Self-Protect une approche orientée flot et centrée sur l'utilisateur pour la QoS des liens d'accès

Dans les chapitres précédents, nous avons considéré des environnements de réseau caractérisés par un grand nombre d'utilisateurs, et une bande passante supérieure de plusieurs ordres de grandeur au débit des flots. Nous y avons proposé et étudié la proposition Cross-Protect, dont les mécanismes d'ordonnancement équitable par flot et de contrôle d'admission pourraient être également appliqués au contexte des *datacenters*.

Le réseau d'accès cependant ne possède pas les mêmes caractéristiques, et nous verrons qu'une solution seulement basée sur du *fair queueing* n'est pas suffisante pour garantir la bonne performance des flots. Il est pourtant nécessaire si l'on souhaite protéger les flots utilisateur de bout en bout de considérer la performance du réseau d'accès, qui est généralement un goulot d'étranglement pour le trafic.

Dans ce chapitre, nous envisageons une déclinaison de l'architecture *Flow-Aware Networking* pour le réseau d'accès, que nous nommons Self-Protect, permettant la gestion du trafic au niveau flot, dans les sens montant et descendant. L'utilisation d'un ordonnanceur de paquets approprié nous permet une réponse aux problèmes récurrents de *bufferbloat*, tout en considérant la performance des flots élastiques requérant de larges buffers. Cette solution présente l'avantage de reposer sur la collaboration de l'utilisateur, et reste donc une approche neutre vis à vis des réseaux

Après avoir proposé un ensemble de mécanismes de gestion des flots, nous décrivons la proposition Self-Protect, pour enfin illustrer ses avantages dans un petit nombre de cas d'utilisation représentatifs pour un utilisateur du réseau d'accès.

Sommaire

Contributions :

- Proposition d'une architecture de QoS au niveau flot adaptée au réseau d'accès.
- Méthode de synchronisation de table de flots, permettant la gestion du trafic descendant
- Réalisation d'un contrôleur permettant la gestion décentralisée d'une table de flots
- Réalisation d'un prototype et démonstrations internes.

Publications et présentations :

Les résultats de ce chapitre n'ont pas été publiés. La réalisation du prototype a été faite conjointement avec Elie Roux, qui a effectué son stage de fin d'études au sein de l'équipe, et pour qui j'ai assuré l'encadrement technique.

7.1 Introduction

Alors que les questions de Qualité de service dans le réseau de cœur sont débattues et divisent les chercheurs, il est généralement reconnu que le réseau d'accès présente un goulot d'étranglement pour le trafic. Récemment, le problème de *bufferbloat* [60] a été exhibé et a entraîné un regain d'intérêt de la communauté pour définir des mécanismes de contrôle efficaces pour le réseau d'accès (notamment les AQM CoDel, FQ-Codel et PIE). Mais le problème est complexe et n'a toujours pas reçu de réponse satisfaisante.

L'introduction de *fair queueing* tel que proposée dans Cross-Protect convient pour le cœur de réseau avec des liens de forte capacité. Elle n'est pas suffisante pour un lien d'accès, où un utilisateur peut souhaiter exprimer des politiques de gestion du trafic complexes, et où un flot *streaming* à protéger peut avoir un débit occupant une grande partie de la capacité d'accès.

Dans ce chapitre, nous considérons un mécanisme de gestion du trafic pour les ressources près du client, comme le dernier kilomètre d'un réseau ADSL, ou la connexion sans-fil à un point d'accès. Dans le sens montant, depuis le réseau domestique jusqu'au premier nœud du réseau, il est relativement naturel que l'utilisateur puisse déterminer les priorités qu'il souhaite donner aux flots concurrents qu'il génère, via les "boxes". Pourtant, dans le réseau actuel, c'est plutôt l'opérateur qui impose ses propres priorités. D'autant plus dans le sens descendant, où l'opérateur donne priorité au trafic à valeur ajoutée provenant de son propre réseau (VoIP, flux TV, etc.), tandis que les autres flots restent traités en *Best-Effort*.

Notre approche propose la réalisation dans le réseau des mécanismes permettant à l'utilisateur de déterminer l'ordonnancement et le contrôle appliqué à ses flots dans le sens descendant, de la même manière que dans le sens montant. Le profil des flots est déterminé par l'équipement domestique de l'utilisateur, qui peut analyser le ou les premiers paquets et les associer à une politique locale. Il peut alors être utilisé localement et envoyé à l'équipement en amont à l'aide d'un protocole de signalisation.

Au delà de permettre une gestion plus appropriée du trafic utilisateur, cette architecture qui réalise les mécanismes FAN dans le réseau d'accès permet de collecter des informations importantes sur le réseau du côté client, permettant le diagnostic des pannes ou de certains problèmes de connexion ou de performance. Un prototype GNU/Linux a été réalisé afin de montrer les avantages et la faisabilité d'une telle architecture.

7.2 Mécanismes actuels de garantie de service pour le réseau d'accès

L'architecture des réseaux d'accès diffère fortement d'un réseau à l'autre en fonction du contexte ou des technologies utilisées : réseau filaire ADSL, réseau mobile UMTS ou LTE, accès VPN d'entreprise, etc. On constate généralement soit l'application de mécanismes de type DiffServ, soit l'absence de gestion du trafic. Nous présentons plus en détail le cas d'un accès ADSL, pour lequel nous illustrons notre proposition. Nous remarquons toutefois son applicabilité aux autres types de réseaux d'accès.

7.2.1 Gestion du trafic montant

La configuration la plus classique dans un accès ADSL est la présence d'une passerelle à la frontière entre le réseau d'accès et le réseau domestique. Il s'agit généralement d'un système d'exploitation GNU/Linux, pour lequel sont disponibles la plupart des mécanismes classiques présentés dans le Chapitre 2, section 2.3[1].

Les premiers cas d'utilisation de mécanismes de QoS, dans les cas favorables d'équipements ouverts à l'utilisateur, remontent à 2002 avec des scripts de type *Wondershaper* [74], ayant pour objectif :

- le maintien d'une faible latence pour le trafic interactif (eg. SSH),
- la navigation à une vitesse raisonnable pour l'*upload* et le *download*,
- l'assurance que les transferts dans les sens montant et descendant ne se gênent pas les uns les autres.

