POPS : service de diffusion de flux vidéos live prenant en compte la popularité

Karine Pires, Sébastien Monnet, Pierre Sens
Sorbonne Universités, UPMC Univ Paris 06, Équipe REGAL, LIP6, F-75005, Paris, France
CNRS, UMR_7606, LIP6, F-75005, Paris, France
Inria, Équipe REGAL, F-75005, Paris, France
Email : prénom.nom@lip6.fr

Résumé
La diffusion en direct (*live*) de flux vidéos est devenue très populaire. De nombreux systèmes, comme justin.tv ou YouTube, collectent des flux vidéos *live* et les diffusent à des utilisateurs *spectateurs* en utilisant un ensemble de serveurs. Cependant, le nombre de spectateurs varie considérablement dans le temps et la distribution de la popularité parmi les flux est extrêmement hétérogène. Cela amène les fournisseurs de service à utiliser des plateformes surdimensionnées afin d'absorber les pics de spectateurs. Dans ce papier, en nous appuyant sur des traces de justin.tv et de YouTube, nous étudions le compromis entre le nombre de serveurs provisionnés et l'utilisation de la bande passante entre ces serveurs. Nous mettons également en évidence l'importance de pouvoir prédire la popularité des flux afin de les placer efficacement sur les serveurs. Nous proposons POPS : un service de diffusion en direct de flux videos se basant sur des prédictions de popularité.

1. Introduction

Nous assistons actuellement à l'émergence de deux phénomènes : d'une part, la popularisation des périphériques de capture video et d'autre part l'explosion de la qualité des réseaux et du nombre d'utilisateurs d'Internet. En effet, de nos jours, le moindre ordinateur portable, *netbook* ou même téléphone cellulaire est équipé d'une caméra. Les périphériques "*full HD*" sont devenus très abordables, y compris dans leur version portable. Il s'en suit qu'une large portion de la population possède le matériel nécessaire à la création de flux vidéo. De plus ces équipements sont de plus en plus disponibles car transportables, à l'image des *Google Glass* ou des caméras embarquées pour le sport. Même les drones avec caméra embarquée sont en phase de devenir accessibles au grand public.
En ce qui concerne les réseaux, nous avons assisté ces dernières années à une arrivée en masse des internautes. En parallèle la qualité des réseaux, en termes de latence et débit, s'est nettement améliorée, et continue à l'être avec notamment l'arrivée du très haut débit et de la fibre optique. De surcroît, les tarifs des connexions sont devenus très raisonnables, et la grande majorité des internautes bénéficie d'un accès illimité. La couverture des réseaux mobiles est également très étendue, et de bonne qualité. La vitesse d'accroissement de couverture du haut débit mobile (3G/4G) permet de croire que l'immense majorité de la population aura accès quasi-permanent à un réseau de bonne qualité à un tarif raisonnable.
Ces deux phénomènes combinés, et la propension des utilisateurs à exposer des parties de leur

vie ont mené à la montée en puissance d'un nouveau type de système : les systèmes de diffusion en direct de flux videos. Ces systèmes, comme justin.tv [1] ou même YouTube [1] [3] proposent aux utilisateurs de diffuser ou de regarder des flux vidéos en direct. La quantité de données à traiter est considérable. Par exemple, chaque minute, la plateforme de justin.tv doit absorber plus de 30 heures de flux vidéo live [10] créés par des milliers de contributeurs. Les flux doivent être diffusés à des centaines de milliers de spectateurs. Nous pensons que cet engouement pour les flux vidéos générés par les utilisateurs en direct va aller en croissant, avec l'apparition du *journalism citoyen* [6], le développement des flux vidéos générés par les joueurs [2, 15] ou la télévision de demain.

La difficulté du traitement de ces flux vient de l'échelle en nombre de flux et de spectateurs, et de la grande variation de popularité, à la fois dans le temps, et parmi les flux. Le problème de la diffusion *en direct* de flux est différent de la vidéo à la demande : il n'est ici pas possible de répliquer *à l'avance* la vidéo sur de multiples serveurs. Il se différencie également de la télévision sur IP : par rapport aux chaînes de télévision, le nombre de flux est bien plus élevé et le nombre d'utilisateurs par flux est extrêmement variable, certains flux ayant notamment très peu de spectateurs.

Il est nécessaire d'adapter dynamiquement le nombre de serveurs de diffusion de flux de manière à répondre à la demande. Lorsqu'un flux devient très populaire, il est possible que le serveur qui le diffuse atteigne sa capacité maximale, dans quel cas, il doit partager la charge avec un autre serveur, induisant un surcoût en terme de bande passante entre les serveurs. Afin de limiter ce surcoût, il faut placer intelligemment les flux sur les serveurs. Dans cet article, nous proposons le service de diffusion POPS pour l'anglais *POPularity-aware live-Stream streaming service*. Notre approche s'appuie sur une étude de traces que nous avons collectées sur les plateformes justin.tv et YouTube qui nous permet de prédire la popularité d'un flux. POPS s'appuie sur ces prédictions afin de provisionner dynamiquement les serveurs de diffusion. Notre évaluation montre que lorsque l'historique permet d'estimer la popularité d'un flux, POPS peut diminuer la quantité de bande passante utilisée entre les serveurs sans utiliser un trop grand nombre de serveurs.

La suite de ce papier est organisée comme suit. La section 2 présente le contexte de cette étude ainsi que les travaux liés. La section 3 présente notre contribution : une stratégie de placement utilisant des prédictions de popularité des flux vidéos en direct. Enfin, la section 4 détaille notre évaluation avant que la section 5 conclut.

2. Modèle et positionnement

Cette section précise le contexte de notre étude et présente les travaux les plus proches.

2.1. Modèle

Nous nous concentrons sur les plateformes de diffusion en direct de flux vidéo utilisateurs. Les utilisateurs de ces services peuvent jouer deux rôles. Certains créent des *flux* vidéos à diffuser, on les appelle les *uploaders*. Certains visionnent les flux en direct, se sont les *spectateurs*. A un instant donné, un utilisateur peut jouer un des deux rôles, ou les deux simultanément. Dans notre étude nous nous concentrons sur chacun des rôles indépendamment. Il est également possible qu'un utilisateur soit spectateur pour plusieurs flux simultanément, même si cela ne représente pas l'usage classique.

