Etude de l'exploitation des failles du protocole BitTorrent

Plan de la présentation

- Présentation du protocole *BitTorrent*
- Proposition d'un modèle informatique
- Etude de la robutesse du protocole face aux comportements abusifs à l'aide d'un programme implémentant le modèle

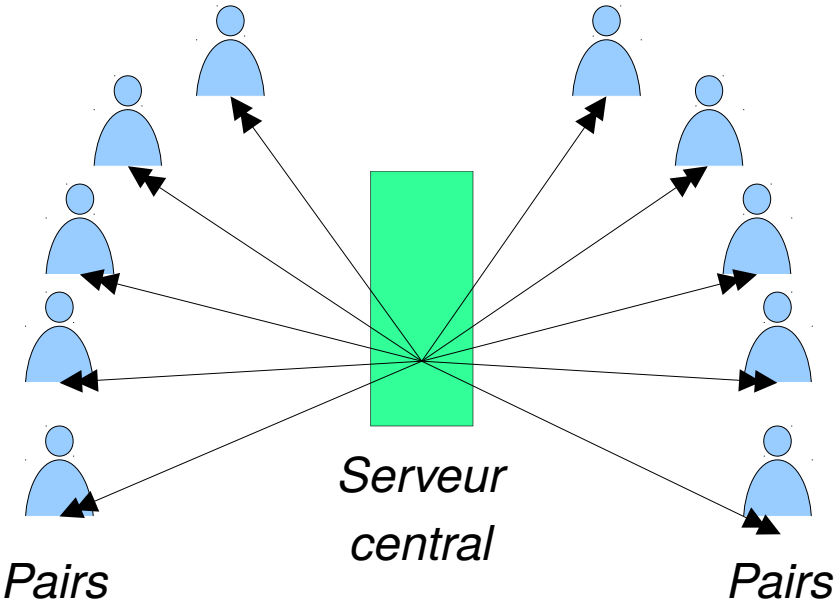
Présentation du protocole

Le protocole de partage de donnée *BitTorrent*

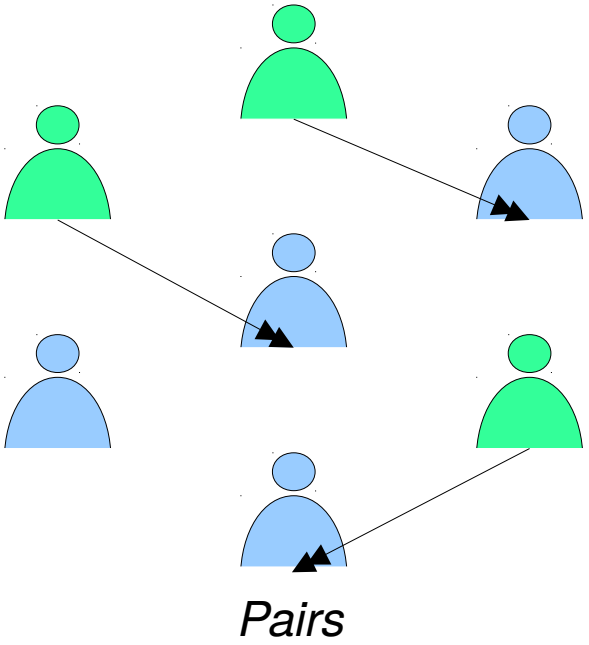
- Mis en place l'été 2002
- C'est un protocole pair à pair, donc décentralisé
- Partage de pièces de fichier plutôt que du fichier complet
- Chaque pair utilise la stratégie de coopération-réciprocité-pardon pour communiquer entre eux

Bram Cohen, créateur de BitTorrent

Un protocole pair à pair

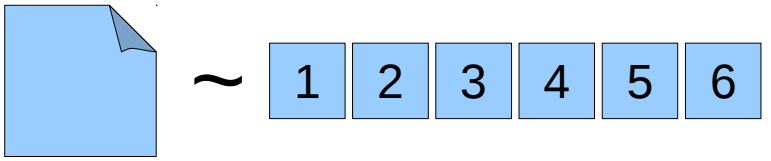


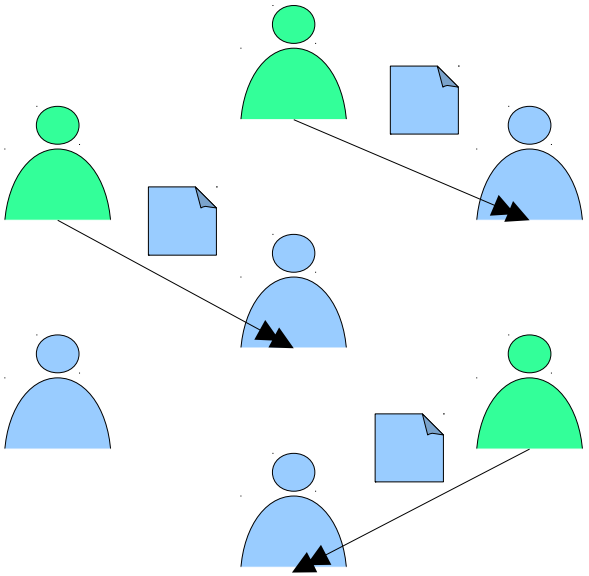
Modèle “classique” :
Le serveur central possède le fichier et l’envoie aux pairs



