Caractérisation, identification dans le réseau et optimisation du transport de trafic à faible latence - le cas du Cloud-Gaming

Philippe Graff

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Caractérisation, identification dans le réseau et optimisation du transport de trafic à faible latence - le cas du Cloud-Gaming

THÈSE

présentée et soutenue publiquement le 12 décembre 2023

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Philippe GRAFF

Composition du jury

Président :	Ye-Qiong SONG	Professeur à l'ENSEM, Université de Lorraine
Rapporteurs :	Chadi BARAKAT	Directeur de Recherche à Inria
	Stefano SECCI	Professeur au CNAM
Examinatrices :	Noura LIMAM	Assistant Professor at University of Waterloo
	Sandrine VATON	Professeure à l'IMT Atlantique
Encadrants :	Thibault CHOLEZ	Maître de conférences à TELECOM Nancy, Université de Lorraine
	Olivier FESTOR	Professeur à TELECOM Nancy, Université de Lorraine

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Remerciements

Je souhaite remercier en premier lieu Olivier Festor et Thibault Cholez, respectivement directeur et co-directeur de ma thèse. Tous deux étaient très favorables à mon inscription en thèse. Merci d'avoir cru en moi ! Merci surtout pour le suivi et pour les bons conseils promulgués tout au long de ces trois années de thèse.

Je tiens à remercier particulièrement les rapporteurs de cette thèse, Chadi Barakat et Stefano Secci, pour le temps qu'ils ont consacré à mon travail. Je souhaite également remercier Noura Limam et Sandrine Vaton pour l'intêret qu'elles ont porté à mes travaux en tant qu'examinatrices de cette thèse. Je remercie également Ye-Qiong Song et Enrico Natalizio pour avoir été mes référents scientifiques au Loria.

Le projet ANR au sein duquel j'ai eu la chance de participer (MOSAICO) m'a permis de faire de très belles rencontres. Je souhaite remercier ici l'ensemble des membres du projet, et plus particulièrement les collègues d'Orange à savoir Bertrand Matthieu, Stéphane Tuffin et Joël Roman Ky avec lesquels j'ai étroitement collaboré et co-écrit des papiers. Il y avait une synergie certaine et de la cohésion entre les différents partenaires. Les réunions étaient pour moi des moments très enrichissants d'échanges scientifiques, mais aussi de camaraderie. Petite pensée aux moments de convivialité que l'on a passés ensemble lors des réunions physiques ; les restaurants, la croisière au large de la Bretagne, etc. Merci !

Je n'oublierai pas de mentionner les membres de mon équipe de recherche (RESIST). Toujours souriants et disponibles pour répondre aux diverses questions. Merci à Xavier Marchal pour son expertise technique sans faille qui a largement contribué à la bonne conduite des expérimentations. Merci à Matthews Jose qui s'est rendu disponible pour répondre à mes questions vis-à-vis du langage P4. Merci à Abir Laraba, Mohammed Oulaaffart, Lama Sleem pour nos petites pauses café ! Merci à Omar Anser et Enzo D'Andréa pour vos blagues.

Mis à part mes collègues, je tiens à remercier ma famille et mes amis. Je pense qu'il est de bon aloi de commencer par remercier mes parents. Ils ont toujours été de mon côté, et m'ont toujours soutenu pendant les périodes de doute. Merci pour le soutien moral et financier !

Enfin, merci à mes amis, à ma bande de *potes* avec qui j'ai tellement rigolé. Jérôme Lichtsteiner, Xavier Marck, James Ren, Lee Senghtavisay... Que de bons moments passés ensemble !

Table des matières

Introduction générale **1**

1 Contexte . . . 1

1.1 Évolution des performances des réseaux d'accès en France . . . 1

1.2 Nouvelles technologies de programmabilité réseau . . . 2

1.3 Émergence d'un nouveau service *temps réel* : le Cloud Gaming . . . 3

2 Problématique . . . 4

2.1 Défis . . . 4

2.2 Problèmes traités . . . 5

3 Organisation des contributions . . . 5

Partie I État de l'Art **7**

1 Les technologies et défis du Cloud Gaming **9**

1.1 Introduction . . . 9

1.2 Contexte Technologique . . . 10

1.2.1 Architecture fonctionnelle . . . 10

1.2.2 Plateformes commerciales . . . 11

1.2.3 Aspects protocolaires . . . 12

1.3 Estimation de la QoS et de la QoE . . . 14

1.4 Optimisation de l'infrastructure côté serveur . . . 16

1.5 Adaptation du trafic aux conditions réseau . . . 17

1.6 Conclusion . . . 19

2 Mécanismes permettant de maîtriser la latence due au réseau **21**

2.1 Introduction . . . 21

2.2 Mécanismes de contrôle de congestion orientés latence . . . 22

2.2.1 BBR . . . 22

2.2.2 GCC . . . 22

2.2.3 C3G 25
2.2.4 Sprout 26
2.2.5 SCReAM 27
2.2.6 Problématique de la cohabitation entre CCAs 29
2.3 Mécanismes de gestion de file d'attente 30
2.3.1 RED 31
2.3.2 Codel 31
2.3.3 PIE 32
2.3.4 HTB 32
2.3.5 L4S 34
2.4 Conclusion 38

Partie II Caractérisation du trafic Cloud Gaming en environnement réseau contraint 39

3 Sur réseau à capacité fixe 41
3.1 Introduction 41
3.2 Testbed 42
3.3 Contraintes préalables à la session de jeu 43
3.3.1 Perte de paquets 44
3.3.2 Limitation de la bande passante 45
3.3.3 Ajout de latence 47
3.3.4 Génération de gigue 48
3.3.5 Variation des IAT et de la taille du payload face aux contraintes . . . 49
3.4 Contraintes durant la session de jeu 50
3.4.1 Perte de paquets 50
3.4.2 Limitation de la bande passante 52
3.4.3 Ajout de latence 53
3.4.4 Génération de gigue 54
3.5 Conclusion 55

4 Sur réseau à capacité variable 57
4.1 Introduction 57
4.2 Contraintes spécifiques aux réseaux cellulaires 58
4.3 Testbed 59
4.3.1 Captures des `txops` d'un réseau cellulaire 60

4.3.2 Émulation des capacités d'un réseau cellulaire . . . 61
4.3.3 Acquisition de trafic CG sur réseau cellulaire . . . 62
4.4 Analyse du bitrate . . . 63
4.4.1 Utilisateur fortement mobile . . . 63
4.4.2 Utilisateur statique . . . 64
4.4.3 Stabilité du débit . . . 66
4.4.4 Relation entre débit et qualité vidéo . . . 67
4.5 Analyse de la latence . . . 69
4.6 Conclusion . . . 71
4.6.1 Bilan . . . 71
4.6.2 Réflexions et pistes d'amélioration des réseaux . . . 72

Partie III Transport optimisé du trafic Cloud Gaming 75

5 Identification du trafic Cloud Gaming 77
5.1 Introduction . . . 77
5.2 Méthodes de classification des flux réseau . . . 78
5.3 Création et exploitation des jeux de données . . . 81
5.3.1 Données brutes . . . 81
5.3.2 Extraction des caractéristiques . . . 82
5.4 Modèles . . . 83
5.4.1 Base de référence . . . 83
5.4.2 Modèles d'apprentissage automatique supervisé . . . 84
5.5 Étude des hyper-paramètres . . . 85
5.5.1 Taille de la fenêtre temporelle . . . 86
5.5.2 Base de référence . . . 87
5.5.3 Decision Tree et Random Forest . . . 88
5.6 Évaluation des modèles . . . 89
5.6.1 Traces CG *normales* . . . 89
5.6.2 Traces CG *perturbées* . . . 90
5.7 Généralisation à d'autres jeux et plateformes . . . 91
5.7.1 Un modèle non supervisé : USAD . . . 91
5.7.2 Comparaison . . . 92
5.8 Conclusion . . . 93

6 Évaluation des stratégies d'implantation **95**
6.1 Introduction . . . 95
6.2 Architecture à base de micro-services . . . 96
6.2.1 Présentation des micro-services . . . 96
6.2.2 Évaluation des performances . . . 97
6.3 Programmabilité réseau grâce à P4 . . . 99
6.4 Classification en P4 logiciel depuis le *Data Plane* . . . 102
6.4.1 Testbed . . . 102
6.4.2 Extraction des caractéristiques en P4 . . . 102
6.4.3 Implantation d'un DT en P4 . . . 104
6.5 Approche de classification *Hybride* . . . 105
6.5.1 Matériel et limitations . . . 105
6.5.2 Impact des approximations . . . 105
6.5.3 Architecture finale . . . 107
6.6 Conclusion . . . 108

7 Priorisation du trafic Cloud Gaming **109**
7.1 Introduction . . . 109
7.2 Plateforme expérimentale de Cloud Gaming . . . 110
7.2.1 Streaming de jeu vidéo . . . 110
7.2.2 *Plugin* SCReAM . . . 112
7.3 Évaluation du CCA SCReAM appliqué au trafic CG . . . 114
7.3.1 Sur réseau cellulaire . . . 114
7.3.2 Face à du trafic concurrent . . . 116
7.4 Priorisation du trafic CG avec la discipline de file d'attente HTB . . . 118
7.4.1 Face à du trafic Cubic . . . 119
7.4.2 Face à du trafic BBR . . . 119
7.4.3 Premier bilan et limites de l'approche . . . 121
7.5 Priorisation du trafic CG avec l'AQM DualPI2 . . . 122
7.5.1 Face à du trafic Cubic . . . 122
7.5.2 Face à du trafic BBR . . . 123
7.6 Conclusion . . . 124

Conclusion Générale **127**

Productions **133**

Table des figures **135**

Liste des tableaux **137**

Annexes **139**

A Schémas complémentaires de la plateforme expérimentale de CG **141**

B Caractéristiques des captures de réseau cellulaire **143**

C SCReAM sur réseau cellulaire, autres traces **145**

Glossaire **149**

Bibliographie **151**

Résumé **165**

Abstract **166**

Introduction générale

1 Contexte

1.1 Évolution des performances des réseaux d'accès en France

Ces dernières années, les performances des accès réseau, que ce soit sur les lignes fixes ou mobiles, n'ont fait que croître en terme de débits.

Les opérateurs nationaux se sont engagés à améliorer leur couverture mobile dans le cadre du *New Deal mobile*, annoncé en 2018. La généralisation de la 4G fait partie des engagements pris pas les opérateurs. Quatre ans plus tard, la presque totalité des sites mobiles est équipée de la 4G [ARC23b]. L'Autorité de Régulation des Communications Électroniques des Postes et de la distribution de la presse (ARCEP) s'est livrée à une étude sur la qualité du réseau mobile en 2022 [ARC23a]. Le débit descendant vaut en moyenne 94 Mbps pour l'ensemble des opérateurs. La technologie 5G contribue fortement à ces résultats. On est bien loin des débits mentionnés par le rapport ARCEP de 2012 [ARC12]. Les débits médians en téléchargement se situaient alors entre 2 et 7 Mbps selon les téléphones et opérateurs.

Les réseaux d'accès filaires ont quant à eux été positivement impactés par le déploiement de la fibre optique. Cette technologie permet une augmentation très importante de la bande passante au niveau de la boucle locale, ce qui a eu pour conséquence de déplacer le goulot d'étranglement plus loin dans le réseau.Selon les cas, les débits descendants peuvent dépasser le Gbps alors que la technologie ADSLv2 ne pouvait atteindre un débit descendant supérieur à 30 Mbps dans les meilleures conditions. L'autre intérêt est que la fibre est beaucoup moins sensible à l'atténuation due à la longueur de la boucle locale qui pouvait énormément réduire les débits ADSL. Cela permet d'augmenter d'autant plus le débit moyen. Depuis 2012, le réseau Ftth (*Fiber to the Home*) s'est ainsi fortement développé. On dénombrait 2.165.000 prises éligibles en fin d'année [ARC12]. Près de 10 ans plus tard, cette grandeur est plus que décuplée et atteint 33.572.000 locaux. Cela représente 77% des connexions THD (*Très Haut Débit*). En décembre 2011, l'ARCEP dénombrait seulement 665.000 abonnements THD [ARC12].

En l'espace d'une décennie, le THD s'est généralisé sur les réseaux filaires et mobiles grâce au déploiement de la fibre optique et de la 4G/5G. Le débit moyen descendant de l'accès à Internet en France a ainsi plus que décuplé, passant de 6.6 Mbit/s en 2013 [1] à 170 Mb/s en 2022, ce qui constitue une augmentation historique. L'état des réseaux a permis l'émergence de nouveaux usages comme la généralisation du *streaming* de vidéos haute définition. L'ARCEP a d'ailleurs émis des statistiques quant au visionnage de vidéos. L'internet mobile permet un *streaming* "*parfait*" dans plus de 80% des cas [ARC23a].

Néanmoins, une autre métrique, pourtant primordiale pour certains services, a fait l'objet d'une bien moindre attention et d'une certaine stagnation : la latence. Celle-ci résulte principalement de la somme de trois facteurs :

1. `https://www.ariase.com/box/actualite/etude-internet-akamai-t4-2013`

1. la latence liée à la technologie d'accès, résumée dans le tableau 1 ;
2. le délai de propagation du signal qui est fonction de la distance entre les entités communicantes et défini par les lois de la physique ;
3. les délais introduits par tous les équipements réseau d'interconnexion traversés par le flux de données, notamment le temps passé par les paquets en file d'attente (*queuing*), c'est à dire dans la mémoire tampon (*buffers*) des équipements.

Ce dernier délai est par ailleurs fluctuant car fonction de la charge du réseau à un instant donné, et donc source de gigue (variation de la latence dans le temps). Il est regrettable que la latence reste aujourd'hui encore assez peu considérée car certaines applications interactives y sont très sensibles.

	Cuivre (xDSL)	FttH	3G	4G
Latence rajoutée	5-45ms	1ms	25-50ms	15-30ms

TABLE 1 – Délais inhérents aux différentes technologies d'accès [ARC23a]

1.2 Nouvelles technologies de programmabilité réseau

En plus de la montée en débit, les réseaux ont gagné de nouvelles capacités d'adaptation dynamique permises par l'émergence des technologies de programmabilité réseau.

En 2012, McKeown & al. créent OpenFlow, un protocole permettant de contrôler des appareils réseau [MAB+08] via une interface *standard* qui facilite l'administration des différents équipements. Un contrôleur peut ainsi spécifier des traitements selon les flux. OpenFlow est souvent associé à SDN (*Software Defined Networking*), une nouvelle approche de gestion des réseaux. Le principe consiste à séparer le *data plane* du *control plane*. L'intelligence est ainsi centralisée au travers d'un ou plusieurs contrôleurs. Ces derniers spécifient le comportement du réseau. À un autre niveau, l'avènement du langage P4 [BDG+13] apporte encore plus de flexibilité. Les équipements réseau compatibles peuvent être reprogrammés, par exemple pour définir le *parsing* des paquets. Cela permet une certaine indépendance par rapport aux protocoles existants ou futurs. Des actions peuvent ensuite être associées aux champs d'en-tête. L'avantage d'un programme P4 est qu'il est également indépendant du matériel sous-jacent (*target*). Les spécificités *hardware* sont seulement prises en compte lors de la compilation.

Nous nous sommes jusqu'à présent intéressés à une catégorie bien spécifique du *data plane*. Il s'agit du *forwarding plane*, incluant les *switchs* et les routeurs. Il existe cependant d'autres appareils, désignés sous le nom de fonctions réseau (NF) ou *middleboxes*. Un mouvement initié en 2012 consiste à virtualiser ces *fonctions réseau*. On parle alors de NFV (*Network Function Virtualization*) [ETS12]. Un appareil physique peut dès lors exécuter plusieurs VNFs (*fonctions réseau virtuelles*) en parallèle. Des solutions comme ClickOS ou OpenBox [MAR+04, BBHH16] permettent de déployer et de gérer ces fonctions. Un des avantages de la virtualisation est qu'elle facilite le déploiement dynamique des fonctions à la demande et leur passage à l'échelle. Exécutées dans des conteneurs ou des machines virtuelles, les VNFs peuvent être facilement dupliquées. Zhang & al proposent d'ailleurs une plateforme basée sur des conteneurs Docker[ZHR+16]. Chaque conteneur est associé à une VNF particulière, elle-même liée au traitement d'un flux ou à une classe de trafic. Afin d'assurer de bonnes performances sans matériel dédié, des technologies comme DPDK (*Data Plane Development Kit*) [Cor23] ou des systèmes de *zero-copy* empêchent que les paquets soient recopiés entre chaque VNF.

La virtualisation des fonctions réseau (NFV) est complémentaire au paradigme SDN. Elle représente un gain de flexibilité et facilite la gestion du *data plane*. L'utilisation de VNFs *monolithiques* comporte cependant des inconvénients. Des opérations peuvent en effet être redondantes le long d'une chaîne de traitement (*Service Function Chain*). Cela explique l'intérêt porté aux architectures par micro services [DGL+17]. Les VNFs sont alors divisées en plusieurs composants indépendants et réutilisables. Un orchestrateur est chargé d'en faire le chaînage, en vue d'aboutir à une SFC. De nombreuses opérations peuvent dès lors être factorisées, limitant l'utilisation des ressources. Chowdhury & al. proposent μNF, une architecture permettant de composer des micro services [CAB+20]. Ils comparent ensuite leur solution à une SFC composée de VNF *monolithiques*. À débit égal, μNF requiert bien moins de ressources que la base de référence considérée. Cela confirme l'avantage de l'architecture par micro services.

Étant donné ce contexte technologique, cette thèse s'inscrit plus précisément dans le cadre du projet ANR MOSAICO (2020-2023) [2] [3] (Multi-layer Orchestration for Secured and low lAtency applICatiOns). Le but de ce projet est d'améliorer la prise en charge par le réseau des services ayant une contrainte forte de latence en mettant à profit les nouveaux traitements rendus possibles par les technologies de programmabilité réseau. Il se pose le double objectif d'améliorer la QoS (Quality of Service) et la sécurité du trafic à faible latence, mais seule la QoS nous concerne ici. Pour de tels services, l'ajout de délais a en effet un impact immédiat et négatif sur la QoE (Quality of Experience). Les effets peuvent même être dramatiques pour certains cas d'usage, comme la télé-chirurgie ou le pilotage de drones à distance. Dans le cadre de ce projet, nous avons cependant choisi comme cas d'étude un autre service très sensible à la latence, qui compte davantage d'utilisateurs, et qui est plus facile à instancier en laboratoire : le Cloud Gaming.

1.3 Émergence d'un nouveau service *temps réel* : le Cloud Gaming

Le "*Cloud Gaming*" (CG) consiste à séparer un joueur de l'équipement qui exécute le jeu à travers le réseau, comme illustré par la figure 1. Un flux vidéo haute résolution est envoyé à la volée du serveur vers le client (*joueur*) pendant que Les commandes de jeu transitent dans le sens inverse. Elles permettent au serveur d'appliquer la "*logique du jeu*". Cela consiste simplement à tenir compte des actions qu'a fait le joueur. Les flux de données engendrés par un service de CG

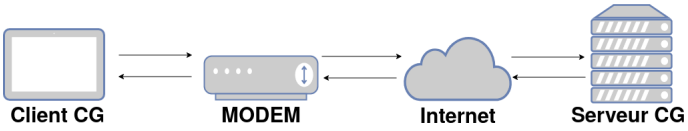


FIGURE 1 – Illustration d'un service de Cloud Gaming

sont particulièrement exigeants pour le réseau. Le flux descendant est très haut débit et peut atteindre typiquement 30Mb/s du fait de la très haute résolution de la vidéo (4K), du nombre important d'images par seconde (60) nécessaire pour assurer la fluidité du jeu, et de la moindre compression comparée à la transmission d'une vidéo pré-enregistrée du fait de sa génération en temps réel. Un retard dans l'acheminement des données dû à la latence se traduit rapidement par un manque de réactivité des contrôles alors que des pertes produisent des artefacts vidéo. Il y a donc un réel enjeu de QoE à maîtriser la QoS réseau d'un tel service.

2. `https://www.mosaico-project.org/`
3. No ANR-19-CE25-0012

Le CG présente pourtant de nombreux avantages. Tout d'abord, un joueur peut jouer à un même jeu sur des supports différents (TV, smartphone, tablette). Pour cela, il suffit que l'équipement dispose d'un client léger pour se connecteur au serveur, parfois un simple navigateur web, et d'une connexion internet. Ses avancées dans le jeu sont alors synchronisées sur chacun de ses appareils. En plus de la pluralité des supports, on peut souligner la pluralité des jeux. Dès lors qu'un utilisateur souscrit un abonnement de Cloud Gaming, il a en général accès à tout un catalogue de jeux, tous disponibles en quelques secondes car sans installation préalable, ce qui favorise la découverte et l'expérimentation. De surcroît, il n'est plus nécessaire pour le joueur d'investir dans un équipement très performant pour bénéficier de graphismes de qualité. La gestion du matériel est tout simplement reléguée au gestionnaire de la plateforme de Cloud Gaming.

La première plateforme de CG à avoir vu le jour est la plateforme "*onlive*". Elle avait été présentée en 2009, lors de la "*Game Developer Conference*". Malheureusement, la société qui l'exploitait n'est jamais parvenue à atteindre son seuil de rentabilité. L'entreprise et la technologie sous-jacente ont ainsi été rachetées par Sony en 2015. Étant donné l'intérêt porté pour ce nouveau concept qui semble promis à un bel avenir, plusieurs multinationales des technologies de l'information et de la communication ou de l'électronique ont mis au point leur propre plateforme :

— Google, avec la plateforme *Stadia* ;
— Nvidia, avec la plateforme *GeForce Now* ;
— Microsoft, avec la plateforme *Xbox Cloud Gaming* ;
— Sony, avec la plateforme *PlayStation Now* ;
— Amazon, avec la plateforme *Luna*.

Le concept présenté est très attrayant, mais s'accompagne d'un certain défi du point de vue du réseau. La QoE, qui est primordiale pour la satisfaction des utilisateurs et la pérennité d'une plateforme, est impactée par plusieurs facteurs : la qualité de la vidéo streamée au client qui est directement liée au débit descendant, mais aussi les délais et pertes rencontrés. Ce besoin antagoniste d'exploiter au mieux la bande passante disponible pour assurer la qualité de la vidéo sans jamais surcharger le réseau afin de ne pas engendrer de délais ou de pertes doit être finement géré par l'algorithme de contrôle de congestion (CCA) de la plateforme. De surcroît, il est primordial que les plateformes prennent en compte les conditions dynamiques du réseau pour ne pas créer de congestion. Elles doivent alors s'adapter au plus juste, la diminution du débit se faisant au détriment du nombre d'images par seconde (FPS : *frames per second*), mais aussi de la résolution adoptée ou du niveau de compression de la vidéo.

2 Problématique

2.1 Défis

Le marché du Cloud Gaming était évalué à 3.37 milliards de dollars en 2022 [Ins23] et une forte croissance du secteur est attendue d'ici à l'horizon 2030. Les projections tablent sur une progression de 84.97 milliards de dollars. Ce vaste essor économique s'accompagnera inéluctablement d'une hausse du trafic CG dans la part du trafic Internet dont il pourrait devenir l'une des principales composantes en volume, comme l'est le streaming vidéo aujourd'hui. Il suffirait par exemple que certains fabricants optent pour un client léger pour leur prochaine génération de consoles. Il convient donc d'étudier dès aujourd'hui comment les réseaux peuvent véhiculer au mieux ce type de flux. En effet, le trafic CG a la particularité d'être à la fois très au débit sur le flux descendant et très sensible aux délais pour garantir une expérience réactive. Ces

particularités rendent son transport particulièrement délicat.

Nous nous sommes intéressés à l'état actuel des réseaux dans la section 1.1. L'idée dominante est que le très haut débit s'est généralisé et ne constitue plus aujourd'hui un facteur limitant. Le débit moyen ne suffit cependant pas à caractériser un réseau. Notons qu'un haut débit n'est pas inconciliable avec une forte latence. Des études ont d'ailleurs mis en avant le problème du *bufferbloat* [GN11]. Il découle d'un remplissage des files d'attente sur les équipements d'interconnexion, le plus souvent en périphérie du réseau où se situent les plus gros buffers. S'ensuit alors une hausse considérable des délais réseau, préjudiciables au trafic CG. Ceci constitue un premier problème. Un document publié par l'IETF en 2020 estimait que les algorithmes de contrôle de congestion (CCA) basés pertes étaient les plus fréquents [Mis20]. Ce type de CCA ne détecte la congestion qu'à partir du moment où des données sont *perdues*. Il faut donc attendre que les files soient pleines pour qu'ils réagissent. La hausse consécutive des délais porte grandement atteinte aux services "*temps réel*". Ce problème de cohabitation entre des CCA basés pertes et ceux basés délais au détriment des seconds constitue un second problème. D'un autre côté, les nouvelles technologies de programmabilité réseau rendent le réseau plus agile comme évoqué en section 1.2.

La problématique de cette thèse peut ainsi être formulée de la manière suivante :

Quelles solutions tirant partie de la programmabilité réseau mettre en œuvre afin d'améliorer la QoS du trafic de Cloud Gaming dans les réseaux ?

2.2 Problèmes traités

La résolution de cette problématique engendre des questions intermédiaires. Nous nous interrogeons tout d'abord sur la capacité des plateformes de Cloud Gaming actuelles d'adapter leur trafic à des conditions réseau difficiles tout en maîtrisant la latence. Étant donné le caractère "temps réel" des données produites et de l'impact de la réduction du débit sur la qualité de la vidéo, sont-elles capables d'adapter leur trafic au plus juste ? Y a t-il des différences de stratégie ou de capacité d'adaptation entre les différentes plateformes ? Quelles sont leurs limites dans ce domaine ? Ces questions trouveront des réponses dans la partie II de cette thèse.

Nous nous interrogeons ensuite sur la possibilité de reconnaître et traiter le trafic Cloud Gaming comme une classe de trafic homogène. Des méthodes d'apprentissage automatique sont-elles en mesure d'appréhender les caractéristiques du trafic Cloud Gaming par rapport à d'autres flux ? De plus, cette identification peut-elle être faite à la volée dans le réseau à des débits compatibles avec un déploiement dans un réseau d'opérateur ? Pour finir, nous voulons savoir si une ingénierie de trafic spécifique est capable d'améliorer la QoS du trafic CG, en permettant de maîtriser les délais induits et la cohabitation avec d'autres flux. Ces questions trouveront des réponses dans la partie III de cette thèse.

3 Organisation des contributions

Dans le cadre de cette thèse, **la caractérisation du trafic Cloud Gaming en environnement réseau contraint** est essentielle afin de bien appréhender ses spécificités et limitations notamment en termes d'adaptabilité du trafic aux conditions réseau. Ces connaissances sont ensuite mises à profit dans un second temps afin de permettre l'**identification du trafic Cloud Gaming dans le réseau puis l'optimisation de son transport**. Ces deux contributions constituent les parties principales de ce document, présentées respectivement dans les parties II et III. Elles sont précédées d'**un état de l'art** donnant les clés des technologies concernées qui est

présenté en partie I. Une conclusion générale, listant notamment des perspectives de recherches, clôture le présent manuscrit.

Partie I : État de l'Art

Dans le chapitre 1, nous commençons par étudier les technologies du Cloud Gaming à travers leur architecture fonctionnelle, les caractéristiques des principales plateformes commerciales et les aspects protocolaires sous-jacents. Nous nous intéressons ensuite à trois défis du Cloud Gaming liés à notre thématique, à savoir l'estimation de la QoS et QoE d'un tel service, les optimisations de l'infrastructure côté serveur, et surtout l'adaptation du trafic aux conditions réseau.

Le chapitre 2 s'intéresse ensuite aux différents mécanismes permettant de maîtriser la latence due au réseau. Nous étudions en particulier deux grandes catégories de solutions. D'une part les mécanismes de contrôle de congestion orientés délais et d'autre part les mécanismes de gestion de file d'attente, qu'ils soient actifs ou non.

Partie II : Caractérisation du trafic Cloud Gaming en environnement réseau contraint

Le chapitre 3 étudie le trafic réseau des quatre principales plateformes commerciales de Cloud Gaming sur un réseau à capacité fixe quand celui-ci est soumis à des contraintes impactant ses différentes caractéristiques (taux de perte, latence, gigue, bande passante). Nous analysons ainsi la capacité d'adaptation des différentes plateformes et leurs limites quand ses contraintes préexistent à une session de jeu ou apparaissent au cours de celle-ci.

Le chapitre 4 transpose cette étude synthétique dans un environnement réaliste en considérant des réseaux à capacité variable. Nous y émulons un réseau 4G selon des relevés réalisés pour l'occasion sur le réseau cellulaire d'Orange. Nous étudions notamment la stabilité du débit des plateformes et son impact sur la qualité vidéo, ainsi que leur capacité à contenir la latence due à la congestion.

Partie III : Transport optimisé du trafic Cloud Gaming

Dans le chapitre 5, nous proposons différents modèles d'apprentissage automatique, supervisés ou non, visant à identifier le trafic de Cloud Gaming parmi d'autres flux UDP à haut débit en se basant sur certaines caractéristiques statistiques des flux. Nous y décrivons la création du jeu de données utilisé pour l'entrainement et les tests. Nous évaluons la capacité des différents modèles à identifier le trafic en conditions réseau nominales ou dégradées, ainsi que leur capacité à identifier de nouvelles plateformes ou jeux non-appris.

Le chapitre 6 propose différentes stratégies exploitant les concepts et technologies de la programmabilité réseau pour implanter ces modèles. Nous proposons tout d'abord une architecture virtualisée à base de micro-services. Nous proposons ensuite de détecter le trafic Cloud Gaming directement dans le *data-plane* grâce au langage P4. Nous présentons finalement une architecture hybride, déchargeant seulement une partie les traitements les plus critiques dans le *data-plane*.

Enfin, nous proposons dans le chapitre 7 deux approches permettant de prioriser le trafic Cloud Gaming. Nous présentons tout d'abord notre plateforme expérimentale de Cloud Gaming nous permettant de maîtriser l'ensemble de la chaîne, du client au serveur. Nous évaluons ensuite l'algorithme de contrôle de congestion SCReAM, qui équipe cette plateforme, appliqué au transport du trafic CG. Nous évaluons enfin l'apport de la discipline de file d'attente HTB ou de l'AQM DualPI2 à la QoS du trafic CG lorsque ces technologies sont déployées dans le réseau. Nous considérons pour cela des flux TCP concurrents Cubic ou BBR.

Première partie

État de l'Art

Chapitre 1

Les technologies et défis du Cloud Gaming

Sommaire

1.1 Introduction 9
1.2 Contexte Technologique 10
1.2.1 Architecture fonctionnelle 10
1.2.2 Plateformes commerciales 11
1.2.3 Aspects protocolaires 12
1.3 Estimation de la QoS et de la QoE 14
1.4 Optimisation de l'infrastructure côté serveur 16
1.5 Adaptation du trafic aux conditions réseau 17
1.6 Conclusion 19

1.1 Introduction

Dans le cadre de cette thèse, nous nous intéressons à l'amélioration du transport des services à faible latence grâce aux nouvelles techniques de programmabilité réseau. Nous avons retenu le Cloud Gaming comme cas d'étude car il est intéressant à plus d'un titre. En effet, comme évoqué en introduction, le Cloud Gaming est tout d'abord un service dont le transport est particulièrement exigeant pour le réseau car il est à la fois haut débit, à faible latence, et génère les données à la volée. Il a de plus l'avantage d'être un service actuel, plusieurs offres commerciales étant d'ores et déjà disponibles et peuvent ainsi être étudiées et comparées. Enfin, il est promis à un bel avenir si l'on en croît certaines analyses prospectives, ce qui offre de bonnes perspectives d'impact en cas d'innovation dans ce domaine.

Ce premier chapitre de l'état de l'art est consacré au contexte technique et scientifique du Cloud Gaming. Nous présentons dans un premier temps en section 1.2 les principales technologies utilisées, puis nous nous intéresserons aux améliorations proposées dans la littérature scientifique. La section 1.3 traite ainsi de la qualité de service (QoS), la section 1.4 des optimisations d'architecture, et enfin la section 1.5 de l'adapation des plateformes CG aux conditions réseau.

1.2 Contexte Technologique

Dans la présente section, nous allons tout d'abord nous intéresser à l'architecture des plateformes de Cloud Gaming à travers l'exemple de la plateforme ouverte GamingAnywhere [HHCC13]. La majorité des autres plateformes de CG n'étant pas *open source*, nous ne pouvons pas décrire le cœur de leur architecture mais nous listons néanmoins certaines informations techniques intéressantes qui sont accessibles publiquement. Dans un second temps, nous porterons notre attention sur les protocoles réseau sous-jacents.

1.2.1 Architecture fonctionnelle

Huang & al. ont proposé en 2013 *GamingAnywhere* (GA), la première plateforme *open source* de Cloud Gaming [HHCC13]. GA fonctionne sous Linux et Windows pour la partie serveur, mais ses clients légers peuvent être exécutés sur d'autres systèmes d'exploitations (OS), comme Android ou iOS.

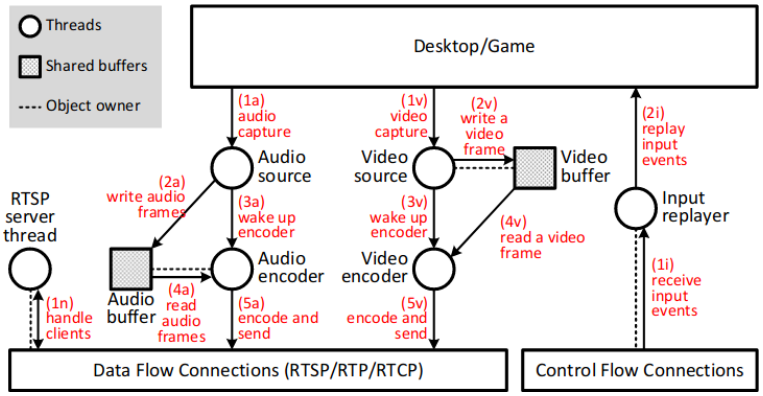


FIGURE 1.1 – Architecture de GamingAnywhere côté serveur [HHCC13]

Nous pouvons voir sur la Figure 1.1 l'architecture côté serveur. Plusieurs *threads* sont dédiés à des tâches spécifiques. "*Input Replayer*" est chargé de capturer les commandes du joueur ayant transité par le réseau. Il va ensuite les appliquer dans le moteur du jeu. Les threads "*Audio source*" et "*Video source*" capturent respectivement les trames audio et vidéo du jeu. Elles sont ensuite placées dans des buffers, en attendant d'être encodées. L'encodeur vidéo est x264, un encodeur *open source* permettant une compression au format H.264/MPEG-4 AVC.
La capture vidéo peut se faire de deux façons différentes :

— soit en faisant des captures d'écran selon une certaine cadence ;
— soit en interceptant un appel API émis dès que le jeu termine le rendu d'un écran.