1. Youtube est connu pour sa partie non-live, mais il propose maintenant également une partie live.

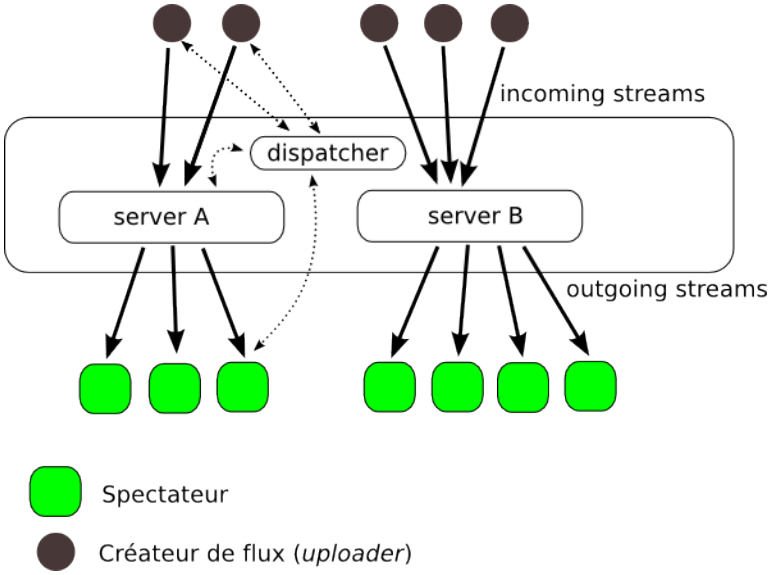


FIGURE 1 – Aperçu du fonctionnement de la diffusion de flux en direct.

Dans notre modèle, illustré par la figure 1, lorsqu'un *uploader* crée un flux, il l'envoie à la plateforme. La plateforme est constituée de serveurs dont le nombre peut évoluer à la demande (e.g., des machines virtuelles dans un *cloud* réservée en fonction de la charge). Nous ne considérons pas de limites sur le nombre de serveurs maximal utilisable par la plateforme (nous considérons qu'il y a suffisamment de ressources disponibles). Nous ne considérons pas non plus le temps d'allocation d'un nouveau serveur. Chaque flux se voit attribuer un serveur en charge de le diffuser aux spectateurs intéressés. Un serveur peut se voir attribuer plusieurs flux. Il est également possible qu'un flux soit pris en charge par plus d'un serveur, dans le cas où sa popularité (nombre de spectateurs intéressés) viendrait à excéder sa capacité de service. Nous considérons qu'il existe un service centralisé sur un nœud *dispatcher* qui reçoit les requêtes :

1. de création de nouveau flux, de la part d'un *uploader* ;
2. d'abonnement à un flux, de la part d'un spectateur.

Ce service est en charge d'affecter un serveur pour diffuser un nouveau flux, soit en en choisissant un diffusant déjà d'autres flux, soit en en réservant un nouveau. L'utilisateur *uploader* enverra alors son flux directement au serveur choisit. Lors de la réception d'une requête de spectateur, ce service identifie le serveur en charge de la diffusion du flux recherché et le retourne à l'utilisateur. Le serveur enverra par la suite directement le flux au spectateur.
Une architecture plus évoluée, distribuée de ce service sort du cadre de ce papier. Nous nous concentrons ici sur l'étude des choix de placement que peut faire un tel service et les conséquences qui en découlent. Pour des raisons de simplification et de clarté, nous faisons également l'hypothèse que tous les flux vidéos requièrent la même bande passante. Un serveur peut donc diffuser un flux vidéo à un nombre fixé de spectateurs ($\frac{\text{bande passante sortante disponible}}{\text{bande passante pour un flux}}$).
Les ordres de grandeur sont importants. Le nombre d'utilisateurs que ces plateformes doivent pouvoir gérer à un instant donné est de l'ordre de plusieurs centaines de milliers. A la fin de la période d'analyse plus de 350000 uploaders différents ont été observés sur justin.tv et plus de 30000 sur YouTube. Il y a à la fois de nombreux flux (des milliers) et de nombreux spectateurs (des centaines de milliers répartis sur les différents flux). Le tableau 1 résume ces valeurs. La répartition des spectateurs par flux étant très hétérogène : de quelques unités à des milliers, et même des centaines de milliers pour les quelques flux les plus populaires. La figure 2 montre les valeurs à chaque instant sur les plateformes durant la période du 6 Janvier 2014 au 21 Janvier

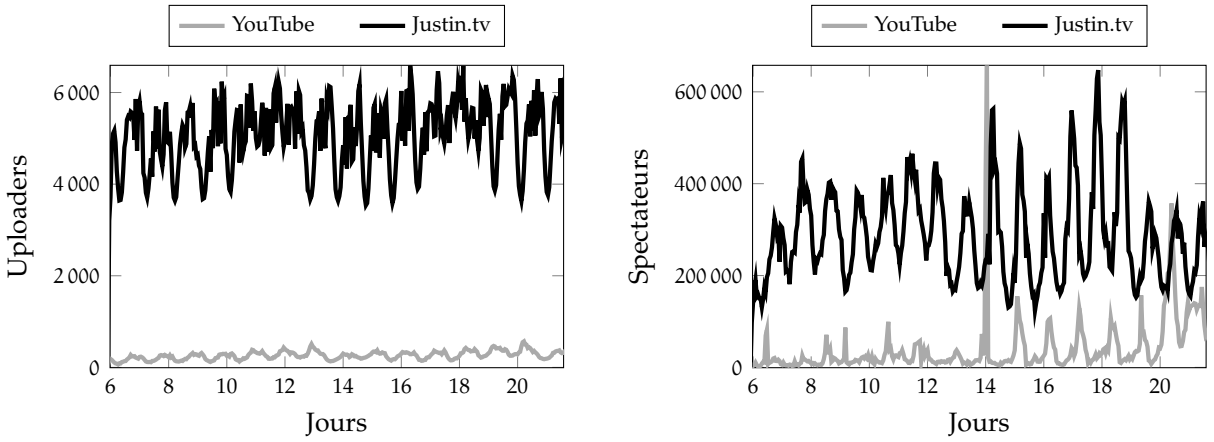


FIGURE 2 – Nombre d'utilisateurs sur les plateformes, uploaders à gauche et spectateurs à droite