Modèle pair à pair :
Certains pairs possèdent le fichier, qu’ils envoient aux autres pairs

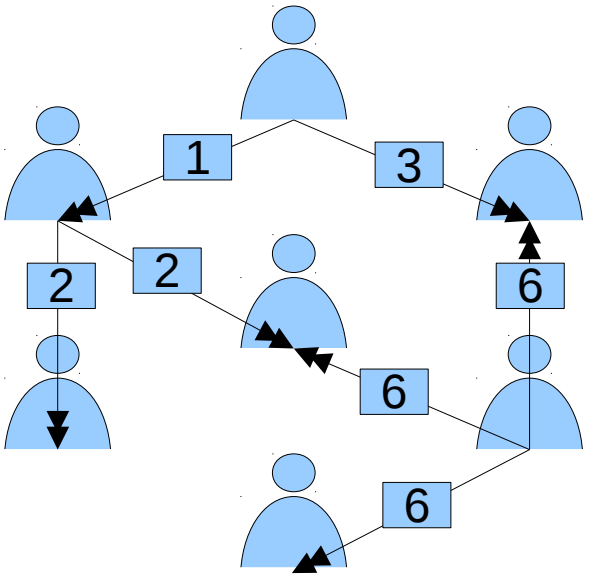
Morcellement du fichier en pièces





Pairs

Modèle pair à pair “classique” :
Le fichier est envoyé de pair à pair en entier, à travers une unique connexion

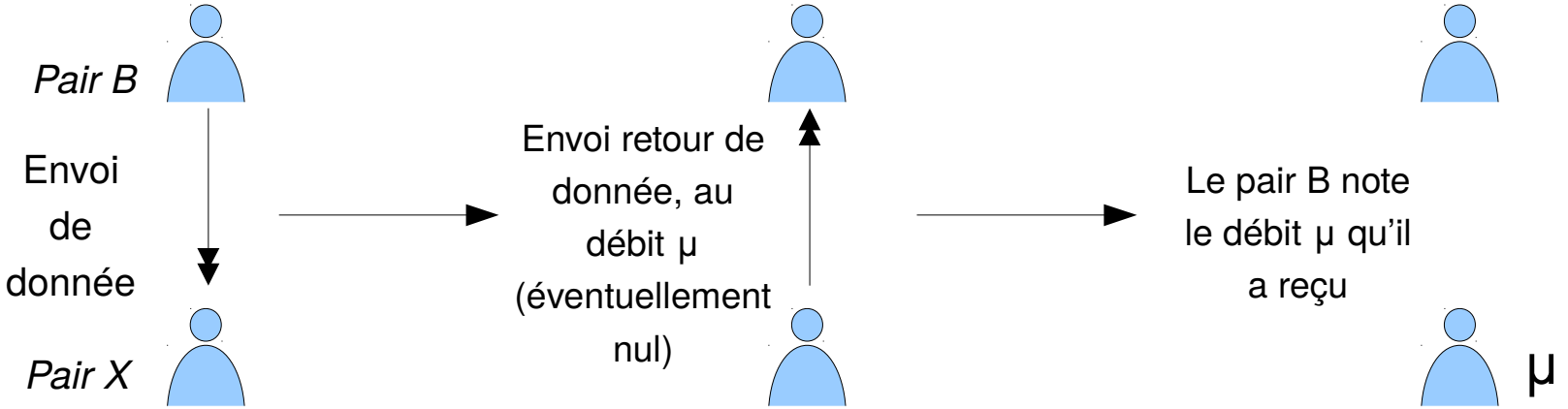


Pairs

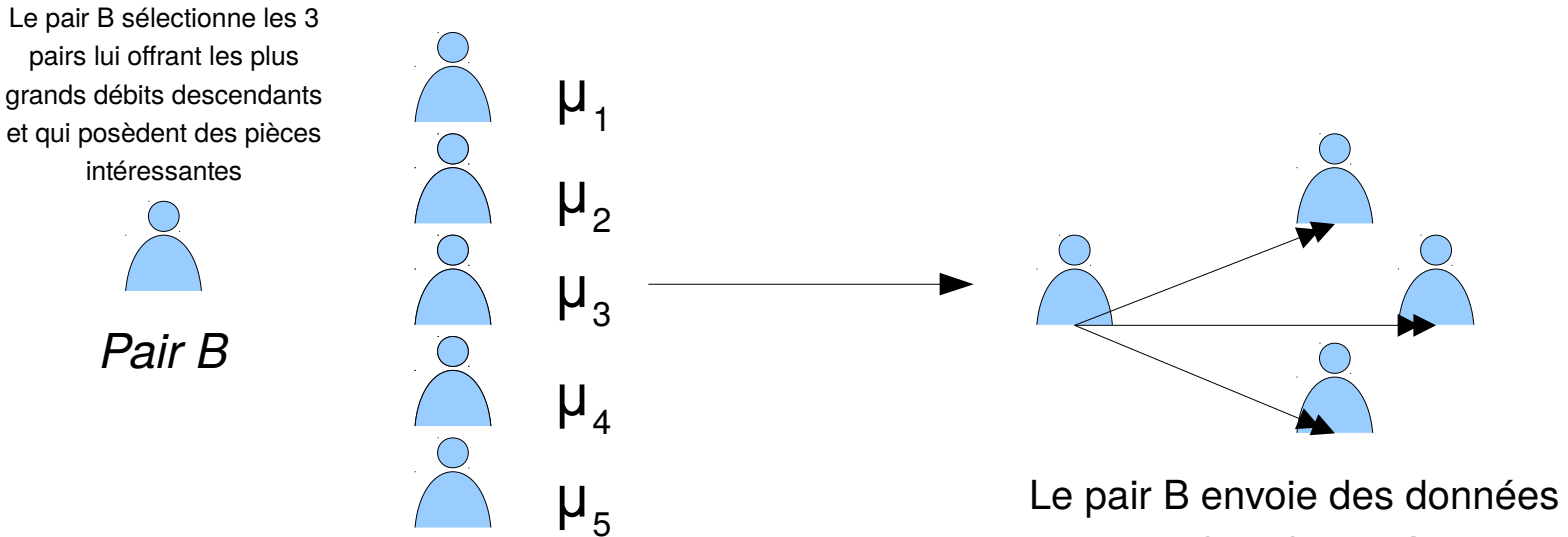
Modèle *BitTorrent* :
Le fichier est morcellé en pièce. Les pairs possédant des pièces les envoient aux autres : chaque pair maintient plusieurs connexions, par lesquelles les pièces sont reçues ou envoyées

