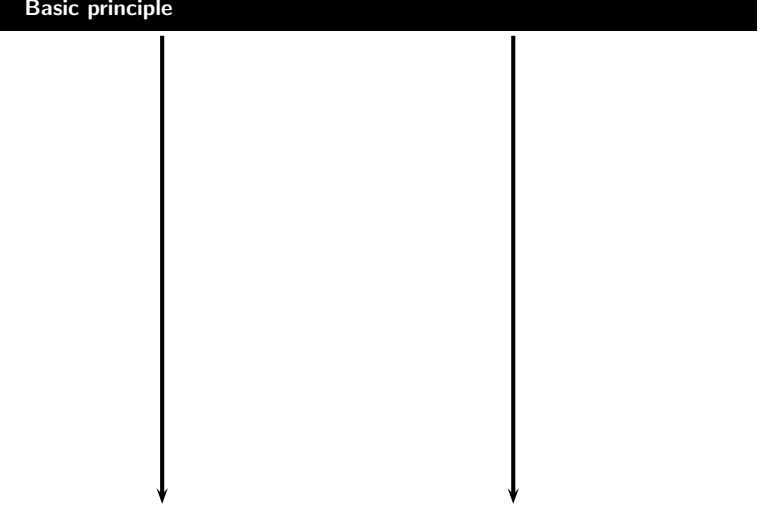
Étapes du mécanisme

1. Envoi de paquets de données ; ajout à la fenêtre d'encodage
2. Envoi de paquets de redondance
3. Les paquets perdus sont reconstruits dès que leur nombre est égal au nombre de paquets de redondance reçus
4. Les paquets acquittés sont retirés de la fenêtre d'encodage

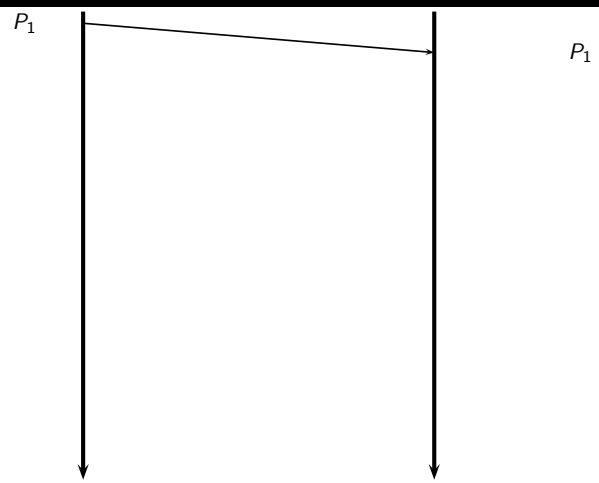
Notation

- P_i **le** i^{eme} **paquet source envoyé**
- W_E **la fenêtre d'encodage**
- $R_{(i..j)}$ **combinaison linéaire de tous les paquets présents dans** W_E
- $R = \frac{n-k}{n}$ **le taux de redondance**