2014.

	justin.tv	**YouTube**
nombre total d'uploaders	354 762	35 833
nombre total de flux	665 979	67 348
pic d'uploaders	6 631	653
pic des spectateurs	647 299	657 079

TABLE 1 – Résumé des valeurs observées sur justin.tv et YouTube

2.2. Positionnement

L'émergence des plateformes de diffusion en direct de flux vidéos générés par les utilisateurs est relativement récente. Il est difficile de savoir comment fonctionnent précisément les systèmes commerciaux actuels. Nous présentons ici les travaux connexes. Nous avons identifié quatre principales catégories.

Mesures sur les contenus vidéos. De nombreuses études se concentrent sur l'analyse du traffic. Les flux vidéos occupent en effet une part grandissante de la bande passante sur Internet [11]. Les auteurs de [8] ont étudié le traffic sur YouTube généré à partir de périphériques mobiles et l'ont comparé à celui généré par les PC, ils ont montré que les schémas d'accès restaient comparables. Dans [19], les besoins en bande passante et en espace de stockage de YouTube sont étudiés, tout comme dans [4] où ils le sont du point de vue d'un fournisseur d'accès à Internet. L'ensemble de ces études montrent l'importance croissante des flux vidéos sur Internet, et la nécessité de concevoir des plateformes passant à l'échelle afin de traiter ces données. Cependant, nous n'avons pas trouvé de traces à grande échelle de systèmes de diffusion en direct de flux vidéos générés par les utilisateurs librement disponibles. C'est pourquoi nous avons du mener notre propre campagne de collecte sur les systèmes justin.tv et YouTube (voir la section 3).

Systèmes pair-à-pair pour la diffusion en direct de flux videos générés par des utilisateurs. Nous nous concentrons sur une solution à base de multiples serveurs. Nous étudions comment répartir les flux vidéos sur les serveurs. D'autre études adoptent une approche complètement décentralisée. Un état de l'art détaillé est donné par [18]. Cependant, la majeure partie des travaux étudiant cette approche se concentre sur la décentralisation et la topologie du réseau logique pair-à-pair sous-jacent [9, 16]. Il nous semble important d'étudier une approche basée sur un ensemble de serveurs, car c'est en général celle

choisie par les systèmes commerciaux. C'est également celle qui fonctionnent réellement à grande échelle en étant capable d'attirer un grand nombre d'utilisateurs. Il existe aussi des solutions utilisant des réseaux de type CDN (*Content Delivery Networks*) [17, 13], taillées pour des flux extrêmement populaires. Ce type de solution nous semble plus adapté, à un nombre restreint de flux vidéos extrêmement populaires, à l'image de ce qui est observable dans le cadre de la télévision sur IP.

Télévision sur IP. [7] propose une analyse d'un des plus grands fournisseur de flux vidéo en direct d'Europe : Zattoo. Dans ce papier, les auteurs ont des informations sur l'architecture de Zattoo d'un point de vue fournisseur. Cependant, Zattoo n'est pas vraiment un système à grande échelle : les pics de charge observés sont un ordre de grandeur en dessous de ceux que nous avons pu voir lors de notre collecte. D'autres études comme [14] étudient la charge des systèmes de télévision sur IP. Cependant, ces systèmes présentent moins de chaînes qu'un système proposant du contenu généré par des utilisateurs. Les spectateurs des systèmes que nous ciblons sont répartis sur un très grand nombre de flux vidéos.

Systèmes de distribution de vidéo à la demande. Enfin, il existe des travaux de recherche sur le placement des données pour les systèmes de vidéo à la demande (VoD pour *Video on Demand*) [5, 12], cependant, ces travaux sont finalement assez éloignés de nos préoccupation. Le fait que nous adressons des flux vidéo *en direct* n'est pas anodin. Il n'est pas possible de placer à l'avance du contenu sur des serveurs. Le nombre de spectateurs regardant le même flux, au même *offset* est très grand, contrairement à ce que l'on peut observer dans les systèmes de VoD.

3. Placement des flux vidéo live sur les serveurs de diffusion

Cette section décrit notre contribution. Nous commençons par présenter brièvement les leçons que nous avons apprises grâce à la collecte de traces utilisateurs des systèmes justin.tv [1] et YouTube [3]. Nous étudions ensuite différentes stratégies de placement de flux sur les serveurs de diffusion avant de présenter notre stratégie prenant en compte une prédiction de la popularité des flux.

3.1. Etude des traces utilisateurs de justin.tv et YouTube

Justin.tv est une plateforme de diffusion en direct de flux videos générés par des utilisateurs. Cette plateforme est très populaire : sur trois semaines, il y a plus de 600 millions de flux, générés par plus de 350000 *uploaders*. Il y a chaque jours plusieurs centaines de milliers de spectateurs. De plus, justin.tv offre une API permettant de récupérer à intervalles réguliers l'ensemble des informations qui nous intéressent : nombre de flux, avec pour chaque flux l'identifiant de l'*uploader* et le nombre de spectateurs courant, etc. L'ensemble des données est disponible à la demande.