Le client GA est quant à lui basé sur un client de bureau à distance (*Remote Desktop Protocol*). Il est constitué de deux threads. Le premier thread ("*Main thread*") a deux tâches en rapport avec les commandes utilisateur. La première tâche consiste à capturer les commandes entrées par le joueur, et la seconde à les transmettre au serveur. Le deuxième thread ("*Real Time Streaming Protocol (RTSP) client*") est chargé du démultiplexage et de la livraison frames audio et vidéo. Les trames audio sont placées dans un *buffer* et une fonction "*callback*" les récupère au fur et à mesure, selon une logique FIFO. En contraste, les frames vidéo ne sont pas bufferisées en tant que telles mais ce sont les paquets les constituant qui le sont. Par le biais d'un bit situé dans l'en-tête RTP ("*Marker bit*"), il est possible de savoir à quelle frame est associé un paquet[SCFJ03]. La

réception d'un paquet avec le "*Marker bit*" positionné à `1` déclenche le décodage d'une nouvelle frame à partir des paquets présents dans le *buffer*.

Quelques études menées par la suite ont cherché à minimiser les délais induits par les plateformes, en particulier au moment de l'encodage du flux vidéo. Dans [SHJ+14], Semsarzadeh & al. proposent un mécanisme qui permet d'accélérer l'encodage vidéo. Pour ce faire, ils se basent sur des informations du moteur de jeu pour bypasser quelques traitements. Notons qu'un encodeur doit estimer le "*Motion Vector*" (MV) de chaque macroblock composant une frame. La contribution consiste à localiser des objets dans une frame, et à appliquer le même MV à tous les macroblocks la composant. Inévitablement, le procédé s'accompagne d'une perte de qualité, qui reste néanmoins négligeable (<1db). Les auteurs affirment que l'encodage est accéléré de 8.86% grâce à leurs travaux. Dans [LPB+11, DAA10], les auteurs cherchent aussi à accélérer la phase de "*Motion Estimation*". Ils ne se basent par contre pas sur les informations du moteur de jeu. Dans [HJS+13, HSRS12, RSS11], il est question de supprimer des objets d'une scène de jeu pour réduire la complexité de l'encodage.

1.2.2 Plateformes commerciales

Les plateformes *OnLive*[4] et *Gaikai*[5] ont été les premières plateformes de CG à voir le jour au tout début des années 2010. De ce fait, la majorité des références scientifiques précédant l'écriture de ce manuscrit s'intéressent encore à cette première génération de plateformes. D'après [SLNC13], ces deux plateformes utilisent l'encodeur vidéo H.264. Sony a acquis Gaikai en 2012 et les brevets de OnLive en 2015. Les technologies mises en œuvre dans ces plateformes ont donc été intégrées aux services de Sony, notamment Playstation Now. Depuis le lancement de ces premières plateformes de CG, des progrès significatifs ont été faits par rapport à la rapidité et à la qualité du streaming. En particulier l'encodage et le décodage matériel des flux vidéos directement au sein des GPU, y compris grand public[6], via des circuits dédiés, a permis de gagner en temps d'encodage, et donc en latence. Ainsi l'architecture Kepler[7] de Nvidia proposait dès 2012 ces fonctionnalités.

Sur la période couverte par cette thèse, les principales plateformes de CG commercialisées en Europe contre un abonnement mensuel étaient Nvidia GeForce Now (GFN), Sony Playstation Now (PSN)[8], Google Stadia (STD)[9] et Microsoft Xbox Cloud Gaming (XC)[10]. Du fait que leur fonctionnement tombe sous le joug du secret industriel, nous ne pouvons pas les décrire aussi bien que GA. Néanmoins, quelques informations techniques intéressantes à relever sont disponibles à travers les communiqués de présentation de ces différents services.

Tout d'abord, GFN et STD exécutent une version PC des jeux sur des serveurs x86 dotés de GPU performants, GPU qui sont parfois virtualisés et partagés entre plusieurs utilisateurs. Ces versions PC peuvent offrir d'avantage d'options aux joueurs concernant les périphériques d'entrée, l'utilisation du couple clavier/souris étant possible, ou les choix d'effets graphiques. Cependant GFN et STD n'ont pas la même philosophie. GFN exécute une version standard des jeux depuis la bibliothèque de jeux dématérialisés possédée par l'utilisateur, alors que STD exécute une version des jeux spécifiquement optimisée pour la plateforme et devant être obligatoirement acquis sur

4. `https://en.wikipedia.org/wiki/OnLive`
5. `https://en.wikipedia.org/wiki/Gaikai`
6. `https://en.wikipedia.org/wiki/Nvidia_NVENC`
7. `https://en.wikipedia.org/wiki/Kepler_(microarchitecture)`
8. Intégré au service PlayStation Plus Premium en juin 2022
9. Stadia fut depuis fermé par Google en janvier 2023
10. Amazon Luna ne fut déployé partiellement en Europe (Allemagne et Royaume-Uni) qu' en Mars 2023

celle-ci [11]. On peut noter qu'une manette Stadia particulière bénéficie de sa propre connexion avec les serveurs Google. Cela permet de réduire le délai de transmission des commandes utilisateur.

A l'inverse, PSN et XC exécutent une version console des jeux sur un matériel similaire aux consoles de salon. Initialement, XC utilisait un matériel équivalent à une Xbox One S, et PSN à une PlayStation 3 (PS3) ou PlayStation 4 (selon le jeu) mais le matériel fut progressivement mis à jour à partir de 2021 pour tirer bénéfice de la nouvelle génération de consoles disponible alors, respectivement vers le matériel de la Xbox Series X et de la PS5. Ceci permet d'offrir de meilleures performances en matière de résolution et nombre d'images par seconde (fps), y compris pour les anciens jeux à travers la rétro-compatibilité. Ces deux plateformes offrent l'accès à un catalogue de jeux qui évolue au fil des mois.

Un autre aspect technique concerne les codecs, résolutions et framerates supportés. Toutes les plateformes utilisent H264. STD supporte en plus le codec VP9 [CB22] qui est même obligatoire pour profiter de la résolution 4K, et GFN supporte AV1 à partir d'octobre 2022 sur certains abonnements premium bénéficiant de la dernière génération de GPU d'architecture *Ada Lovelace*. STD et GFN fournissent principalement un flux vidéo de résolution 1080p à 60 fps. Une résolution supérieure jusqu'à 4K est possible, à partir de 2022 et avec un abonnement premium sur GFN et au détriment du framerate qui passe à 30 fps sur STD. Fin 2022, l'abonnement premium de GFN autorise même un framerate de 120 fps jusqu'à une résolution de 1440p. XC et STD furent limitées jusqu'en 2021 à une résolution de 720p et à 30fps. La mise à jour des plateformes a ensuite permis d'atteindre 1080p dans les jeux compatibles, et même 60 fps sur XC. Le tableau 1.1 résume ces différentes caractéristiques. À noter qu'il s'agit là des valeurs nominales des plateformes présupposant des conditions réseau satisfaisantes. Les plateformes peuvent en effet réduire leur résolution et/ou framerate pour s'adapter aux conditions réseau quand cela est nécessaire, mais nous détaillerons cela dans la partie II.

Plateforme	Résolution	framerate (fps)	Encodeur vidéo
GFN	1080p, 4K (2022)	60, 120 (2022)	H.264, AV1 (2022)
PSN	720p, 1080p 2021)	30	H.264
STD	1080p, 4K	60	H.264, VP9 (4K)
XC	720p, 1080p (2021)	30, 60 (2021)	H.264

TABLE 1.1 – Caractéristiques nominales des flux multimédia des plateformes de CG

Pour finir, nous pouvons mentionner d'autres logiciels offrant un service de jeux à distance tels que Steam Remote Play, Parsec, Moonlight, Rainway, etc. Ces logiciels reposent fondamentalement sur les mêmes technologies que les plateformes de CG mais ont été principalement conçus pour un usage domestique où l'utilisateur opère à la fois le client et le serveur au sein de son propre réseau local. Typiquement, un ordinateur fait office de serveur. Une télévision connectée ou un smartphone peuvent alors servir de client. La plupart de ces logiciels ont évolué pour autoriser une diffusion au delà du réseau local, sur Internet. Cependant cette dualité d'usages associée au manque d'infrastructure dédiée côté serveur les rend difficilement comparable à des plateformes intégrées de Cloud Gaming.

1.2.3 Aspects protocolaires

Nous nous concentrons ici sur les protocoles employés par les plateformes de CG. Chronologiquement, nous allons tout d'abord caractériser l'établissement d'une session de jeu. Nous

11. `https://en.wikipedia.org/wiki/Google_Stadia`

porterons ensuite notre attention sur les protocoles employés lors du streaming de la partie.

Phase d'initialisation

Dans le cas de *Gaming Anywhere* (GA) [HHCC13], l'utilisateur s'authentifie sur un serveur à part (*portal server*). Il y sélectionne un jeu, avant d'être redirigé vers un *game server*. Huang & al. précisent que les actions préalables à une session de jeu peuvent transiter via des requêtes REST sur HTTP ou HTTPS. En ce qui concerne OnLive et GFN, l'authentification se fait par le biais d'une connexion TLS/TCP [MUS+14, DDPT+21]. S'ensuit une évaluation des conditions du réseau. Dans le cas de OnLive (OL), le client commence par envoyer des paquets RTP/UDP vers les 4 datacentres de l'entreprise. Ceci lui permet de sélectionner le serveur de *probing* le plus proche, avec lequel il effectuera une deuxième session de mesures. Ces mesures semblent porter sur la latence et sur la bande passante descendante [MUS+14]. Ce fonctionnement a probablement été repris par PSN qui a été construit sur la base de OL. GFN réalise le même type de mesures, mais selon un mécanisme similaire à l'outil *Iperf*[12] dont il reprend le port 5001 sur UDP. En inspectant les messages *JSON* en transit, Domenico & al. soulignent que les tests de latence portent sur plusieurs serveurs. La plateforme utilise en effet des protocoles inhabituels, ce qui rend son fonctionnement opaque [DDPT+21]. En contrepartie, STD utilise des protocoles bel et bien documentés, avec l'usage de WebRTC et des protocoles associés. On retrouve par exemple le protocole DTLS, qui est une implantation de TLS sur UDP. Ce protocole permet notamment d'établir la session entre le client et le serveur. Dans [CB22], Carrascosa & al. rappelent que STD a recours au protocole *Interactive Connectivity Establishment* (ICE) pour faciliter les échanges P2P. ICE se base sur les protocles STUN (*Session Traversal Utilities for Nat*) et TURN (*Traversal Using Relay Nat*). Pour permettre une connexion directe entre deux *endpoints*, le protocole STUN est utilisé. Il permet à un pair de connaître son adresse IP publique et son port associé. Il suffit que deux pairs se partagent ces informations pour qu'une connexion directe puisse avoir lieu. Dans certains cas, il n'est pas possible d'établir une connexion directe, notamment si l'un des pairs a un "*symmetric NAT*". Il convient alors d'utiliser le protocole TURN, où un serveur fait l'intermédiaire entre les deux parties pour permettre l'inialisation des flux.

Session de jeu

Concernant les flux relatifs aux sessions de jeu à proprement parler, nous distinguons le *flux de données* du *flux de contrôle*. Le *flux de données* est chargé d'acheminer le contenu multimédia vers le joueur. De fait, il se fait dans le sens *serveur* → *client*. Le *flux de contrôle* a lieu quant à lui dans le sens opposé. Il permet de transmettre les actions du joueur vers le serveur. La plateforme GA [HHCC13] propose d'acheminer le *flux de données* de deux façons différentes. Cela peut tout d'abord être fait au-dessus du protocole TCP. Dans ce cas, la session de jeu repose sur un seul flux RTSP/TCP. Le Real Time Streaming Protocol (RTSP [13]) est un protocole niveau applicatif permettant de contrôler une session de flux multimédia quelque soit le protocole de transport sous-jacent. Parmi la profusion d'accronymes de protocoles réseau, il est important de ne pas le confondre avec le Real-time Transport Protocol (RTP [14]) qui est utilisé pour le transport des flux multimédia eux-mêmes au dessus d'UDP avec des contraintes temps réel, ainsi que son protocole compagnon, le RTP Control Protocol (RTCP) qui permet de superviser et contrôler une session

12. https ://iperf.fr
13. `https://www.rfc-editor.org/rfc/rfc7826`
14. `https://www.rfc-editor.org/rfc/rfc3550`

RTP. Tous trois sont standardisés par l'IETF. Dans le cas où c'est le protocole UDP qui est utilisé, on distingue trois flux réseau :

— deux flux RTP/UDP pour l'audio et la vidéo ;
— un flux RTSP/TCP pour les commandes.

La plateforme GFN [DDPT+21] a aussi recours à plusieurs canaux UDP. Chaque canal utilise un port dédié à un média spécifique ou aux commandes du joueur. Domenico & al. soulignent que bien que le protocole RTP soit utilisé, on n'observe pas de paquets RTCP. Notons que les paquets RTCP servent à monitorer la livraison des données. Ils transitent dans le sens *client* → *serveur* [SCFJ03]. Contrairement à GFN, plusieurs plateformes fonctionnent avec un seul flux UDP. On parle alors de multiplexage. Dans le cas du protocole RTP, un champ SSRC (*Synchronization source*) est prévu dans l'en-tête des paquets. Ce champ permet au récepteur de distinguer la nature des données qu'il reçoit [SCFJ03]. La plateforme OL utilise d'ailleurs 7 SSRCs dans le sens descendant et 4 SSRCs dans le sens montant [MUS+14]. Un des flux RTP descendant est destiné à traquer la QoS tout au long de la session de jeu. Les plateformes STD et XC se basent aussi sur le protocole RTP [DDPT+21, CB22] pour le transport des flux. Au même titre que OL, elles ont recours au multiplexage. [CB22] remarque que pour STD, le flux RTP du serveur vers le client est composé de plusieurs groupes de paquets apparaissant toutes les 16,67ms, ce qui correspond à un framerate de 60fps. Les groupes sont dus à la fragmentation réseau d'un grand segment applicatif correspondant à une image. Notons que STD et la version navigateur (*Beta*) de XC réalisent leur service grâce à l'API WebRTC, qui est standardisée par le W3C et l'IETF pour le langage JavaScript et supportée par tous les navigateurs web modernes. Pour finir, PSN utilise une approche entièrement personnalisée. Domenico & al. notent [DDPT+21] qu'un seul flux UDP est utilisé lors de la phase de *streaming*. Les auteurs parviennent tout de même à comprendre le mécanisme de multiplexage utilisé, le premier octet de chaque paquet permettant de distinguer les différents canaux. Une analyse plus poussée leur permet ainsi d'associer chaque canal à un flux de données spécifique.

Plateforme	Testing	Flux multimédia	Commandes
GFN	UDP	RTP	UDP
PSN	UDP	UDP	UDP
STD	RTP	RTP	DTLS

Table 1.2 – Protocoles de transports utilisés

1.3 Estimation de la QoS et de la QoE

La QoE (Quality of Experience) est une mesure permettant d'estimer la qualité d'une expérience de jeu du point de vue de l'utilisateur. C'est donc une mesure subjective, car elle dépend de la perception de chaque joueur. Plus elle est élevée, et plus l'expérience de jeu est jugée satisfaisante. Notons que des facteurs concrets viennent directement l'impacter. En effet, un joueur pourrait être dérangé par une faible qualité graphique, un temps de réponse long, etc. Ces grandeurs mesurables constituent la QoS (Quality of Service). QoE et QoS sont ainsi intimement liées. Ici, nous nous intéressons à ces deux métriques dont la mesure et l'évaluation sont des enjeux importants pour ce type de service, ce qui a donné lieu à plusieurs études.

Les articles [CCH+14] et [SLNC13] visent à estimer les sous-délais constituant le temps de réponse. Cette durée correspond au laps de temps séparant l'entrée d'une commande de

jeu du moment où son résultat s'affiche sur l'écran du joueur. [SLNC13] définit globalement le "*Cloud Overhead*". Ce délai est le temps induit par la plateforme de Cloud Gaming (*encodage et streaming*). Les auteurs soulignent que OnLive rajoute entre 100 et 120ms de temps de réponse et concluent que des progrès doivent être principalement faits au niveau des encodeurs vidéo.

[CCH+14] décompose ce délai global en 4 sous-délais :

1. Le délai réseau (RTT = Round Trip Time) ;
2. Le "*processing delay*" : réception des commandes de jeu et encodage des frames ;
3. Le "*game delay*" : application de la logique du jeu, génération d'une "*game frame*" ;
4. Le "*playout delay*" : temps mis pour décoder une frame et l'afficher à l'écran ;

Parmi ces délais, seul le *game delay* n'est pas spécifique au cloud-gaming. Au moment de l'étude, le "*processing delay*" représente la principale latence ajoutée par une plateforme de Cloud Gaming mesurée à plus de 100ms sur OnLive et StreamMyGame ce qui est prohibitif pour certains types de jeux. Ces résultats sont cependant à contextualiser car les progrès réalisés depuis sur le support matériel des codecs, en particulier l'encodage des trames vidéo par GPU, ont permis de réduire significativement cette valeur aujourd'hui.

Plusieurs études se sont intéressées à l'impact de la latence sur l'expérience de jeu. Dans [REG14], les auteurs demandent à des participants d'ajuster le délai entre une commande de jeu et sa représentation visuelle à l'écran (*temps de réponse*). Le but est d'avoir une impression d'instantanéité. En conclusion, plus de la moitié des participants a du mal à tolérer des délais dépassant les 100ms. En fonction du type de jeu, la latence aura cependant plus ou moins d'impact. Dans [YCHL12], les auteurs cherchent justement à évaluer la sensibilité d'un jeu à la latence. On parle de RS (Real-Time Strictness). Un modèle de régression permettant d'estimer cette grandeur y est présenté. Il se base sur la notion de "*command heaviness*", soit le taux de changements à l'écran pour chaque commande ($Pixel.Command^{-1}$). En plus de l'élaboration de leur modèle, les chercheurs montrent que la sensibilité à la latence dépend du type de jeu. Enfin les auteurs de [NGJ14] étudient l'impact de la latence et de la gigue sur la qualité d'expérience d'un joueur contrôlant un avatar. Ils montrent en particulier que les joueurs arrivent à s'accoutumer à une certaine latence constante mais que la gigue leur pose beaucoup plus de problème.

Dans [SBSK+14], les auteurs s'intéressent aux spécificités de différents types de jeu. Pour ce faire, ils utilisent des métriques portant sur la vidéo et sur le réseau. Les métriques portant sur la vidéo visent à mesurer leur aspect temporel (*mouvement*) et spatial (*complexité scénique*). Les auteurs ont montré que les jeux de type "*action*" ou "*shooter*" ont les métriques temporelles les plus importantes. Cela se traduit par une certaine dynamique de jeu. À l'inverse, ce sont les jeux du type "*strategy*" ou "*role-playing*" qui ont les métriques spatiales les plus importantes. Les auteurs ne sont toutefois pas parvenus à établir une relation entre métriques vidéo et métriques réseau.

Quelques études ont cherché à mettre en avant l'impact des facteurs QoS sur la QoE. Dans [CHL09], les auteurs étudient une plateforme commerciale de MMORPG (Massive Multiplayer Online Role Playing Game). La QoE est estimée en considérant la probabilité qu'un joueur arrête prématurément sa partie. Un modèle de régression logistique est alors employé. Son vecteur de coefficients permet d'estimer l'intolérance des joueurs face à chaque facteur de QoS considéré. Les auteurs nous donnent alors les niveaux d'intolérance :

— 10% : délais réseau ;

— 20% : jigue ;
— 30% : perte de paquets du serveur ;
— 40% : perte de paquets du client ;

La perte de paquets *client* → *serveur* serait donc le facteur impactant le plus la QoE. Il l'impacterait 4 fois plus que les délais réseau. Ces résultats sont toutefois à nuancer. D'une part, il s'agit de jeux en ligne et non de cloud-gaming. La nature des flux, notamment du serveur au client diffère totalement. D'autre part, l'utilisation de TCP par la plateforme fait que la retransmission d'un paquet égaré peut accroître les délais. Les auteur proposent pour finir de traiter prioritairement les sessions de jeu dont la probabilité de départ serait élevée.

Dans [JSSH11], Jarschel & al. prennent en compte plusieurs scénarios. Tour à tour, ils ajoutent de la latence et engendrent des pertes de paquets. Ils varient l'importance de ces deux grandeurs et cherchent ensuite à les composer. Les auteurs s'intéressent de plus à l'impact de la direction dans laquelle les perturbations sont appliquées. Trois différentes perspectives de jeux sont étudiées :

1. "*Omnipresent perspective*", pour des jeux qualifiés de "*slow-paced games*" ;
2. "*Third person perspective*", pour des jeux qualifiés de "*medium-paced games*" ;
3. "*First person perspective*", pour des jeux qualifiés de "*fast-paced games*" ;

Ici, la QoE consiste en une note entre 1 et 5 saisie par l'utilisateur (MOS : Mean Opinion Score). Suite à leur expérience, les auteurs remarquent que les "*slow-paced games*" sont moins sensibles à la latence que les autres types de jeux. Ils sont néanmoins sensibles aux pertes, du fait des dégradations engendrées sur la qualité vidéo. Une tendance inverse est constatée pour les "*fast-paced games*" qui sont plus sensibles à la latence et moins aux pertes. Pour finir, les auteurs démontrent que l'application de contraintes de délais et de pertes dans le sens *serveur* → *client* impacte bien plus la QoE que dans le sens opposé.

Une approche toute autre pour estimer la QoE est appliquée dans [SBB+20]. Ici, les auteurs partent du principe que la QoE est corrélée au score des joueurs. Ainsi, Sviridov & al. mettent au point des agents IA chargés d'évaluer la qualité d'une session de jeu. Dans le cas où les scores sont élevés, une bonne QoE est présumée. L'entraînement de ces intelligences artificielles se base sur la méthode "*trial and error*". Les agents s'améliorent ainsi au fur et à mesure des itérations. À chaque étape, les agents prennent comme entrée une trame vidéo (s_t). Une action a_t est alors déclenchée. En découleront une nouvel état s_{t+1} ainsi qu'une récompense r_t. Le but est de maximiser les récompenses. Une fois les agents entraînés, l'impact des conditions réseau sur leurs performances de jeu est étudié. Les auteurs dressent trois conclusions :

1. Tous les jeux ne sont pas impactés de la même façon ;
2. Au-delà de 100ms de latence, il n'est plus possible de jouer ;
3. La latence est plus impactante que la perte de paquets *client* → *serveur* ;

Les auteurs expliquent ce dernier point de par la répétition instinctive d'actions côté joueur. Ceci constitue une FEC (Forward Error Correction) implicite. Une application du présent travail est ensuite considérée. Eu égard de la sensibilité de chaque jeu face à la latence, il est possible de mettre au point un ordonnanceur centré sur la QoE. Dans le cas de sessions de jeu concurrentes, l'idée est de prioriser les jeux les plus sensibles à la latence. Ceci permet de répartir plus équitablement la QoE.

1.4 Optimisation de l'infrastructure côté serveur

Nous avons vu dans la section précédente que la QoS a un impact certain sur la QoE. Nous nous intéressons ici à des choix qui peuvent être faits avant même de lancer une session de jeu. Ces choix ont une incidence sur la QoE, mais aussi sur les coûts incombant aux plateformes de Cloud Gaming. Des techniques de recherche opérationnelle permettent de mettre en lumière le choix le plus approprié pour satisfaire une certaine fonction objectif.

Dans [FCJ+14, HCH+13, LTC15], il est question de placement. L'article [FCJ+14] s'intéresse à comment placer des jeux sur des groupes de serveurs. Pour ce faire, les auteurs modélisent la demande utilisateur, ainsi que la distribution des jeux. L'objectif est de minimiser les "*miss*" et l'usage de la mémoire. Un "*miss*" se traduit par un utilisateur ne pouvant pas jouer au jeu qu'il souhaite. Les auteurs montrent qu'avec un algorithme de type "*hill-climbing*", on atteint des performances optimales.
Dans [HCH+13], on cherche cette fois à placer des machines virtuelles (VMs) sur des serveurs. Chaque VM est chargée d'exécuter un jeu. Les auteurs présentent deux heuristiques visant à favoriser deux fonctions objectif distinctes. La première fonction objectif, ou "*provider-centric*", cherche à maximiser les profits tout en satisfaisant une QoE minimale. La deuxième fonction objectif cherche à maximiser la QoE tout en satisfaisant les contraintes de ressources. Les auteurs montrent que leurs heuristiques sont quasi-optimales.
Enfin, [LTC15] s'intéresse au placement des joueurs. Les auteurs se basent sur des réseaux neuronaux pour prédire la durée d'une session de jeu. Forts de cette information, ils vont placer sur une même VM les sessions de jeu devant se terminer à peu près au même moment. Ce procédé permet de minimiser les coûts de fonctionnement des plateformes de Cloud Gaming.

Les articles [MFD12, TWH+15] portent sur l'allocation des ressources. L'article [TWH+15] aide à définir le datacentre, le type de VM, et enfin le *bitrate* associés à un joueur. Les auteurs se basent sur divers groupes d'utilisateurs. À chaque groupe est associé un bitrate minimal, qu'il convient de satisfaire. L'algorithme proposé vise à satisfaire cette contrainte tout en minimisant les coûts de fonctionnement. Il s'agit d'un problème d'optimisation stochastique, sur lequel les auteurs appliquent la théorie d'optimisation de Lyapunov. Ils soulignent que leur solution peut baisser les coûts de fonctionnement de 25%.
Dans un contexte un peu différent d'une infrastructure de serveurs de jeu en ligne, les auteurs de [MFD12] visent à garantir un temps de réponse inférieur à un certain seuil tout en minimisant le nombre de nœuds associés à chacun des trois paliers de l'architecture définis comme suit :

1. Gateway (*vérification et contrôle du protocole de jeu*) ;
2. Cell (*gérer le monde virtuel et son évolution*) ;
3. DataBase (*stocker les informations persistantes*) ;

Les ressources sont allouées dès que le temps de réponse dépasse un certain seuil R_M ou libérées si le temps de réponse est inférieur à un seuil R_m. Un problème d'optimisation permet de choisir le nombre et le placement des nœuds à déployer.

1.5 Adaptation du trafic aux conditions réseau

La QoE peut être mise à mal du fait des conditions du réseau. Nous avons notamment vu qu'elle peut être négativement impactée de par l'ajout de latence et la perte de paquets. Les plateformes de Cloud Gaming doivent donc s'adapter aux performances du réseau à un instant

donné. Dans un réseau congestionné, il est courant qu'une application abaisse son débit de données (*bitrate*) pour limiter sa congestion. C'est le rôle des algorithmes de contrôle de congestion (CCA). Cette adaptation est cependant délicate pour des services produisant des données à la volée. Il leur est en effet impossible de puiser comme bon leur semble dans un *buffer* d'émission. Les marges de manœuvre pour adapter le débit son ainsi plus réduites et se limitent à compresser davantage le flux vidéo, en réduire la définition ou en réduire le nombre d'images par seconde. De plus, l'encodeur a besoin d'un certain temps pour pouvoir les prendre en compte. D'autres mécanismes permettent une certaine résilience vis-à-vis des perturbations occasionnées par le réseau. Quand des contraintes de temps empêchent une retransmission des données, il est courant d'y adjoindre des FEC (Forward Error Correction), données redondantes qui peuvent composer la perte de paquets dans une certaine limite. Dans un autre registre, le recours à des techniques de prédiction temporelle ou spatiale au niveau applicatif peut limiter l'impact de la latence réseau sur la QoE en donnant l'illusion de la réactivité [LCC+15].

Les papiers [WD10b, WD10a, HHT+15] s'occupent de l'adaptation des paramètres de rendu aux conditions du réseau. L'article [WD10b] prend en compte deux coûts ; les coûts de communication (*bitrate*) et les coûts de calcul (*utilisation du GPU*). Les auteurs identifient ensuite 4 paramètres de rendu qui ont un impact sur les coûts en question. Étant donné un ensemble de configurations, ils associent à chaque élément (*4-tuple*) les coûts lui étant associés. Dès lors qu'une contrainte réseau se fait ressentir (*perte ou délais*), il convient de modifier la configuration actuelle de sorte à diminuer les coûts de communication. Dans le cas où le GPU serait sur-chargé, une configuration diminuant cette fois les coûts de calcul est adoptée.
Les auteurs de [WD10a, HHT+15] se basent sur des estimations de QoE pour faire des choix de configuration. Pour ce faire, [WD10a] se donne plusieurs configurations possibles (*ensemble de couples*). Un modèle d'estimation de la QoE permet aux auteurs d'estimer l'impact qu'a chaque couple (*framerate et bitrate*) sur la QoE (*impairment*). Se donnant un certain bitrate, les auteurs sont ainsi en mesure de choisir le framerate impactant le moins possible l'expérience de jeu. Le choix du bitrate repose quant à lui sur les délais et les pertes du réseau. Dès lors que le délai descendant ou que le taux de pertes dépassent un certain seuil, le bitrate est diminué d'un cran. Dans le cas contraire, il est augmenté. Dans [HHT+15], les auteurs ont recours à une estimation de la QoE reposant sur trois paramètres ; le bitrate, le framerate et le jeu. Ils utilisent un modèle quadratique donné par l'équation 1.1 ;

$$m(g, f, b) = \alpha_{g,1} f + \alpha_{g,2} b + \alpha_{g,3} f^2 + \alpha_{g,4} b^2 + \alpha_{g,5} fb + \alpha_{g,6} \tag{1.1}$$

Ici, 'g' désigne un jeu, 'f' désigne un framerate et 'b' un certain bitrate. Du fait que l'on veut maximiser la QoE, les dérivées doivent valoir 0, et ainsi $\frac{\partial m}{\partial f} = 0$. Les auteurs sont alors en mesure d'exprimer le framerate en fonction du jeu et du bitrate. Ici, deux algorithmes efficients visant à choisir le bitrate sont proposés. Chacun correspond à une fonction objectif différente (*que l'on souhaite maximiser la QoE moyenne ou bien la QoE minimale*).

Les articles [YXLL14, WYC+15] traitent des mécanismes de FEC et des conditions du réseau. Dans [YXLL14], les auteurs font une campagne de mesures sur des plateformes de visio conférence. Une des plateformes étudiées augmente la redondance des données dès lors qu'une perte de paquets est détectée. Nonobstant, l'ajout de paquets dédiés à la FEC accentue d'autant plus le taux de pertes. On rentre ainsi dans une sorte de cercle vicieux, qui se fait au détriment de la QoE. Le papier [WYC+15] permet justement d'adapter l'encodage des FEC. Les auteurs cherchent un compromis entre résilience aux pertes et délais induits par la redondance des données. De plus, du fait de l'importance des I frames dans la reconstitution d'une image, les I-frames

et les P-frames sont traitées différemment (*unequal FEC coding*).

L'adaptation des premières plateformes de CG aux conditions du réseau a été étudiée dans [CCH+14] et [CFGS12], tandis que [CB22] et [SSSK16] se sont respectivement concentrés sur STD et GFN. Chen & al. [CCH+14] ont évalué la qualité du streaming dépendamment de la bande passante, des délais et du taux de pertes. Ils ont montré que OL et SMG gèrent bien les délais. SMG est cependant *confuse* face aux pertes ou à des contraintes de bande passante. De même, dans [CFGS12], les auteurs ont prouvé que la fréquence d'images d'OL est diminuée lorsque les pertes sont trop élevées ou que la bande passante disponible est trop faible (*de 60 à 25 fps*). Ils soulignent également que le débit binaire ne semble pas être affecté par la perte de paquets ni par la latence. Les auteurs de [CB22] rapportent que STD semble diminuer sa résolution dès lors qu'une contrainte réseau est appliquée. La plateforme entre alors dans une phase de transition, où elle tente d'ajuster ses paramètres au réseau. Cette phase entraîne une faible QoE, car la résolution et la fréquence d'images fluctuent considérablement. Selon [DDPT+21], la résolution 720p a un débit binaire moyen de 11 Mbps, contre 29 Mbps pour 1080p et 44 Mbps pour 2160p. Dans [SSSK16], le trafic de GFN est perturbé par l'ajout de latence, de gigue et de perte. Ils ont noté que GFN privilégie la fréquence d'images par rapport à la résolution lorsque les ressources du réseau viennent à manquer. Ce choix garantit la fluidité du streaming au détriment de la qualité.

1.6 Conclusion

En conclusion, nous avons tout d'abord introduit le contexte technologique du CG. La présentation d'une plateforme *open source* a ainsi été réalisée. Nous nous sommes ensuite intéressés aux plateformes commerciales. Une attention particulière a été accordée aux aspects protocolaires. Pour continuer, nous couvrons plusieurs axes de recherche associés au CG. Nous y présentons des études sur l'estimation de la QoS/QoE, sur l'optimisation de l'infrastructure et sur l'adaptation du trafic aux conditions réseau.

Il ressort de ces études que les plateformes adoptent des architectures fonctionnelles similaires mais se différenciant quant au matériel ou aux protocoles utilisés. De nombreuses études montrent que, bien que plus ou moins marquée selon le type de jeu, la latence reste un facteur primordial pour la QoE d'un service de Cloud Gaming. La latence due à l'encodage / décodage des flux vidéo est aujourd'hui bien maîtrisée. Les délais inhérents au réseau constituent quant à eux un enjeu majeur du fait de leur imprévisibilité. La latence du serveur vers le client est particulièrement impactante, et dépend de la congestion sous-jacente. Le flux multimédia transporté peut de surcroît contribuer à l'engorgement du lien. Il est donc compréhensible que les plateformes cherchent à adapter leur fonctionnement aux conditions réseau. Aucune évaluation n'a pour le moment comparé l'adaptation de plateformes face à des conditions réseau différentes. Nous proposons de mener cette étude dans la partie II.

Nous verrons que l'enjeu est de maîtriser les délais induits par le réseau. Pour ce faire, il convient de maîtriser les files d'attente se constituant au niveau du goulot d'étranglement. Il s'agit d'un équipement d'interconnexion, situé en périphérie du réseau. Le prochain chapitre s'intéresse donc aux techniques de contrôle de congestion et de gestion de files d'attente.

Chapitre 2

Mécanismes permettant de maîtriser la latence due au réseau

Sommaire

2.1 Introduction 21
2.2 Mécanismes de contrôle de congestion orientés latence 22
2.2.1 BBR 22
2.2.2 GCC 22
2.2.3 C3G 25
2.2.4 Sprout 26
2.2.5 SCReAM 27
2.2.6 Problématique de la cohabitation entre CCAs 29
2.3 Mécanismes de gestion de file d'attente 30
2.3.1 RED 31
2.3.2 Codel 31
2.3.3 PIE 32
2.3.4 HTB 32
2.3.5 L4S 34
2.4 Conclusion 38

2.1 Introduction

Le présent chapitre s'intéresse aux mécanismes permettant de maîtriser les délais réseau d'un flux. Des algorithmes de contrôle de congestion ou des politiques de gestion de file d'attente sont alors mis à l'oeuvre. Un réseau congestionné induit en effet des délais supplémentaires au fur et à mesure que les files d'attente des équipements d'interconnexion se remplissent. Ces délais se soldent par des pertes de paquets dès lors que les files d'attente sont pleines. Délais et pertes peuvent fortement impacter certaines classes de trafic, en particulier celles relatives à des applications avec des contraintes de temps qui empêchent les retransmissions. Il convient donc de les limiter au maximum, tout en maximisant l'utilisation de la bande passante disponible. Nous présenterons ici deux types de mécanismes ; (i) les algorithmes de contrôle de congestion (CCA) et (ii) la gestion active de files d'attente (AQM).