Coopération-Réciprocité-Pardon : l'*unchoking* et l'*optimistic unchoking*

Optimistic unchoking (Coopération)



(Regular) unchoking (Réciprocité)



Modèle proposé

Représentation du réseau à un instant *t*

Objet	**Représentation informatique**	
	Structure informatique associée	**Signification**
Réseau	Dictionnaire (arbre de recherche) : entier → Objet **Pair** noté **m_pair** (std::map)	Chaque pair est identifié par un entier **n** et **m[n]** donne l'objet **Pair** contenant les données associées à ce pair
Pair	Liste d'entiers noté **l_v** (std::list)	Représente la liste des voisins du pair
	Liste d'entiers noté **l_unch** (std::list)	Représente la liste des voisins autorisés à télécharger à la suite de l'*unchoking*
	Liste d'entiers noté **l_opt** (std::list)	Représente la liste des voisins autorisés à télécharger à la suite de l'*optimistic unchoking*
	Tableau d'entiers de 0 à 16 de taille **n_piece** noté **f** (std::vector)	Chaque indice du tableau correspond à une pièce et **f[i]** donne le nombre de sous pièces de la pièce d'indice **l** que le pair possède

Représentation du réseau à un instant *t*

```
/**Attributes**/
private:
  File file;
  std::set<Peer::id> s_leecher;
  std::set<size_t> dw_pieces;
  std::set<size_t> pend_pieces;

  File::kbyte up_bandw;
  File::kbyte used_upbandw;
  File::kbyte dw_bandw;
  File::kbyte used_dwbandw;

  std::list<id> l_peer;
  std::list<id> unch_peers;
  std::list<id> optunch_peers;
  size_t nb_unchpeer;
  size_t nb_optunchpeer;

  bitfreq* on_bitfrequest;
  onevent* on_entrance;
  onpiece* on_piececomplete;
  onpiece* on_piececancel;
  onevent* on_piecefetching;
  onevent* on_peerfetching;
  onunch* on_unchoking;
  onunch* on_optunchoking;
  onevent* on_filecomplete;
```

Tableau des pièces du fichier

Liste des voisins

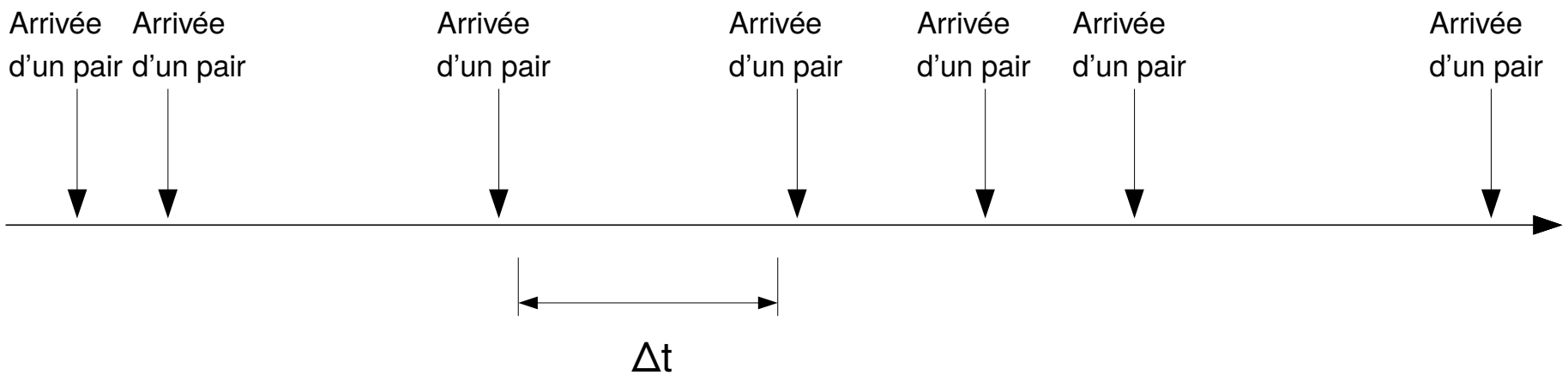
Liste des voisins autorisés à télécharger (*regular unchoking*)

Liste des voisins autorisés à télécharger (*optimistic unchoking*)