L'analyse des données obtenues sort du cadre de ce papier. Les principaux enseignements que nous en avons tirés sont :

1. la popularité parmi les flux est très hétérogène, les plus populaires générant l'essentiel du trafic ;
2. la charge globale du système est très variable dans le temps ;
3. il y a en permanence un très grand nombre de flux, des ordres de grandeur en dessus du nombre de chaînes d'une télévision câblée traditionnelle ;
4. le comportement des utilisateurs est assez prédictible : il est possible d'observer une oscillation jour/nuit, mais surtout, que la popularité d'un flux est très fortement liée à

l'*uploader* qui la diffuse et qu'elle a tendance à croitre avec la durée du flux. C'est-à-dire que la popularité parmi les *uploaders* est très hétérogène, mais que les différents flux vidéos d'un même *uploader* auront une popularité assez semblable. Les *uploaders* populaires ayant un ensemble de "fans" qui les suivent.

Lors de la conception de services devant servir un très grand nombre d'utilisateurs il est important de prendre en compte la topologie de la charge de travail. L'utilisation de données réelles permet la mise en œuvre de mécanismes adaptés.

3.2. Compromis nombre de serveurs/surcoût en bande passante

Dans notre modèle, un serveur ne peut servir qu'un nombre limité, fixe, de spectateurs, noté MAX_SERV. Une approche simple et économique, en terme de nombre de serveurs utilisés, consiste à attribuer des flux à un serveur tant que celui-ci n'est pas complètement chargé. Cependant, la popularité des flux varie au cours de leur vie. Aussi, si un serveur déjà chargé voit un (ou plusieurs) des flux qu'il sert acquérir de nouveaux spectateurs, il risque de dépasser MAX_SERV spectateurs à servir. Si l'on souhaite pouvoir continuer la diffusion de l'ensemble des flux de manière convenable à l'ensemble des spectateurs, il est alors nécessaire d'allouer un nouveau serveur, ou d'en utiliser un servant déjà d'autres flux mais n'ayant pas atteint sa capacité maximale. Le serveur d'origine, continue à diffuser le flux au spectateurs qu'il servait, il envoie également ce flux au nouveau serveur afin qu'il puisse le diffuser aux nouveaux clients, ce mécanisme de "réplication de flux" est illustré par la figure 3. Ce mécanisme, si il permet de servir tous les spectateurs, à un coût en terme de bande passante. Chaque flux nécessitant la même bande passante dans notre modèle, il est possible de mesurer ce surcoût en $nombre\ de\ flux \times heure$. Un flux répliqué pendant une heure d'un serveur A à un serveur B compte pour $1 flux.h$. Nous ne prenons pas en compte les flux envoyés aux spectateurs : il y en a un par spectateur, ce qui représente un traffic bien plus important. Nous nous concentrons ici sur les flux inter-serveurs sur lesquels le placement à un impact important.

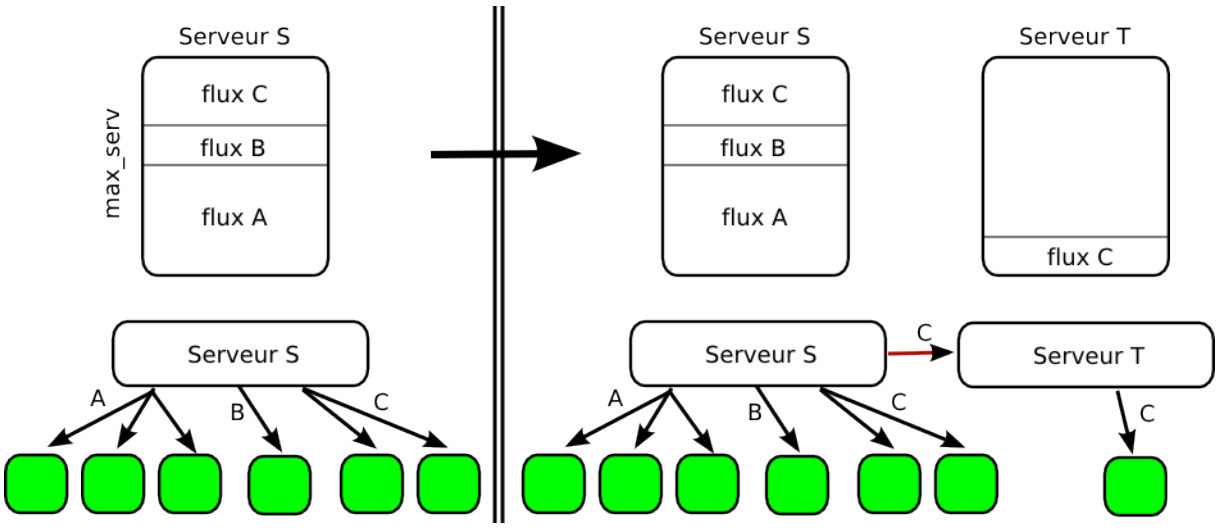


FIGURE 3 – Le serveur S a atteint sa capacité maximale, l'arrivée d'un nouveau client implique une réplication du flux demandé (C) sur un nouveau serveur (T).

Le nombre de spectateurs pour un flux variant au cours du temps (allant croissant ou décroissant), il est possible que le phénomène décrit ci-dessus mène à des configurations pathologiques

au sein desquelles de nombreux flux sont transférés entre les serveurs. Dans ce travail, nous n'explorons pas la possibilité de *reconfigurer* le positionnement des flux vidéos sur les serveurs. C'est-à-dire qu'une fois qu'un spectateur se voit attribuer par le dispatcher un serveur pour un flux vidéo donné, il recevra le flux de ce serveur uniquement. Une reconfiguration consisterait en déplacer des spectateurs entre les serveurs.
Une solution simple permettant de limiter la nécessité de transférer des flux entre serveurs est de sur-provisionner, prévoir une marge de progression "MARGIN" du nombre de spectateurs pour les flux diffusés par un serveur, comme illustré par la figure 4. Ainsi, si l'attribution d'un nouveau flux à un serveur amènerait à dépasser MAX_SERV − MARGIN il ne sera pas choisi par le dispatcher. Cela permet au serveur de pouvoir accepter au moins MARGIN nouveaux spectateurs pour les flux qu'il sert, diminuant automatiquement la probabilité d'excéder sa capacité MAX_SERV, et donc de devoir transférer un flux à un autre serveur. En revanche, une telle solution nécessite l'allocation d'un plus grand nombre de serveurs. Un serveur peut être libéré lorsqu'il ne sert plus de spectateur. La valeur à laquelle la marge de progression est fixée permet de régler le compromis nombre de serveurs alloués/bande passante inter-serveurs utilisée. D'autres paramètres, comme la stratégie de choix d'un serveur parmi plusieurs déjà utilisés, *best-fit* ou *worstfit*, peuvent impacter le nombre de serveurs alloués et la quantité de bande passante inter-serveur utilisée.