Les CCAs sont des mécanismes de bout-à-bout. Ils sont situés aux extrémités du réseau, sur les équipements terminaux, leur permettant d'adapter dynamiquement leur débit à la capacité

du réseau à un instant donné. Un débit démesuré par rapport à la capacité du lien entraîne le remplissage des *buffers* intermédiaires. Divers *signaux de congestion* peuvent être utilisés pour détecter la congestion. Les mécanismes présentés ici tiennent compte de l'évolution des délais. Ce type de signal permet de détecter la congestion de façon anticipée. Les CCAs se basant sur les pertes de paquets réagissent bien plus tard.

Les AQMs se situent quant à eux sur le réseau. La supervision des files d'attente leur permet de détecter la congestion prématurément. Certains monitorent la taille des files, tandis que d'autres se basent sur le "*packet sojourn time*". Cette grandeur représente le temps d'attente des paquets. Une fois la congestion détectée, un signal de congestion peut être émis vers les terminaux. Ce signal peut être de plusieurs natures. Le but est que les CCAs régissant leur comportement en tiennent compte pour adapter le débit.

2.2 Mécanismes de contrôle de congestion orientés latence

2.2.1 BBR

L'algorithme de contrôle de congestion "Bottleneck Bandwidth and Round-trip propagation time" (BBR) a été développé par Google en 2016 pour TCP [CCG+16]. Le but est d'utiliser au maximum les ressources du réseau tout en minimisant les délais induits. Comme toujours, il revient à l'émetteur d'adapter son débit. Le débit optimal à adopter est appelé l'OOP : ***O**ptimal **O**perating **P**oint*. Pour le calculer, les auteurs estiment deux variables :

— $BtlBW$: la bande passante du *bottleneck* ;
— $RTprop$: le RTT (**R**ound **T**rip **T**ime) ;

Avec TCP, l'émetteur reçoit des acquittements (ACK) par rapport aux données qu'il a envoyées. La réception d'un ACK donne deux informations à l'émetteur ; le RTT du paquet, et la confirmation que des données envoyées ont bien été reçues. Fort de ces informations, l'émetteur est en mesure de calculer le débit moyen de livraison. Cette même valeur est bornée par $BtlBW$, car le récepteur ne peut pas avoir un débit meilleur que ce qui est permis par le réseau. L'estimation de $BtlBW$ est donc obtenue en faisant le maximum de débits de livraison estimés sur une fenêtre temporelle W_B.

$$\hat{BtlBW} = max(deliveryRate_t) \forall t \in [T - W_B; T]$$

L'estimation de $RTprop$, quant à elle, consiste à faire le minimum des RTTs obtenus sur une fenêtre temporelle W_R. Ceci fait, il est de rigueur de calculer le BDP (**B**andwidth **D**elay **P**roduct) en multipliant $BltBW$ par $RTprop$. Le principe de BBR est que lorsque le volume des données en vol (*données envoyées mais non acquittées*) est égal au BDP, alors on a trouvé le débit optimal OOP. L'algorithme a recours à une *exploration périodique* de sorte à traquer la bande passante disponible. Dans l'idée, l'émetteur augmente régulièrement son débit de 25% pour tester les conditions du réseau. On parle de mécanisme MIMD (**M**ultiplicative **I**ncrease **M**ultiplicative **D**ecrease).

Comme vu dans le chapitre précédent, les plateformes de cloud-gaming n'utilisent pas TCP pour transporter leurs données. BBR n'est donc pas directement applicable à notre cas d'étude.

2.2.2 GCC

Google propose un autre CCA pour UDP, appelé "Google Congestion Control" (GCC) [CDCHM16]. Il a été mis au point pour le trafic ayant une contrainte "*temps réel*", comme les visio-conférences.

L'introduction du Draft IETF de GCC [HLC+16] publié dans le cadre du groupe de travail "RTP Media Congestion Avoidance Techniques" (RMCAT) explique bien la difficulté de concevoir un CCA pour les communication "temps-réel", dont fait partie le cloud gaming.

"Congestion control for real-time media is challenging for a number of reasons :
- *The media is usually encoded in forms that cannot be quickly changed to accommodate varying bandwidth, and bandwidth requirements can often be changed only in discrete, rather large steps*
- *The participants may have certain specific wishes on how to respond - which may not be reducing the bandwidth required by the flow on which congestion is discovered*
- *The encodings are usually sensitive to packet loss, while the real-time requirement precludes the repair of packet loss by retransmission"*

L'algorithmique de GCC se base sur deux contrôleurs. Chaque contrôleur est chargé de calculer le débit optimal à adopter. Comme toujours, le but consiste à limiter la latence tout en profitant un maximum de la bande passante disponible. Nous retrouvons :
- un contrôleur basé délais côté récepteur. Il est chargé de calculer le débit A_r ;
- un contrôleur basé pertes côté émetteur. Il est chargé de calculer le débit A_s.

Le débit final est choisi en faisant le minimum des deux débits. Le tout repose sur les protocoles de transport RTP/RTCP, eux-mêmes étant sur UDP. Les messages RTCP permettent à l'émetteur de connaître le pourcentage de paquets perdus. Cette donnée est utile à son contrôleur basé pertes. De surcroît, le récepteur notifie l'émetteur de son débit A_r via des messages "REMB", une extension de RTCP. Nous pouvons voir sur la figure 2.1 une vue d'ensemble de l'architecture.

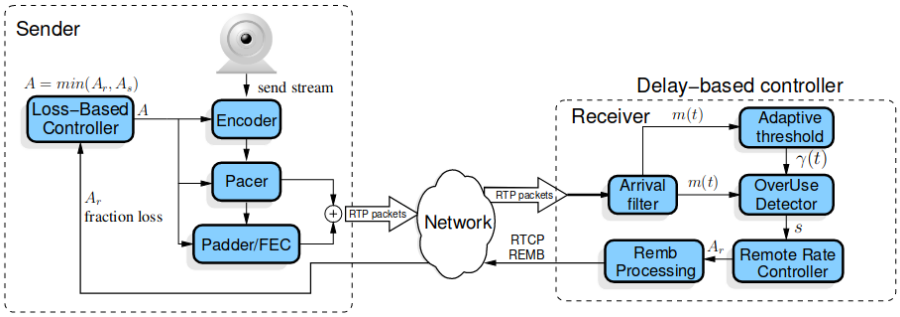


FIGURE 2.1 – Architecture de GCC [HLC+16]

Contrôleur basé délais

Le contrôleur basé délais se base sur le "*queuing delay gradient*", ou la dérivée du temps passé en file d'attente. Une valeur positive signifie que les délais augmentent, alors qu'une valeur négative se traduit par un drainage de la file. Si le "*queuing delay gradient*" est nul, il y a trois possibilités :
- soit la file est vide, et donc le taux de remplissage est inférieur à la capacité du lien ;
- soit la file est pleine, et donc les nouveaux paquets sont immédiatement *droppés* ;
- soit la taille de la file est constante ; le taux de remplissage est égal à la capacité du lien.

GCC étant adapté à RTP, il travaille au niveau des *frames*. Ils définissent le "*one-way delay*" comme le laps de temps entre l'envoi du premier paquet d'une *frame* (*instant* T_i) et la réception de son dernier paquet (*instant* t_i). Le "*one way delay gradient*" à l'instant t_i est la différence entre le "*one-way delay*" des *frames* 'i' et 'i-1'. La formule est donnée par l'équation 2.1.

$$d_m(t_i) = (t_i - T_i) - (t_{i-1} - T_{i-1}) \tag{2.1}$$

Du fait que les valeurs mesurées peuvent être impactées par du *bruit*, un filtre de Kalman est appliqué sur $d_m(t_i)$. On obtient en sortie $m(t_i)$, une estimation du "*queuing delay gradient*" dépourvue de bruit. Un seuil adaptatif $\gamma(t_i)$ est ensuite appliqué sur $m(t_i)$. Il permet d'adapter la sensibilité de l'algorithme aux variations des délais. Le choix du seuil est fixé par l'équation 2.2. Le but est de suivre l'évolution de $m(t)$.

$$\gamma(t_i) = \gamma(t_{i-1}) + (t_i - t_{i-1}) * k_\gamma(t_i) * (|m(t_i)| - \gamma(t_{i-1})) \tag{2.2}$$

Si $|m(t_i)|$ est inférieur au précédent seuil $\gamma(t_{i-1})$, il convient d'abaisser $\gamma(t_i)$. Cet amenuisement se fait à une vitesse k_d, alors donnée par $k_\gamma(t_i)$. Dans le cas contraire, $\gamma(t_i)$ croît à une vitesse k_u. La nouvelle valeur de $\gamma(t_i)$ sert à générer un signal. Le signal en question permet de définir un état, et au final de choisir le nouveau débit A_r.

— Si $m(t_i) > \gamma(t_i)$: générer le signal "`overuse`" si ça a duré plus de 100ms.
— Si $m(t_i) < -\gamma(t_i)$: générer le signal "`underuse`". Sous-utilisation de la bande passante.
— Si $-\gamma(t_i) < m(t_i) < \gamma(t_i)$: générer le signal "`normal`". Bonne utilisation des ressources.

Nous pouvons voir sur la figure 2.2 une représentation de la machine à états. Les paramètres α et η sont à fixer. R_r dénote le débit reçu les 500 dernières millisecondes.

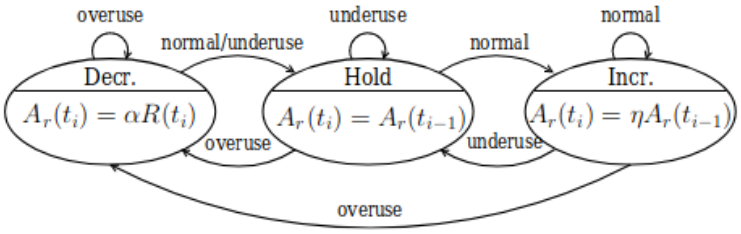


FIGURE 2.2 – Machine à états de GCC [CDCHM16]

Contrôleur basé pertes

Le contrôleur basé pertes est quant à lui bien plus simple. Dans l'idée, lorsque le taux de pertes $fl(t_k)$ est faible, A_s est laissé tel quel. Dans le cas où il est élevé, A_s est diminué multiplicativement. Par symétrie, il est augmenté multiplicativement dès lors que les pertes sont négligeables.

$$A_s(t_k) = \begin{cases} A_s(t_{k-1})(1 - 0.5 f_l(t_k)) & \text{si } f_l(t_k) > 0.1 \\ 1.05 A_s(t_{k-1}) & \text{si } f_l(t_k) < 0.02 \\ A_s(t_{k-1}) & \text{sinon} \end{cases}$$

Par conception, GCC est plus adapté au trafic CG que BBR. Le trafic CG a néanmoins des exigences allant au-delà de la visio-conférence, cas d'application typique de GCC. On peut évoquer une plus grande sensibilité aux délais et un débit bien plus large.

2.2.3 C3G

Justement, dans [AMC+19], les auteurs cherchent à adapter BBR [CCG+16] au protocole UDP dans le cas spécifique d'un service de cloud gaming. L'algorithme de contrôle de congestion en découlant est nommé "C3G". La tâche n'est pas aisée, du fait que UDP n'utilise pas de mécanisme d'acquittement. Les auteurs définissent alors le RTVL (**R**ound **T**rip **V**ideo **L**atency). C'est l'analogue au RTT utilisé dans BBR. Ici, le client n'accuse pas réception des paquets qu'il reçoit, mais des frames reçues. Une illustration est fournie dans le schéma situé à gauche de la figure 2.3.

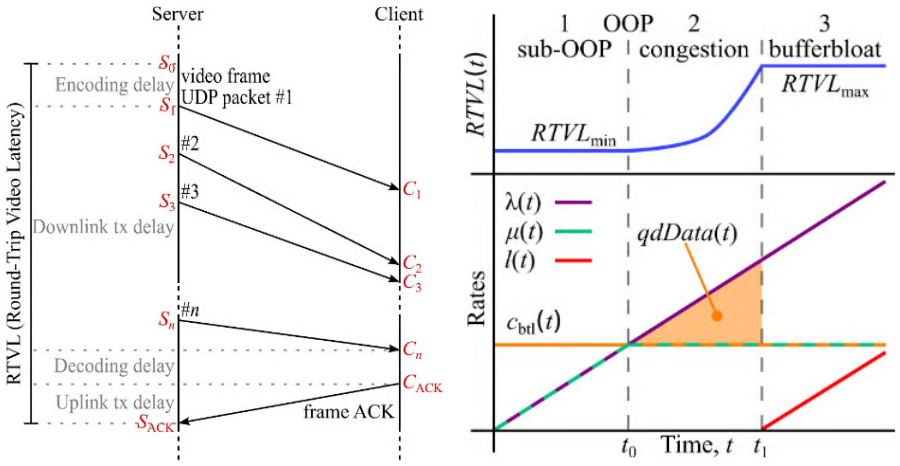


FIGURE 2.3 – Visualisation du RTVL et des phases de l'algorithme C3G [AMC+19]

Le RTVL désigne ainsi le laps de temps entre :
— La sortie d'une frame du moteur de jeu (*début de l'encodage*) ;
— La réception de l'aquittement de la frame en question (*frame décodée*) ;
Nous pouvons voir à droite de la figure 2.3 les différentes phases de congestion ;

1. $t<t_0$: absence de congestion. Le débit envoyé $\lambda(t)$ est égal au débit reçu $\mu(t)$.
2. $t_0 < t < t_1$: bande passante $c_{btl}(t)$ du *bottleneck* dépassée. Augmentation des délais.
3. $t>t_1$: le *buffer* est plein. Les nouveaux paquets sont *droppés*.

Bien entendu, l'algorithme présenté a recours à des estimations. L'estimation de *qdData* est donnée par l'équation 2.3. Au même titre que pour BBR, les auteurs utilisent des fenêtres temporelles. Les valeurs $\bar{\mu}(t_{now})$, $\bar{\lambda}(t_{now})$ et $\overline{RTVL}(t_{now})$ sont les moyennes des échantillons $\mu(t)$, $\lambda(t)$ et $RTVL(t)$ observés pendant la période $[t_{prev}\,;\, t_{now} = t_{prev} + \delta_t]$.

$$qdData(t) = \int_{t_0=t_{cong}}^{t} [\lambda(t) - \mu(t)]dt \approx \sum_{t_0=t_{cong}}^{t} [\lambda(t) - \mu(t)] \tag{2.3}$$

Dans le cas où $qdData(t_{now}) > 0$ et $\overline{RTVL}(t_{now}) > \overline{RTVL}(t_{prev})$, l'algorithme se considère en phase de congestion. Il va alors prédire de deux façons différentes la valeur de RTVL en $t=t_{now} + \delta_t$. Dans le cas où une des prédictions dépasse le seuil maximum autorisé pour RTVL, la phase de drainage est enclenchée. Cette phase consiste à appliquer le débit minimal ν_{min} pendant un temps T_{drain}, donné par l'équation 2.4.

$$T_{drain} = \frac{qdData(t_{drain})}{\bar{\mu}(t_{drain}) - \nu_{min}} \tag{2.4}$$

A l'issue de cette période, le *buffer* du *bottleneck* est à nouveau vide. Cela se traduit par $qdData(t_{cong} + T_{drain}) = 0$. En l'absence de congestion, l'algorithme rentre périodiquement en phase d'exploration. Le but est de traquer la bande passante disponible. Au même titre que pour la phase de drainage, la phase d'exploration consiste à appliquer un débit λ_{exp} pendant T_{exp} secondes. A l'inverse de BBR, l'algorithme tient compte de l'historique des explorations précédentes. Dans l'idée, le comportement sera plus conservatif si une récente tentative d'accroissement du débit a conduit à une phase de drainage.

Les mécanismes de contrôle de congestion ne sont pas qu'optimisés pour un type de trafic précis mais également pour un type de réseau. Le prochain CCA présenté a la particularité d'être particulièrement adapté aux réseaux mobiles.

2.2.4 Sprout

Sprout est destiné aux applications *temps réel* à haut débit [WSB04] et a été développé pour fonctionner sur les réseaux à capacité variable. Ce sont typiquement des réseaux cellulaires dont la capacité peut varier fortement dans un cours laps de temps en fonction de l'environnement et pouvant aller jusqu'à des micro-coupures. Il se distingue de l'ensemble des CCAs par rapport à son signal de congestion. C'est l'évolution des temps d'inter-arrivées qui lui permet d'identifier une détérioration des conditions réseau. Cette grandeur est en fait utilisée par le récepteur pour prédire l'état du réseau. Il envoie à l'émetteur une prédiction des données livrées sur 8 intervalles consécutifs (*e.g 160 ms*). Un récapitulatif des données reçues (*resp.* perdues) est joint au rapport. Il permet à l'émetteur d'estimer la quantité de données sur le réseau. Le contrôle de congestion se base sur les 8 prédictions susvisées.

L'émetteur établit son débit en se basant sur ce *feedback* et sur sa propre estimation des *données en vol*. Il l'incrémente à chaque émission et la décrémente selon les prédictions du récepteur. Cette estimation lui permet de déterminer les données qu'il peut émettre *en toute sécurité*. La probabilité qu'un paquet soit livré en moins de 100ms doit être de 95%. Pour ce faire, l'émetteur estime la quantité de données drainées au cours des 100 prochaines millisecondes. La quantité de données à émettre est la différence entre cette grandeur et la charge du réseau (*données en vol*). Le procédé est répété pour chaque intervalle de 20ms. Il est illustré par la Figure 2.4.

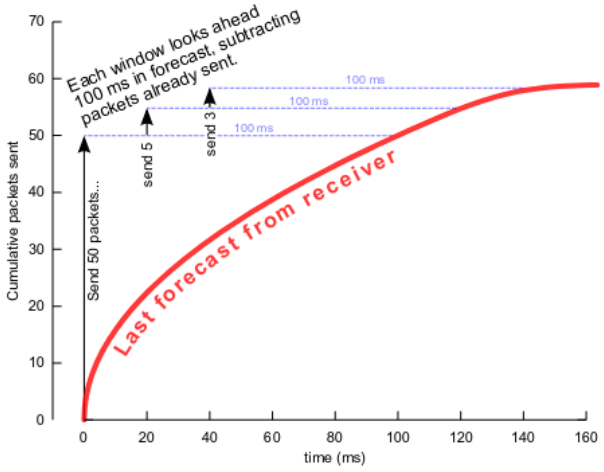


FIGURE 2.4 – Calcul de la taille de fenêtre selon la prédiction [WSB04]

Les auteurs soulignent que Sprout est un CCA destiné aux réseaux à capacité variable. Il n'a pas été conçu pour faire face à du trafic concurrent. Notons qu'un réseau cellulaire maintient une file d'attente dédiée par utilisateur. Les retards y étant perçus sont généralement auto-infligés. Dans le cas où l'équipement d'un utilisateur utiliserait TCP en-dehors de Sprout, les délais deviendraient alors considérables. Dans le cas de réseaux à plus faible degré d'isolement comme le WI-FI, l'utilité d'un AQM serait donc nécessaire pour pallier ce problème. Les auteurs préconisent alors d'utiliser l'AQM `CoDel` qui permettrait de contrôler les délais tout en garantissant une bonne utilisation du lien. L'ensemble du trafic est cependant soumis à un même compromis *débit / délais*.

Une des limites des mécanismes de contrôle de congestion précédents est qu'ils restent end-to-end et peuvent mettre trop de temps à détecter une congestion et donc à y réagir, alors que d'autres signaux de congestion précoces peuvent aujourd'hui être émis par les équipements réseau intermédiaires grâce à l'émission d'un signal de type Explicit Congestion Notification (ECN), normalisé au niveau de l'en-tête IP [KR01].

2.2.5 SCReAM

C'est le cas de SCReAM : "*Self-clocked Rate Adaptation for Conversational LTE*", un algorithme de contrôle de congestion basé pertes et délais proposé par Ericsson pour RTP et décrit initialement par Johansson et al [Joh14]. De plus amples détails sont donnés dans la RFC chargée de décrire l'algorithme [JS17].

Algorithme initial

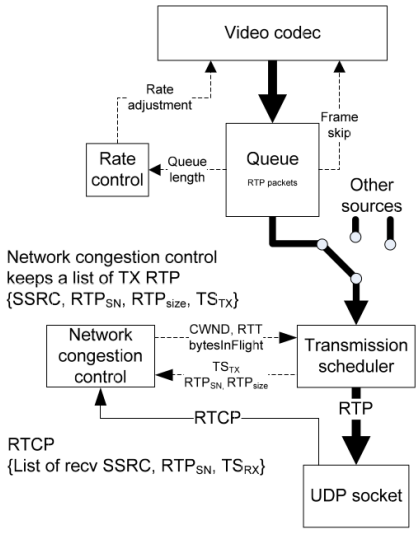


FIGURE 2.5 – Architecture Scream [Joh14]

Au même titre que pour GCC, l'émetteur requiert un *feedback* du récepteur. Pour ce faire, le couple protocolaire RTP/RTCP est mis à contribution. RTP n'ayant pas d'acquittement, le récepteur doit en effet confirmer à l'émetteur la bonne réception de paquets via RTCP. On retrouve ainsi dans chaque *feedback* les numéros de séquence des derniers paquets RTP reçus. Le récepteur y rajoute le *timestamp* de réception du paquet ayant le plus haut n° de séquence (seq_{Max}).

Avec ces informations, l'émetteur est en mesure de calculer la quantité de données envoyées, mais non acquittées ("*bytes in flight*"). Il fait une estimation des délais de *queuing* en se basant sur plusieurs mesures. À chaque rapport RTCP reçu, une nouvelle mesure est prise en compte. Cette valeur est issue des *timestamps* d'envoi et de réception du paquet seq_{Max}.

Le contrôle de congestion consiste à définir la quantité de données que l'on peut envoyer sans congestionner le réseau. Il convient donc de définir une limite supérieure sur la quantité de "*bytes in flight*". Cette limite est la fenêtre de congestion ; `CWND`. Elle est calculée en fonction de la quantité de données acquittée lors du dernier *feedback*. Le décalage entre les délais de *queuing* et le seuil "*qdelay_ target*" est aussi pris en compte. Notons que la valeur de ce seuil n'est pas constante. En effet, dans le cas où le flux considéré serait en concurrence avec un flux "*loss-based*", il baisserait son débit au bénéfice de ce dernier. Il convient donc de baisser la sensibilité de l'algorithme aux délais, et donc d'augmenter "*qdelay_ target*" en présence de tels flux. De fait, cette valeur est définie en fonction de l'évolution de *qdelay* sur un certain intervalle de temps. La perte de paquets, signe direct de congestion, entraîne une baisse immédiate de `CWND`. A l'inverse, "*qdelay_ target*" est augmenté.

Une fois que la valeur de `CWND` a été définie, le "*Transmission Scheduler*" détermine la quantité de données à envoyer sur le réseau. C'est tout simplement la différence entre `CWND` et le nombre de "*bytes in flight*". Dans le cas où *qdelay* est inférieur à "*qdelay_ target*", un MSS (Maximum Segment Size) supplémentaire peut être transmis. Toutes les données qui ne peuvent pas être envoyées sont placées en file d'attente.

Le "*Rate Control*" est chargé de spécifier un débit cible aux encodeurs. Dans le cas où une perte de paquets est détectée, le "*target_ bitrate*" est immédiatement baissé. Dans le cas contraire, l'algorithme tient compte de la quantité de paquets RTP placés en file d'attente. Le débit des encodeurs doit diminuer dès lors que cette quantité croît. Pour permettre une bonne QoE, le "*Rate Control*" cherche, de surcroît, à limiter la gigue. Il va ainsi considérer l'évolution des délais dans son calcul. Au final, un débit théorique $R_{unlimit}$ est obtenu. Du fait que les encodeurs ont une qualité maximale, le débit théorique est toutefois borné. Une fois les contraintes des encodeurs appliquées, on obtient un débit pratique R_{limit}.

Support d'ECN

D'après la RFC 3168 [FRB01], ECN permet aux routeurs de signaler un début de congestion. Pour ce faire, il leur suffit de modifier l'en-tête IP des paquets. On parle alors de marquage `CE` (*Congestion Explicit*). Après la réception d'un paquet `ECN-CE`, le récepteur en informe l'émetteur. Il place alors un indicateur `ECN-echo` dans un paquet lui étant destiné. L'émetteur y réagit comme s'il avait subit une perte de paquet [AGM+10]. La RFC 8311 [Bla18] vient cependant assouplir cette règle. Elle permet de procéder à des expérimentations sur ECN.

L'AQM DualPI2 que nous présenterons plus tard (*Section 2.3.5*) utilise ECN de façon détournée [BSBW23]. Il a recours au marquage `CE` pour limiter les délais de *queuing*. Les paquets sont donc marqués dès les premiers signes de congestion. Une telle approche ne convient pas aux CCAs conformes à la RFC 3168. Leurs débits respectifs se retrouveraient fortement diminués. Les CCAs doivent ainsi être adaptés à cette stratégie de marquage. C'est ainsi qu'a été crée la version L4S de SCReAM [Res23]. Elle appartient à la famille des CCAs '*scalables*'. En d'autres termes, son temps de récupération doit être indépendant du débit ;

"*A congestion control where the average time from one congestion signal to the next (the recovery time) remains invariant as flow rate scales, all other factors being equal. This maintains the same degree of control over queuing and utilization whatever the flow rate, as well as ensuring that high throughput is robust to disturbances.* " [SB23]

Le débit de SCReAM est adapté selon la proportion de paquets marqués `ECN-CE`. Le *feedback* client se fait via le protocole RTCP, comme défini dans la RFC 8888 [SPSR21]. La fréquence des rapports RTCP doit néanmoins respecter un compromis entre rapidité et charge du réseau. Elle ne doit généralement pas dépasser un rapport par *frame*. Une vidéo *streamée* en 30 fps contient une *frame* toutes les 33 ms. Dans le cas de SCReAM, la périodicité des rapports tient compte du débit. Notons qu'elle est toutefois bornée dans l'intervalle $[2ms; 100ms]$.

Amélioration de l'algorithme

Une amélioration de cet algorithme est présentée dans [FR21]. Le but est de résoudre le problème des "*self-inflicted delays*" de WebRTC. On retrouve deux types de canaux dans WebRTC ;

- Les "*Media Channels*" ;
  Permettent de transférer de l'audio ou de la vidéo via le protocole SRTP. Selon les recommandations de l'IETF, SRTP utilise SCReAM ou NADA[ZP13] comme CCA.
- Les "*Data Channels*" ;
  Basés sur le protocole SCTP. Le CCA utilisé est un variant de TCP.

Le problème réside dans le fait que l'interaction des deux protocoles porte atteinte aux "*Media Channels*". L'étude en question va donc coordonner les flux SRTP et SCTP issus d'une même application. Pour ce faire, les auteurs vont distinguer deux cas. Dans le cas où le débit des canaux RTP est borné par le *bottleneck*, la bande passante allouée aux flux SRTP (R_{rtp}) est dérivée de la fenêtre de congestion `CWND`. En se basant sur la priorité de chaque flux, les auteurs estiment la bande passante totale disponible SR_{total}. La formule est donnée dans l'équation 2.5. Notons que p_{rtp} et p_{sctp} désignent la somme des priorités des flux SRTP (resp. SCTP). La variable p_{tot} est la somme de toutes les priorités.

$$SR_{total} = \frac{p_{tot}}{p_{rtp}} * R_{rtp} \tag{2.5}$$

Dans le cas où le débit serait borné par la capacité des encodeurs, le taux de bande passante alloué aux flux SRTP est augmenté de façon relative. On se base en effet sur la différence de taille entre les deux dernière fenêtres de congestion. En ce qui concerne les flux SCTP, on leur accorde la différence entre le débit théorique $R_{unlimit}$ et le débit pratique R_{limit}, qui tient compte des limitations des encodeurs. De sorte à ne pas occasionner de congestion dès lors que les délais sont élevés, cette valeur est bornée. Elle ne peut dépasser le taux de données acquitté lors du dernier *feedback*. La dernière étape consiste à partager la bande passante SR_{total} parmi les différents flux. Le partage se fait selon les priorités assignées à chaque flux.

2.2.6 Problématique de la cohabitation entre CCAs

Le rapport de 2023 de l'ARCEP [ARC23a] s'intéresse aux interactions entre *Cubic* et *BBR*. Les auteurs y pointent un soucis d'équité entre ces deux algorithmes. Les flux régis par *BBR* auraient tendance à monopoliser trop de ressources, ce qui devrait être corrigé par la prochaine version de l'algorithme *BBR2*. Ces observations semblent en contradiction avec une étude menée par Turkovic & al. [TKU19]. Les auteurs y dénoncent l'avantage des CCAs basés *pertes* sur

les autres types de CCA. Ces derniers diminueraient leur débit sans pour autant aboutir à une diminution de la latence. *BBR* serait de plus très sensible aux variations du RTT. Cao & al. apportent quelques éclaircissements sur les performances de *BBR* [CJS+19]. D'après eux, elles dépendraient des caractéristiques du réseau. Ils pointent alors deux facteurs :

— le BDP, donné par le produit entre la bande passante et le RTT ;
— la taille du *buffer* situé au niveau du *bottleneck*.

Ce n'est pas étonnant, car comme présenté en section 2.2.1 *BBR* détermine son débit selon son estimation du BDP [CCG+16]. Une règle empirique vient de plus mettre en rapport le BDP et la taille de *buffer* [MB02]. Dans [MSOY18], Miyazawa & al. mettent *Cubic* et *BBR* en concurrence. Ils constatent un débit cyclique pour les deux flux respectifs. *Cubic* augmente ainsi son débit jusqu'à l'apparition de pertes. Cette hausse se fait au détriment de *BBR*, contraint de réduire ses émissions. Une fois le lien saturé, le phénomène s'inverse. L'apparition de pertes contraint *Cubic* à baisser son débit. La bande passante libérée est alors employée par *BBR*. D'après les auteurs, il faudrait limiter la taille du *buffer*. Le procédé accorderait plus de stabilité à *BBR*. Dans leurs expériences, Claypool & al [XC22b, XC22a] prennent plusieurs tailles de file en considération. Contrairement aux autres études, ils n'opposent pas *Cubic* et *BBR*. Ils étudient à la place leurs interactions avec des flux CG issus des trois plateformes suivantes : Google Stadia, NVidia GeForce Now, et Amazon Luna. Il apparaît qu'une grande file d'attente (*buffer du bottleneck*) profite à *Cubic*. Dans cette configuration, les flux CG baissent leur débit tout en subissant une hausse des délais. On retrouve le phénomène mis en avant pas Turkovic & al. [TKU19]. À l'inverse, une petite file et/ou une forte bande passante profitent à *BBR*. Un *buffer* de taille restreinte limite en effet les RTTs, alors utilisés comme indicateurs de congestion.

La cohabitation des CCA est un vaste problème mais il ressort notamment de ces diverses études que la taille de file d'attente au niveau du *bottleneck* influe sur le rapport de force entre les CCA basés pertes comme Cubic et ceux basés délais puis pertes comme BBR ou SCReAM, une grande file profitant aux premiers au détriment des seconds.

Type de CCA	Support ECN	Réaction / Pertes	Réaction / délais	Protocole	Optimisé r. cellulaires	Open source
BBR	Non	Non	RTT	TCP	Non	Oui
GCC	Non	Oui	Variations OWD	RTP/RTCP	Non	Oui
C3G	Non	Non	RTVL	UDP	Non	Non
Sprout	Non	Non	Temps d'arrivée	SSP	Oui	Oui
SCReAM	Oui	Oui	Seuil sur OWD	RTP/RTCP	Oui	Oui

TABLE 2.1 – Récapitulatif des principales caractéristiques des CCAs étudiés

2.3 Mécanismes de gestion de file d'attente

Nous avons vu plusieurs CCA utilisant chacun différentes techniques pour contenir les délais dûs à la congestion réseau engendrée par les flux qu'ils gèrent. Une manière complémentaire de maîtriser ces délais est l'utilisation de mécanismes de gestion de file d'attente dans les réseaux. Nous distinguerons les algorithmes de gestion active des files d'attente (AQM) qui visent à dropper ou marquer des paquets selon une stratégie particulière (RED, PIE, Codel, L4S) des disciplines de file d'attente (HTB) qui définissent plus simplement comment les paquets sont ordonnancés en attente d'être transmis. La discipline la plus simple et largement utilisée est un ordonnancement de type first-in, first-out (FIFO) couplé à une politique de délestage par la queue (drop tail) quand il n'y a plus de place. D'autres disciplines plus avancées et coûteuses de "*Fair Queing*" peuvent garantir l'équité entre les flux via un système de file d'attente par

flux couplé à un round-robin. Ces derniers sont souvent utilisés sur les machines terminales pour garantir l'équité des flux d'un même utilisateur.

2.3.1 RED

Dans [FJ93], Floyd & Jacobson présentent RED : "*Random Early Detection Gateways*", un AQM permettant de signaler la congestion proactivement. Pour détecter la congestion, l'algorithme présenté se base sur la taille moyenne de la file d'attente. Cette taille est calculée à l'aide d'une moyenne mouvante exponentielle :

$$avg = (1 - w_q)avg + w_q q$$

Dans l'idéal, cette grandeur doit être maintenue basse pour garantir de faibles délais. Dans le cas où elle serait trop importante, il incomberait à la *gateway* de signaler la congestion aux *endpoints*. Les auteurs définissent alors deux seuils ; min_{th} et max_{th}.

— Si $avg < min_{th}$: ne rien entreprendre, les délais sont assez courts.
— Si $min_{th} \leq avg < max_{th}$: signaler la congestion avec une probabilité p_a.
— Si $max_{th} \leq avg$: signaler la congestion sur tous les paquets.

Le signal de congestion peut se manifester de différentes formes. Il est possible de *dropper* les paquets, de sorte à ce que les *endpoints* ayant un CCA basé pertes baissent leur débit. L'inconvénient de ce type de signal est que la perte de paquets porte atteinte à la QoE. L'autre possibilité consiste à positionner un bit dans l'en-tête des paquets à marquer. L'écueil est que tous les protocoles de transport ne prennent pas en compte ce signal. Il est donc recommandé de privilégier le premier signal dès lors que le délai moyen dépasse un certain seuil.

La probabilité p_a est donné par l'équation 2.6. On voit que p_b augmente linéairement entre min_{th} et max_{th}. La probabilité finale, p_a, augmente dès lors que l'écart avec le dernier paquet marqué (*count*) augmente.

$$\begin{cases} p_a = \dfrac{p_b}{1 - count * p_b} \\ p_b = max_p * \dfrac{avg - min_{th}}{max_{th} - min_{th}} \end{cases} \tag{2.6}$$

2.3.2 Codel

Dans [NJ12], Nichols & al. présentent CoDel : "*Controlled Delay*". Cet AQM aspire à détecter les délais excessifs causés par une congestion persistante. Pour ce faire, le temps passé par chaque paquet en file d'attente est monitoré. On parle de *packet-sojourn time*. Dès lors qu'un paquet est sorti de la file d'attente, son *packet-sojourn time* est comparé à un seuil `target`. Le but est de ne pas avoir de délais persistants au-delà de cette limite. Autrement, l'AQM commence à dropper volontairement des paquets pour signaler la congestion aux *endpoints*. L'équation 2.7 permet de définir à quel moment *dropper* le prochain paquet. Cette phase de *dropping* prend fin à partir du moment où un *packet-sojourn time* passe en-dessous du seuil `target`.

$$next\_drop = current\_time + \frac{\texttt{Interval}}{\sqrt{count}} \tag{2.7}$$

Ici, `Interval` est une constante. Elle est choisie de sorte à ce que les *endpoints* puissent réagir aux pertes engendrées. De fait, sa valeur est liée aux RTTs des différents flux. Les auteurs

recommandent une valeur de 100ms dès lors que les RTTs sont compris entre et 10ms et 1s. La valeur *count* représente le nombre de paquets *droppés* depuis que la phase de *dropping* a commencé. Plus le nombre de paquets droppés est important et plus l'écart séparant deux *drops* consécutifs est faible. Dans le cas où deux phases de *dropping* seraient temporairement proches, le taux de paquets *droppés* est conservé. Dans les faits, au lieu de ré-initialiser la variable *count* à 0, une valeur proche de l'ancienne lui est attribuée. On peut en effet supposer que le taux de pertes adapté lors du dernier cycle de *dropping* est un bon point de départ. De plus amples détails sont donnés dans la RFC dédiée [NJMI18] et sur le pseudo-code mis à disposition par les auteurs [15].