Fonctions déterminant le comportement du pair

peer.hpp : lignes 168 à 194

Arrivée des pairs dans le réseau



Modélisation par la loi exponentielle de paramètre λ

$$P(X > t) = 1 - exp(-\lambda t)$$

donc $$\Delta t = -\frac{ln(1-u)}{\lambda}$$

Dans l'implementation, u est une variable aléatoire suivant une loi uniforme continue portant sur l'intervalle [0; 1[. Une valeur est tirée à chaque calcul de Δt

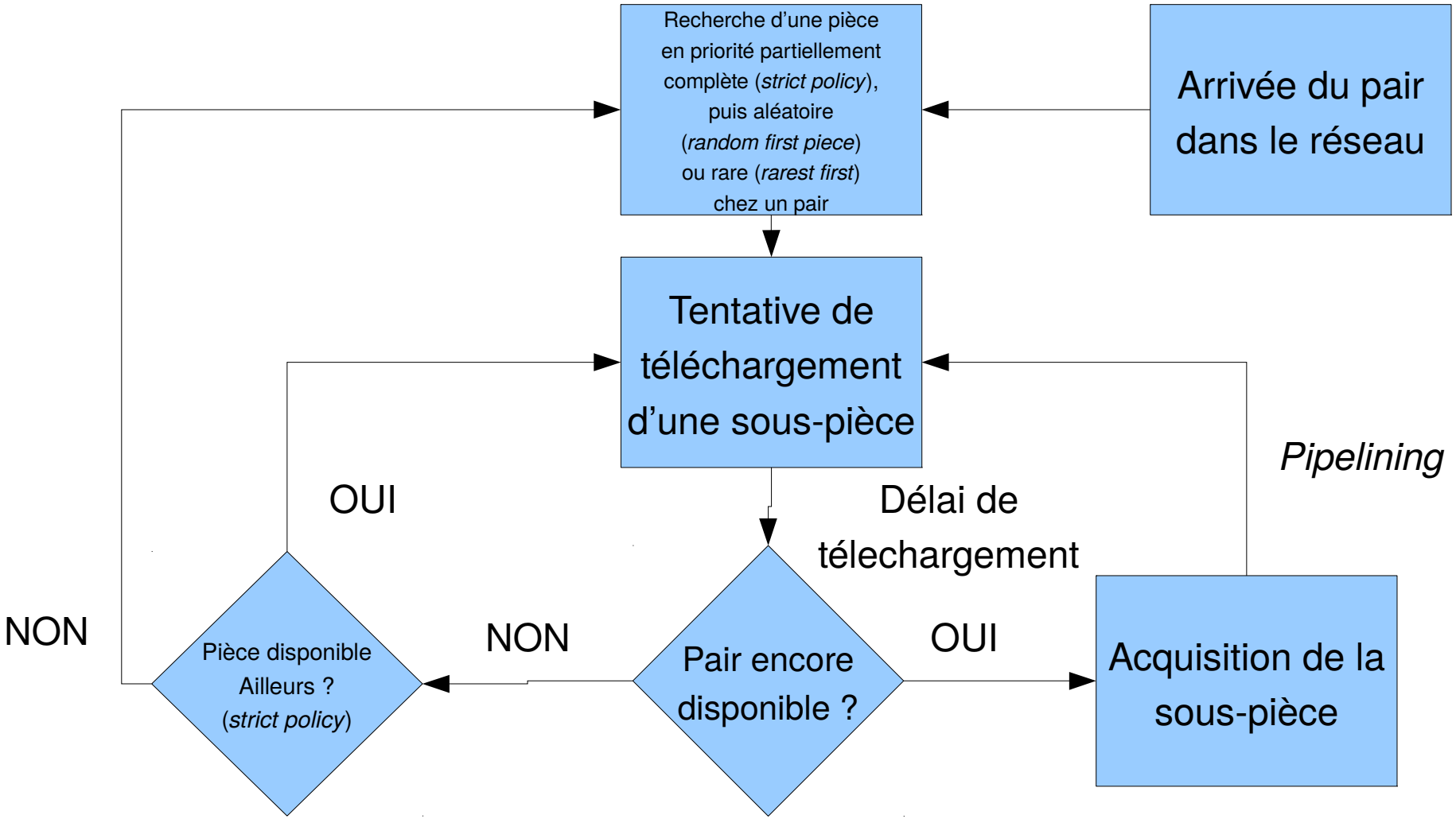
Arrivée des pairs dans le réseau

```
void addPeer(Peer::Network& network,
             std::mt19937& rng,
             Peer::Schedule& schedule,
             Peer::EventData&& edata)
{
  Peer::EventData::Spawner sp = edata.sp;

  network.addPeer(sp.is_filecomplete, sp.upbw, sp.dwbw, sp.nb_unchpeer,
                  sp.nb_optunchpeer, *sp.on_bitfreq, *sp.on_entrance,
                  *sp.on_piececomplete, *sp.on_piececancel,
                  *sp.on_piecefetching, *sp.on_peerfetching, *sp.on_unchoking
                  *sp.on_optunchoking, *sp.on_filecomplete, schedule, rng);
  std::uniform_real_distribution<double> distr;
  schedule.pushEvent(schedule.getDate()
                     - sp.rt_nextspawn * log(1 - distr(rng)),
                     sp, addPeer);
}
```