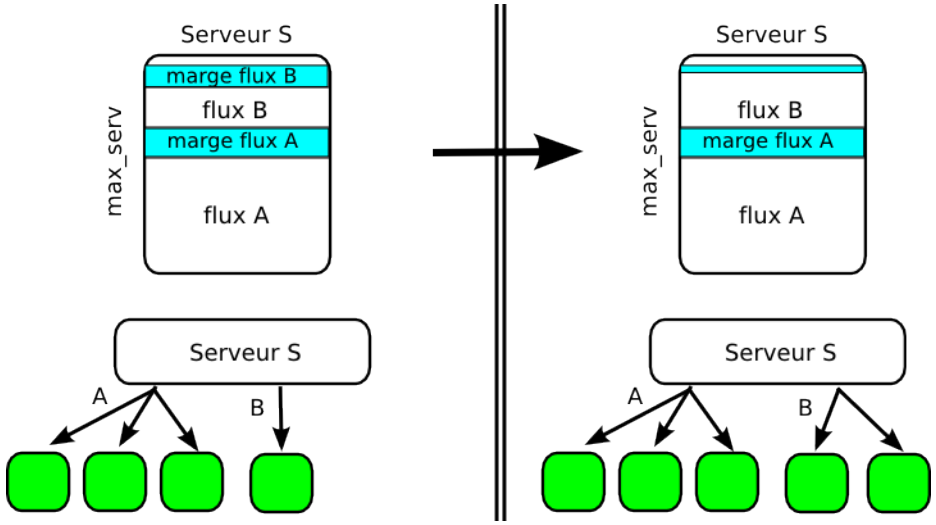


FIGURE 4 – Le serveur a atteint sa capacité maximale (en comptant les marges). Un nouveau flux ne lui sera donc pas confié. A l'arrivée d'un nouveau client pour le flux B, il peut le servir en réduisant la marge provisionnée pour ce flux.

3.3. Prise en compte de la popularité

Notre approche, *POPS*, consiste à prendre en compte une estimation de la popularité à venir d'un flux pour la détermination de la marge. S'il était possible de connaitre à l'avance le nombre de spectateurs qu'un flux vidéo va attirer, il serait possible de provisionner exactement ce dont il aura besoin (c'est la stratégie *Oracle* dans notre évaluation). On réservera par exemple un demi-serveur pour un flux qui aura un pic de $\frac{\text{MAX\_SERV}}{2}$ spectateurs. Notons tout de même que même avec un oracle parfait, cette solution ne représente pas l'optimum. Par exemple, il serait peut-être possible, en dehors de la période de pic de spectateurs, de servir un flux de courte durée en utilisant une portion de serveur provisionnée. Il serait alors également nécessaire de pouvoir

prédire les heures d'arrivées de nouveaux flux et leur durée. Ces considérations sortent du cadre de ce papier.
POPS s'appuie sur la constatation que la popularité des flux vidéos d'un même *uploader* varie peu. Aussi, lorsqu'un *uploader* connu (c'est-à-dire dont au moins un flux vidéo est contenu dans notre historique) diffuse un nouveau flux, POPS provisionne suffisamment de ressources (serveurs) pour servir son pic estimé de spectateurs. Ainsi, si la prédiction était parfaite, chaque serveur devrait pouvoir faire face à un pic concurrent de tous les flux videos qu'il diffuse. La prédiction étant imparfaite, et certains *uploaders* n'ayant pas d'historique, il est également possible d'ajouter une petite marge fixe. Dans nos évaluations l'estimation est q'un flux aura la même popularité que le précédent flux émis par le même *uploader*.

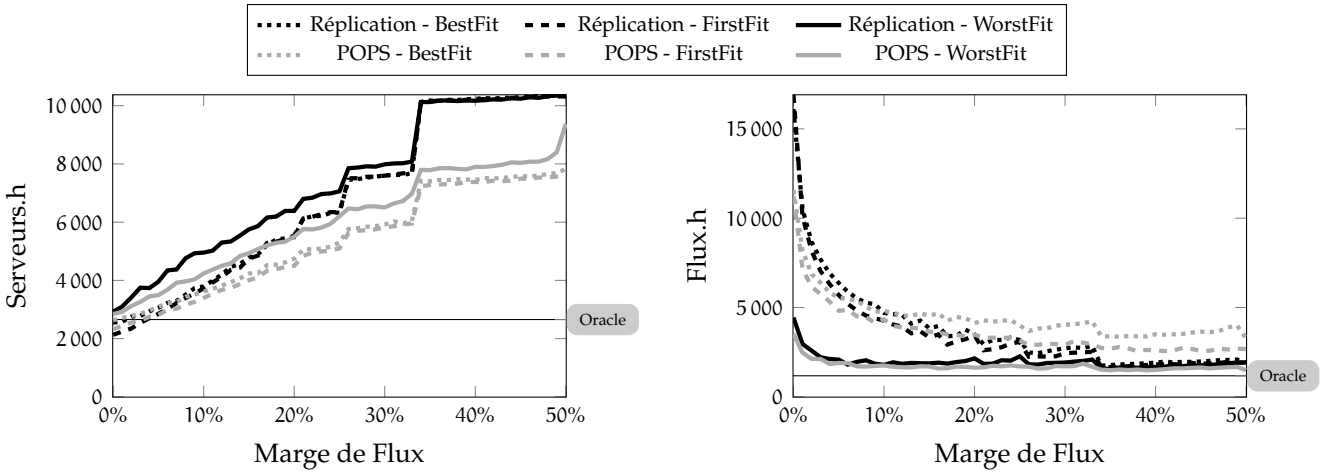
4. Evaluation

Paramètres	Valeurs
traces	justin.tv, YouTube
MAX_SERV	1 000
stratégie de choix	FirstFit, BestFit, WorstFit
approches	Réplication, POPS, Oracle
marge de flux	0%,1%,2%,...,50% de MAX_SERV