2.3.3 PIE

Dans [PNP+13], Pan & al. présentent PIE : "*Proportional Integral controller Enhanced*", un AQM contrôlant le délai moyen passé en file d'attente.

Estimation des délais de *queuing*

Les délais de *queuing* sont estimés en faisant le rapport entre la taille *qlen* de la file et le débit moyen de sortie avg_drate. La mise à jour se fait de façon périodique, selon un intervalle de temps T_{update}. La variable avg_drate est estimée selon une moyenne mouvante exponentielle ;

$$avg\_drate = (1 - \epsilon * avg\_drate) + \epsilon * dq\_rate \tag{2.8}$$

Elle est mise à jour après chaque nouvelle mesure du taux courant dq_rate. Au cours d'une mesure, la somme des tailles de paquets envoyés est répertoriée. Dès lors que cette somme dépasse un certain seuil $dq\_threshold$, une estimation de dq_rate est accomplie. Elle consiste à faire le rapport entre cette somme et la durée de l'observation. Cette estimation est suivie d'une mise à jour du débit moyen de sortie, comme stipulé dans l'équation 2.8.

Calcul de la probabilité de *dropping*

La probabilité p de *dropper* un paquet dépendra des dernières estimations sur les délais de *queuing*. Elle est donnée par l'équation 2.9.

$$p = p + \alpha * (cur\_del - ref\_del) + \beta * (cur\_del - old\_del) \tag{2.9}$$

Ici, les variables cur_del et old_del sont les deux dernières estimations réalisées. La variable ref_del représente quant à elle le délai de référence. Dès lors que le délai courant dépasse ce seuil, la probabilité de *dropper* un paquet croît. Le paramètre α pondère l'impact qu'a cet écart sur p. Outre ce premier indicateur, la congestion peut s'accompagner d'une augmentation des délais. Il convient donc d'accroître la probabilité de *dropping* dès lors qu'un tel événement est constaté. Cette démarche permet de signaler la saturation du réseau aux *endpoints*. L'importance de ce facteur est pondérée par le paramètre β.

2.3.4 HTB

Sans discipline particulière, le partage des ressources n'est pas toujours équitable entre flux concurrents. De surcroît, les CCAs uniquement basés sur les pertes auront toujours tendance à

15. `https://queue.acm.org/appendices/codel.html`

remplir les files d'attente. Il s'ensuit une inexorable hausse des délais. Un ensemble de techniques visant à limiter la latence est répertorié par Briscoe & al. [BBP+16]. Les auteurs s'intéressent notamment à l'ordonnancement et à la gestion des files d'attente. Janczukowicz exploite ces deux pistes pour améliorer la QoS du trafic WebRTC [Jan17]. Ce trafic est régit par l'algorithme GCC (Google Congestion Control) présenté précédemment. Lorsqu'il est exposé à du trafic TCP concurrent, GCC a tendance à s'effacer. Deux méthodes sont alors mises à l'étude pour palier le problème.

La première consiste à limiter les délais d'attente de tout le trafic. L'AQM `fq_codel` est alors employé, muni de ses paramètres par défaut. Un seuil de 5ms porte ainsi sur les délais d'attente. Au-delà de cette limite, des signaux de congestion (*pertes*) sont envoyés aux terminaux. Le lecteur peut se reporter à la Section 2.3.2 pour plus de détails. La partie `fq` signifie *fair queuing* [DKS89]. Le principe consiste à promouvoir une certaine équité entre les flux. Dans le cas de `fq_codel`, chaque flux se voit attribuer une file d'attente (FIFO). L'ordonnanceur traite en priorité les files ne s'accumulant pas. Le procédé est donc désirable pour les flux *légers* à contrainte faible latence [CM12]. Cette approche ne semble cependant pas adaptée au trafic CG.

Intéressons-nous à présent à la deuxième méthode, recourant à des classes de trafic. Le principe consiste à regrouper les paquets nécessitant un même traitement [BBP+16]. Une politique d'ordonnancement tenant compte des spécificités de chaque classe est mise en oeuvre. Il existe deux moyens permettant de mettre en relation des classes et des politiques. Selon le modèle des *services différenciés* [BWC+98], un champ IP (DSCP) associe chaque paquet à une classe. La classification (*attribution des labels*) est alors effectuée en périphérie du réseau. Les routeurs centraux ont juste à adopter leur comportement selon la valeur du champ DSCP. Il existe aussi le modèle des *services intégrés* [BCS94]. Un protocole est ici exploité pour signaler les exigences des différents flux. L'ensemble des politiques peut alors être configuré en temps réel. Un routeur adaptant ses traitements selon les classes de trafic fait de l'*ordonnancement hiérarchique*. Certaines classes se voient donc attribuer un traitement préférentiel par rapport à d'autres [BBP+16]. La RFC 3246 définit par exemple un traitement offrant de faibles délais et limitant les pertes [BCB+02]. Janczukowicz s'appuie sur `HTB` (*Hierarchical Token Bucket*) pour gérer les classes de trafic. La hiérarchie prend la forme d'un arbre, dont les noeuds correspondent à des classes [Dev99a, Dev99b]. La racine désigne l'entièreté du lien. Les *feuilles* sont quant à elles rattachées à une discipline d'attente (*queuing discipline*). Une fois les classes définies, il est possible de leur conférer des attributs QoS. Janczukowicz cite ces trois grandeurs ;

— `Ceil` : débit maximum autorisé ;
— `Rate` : débit garanti ;
— `Prio` : priorité par rapport aux autres classes ;

Chaque classe peut donc émettre jusqu'à atteindre un débit `Rate`. Elle est alors en mode *vert*. Une fois qu'une classe a atteint son débit garanti, il lui est possible d'emprunter des *tokens* à la classe parent. Ce sont des ressources permettant la transmission de données. Une telle situation correspond au mode *jaune*. Le troisième et dernier mode est le mode *rouge*. Dans un tel cas, la classe a atteint son débit maximal autorisé. Elle ne peut donc plus transmettre. Notons que `prio` permet de départager les classes de même mode. Quelle que soit sa priorité, une classe *verte* passera toujours avant les autres.

Dans ses travaux, Janczukowicz définit deux classes ; une classe WebRTC et une classe BE (*Best Effort*). Un aperçu est promulgué sur la Figure 2.6. Deux configurations sont ensuite mises à l'étude. La première d'entre elles assure un débit minimal au trafic WebRTC. Ce débit est fixé à 204 kbps, selon le cas d'étude envisagé (*visioconférence*). Bien que le débit garanti assure une qualité minimale, la classe WebRTC est prioritaire. La deuxième configuration assure un débit bien plus grand à la classe WebRTC (*654 kbps*). La priorité est cependant accordée à la

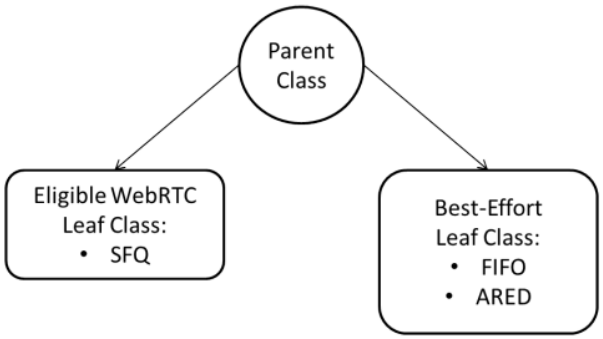


FIGURE 2.6 – Aperçu des deux classes utilisées par Janczukowicz [Jan17]

classe BE. Quelle que soit la configuration envisagée, la file WebRTC est toujours régie par `SFQ` [McK90]. Ce mécanisme, au même titre que `fq`, assure l'équité entre flux d'une même classe. En ce qui concerne la classe BE, deux mécanismes différents sont testés ; FIFO et ARED [FGS01]. ARED est une amélioration de RED, présenté en section 2.3.1.

Les résultats de Janczukowicz prouvent qu'il est préférable de séparer les classes de trafic. De plus, la première configuration apparaît plus efficace. Il convient donc de garantir un débit strictement nécessaire à WebRTC, tout en lui accordant la priorité. Quelques remarques ont de plus été formulées par rapport à la file BE. La taille de la file doit tout d'abord être limitée. Dans le cas contraire, TCP tarde à détecter les pertes, et donc à réagir à la congestion. De surcroît, les acquittements peuvent être reçus tardivement. L'émetteur est donc susceptible de saturer le lien par l'envoi des paquets reportés. Les meilleurs résultats ont été obtenus avec un file ARED de taille réduite.

2.3.5 L4S

Dans [BDST+16], Bondarenko & al. s'intéressent aux CCAs "*scalables*". Ceux-ci permettent de respecter de très faibles délais de *queuing*, indépendamment de la charge réseau. L'inconvénient est que ce type de CCAs ne peut pas être mélangé aux autres. DCTCP en est un exemple [AGM+10]. Initialement destiné aux *datacentres*, il s'appuie sur la technologie ECN.

Il existe plusieurs règles régissant le marquage des paquets. Dans [BDST+16], Bondarenko & al. considèrent un seuil d'occupation. Tant que le seuil susvisé est dépassé, le routeur marque l'ensemble des paquets. D'autres stratégies de marquage peuvent être mises en place. Dans [SSH+23], Son & al. se fondent sur l'AQM RED (voir section 2.3.1). Le marquage des paquets suit une probabilité proportionnelle à la hausse des délais. Cette probabilité augmente linéairement entre deux seuils pré-définis ; α et β.

Bondarenko & al. proposent de séparer le trafic *classique* du trafic *scalable*. Ils placent alors deux files distinctes au niveau du *bottleneck*. Chacune de ces deux files est gérée par un AQM, et génère donc des signaux de congestion. La file L4S, destinée au trafic *scalable*, est la plus agressive quant à l'envoi de tels signaux. Le procédé permet d'aboutir à de très faibles délais, quelle que soit la *charge* du réseau.

C'est ainsi qu'est né DualPI2 [ADSB+19], un AQM constitué de deux files d'attente pour résoudre ce problème de coexistence entre trafic *classic* et trafic *scalable*.

Il est donc intéressant d'appliquer deux traitements différents, en adaptant le signal selon le type de trafic concerné. Les auteurs définissent ainsi une file *classique* et une file "*Low Latency Low Loss Scalable Throughput*" (L4S). La file L4S est réservée aux flux sensibles aux pertes et/ou

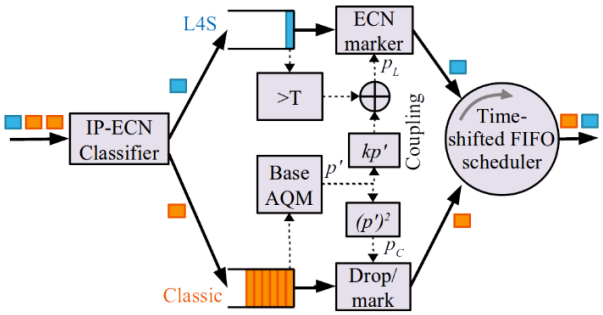


FIGURE 2.7 – Architecture de L4S [ADSB+19]

à la latence. Pour préserver de faibles délais, les paquets sont signalés par un marquage ECN dès lors que la taille de la file L4S dépasse un certain seuil T. La compatibilité avec ECN est donc une condition nécessaire pour bénéficier d'un placement dans cette file. Nous pouvons voir sur la figure 2.7 un schéma représentatif de L4S.

Départager le trafic

La première étape consiste à définir dans quelle file acheminer les paquets (IP-ECN Classifier). Cette classification se fait par le biais de 2 bits positionnés dans l'entête des paquets IP.

— `00` = NECT, ECN non pris en charge ;
— `01` = ECT(1), compatibilité ECN ;
— `10` = ECT(0), compatibilité ECN ;
— `11` = ECN-CE, notification de congestion (*marquage d'un paquet dans le réseau*) ;

La distinction entre ECT(0) et ECT(1) permet de définir quels paquets iront dans la file L4S. Dans la configuration par défaut, ce sont les paquets marqués ECT(1) qui y sont placés.

Probabilité de signalisation

Au même titre que pour [FJ93] et [PNP+13], l'AQM a recours à des probabilités de signalisation. La signalisation d'un paquet consiste à le *dropper* s'il ne supporte pas ECN. Dans le cas contraire, il est marqué (*positionnement du flag ECN-CE*). Chaque file d'attente a une probabilité associée ; p_l pour la file L4S et p_c pour la file classique. Leur calcul est dérivé d'une probabilité de base p', calculée périodiquement. Le calcul de p' s'appuie sur les délais de *queuing*. Nous retrouvons la même formule qu'en [PNP+13] ; il convient donc de se référer à l'équation 2.9. La variable *cur_del* est ici le maximum entre le délai d'attente des deux files. Afin d'assurer une certaine équité, un couplage est mis en place. L'équation 2.10 en traduit le mécanisme. La variable k y apparaissant est le facteur de couplage.

$$\begin{cases} p_l = k * p' \\ p_c = (p')^2 \end{cases} \tag{2.10}$$

Ordonnanceur FIFO

Le dernier composant de l'architecture est un ordonnanceur FIFO (FIFO Scheduler). Il doit choisir la file à partir de laquelle il récupérera le prochain paquet. Le choix portera sur le paquet ayant passé le plus de temps en file d'attente. Afin de prioriser les paquets de la file L4S, un

décalage temporel est appliqué.

Études supplémentaires sur L4S

Oljra & al. soulèvent néanmoins quelques défaillances [OGBT20b]. Au cours de leurs expériences, ils opposent un flux *Cubic* à un flux *DCTCP* supportant ECN. Ils étudient alors l'évolution de leurs débits et de leurs délais de *queuing*. Comme nous pouvons le voir sur la Figure 2.8, plusieurs scénarios sont envisagés. On y voit que *DCTCP* est plus *agressif* lorsque le BDP est bas (*coin supérieur gauche*). Dès lors que le RTT atteint les 50 ms, le débit de *DCTCP* se situe en deçà de l'équité.

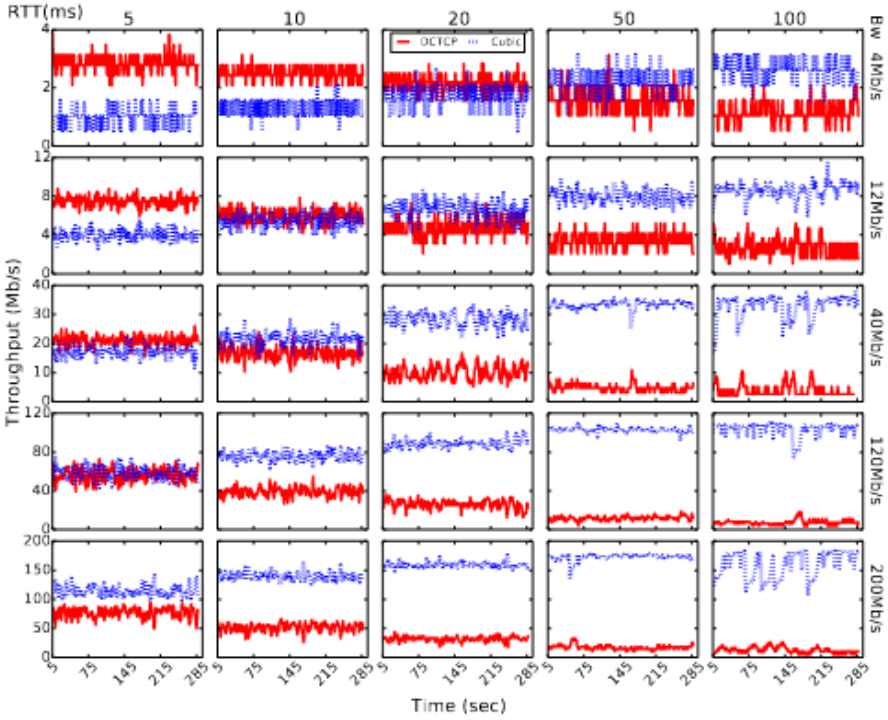


FIGURE 2.8 – Comparaison entre le débit de DCTCP et Cubic [OGBT20b]

Pour palier ce problème, Nadás & al. mettent au point un nouvel ordonnanceur [NGFL20]. Leur contribution est la généralisation de CSAQM (*Core Stateless AQM*)[NGHL18], ne s'appliquant qu'à une seule file. L'ordonnancement se base sur les PPV (*Per Packet Value*), des valeurs assignées à chaque paquet en périphérie de réseau. Elles sont comparées à un seuil CTV (*Congestion Threshold Value*) au niveau du *bottleneck*. Le seuil susvisé est calculé périodiquement, et reflète le niveau de congestion actuel. Les paquets se situant en-dessous du seuil sont marqués (`ECN-CE`) ou droppés par l'ordonnanceur. La technique permettrait une meilleure équité entre flux, indépendamment de l'hétérogénéité entre RTTs.

Un travail a aussi été mené du côté des terminaux. Une RFC vient d'ailleurs spécifier le comportement des CCAs transitant par L4S [SB23]. On parle des "*Prague L4S Requirements*". Une des exigences consiste à réduire leur dépendance vis-à-vis du RTT. Cette propriété devrait résoudre le problème soulevé par Oljra & al. [OGBT20b]. Une norme supplémentaire impose l'utilisation de *accurate ECN* [BKS23]. Ce mécanisme permet de remonter plus de un *feedback* (`ECN echo`) par RTT. Trois bits de l'en-tête TCP sont alloués à cet effet. Il s'agit des bits `AE`

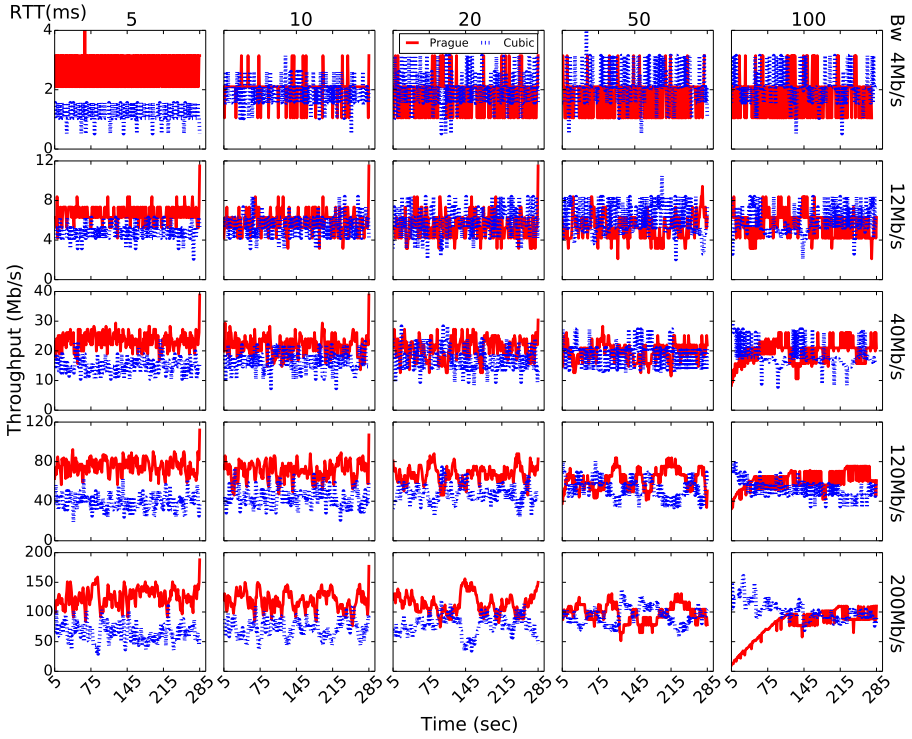


FIGURE 2.9 – Comparaison entre le débit de Prague et Cubic [OGBT20a]

(*Accurate ECN*), `CWR` (*Congestion Window Reduction*) et `ECE` (*ECN echo*). D'autres spécifications sont introduites, mais nous ne les présenterons pas. Le lecteur peut se référer à la RFC 9331 [SB23] pour bénéficier de plus amples détails.

Le routage se faisant selon le positionnement des bits ECN dans l'en-tête IP, il repose sur la bonne foi des *endpoints*. Des travaux sur la détection du non-respect du protocole ECN ont été menés [LFC+20]. Les auteurs ont mis en place un programme permettant de détecter et pénaliser les clients ne réagissant pas au signal `ECN-CE` (*congestion*). Il en est de même pour les émetteurs qui ne baissent pas leur fenêtre de congestion après réception d'un signal `ECE`.

TCP Prague est l'adaptation de DCTCP aux "*Prague L4S Requirements*" [BSA+18]. Ce CCA permet d'aboutir à un meilleur partage des ressources vis-à-vis du trafic *classique*. La Figure 2.9 confirme cette assertion. On y voit que le RTT n'est pas aussi impactant que précédemment pour DCTCP (Figure 2.8). L'expérience a été menée en plaçant un ordonnanceur FQ (*Fair Queuing*) sur l'interface de sortie du serveur. Le dispositif permet de limiter les *bursts* via des techniques de *pacing*. Dans le cas de TCP Prague, l'ordonnanceur n'apporte aucune plus-value. Ce CCA comporte en effet des mécanismes atténuant les *bursts*. À l'inverse, DCTCP aboutit à de meilleurs résultats en présence de l'ordonnanceur FQ. Les débits des flux *classiques* et *scalables* sont alors mieux équilibrés. Les auteurs de [OGBT20b] en concluent que L4S est sensible aux *bursts*.

Depuis TCP Prague, d'autres CCAs ont été créés en vue de transiter par L4S. Son & al. proposent une solution destinée aux réseaux 5G [SSH+23]. Ils situent l'AQM DualPI2 au niveau de la station de base. Le marquage de la file L4S s'y fait au prorata des délais de *queuing*. Le procédé permet à l'émetteur d'en faire une estimation. Contrairement à TCP Prague, le *feedback* du récepteur transite via RTCP. L'émetteur calcule alors un débit cible $T(f_t)$ pour chaque rapport RTCP f_t reçu. L'évolution du débit est conditionnée par l'estimation des délais $e(f_t)$. Un ensemble de seuils permet d'adapter l'intensité de la variation. L'évaluation conduite

par les auteurs est concluante ; leur solution surpasse GCC. D'autres CCAs ont été adaptés pour répondre aux *exigences de Prague*. On peut citer BBRv2 [CCG+18] et la variante L4S de SCReAM[16]. SCReAM a d'ailleurs été appliqué à une session de jeu en réalité augmentée (AR) [BJSOC21]. Les auteurs étudiaient alors les bénéfices de L4S sur un réseau 5G. Ils observent un amoindrissement des délais lorsque le trafic transite par cette file. S'y rajoute un très faible taux de pertes, aboutissant à bon niveau de QoS.

2.4 Conclusion

Dans ce chapitre, nous avons présenté différents mécanismes permettant de maîtriser la latence due au réseau que ce soit au niveau des end-points via les algorithmes de contrôle de congestion ou au niveau du réseau via les mécanismes de gestion des files d'attente. À l'issue de ce tour d'horizon, nous pouvons tirer plusieurs enseignements pour notre cas d'étude.

D'une part, nous avons vu que tous les CCAs ne sont pas adaptés au trafic CG, véhiculé par RTP. Nous fournissons un récapitulatif de ceux que nous avons étudié en Table 2.1. Parmi les candidats, nous retenons SCReAM qui paraît bien adapté et dont le support d'ECN permet en théorie une meilleure réactivité aux problèmes de congestion. De plus, le support d'ECN par SCReAM et d'autres de ses propriétés intrinsèques le classent dans les CCAs dits scalables et compatibles avec la file basse latence (L4S) de l'AQM DualPI2. Dans ce cas précis, CCA et AQM ne sont plus orthogonaux mais travaillent en synergie et permettraient donc de meilleures performances que les technologies actuellement utilisées.

D'autre part, l'utilisation d'une discipline de file d'attente pouvant gérer des priorités entre classes de trafic, HTB en particulier, semble également pertinente dans notre cas. Les flux de type Cloud Gaming pourraient ainsi constituer une classe prioritaire et disposer d'une petite file du fait de leur forte contrainte de latence. Il faut cependant être en mesure de pouvoir reconnaître le trafic de cloud gaming en tant que tel avant de pouvoir mettre en œuvre cette solution.

Le chapitre 5 est ainsi dédié à la classification du trafic CG. Nous présentons ensuite une stratégie d'implantation se basant sur les nouvelles technologies de programmabilité réseau. Enfin, le chapitre 7 traitera des bénéfices inhérents aux deux approches susvisées. Nous y évalueront leurs bénéfices qu'elles apportent au transport du trafic CG.

Pour lors, les deux prochains chapitres vont s'intéresser à la caractérisation précise du trafic CG. Nous y considérerons plusieurs plateformes, dans des environnements réseau distincts. L'étude de leur comportement en conditions réseau dégradées nous permettra de découvrir leurs spécificités, mais aussi de mettre en évidence leurs limites actuelles.

16. `https://l4steam.github.io/PragueReqs/Scream_L4S_requirements_Compliance_and_Objections.pdf`

Deuxième partie

Caractérisation du trafic Cloud Gaming en environnement réseau contraint

Chapitre 3

Sur réseau à capacité fixe

Sommaire

3.1 Introduction 41
3.2 Testbed 42
3.3 Contraintes préalables à la session de jeu 43
3.3.1 Perte de paquets 44
3.3.2 Limitation de la bande passante 45
3.3.3 Ajout de latence 47
3.3.4 Génération de gigue 48
3.3.5 Variation des IAT et de la taille du payload face aux contraintes . . 49
3.4 Contraintes durant la session de jeu 50
3.4.1 Perte de paquets 50
3.4.2 Limitation de la bande passante 52
3.4.3 Ajout de latence 53
3.4.4 Génération de gigue 54
3.5 Conclusion 55

3.1 Introduction

Dans notre partie d'état de l'art, nous avons abordé divers mécanismes de résilience vis-à-vis des conditions du réseau. Si le réseau est congestionné, il est primordial pour une application qui ne souhaite pas subir des délais et des pertes d'adapter son débit en conséquence. Dans le cas contraire, l'engorgement ne fera qu'empirer, accentuant la latence de bout-en-bout et générant des pertes dans le réseau, ce qui porte grandement atteinte à certains types de services. En effet, comme nous l'avons vu dans le chapitre 1, les services "temps réel" comme le Cloud Gaming sont sensibles à la latence qui nuit à la qualité des interactions, mais également aux pertes qui ne peuvent être compensées à posteriori par un mécanisme d'aqcuittement et de retransmission. On s'attend donc à ce que les plateformes de Cloud Gaming soient en mesure d'adapter rapidement et fortement leur débit à différentes contraintes réseau.

Dans le présent chapitre, nous nous intéressons aux mécanismes d'adaptation de 4 plateformes de CG ; Nvidia GeForce Now (GFN), Sony Playstation Now (PSN) [17], Google Stadia

17. Intégré au service PlayStation Plus Premium en juin 2022

(STD), Microsoft Xbox Cloud Gaming (XC). Au moment de cette étude, ce sont les 4 principales plateformes commerciales disponibles en Europe. Au cours de nos recherches, il a été fait mention de serveurs de "*probing*". Ce sont des serveurs chargés d'estimer la QoS (*les conditions réseau*) au préalable d'une session de jeu [MUS+14]. En plus de cette estimation initiale, d'autres mesures peuvent être faites pour traquer l'évolution desdites conditions. Domenico & al. font d'ailleurs mention de flux RTP consacrés à cet effet [DDPT+21].

Pour mener à bien notre étude, nous allons observer le trafic CG sous différentes conditions réseau. À cet effet, nous apportons des perturbations synthétiques par le biais de l'outil "`tc-netem`". Les perturbations portent sur quatre grandeurs :

— le pourcentage de pertes ;
— la bande passante disponible ;
— la latence ;
— la gigue (*variations sur la latence*).

Chacune de ces grandeurs se verra impactée selon diverses intensités.

Nous choisissons, de plus, d'appliquer nos contraintes de deux façons différentes. Dans un premier temps, dans la section 3.3 nous perturbons les conditions réseau avant même de lancer une session de jeu. Au terme de leur estimation préalable de la qualité du lien, les plateformes devraient adapter leur débit en conséquence. Ceci permet d'évaluer leurs conditions minimales de fonctionnement. Le deuxième scénario, étudié dans la section 3.4, consiste à appliquer une perturbation passagère durant la session de jeu. Ceci permet d'observer les mécanismes d'adaptation des plateformes, la réactivité, l'étendue et la stabilité de l'adaptation, mais aussi leur capacité à se rétablir après à un retour des conditions normales.

3.2 Testbed

Dans la présente section, nous allons avant toute chose décrire le "*testbed*" mis en place. Notons que le réseau d'accès est un réseau à capacité fixe de type Fiber To The Home (FttH) à 10 Gbps. En vue d'appliquer des perturbations réseau, nous plaçons un *bridge* entre le client CG et la passerelle. Cet équipement est en charge d'appliquer des règles `tc-netem`, outil reposant sur le noyau linux et permettant d'appliquer différentes contraintes sur les paquets transitant par une interface. Une vue d'ensemble du *testbed* est donnée dans la figure 3.1. Comme nous pouvons le voir, nous appliquons les perturbations dans le sens descendant qui est le plus critique pour la QoS [JSSH11] et exigeant pour le réseau car il véhicule un flux multimédia haut débit pouvant créer de la congestion.

Pour pouvoir étudier le comportement des plateformes CG en conditions dégradées, nous procédons à des captures du trafic réseau généré dans chaque expérience. Ces captures sont directement réalisées au niveau de la machine exécutant un des clients CG sur Windows 10, grâce à l'outil *wireshark* (*v 3.4.6*). Les clients lourds de PSNow (*v 11.7.0*) et GeForce Now (*v 2.0.32.95*) ont été utilisés tandis que Stadia et XCloud étaient utilisés via le navigateur Chrome (Version 92.0.4514.131).

Tous les serveurs de jeu contactés sont situés en Europe d'après nos mesures de latence ou la base de données GeoIP. Sur un total de 100 pings, les serveurs STD et PSN ont un Round Trip Time (RTT) moyen de 13,65 ms et 9,82 ms. Concernant GFN et XC, nous avons utilisé la bases de données GeoIP [18] que nous avons importée dans Wireshark. Il semble que les serveurs GFN soient situés à Paris, et ceux de XC à Amsterdam.

18. https ://www.maxmind.com/en/geoip2-precision-country-service

Ces captures ont toutes été réalisées entre avril et mai 2021. Dans l'optique de promouvoir la reproductibilité des expériences, nous avons mis l'ensemble de nos données en accès libre. Elles sont accessibles via le lien suivant ; `https://cloud-gaming-traces.lhs.loria.fr`.

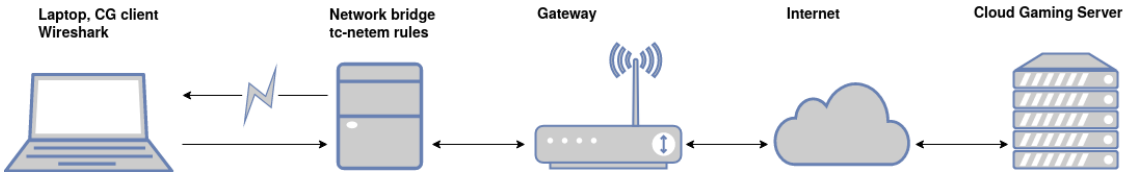


FIGURE 3.1 – Vue du testbed

3.3 Contraintes préalables à la session de jeu

Nous allons perturber les conditions réseau selon plusieurs intensités. On considère au total 3 niveaux d'intensité ; "*Modéré*", "*Élevé*" et "*Extrême*". Le tableau 3.1 lie chaque intensité aux perturbations lui étant associées. Certaines valeurs comme les niveaux de perte de paquets ou de gigue peuvent sembler très pessimistes, voire irréalistes pour un réseau actuel alors que les valeurs de bande passante et de latence représentent de mauvaises connexions ADSL. Cela ne pose cependant pas de problème, car nous cherchons plutôt ici à tester les limites des plateformes. La dernière colonne du tableau correspond aux valeurs mesurées sans qu'aucune contrainte ne soit déclenchée.

	Modéré	Élevé	Extrême	Référence
Pertes	5%	10%	20%	0%
Bande passante	20 Mbps	10 Mbps	5 Mbps	10Gbps
Latence	20 ms	50 ms	100 ms	8 ms
Gigue	5 ms	10 ms	20 ms	0 ms

TABLE 3.1 – Paramètres appliqués pour dégrader les conditions réseau

Il est impossible d'exécuter exactement le même jeu sur les quatre plateformes en raison de catalogues de jeux inconciliables mais nous avons choisi le même type de jeu de course pour chacune afin d'avoir une dynamique d'écran similaire. Au cours de nos expérimentations, nous avons remarqué que les plateformes peuvent mettre plus ou moins de temps à stabiliser leur débit en début de session de jeu. Pour mieux pouvoir les comparer, nous attendons qu'elles se stabilisent en ne prenant pas en compte les 200 premières secondes de chaque capture.

Pour chaque capture réseau, nous calculons et analysons l'évolution des trois métriques au fil du temps : le débit binaire, le temps d'inter-arrivée des paquets (IAT) et la taille de la charge utile (payload). Compte tenu du très grand nombre de courbes produites par notre campagne de mesures, à savoir 96 tracés au total : 4 plateformes * 4 contraintes réseau * 3 métriques rapportées * 2 directions, tous les résultats ne sont pas présentés par souci de concision. Nous nous concentrons sur les comportements les plus intéressants et principalement sur les variations de débit binaire entre serveur et client. Nous discutons des deux autres mesures lorsque cela est nécessaire et nous commentons parfois la direction client vers serveur.

3.3.1 Perte de paquets

Nous étudions ici l'impact occasionné par la perte de paquets. La figure 3.2 représente l'évolution du débit de chaque plateforme selon le taux de pertes infligé. Chacune des courbes correspond ainsi à un taux de pertes spécifique. Si on se réfère à la légende, "`los x`" signifie que l'on cause `x` % de pertes. Chaque courbe associée au label `ref` correspond au débit observé en conditions normales. Dans le deuxième graphe de la figure 3.2, on constate que la plateforme PSN n'a que deux courbes. Pour un taux de pertes dépassant les 5%, la plateforme refuse en effet de lancer une session de jeu. Les conditions réseau sont considérées comme insuffisantes pour le service.