bittorrent.cpp : lignes 300 à 316

Comportement d'un pair dans le réseau

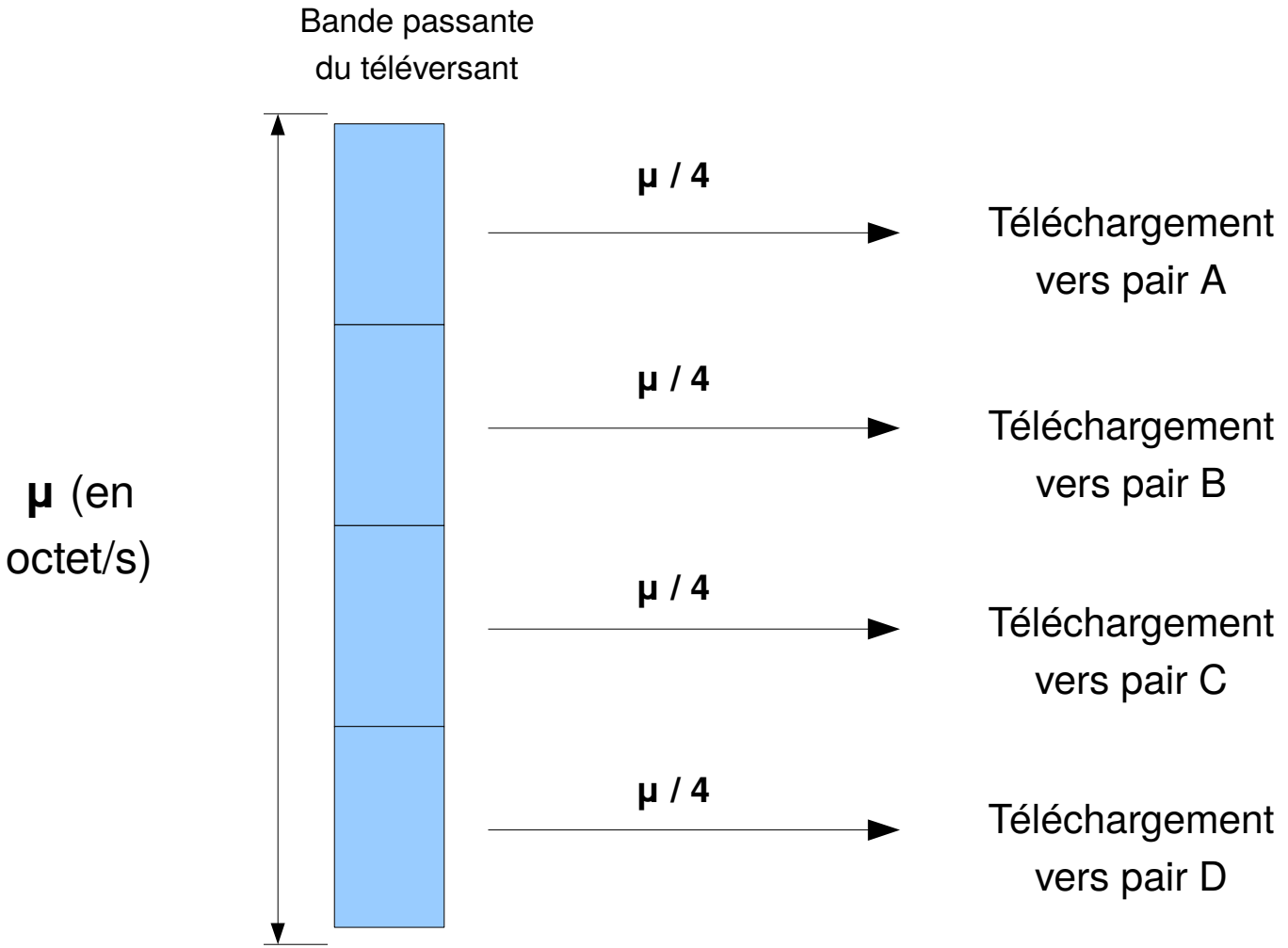


Comportement d'un pair dans le réseau

```
  if(!dwpeer.isPieceComplete(fs.piece)){ //Piece incomplète
    /*Pipelining*/
    createDownload(fs.dwpeer, fs.uppeer, fs.piece, network, schedule);
  }
  else if(dwpeer.isFileComplete()){ //Fichier complet
    /*Téléchargement terminé*/
    uppeer.unfileLeecher(fs.dwpeer);
    dwpeer.onPieceComplete(fs.dwpeer, fs.piece, network, schedule, rng);
    dwpeer.onFileComplete(fs.dwpeer, network, schedule, rng);
  }
  else{ //Pièce complète
    uppeer.unfileLeecher(fs.dwpeer);
    dwpeer.onPieceComplete(fs.dwpeer, fs.piece, network, schedule, rng);
  }
}
else{ //Pair téléversant indisponible
  uppeer.unfileLeecher(fs.dwpeer);
  dwpeer.onPieceCancel(fs.dwpeer, fs.piece, network, schedule, rng);
}
```

bittorrent.cpp : lignes 235 à 253

Modélisation de la bande passante



Modèle simplifié : répartition équitable de la bande passante montante entre les pairs téléchargeant

Comportement des *free riders* (inspiré de *BitThief*)

Profit du réseau sans y contribuer en retour :

- Ne téléverse aucune donnée
- Ne procède donc pas à l'*unchoking* et à l'*optimistic unchoking*
- Ne déclare aucune pièce
- **Conséquence** : Les free riders sont exclus par l'algorithme d'*unchoking,* mais pas d'*optimistic unchoking.*

Requêtes plus fréquentes auprès du serveur centralisant les adresses des pairs (*tracker*) :

- Requêtes d'adresses de pairs plus fréquentes, à intervalle de temps très courts puis croissant de manière exponentielle
- **Conséquence** : Le free rider maintient donc plus de connexions que le pair standard, et profite plus souvent de l'*optimistic unchoking*.