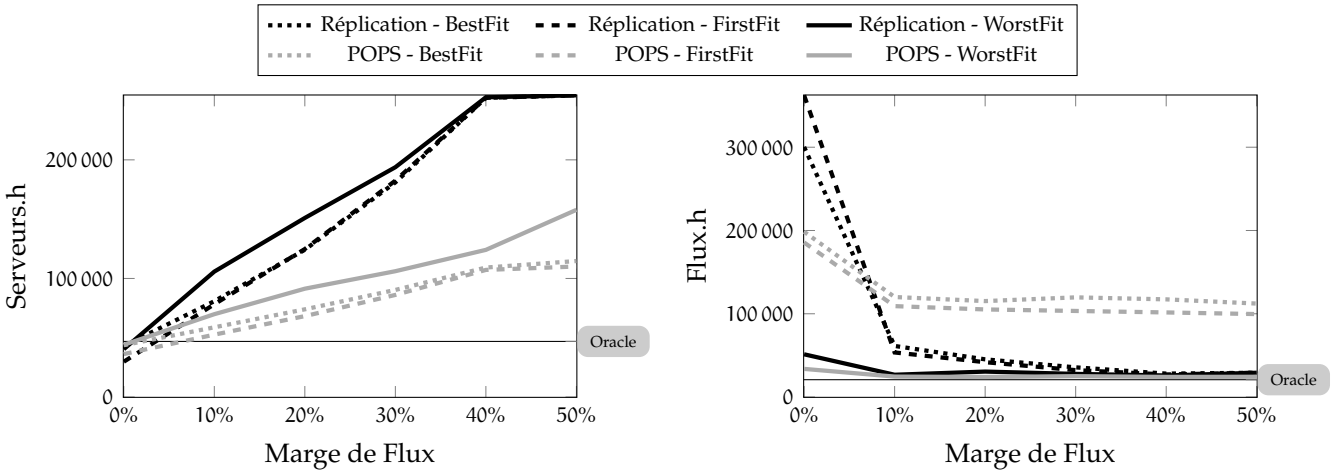
TABLE 2 – Paramètres utilisés sur la simulation

Pour les besoins de cette section nous définissons quelques variables. Le tableau 2 représente les paramètres utilisés dans notre étude. Nous avons utilisé les traces de justin.tv et de YouTube que nous avons collectées. Nous avons fixé arbitrairement la capacité maximale de diffusion de chaque serveur, MAX_SERV, à 1 000 flux vidéos (nous considérons que tous les flux nécessitent la même bande passante). Cette limite correspond à 2Gbps de bande passante sortante pour chaque serveur si l'on considère des flux vidéo de $2\text{Mbp}/s$.
Pour choisir parmi plusieurs serveurs auxquels ils reste de la capacité, nous avons utilisé les trois stratégies bien connues : FirstFit (premier serveur pouvant servir le flux trouvé), BestFit (serveur avec la plus petite capacité restante parmi ceux pouvant servir le flux) et WorstFit (serveur avec la plus grande capacité restante). En ce qui concerne la réservation de serveurs, nous avons évalué 3 approches : *Réplication*, *POPS*, et *Oracle*. *Réplication* consiste en la stratégie simple, où une marge fixe est associée à chaque flux, à l'image de la figure 4. *POPS* est notre approche prenant en compte une estimation de la popularité des flux vidéos en fonction de l'*uploader* qui les diffuse. Enfin, *Oracle* correspond au résultats que nous aurions avec une prédiction parfaite (pour chaque flux, nous provisionnons suffisamment de capacité pour soutenir le plus grand pic de spectateurs). Ceci est rendu possible car nous connaissons à l'avance la trace que nous fournissons à notre simulateur.
Nous faisons varier les marges fixes de 0% à 50%. Les marges fixes sont utilisées dans le cadre de l'approche *Réplication*, mais aussi avec l'approche *POPS* lorsque l'on ne peut pas faire de prédiction. C'est le cas lorsque l'historique d'un *uploader* diffusant un flux est vide. Comme nous démarrons notre simulation avec un historique vide, chaque premier flux d'un *uploader* se voit donc attribuer une marge fixe. Pour les prochains flux du même *uploader*, *POPS* calcule la marge en fonction de la popularité passée de l'*uploader* (celle du dernier flux diffusé par celui-ci). La prise en compte d'un historique plus long, permettant de prendre en considération l'évolution de la popularité d'un *uploader* fait partie de nos travaux en cours. Il est important de noter que nous utilisons une trace d'une durée limitée dans le temps. Plus l'historique est important, plus il est possible d'établir des prévisions fiables car plus il y a d'*uploaders* dont la popularité est

connue. L'approche *Oracle* n'utilise jamais de marge fixe, mais la valeur précise du pic de spectateurs sur le flux vidéo.
Nous calculons le coût en serveur en *serveurs.h* : un serveur utilisé une heure compte pour 1 *serveur.h*.