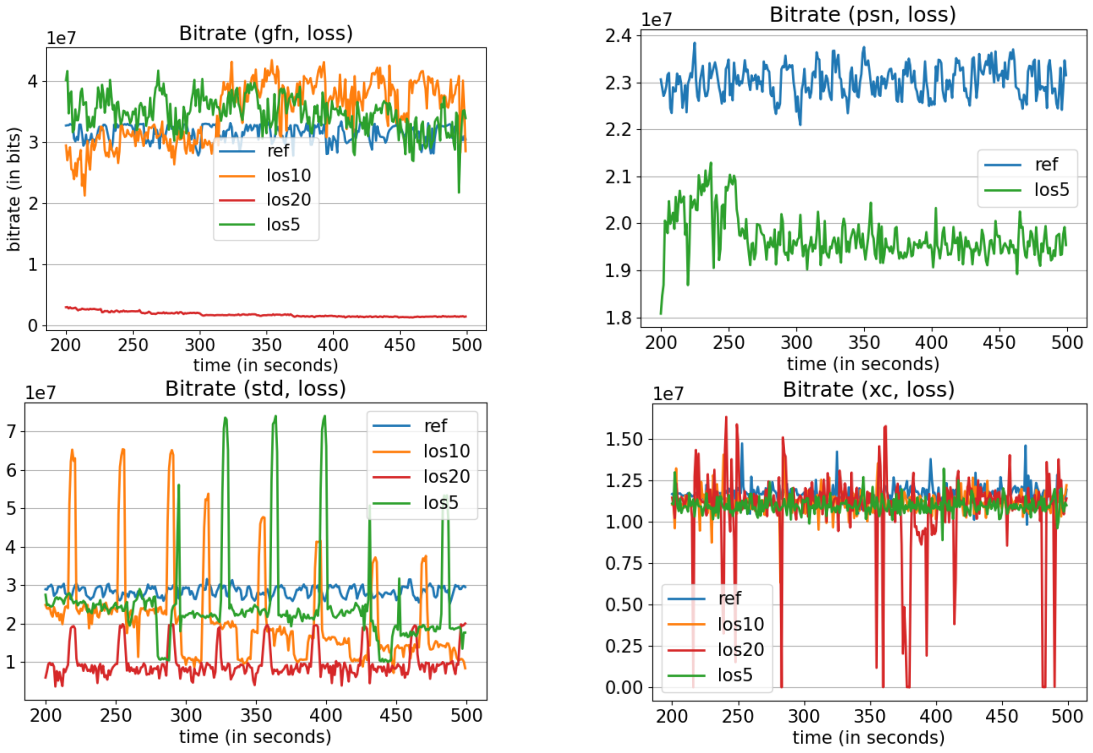


FIGURE 3.2 – Débit de GFN, PSN, STD et XC en fonction du taux de pertes (service⇒client)

Nous commençons par nous intéresser à la plateforme GFN. L'étude de son graphe montre qu'un taux de pertes inférieur à 10% ne va pas grandement impacter son débit. On peut néanmoins constater une légère hausse pour les deux premiers taux de pertes appliqués ; 5% et 10%. On pourrait expliquer cette augmentation par l'ajout de FEC (*Forward Error Correction*) dans les données. Le mécanisme consiste à rajouter de la redondance, de sorte à accroître la résilience face aux pertes survenues dans le réseau. Nous constatons à l'inverse un fort amenuisement du débit quand nous appliquons 20% de pertes. L'expérience de jeu est alors fortement altérée, du fait d'un manque de fluidité et d'une baisse de la qualité vidéo. À ce moment, la résolution est fixée à 540p pour un *framerate* de 30 fps, soit la qualité minimale possible.

La plateforme PSN est bien moins résiliente aux pertes que GFN. De fait, nous pouvons seulement commenter sa réaction face à un taux de pertes de 5%. Comme nous pouvons le voir sur le graphe associé, PSN choisit de décroître légèrement son bitrate. On passe alors d'un débit d'environ 23 Mbps à un débit de 20 Mbps. C'est un baisse d'environ 13% par rapport à ce que l'on peut observer en conditions réseau normales. En observant plus dans le détail les caractéristiques du trafic, on peut dresser quelques remarques supplémentaires. Tout d'abord, le temps d'inter-

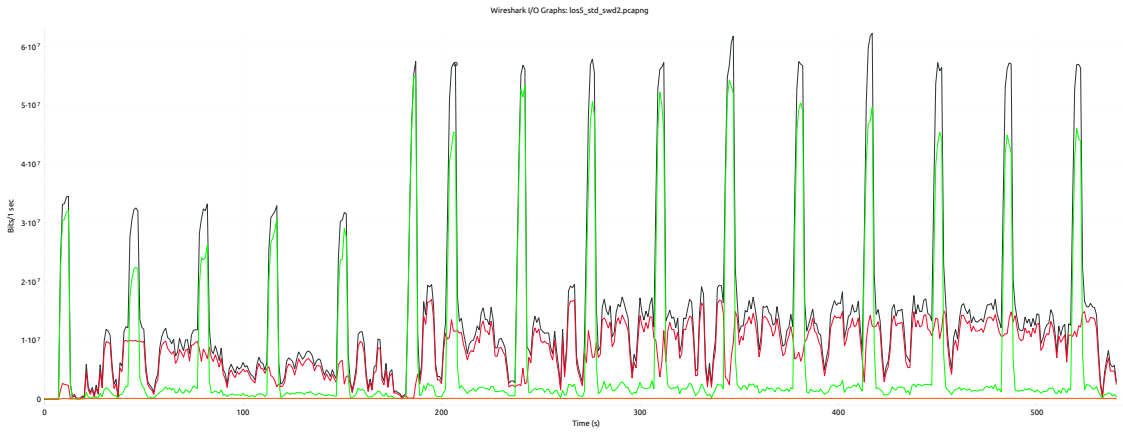


FIGURE 3.3 – Bitrate de Stadia pour 5% de pertes

arrivées (IAT) entre paquets descendants est considérablement accru. En moyenne, on observe une hausse de 39.6%. Au même titre que pour les IATs, la taille des paquets a également tendance à croître. Cela porte à croire que PSN a également recours à l'usage de FECs.

Dans le cas de STD, on voit que le bitrate est diminué dès lors que le taux de pertes augmente. De plus, on observe la présence de pics périodiques pour chaque perturbation se répétant toutes les 30 secondes. Il s'agit très probablement d'un mécanisme de sondage périodique de la bande passante disponible similaire à BBR (cf section 2.2.1). On note que leur amplitude est plus importante pour des débits plus hauts. Au terme d'une étude plus approfondie, nous remarquons le multiplexage de trois différents flux. On les identifie à l'aide de leurs SSRCs, présents dans l'entête RTP. Nous pouvons voir sur la Figure 3.3 le débit de chacun de ces flux. Le flux représenté par la courbe rouge correspond à la transmission vidéo. Sa courbe suit d'ailleurs celle du débit RTP global, représenté en gris. On constate que chacun des pics est causé par un flux d'ordinaire inactif, apparaissant en vert. Le troisième flux, représenté en orange, est négligeable par rapport aux deux autres et permet probablement de diffuser le contenu audio.

Pour finir, XC ne semble pas tenir compte des pertes. Quel que soit le taux de pertes infligé, le bitrate se situe toujours entre 10 et 12 Mbps. En ce qui concerne l'expérience de jeu, il est à noter qu'elle n'a pas été particulièrement impactée. Forts de ces observations, nous pouvons conjecturer que XC a une forte redondance de codage, ce qui peut s'expliquer par le fait que cette plateforme ait été conçue en priorité pour des clients sur smartphone, et donc devant recourir à des réseaux sans-fil (réseau Wi-Fi ou cellulaire).

3.3.2 Limitation de la bande passante

Dans cette sous-section, nous apportons des contraintes sur la bande passante disponible. La figure 3.4 représente le bitrate de chaque plateforme suite à nos contraintes. Une fois encore, chaque graphe correspond à une plateforme, et chaque courbe est associée à une limitation. En ce qui concerne le nommage, "`ratx`" signifie que nous bornons la bande passante à `x` Mbps.

Pour commencer, la plateforme GFN semble bien s'adapter aux différentes limitations. Elle réduit son bitrate de plus de 50% dès l'application de la première contrainte, passant ainsi de 33 Mbps à 16 Mbps. On observe que pour les trois contraintes appliquées, le bitrate occupe environ 80% de la bande passante disponible. Il semblerait qu'une marge soit conservée, par exemple, pour que l'établissement d'un flux concurrent ne sature pas immédiatement le bottleneck, ce qui créerait des délais. Il en est de même pour la plateforme PSN, qui adopte cependant une marge

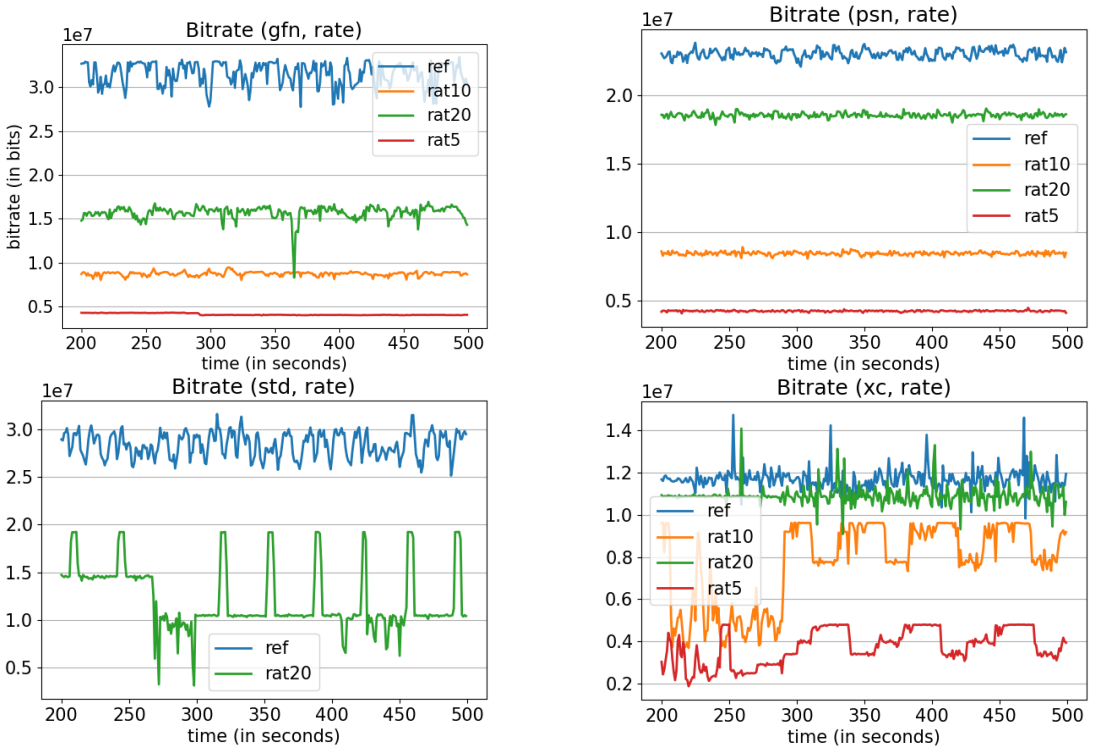


FIGURE 3.4 – Débit de GFN, PSN, STD et XC en fonction de la bande passante disponible (service⇒client)

un peu plus faible.

Pour STD, nous ne pouvons commenter son adaptation que pour une limitation "*Modérée*" à 20Mb/s de la bande passante. La plateforme refuse en effet de lancer une session de jeu pour des contraintes plus conséquentes, ce qui en fait la plateforme la plus exigeante en terme de débit minimum. En nous référant au graphe associé, on constate une brusque chute du débit aux alentours de la seconde 250. Nous passons alors d'une moyenne de 15 Mbps à environ 10 Mbps. Ce changement de débit s'accompagne d'un changement de résolution ; le flux vidéo passe alors de 1080p à 720p. Ce changement soudain après plus de 4 minutes d'expérimentation trahit une certaine instabilité. De plus, la réaction est inadaptée eu égard de la contrainte appliquée. Après le changement de résolution, environ 50% de la bande passante est alors inutilisée. On pouvait manifestement se prémunir de cette baisse, qui impacte fortement la QoE (*l'expérience de jeu*). Pour finir, au même titre qu'en 3.3.1, le débit de la plateforme est sujet à des pics périodiques. Que ce soit avant ou après le changement de résolution, tous les pics frôlent la bande passante maximale, fixée à 20 Mbps.

Examinons maintenant la plateforme XC. Du fait que son débit natif soit inférieur à la première limitation, nous ne commenterons pas la courbe associée. Nous pouvons cependant dresser quelques remarques sur les courbes "`rat10`" et "`rat5`". Dans les deux cas, la plateforme est dans l'incapacité de stabiliser son débit. On constate tout d'abord un accroissement pouvant atteindre 95% de la bande passante disponible. Le débit est alors stable pendant quelques secondes, avant d'être brutalement rabaissé à 80%. Tout le long de l'expérimentation, le débit alternera entre ces deux valeurs. Ces variations se font au détriment de la QoE, du fait des changements notables engendrés sur la qualité vidéo dont les niveaux de compression oscillent. Tout ceci met en relief les difficultés que rencontre XC pour adapter son débit.

3.3.3 Ajout de latence

À présent, nous nous intéressons à l'impact de la latence sur les 4 plateformes étudiées. Comme spécifié dans le tableau 3.1, nous rajoutons à tour de rôle 20ms, 50ms, et enfin 100ms de délais. Chacune de ces valeurs donne lieu à une courbe représentant l'évolution du *bitrate*. Nous pouvons voir sur la figure 3.5 le débit de chaque plateforme eu égard de la contrainte appliquée. Au niveau de la légende, l'ajout de `x` ms de latence correspond à la courbe "`latx`".

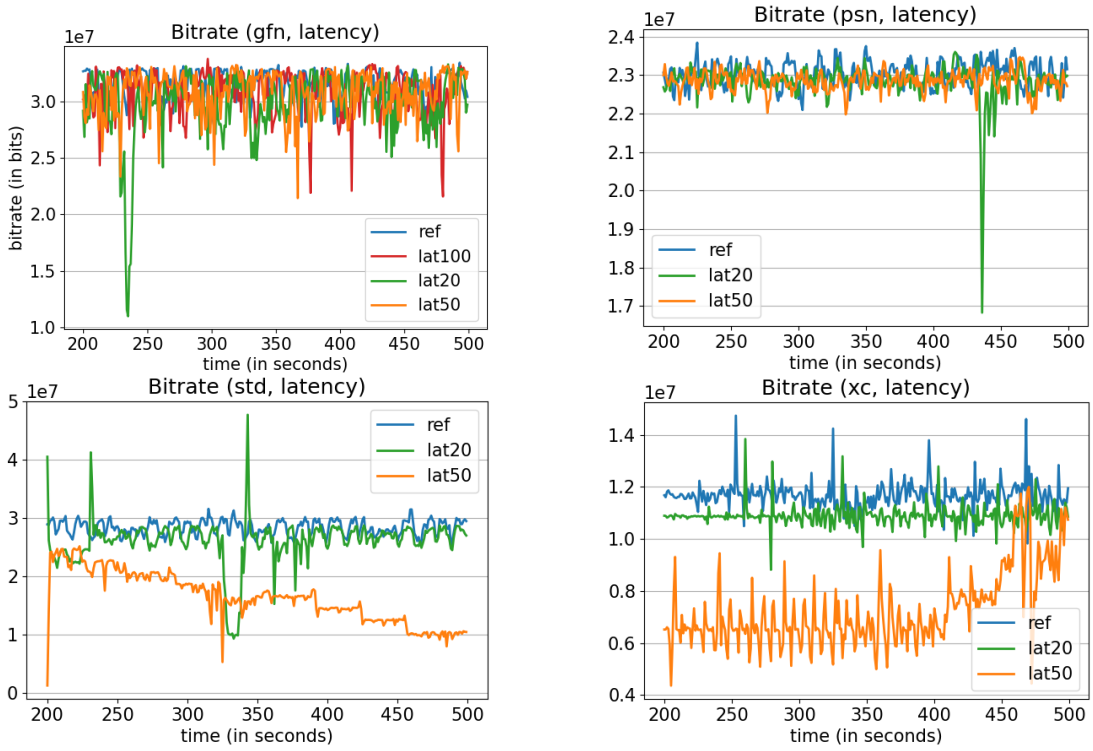


FIGURE 3.5 – Débit de GFN, PSN, STD et XC en fonction des délais (service⇒client)

La plateforme GFN ne réagit pas fortement aux délais induits. Son débit moyen semble d'ailleurs identique quelle que soit l'intensité de la contrainte appliquée. Il faut cependant noter que sa variance augmente légèrement dès lors que les délais croissent. Une âpre diminution du débit survient pour une des courbes aux alentours de la seconde 240. Cette altération passagère est due à une erreur dans le jeu lui même. Dans l'ensemble, GFN se montre très résiliente face aux délais. C'est d'ailleurs la seule plateforme à être opérationnelle après l'ajout de 100ms de latence. Si l'on comparait les plateformes entre elles, GFN et PSN seraient les plus semblables. Cette dernière est en effet insensible aux différents délais. Son débit se situe dans tous les cas aux alentours de 23 Mbps. L'absence d'adaptation n'est cependant pas synonyme de résilience. Il est d'ailleurs impossible d'accomplir une session de jeu pour 100ms de délais, à l'inverse de GFN.

Le comportement de STD dépend de l'intensité de la contrainte. On constate qu'aucune réaction n'est enclenchée quand on rajoute 20ms de délais. Lorsque l'on fait passer la latence au niveau d'intensité "*Élevé*", le comportement est tout autre. Au regard de la courbe associée, on remarque que le débit est progressivement réduit. Il passe alors de 30 Mbps à 10 Mbps. Une fois cette valeur atteinte, le débit se stabilise. La même observation a d'ailleurs été faite sur la Figure 3.4. Un contraste doit cependant être souligné ; l'absence de pics périodiques. Pour 100ms

de délais, il nous a été impossible de terminer l'expérimentation. Il nous a certes été possible de lancer une session de jeu, mais la plateforme y a rapidement mit fin.

Pour 20ms de latence supplémentaire, XC diminue légèrement son débit. Il perd en moyenne 1 Mbps. En cas d'augmentation des délais, on s'aperçoit que le débit devient très instable. Il est même réduit de moitié, comme on peut le voir sur la courbe "`lat50`". Notons que son instabilité est due à une forte fluctutation des IATs. À compter de la seconde 400, le débit entame une lente progression, jusqu'à atteindre sa valeur de référence. L'ajout de 100ms rend le service totalement indisponible, à l'instar de STD et PSN.

3.3.4 Génération de gigue

Pour finir, nous nous concentrons sur l'impact engendré par la gigue. Dans cette optique, nous allons faire varier la latence selon plusieurs niveaux d'intensité. Cette variation se fait selon une loi normale, centrée sur la quantité de délais rajoutée. Nous fixons cette valeur à 1 ms. En considérant la Figure 3.6, on remarque l'absence de graphe pour la plateforme PSN. Aucune session de jeu n'a en effet pu parvenir au bout de nos expériences. En ce qui concerne la notation, l'ajout de `x` ms de gigue se note "`jitx`".

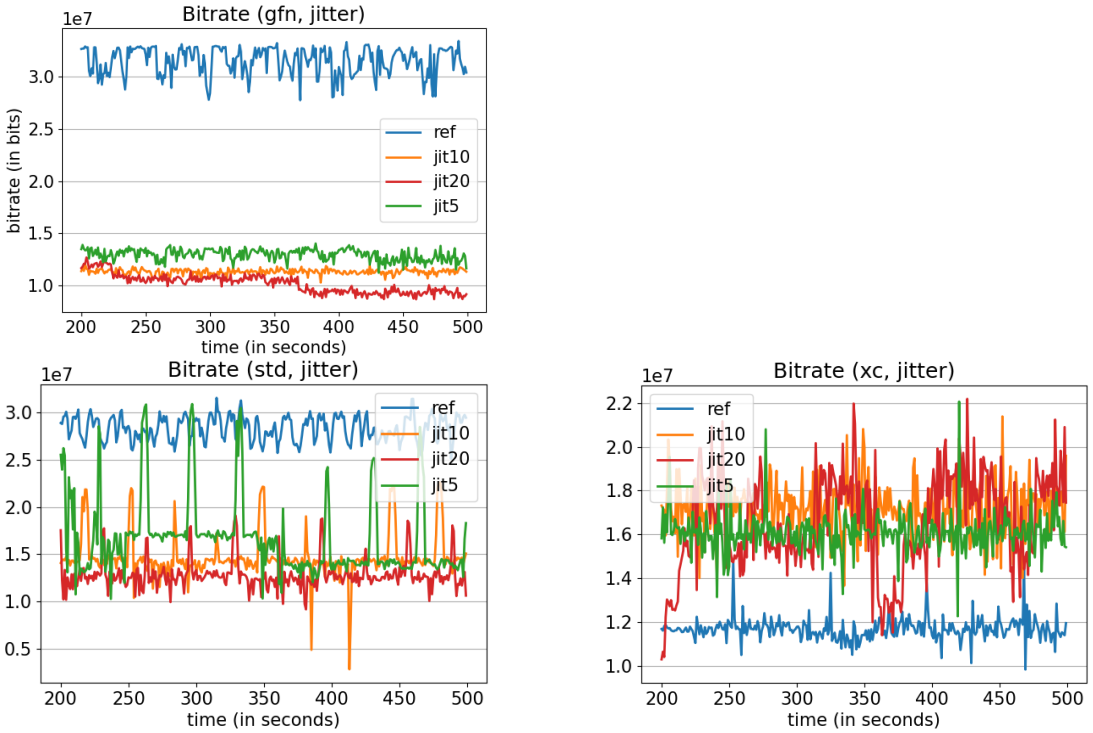


FIGURE 3.6 – Débit de GFN, PSN, STD et XC en fonction des valeurs de gigue (service⇒client)

La réponse de GFN est très forte vis-à-vis de la gigue. On constate que dès la première intensité, le bitrate est réduit d'environ 20 Mbps pour atteindre 13 Mbps. Lorsque l'on induit 20ms de gigue, le débit moyen est encore réduit pour atteindre 9 Mbps à la seconde 375. Cette réduction s'accompagne d'une diminution de la résolution. La qualité vidéo est alors à son plus bas.

La plateforme STD réduit fortement son débit dès lors que l'on ajoute de la gigue. Pour une intensité de 5ms, il est réduit de plus de 10 Mbps. On note de plus une certaine instabilité, avec

2 changements notables de débit. On distingue en effet un premier palier à ≈ 17 Mbps, puis un second à ≈ 14 Mbps. Ce second palier correspond justement au débit adopté par STD pour une gigue "*Élevée*". Pour finir, on peut souligner que la plateforme adopte un débit moyen de 13 Mbps pour une gigue de 20 ms.

La plateforme XC a un comportement très particulier envers la gigue. En contraste par rapport aux autres plateformes, le débit est accru en sa présence. Dès la première intensité (*5 ms*), il est augmenté de plus de 30%. Un analyse plus fine du trafic nous permet d'identifier le principal facteur de cet accroissement. On constate en effet une diminution des IAT, ce qui entraîne une hausse du nombre de paquets par seconde. S'en suit naturellement une hausse du débit. Nous attribuons ce phénomène à une retransmission des paquets arrivés dans le désordre.

3.3.5 Variation des IAT et de la taille du payload face aux contraintes

A présent, nous nous intéressons à deux grandeurs liées au débit des plateformes ; l'IAT et le "*payload*" moyens. L'objectif est de voir leur évolution en fonction des contraintes appliquées.

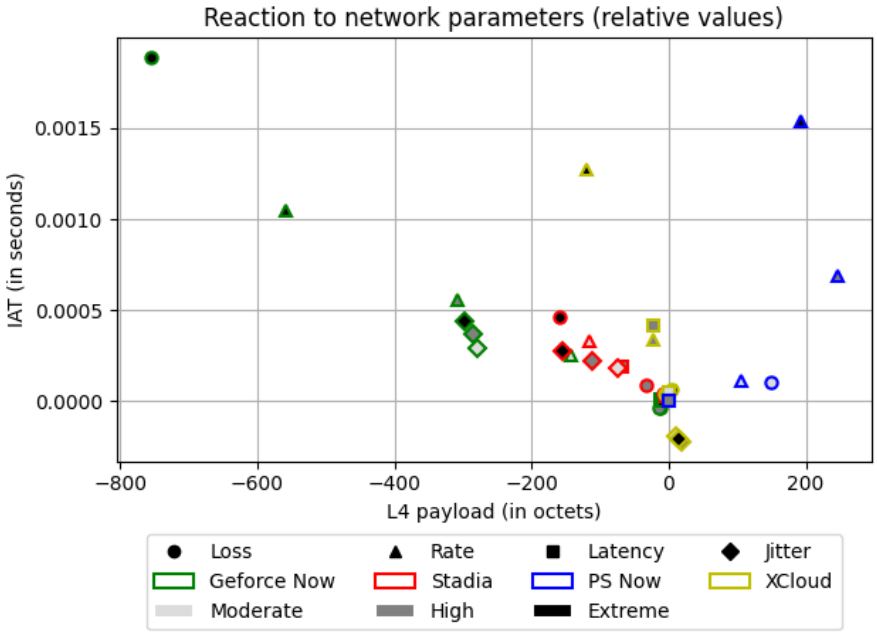


FIGURE 3.7 – Variation relative des IAT et de la taille du Payload UDP selon les contraintes

Sur la Figure 3.7, l'axe des abscisses permet de mesurer l'écart (en octets) entre une valeur et son "*payload*" de référence en l'absence de contrainte. De façon symétrique, l'axe des ordonnées concerne les IATs et s'exprime en secondes. On constate que les plateformes GFN, STD et XC tendent à se diriger vers le coin supérieur gauche. Cela se traduit par une diminution de la taille des paquets et une augmentation des IATs. Chaque plateforme s'y adonne selon sa propre tendance. Nous pouvons souligner que GFN est la plateforme s'adaptant le plus aux contraintes. Les points lui étant associés sont en effet les plus éloignés de l'origine. À l'inverse, PSN adopte un comportement aux antipodes de ce que l'on vient de voir. Dès lors que l'on limite le débit ou que l'on applique un taux de pertes "*Modéré*", la taille des paquets est accrue. Il convient de souligner qu'en présence de gigue, XC diminue les IATs et augmente très légèrement la taille de ses paquets.

3.4 Contraintes durant la session de jeu

Dans la présente section, nous appliquons des contraintes temporaires. Le but est d'étudier l'adaptation des plateformes face à une contrainte inopinée. Une fois la contrainte retirée, nous observons la capacité des plateformes à retrouver un état normal. Pour ce faire, chaque expérimentation sera divisée en trois parties :

1. une première partie, prise sous des conditions réseau normales ;
2. une seconde partie, en conditions réseau *dégradées* ;
3. une troisième partie, constituant un retour en conditions réseau normales.

Chacune de ces parties a une durée de 2 minutes. La première d'entre elles permet d'observer les plateformes dans leur fonctionnement classique. La deuxième permet d'établir les mesures prises pour faire fasse à la perturbation générée. Enfin, la troisième phase est dédiée au *recouvrement* des plateformes. Combien de temps mettent-elles à retrouver un fonctionnement normal après l'abrogation d'une contrainte ? Au même titre que dans la section 3.3, nous allons impacter le pourcentage de pertes, la bande passante disponible, la latence, et enfin la gigue. Pour chacune de ces grandeurs, nous prenons garde de choisir des valeurs permettant à chaque service d'opérer. Les niveaux d'intensité retenus sont les plus élevés eu égard de la contrainte de "*bon*" fonctionnement. Nous entendons par là que chaque session de jeu doit arriver au bout des 6 minutes d'expérience. Les valeurs retenues sont affichées dans la Table 3.2.

	Référence	Contrainte
Pertes	0%	5%
Bande passante	10 Gbps	15 Mbps
Latence	8 ms	50 ms
Gigue	0 ms	2 ms

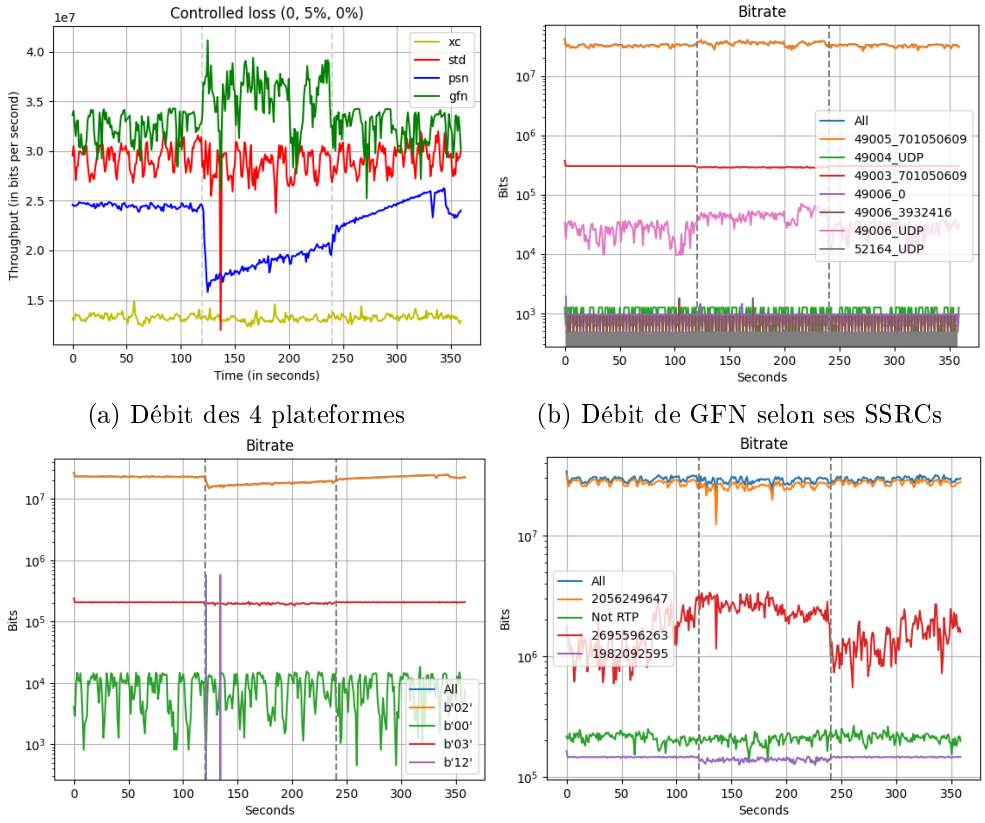
TABLE 3.2 – Paramètres appliqués pour dégrader temporairement les conditions réseau

3.4.1 Perte de paquets

Nous commençons par nous intéresser à l'impact occasionné par 5% de pertes. La Figure 3.8a dépeint l'adaptation des 4 plateformes eu égard de la contrainte adoptée. Les trois autres figures sont dédiées à l'étude des flux des plateformes. Le but est de voir si des flux spécifiques se manifestent en période de perturbation. De surcroît, cette étude permet d'estimer la contribution de chacun des flux au débit global. Nous n'avons pas pu étudier la plateforme XC à ce niveau de détail à cause de son usage de DTLS. De plus, un seul canal UDP est utilisé par la plateforme.

La plateforme GFN se distingue des autres dans la mesure où le *bitrate* est accru dès l'application de la contrainte. Cet accroissment peut s'expliquer de par l'ajout de FEC ou de par une retransmission de données égarées. À l'inverse, STD et XC ne semblent pas être bouleversées par les 5% de pertes. Leur débit reste en effet inchangé au cours de l'expérience. De surcroît, la QoE est restée acceptable malgré les pertes infligées. Cette résilience aux pertes s'explique par l'utilisation de FEC dans les données transmises. Le taux de redondance présent par défaut semble donc suffire à absorber la contrainte. En ce qui concerne PSN, le comportement est tout autre. Le débit est grandement diminué dès l'application des premières pertes. On passe alors d'environ 25 Mbps à environ 16 Mbps. Le débit est ensuite augmenté linéairement. Lors de l'abrogation

de la contrainte, plus de 60 secondes sont encore nécessaires pour recouvrer le *bitrate* initial.



(a) Débit des 4 plateformes

(b) Débit de GFN selon ses SSRCs

(c) Débit de PSN selon ses SSRCs

(d) Débit de STD selon ses SSRCs

FIGURE 3.8 – Perte soudaine de paquets

La Figure 3.8b représente les flux générés par GFN. La plateforme emploie plusieurs canaux UDP, chacun dédié à un média spécifique. Nous les distinguons selon le numéro de port destination. Notons qu'un même canal peut voir passer plusieurs flux différents. Nous rencontrons par exemple du trafic UDP et RTP/UDP en direction du port `49006`. En analysant la figure, on constate qu'un flux est très proche du débit global (orange). Il s'agit très certainement du flux vidéo. Son SSRC est identique à celui du flux représenté en rouge. Ce dernier semble constant, avec un débit se situant aux alentours de 320 Kbps. On peut en déduire qu'il s'agit du flux audio. Le troisième flux le plus important est le trafic UDP destiné au port `49006`. Son débit oscille entre 10 et 70 Kbps. D'après la documentation de Nvidia, le port concerné est destiné à l'envoi et à la réception de commandes [GFN]. Nous apercevons la présence de nombreux flux périodiques au bas de la figure. Leurs débits respectifs sont tous inférieurs à 2 Kbps. Nous ne rentrerons donc pas plus dans le détail car leur contribution au débit global est anecdotique.

La Figure 3.8d représente les flux de STD. Au même titre que GFN, la plateforme utilise le protocole RTP. Du fait qu'un seul canal UDP soit employé, le multiplexage a lieu dans une couche supérieure. C'est par le biais du champ SSRC (RTP) que les flux sont alors différenciés [SCFJ03]. Comme d'ordinaire, le flux vidéo est le flux ayant le plus gros débit. Il est représenté en orange. Le flux le plus stable se situe quant à lui aux alentours de 140 Kbps. On peut en

déduire qu'il s'agit de l'audio. Le flux rouge au débit variable est très certainement dédié à la correction des erreurs de transmission. Si l'on calcule son débit moyen en conditions dégradées (*intervalle* [120; 240]), il se situe à environ 2.6 Mbps. Il est bien plus élevé que lorsqu'aucune contrainte n'est appliquée où son débit moyen est alors de 1.5 Mbps.

Enfin, la Figure 3.8c représente les flux de PSN. A l'instar de STD, PSN utilise un seul canal UDP. Nous ne reconnaissons cependant aucun protocole dans son SDU. D'après Domenico & al., le multiplexage s'effectue grâce au premier octet du *payload* UDP [DDPT+21]. Nous avons ainsi départagé les flux en fonction de cette valeur. Nos observations sont en accord avec les conclusions des auteurs ;

— Le flux `'00'` n'est pas très actif. Il pourrait être dédié à de la signalisation ;
— Le flux `'02'` est dédié à la transmission vidéo (*plus haut bitrate, moyenne de 22 Mbps*) ;
— Le flux `'03'` est dédié à la transmission audio (*bitrate constant à* $\approx$ *240 Kbps*) ;
— Le flux `'12'` serait dédié à l'ajout de FEC ou à de la retransmission ;

Nous constatons que le flux `'12'` subit un pic de presque 600 Kbps dès lors que l'on engendre des pertes. S'en suit un deuxième pic de même intensité une dizaine de secondes plus tard. L'activité de ce flux en conditions réseau altérées rejoint la thèse avancée par Domenico & al.

3.4.2 Limitation de la bande passante

Ici, nous nous intéressons à l'impact occasionné par une baisse soudaine de la bande passante. La contrainte est fixée à 15 Mbps. La figure 3.9 trace le débit de chaque plateforme. Nous ne commenterons pas le débit de XC, du fait que la contrainte se situe au-dessus de son *bitrate* par défaut.