De telles solutions sont très complexes et mettent en œuvre des mécanismes de partage de bande passante, des limites de débit (*shaping*), etc. Avec l'avènement des offre de *triple-play*, la présence de *box* opérateur s'est généralisée. Ces équipements ne disposaient généralement d'aucun mécanisme de QoS et ont longtemps géré le trafic en FIFO. Une exception est la Freebox en France, qui propose nativement une file prioritaire classifiant le trafic en fonction du champ IP ToS (*Type of Service*), et applique l'ordonnancement *Stochastic Fair Queueing* (SFQ) pour les files contenant du trafic élastique. Plus récemment, l'utilisation de l'ordonnanceur *Shortest Queue First* (SQF) est proposé pour la Livebox d'Orange [23,

1. voir par exemple le tutoriel de référence [102] pour une description détaillée de ces mécanismes

29, 16]. Il possède l'avantage de permettre un traitement du trafic au niveau des flots, ainsi qu'une différenciation implicite du trafic *streaming* basé sur le caractère *bursty* des flots. L'utilisation de SQF permet de se passer d'un marquage explicite généralement effectué par les applications, qui a tendance à ne plus trop être utilisé.

Aujourd'hui, l'identification du problème de *bufferbloat* [60] a relancé l'intérêt de la communauté pour proposer de nouvelles techniques de gestion du trafic pour le réseau d'accès, comme PIE [123] ou CoDel [115] qui permettent de contrôler les files d'attente afin d'y limiter le temps de séjour des paquets. Une combinaison de SFQ et CoDel (FQ-CoDel) a également été proposée par Dumazet [70], permettant en plus de donner priorité aux nouveaux flots afin de favoriser le trafic interactif, un mécanisme déjà présent dans la solution Cross-Protect (Chapitre 3). Enfin, la discipline *Fair Queue Packet Scheduler* [48], permet d'offrir un service de lissage du trafic à la source (*pacing*), qui peut-être utile pour éviter les rafales au sein des flots élastiques.

7.2.2 Gestion du trafic descendant

Dans le sens descendant ADSL, la plupart du trafic Internet est géré en *Best Effort*, alors que des mécanismes de gestion de trafic protègent généralement les flux de l'opérateur, comme la téléphonie, la TV ou la VoD. Il s'agit par exemple des circuits virtuels de l'architecture ATM, ou d'une classe DiffServ pour les réseaux IP. Il en est de même pour les réseaux mobiles, où la technologie UMTS définit par exemple 4 niveaux de priorités pour protéger le trafic le plus sensible : conversationnel, *streaming*, interactif, et *background*.

Afin de classifier les flots dont l'origine n'est pas connue, il est également possible de recourir à des techniques d'inspection plus ou moins sophistiquées (dites DPI, *Deep Packet Inspection*), qui peuvent inspecter jusqu'au contenu applicatif des paquets ou des flots. De telles solutions sont généralement utilisées pour identifier et appliquer des traitements aux flux applicatifs (dans des réseaux VPN d'entreprise par exemple), voire pour bloquer ou limiter certains types de trafic (pair-à-pair, applications non incluses dans un forfait mobile, etc.). Ces méthodes sont en conflit avec le principe de neutralité des réseaux, et sont actuellement sujettes à de nombreux débats.

7.2.3 Discussion

Comme cela a déjà été évoqué dans le Chapitre 2, même ces nouveaux mécanismes restent complexes à paramétrer et reposent pour certains sur une coopération des utilisateurs au moyen de TCP. Leur capacité de différenciation reste limitée, et ils n'offrent pas de performance prévisible et garantie, ni de protection contre une surcharge au niveau flot.

Il est également possible que l'utilisateur souhaite formuler des besoins plus précis, comme donner plus de débit à certains flots, ou en traiter d'autres en basse priorité. L'interaction entre les protocoles TCP et les AQM est mal connue et peut causer des résultats inattendus [63]. C'est pourquoi nous recommandons l'utilisation d'un ordonnancement par flot, guidé explicitement par l'utilisateur. Ceci nous permettra en outre une solution respectant le principe de neutralité des réseaux.

L'utilisation de *fair queueing* ou de FQ-CoDel seul n'est pas satisfaisante pour le réseau d'accès, où le profil de trafic d'un utilisateur peut être très complexe. Par exemple, une vidéo HD peut avoir un débit crête qui est une part significative du débit d'accès ; un utilisateur n'acceptera pas de dégrader la qualité de cette vidéo parce qu'elle sera en compétition avec des téléchargement et que son débit excèdera le débit équitable.

Enfin, nous proposons comme précédemment de réaliser une différenciation des flots *streaming*, afin de leur garantir un réseau transparent et leur assurer de faibles délais au sein des buffers, réglant ainsi le problème de *bufferbloat*. L'espace dans les buffers sera mis à profit pour garantir un débit satisfaisant pour les connexions TCP (voir Chapitre 4).

7.3 Vers une architecture de QoS pour le réseau d'accès

Dans cette section, nous discutons la réalisation d'un mécanisme de garantie de QoS pour le réseau d'accès, à la lumière des résultats présentés dans les chapitres précédents. La gestion du trafic concerne à la fois les sens montant et descendant, afin de traiter respectivement les flots envoyés et reçus par un utilisateur. Bien qu'elle n'offre pas de réponse complète et définitive au problème, l'architecture que nous présentons ici offre un compromis raisonnable entre performance et simplicité de mise en œuvre.

7.3.1 Classes de trafic et ordonnancement

Comme pour le cœur de réseau, nous supposons qu'il est possible de gérer le trafic utilisateur au moyen d'un faible nombre de classes de trafic, afin de permettre la réalisation de la solution sur un équipement gérant un grand nombre d'utilisateurs comme un DSLAM (nous supposons que l'allocation d'une certaine capacité à un utilisateur à été effectuée, indépendamment de la manière dont cette capacité sera partagée entre les flots) :

trafic *streaming* protégé : le trafic *streaming* sera traité en priorité selon les critères du multiplexage "sans buffer" présenté dans le Chapitre 4. Il conviendra pour cela de contrôler la charge de cette file d'attente. Lorsque cette classe de trafic constitue une part significative du trafic, Bonald *et al.* [25] suggèrent qu'il peut être nécessaire de distinguer entre les flots VoIP et les autres flots *streaming* moins sensibles à la latence (TV, etc.).

trafic élastique : ce sont les flots élastiques correspondant à du trafic de navigation (page web, email, etc) et nécessitant des garanties au niveau du temps de réponse. Nous recommandons ici l'utilisation de *fair queueing* entre les flots, tout en reconnaissant qu'il existe certainement des cas où l'on voudrait raffiner la gestion du trafic.