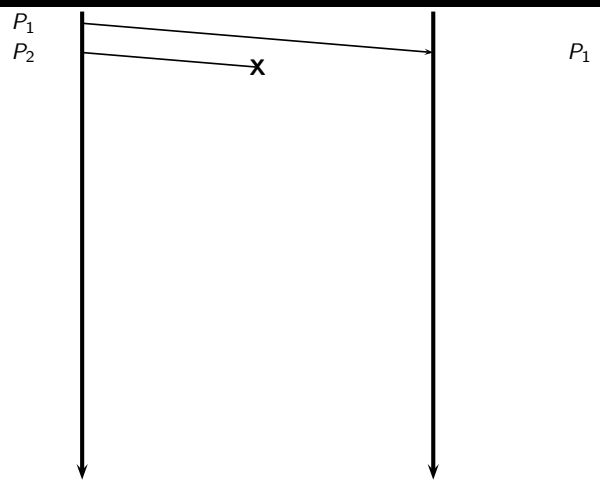
Basic principle



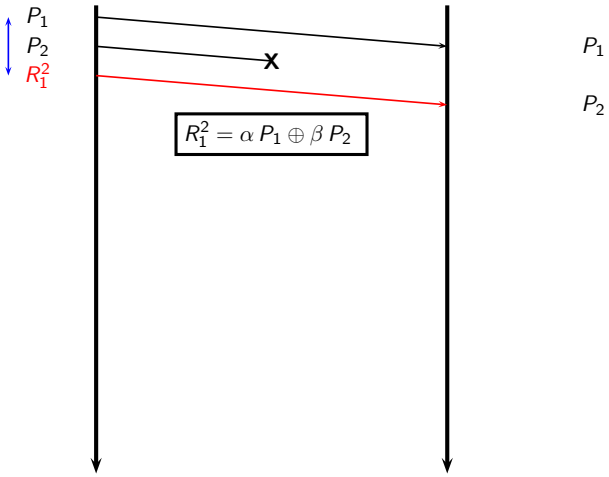
Basic principle



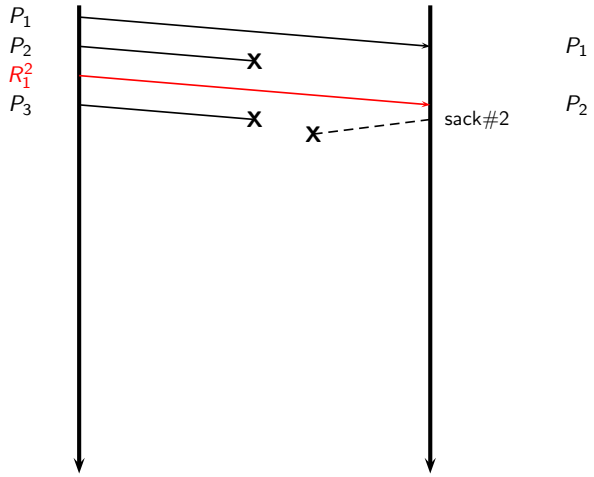
Basic principle



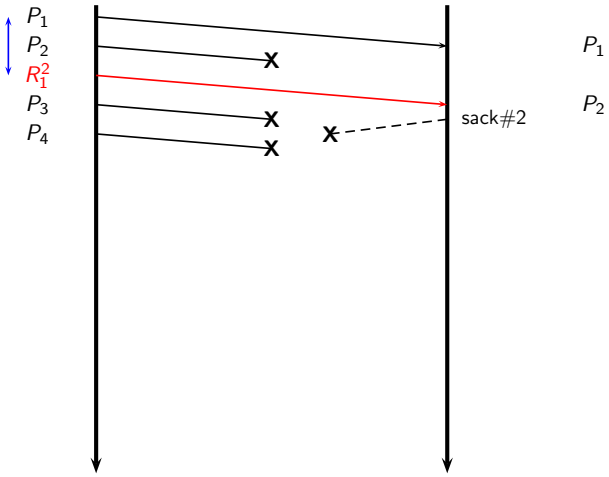
Basic principle



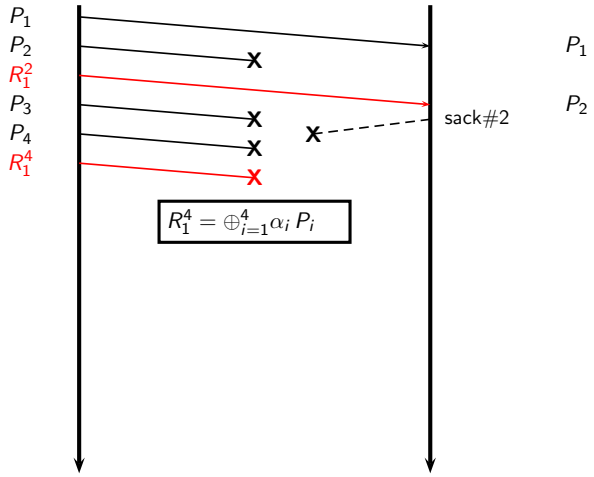
Basic principle