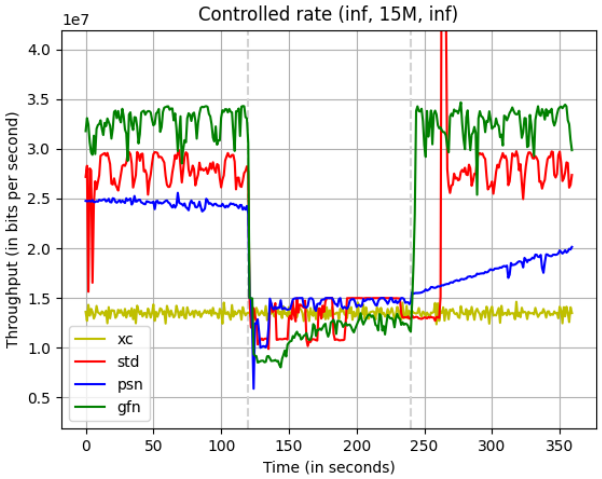
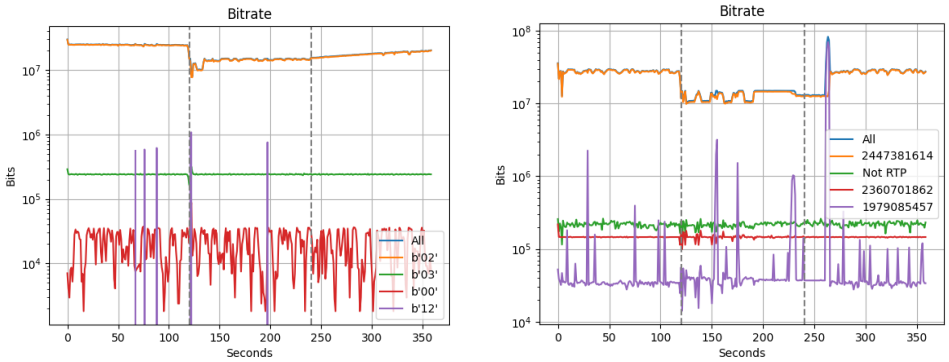


FIGURE 3.9 – Limitation soudaine de la bande passante

Lorsque l'on applique la contrainte sur la bande passante, le débit de GFN passe en-dessous des 10 Mbps. Son *bitrate* est alors constant pendant quelques secondes, avant d'être augmenté progressivement. La croissance semble s'estomper aux alentours des 13 Mbps. Similairement à ce que nous avons vu en section 3.3.2, GFN n'utilise pas toute la bande passante disponible. Une marge de sécurité est ainsi préservée. Dès la révocation de la contrainte, le débit est fortement augmenté. Il ne lui faut d'ailleurs pas plus de 2 secondes pour retrouver sa valeur initiale.

PSN décroît fortement son débit durant les premières secondes de perturbation. La vidéo est alors momentanément stoppée. Cela peut se justifier par une perte de paquets. L'analyse

des flux de la plateforme vient d'ailleurs confirmer cette hypothèse. Le canal '12', d'ordinaire inactif, subit un pic d'environ 1 Mbps à cet instant précis. Comme souligné en section 3.4.1, le flux susvisé est sûrement dédié à la retransmission de données. L'illustration de ce que nous venons de décrire est donnée par la Figure 3.10a. Une dizaine de secondes après le début de la contrainte, le *bitrate* est augmenté jusqu'à atteindre la limite imposée. Il se stabilise alors jusqu'à l'abrogation de la contrainte. La phase de rétablissement est cependant très lente. Après le retour en conditions normales, PSN ne sera pas parvenue à atteindre son débit initial. Le débit mesuré en fin d'expérience en représente environ 80%.



(a) SSRC de PSN (b) SSRC de STD

FIGURE 3.10 – Limitation soudaine de la bande passante : PSN et STD

Dans les 80 secondes suivant l'enclenchement de la contrainte, le *bitrate* de STD varie fortement. Il suit un motif crénelé, alternant entre 11 et 15 Mbps. Cette fluctuation trahit les difficultés d'adaptation de la plateforme. Le débit semble ensuite se stabiliser à 15 Mbps, avant d'être revu à la baisse. Après la suppression de la contrainte, la plateforme met une dizaine de secondes à réagir. S'en suit un pic de grande intensité ($\approx$ *82 Mbps*), sûrement dédié à l'évaluation de la bande passante. Parmi les flux constituant le débit de STD, c'est le flux d'ordinaire le plus '*faible*' qui en est à l'origine. Le flux en question a pour débit moyen 654 Kbps. Lors du pic, son débit atteint les 70 Mbps. Nous pouvons nous rapporter à la Figure 3.10b pour en avoir une illustration. Une fois le pic passé, la plateforme retrouve immédiatement son débit originel preuve que la bande passante disponible a correctement été sondée.

3.4.3 Ajout de latence

Nous étudions à présent l'impact engendré par 50ms de latence supplémentaire. La Figure 3.11 dépeint le débit résultant pour les 4 plateformes étudiées.

Pour GFN et STD, aucun mécanisme d'adaptation ne semble être activé par les délais supplémentaires. On ne constate en effet aucune fluctuation dans leurs débits respectifs. De plus, aucun changement n'est à souligner dans les IATs ou les tailles de paquets. L'expérience de jeu est restée fluide malgré tout. Pendant les deux minutes de perturbation, le débit de PSN est aussi resté constant. Il est cependant diminué une fois la contrainte ôtée, ce qui est difficilement justifiable. Plus d'une minute est nécessaire à la plateforme pour retrouver son débit initial.

Parmi l'ensemble des plateformes étudiées, XC semble être la plus sensible à la latence. C'est d'ailleurs avec cette plateforme que l'expérience de jeu a été la plus désagréable. Dès l'application de la contrainte, la réactivité a été grandement impactée. Certaines commandes semblaient d'ailleurs totalement ignorées. De surcroît, une très forte fluctutation du *bitrate* est à

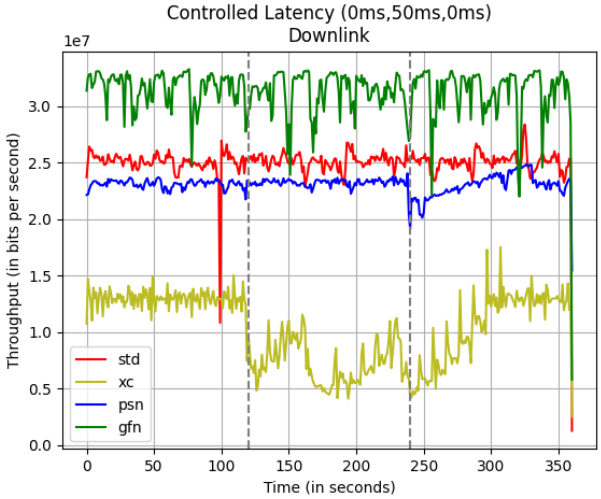


FIGURE 3.11 – Ajout soudain de latence

souligner. Dès l'application de la contrainte, il passe d'environ 14 Mbps à 5 Mbps. Deux tentatives d'accentuation sont néanmoins perpétrées. Pour chacune d'elles, le débit finit par être fortement réduit une fois dépassé les 11 Mbps. Après le retour en conditions normales, une minute sera nécessaire à XC pour retrouver son état original.

3.4.4 Génération de gigue

Nous portons à présent notre attention sur l'impact occasionné par la gigue. La Figure 3.12 retrace le débit des plateformes selon l'intensité de la contrainte. Comme stipulé en début de section, nous introduisons 2ms de gigue pendant deux minutes. La plateforme PSN ne semble pas réagir à la présente perturbation.

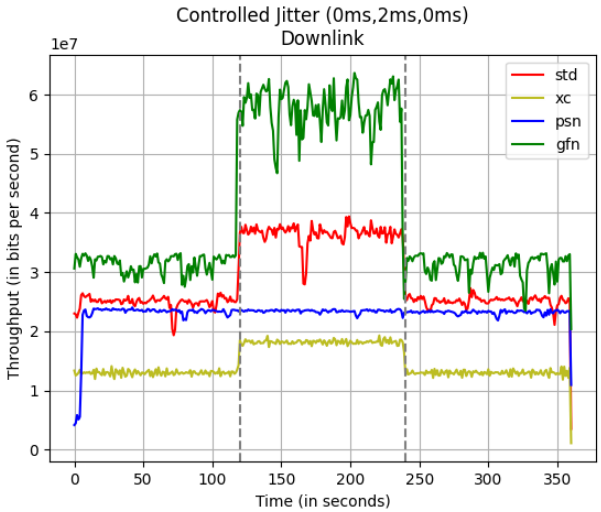


FIGURE 3.12 – Ajout soudain de gigue

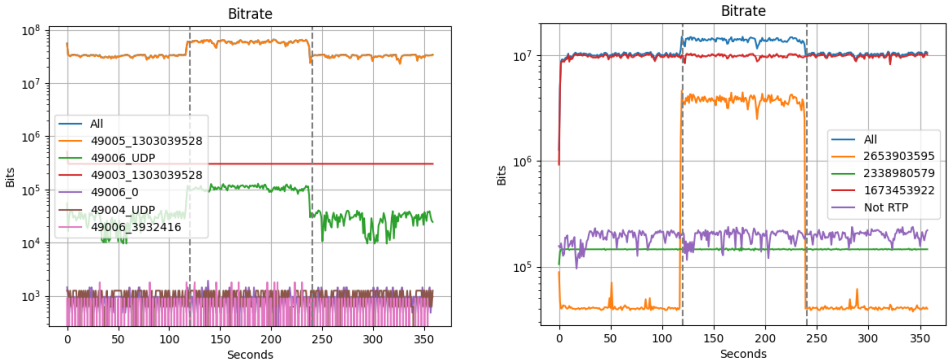
Les trois autres plateformes considérées augmentent fortement leur débit dès l'ajout de gigue. L'augmentation la plus manifeste concerne GFN, atteignant les 60 Mbps. Son débit est alors deux

fois plus important qu'en conditions normales. De façon analogue, STD augmente son *bitrate* de 66% passant de 25 à 37 Mbps.

L'acroissement du débit de XC est moins perceptible que pour GFN et STD. En revanche, XC est la plate-forme la plus touchée dans le sens client-serveur dont le trafic est multiplié par trois sous l'effet de la gigue, passant de 420Kbps à 1200Kbps. Nous supposons que ces différences proviennent de l'utilisation de différents modes de fiabilité au niveau de la couche de transport pour les commandes de jeu. Par exemple, il est possible de configurer les canaux de données de WebRTC dans un mode de livraison fiable et dans l'ordre qui utilise des retransmissions ou en mode non-fiable, qui limite le nombre de retransmissions ou la durée pendant laquelle elles sont autorisées.

Dans tous les cas, nous supposons que ces hausses de débit sont dues à des retransmissions. En effet, l'ajout de gigue impacte l'ordre d'arrivée des paquets. Il est alors essentiel que le récepteur les réordonne. C'est d'ailleurs la raison d'être du *buffer* de gigue. Dans notre cas, sa taille n'est sûrement pas assez grande pour amortir les 2ms de gigue.

Nous nous sommes livrés à une étude supplémentaire pour conforter cette thèse. Nous avons ainsi représenté l'évolution des flux de GFN 3.13a et de STD 3.13b. Dans le cas de GFN, on constate que deux flux sont revus à la hausse après l'application de la contrainte. Son flux vidéo passe d'environ 32.2 Mbps à 59 Mbps. L'autre flux, représenté en vert, subit une augmentation de moindre ampleur. Son débit moyen passe alors de 29 Kbps à 105 Kbps. De plus, la taille des paquets semble être constante. La hausse du débit est donc due à une diminution des IATs, c'est à dire à l'envoi davantage de paquets.
Dans le cas de STD, un seul flux est augmenté lors du déclenchement de la contrainte. Le flux en question n'est pas consacré à la vidéo. On peut ainsi en conclure qu'il permet de retransmettre des données. L'évolution de son débit est massive ; on passe d'une moyenne de 82 Kbps à 3.7 Mbps. Une fois la perturbation terminée, les 3 plateformes retrouvent rapidement leur débit initial.



(a) SSRC de GFN (b) SSRC de STD

FIGURE 3.13 – Ajout soudain de gigue : GFN et STD

3.5 Conclusion

Dans le présent chapitre, nous avons perturbé les conditions réseau par le biais de règles `tc-netem`. Nous avons ainsi pu observer les mécanismes d'adaptation de 4 plateformes de CG. Au cours de nos expériences, nous avons impacté à tour de rôle le taux de pertes, la bande passante disponible, la latence, et enfin la gigue. Nous avons fait varier l'intensité de chacune de

ces grandeurs. Notre étude s'est déroulée selon deux scénarios. Le premier, "*Contraintes initiales*", consiste à appliquer les contraintes avant même le lancement d'une session de jeu. Chacune des plateformes a alors le temps de s'adapter. Pour finir, nous appliquons les contraintes au cours d'une session de jeu. Nous pouvons ainsi observer la réaction des plateformes face à l'ajout et à la suppression d'une contrainte.

Cette étude diffère des autres travaux sur le sujet par le fait de comparer 4 plates-formes CG modernes sous contraintes réseau préalables à la session et au cours de celle-ci. Il s'agit de l'étude la plus complète à ce jour concernant le nombre de plates-formes considérées et la variété des conditions de réseau appliquées, en mettant l'accent sur les différences de capacité d'adaptation de leur trafic.

Au terme de cette étude, nous pouvons souligner que toutes les plateformes tentent d'adapter leur trafic à un moment donné. Leur comportement est régi par un algorithme d'adaptation se situant au niveau applicatif. Celui-ci peut réduire l'utilisation de la bande passante en réduisant la résolution, le taux d'images par seconde, ou le niveau de compression. Des FECs peuvent néanmoins être rajoutés pour limiter l'impact des pertes de paquets.

On remarque que les plateformes ne réagissent pas de la même manière aux contraintes réseau. PSN est par exemple impassible aux délais et à la gigue. On constate une dégradation de sa qualité de service dès lors que ces contraintes sont appliquées. L'absence de réaction n'est donc pas synonyme de prospérité. Certaines plateformes ont du mal à stabiliser leur débit après l'application d'une contrainte. Le phénomène se poursuit parfois même après leur abolition, ce qui impacte d'autant plus la QoE. Dans l'ensemble, GFN semble être la plateforme la plus à même d'adapter son trafic aux contraintes du réseau. Elle préserve de plus un compromis entre stabilité et adaptabilité lorsque cela est nécessaire. Les deux plateformes WebRTC, Stadia et Xcloud, sont très réactives pour adapter leur trafic aux conditions du réseau, mais ont tendance à réagir de manière excessive.

Les expérimentations conduites dans le présent chapitre ont été réalisées sur un réseau à capacité fixe et les perturbations réseau que nous avons engendrées sont de nature synthétique, ce qui aide à la compréhension des mécanismes sous-jacents. Il est d'autant plus intéressant d'étudier les plateformes de CG dans un environnement réseau réaliste mais contraint. Les réseaux à capacité variable, en particulier les réseaux cellulaires, sont sujets à de fortes fluctuations de la bande passante. Ils constituent de plus un cas d'utilisation typique en situation de mobilité. Le prochain chapitre est donc dédié à cette nouvelle étude.

Chapitre 4

Sur réseau à capacité variable

Sommaire

4.1 Introduction 57
4.2 Contraintes spécifiques aux réseaux cellulaires 58
4.3 Testbed 59
4.3.1 Captures des `txops` d'un réseau cellulaire 60
4.3.2 Émulation des capacités d'un réseau cellulaire 61
4.3.3 Acquisition de trafic CG sur réseau cellulaire 62
4.4 Analyse du bitrate 63
4.4.1 Utilisateur fortement mobile 63
4.4.2 Utilisateur statique 64
4.4.3 Stabilité du débit 66
4.4.4 Relation entre débit et qualité vidéo 67
4.5 Analyse de la latence 69
4.6 Conclusion 71
4.6.1 Bilan 71
4.6.2 Réflexions et pistes d'amélioration des réseaux 72

4.1 Introduction

Dans le chapitre précédent, nous avons étudié les mécanismes d'adaptation de 4 plateformes de CG. Le réseau considéré était alors un réseau à capacité fixe. Dans le présent chapitre, nous complétons notre étude en considérant un réseau cellulaire. Ce type de réseau se distingue des réseaux d'accès filaires utilisés jusqu'alors par l'extrême variabilité des communications. Nous pouvons dès lors nous questionner sur le comportement qu'auront ces 4 plateformes.

Nous devons pouvoir mesurer le trafic de STD, GFN, PSN et XC sous exactement les mêmes conditions de réseau mobile. Du fait de la nature de ces réseaux, nous devons ainsi mettre en place un environnement réseau émulé qui soit reproductible. Celui-ci est alimenté par de vrais traces des réseaux cellulaires grâce à *MahiMahi*[19], un outil d'émulation de réseau développé par le Massachusetts Institute of Technology (MIT) qui permet de rejouer le timing et la quantité d'opportunités de transmission fournies par un réseau mobile tel que capturé lors d'une campagne de mesure. Selon les opportunités de transmission dans le sens montant (*resp. descendant*) (traces

19. `http://mahimahi.mit.edu/`

`txops`), l'outil diffère la transmission (*resp. réception*) des données. Dans le cadre de notre projet, Orange nous a fournit plusieurs captures `txops`. Elles nous ont permis d'émuler son réseau mobile de façon réaliste. Chacune de ces captures correspond à une condition réseau particulière.

Une fois les conditions radio émulées, nous nous livrons à une étude de QoS. Cette étude porte sur plusieurs critères. Tout d'abord, nous analysons le débit des plateformes. Nous nous livrons ensuite à une étude sur la latence et le taux de pertes estimés. Pour finir, nous mettons en lien le débit et la QoE. Celui-ci étant lié à la résolution et au *framerate*, nous verrons comment il impacte directement l'expérience de jeu.

4.2 Contraintes spécifiques aux réseaux cellulaires

Dans la présente section, nous nous intéressons aux délais inhérents aux réseaux cellulaires. Dans [TLL+04], Tan & al. proposent `LTE-VR`, une solution destinée à faire de la réalité virtuelle (VR) sur réseau cellulaire. Les auteurs y soulignent que les opérations de signalement constituent une proportion non négligeable de la latence totale. Leur solution, prenant place côté client, permet de limiter ces délais. Nous présentons seulement les problèmes soulevés par les auteurs.

Un des avantages du Cloud Gaming est que les smartphones constituent un client léger permettant des sessions de jeu en situation de mobilité sans être contraint pas les modestes capacités graphiques des équipements. Il peut être ainsi amené à passer d'une station de base à une autre (BS). Ce phénomène porte le nom de "*handover*". Les réseaux d'accès 3G permettent de maintenir une connexion simultanée sur deux BS. Le service est alors assuré continûment ; on parle de "*soft handover*". Ce n'est malheureusement pas supporté par la technologie LTE. On parle alors de "*hard handover*" ; le client se déconnecte d'une BS avant de se connecter à une autre. Le processus peut d'ailleurs durer plus de 70 ms pour l'opérateur AT&T. La Figure 4.1 représente les différentes étapes constituant un *Handover*.

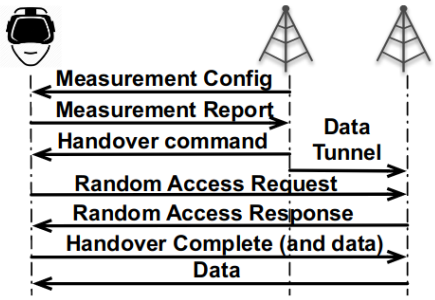


FIGURE 4.1 – Représentation du phénomène de *Handover* [TLL+04]

Un autre phénomène, intrinsèquement lié au "*handover*", engendre aussi des délais colossaux. En principe, l'UE (*User Equipment*) accuse réception des données reçues à la station de base. Ces messages émanent du protocole "*Radio Link Control*" (RLC) se situant sur la couche 2 du client. Dans l'optique de réduire la charge inhérente à la signalisation, RLC n'acquitte pas toutes les données reçues. L'envoi d'un acquittement (`R-ACK`) est déclenché par un *timer* ou par une requête de la BS (*polling*). Au moment du *handover*, les derniers paquets envoyées par l'ancienne station de base peuvent ne pas être acquittés. Le cas échéant, ils sont ré-envoyés par la nouvelle BS, générant alors des délais supplémentaires. On parle de "*HOL Blocking*", un phénomène pouvant représenter entre 30 et 45 ms de délais supplémentaires.

Nous pointons à présent une autre lacune, associée au mécanisme d'acquittement. Lors de la réception d'une donnée corrompue, l'UE en informe immédiatement la BS. Côté client, c'est la couche "*Medium Access Control*" (MAC) qui s'en charge. Dès la détection d'une donnée corrompue, un acquittement négatif (`M-NACK`) est envoyé. Un seul bit distingue cependant le `M-NACK` de l'acquittement (`M-ACK`). Si au cours du trajet ce bit se retrouve inversé (*bit flipping*), la BS ne procèdera pas à la retransmission. La suite des données est donc envoyée vers le client, et sera mise en attente par la couche RLC, chargée de livrer les paquets de façon ordonnée. Au terme d'un certain délai $T_{reorder}$, la couche RLC demande la retransmission de la donnée manquante. D'après [LLM+09], le phénomène de *bit flipping* surviendrait selon une probabilité d'environ 10^{-3} à 10^{-4}. Dans le cas d'une transmission descendante continue, les risques sont néanmoins accrus. Le phénomène peut occasionner jusqu'à 65 ms de délais, selon la valeur du *timer* $T_{reorder}$.

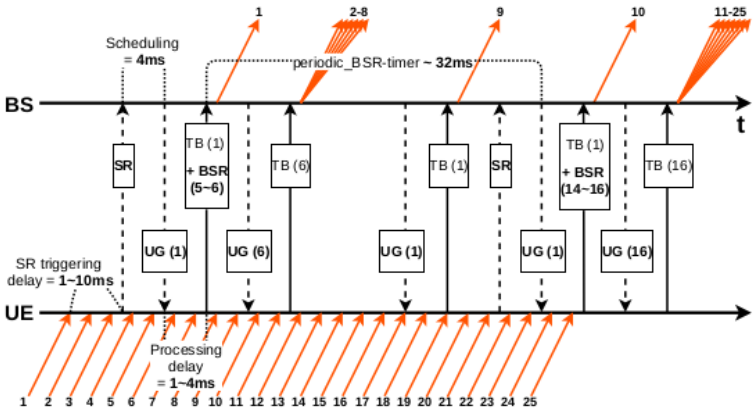


FIGURE 4.2 – Demande de transmission depuis l'UE [RJLHF22]

Nous avons jusqu'à présent abordé les différents délais intervenant dans le sens descendant. Le sens opposé est aussi sujet à des lenteurs. Dès lors que l'UE a des données à émettre, il doit effectuer une demande de transmission auprès de la BS. Il lui envoie alors une "*Scheduling Request*" (SR), signifiant la présence de données dans son *buffer* d'envoi. Une fois que la BS a reçu ce message, elle consacre des ressources à l'UE. Un "*Uplink Grant*" (UG) est alors envoyé pour autoriser la transmission de données au prochain *slot*. L'UE peut mettre à profit ces ressources pour informer la BS de la taille de son *buffer*. Il génère ainsi des *Buffer Status Report* (BSR), qu'il envoie périodiquement. La Figure 4.2 représente les mécanismes de demande et d'octroi d'opportunités de transmission. D'après Tan & al., plus de 80% de paquets seraient bloqués dans le *buffer* d'envoi. Dans [RJLHF22], Ronteix-Jacquet & al. proposent de prédire la taille du *buffer* depuis la BS. Cela leur permet d'allouer des ressources proactivement, sans avoir à attendre de demande de l'UE. Le procédé limite la latence et la gigue dans le sens montant.

Enfin les réseaux cellulaires sont connus pour avoir des buffers assez importants faisant courir un risque de délais importants en cas de congestion. Pour toutes ces raisons, ils constituent un environnement particulièrement difficile pour un service à très haut débit et faible latence.

4.3 Testbed

Dans cette section, nous expliquons comment procéder à des captures `txops` représentant des opportunités de transmission sur réseau cellulaire. Malheureusement, les traces de réseau mobile référencées dans le dépôt public de Mahimahi sont trop anciennes (2016) et peu représentatives

des réseaux actuels. Dans le cadre du projet ANR MOSAICO, nos collègues d'Orange ont créé de nouvelles traces sur leur réseau 4G.

Dans un second temps, nous nous intéresserons à l'émulation d'un réseau cellulaire. À cet effet, nous dépeindrons le fonctionnement de *MahiMahi*. Pour finir, nous présenterons les captures CG réalisées par le biais de cet outil. Ces captures consistent en des fichiers `pcap`, dont l'analyse nous permet d'estimer la latence et les pertes occasionnées.

4.3.1 Captures des `txops` d'un réseau cellulaire

Afin d'assurer la reproductibilité des expériences sur réseau cellulaire, Winstein & al. ont mis au point "*Saturator*"[20] [WSB04]. Cet outil a pour but de caractériser un tel réseau. Pour ce faire, les auteurs surchargent les files d'attente montante et descendante d'une même liaison radio. De part et d'autre, chaque émetteur respecte une fenêtre de N paquets *en vol*[21]. La variable N est ajustée en fonction du RTT. Un RTT trop important est à bannir, car des pertes et/ou un engorgement du lien peuvent en découler. Au contraire, une valeur trop faible peut signifier que le lien est sous-utilisé. Du fait que les deux directions soient simultanément congestionnées, les acquittements se retrouveraient retardés. Pour se prémunir de cet effet de bord, *Saturator* achemine les acquittements par le biais d'un autre canal. Nous pouvons le constater sur la figure 4.3. On y voit qu'un deuxième téléphone est dédié au transfert des acquittements. Pour mener à bien nos expériences, nous avons donc eu recours à 4 appareils ;

— Un serveur (*Ubuntu 18.04*) pour saturer la liaison descendante ;
— Un laptop (*Ubuntu 20.04*) pour saturer la liaison montante ;
— Deux mobiles (*Xiami MI 10 5G*) pour envoyer des données (*resp. des acquittements*) ;

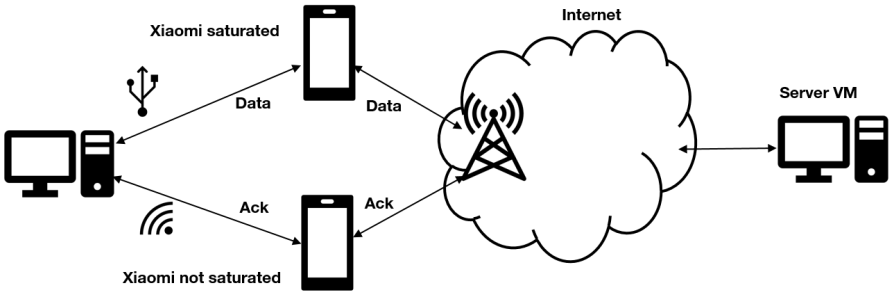


FIGURE 4.3 – Fonctionnement de l'outil *Saturator*

L'échange des Acks entre l'ordinateur portable se fait via la liaison radio non encombrée de l'UE connecté au Wi-Fi. Le serveur Linux est hébergé dans un Datacenter surprovisionné. Pendant les captures txops, l'ordinateur portable envoie des paquets de données horodatés au serveur qui sont utilisés pour calculer le RTT à réception des Acks correspondants. Cette valeur RTT est utilisée pour ajuster la fenêtre de congestion du client. Le même processus est appliqué côté serveur, qui peut donc modifier sa fenêtre de congestion en conséquence et ainsi saturer la liaison radio de la station de base vers l'UE.

À l'issue de l'expérimentation, les mesures servent à générer deux fichiers `txops`. Chaque fichier est affilié à une direction spécifique. Un fichier contient autant de lignes que d'opportunités de transmission d'un paquet complet à la taille du MTU. La valeur associée à chaque ligne caractérise le moment où la transmission peut avoir lieu. Elle est donnée par la différence (*en*

20. `https://github.com/keithw/multisend/`
21. Paquets réseau n'ayant pas été acquittés

ms) avec le début de la capture. Si plusieurs paquets de taille MTU peuvent être envoyés dans la même milliseconde, plusieurs lignes sont présentes dans le fichier avec le même horodatage. Si aucun paquet ne peut être envoyé pendant quelques millisecondes, il y a un intervalle de temps correspondant entre les horodatages de deux lignes successives. Ces fichiers txops peuvent ensuite être utilisés par l'outil LinkShell de Mahimahi pour émuler un réseau cellulaire à la capacité variable dans le temps présentant les mêmes opportunités de transmission que capturé lors de la mesure.

Dans le cadre du projet ANR MOSAICO, nos collègues d'Orange ont réalisé au total 6 captures `txops`, prises dans des conditions différentes. Toutes ont été réalisées en janvier 2022, sur le réseau cellulaire d'Orange. Nous avons tout d'abord considéré un scénario statique. L'UE est donc stationnaire, et ne devrait pas subir de *handover*. Les 5 captures ainsi réalisées (*Traces* $[A-E]$) se distinguent de par la qualité du signal radio. L'étude des `txops` associés nous permet de constater leur disparité quant à la bande passante disponible. Une synthèse est donnée dans l'Annexe B. D'autres métriques viennent appuyer leur divergence. Nous pouvons pour cela nous référer au Tableau 4.1. Les première et deuxième colonnes représentent la moyenne (*resp. la variance*) de la bande passante. La troisième colonne désigne la probabilité que la bande passante disponible dépasse 20 Mbps ce qui laisse assez de place pour les flux des plateformes. Les trois métriques susvisées ont une résolution de 1 seconde. Pour finir, nous nous intéressons aux périodes *de vide*. Ce sont des périodes où aucune transmission ne peut avoir lieu, fréquentes sur les réseaux cellulaires. Nous comptons le nombre de périodes sans opportunité de transmission pendant 33 ms, de sorte à ce qu'elles engendrent la perte d'une frame vidéo, ce qui a un impact direct sur la QoE. Dans le dernier scénario, nous considérons un UE en déplacement. Pour ce faire, les mesures ont été perpétrées depuis une voiture circulant à 110 km/h. Nous avons remarqué que la qualité du signal est plus élevée à proximité des zones urbaines. Cela fait sens, car les opérateurs privilégient les zones ayant le plus de densité de population. En nous référant au Tableau 4.1, nous constatons l'importante proportion de périodes de vide. Nous dépassons en effet les 17% en situation de mobilité rapide. Nous pouvons dès lors nous attendre à subir d'importantes pertes dans le cadre de ce scénario.

Dans une démarche de science reproductible, les fichiers `txops` sont mis à disposition de la communauté. Les 6 captures sont accessibles via ce lien ; `https://cloud-gaming-traces.lhs.loria.fr`. Chacune d'elle s'étend sur une dizaine de minutes.

TABLE 4.1 – Caractéristiques des traces `txops` sur le réseau mobile d'Orange

	Moyenne (Mbps)	Variance (Mbps)	% du temps >= 20Mbps	% de vides de 33ms
Trace A	45.0	7.4	99.3	5.8
Trace B	78.8	20.3	99.5	8.0
Trace C	141.6	49.5	97.4	0.4
Trace D	173.3	43.4	99.8	3.8
Trace E	230.0	48.7	99.9	2.6
Highway	43.7	31.5	70.5	17.7

4.3.2 Émulation des capacités d'un réseau cellulaire

Dans [NSD+15], Netravali & al. présentent *mahimahi*. L'outil en question a été développé pour évaluer des protocoles réseau. Au total, quatre "*Shell*" sont proposés ;

— *RecordShell* : enregistre le trafic HTTP en le faisant passer par un proxy ;

— *ReplayShell* : rejoue le trafic HTTP. Chaque serveur contacté est émulé sur le proxy ;

— *DelayShell* : permet de retarder les paquets réseau selon une valeur pré-établie ;
— *LinkShell* : émule un réseau selon des opportunités de transmission ;

Chacun de ces "*Shell*" est dissocié de la machine hôte. Pour y parvenir, *Mahimahi* a recours à des *namespaces* réseau. Chacun de ces *namespaces* a sa propre configuration, de sorte à ne pas impacter la machine hôte. Au même titre que pour un conteneur, les processus y étant lancés n'ont pas de vue sur les ressources sous-jacentes. Grâce au cloisonnement, il est possible de faire fonctionner plusieurs "*Shell*" sur une même machine. Les auteurs soulignent qu'il est d'ailleurs possible de les combiner. Dans le cas de notre étude, nous retiendrons seulement le *LinkShell*.

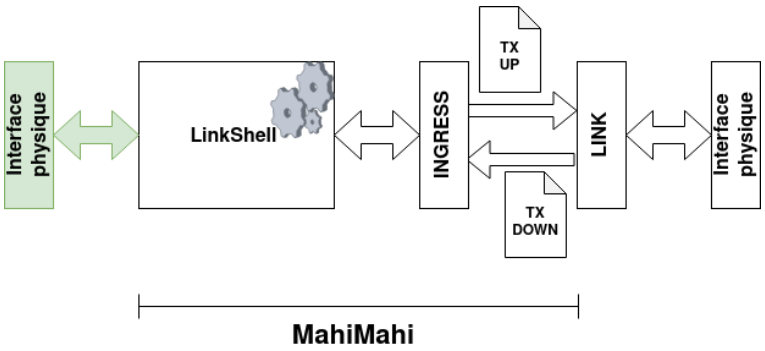


FIGURE 4.4 – Placement de *Mahimahi* sur un routeur intermédiaire

Le "*LinkShell*" ne possède qu'une interface virtuelle. Cette dernière permet de faire le lien avec la machine physique. Pour pouvoir évaluer une application, il convient ainsi de la lancer depuis le "*LinkShell*". Son trafic transite alors par l'interface `ingress`, qui diffuse les données selon ses opportunités de transmission. Un schéma est donné dans la Figure 4.4. Une fois diffusées, les données apparaissent à la machine hôte. Elles lui parviennent par le biais d'une interface virtuelle `Link`. Des règles `ip-tables` permettent ensuite de les acheminer vers une interface physique. Le trafic descendant suit la même logique.

Dans notre cas d'étude, aucune application ne peut être exécutée depuis le `LinkShell`. Nous avons donc manipulé le *namespace* pour lui associer une interface physique. L'interface susvisée est représentée en vert sur la Figure 4.4. Elle permet de faire la jonction avec le client CG. De l'autre côté, la deuxième interface assure la connectivité avec le serveur.

4.3.3 Acquisition de trafic CG sur réseau cellulaire

Au cours de nos expérimentations, nous procédons à deux captures *Wireshark*. Toutes deux sont réalisées en même temps, mais prennent place de part et d'autre du *bottleneck*. La première est réalisée sur l'interface `Link`, tandis que l'autre est effectuée depuis le `Linkshell`. Le procédé permet de calculer la latence et le taux de pertes en comparant les deux captures qui se situent ainsi avant et après le *bottleneck*. Un débit trop important va tout d'abord induire de la latence. Une fois les *buffers* remplis, les données seront *droppées*. On peut alors constater la présence de pertes. Dans le cadre de nos expérimentations, nous utilisons des files classiques de type "*droptail*". Le principe est simple ; les données sont mises en attente dès lors que le réseau est congestionné. Elles suivent le principe FIFO ; les premières données mises en queue sont les premières à en sortir. Nous fixons la taille du *buffer* descendant à 1875Ko. Il correspond à la file d'attente de la station de base. Dans le sens opposé, le *buffer* dispose de 250 Ko. Il se situe au niveau de l'UE et contient les données en attente d'envoi. Ces deux grandeurs ont été définies par Orange de sorte à coïncider avec un réseau concret.