L'extension de la proposition avec des mécanismes permettant une différenciation entre flots élastiques serait une piste intéressante pour l'extension de la proposition. Nous n'avons pas non plus considéré l'introduction de mécanismes de *pacing*.

trafic à faible priorité : cette classe contiendra les flux élastiques non prioritaires, pouvant par exemple être interrompus et reprendre plus tard, et ne manifestant pas de phénomène d'impatience. C'est pas exemple le cas des gros téléchargement ou des transferts de fichiers en peer-to-peer.

Nous recommandons l'utilisation de *fair queueing* pour cette classe également, bien que sa performance ne nous intéresse pas. Le trafic de faible priorité sera géré en *Best Effort* lorsque de la bande passante sera laissée disponible par les autres classes. Il serait également possible de réserver une fraction de la bande passante pour cette classe.

trafic bloqué : comme son nom l'indique, elle contiendra les flux que l'on ne souhaite pas voir acheminés sur le lien.

Il conviendra d'associer des estimateurs de performance aux différentes classes de trafic pour lesquelle on veut offrir des garanties de performance, à l'image du *priority load* et du *fair rate* introduits dans le Chapitre 3 pour les flots *streaming* et élastiques.

7.3.2 Différenciation

Différenciation implicite streaming-élastique

La discipline *Shortest Queue First* (SQF) a été proposée pour gérer les flots au niveau des liens d'accès [23, 29]. Elle exploite le fait que les packets des flots *streaming* sont typiquement régulés par un codec, plutôt que par un algorithme de contrôle de congestion, et ce quel que soit le protocole de transport utilisé. Ils génèrent peu d'accumulation au sein des files d'attente, ce qui peut-être utilisé pour les différencier des flots élastiques. SQF nécessite que la charge des flots *streaming* ne soit pas trop importante, et que le buffer soit convenablement dimensionné. Ce mécanisme suppose de plus que les flots élastiques n'implémentent pas de mécanisme de *pacing*.

Différenciation implicite élastique-élastique

La discipline SRPT (*Shortest Remaining Processor Time*) [90] est connue pour être optimale en ce qui concerne le temps moyen de réponse des flots ; elle nécessite cependant une connaissance préalable des tailles de flots, ce qui n'est généralement pas le cas pour les réseaux. Plusieurs propositions ont été faites afin d'exploiter les avantages d'une différenciation basée sur la taille des flots, comme LAS (*Least Attained Service*) [69], qui est une bonne approximation de SRPT, ou encore RuN2C [10]. L'intêret d'une telle différenciation est également discuté par Bonald *et al* [25], notamment en cas de congestion, où les auteurs suggèrent que la solution est une alternative au contrôle d'admission qu'il serait intéressant de considérer et d'étudier. Ce même article suggère que les flots *streaming* peuvent souffrir d'une telle différenciation, et doivent être traités séparément, comme nous le faisons ici.

Différenciation explicite

L'avantage de la différenciation implicite est de ne pas dépendre des protocoles/contenus/chiffrements/etc., mais elle n'est pas toujours possible ou suffisante. Le nombre limité de flots utilisateur permet cependant l'utilisation de mécanismes avancés de différenciation explicite pilotés par l'utilisateur. Cela permet l'identification du trafic dans le sens montant (typiquement contrôlé par l'utilisateur), mais également dans le sens descendant comme nous le verrons dans la section 7.4.3.

7.3.3 Gestion de la surcharge

La garantie de faibles délais et taux de pertes pour la classe prioritaire, ainsi que d'un débit minimum pour la classe élastique requiert l'utilisation de mécanismes de contrôle de charge, similaires à ceux proposés dans l'architecture Cross-Protect. Il est désirable en effet de limiter l'accès aux files protégées lorsqu'un nouveau flot dégraderait la performance globale du système, et/ou de pouvoir en informer l'utilisateur. Il est alors possible de limiter la dégradation de performance à un sous-ensemble de flots bien délimité, qu'il est facile de communiquer à l'utilisateur.

Estimateurs de surcharge et contrôle d'admission

Afin de préserver les flots *streaming*, il est pratique de se reposer sur un algorithme de contrôle d'admission garantissant les conditions de multiplexage sans buffer (voir Chapitre 6, section 3.2.1), tel que celui proposé par Grossglauser et Tse [65, 66] et détaillé dans la section 6.2.7 du Chapitre 6 (mesure de la moyenne et de la variance de la charge prioritaire). La charge critique dépendra en effet de la composition du trafic entrant et de ses caractéristiques, notamment la combinaison des débits crête.

L'utilisation d'un ordonnancement *fair queueing* pour la classe élastique nous permet d'estimer le débit équitable sur le lien, ou *fair rate* (Chapitre 3, section 3.5.2) : il s'agit du débit qu'aurait un flot en l'absence de contrainte extérieure. Il est possible de détecter une surcharge lorsque ce débit équitable devient trop faible (il reste toutefois à définir un seuil efficace comme pour le cœur de réseau).

Le rejet d'un flot est difficilement envisageable sur le lien d'accès de l'utilisateur, sauf s'il est possible de l'utiliser comme un signal de reroutage dans le cas d'un *multi-homing*. Il est par contre possible d'admettre le flot dans la classe *Best-Effort*, avec information à l'utilisateur, en attendant une admission ultérieure. Cette solution permet d'admettre le flot tout en protégeant ceux déjà établis, un peu à la manière de la solution *Conditionally Dedicated Bandwidth* présentée dans le Chapitre 2 en section 2.4.1.

Préemption de flots

Dans notre contexte, un erreur d'admission, ou tout simplement le changement de profil d'un flot peut remettre en cause les décisions d'admission faites précédemment. Il convient alors de considérer la préemption de certains flots. Encore une fois, le choix des flots à interrompre est une tâche complexe qui peut se reposer sur des heuristiques (comme le rejet des derniers flots admis) ou l'interaction avec l'utilisateur.