En pratique :

- La $n^{ème}$ recherche de pairs se fait après la durée $\Delta t_{r,f} \times \rho_r^{n-1}$ suivant la dernière recherche (pour la première recherche, elle se fait à la date $t = \Delta t_{r,f}$).
- Cette période continue de croître jusqu'à dépasser Δt_r. Après ce dépassement, la période est constante égale à Δt_r.

Relevés de mesures à partir de la simulation

Paramètres utilisés

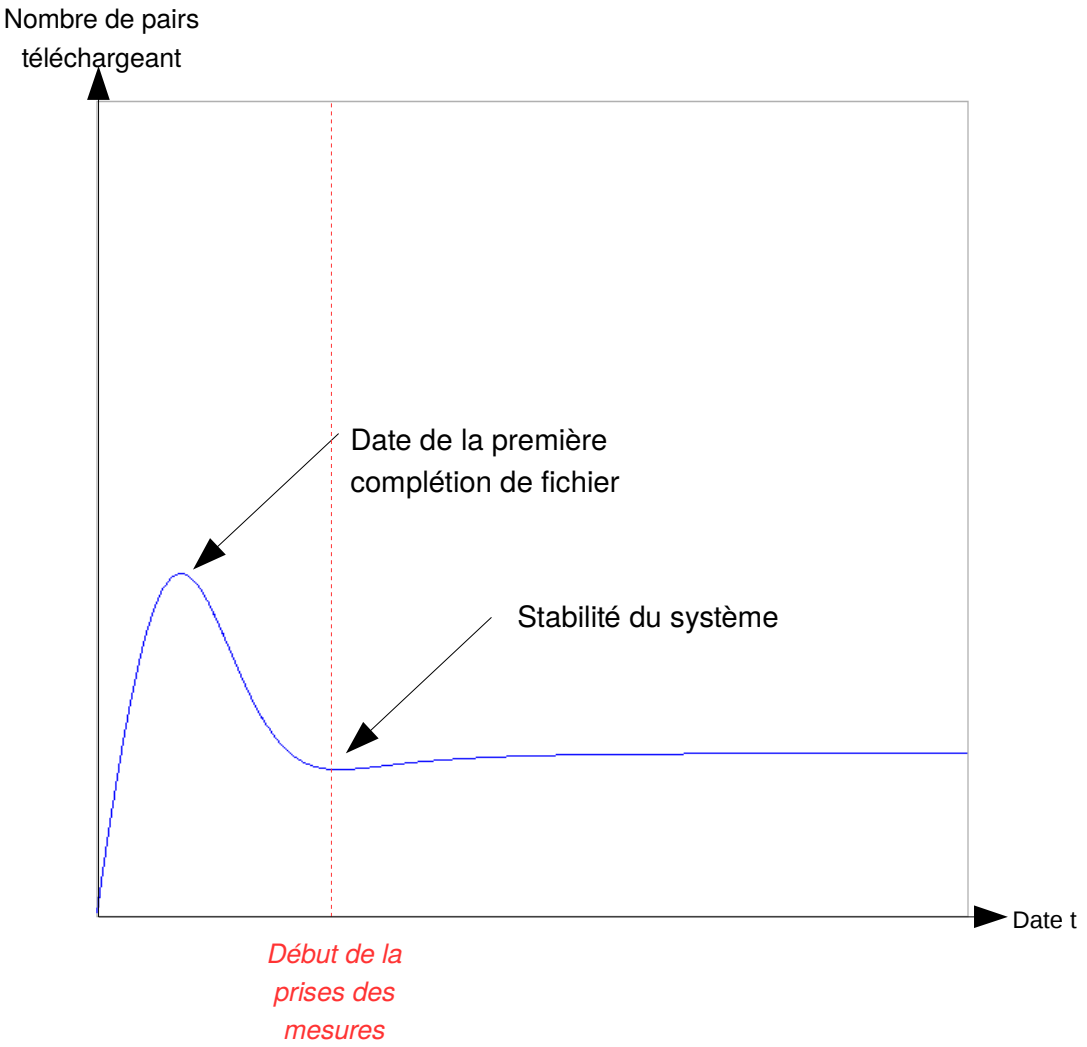
- Nombre de pièces : 1000
- Nombre de sous-pièces par pièce : 16
- Taille d'une sous-pièce : 16 ko
- Bande passante ascendante : μ = 128 ko/s
- Période d'*unchoking* : Δt_{unch} = 10 s
- Période d'*optimistic unchoking* : Δt_{opt} = 30 s
- Nombre de pairs sélectionnés par l'algorithme de *regular unchoking* : n_{unch} = 3
- Nombre de pairs sélectionnés par l'algorithme d'*optimistic unchoking* : n_{opt} = 1
- Nombre de pairs annoncés à chaque requête : n_r = 50
- Intervalle entre chaque recherche de pair : Δt_r = 1800 s
- Intervalle entre chaque recherche de pair (*free rider*) : $\Delta t_{r,f}$ = 30 s
- Coefficient de croissance de l'intervalle : ρ_r = 2.78
- Durée moyenne de partage altruiste entre la complétion du fichier et le départ du pair : Δt_p = 300 s

Etat initial du réseau :

- La simulation débute à t = 0
- Le réseau contient un pair altruiste possèdant un fichier complet et qui reste dans le réseau durant toute la durée de l'expérience

Méthode employée pour la relevé des mesures

On mesure les performances des pairs arrivés après que le système soit devenu stable et ayant quitté le réseau avant la fin de la simulation.