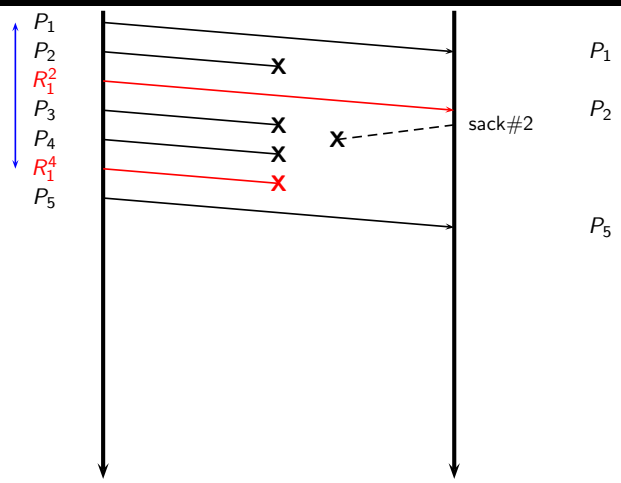
Basic principle



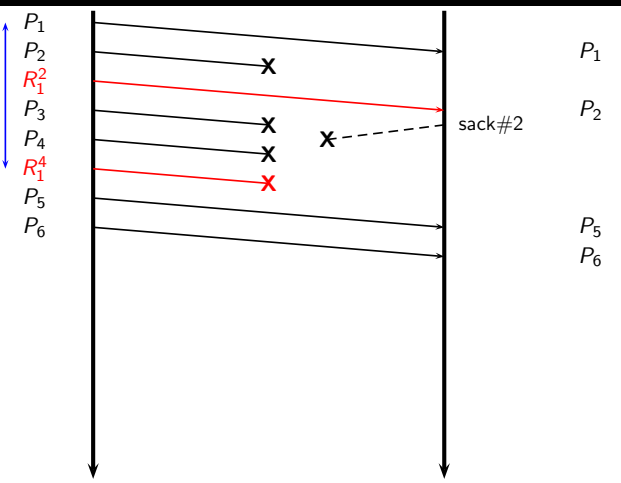
Basic principle



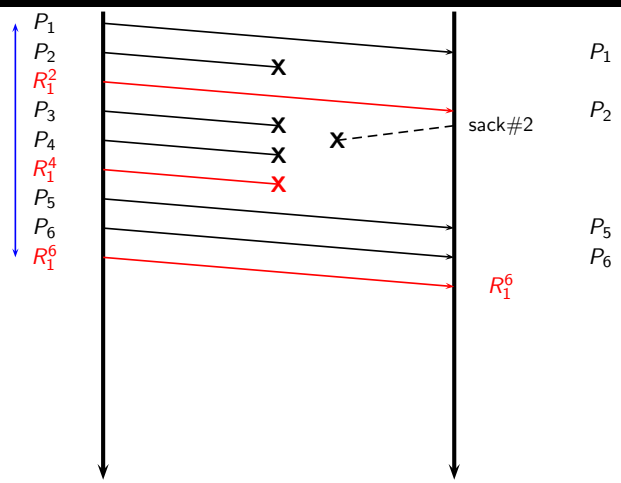
Basic principle



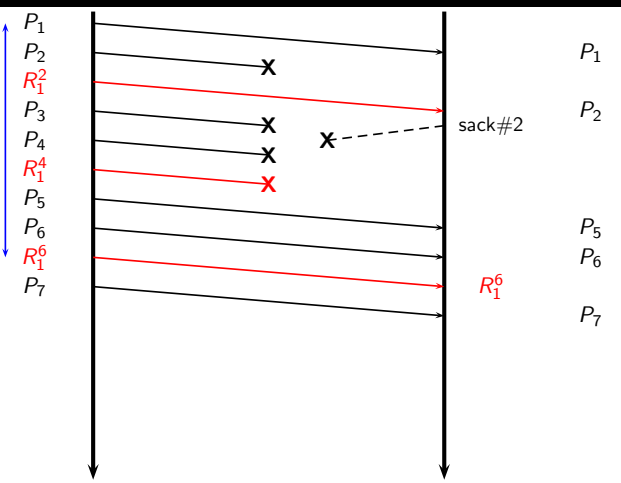
Basic principle



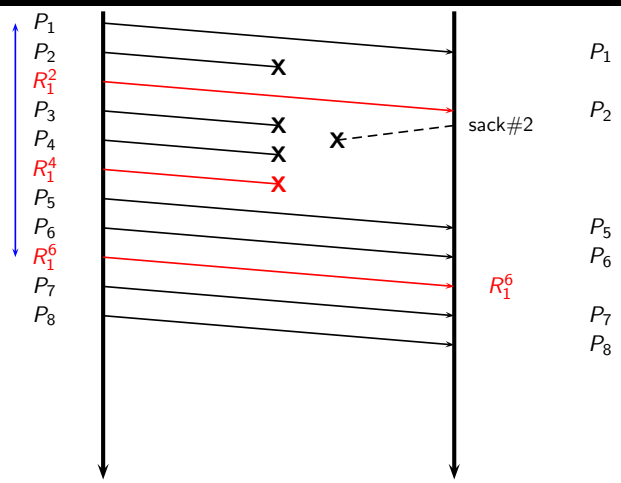
Basic principle