7.4 L'architecture Self-Protect

L'architecture Self-Protect propose la gestion au niveau flot du trafic montant et descendant sur le lien d'accès d'un utilisateur. Elle repose sur le maintien d'une table des flots au sein des équipements, et d'un découplage entre les décisions de QoS faites pour les flots (par un contrôleur), et leur application. La gestion du trafic dans le sens descendant est rendue possible par la décentralisation du contrôleur vers le client (par exemple dans la passerelle domestique), qui communique le profil des flots à l'équipement en amont au travers d'un protocole de signalisation (voir Figure 7.1).

La proposition nécessite la définition a priori d'une architecture de gestion du trafic telle que faite dans la section précédente. Par souci de généralité, nous considérerons ici qu'un certain nombre de classes de service ont été définies, et qu'il est possible d'associer un profil de trafic à chaque flot afin qu'il soit affecté à l'une de ces classes. Ce profil peut correspondre simplement à un identifiant de priorité, mais être également plus complexe avec par exemple des garanties minimales et/ou maximales de bande passante.

7.4.1 Fonctionnement de Self-Protect

Supposons qu'un nouveau flot arrive depuis le réseau, afin d'illustrer le fonctionnement de notre proposition. Il est inconnu sur le DSLAM qui applique un traitement par défaut. Lorsqu'il atteint la

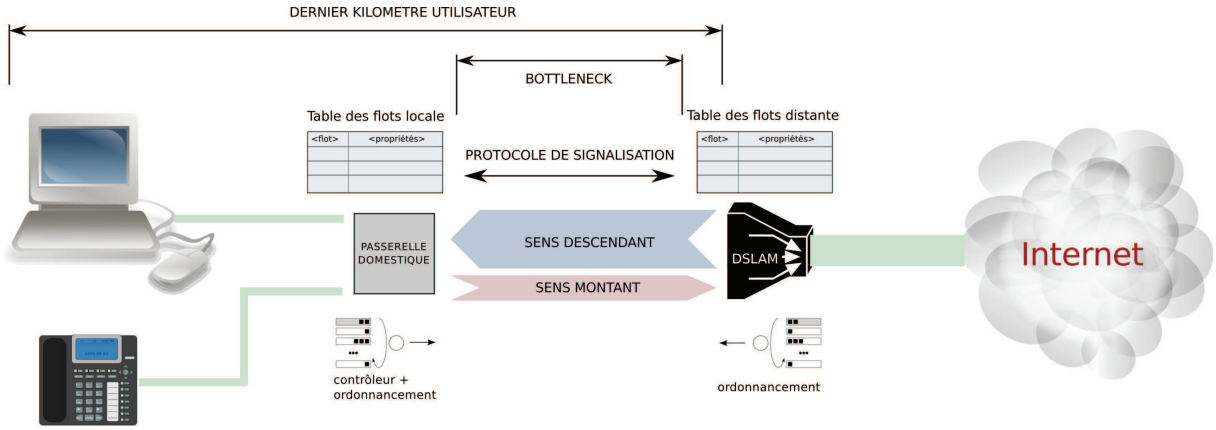

FIGURE 7.1 – Positionnement de l'architecture Self-Protect dans le réseau d'accès

passerelle domestique, il est reconnu comme nouveau flot, une nouvelle entrée est crée dans la table des flots, et un profil de QoS lui est affecté en fonction de la politique locale. L'identifiant de flot et son profil sont transmis au DSLAM au moyen d'un protocole de signalisation, qui les ajoute à sa table de flots. Une fois le flot reconnu, les paquets suivants seront ordonnancés de la même façon dans les deux directions[2]. Parfois, le premier paquet n'est pas suffisant pour déterminer la nature réelle du trafic, et il est tout à fait possible d'envoyer ou renvoyer la signalisation plus tard.

Le seul besoin essentiel de la proposition est de maintenir une table de flots dans les équipements en bordure de réseau. Elle peut ainsi s'appliquer à plusieurs types de réseaux d'accès filaires, comme xDSL ou PON – elle serait alors respectivement dans le DSLAM ou dans le switch/routeur –, ou sans-fil – elle serait alors localisée dans le RNC pour le réseau UMTS, ou dans le point d'accès d'un hotspot WiFi par exemple. Par souci de simplicité, nous nous restreindrons ici à l'étude d'un réseau d'accès ADSL.

7.4.2 Gestion des tables de flots

Une table des flots permet d'associer à chacun un profil de trafic (par exemple l'identifiant d'une file d'attente). Les deux fonctions minimales suivantes doivent être appliquées par les équipements.

traiter les paquets recus : Pour chaque paquet entrant, l'équipement cherchera une correspondance dans la table des flots, et marquera le paquet en fonction du profil de QoS, afin de l'affecter à la bonne file d'attente. Un traitement par défaut sera appliqué pour les paquets pour lesquels aucun flot n'est trouvé. Ce sont typiquement les premiers paquets d'un flot, ou ceux pour lesquels on ne souhaite pas associer de profil de trafic (flots courts de 1 paquet par exemple). Par souci de performance, il sera possible d'affecter un certain nombre de profils par défaut, qui ne nécessiteront pas de signalisation. Par exemple pour traiter en priorité les requêtes DNS, qui dépassent rarement 1 paquet.

gérer les flots inactifs : La date de dernière activité du flot sera mémorisée afin de pouvoir purger périodiquement la table des flots terminés. Il sera possible d'affecter un *timeout* plus faible sur le DSLAM afin de limiter la taille de la table des flots, quitte à ce que l'entrée d'un flot qui redevient actif soit rétablie au travers du protocole de signalisation.

Le DSLAM devra en plus obéir à la signalisation reçue par le contrôleur.

7.4.3 Contrôleur et protocole de signalisation

La reconnaissance des flots, l'attribution d'un profil de trafic, ainsi que les décisions d'admission sont entièrement réalisées par un contrôleur, décentralisé sur les équipements client qui sont seuls responsables du contrôle de leurs flots. Cette solution permet de distribuer la charge d'identification des flots et d'attribution d'une classe de service sur l'équipement utilisateur, en l'occurrence la passerelle domestique pour l'ADSL. Le logiciel tournant sur le DSLAM est réduit à son strict minimum : une interface permettant la manipulation d'une partie restreinte de la table des flots utilisée pour la gestion du trafic.