Vérification de la cohérence des résultats fournis par la simulation :
Simulation d'une situation classique

- Seul des pairs coopératifs arrivent dans le réseau.
- Période moyenne d'arrivée des pairs : 30s
- Durée considérée pour les mesures : 10 000s

Ce que l'on souhaite vérifier :

- On cherche à vérifier la loi de Little ($N = \Delta t \times \lambda$) ainsi que la quantité de donnée téléchargée via le *regular unchoking* soit 3 fois plus importante que via l'*optimistic unchoking*.

Résultat :

```
Number of considered leechers : 237
Number of considered free-riders : 0
Leechers average uptime : 4096
Free-rider average uptime : 0
Leechers average download period : 3783
Free-rider average download period : 0
Number of subpiece downloaded by the leechers through regular unchoking : 2780782
Number of subpiece downloaded by the leechers through optimistic unchoking : 1011218
```

- 2.78e6 / 1.01e6 = 2.75

  Le quotient est plus petit : en pratique, il arrive que le pair n'a pas au moins trois voisins "intéressants". La valeur est donc cohérente.

- 3783 / 30.00 = 126.1

```
[ t = 20099.544182 np = 140 nl = 125 ns = 15 nfr = 0] - Peer 594 is downloading piece 553 from peer 596
[ t = 20099.544182 np = 140 nl = 125 ns = 15 nfr = 0] - Peer 594 is downloading piece 608 from peer 568
[ t = 20099.547407 np = 140 nl = 125 ns = 15 nfr = 0] - Peer 575 is downloading piece 648 from peer 587
[ t = 20099.547407 np = 140 nl = 125 ns = 15 nfr = 0] - Peer 575 is downloading piece 649 from peer 622
[ t = 20099.550157 np = 140 nl = 125 ns = 15 nfr = 0] - Peer 566 is downloading piece 712 from peer 576
[ t = 20099.553666 np = 140 nl = 125 ns = 15 nfr = 0] - Peer 616 is downloading piece 413 from peer 581
```

On vérifie la loi de Little.

Vérification de la cohérence des résultats fournis par la simulation :
Les 25% de Qiu et Srikant

- Des pairs coopératifs et des *free riders* arrivent dans le réseaux.
- Les *free riders* se contentent de télécharger sans téléverser en retour, mais ne cherchent pas à réccupérer plus de pairs.
- Période moyenne d'arrivée des pairs : 30s
- Période moyenne d'arrivée des *free riders* : 3 000s
- Durée considérée pour les mesures : 30 000s

Ce que l'on souhaite vérifier :

- La prédiction de Qiu et Srikant selon laquelle un pair ne participant pas et ne déployant aucune stratégie supplémentaire bénéficierait de 25% du débit dont bénéficie un pair coopératif.

Résultat :

```
Number of considered leechers : 847
Number of considered free-riders : 6
Leechers average uptime : 4163
Free-rider average uptime : 12164
Leechers average download period : 3847
Free-rider average download period : 12164
Number of subpiece downloaded by the leechers through regular unchoking : 10024993
Number of subpiece downloaded by the leechers through optimistic unchoking : 3527007
Number of subpiece downloaded by the leechers from seeds : 111165
Number of subpiece downloaded by the free-riders through regular unchoking : 20863
Number of subpiece downloaded by the free-riders through optimistic unchoking : 75137
Number of subpiece downloaded by the free-riders from seeds : 2057
```

- 3847 / 12164 = 0.316

Le quotient est plus grand : les *free-riders* profitent également du partage du fichier par les pairs altruistes.

En prenant en compte les pairs altruistes, on reprend le raisonnement derrière la formule de Q-S et on obtient :

$$\frac{n_c}{n_c - n_u}\frac{1}{n_u + 1} + \frac{n_s}{n_c}$$

En utilisant cette formule ainsi que la loi de Little, on obtient 0.33. On vérifie donc la prédiction de Q-S de manière plus précise.

Conclusion : le programme développé fourni des résultats déjà prouvés, en particulier lorsqu'il s'agit de *free-riding*. Il est donc raisonnable de l'utiliser pour effectuer d'autres mesures.