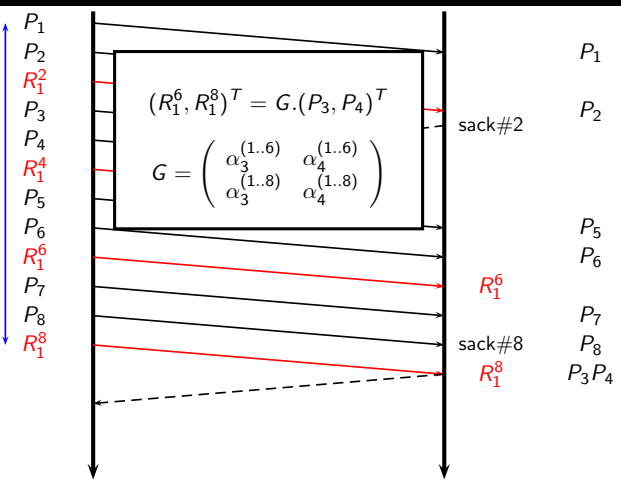
Basic principle



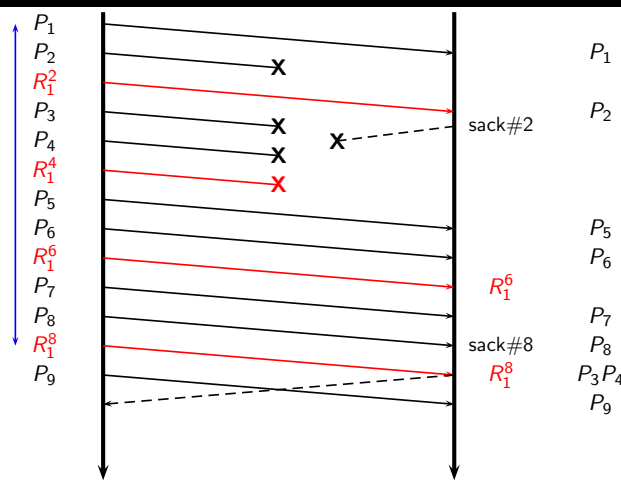
Basic principle



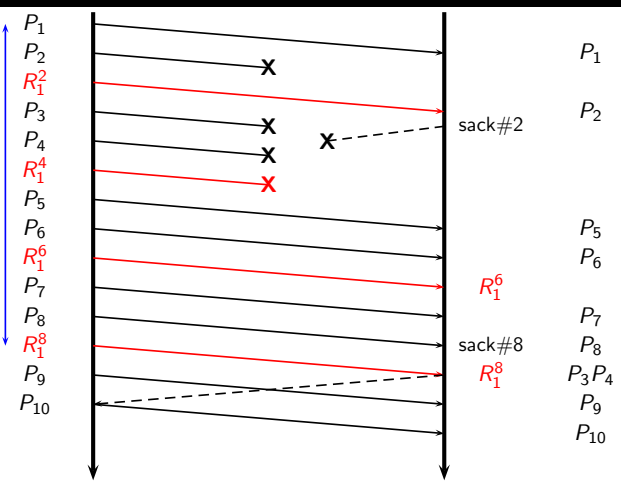
Basic principle

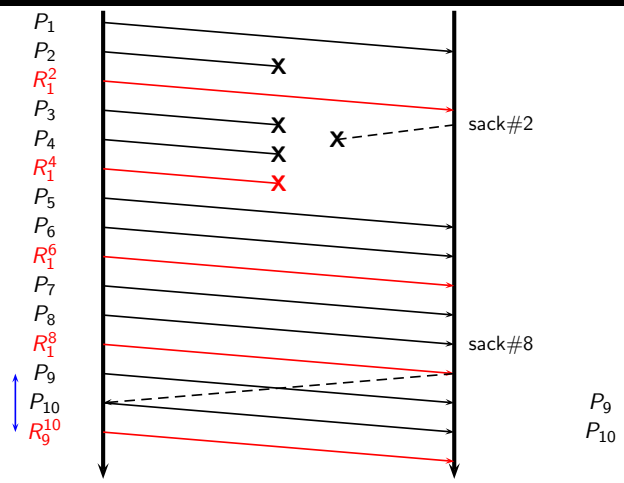


Basic principle



Basic principle





Emmanuel Lochin `http://personnel.isae.fr/emmanuel-lochin/`