2. En fait il sera possible de donner des profils différents en fonction du sens, par exemple pour donner priorité au flux d'acquittement des connexions TCP

Reconnaissance des flots

Les premiers paquets de chaque flot sont redirigés vers un contrôleur dont la responsabilité est de leur affecter un profil de trafic. La reconnaissance des flots est faite de manière classique (sur le quintuplet IP et ports source et destination + protocole pour IPv4, sur les IP + le *flow label* pour IPv6).

La reconnaissance du profil des flots peut-être faite de manière implicite comme présenté plus haut, ou de manière explicite par un ensemble de règles, sous le contrôle de l'utilisateur. Ces règles pourront être choisies sur mesure par chaque utilisateur, par exemple à partir d'un dépôt contenant les applications les plus populaires, ainsi qu'un ensemble de profils-type d'utilisateur (par exemple un joueur voudra que les flots associés aux jeux-vidéos soient traités en priorité afin d'avoir une latence minimale).

La table de flots du côté utilisateur pourra également contenir un ensemble de statistiques utile pour le contrôleur afin de procéder à l'identification des flots (comme sa taille en octets, le nombre de paquets ou encore sa durée). L'utilisateur pourra bien sûr à tout moment ajouter de nouvelles règles ou modifier manuellement le profil d'un flot. Il sera également possible d'offrir un mode d'apprentissage comme pour les *firewalls* qui permettra la construction d'un jeu de règles personnalisé pour l'utilisateur. Enfin, pour le trafic montant, il sera toujours possible de s'appuyer sur le champ ToS (*type of service* de l'en-tête IP, qui contient des préférences de QoS demandées par les applications supportant cette fonctionnalité.

Gestion de la surcharge

Les décisions prises par le contrôleur se font à partir de l'inspection des paquets entrants, ainsi que des statistiques collectées par la table des flots. Il sera nécessaire, notamment pour les décisions d'admission, d'obtenir des métriques indiquant le niveau de congestion des différentes files d'attente, à la fois dans les sens montant et descendant.

Ces informations pourront être envoyées périodiquement par le DSLAM (à des fins de statistiques), voire uniquement en cas de congestion. La meilleure solution reste encore à déterminer. Il conviendra à cet effet de donner une priorité absolue au trafic de signalisation afin de permettre le fonctionnement du système. Le contrôleur disposera de la classe "trafic bloqué" afin de bloquer l'admission d'un nouveau flot, ou de préempter un flot en cours.

Faisabilité

Tables de flots et signalisation

La plupart des briques nécessaires à la réalisation de la proposition Self-Protect existent depuis longtemps.

Par exemple, les équipements tournant sous le système d'exploitation Linux possèdent déjà une implémentation de table de flots au travers du système *netfilter* [114], qui permet un suivi de connexion pour la réalisation de *firewalls*. Le système est utilisé sur de nombreuses passerelles domestiques, et même sur des équipements réseau [153].

Plus récemment cependant, le protocole OpenFlow [108] a été proposé par McKeown *et al.* et a gagné suffisamment de reconnaissance de la communauté et de l'industrie pour être reconnu comme un standard et implémenté dans de nombreux équipements. OpenFlow repose sur des concepts similaires à Self-Protect en ce qu'il permet de piloter une table des flots distante en découplant le contrôleur de la gestion de la table des flots. Initialement prévu pour le routage, les dernières versions d'OpenFlow proposent un support très basique de la QoS, permettant l'affectation d'un flot à une file préexistante. Les extensions QoS d'OpenFlow pourraient s'inspirer des protocoles classiques tels que RSVP [165] ou uPNP pour décrire le profil des flots, et être éventuellement étendues pour permettre le rapatriement des statistiques sur les files d'attente nécessaire pour l'application d'un contrôle de surcharge.

Des implémentations libres de contrôleurs existent, comme NOX [68], et pourraient être étendues pour le support de nos besoins de QoS. Enfin, l'isolation entre les utilisateurs pourrait être assurée par des solutions de virtualisation de l'espace de flots, comme FlowVisor [141].

Sécurité et charge du DSLAM

La gestion d'une table de flots à des fins de QoS est typiquement moins sensible que des problèmes similaires de pare-feu. La décentralisation du contrôleur permet de déplacer sur l'équipement client les opérations coûteuses d'identification de flots et de traitement, ainsi que celles pouvant poser des problèmes de sécurité.

Reste le problème de la charge induite sur le DSLAM par la signalisation proportionnelle au nombre de flots. Il est possible d'une part de limiter ce trafic par utilisateur, et d'autre part de mettre en place des traitements par défaut permettant d'éviter une signalisation pour une grande partie des flots courts qui représentent la majorité des flots (par exemple les requêtes DNS).

Valeur ajoutée

La présence de tables de flots au sein des équipements n'est pas limitée au fonctionnement de la proposition. Elles sont déjà utilisées aujourd'hui pour l'implémentation de *firewalls*, voire par certains constructeurs pour faire du routage au niveau flot (voir Chapitre 2, section 2.4.1).

Les données recueillies sur la performance des flots permettent à un utilisateur de mieux comprendre la performance de son trafic, et de déterminer par exemple si un problème de congestion est dû à son réseau domestique, à une congestion sur le lien d'accès ou encore plus en amont dans le réseau. La mise en commun de ces données permettrait d'étendre les fonctionnalités de monitoring collaboratif qui sont actuellement développées, ou encore de faire de la détection d'attaque par DDOS (*Distributed Denial of Service*, déni de service distribué)).

Ordonnancements et stardardisation

Le plus difficile reste l'introduction d'ordonnanceurs appropriés au sein des équipements réseaux. La communauté pousse cependant pour l'adoption d'interfaces ouvertes [142], et les évolutions du côté des organismes de standardisation montrent l'intérêt des constructeurs à fournir des mécanismes de gestion du trafic sur le dernier kilomètre.

Parmi les nombreuses propositions de standardisation, nous pouvons retenir d'une part le *Broadband Forum* (anciennement *ADSL forum*) [42] et la *Home Gateway Initiative* (HGI) [73] qui proposent respectivement la standardisation des aspects réseaux du réseau d'accès, et des spécifications techniques de la passerelle domestique. Dans les deux cas, les objectifs de QoS visent à fournir une faible latence pour certains flots, et de bénéficier du multiplexage statistique.