(a) YouTube

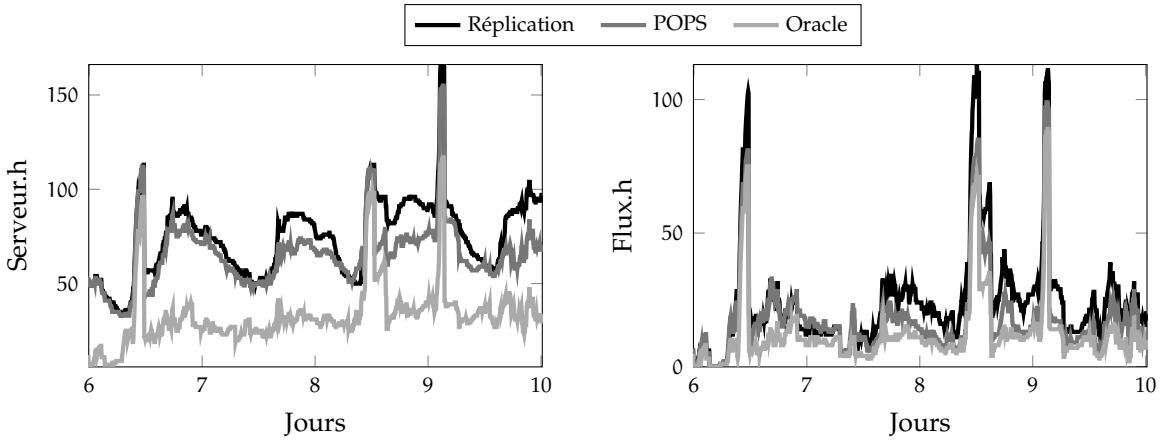


(b) justin.tv

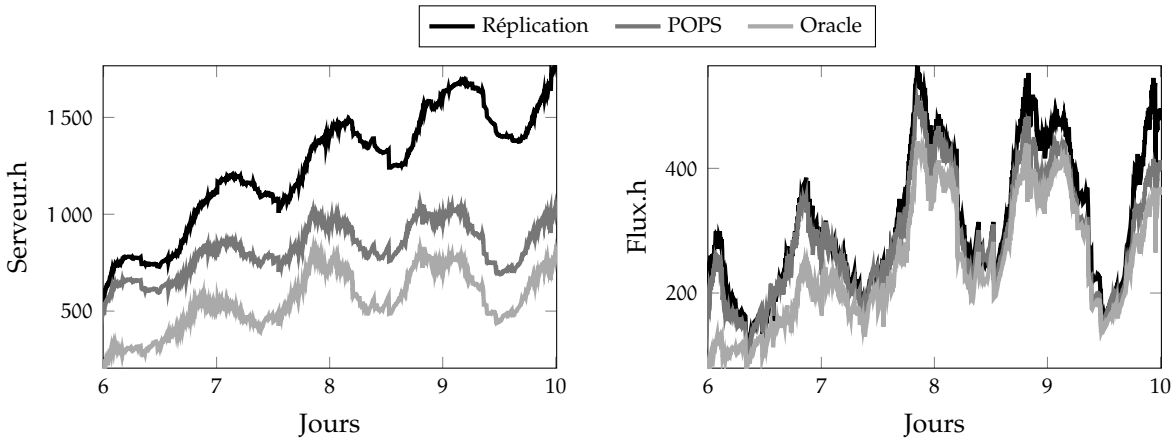
FIGURE 5 – Comparaison des stratégies

Nous comparons toutes les stratégies dans la figure 5. Pour chaque stratégie, nous avons évalué le coût en serveur (*serveurs.h*) et en bande passante (*flux.h*). Une ligne horizontale donne le coût que nous obtenons en appliquant la stratégie *Oracle*.
Le premier enseignement de la figure 5 est que la stratégie de choix de placement (FirstFit, BestFit ou WorstFit) à un impact important sur les résultats. Pour l'approche *Oracle*, nous avons choisi de ne montrer que la meilleure des trois stratégies, la stratégie WorstFit, pour des raisons de lisibilité. La stratégie BestFit, comme c'était prévisible, à le plus petit coût en terme de serveurs, la stratégie FirstFit montre des performances semblables. Cependant, ces deux stratégies mènent à de nombreux transferts de flux inter-serveurs. Nous observons qu'avec ces stratégies, la partition des flux est importante.
Nous observons également que l'augmentation de la marge fixe induit une augmentation pro-

portionnelle de l'utilisation de serveurs pour la diffusion des flux. Par contre, en terme de bande passante, on s'aperçoit qu'à partir de 25% de marge, le gain devient négligeable.
Les différences entre les résultats des deux traces (YouTube en haut en justin.tv en bas) sont importantes. Ce qui souligne l'intérêt d'étudier différentes traces. Nous pouvons observer que les gains apportés par *POPS* en termes de coûts de serveur sont plus importants dans le cas de justin.tv. Ceci est du au fait que les uploaders sur justin.tv ont tendance à produire des flux vidéo plus souvent, l'historique est donc rempli plus vite au sein de cette plateforme. Dans YouTube, le délai inter-emission de flux d'un même *uploader* est plus grand. Une grande partie des flux que nous recevons viennent donc d'un *uploader* non encore connu, pour lequel nous n'avons pas d'historique. En revanche, nous avons détecté, dans le cas de justin.tv, que certains *uploaders* ont une popularité croissante, c'est-à-dire que leurs flux successifs sont de plus en plus populaires. Notre modèle ne prévoit pas cette augmentation de spectateurs, il se contente d'estimer qu'un flux aura la même popularité que le dernier flux émis par le même *uploader*. Cela affecte les résultats de "flux.h" pour les stratégies FirstFit et BestFit sur justin.tv.



(a) YouTube avec stratégie *WorstFit* et marge de 20%



(b) justin.tv avec stratégie *WorstFit* et marge de 20%

FIGURE 6 – L'évolution des stratégies dans le temps

Afin de mieux analyser ces premiers résultats, nous avons étudié l'évolution du système dans le temps pour les trois approches (*Réplication*, *POPS*, et *Oracle*) dans leur meilleure configuration, avec la stratégie de placement WorstFit. La figure 6 présente les résultats obtenus. Comme nous