Avant d'étudier le trafic CG sur réseau cellulaire, il convient de l'observer dans des conditions réseau favorables. La base de référence ainsi constituée établit le débit et la latence moyenne des 4 plateformes constatée sur un réseau FttH sans perturbation. Ses valeurs sont présentées dans le Tableau 4.2.

La mesure de la latence de bout-à-bout diffère selon les plateformes. Pour ce qui est de STD et XC, nous nous basons sur les rapports internes de `WebRTC`. Pour GFN et PSN, nous nous appuyons sur des mesures de *ping* `ICMP`. Par rapport à l'étude menée au Chapitre 3, on constate une différence dans le débit moyen de XC. Il passe de 12.6Mbps à 16Mbps du fait d'une meilleure résolution disponible. La plateforme a en effet rendu possible le *streaming* en 1080p entre mai 2021 et mai 2022.

TABLE 4.2 – Statistiques nominales des 4 plateformes sur réseau FttH

	Geforce Now	Playstation Now	Stadia	Xcloud
Average bitrate (Mbps)	31.1	23.9	29.6	16.0
Latency (ms)	6.7±0.3	6.6±0.3	10.2±0.5	12.6±0.7

4.4 Analyse du bitrate

Nous étudions ici le débit des plateformes de CG en conditions réseau cellulaires. Nous représentons ainsi la moyenne et la variance du débit de chacune d'elles en Table 4.3 que nous comparons ensuite à deux autres valeurs en Table 4.4 :

— la bande passante disponible, afin d'estimer le pourcentage d'utilisation du lien ;

— leur débit effectif, tel qu'observé dans des conditions réseau *favorables* (*base de référence*) ;

Afin d'enrichir notre étude, nous représentons, de surcroît, l'évolution des délais et du taux de pertes.

TABLE 4.3 – Moyenne et variance du débit de chaque plateforme selon les traces `txops`

	Geforce Now		PS Now		Stadia		Xcloud	
	Mean thpt	Std thpt	Mean thpt	Std thpt	Mean thpt	Std thpt	Mean thpt	Std thpt
Highway	15.16	11.07	6.79	7.66	8.25	4.32	6.76	3.64
Trace A	28.84	4.53	21.69	2.00	12.21	2.19	10.77	2.37
Trace B	28.95	3.40	23.80	1.12	28.41	4.19	14.34	2.35
Trace C	30.71	2.53	23.72	0.29	30.01	3.93	13.36	0.53
Trace D	29.80	4.02	23.71	1.16	27.30	7.92	12.69	3.69
Trace E	30.06	2.44	22.17	2.09	27.69	6.23	11.81	3.12

4.4.1 Utilisateur fortement mobile

Les résultats obtenus en situation de mobilité (*Highway*) se distinguent nettement des autres scénarios. Il s'agit de loin la trace la plus difficile pour les plateformes. L'expérience de jeu est globalement insatisfaisante quelle que soit la plateforme, avec des périodes de très faible débit et d'images figées dues aux pertes pour chacune d'entre elles. Nous dispensons en Figure 4.5 l'évolution de leur débit. Le fond en bleu clair représente la bande passante disponible. On constate en effet que par moments, aucune donnée ne peut être transmise.

TABLE 4.4 – Utilisation de la bande passante (%*avlb*) / Proportion du débit de référence %*ref*

	Geforce Now		PS Now		Stadia		Xcloud	
	% avlb	% ref	% avlb	% ref	% avlb	% ref	% avlb	% ref
Highway	34.5	48.8	15.4	28.4	18.8	27.9	15.4	42.3
Trace A	65.4	92.8	49.2	90.9	27.7	41.3	24.4	67.4
Trace B	38.4	93.1	31.6	99.7	37.7	96.1	19.0	89.7
Trace C	19.2	98.8	14.9	99.4	19.3	101.5	8.4	83.6
Trace D	17.3	95.9	13.8	99.4	15.8	92.3	7.4	79.4
Trace E	12.8	96.7	9.5	92.9	11.8	93.6	5.0	73.9

On constate également que STD et XC, les deux plateformes basées sur WebRTC obtiennent les meilleurs résultats dans ces conditions : leur algorithme de contrôle de congestion s'adapte rapidement à l'augmentation et à la diminution de la bande passante, ce qui entraîne un écart type maîtrisé. Leurs courbes respectives suivent d'ailleurs une même tendance. Au même titre que STD et XC, GFN est très réactive. Elle se distingue cependant au travers de plusieurs aspects. Tout d'abord, son débit est bien plus important que celui des autres plateformes. L'analyse de la Table 4.3 est d'ailleurs conforme à cette observation. De surcroît, GFN est la plateforme la moins sensible aux diminutions de la bande passante. Son débit est certes réduit, mais de façon bien plus modérée que pour les autres. En conséquence, ses délais tendent à croître à chaque baisse des opportunités de transmission. Cette hausse reste néanmoins négligeable.
Pour finir, la plateforme PSN est en difficulté tout du long de la capture. L'irrégularité de la trace *Highway* porte grandement atteinte à l'expérience de jeu. Au cours du chapitre 3, nous avons souligné qu'elle ne réagissait pas aux délais. Nous avons par conséquent supposé que son algorithme de contrôle de congestion était *basé pertes*. Tant que la plateforme ne subit pas de pertes, elle accumule une quantité de données déraisonnable dans le *buffer* et ses délais continuent de croître au détriment de la QoE, jusqu'à ce qu'il y ait une opportunité de transférer tous les paquets accumulés dans la queue. Après une amélioration des conditions radio, PSN peut être longue à rehausser son débit. Aux alentours de la seconde 525, son débit est très faible par rapport à la bande passante disponible. Il faudra environ une minute à la plateforme pour le revoir à la hausse. L'accroissement se fait cependant promptement ; on constate une hausse de plus de 25 Mbps en l'espace de quelques secondes.

4.4.2 Utilisateur statique

Parlons à présent des autres scénarios ; *Traces* $[A - E]$. Le comportement des plateformes se rapporche de l'étude menée en section 3.4. Les deux plateformes basées `WebRTC` (STD et XC) réagissent démesurément aux baisses de la bande passante. Leurs débits respectifs sont par conséquent trop modestes par rapport à leurs débits de référence. Une baisse excessive du débit nuit à la QoE du fait qu'elle se répercute sur la qualité vidéo. En guise d'exemple, STD reste en 720p durant toute la trace `A`. La quantité de bande passante disponible aurait toutefois permis de diffuser en 1080p. Des pics périodiques viennent toutefois accentuer le débit moyen de la plateforme. Il en est de même pour la trace `B`. Au cours du précédent chapitre, nous avons associé ces pics à de la retransmission vidéo (Section 3.3).
Dès lors que les délais augmentent, XC baisse fortement son débit. Nous pouvons observer le phénomène sur la Figure 4.6b. Une fois le pic de latence passé, la plateforme tarde à retrouver

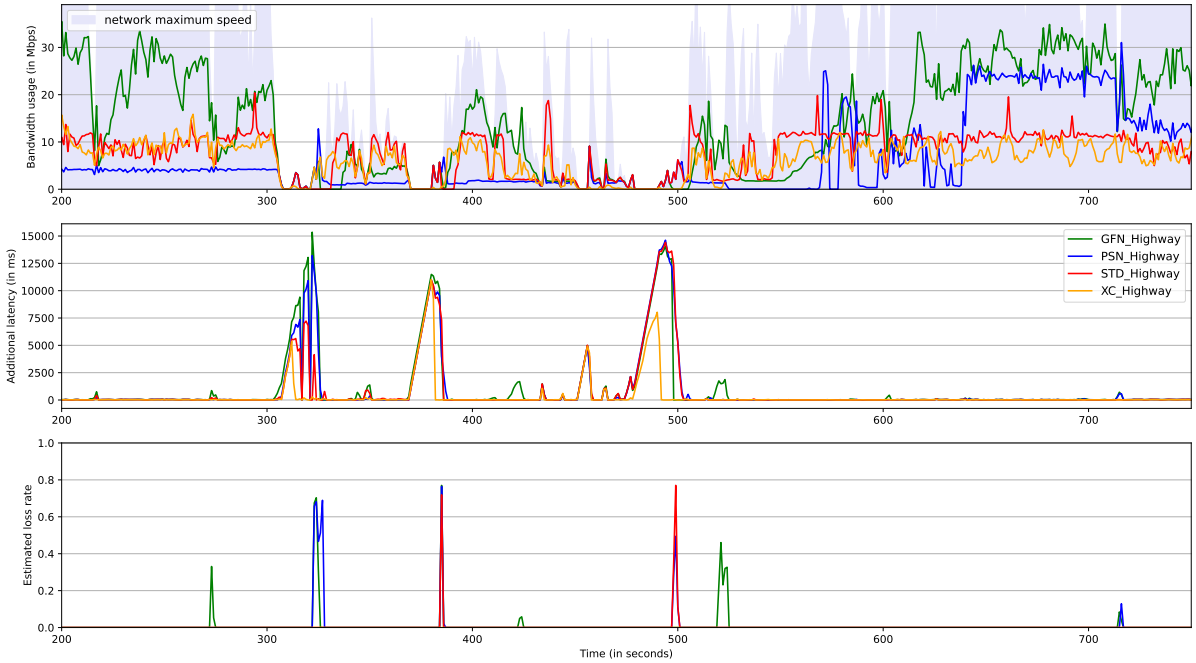


FIGURE 4.5 – Analyse des plateformes sous un scénario de mobilité (*Highway*)

son débit initial. Cette observation concorde avec les résultats présentés en Section 3.4.3 durant l'expérience de latence contrôlée : une réaction rapide pour contenir l'augmentation de la latence, mais un temps de récupération lent. PSN se situe à l'opposé de XC. Comme précisé auparavant, aucune action n'est enclenchée pour pallier les hausses de délais. L'étude de PSN sous les conditions `B`, `C` et `D` le confirme. Nous pouvons nous reporter sur la Figure 4.6b , où une partie de la trace `D` est utilisée pour émuler le réseau. On constate une brusque baisse de la bande passante aux alentours de la seconde 545. PSN est la seule plateforme à maintenir son débit constant, malgré la latence additionnelle. Cette absence de réaction fait que son débit est le plus stable des 4 plateformes. Après une amélioration des conditions radio, PSN augmente son débit de façon lente et progressive. Cette stratégie d'adaptation est néanmoins opposée aux observations faites avec la trace `highway`. On peut donc conjecturer que PSN a deux stratégies de récupération. Les conditions réseau inhérentes à la trace `highway` pourraient avoir déclenché une stratégie plus "*brutale*". Il est intéressant de comparer le débit de PSN à son débit de référence donné dans le Tableau 4.4. On y voit que pour les traces $[A-E]$, le débit de PSN représente plus de 90% de son débit de référence. La proportion est d'autant plus importante pour les traces `B`, `C` et `D`, où l'on dépasse les 99%. Son débit est donc très proche de ce que nous avons mesuré en Table 4.2. Au même titre que PSN, GFN se rapproche de son débit de référence pour les traces $[A-E]$. Eu égard des ressources disponibles, la plateforme cherche à maximiser son débit. En nous reportant à la Table 4.3, on constate que son débit moyen dépasse toujours 28 Mbps. L'écart avec les autres plateformes est d'autant plus flagrant pour la trace `A`. Le débit de STD et XC représente alors moins de 45% de celui de GFN. Dès lors que les conditions se dégradent, GFN adapte modérément son débit. Nous pouvons d'ailleurs le voir sur la Figure 4.6b. Le débit de STD et de XC est plus fortement impacté que celui de GFN. Malgré tout, les délais de GFN ne dépassent pas ceux des deux plateformes susvisées. Ceci souligne une sur-réaction de la part de STD et XC.

(a) Faible bande passante

(b) Baisse soudaine de la bande passante

FIGURE 4.6 – Comportement des plateformes de CG sous différentes conditions

4.4.3 Stabilité du débit

Nous nous intéressons ici à la stabilité du débit des plateformes. Nous avons ainsi mis au point une métrique permettant de l'estimer. Nous commençons par diviser une session de jeu en fenêtres de 30 secondes. Chaque fenêtre est composée de 30 sous-fenêtres de 1 seconde. Pour toute fenêtre w_i, nous calculons le débit moyen $\bar{x}_{i,j}$; $j \in [1;30]$ de chacune de ses sous-fenêtres. Nous procédons ensuite à un calcul de variance portant sur les 30 moyennes obtenues. Nous normalisons cette variance en la divisant par le débit moyen $\bar{x}_i$ de la fenêtre courante. $\bar{x}_i$ est dérivé des 30 débits moyens, tel qu'apparaissant sur l'équation 4.1. Au final, la mesure de stabilité d'une capture est donnée par la sommes des variances normalisées. Les résultats sont inscrits dans le Tableau 4.5. Il convient de souligner que les 120 premières secondes de chaque session sont ignorées. Bien entendu, toutes les valeurs renseignées sont calculées à partir du même nombre de fenêtres.

$$\bar{x}_i = \frac{\sum_{j=1}^{30} \bar{x}_{i,j}}{30} \tag{4.1}$$

TABLE 4.5 – Somme des variances normalisées des débits de chaque plateforme

	Geforce Now	Playstation Now	Stadia	Xcloud
Highway	10.82	10.95	10.22	9.6
Trace A	1.54	0.77	2.47	2.80
Trace B	1.64	0.40	1.97	2.11
Trace C	1.45	0.67	1.55	0.77
Trace D	1.55	0.49	2.68	2.59
Trace E	1.16	0.81	1.82	2.25

L'étude de la Table 4.5 nous permet de dresser plusieurs constatations. Premièrement, nous pouvons souligner que PSN a de loin le débit le plus stable. Si l'on fait abstraction de la trace *Highway*, toutes ses valeurs sont inférieures à 1. Le débit de GFN semble aussi plutôt stable, avec des valeurs ne dépassant jamais 1.7. Les plateformes STD et XC sont généralement moins

stables que les deux autres. Certaines de leurs mesures vont jusqu'à dépasser 2.5 sur les traces $[A; E]$. On peut expliquer ces valeurs de par une forte réactivité des deux plateformes ce qui leur permet d'offrir la meilleure expérience de jeu en cas de fortes turbulences. En situation de mobilité (*Highway*), la métrique dépasse 10 pour chacune des plateformes considérées. Ces variations sont néanmoins "normales", eu égard de la trace considérée. Il est primordial de savoir rapidement adapter son débit pour faire face à l'instabilité du réseau.

Il convient de noter qu'aucune corrélation forte ne peut être déduite en comparant la bande passante moyenne disponible pour une trace du Tableau 4.1 et l'écart au débit de référence 4.4 ce qui signifie que les réactions sont principalement déclenchées par des événements spécifiques se produisant dans chaque trace telles que des micro-coupures et qui deviennent invisibles à une plus grande échelle.

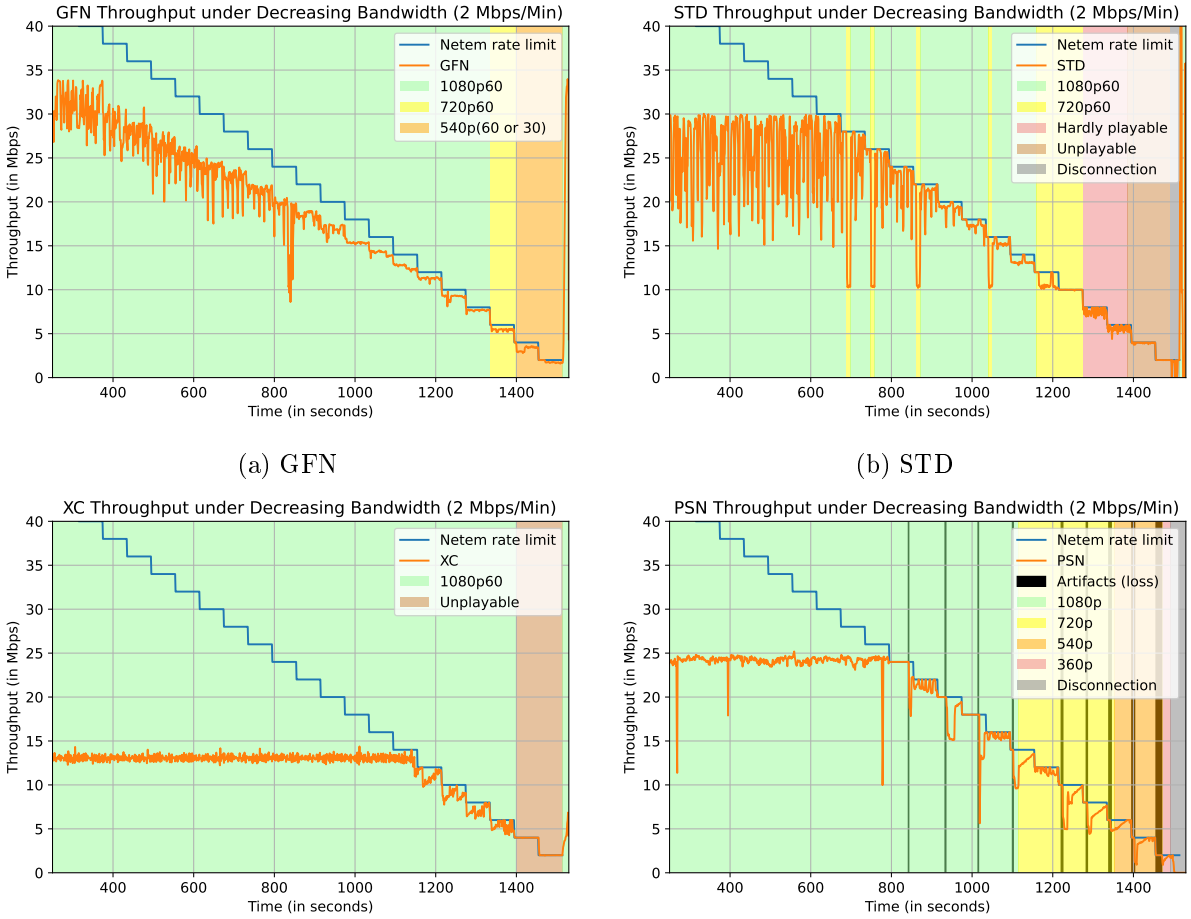
4.4.4 Relation entre débit et qualité vidéo

Pour parfaire l'analyse du bitrate menée dans cette section, nous terminons par étudier son rapport avec la qualité de la vidéo transmise. Cela permet de donner un second éclairage orienté QoS aux valeurs précédemment relevées. En effet, la baisse du débit nécessaire afin de limiter délais et pertes en cas de congestion doit être réalisée avec parcimonie car elle impacte négativement la qualité vidéo. L'enjeu étant, pour les plateformes, de trouver un compromis entre délais/pertes et qualité vidéo.

Pour établir ce lien, nous limitons par paliers progressifs la bande passante disponible pour chaque plateforme et notons les caractéristiques du flux vidéo généré. Durant les 15 premières secondes de l'expérimentation, aucune contrainte n'est appliquée, puis une baisse progressive de la bande passante est appliquée à raison de de 2 Mbps toutes les minutes, passant de 50 Mbps à 2 Mbps.La Figure 4.7 associe à chacune des plateformes les résultats de nos expériences. La courbe en bleu trace la bande passante disponible et celle en orange le débit effectif de la plateforme. S'y rajoute un code couleur, nous renseignant sur la résolution et le *framerate* associés. Ces deux indicateurs de QoS sont récupérés grâce aux statistiques des plateformes. Le procédé diffère quelque peu pour PSN, du fait que ces informations ne soient pas disponibles. Pour pallier le problème, nous procédons à des captures d'écran. Afin d'éviter tout biais, nous utilisons un codec vidéo "*sans pertes*". Nous nous livrons ensuite à une analyse manuelle de ces captures vidéo pour en déceler les caractéristiques. Nous pouvons dès à présent commenter nos résultats.

Lorsque la bande passante passe en-dessous de 40 Mbps, GFN commence par diminuer le paramètre de quantification (quantization) qui a pour conséquence d'augmenter le niveau de compression de la vidéo. Cela lui permet de diminuer son débit sans pour autant impacter la résolution ou le *framerate*. Il faut atteindre les 6 Mbps pour constater un changement de résolution ; on passe alors en 720p. Une minute plus tard, lorsque la bande passante est réduite à 4 Mbps, on constate deux changement notoires. Premièrement, le *framerate* passe à 30 fps. La résolution est de surcroît réduite à 540p. Lors de l'application de la contrainte, la plateforme a semblé perdre en réactivité. Les mesures prises vont toutefois permettre d'atteindre à nouveau une certaine fluidité. Une trentaine de secondes après l'application de la contrainte, le *framerate* reprend sa valeur initiale. Il faut attendre la prochaine réduction de bande passante pour rester en `540p@30fps`. La plateforme est encore très réactive.

La plateforme STD est sujette à de nombreux changements de résolution. Avant que la bande passante ne soit limitée à 12 Mbps, on en dénombre 4. Chacun de ces changements s'étale sur une durée de 10 secondes. La résolution est alors réduite, passant de 1080p à 720p. Dans tous les cas, le *framerate* est maintenu à une valeur de 60 fps. Dès lors que la bande passante se situe



(a) GFN

(b) STD

(c) XC

(d) PSN

FIGURE 4.7 – Lien entre débit et qualité vidéo

en-dessous de 12 Mbps, la résolution est fixée à 720p. L'expérience de jeu se retrouve de plus en plus impactée au fur et à mesure que le restriction s'intensifie. À partir de 8 Mbps, des *frames* floues apparaissent périodiquement à l'écran et le joueur s'expose à un fort manque de réactivité. La plateforme semble alors ne pas réagir aux commandes de jeu. Au terme de quelques minutes, STD met un terme à la connexion.

Il n'est pas aisé de comparer XC aux autres plateformes. Son débit initial est en effet bien plus faible que les autres. Il faut ainsi attendre la fin de l'expérience pour observer les mécanismes d'adaptation de la plateforme. Il n'y a qu'à partir de 12 Mbps que XC doit revoir son débit à la baisse. Sur toute la durée de l'expérience, ni la résolution ni le *framerate* ne changent. La plateforme va cependant jouer sur le taux de compression pour adapter son débit. Dès que la bande passante est limitée à 4 Mbps, l'image a tendance à se figer. Le manque de réactivité inhérent nuit fortement à l'expérience de jeu.

Pour finir, PSN est la plateforme subissant les plus grosses chutes de débit. Ces chutes se situent généralement aux moments où la bande passante est abaissée. Elles s'accompagnent d'artéfacts visuels dans la vidéo. Nous représentons ces événements par des barres noires verticales. En accord avec nos conclusions initiales, nous attribuons ces baisses de débit à une perte excessive de paquets. Comme nous pouvons le voir sur la Figure 4.7d, PSN passe par plusieurs étapes

pour diminuer son débit. La résolution est tout d'abord abaissée à 720p pour 14 Mbps de bande passante. Au fur et à mesure que la contrainte s'intensifie, la résolution passera respectivement de 540p à 360p. Peu après la limitation à 2 Mbps, PSN met fin à la session de jeu. Au cours de notre analyse, nous avons remarqué une certaine instabilité du *framerate*. Il subit des changements périodiques, alternant entre 60 fps et 30 fps durant toute l'expérience. En moyenne, il est maintenu à 60 fps pendant environ 75 secondes, et passe à 30 fps pendant les 90 secondes suivantes.

4.5 Analyse de la latence

Le CG étant un trafic à contrainte temps réel, la QoE est fortement sensible à la latence. Un débit supérieur à bande passante disponible induira ainsi des délais additionnels de séjour en file d'attente que nous étudions dans cette section.

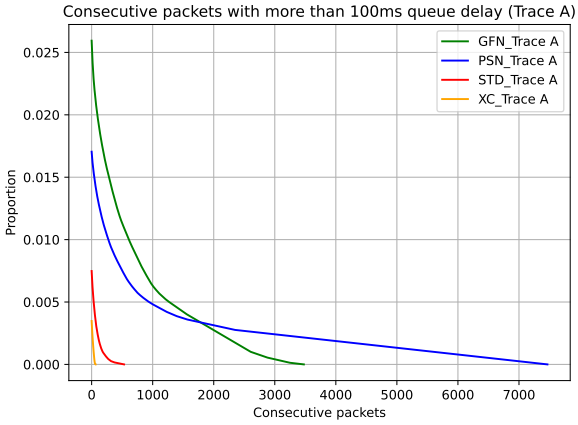
TABLE 4.6 – Pourcentage de paquets UDP dont les délais dépassent 10,20,50 et 100ms

	Geforce Now				PlayStation Now				Stadia				Xcloud			
Highway	51.6	42.8	29.3	14.9	45.5	38.4	27.0	15.0	37.2	28.6	18.4	8.7	27.7	17.9	9.7	4.0
Trace A	52.4	30.4	10.0	2.6	31.2	18.5	6.4	1.7	25.4	14.3	4.5	0.7	22.5	10.9	3.2	0.3
Trace B	26.3	17.4	13.6	9.2	18.5	15.1	11.9	7.8	18.3	12.6	9.7	6.5	7.2	2.8	2.0	1.5
Trace C	8.3	3.1	1.4	1.0	6.2	3.2	1.9	1.4	6.7	2.4	1.0	0.5	5.7	2.0	1.0	0.5
Trace D	8.9	4.9	3.1	2.2	8.4	5.7	4.1	3.0	6.1	2.8	1.4	0.9	4.0	1.5	0.6	0.4
Trace E	4.1	3.1	2.4	1.5	4.5	3.8	3.0	2.0	2.9	2.1	1.7	1.1	1.4	0.8	0.7	0.6
Délais	>10 ms	>20 ms	>50 ms	>100 ms	>10 ms	>20 ms	>50 ms	>100 ms	>10 ms	>20 ms	>50 ms	>100 ms	>10 ms	>20 ms	>50 ms	>100 ms

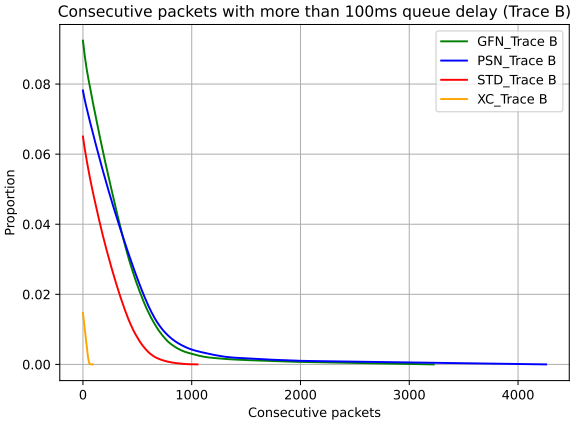
Le Tableau 4.6 liste le pourcentage de paquets dont les délais dépassent certains seuils. Nous retenons ces 4 valeurs ; 10, 20, 50 et 100 ms. Les pourcentages associés sont affichés de gauche à droite. Il convient de noter que chaque délai désigne le temps passé dans la file d'attente du lien descendant du Linkshell Mahimahi, comme décrit dans la section 4.3.3.

On remarque tout d'abord que STD et XC présentent les délais les plus faibles. Cela est en concordance avec ce que nous avons observé en section 4.4. Du fait que ces deux plateformes soient très réactives, la congestion reste maîtrisée. Si nous comparons ces deux plateformes entre elles, nous remarquons que XC a des délais encore plus faibles. Plusieurs raisons viennent justifier cette observation. Tout d'abord, le débit initial de XC est plus faible que celui de STD. Nous pouvons nous reporter à la Table 4.2 pour le vérifier. XC aura donc moins tendance à saturer le lien, et donc à rajouter des délais supplémentaires. Deuxièmement, XC a été conçu pour fonctionner sur *smartphone*. Un effort a donc probablement été réalisé pour fonctionner sur réseaux cellulaires. À l'inverse, PSN et GFN semblent être un peu plus en difficulté. Si l'on se réfère à la trace *Highway*, plus du tiers de leurs paquets dépassent les 20ms de délais. Cette proportion se situe aux alentours de 25% pour STD et XC. L'écart est d'autant plus perceptible pour 100ms de délais supplémentaires. Un paquet a alors deux fois plus de chances de dépasser ce seuil pour PSN et GFN. L'absence de réactivité de PSN engendre des délais dès lors que la bande passante diminue. Pour ce qui est de GFN, on a vu que son débit est baissé avec parcimonie. La Figure 4.6b est d'ailleurs en adéquation avec cette observation. On observe une brusque chute des opportunités de transmission aux alentours de la seconde 545. PSN ne réagit pas à cette détérioration, comme nous pouvions nous y attendre. GFN va quant à elle baisser son débit, mais de façon bien plus modeste que STD et XC. Ses délais sont par conséquent plus importants. Les délais de PSN semblent convenables sur la trace *Highway*, en dépit des déficiences de la plateforme. D'après la Table 4.6, elle semble même surpasser GFN. Ces résultats sont néanmoins à prendre avec précaution. Si l'on considère la Figure 4.5, on constate aisément que le débit de PSN est parfois dérisoire. Le taux de pertes colossal subit par la plateforme en est à l'origine. Entre les

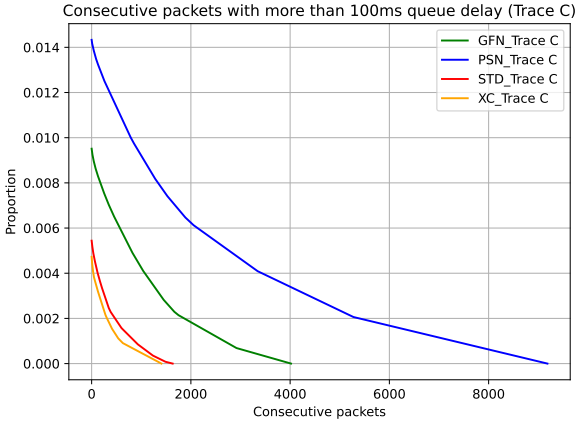
secondes 525 et 560, son débit est presque nul. La QoE se retrouve dès lors fortement altérée. Au cours des traces $[C - E]$, GFN semble surpasser PSN sur les aspects "*temps réel*". La proportion de paquets dépassant les 20ms de délais est systématiquement plus faible pour GFN.



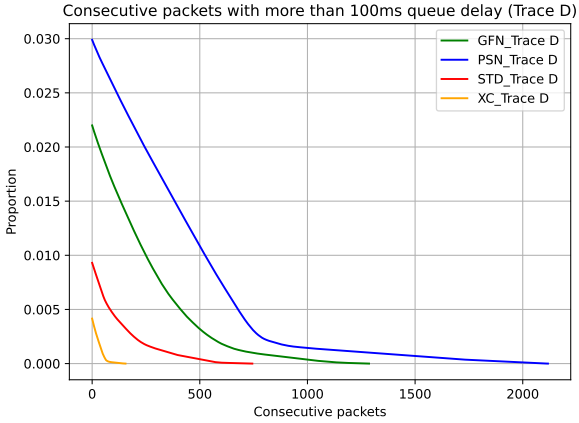
(a) Trace A



(b) Trace B



(c) Trace C



(d) Trace D

FIGURE 4.8 – Distribution cumulée des paquets avec plus de 100ms de délais

Nous avons jusqu'alors considéré la proportion de paquets dépassant certains délais. Nous nous intéressons à présent à la durée des pics de latence. Pour ce faire, nous commençons par nous fixer un seuil. Dans leur étude, Raaen & al. évoquent un seuil emblématique de 100ms [REG14]. Des délais supplémentaires importuneraient grandement l'expérience de jeu. Nous allons donc comptabiliser le nombre de paquets consécutifs dépassant ce seuil. La Figure 4.8 présente leur distribution cumulée. La plateforme XC est inégalable ; très peu de paquets consécutifs dépassent les 100ms. STD se place en seconde position. Les résultats des deux plateformes sont très similaires pour la Trace C (Figure 4.8c). Malgré son fort débit initial (voir Tableau 4.2), STD parvient à limiter ses délais. Il est rare que plus de 1000 paquets consécutifs dépassent le seuil. PSN et GFN sont les plateformes pour lesquelles la métrique considérée est la moins bonne. Mise à part pour la trace `A` et de petits intervalles de paquets, GFN fonctionne mieux que PSN. Elle ne peut cependant pas rivaliser avec STD et XC. L'absence d'adaptation de PSN ne va d'ailleurs pas pour arranger les choses. Cette dernière présente les pires résultats. L'écart avec les autres

plateformes est d'autant plus prononcé pour les traces C et D. Cependant, PSN peut présenter ponctuellement de meilleurs résultats que GFN (Figures 4.8a et 4.8b). Ces propos sont toutefois à nuancer car PSN dépasse le seuil fatidique de 100 ms sur de plus longues périodes.

4.6 Conclusion

4.6.1 Bilan

Dans ce chapitre, nous avons analysé la réaction de quatre plateformes CG (*GeForce Now, Stadia, Playstation Now, Xbox Cloud Gaming*) à différentes contraintes de réseau cellulaire. Grâce à des traces `txops` et au *Linkshell* MahiMahi, nous avons été en mesure d'émuler de tels réseaux. Au même titre que dans le Chapitre 3, les plateformes se sont distinguées de par leur stratégie d'adaptation. Nous avons d'ailleurs pu dégager quelques grandes tendances.

Dans l'ensemble, GFN semble être celle ayant trouvé le meilleur compromis entre débit binaire élevé, stabilité et adaptabilité. Bien que n'étant pas conçue pour les réseaux cellulaires, elle parvient tout de même à adapter son trafic s'il n'y a pas trop de perturbations. Son algorithme de contrôle de congestion n'est pas des plus réactifs, mais semble assez bien adapté dans la plupart des cas. Lorsque la qualité du réseau chute soudainement, les utilisateurs peuvent subir des périodes où les contrôles manquent de réactivité. S'ensuit une phase de récupération plus longue, durant laquelle la qualité vidéo est réduite. Sa logique semble réagir de manière mesurée aux petites perturbations, mais reste prudente au cas où d'autres perturbations pourraient encore survenir. Son utilisation de la bande passante est en revanche la meilleure de toutes. On peut par conséquent profiter d'une meilleure qualité vidéo eu égard des circonstances. Une brusque diminution de la bande passante induira une baisse modérée du débit. Cela est préférable pour la QoE, évitant ainsi de fortes fluctuations de la qualité vidéo. À l'inverse, les deux plateformes WebRTC, STD et XC, ont un débit très réactif. On a vu qu'elles réagissent parfois de manière excessive, réduisant exagérément leur qualité vidéo. La moindre dégradation radio influence grandement leur débit. C'est particulièrement vrai pour STD qui essaie d'offrir la meilleure qualité vidéo. Cela peut s'avérer contre-productif en raison des changements intermittents de résolution ou de qualité dans des conditions instables. Leur forte réactivité permet toutefois de limiter leurs délais et ces deux plateformes, en terme de latence, affichent les meilleurs résultats. Enfin, PSN ne semble pas du tout destiné à être utilisé sur les réseaux cellulaires. La plateforme se distingue de par son insensibilité aux délais émanant d'une congestion du réseau. Le service continue ainsi comme si de rien n'était, et seule la perte de paquets est prise en compte. Cela peut fonctionner lorsque les conditions réseau sont bonnes. Dans le cas d'un réseau *perturbé*, l'expérience de jeu ne peut pas être satisfaisante. L'utilisateur s'expose à de fortes dégradations vidéo avant que le débit ne soit diminué. Au terme de notre étude de QoS, nous avions relevé la présence d'artéfacts à chaque amenuisement du débit. Après une amélioration des conditions réseau, PSN peut être aussi très longue à ré-adapter son débit.