Les deux propositions reposent fortement sur les mécanismes DiffServ pour attribuer les flots à plusieurs classes de service, et de leur associer soit une priorité, soit un poids pour un algorithme WRR, en fonction de leurs besoins. Il est prévu que la passerelle domestique et le DSLAM puisse effectuer une classification des flots dans les deux sens à partir d'un ensemble de règles stockés sur un serveur tiers, et interrogé au travers d'un protocole de signalisation approprié. Il est également prévu, mais non spécifié, que l'équipement utilisateur puisse demander un service différencié aux équipements en amont. La plupart de ces recommandations concernent le BRAS puisque les DLAMs opèrent encore souvent sur la couche 2 uniquement, mais l'évolution des réseaux d'accès pourrait déplacer le traitement vers le DSLAM. La spécification insiste enfin sur le besoin d'effectuer une protection contre la surcharge, par exemple à l'aide d'un contrôle d'admission.

7.5 Réalisation et évaluation de l'architecture Self-Protect

7.5.1 Prototype

Afin d'évaluer la faisabilité et l'efficacité de la solution, nous avons réalisé un prototype de Self-Protect au sein d'une petite plateforme expérimentale recréant les conditions d'un réseau d'accès. L'objectif de notre prototype était de se baser majoritairement sur des briques logicielles existantes en effectuant un minimum de modifications, afin de prouver qu'une telle solution est d'ores et déjà réalisable dans un réseau de production.

Architecture

La plateforme est composée de trois machines standard sous GNU/Linux, faisant respectivement office de client ADSL, de passerelle domestique et de DSLAM. Elle est illustrée sur la Figure 7.2. D'autres machines permettent la création de trafic artificiel et la diffusion de vidéos. Toutes les machines sont connectées avec une interface à 100MB/s, et le débit ADSL est simulé en forçant l'auto-négociation des cartes réseaux à 10Mb/s pour le sens descendant, et en utilisant un limiteur de débit logiciel (HTB) configuré à 1Mb/s pour le sens montant. Une passerelle domestique se comporte soit en routeur, soit comme un noeud transparent. Nous avons retenu ce dernier choix pour des raisons de simplicité (pas de NAT, d'adresses privées, etc.), ce qui ne change en rien les évaluations.

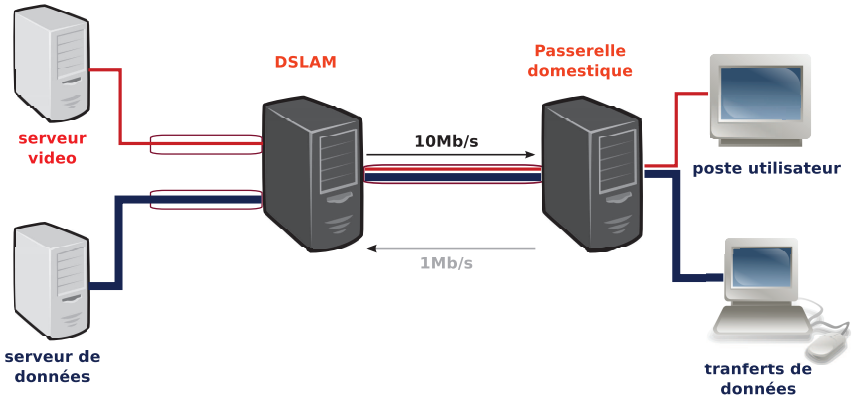

FIGURE 7.2 – Prototype Self-Protect

Classes de flots et ordonnancement

Le framework *netfilter* offre une interface permettant de contrôler le type d'ordonnancement souhaité pour les différents équipements, appelée *tc* (pour *traffic control*). Notre prototype utilise un mécanisme simple de priorités (*pfifo_fast*). Un ordonnancement *fair queueing* est ajouté pour les files de faibles et moyennes priorités afin d'assurer un partage équitable des ressources (nous avons utilisé l'algorithme *sfq*, qui est également disponible en standard).

Il suffit ainsi d'attribuer une marque (un entier) aux différents flots, et d'attribuer les paquets ainsi marqués à la file correspondante, par des règles *iptables*, de la même manière que l'on configurerait un pare-feu, au travers d'une interface spécifique (table *mangle*). Il est enfin possible d'assurer une bande passante minimale à la file la moins prioritaire grâce à un algorithme Weighted Fair Queueing (WFQ).

Contrôleur et caractérisation des flots

La passerelle domestique, le contrôleur, est responsable d'effectuer la reconnaissance des flots et leur signalisation. Les règles de caractérisation des flots pourraient être faites au niveau de l'interface *iptables* du framework *netfilter*, mais nous avons choisi de les faire au sein d'un démon tournant sur la machine pour plus de flexibilité.

L'intégration de la reconnaissance des protocoles au niveau 7 se fait au travers de l'outil *iptables*, qui supporte un grand nombre de protocoles reconnus. La démarche est d'installer les règles nécessaires permettant d'identifier les protocoles utilisés dans les filtres définis par l'utilisateur. Une marque sera alors associée au paquet, spécifiant le protocole identifié, et qu'il sera ensuite possible de réutiliser lors de l'application des règles de filtrage.

Les règles sont basées sur un ensemble de métriques maintenues par le démon en fonction du contenu ou de la taille des paquets, de leurs délais, ou de statistiques sur les flots (taille, durée, etc.). Nous utilisons également les possibilités de suivi de connexion offertes par le système. Enfin, un démon est installé sur la machine client afin de surveiller les applications lancées, et de leur associer des propriétés supplémentaires : utilisateur, nom de l'application, etc.

Tables de flots

Notre implémentation des tables de flots réutilise le système de suivi de connexions (*conntrack*) du framework *netfilter*, qui est intégré dans le noyau Linux, et utilisé dans le cadre de pare-feu ou de NAT. Le flot considéré est associé à un socket, ce qui correspond exactement à la notion de quintuplet classique. Nous avons uniquement redéfini la notion de *timeout* à quelques secondes (les spécifications usuelles sont plus longues). Il est possible d'associer une marque (*connmark*) à un flot, qui sera appliquée ensuite à chaque paquet. Le trafic de signalisation n'est pas traqué afin d'éviter une réaction infinie (grâce à la cible particulière NOTRACK).

Le système *conntrack* peut être piloté depuis n'importe quelle application grâce à la bibliothèque *libnfconntrack*, qui disponible pour de nombreux langages.