le pensions, l'efficacité de *POPS* s'améliore dans le temps, au fur et à mesure que l'historique du système se remplit. En effet lorsqu'un *uploader* inconnu du système émet un nouveau flux vidéo, *POPS* ne peut estimer la popularité de ce flux et applique donc la même stratégie que *Réplication*. Pour les flux suivants, *POPS* pourra s'appuyer sur les flux précédents afin d'estimer la marge nécessaire. Dans un système réel, la grande majorité des *uploaders* sont déjà connus, et ceux qui diffusent une vidéo pour la première fois sont rarement populaire. Cependant, nos traces collectées débutent à un moment arbitraire de la vie du système.
Malgré cela, le gain obtenu avec *POPS* est significatif et les résultats de cette approche sont comparables avec l'approche *Oracle* lors des pics de charge sur YouTube et régulièrement sur justin.tv. La divergence entre *POPS* et *Réplication*, concernant les couts des serveurs, est visiblement un effet du nombre d'*uploaders* qui ont déjà émis des flux de par le passé, bien plus important sur justin.tv que YouTube. Sur justin.tv la stratégie *POPS* converge rapidement vers des performances proches de *Oracle*.
Il reste cependant encore de nombreuses améliorations à apporter, par exemple la prise en considération de l'augmentation de la popularité d'un *uploader*, ainsi que la consideration d'un historique du temps de durée de chaque flux pour mieux choisir les regroupements sur un même serveur. Prévoir les instants d'arrivée et les durées des flux pourrait par exemple permettre d'utiliser, pour une courte durée, la bande passante réservée pour un flux populaire dont les spectateurs ne sont pas encore tous présents.

5. Conclusions et perspectives

La qualité et l'accessibilité des réseaux informatiques actuels couplées à la popularisation des périphériques de capture video ont mené à la naissance de services de diffusion en direct de flux vidéos générés par les utilisateurs. Les plateformes qui offrent ce service doivent faire face à un grand nombre de flux vidéos à collecter et à diffuser à un nombre encore plus grand de spectateurs. Dans ce papier, nous montrons qu'il est important de prendre en compte la popularité à venir des flux videos lors de leur affectation à un serveur de diffusion afin d'éviter le partitionnement de flux sur de multiples serveurs, coûteux en terme de bande passante. Nous nous basons sur des données réelles, collectées sur la plateforme justin.tv afin de montrer qu'il existe un compromis bande passante inter-serveurs utilisée/nombre de serveurs alloués ; et proposons une approche utilisant une estimation de la popularité à venir afin de placer les flux vidéos.
Dans ce travail, nous n'avons pas envisagé la possibilité de reconfiguration. POPS s'interdit de déplacer des spectateurs d'un serveur à un autre. Lorsque des pics de spectateurs sont passés, ou que des flux vidéos se terminent, il pourrait être intéressant de migrer des spectateurs, voir des flux, d'un serveur à un autre afin de pouvoir libérer des serveurs. Ceci fait l'objet de nos travaux de recherche en cours.

6. Remerciements

Ces travaux ont été financés par l'Agence nationale française pour la recherche (ANR), dans le cadre du projet MyCloud (ANR-10-SEGI-0009, `http://mycloud.inrialpes.fr/`) ainsi que par le projet FUI ODISEA2.

Bibliographie

1. Justin.tv, `http://www.justin.tv/`, January 2014.
2. Twiitch, `http://www.twitch.tv/`, January 2014.
3. Youtube, `http://www.youtube.com/live/`, January 2014.

4. Adhikari (V. K.), Jain (S.) et Zhang (Z.-L.). – Youtube traffic dynamics and its interplay with a tier-1 isp : an isp perspective. *In : IMC*. – ACM.
5. Applegate (D.), Archer (A.), Gopalakrishnan (V.), Lee (S.) et Ramakrishnan (K. K.). – Optimal content placement for a large-scale vod system. *In : CoNEXT*. – ACM.
6. Bulkley (K.). – The rise of citizen journalism. – The Guardian, Jun. 2012. `http://gu.com/p/386de`.
7. Chang (H.), Jamin (S.) et Wang (W.). – Live streaming performance of the zattoo network. *In : IMC*. – ACM.
8. Finamore (A.), Mellia (M.), Munafò (M. M.), Torres (R.) et Rao (S. G.). – Youtube everywhere : impact of device and infrastructure synergies on user experience. *In : IMC*. – ACM.
9. Ganjam (A.), Rao (S. G.), Sripanidkulchai (K.), Zhan (J.) et Zhang (H.). – On-demand waypoints for live p2p video broadcasting. *Peer-to-Peer Networking and Applications*, vol. 3, n4, 2010, pp. 277–293.
10. Hoff (T.). – Gone fishin' : Justin.tv's live video broadcasting architecture. – High Scalability blog, Nov. 2012. `http://is.gd/5ocNz2`.
11. Ihm (S.) et Pai (V. S.). – Towards understanding modern web traffic. *In : IMC*. – ACM.
12. Jayasundara (C.), Nirmalathas (A.), Wong (E.) et Chan (C. A.). – Energy efficient content distribution for vod services. *In : Optical Fiber Communication Conference*. – Optical Society of America.
13. Liebsch (M.) et Yousaf (F. Z.). – Runtime relocation of cdn serving points - enabler for low costs mobile content delivery. *In : WCNC*. pp. 1464–1469. – IEEE.
14. Qiu (T.), Ge (Z.), Lee (S.), Wang (J.), Xu (J. J.) et Zhao (Q.). – Modeling user activities in a large iptv system. *In : IMC*. – ACM.
15. Shen (S.) et Iosup (A.). – XFire online meta-gaming network : Observation and high-level analysis. *In : MMVE Workshop*.
16. Vieira (A. B.), Gomes (P.), Nacif (J. A. M.), Mantini (R.), Almeida (J. M.) et Campos (S. V. A.). – Characterizing sopcast client behavior. *Computer Communications*, vol. 35, n8, 2012, pp. 1004–1016.
17. Yin (H.), Liu (X.), Zhan (T.), Sekar (V.), Qiu (F.), Lin (C.), Zhang (H.) et Li (B.). – Design and deployment of a hybrid cdn-p2p system for live video streaming : experiences with livesky. *In : ACM Multimedia*. – ACM.
18. Zhang (X.) et Hassanein (H. S.). – A survey of peer-to-peer live video streaming schemes - an algorithmic perspective. *Computer Networks*, vol. 56, n15, 2012, pp. 3548–3579.
19. Zhou (J.), Li (Y.), Adhikari (V. K.) et Zhang (Z.-L.). – Counting youtube videos via random prefix sampling. *In : IMC*. – ACM.