Mesure de la robustesse du protocole *BitTorrent* face au *free-riding* : Paramètres

- Des pairs coopératifs et des *free riders* arrivent dans le réseau.
- Les *free riders* suivent complétement la stratégie décrite précedement : téléchargement seulement + maitient plus de connexions
- On réalise une série de simulations, en faisant varier la proportion de *free riders* parmis le nombre de pairs arrivants, notée p :

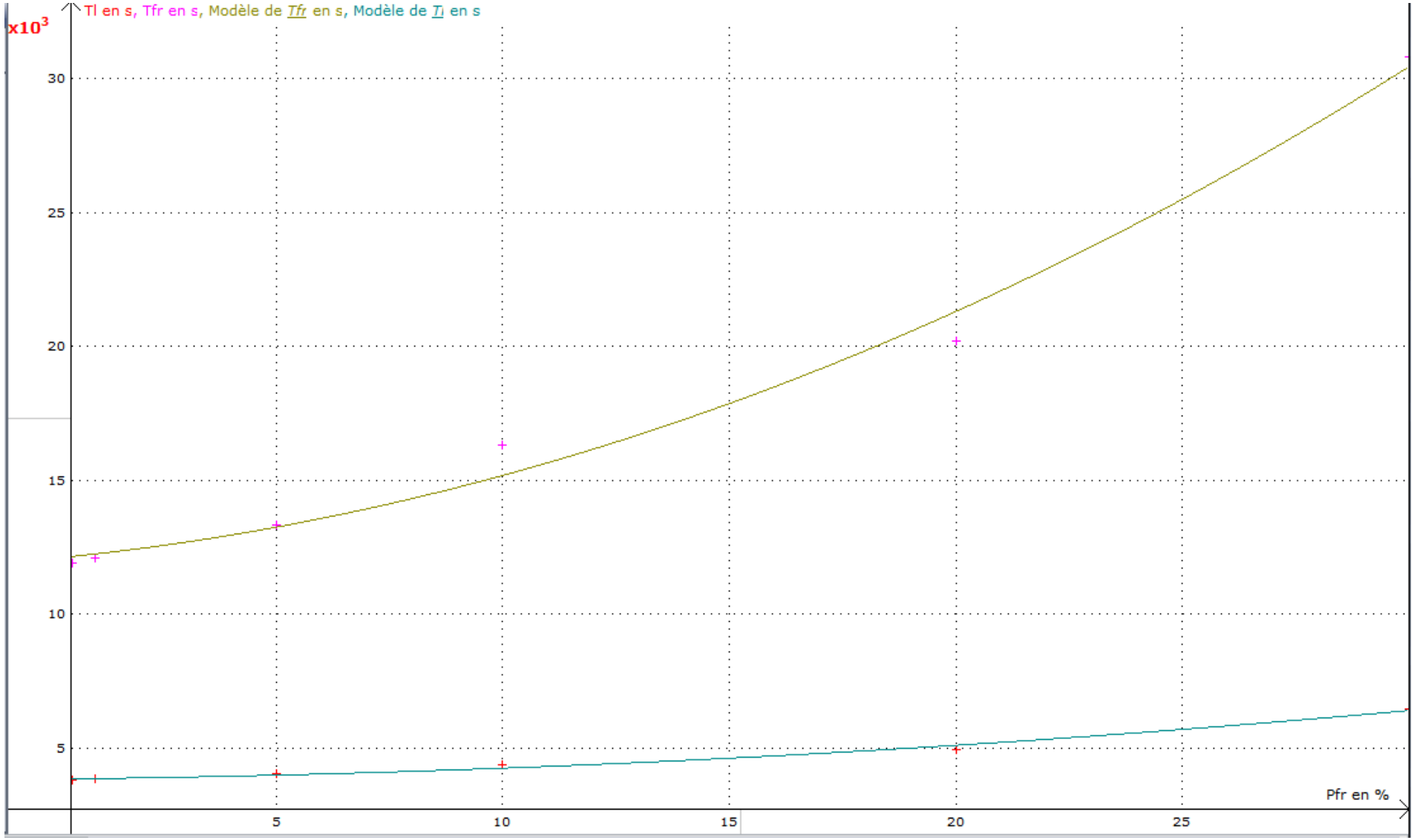
  p = fréquence d'arrivée *free riders* / (fréquence d'arrivée *free riders* + fréquence d'arrivée pairs coopératifs)

- On fixe la période d'arrivée des pairs (coopératifs + *free rider*) à 30s.

Ce qu'on cherche à obtenir :

- Une relation entre la proportion de *free riders* parmis les pairs entrants et la durée moyenne nécessaire au téléchargement du fichier complet pour les pairs coopératifs et les *free riders*.

Mesure de la robustesse du protocole *BitTorrent* face au *free-riding* : Résultat



Mesure de la robustesse du protocole *BitTorrent* face au *free-riding* : Résultat

$$T = ax^2 + bx + c$$

Type de pair	a	b	c
Coopératif	2.219	18.91	3.833e3
Free-rider	15.09	160.1	12.071e3

Mesure de la robustesse du protocole *BitTorrent* face au *free-riding* : Conclusions

- **Le temps de téléchargement est quadratique selon la proportion de *free riders*** : intérêt de mettre en place des mécanismes contre le *free-riding* afin de limiter sa pratique
- **Le temps de téléchargement croît moins vite pour les pairs coopératifs** : motive les *free riders* à arrêter et à coopérer, montre une certaine robustesse face au *free-riding*
- **Les performances des *free riders* sont conformes aux performances réelles de *BitThief* :** "Cela prend environs quatre fois plus de temps avec notre client" (extrait de l'article traitant des performances de *BitThief*, au sujet d'une situation similaire à la simulation) La simulation fournie donc encore une fois des résultats crédibles.
- **Le maintient de plus de connexions ne semble pas améliorer les performances des *free riders* :** semble se réveler plus efficace avec des réseaux plus gros (donc plus lent à simuler, d'où une difficulté technique à montrer le potentiel de *BitThief*)