Deux démons (écrits en Python) tournent sur le DSLAM et la passerelle domestique, afin de gérer la table des flots. Le démon de la passerelle est chargé d'analyser les paquets afin d'associer le profil de trafic à chaque flot, de mettre à jour la table locale et d'envoyer un message au démon tournant sur le DSLAM. Ce dernier se contente de peupler sa table en fonction des messages reçus.

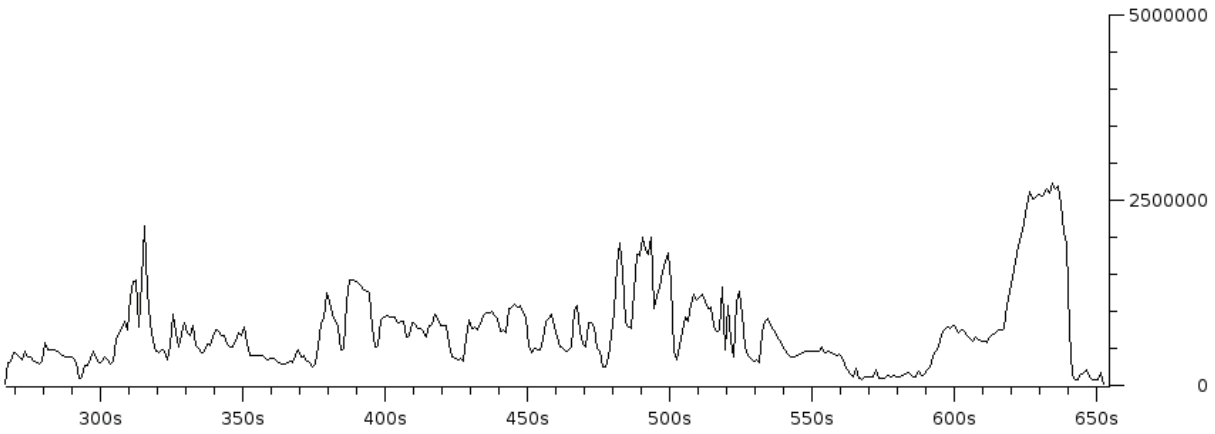

FIGURE 7.3 – Débit instantané de la vidéo en fonction du temps

Signalisation

A des fins de simplicité, nous avons implémenté notre protocole de signalisation sur XMLRPC. Chaque message contient un identifiant de l'action à faire (ajout/suppression de flot, modification de la marque, etc.), un descripteur de flot ainsi que les paramètres requis. En pratique, il conviendrait de réutiliser un protocole standard afin d'assurer l'interopérabilité des équipements.

Interface utilisateur

L'interface du prototype offre une vision des différentes classes de trafic, avec la liste des flots concernés, ainsi que leurs caractéristiques et statistiques. L'utilisateur peut promouvoir ou rétrograder un flot, le bloquer ou encore supprimer son entrée des tables de flots. L'interface permet la création et la gestion des différentes règles. Des graphes présentant la charge des différentes files d'attente sont également disponibles.

7.5.2 Évaluations

Cette section présente un ensemble de résultats expérimentaux qui ont été établis lors de démonstrations internes. Elle est volontairement courte puisque l'évaluation de la plateforme se ramène à celle des mécanismes d'ordonnancement associés (tels qu'une file prioritaire, etc.). Nous mettons en avant les choix justifiant les mécanismes introduits dans la Section 7.3, et le comparons aux solutions FIFO et FQ.

Différenciation *streaming*/élastique

Nous considérons un scénario classique où un flux vidéo à haut débit est en compétition avec un petit nombre de connexions TCP possédant également un fort débit. Nous avons choisi le film *Elephant Dream*, un court métrage d'animation libre [3], en version moyenne définition (1024x576) ; sa taille est de 446Mo. Nous avons capturé les paquets de la vidéo diffusée en UDP par le logiciel VLC, et analysé leur débit instantané avec Wireshark. Il est représenté en figure 7.3. On constate que la vidéo présente des instants avec un fort débit crête. Les connexions TCP concurrentes sont réalisées soit avec le logiciel `iperf`, soit en téléchargeant par FTP un gros fichier. Ces connexions ne sont pas limités en débit et se partagent la bande passante disponible.

Avec l'ordonnancement FIFO, la présence d'une connexion concurrente dégrade la vidéo notamment aux instants ou son débit crête est le plus élevé. En présence de 3 connexions parallèles, la vidéo n'est plus regardable.

L'utilisation de *fair queueing* permet de préserver la vidéo lorsqu'il n'y a qu'une connexion TCP (le débit équitable d'environ 5Mb/s est globalement supérieur au débit de la vidéo). Par contre, avec 4 connexions, ce débit équitable (d'environ 2Mb/s), n'est plus suffisant pour protéger la vidéo et l'on retrouve des dégradations aux instants ou son débit est plus élevé.

La priorité stricte offerte par Self-Protect permet la protection de la vidéo en toute circonstance.

3. disponible sur `http://www.elephantsdream.org`

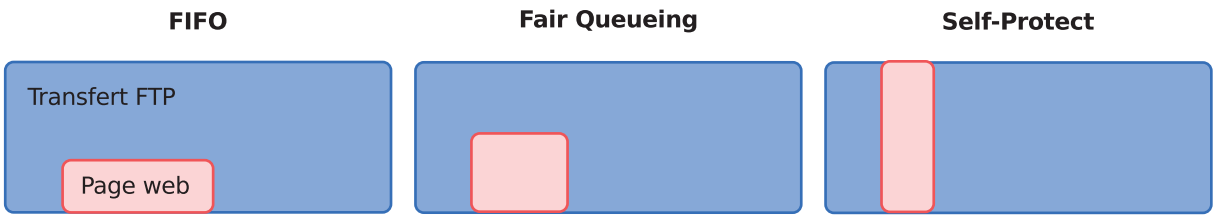

FIGURE 7.4 – Illustration du débit réalisé par le transfert web interactif en compétition avec des gros transferts de données dans les cas FIFO (à gauche), (*fair queueing*) au centre, et avec une priorité stricte (à droite).

Différenciation élastique/élastique

Afin de montrer l'intérêt de différencier les applications élastiques entre elles, nous considérons un scénario impliquant à la fois une session utilisateur de navigation Web, et des téléchargements de données à fort débit comme précédemment. Nous utilisons ici 4 connexions.

La session Web est simulée par le téléchargement d'une page web contenant de nombreuses images (une image découpée en plusieurs petits morceaux). Elle correspond typiquement à des flots dont l'interactivité ou la réactivité importent pour l'utilisateur.

Avec FIFO, l'affichage de la page est plutôt long. La présence de *fair queueing* améliore légèrement la performance, mais la différence est notable avec l'utilisation de deux files prioritaires afin de distinguer ces deux types d'applications élastiques.

L'utilisation de *fair queueing* ou de files prioritaires n'impacte pas le temps de transfert des fichiers concurrents, qui se produit sur une échelle de temps plus importante, comme illustré en figure 7.4. Les blocs représentent le volume de trafic à écouler pour chaque classe de flots. L'axe des abscisses représente le temps, et on représente en ordonnée le partage de la bande passante à chaque instant. Dans le cas FIFO (à gauche), le flux Web parvient à obtenir une part plus ou moins équitable de la bande passante, en fonction des mécanismes TCP sous-jacents. FQ (au centre), assure un partage équitable, et permet au flux Web d'obtenir la moitié des ressources. Enfin, le service en priorité effectué par Self-Protect (à droite), lui permet d'être servi immédiatement, donnant un temps de réponse maximum pour les flux importants pour l'utilisateur. On peut remarquer qu'un tel traitement n'affecte par le temps de transfert FTP, du moment que la charge des flux Web n'est pas trop importante.

Intérêt du *fair queueing* au sein d'une classe contenant des flux élastiques

Afin de mettre en évidence les apports du *fair queueing* au sein des classes les moins prioritaires recevant des flux élastiques, nous considérons les deux scénarios suivants :

connexions de RTT différents : Nous considérons deux sources de trafic avec un RTT différent. Pour cela, nous émulons un délai de propagation plus important pour l'une des deux sources à l'aide de la discipline de service `netem` disponible dans `tc`.

utilisation de protocoles TCP différents : Cette fois-ci, les RTT sont équivalents, mais l'une des deux connexions utilise un algorithme TCP avancé, comme BIC ou CUBIC, tandis que l'autre utilise la version standard TCP Reno.

Avec une discipline FIFO, la connexion ayant le RTT le plus faible ou l'algorithme TCP le plus agressif obtient un débit plus élevé. Une équité approximative est restaurée par l'utilisation de *fair queueing*.

Compétition sens montant et descendant

Ce scénario met en évidence le besoin de mettre en priorité les paquets d'acquittement (ACK) des connexions TCP qui sont dans la file la moins prioritaire. Pour cela nous générons une connexion TCP simple entre un serveur et la HGW que l'on met en priorité 3, et une connexion entre la HGW et le serveur que l'on met en priorité 2.

Avec une discipline FIFO sur la passerelle domestique, la connexion montante sature la bande passante et les ACK de la connexion descendante sont considérablement ralentis. La connexion descendante n'occupe pas toute la bande passante et obtient un débit très faible. La présence de files prioritaires bloque les ACK montants, ce qui stoppe la connexion descendante. Si on met les ACK de la connexion descendante en priorité 2 au lieu de 3, mais sans fair queuing sur la priorité 2, la file de priorité 2 contiendra les ACK de la connexion descendante et la connexion montante, on se ramène

donc au premier cas et on observe le même débit. Si enfin on met une discipline de fair queuing dans la file 2, la connexion descendante prend son débit maximum, sans pénaliser la connexion montante. Ce scénario illustre la nécessité de prioriser les ACK des connexions de priorité 3.

Surcharge

Pour terminer, nous illustrons l'importance de détecter des situations de surcharge afin de préserver la performance des applications, notamment en ce qui concerne les flux *streaming*. Nous considérons un scénario ou beaucoup trop de flux sont affectés à la classe la plus prioritaire (servie en FIFO).

Dans une telle situation, il n'est plus possible de garantir la performance des flots, et il convient de terminer ou déclasser certains flots. L'identification de la surcharge n'est pas faite par le prototype, mais il serait intéressant de détecter des situations prolongées où le trafic prioritaire excède une certaine limite, afin d'alerter l'utilisateur ou de prendre les mesures nécessaires automatiquement.

7.6 Conclusion

Dans les chapitres précédents, nous avons motivé l'introduction d'une architecture de gestion de la qualité de service au niveau flot, afin d'obtenir des garanties robustes de performance pour le trafic IP. Les modèles utilisés nous permettent le dimensionnement du réseau de cœur afin de transporter le trafic de manière transparente, et gérer les éventuelles situations de congestion et de surcharge.

Dans la plupart des déploiements, il est reconnu que le réseau d'accès est le principal goulot d'étranglement pour le trafic. Une solution telle que Cross-Protect n'est pas envisageable pour le réseau d'accès, qui possède des caractéristiques différentes en termes de capacité, de nombre d'utilisateurs, et de trafic. Nous avons proposé ici une réalisation alternative de l'architecture *Flow-Aware Networking*, dite Self-Protect, plus appropriée pour ce contexte.

Self-Protect repose sur la gestion du trafic au niveau flot, à la fois dans les sens montant et descendant, grâce à un ordonnancement approprié. Cet ordonnancement permet notamment de garantir la performance des différents flots *streaming* et élastique, envoyés et reçus par l'utilisateur (et résout les problèmes récurrents de *bufferbloat*). Les flots sont reconnus et analysés par un contrôleur situé sur l'équipement utilisateur (la passerelle domestique pour un réseau ADSL), afin d'être traités dans le sens montant. Pour le sens descendant, l'équipement (par exemple le DSLAM) maintient uniquement une table de flots, qui est gérée de manière distante par le contrôleur, au travers d'un protocole de signalisation. Il en résulte une solution distribuée, permettant sa réalisation dans un réseau d'opérateur, et satisfaisant les contraintes de neutralité des réseaux puisque les traitements effectués sont à l'initiative de l'utilisateur.

Nous avons proposé un prototype simple basé sur un petit nombre de classes de trafic, et démontré son efficacité dans des scénarios simples. Bien que suffisant pour de nombreux cas d'utilisation, il est possible d'étendre ce prototype afin de supporter des profils de trafic plus complexes, afin de satisfaire des besoins plus stricts pour des utilisateurs professionnels par exemple. Une réalisation exploitant les nouveaux mécanismes de *Software Defined Networking* (notamment *OpenFlow*) est à envisager…