Nos expériences montrent que les services CG ne se soucient parfois pas de la latence et/ou n'arrivent pas à exploiter la bande passante permise par les réseaux cellulaires 4G/5G. Ils peuvent être mis à mal par les instabilités du réseau (chutes soudaines de bande passante et déconnexions temporaires). Somme toute, les mécanismes d'adaptation étudiés tardent à réagir aux fluctuations réseau, ce qui rend souvent le service difficilement jouable. Ceci est compréhensible car en l'absence de mécanisme niveau réseau, la latence ne peut être gérée que de bout en bout sur la base du meilleur effort. Cela exacerbe les différences entre les applications, qui doivent mettre en œuvre le CCA et les mécanismes d'adaptation. Il en découle une dégradation de la QoE, liée à la hausse des délais et/ou à la baisse de la qualité vidéo, voire des pertes handicapantes. Il existe

cependant plusieurs pistes possibles d'amélioration.

4.6.2 Réflexions et pistes d'amélioration des réseaux

Tout d'abord, on peut remarquer que les métriques mises en avant par les FAI (Fournisseurs d'Accès à Internet) pour promouvoir leur réseau cellulaire sont inadaptées pour de tels services. Dans le cas du cloud gaming, il est en effet insuffisant de se limiter à la bande passante moyenne. Cette métrique devrait être complétée a minima par un critère de stabilité. Si nous prenons pour exemple la `Trace A`, la bande passante moyenne se situe à 45 Mbps. Une telle valeur dépasse le débit de référence des 4 plateformes considérées. Malgré tout, le débit effectif de chaque plateforme est bien inférieur à ce qui a été observé sur un réseau FttH. Pour STD, il en représente seulement 41.3%. Il conviendrait aussi de prendre en compte les délais intrinsèques du réseau mobile.

Ensuite au niveau radio, nous pourrions être tentés de garantir un certain débit (GBR). Le mécanisme normalisé permettant de s'acquitter de la tâche porte le nom de QCI ; "*QoS Class Identifier*". Il permet de faire de la différentiation de service. Ainsi, tous les flux ayant les mêmes contraintes de QoS transiteront par un même canal (*porteuse*). À chaque QCI sont associées des contraintes spécifiques. On distingue ;

— Un type ; débit garanti (*GBR*) ou non-garanti (*Non-GBR*) ;
— Un budget de latence par paquet ;
— Le taux de pertes maximum autorisé ;

Les flux GBR seront plus prioritaires que les autres. Dans [ZSGJ19], Zhang & al. montrent qu'une porteuse dédiée peut fortement limiter la latence. Garantir un débit minimum peut cependant s'avérer pernicieux. Lors de conditions réseau dégradées, les flux *Non-GBR* seraient alors annihilés au profit des autres, ce qui explique peut être pourquoi ces mécanismes standardisés sont encore peu utilisés aujourd'hui.

Enfin, au delà des seuls aspects radio, les réseaux cellulaires sont mal adaptés aux applications "*faible latence*". L'usage de gros buffers en périphérie du réseau (*edge*) en est la preuve [GN11]. Leur utilisation permet de maximiser le taux d'utilisation de la bande passante et de mieux supporter le trafic de type "*bursty*" au prix d'une hausse importante des délais en cas de congestion, même modérée. De plus, les CCAs basés pertes ont tendance à opprimer les CCAs orientés délais dans un tel réseau. Une taille importante de *buffer* retarde en effet les pertes de paquet [XC22a]. Un CCA orienté *délais* réagira à la hausse des délais, survenant bien avant l'apparition de pertes. Il est évident qu'une application à contrainte de faible latence a intérêt à opter pour un CCA prenant en compte les délais. Pour une telle application, de plus petits buffers seraient ainsi préférables. Bien que le procédé l'assujettirait à des pertes de paquets, les délais seraient alors limités.

Nous nous trouvons face à des objectifs en apparence contradictoires :

— Maximiser l'utilisation de la bande passante ;
— Limiter les délais ;
— Permettre une cohabitation équitable entre les CCA orientés délais et ceux orientés pertes.

À l'issue de l'état de l'art mené dans le chapitre 2 sur les mécanismes permettant de maîtriser la latence due au réseau, nous avions évoqué l'AQM DualPI2 [ADSB+19]. Pour rappel, celui-ci est composé de deux files, dont une file de type "*best effort*" et une file *faible latence*, ce qui règle le problème de cohabitation. La file faible latence (L4S) est réservée au trafic ECN *scalable*. Le routage se fait selon le positionnement des bits ECN dans l'en-tête IP. ECN permet aux équipements réseau de signaler la congestion de façon anticipée et précise. Nous évaluerons

l'avantage de cette approche au cours du Chapitre 7.

L'autre piste évoquée est l'utilisation d'une discipline de file d'attente pouvant gérer des priorités entre classes de trafic, comme HTB. Les flux de type cloud gaming pourraient ainsi constituer une classe prioritaire. Ils transiteraient par une plus petite file, du fait de leur contrainte de latence. La troisième et dernière partie de la thèse s'intéresse ainsi au transport optimisé du trafic Cloud Gaming. Le prochain chapitre est dédié à son identification dans le réseau. Nous présentons ensuite quelques solutions de déploiement, s'appuyant sur les nouvelles technologies de programmabilité réseau. Le Chapitre 7 se base sur l'ensemble de ces travaux pour appliquer un traitement différencié au CG.

Troisième partie

Transport optimisé du trafic Cloud Gaming

Chapitre 5

Identification du trafic Cloud Gaming

Sommaire

5.1 Introduction 77
5.2 Méthodes de classification des flux réseau 78
5.3 Création et exploitation des jeux de données 81
5.3.1 Données brutes 81
5.3.2 Extraction des caractéristiques 82
5.4 Modèles 83
5.4.1 Base de référence 83
5.4.2 Modèles d'apprentissage automatique supervisé 84
5.5 Étude des hyper-paramètres 85
5.5.1 Taille de la fenêtre temporelle 86
5.5.2 Base de référence 87
5.5.3 Decision Tree et Random Forest 88
5.6 Évaluation des modèles 89
5.6.1 Traces CG *normales* 89
5.6.2 Traces CG *perturbées* 90
5.7 Généralisation à d'autres jeux et plateformes 91
5.7.1 Un modèle non supervisé : USAD 91
5.7.2 Comparaison 92
5.8 Conclusion 93

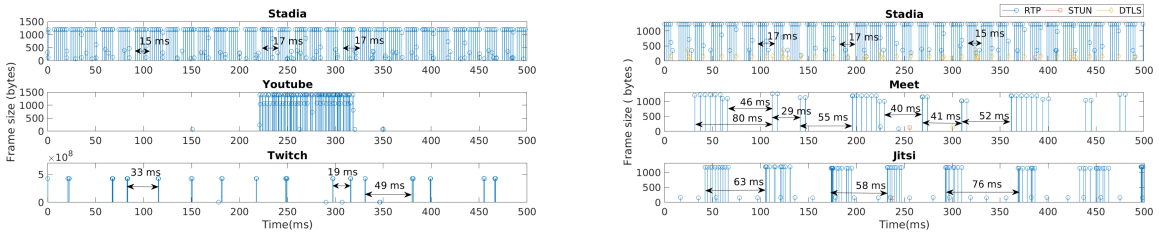
5.1 Introduction

Dans ce chapitre, nous nous intéressons à la reconnaissance du trafic Cloud Gaming dans le réseau. Comme nous l'avons mentionné précédemment, le transport de ce type de trafic constitue un défi du fait de sa nature très haut débit et de son besoin de faible latence. Dans la précédente partie, à travers l'étude de quatre plateformes CG commerciales, nous avons pu mettre en évidence leur difficulté à exploiter au mieux le réseau et à maintenir une bonne qualité de service quand les conditions de réseau sont dégradées, par exemple dans un contexte de réseau cellulaire.

La reconnaissance d'un tel trafic doit nous permettre de constituer une classe de service spécifique afin d'ouvrir la voie à de l'ingénierie de trafic appropriée. Typiquement, l'utilisation d'une discipline de file d'attente comme HTB pouvant gérer plusieurs classes et files d'attentes permettrait de résoudre le problème de *bufferbloat*, principale cause de latence en périphérie de

réseau dû à des files d'attente surdimensionnées, ou encore celui de la cohabitation équitable entre des CCA basés délais et ceux basés pertes.

Notre hypothèse de travail est la suivante : de par ses caractéristiques intrinsèques, le trafic de Cloud Gaming devrait pouvoir être identifié dans le réseau en calculant certaines statistiques sur les flux. En effet, si on le compare à première vue au trafic de certaines applications voisines, le trafic CG semble se distinguer sur plusieurs points. Il est par exemple, à résolution équivalente, de plus haut débit que du streaming classique (du fait d'une compression moindre). Ses données sont envoyées avec une régularité multiple du framerate dans un sens et de la fréquence d'échantillonage des commandes dans l'autre. Un flux montant spécifique est dédié aux commandes de jeu. Dans un autre registre, il est beaucoup plus haut débit qu'un flux de visio-conférence du fait d'une résolution et d'un framerate supérieurs et, là encore, il dispose d'un flux montant différent (commandes de jeu vs webcam).



(a) Trafic de streaming

(b) Trafic de visio-conférence

FIGURE 5.1 – Comparaison des motifs temporels du trafic CG avec d'autres applications [CB22]

L'enjeu de ce classificateur est multiple. Il faut qu'il soit assez fiable pour garantir que tous les flux CG soient bien identifiés afin de constituer une classe de trafic homogène et respecter ainsi la neutralité d'Internet [22]. D'autre part, il faut qu'une implantation efficiente soit possible pour permettre de traiter un volume de trafic conséquent compatible avec les contraintes opérationnelles d'un opérateur, au minimum de 10Gb/s pour un déploiement en périphérie de réseau. Ce second point sera traité dans le chapitre 6.

Pour commencer, nous nous intéresserons aux méthodes d'apprentissage automatique ("*Machine Learning*") ayant été appliquées avec succès à la classification de flux réseau en fonction de leurs caractéristiques. Nous présentons ensuite le jeu de données que nous avons constitué pour l'entrainement et le test avant de présenter les modèles que nous étudions. Après avoir configuré leurs hyper-paramètres, nous les évaluons avec rigueur. Nous introduisons enfin un dernier modèle permettant une meilleure généralisation.

5.2 Méthodes de classification des flux réseau

La classification de trafic réseau est très importante pour les FAIs. Cela leur permet de mieux s'adapter aux exigences de sécurité et/ou de QoS des différents flux. Il existe plusieurs méthodes permettant de s'acquitter de la tâche. Une approche naïve consiste à se limiter aux numéros de port. Le procédé est simple ; on fait correspondre un numéro de port à une application sous-jacente. La correspondance peut se faire par le biais des registres IANA [Aut99]. Cependant, de plus en plus d'applications font usage de numéros de port dynamiques, ou encore vont passer des services de nature différente sur un même port. Ce type de classification, bien que simple et rapide, est ainsi rendu désuet. Une autre façon de faire consiste à étudier les données bit par

22. `https://www.arcep.fr/nos-sujets/la-neutralite-du-net.html`

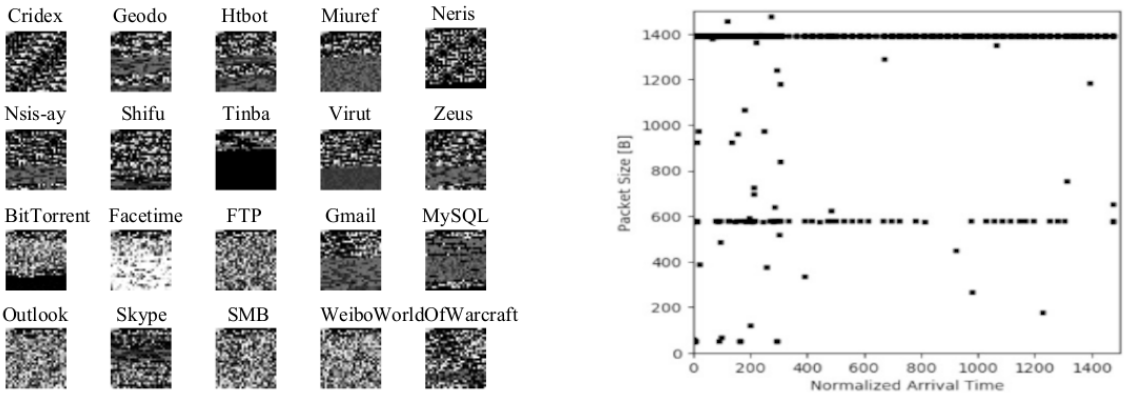
bit. La classification se fait alors en reconnaissant des séquences d'octets ; on parle de *signature* [LPYK15]. Un inconvénient notoire est que le moindre changement protocolaire portera atteinte aux résultats de la classification. De surcroît, l'accès au *payload* des paquets est une opération complexe. La banalisation du chiffrement (plus de 85% du trafic aujourd'hui) restreint l'usage de cette méthode. Dans [VvvD15], Velan & al. présentent plusieurs méthodes permettant de classifier du trafic chiffré. En plus de la méthode susvisée, ils mentionnent un procédé se basant sur les "*caractéristiques*" des flux. Cela consiste à apprendre les motifs (*patterns*) propres à un type de communication, en se basant sur des statistiques telles que les tailles des paquets ou leurs temps d'inter-arrivée. On est ainsi en mesure de reconnaître une catégorie de service, comme par exemple le *streaming vidéo*. Dhote & al. soulignent que la reconnaissance précise d'une application est encore ardue [DAD15]. Dans notre cas, il est seulement question de reconnaître le trafic CG en tant que classe de service afin d'être conforme à la neutralité du net. Nous ne chercherons donc pas à faire une classification plus fine.

Dans [BFC+19], Brissaud & al. cherchent à détecter une action utilisateur sur un site web en dépit du chiffrement par défaut d'HTTP/2. Pour ce faire, ils ont recours à une apprentissage supervisé qui associe des flux aux mots-clé en étant à l'origine. Les flux sont définis par un ensemble de caractéristiques, qui serviront à entraîner le classificateur. Y figurent des statistiques sur les tailles de paquets, mais aussi sur l'évolution du trafic descendant. Le classificateur utilisé est un *Random Forest* (RF), aboutissant à une *accuracy* globale de 94-99% selon les sites web. Les articles [ASV+21, YFJR21] ont recours à des modèles plus complexes. On parle alors de *Deep Learning* (DL), ou d'apprentissage profond. Dans [ASV+21], Akbari & al. utilisent deux types de caractéristiques. Ils considèrent premièrement l'évolution des tailles des paquets et des temps d'inter-arrivées. Cette évolution se modélise par le biais de séries temporelles (TS). En plus de cela, les auteurs considèrent les données brutes des paquets. Elles sont extraites à l'initiation de la connexion, lors du *handshake* TLS. À chaque type de caractéristique est associé un classificateur spécifique. Les séries temporelles servent d'entrée à un réseau neuronal de type LSTM. À l'inverse, les données brutes sont traitées par un réseau de neurones convolutionnel (CNN), une architecture très fréquente dans la reconnaissance d'images. La classification se fait en combinant la sortie des deux réseaux neuronaux. Dans [YFJR21], Yang & al. adoptent une approche à peu près similaire. La différence réside dans le fait que les deux classificateurs sont placés en série. Un premier classificateur se basant sur de l'apprentissage supervisé est chargé d'identifier une application. Un label est alors apposé à la TS représentant un flux. S'en suit un deuxième modèle, cette fois-ci basé sur de l'apprentissage non supervisé. Le but est de décréter si la TS en entrée appartient ou non à une application connue. Selon le résultat, le label sera maintenu ou invalidé. Deux remarques intéressantes sont pointées par les auteurs :

1. pour de la classification de trafic connu, ML et DL ont des résultats à peu près similaires ;
2. le DL est plus à même de détecter une anomalie que le ML.

Dans le paragraphe précédent, nous avons évoqué un réseau de type CNN. Ce type de réseau est d'ordinaire destiné à de la reconnaissance d'images. Les articles [Wei20, WZZ+17, SS21] utilisent cette orientation pour se livrer à de l'apprentissage par représentation. Le principe consiste à symboliser un flux par une image, qui servira d'entrée au modèle. La classification d'un flux consiste alors à comparer deux images. Dans [Wei20, WZZ+17] les données brutes d'un flux sont transformées en une image 2D. Une représentation est illustrée par la Figure 5.2a. La particularité de [WZZ+17] et de ne pas se limiter à un seul modèle de CNN. Les auteurs ont en effet étudié deux scénarios. Le premier allie un classificateur binaire à deux classificateurs 10-classes. Le classificateur binaire doit détecter si le trafic étudié est légitime ou non. Selon le résultat obtenu, un second classificateur identifiera l'application (*resp. le malware*) concerné. Le

deuxième scénario utilise un seul CNN à 20 classes. On y retrouve les 10 malwares et les 10 applications considérées. L'*accuracy* moyenne des 3 classificateurs étudiés s'élève à 99.41%.



(a) Malwares et trafic légitime[WZZ+17] (b) Trafic vidéo [SS21]

FIGURE 5.2 – Transformation de flux en images 2D

Dans [SS21], Shapira & al. constituent une image 2D à partir des caractéristiques des flux. Les deux grandeurs considérées sont la taille des paquets et les inter-arrivées, ce qui donne deux dimensions. La représentation d'un flux eu égard de ces deux grandeurs est visible sur la Figure 5.2b. Les auteurs considèrent ici des flux unidirectionnels, c'est-à-dire que les caractéristiques ne concernent qu'une seule direction. Trois techniques de chiffrement sont considérées ; non-VPN, VPN et TOR. La première technique désigne les flux originels, généralement chiffrés via le protocole TLS. Fait remarquable ; pour de la catégorisation de trafic non-VPN, un arbre de décision (DT) a donné de meilleures performances que le CNN, alors que sa conception est beaucoup plus simple. Nous retenons ce résultat car la classification de trafic chiffré non-VPN est très proche de la tâche que l'on souhaite accomplir. Le DT avait été entraîné sur des statistiques portant sur la taille des paquets et les temps d'inter-arrivées. S'y rajoutent deux autres caractéristiques ; le débit et le taux de paquets par seconde. Nous atteignons au total 12 caractéristiques, extraites toutes les 15 secondes.

En l'état actuel de nos connaissances, aucune étude ne s'est intéressée à la classification du trafic CG. Nous avons cependant parcouru de nombreux articles traitant de classification réseau. Beaucoup de travaux récents ont recours à des méthodes d'apprentissage profond (DL), bien que des méthodes plus simples parviennent aussi à de bons résultats [BFC+19]. L'inconvénient du DL est que le processus de décision est insaisissable. En effet, la complexité des architectures et le nombre démesuré de paramètres compliquent l'explicabilité des modèles, ce qui a ouvert un pan de recherche à part entière. Dans [ASV+21], Akbari & al. soulèvent d'ailleurs ce problème. Ils expliquent qu'un réseau neuronal peut baser sa décision sur des caractéristiques triviales ("*canary features*"). Dans [RKL20], le modèle considéré accorde une importance prépondérante à l'extension `SNI` (*Server Name Indication*) du protocole TLS. Dès lors que cette information est masquée, les auteurs constatent une baisse fulgurante des performances. Le modèle est ainsi tributaire des évolutions protocolaires, ce qui est à proscrire. De surcroît, la simplicité de la classification remet en question l'usage du DL. Akbari & al. décident de masquer ce type de caractéristiques lors de la phase d'entraînement. En dépit des efforts, un biais peut toujours s'immiscer dans les données. De nombreuses études cherchent à résoudre ce problème d' *explicabilité*. On parle d' "*explainable AI*" (XAI). Dans [AKPC21], Ahn & al. ont recours à un algorithme *génétique*. Cela leur permet de sélectionner les caractéristiques primordiales au modèle. Une sorte de compromis est trouvé

entre précision et nombre de caractéristiques retenues. Les algorithmes traditionnels de ML sont beaucoup moins concernés par cette problématique. Pour un DT ou un RF, il suffit de visualiser le modèle pour comprendre son fonctionnement.

5.3 Création et exploitation des jeux de données

Le *testbed* que nous utilisons, que ce soit en termes de plateformes de CG ou d'installation, reprend celui décrit dans la Partie II ; *Caractérisation du trafic CG*. Il suffit de se rapporter au Chapitre 3, Section 3.2 pour en avoir la description. Dans la présente section, nous décrivons plutôt comment nous avons constitué nos jeux de données ("*dataset*"). Nous expliciterons ensuite comment nous en extrayons des caractéristiques.

5.3.1 Données brutes

Désireux d'avoir un jeu de données représentatif, nous avons procédé à de multiples captures réseau (fichiers `pcap`). Chacune d'elles correspond à une certaine configuration. Nous avons ainsi fait varier les plateformes de jeu parmi les 4 dont nous disposons (GFN, PSN, STD, XC) et diversifié le type d'entrée des commandes (*gamepad ou clavier+souris*). Lorsque c'était possible, nous avons de plus spécifié la résolution et le *framerate*. Nous retrouvons ainsi des captures correspondant à une résolution de 720p, 1080p et 4K. Il en est de même pour le nombre d'images par seconde, avec du 30 et du 60 fps. Notons qu'au moment de l'étude, STD était la seule plateforme à supporter le *streaming* en 4K. Pour finir, nous avons aussi fait varier les jeux. Plusieurs perspectives créant des dynamiques d'image différentes ont ainsi été considérées :

— vue à la première personne (FPS) : point de vue du personnage incarné ;
— vue à la troisième personne (TPS) : point de vue derrière l'avatar ;
— vue globale : angle fixe qui montre l'avatar et la zone autour de celui-ci.

Nous avons de surcroît exploré différents types de jeux. On retrouve notamment des jeux de tir, des jeux de plates-formes et des jeux de course.

Nous avons également généré et collecté du trafic de type non-Cloud Gaming (NCG) mais pouvant y ressembler de par certaines caractéristiques communes avec le trafic CG. Notre objectif est d'avoir du trafic "*similaire*" au CG, de sorte à ce que la différenciation ne se fasse pas trivialement. Nous nous limitons ainsi aux applications UDP à fort débit descendant. Au total, nous avons identifié 4 catégories de services, avec pour chacune d'elles, plusieurs applications typiques :

— la visio-conférence (VC) : *Discord*, *Google Meet*, *Jitsi Meet* ;
— le streaming vidéo (VS) : *YouTube*, *Facebook Watch* ;
— le streaming vidéo live (LV) : *YouTube Live*, *Facebook Live*, *Twitch* ;
— la navigation facebook (FN) : *consultation de profils, du fil d'actualités...*.

Au même titre que pour le CG, nous avons cherché à avoir un jeu de données représentatif. De la sorte, nous avons varié la qualité vidéo dès qu'il nous était possible de le faire. On peut de plus noter que le trafic VC a recours au protocole RTP, bien souvent utilisé pour le CG. Nous avons choisi d'inclure de la navigation facebook parce que le protocole QUIC [IT21] est utilisé sur UDP. Ce protocole est en passe de devenir un standard, et devrait à terme remplacer HTTP/2. Notons que la totalité du trafic NCG considéré n'est pas aussi sensible à la latence que le CG. Cela justifie la différence de traitement souhaitée dans le cadre d'une ingénierie de trafic par classe.

Nous avons vu dans la partie II que l'ensemble des plateformes de CG régulent leur débit. Le trafic évolue donc en fonction des conditions réseau. Les conditions de réalisation de l'expérience impactent par conséquent les caractéristiques des flux. Nous pouvons dès lors nous questionner sur la capacité de généralisation d'un modèle de détection entraîné dans des conditions optimales. Est-il possible pour un tel modèle de reconnaître du trafic CG en conditions réseau dégradées ? Pour répondre à cette question, nous procédons à des captures CG dans de telles conditions (CGP). Ces données peuvent aussi servir à diversifier le jeu d'entraînement.

La dégradation des conditions réseau se fait par l'application de règles `tc-netem`. Un appareil (*Network bridge*) situé entre le client CG et la *gateway* va tour à tour impacter :

— La latence : +20ms ;
— La gigue : +5ms ;
— Le taux de pertes : +5% ;
— La bande passante : 10 Mbps ;

Les 4 contraintes retenues sont suffisantes pour déclencher une réaction des plateformes comme nous l'avons vu dans le chapitre 3. Pour rappel, l'architecture du *testbed* est donnée par la Figure 3.1.

Au total, nous avons réalisé 57 captures distinctes représentant 17 Go de traces, 19 millions de paquets et d'une durée de 240 minutes. Dans une démarche de reproductibilité, toutes nos données sont mises à disposition de la communauté : `https://cloud-gaming-traces.lhs.loria.fr`.

En plus des 4 plateformes commerciales de CG, il existe des solutions permettant d'exécuter ses jeux à distance. Nous avons retenu *Steam Remote Play*[23] et *Moonlight*[24]. Dans chaque cas, nous avons instancié un serveur de jeu sur le réseau local. Les deux solutions y sont tour à tour déployées pour pouvoir procéder au *streaming* vers le client. Tous les jeux sont lancés depuis l'application *Steam* du serveur. Nous aboutissons dès lors à un autre *dataset* de test qui servira à évaluer la capacité du modèle à identifier le trafic CG de plateformes pour lesquels il n'a pas été entraîné, contrairement aux 4 principales.

5.3.2 Extraction des caractéristiques

À présent que nous disposons de données "*brutes*", il convient d'en extraire les caractéristiques. Le trafic CG est asymétrique ; le *downlink* diffère fortement du *uplink*. Nous prenons par conséquent le parti de différencier les deux directions. Nous aurons de la sorte un fort trafic vidéo descendant contre quelques commandes de jeu dans le sens montant. Pour chacune des directions, nous extrayons en ensemble de caractéristiques.

Tout d'abord, nous calculons l'inter-temps d'arrivée (IAT) **moyen** et sa **variance**. Du fait que le CG soit "*temps réel*", nous nous attendons à ce que le trafic vidéo génère des paquets de manière régulière à un multiple de la fréquence de rafraîchissement. Il s'ensuivrait des IATs très courts et réguliers dans le sens descendant, donc avec une faible variance. La variance doit permettre de différencier le trafic CG d'autres trafics de streaming vidéo qui envoient les données par salves ("*burst*"). Ceci se répercute donc sur les IATs, qui ne seront par conséquent pas réguliers. Le phénomène est illustré sur la Figure 5.1a.

En plus de l'aspect "temporel" susvisé, nous prenons aussi en compte les tailles des paquets. Au même titre que pour les IATs, nous en dérivons deux statistiques : la **moyenne** et la **variance**. Si nous considérons des services de type visio-conférence, le trafic montant peut contenir de l'audio et/ou de la vidéo du participant. Ce type de données est plus conséquent que les seules

23. `https://store.steampowered.com/remoteplay/`
24. `https://moonlight-stream.org/`

commandes utilisateur du trafic CG. En plus de cela, les trafics CG et VC n'ont pas le même débit descendant. Le débit du CG est bien plus conséquent que celui de la VC. Cette différence se répercute naturellement sur les IATs et/ou les tailles des paquets.

L'ensemble de nos caractéristiques est calculé sur une fenêtre temporelle. La valeur de cette fenêtre est encore à définir ; nous nous y intéresserons en Section 5.5. Nous avons pour le moment présenté 4 caractéristiques, portant sur les IATs et les tailles des paquets. Nous en prenons deux autres en compte, bien qu'elles soient liées au 4 premières. Pour chaque fenêtre temporelle w_t, nous extrayons de surcroît ;

— Le **nombre** de **paquets** la composant (*lié à l'IAT moyen*) ;

— La **somme** des **tailles** des paquets (*lié au nombre de paquets et à la taille moyenne*) ;

Nous arrivons à un total de 12 caractéristiques par flux, soit six pour chaque direction du trafic.

Comme nous l'avions mentionné en Partie I, les plateformes ont recours à différentes techniques de multiplexage. PSN se base par exemple sur le premier octet du payload pour différencier les flux. Les plateformes STD et XC ont quant à elles recours au champ SSRC de RTP. GFN dispatche ses flux sur différents ports. Un couple de ports sera ainsi dédié à la transmission vidéo tandis qu'un autre servira à transmettre les flux audio. Devant toutes ces disparités, nous avons décidé d'élargir la notion de flux en considérant l'ensemble des données échangées entre un couple d'adresses IP à un instant donné. Nous bénéficions de la sorte d'un plus haut niveau d'abstraction en phase avec la réalité des échanges des plateformes. Quel que soit leur fonctionnement particulier, l'ensemble des données échangées est ainsi pris en compte.

5.4 Modèles

Dans cette section, nous allons présenter nos différents classificateurs. Comme de coutume en apprentissage machine (ML), il convient de comparer nos modèles à une base de référence naïve afin de montrer leur plus-value. Nous commencerons donc par présenter notre classificateur basés sur des seuils“ *Thresholds*”. S'ensuivra une rapide présentation générale du fonctionnement des classificateurs DT et RF. Nous livrons ensuite les principaux hyper-paramètres des deux classificateurs en Table 5.1.

5.4.1 Base de référence

Comme son nom l'indique, le classificateur “ *Thresholds*” base sa décision de classification sur un ensemble de seuils. Nous avons présenté 12 caractéristiques au cours de la Section 5.3.2. Elles sont extraites périodiquement, pour chaque couple d'adresses IP. À chaque caractéristique i seront associés deux seuils ; $Q_{1,i}$ et $Q_{3,i}$. Ces seuils sont appris sur le jeu d'entraînement. Soit $f_{t,i}$ la valeur de i sur la fenêtre temporelle w_t. Nous définissons V_i, comme l'ensemble des valeurs prises par i. Ainsi ;

$$\forall i \in [1;12], V_i = \{f_{t,i},\, t \in [1;N]\}$$

La variable N correspond au nombre de fenêtres temporelles dans le jeu d'entraînement. Une fois ces ensembles conçus, nous sommes en mesure de calculer $Q_{1,i}$ et $Q_{3,i}$. Ils correspondent au premier (*resp. troisième*) quartile de V_i. L'ensemble du trafic considéré est du type CG.

Le processus d'inférence est assez simple. Étant donné une fenêtre w_x à classifier, on considère le nombre de caractéristiques λ se trouvant entre leurs seuils repectifs.

$$\lambda = \#\{f_{x,i} \text{ tel que } Q_{1,x} \leq f_{x,i} \leq Q_{3,x}\}$$

La classification consiste à comparer λ à un seuil Thr. Dès lors que λ dépasse Thr, la fenêtre est labélisée "*trafic CG*". Thr est une valeur fixée au préalable ; on parle d' *hyper-paramètre*. Nous aborderons en Section 5.5 le choix de cette valeur.

5.4.2 Modèles d'apprentissage automatique supervisé

Nous présentons ici l'utilisation de modèles basés sur les forêts d'arbres décisionnels RF et les arbres de décision DT pour la classification. Nous commençons par présenter le DT car RF n'en n'est qu'une version complexifiée. Nous pourrons ensuite nous concentrer sur les spécificités du RF.

Soit un ensemble Q_m composé de plusieurs classes $k \in [0; K-1]$. Le but du DT est de scinder Q_m en deux ensembles distincts ; Q_m^{left} et Q_m^{right}. Chaque échantillon de Q_m sera attribué à l'un des deux ensembles. La répartition se fait par le biais d'un test sur une des caractéristiques. Lors de la phase d'entraînement, le DT définit une caractéristique j et lui associe un seuil t_m. Le couple $\theta = (j, t_m)$ est choisi de sorte à départager au mieux les classes. Plusieurs métriques permettent d'évaluer cette grandeur. La métrique la plus fréquente est le *coefficient de Gini*. Elle s'écrit de la sorte ;

$$H(Q_m) = \sum_k p_{m,k}(1 - p_{m,k})$$

Dans l'équation, la valeur $p_{m,k}$ désigne la fréquence de la classe k dans l'ensemble Q_m. Au final, le couple θ doit minimiser la fonction H pour les sous-ensembles Q_m^{left} et Q_m^{right}. Cette grandeur est donnée par la fonction $G(Q_m, \theta)$, définie dans l'équation 5.1. On y calcule la somme pondérée de la fonction H sur les sous-ensembles.

$$G(Q_m, \theta) = \frac{n_m^{left}}{n_m} H(Q_m^{left}(\theta)) + \frac{n_m^{right}}{n_m} H(Q_m^{right}(\theta)) \tag{5.1}$$

Le procédé susvisé est récursif. Une fois les ensembles Q_m^{left} et Q_m^{right} obtenus, ils sont à leur tour scindés en deux. On aboutit ainsi à une structure arborescente, d'où le nom de *Decision Tree*. On peut en apercevoir une illustration sur la Figure 5.3. Un ensemble de conditions permet de

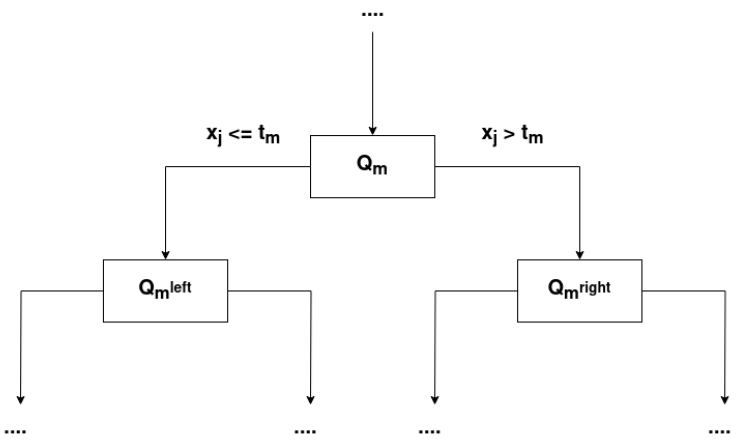


FIGURE 5.3 – DT : arborescence du modèle - *splitting* de Q_m

spécifier s'il convient ou non de scinder un ensemble. Ce sont ces conditions qui permettent à

l'algorithme de s'arrêter. Les ensembles non scindés sont les *feuilles* de l'arbre ; c'est à ce niveau que la classification a lieu.

La phase d'entraînement aboutit en un arbre de décision. Chacun de ses noeuds consiste à comparer une caractéristique à un seuil. Les attributs d'une instance à classifier déterminent le chemin emprunté dans l'arbre. On s'arrête dès lors qu'une *feuille* est atteinte. Soit Q_m l'ensemble des échantillons associés à la feuille en question. La classification peut se faire de deux façons :

— Attribuer à x le label k^* le plus fréquent ; $k^*,\ \forall i\ p_{m,k*} \geq p_{m,i}$

— Labéliser aléatoirement, selon la fréquence des classes représentées (*probabiliste*) ;

Plusieurs algorithmes permettent de créer un DT [SL91]. Dans notre cas, nous retiendrons l'algorithme `CART`, implanté dans la bibliothèque *scikit learn* [PVG+11].

Le classifieur RF entraîne plusieurs DT en parallèle. Les arbres de décision le composant peuvent être entraînés sur des sous-ensembles distincts (*bootstrap*). De plus, le calcul de chaque couple θ se base sur un sous-ensemble de caractéristiques tirées aléatoirement. Ce procédé propre aux RFs permet d'assurer une faible corrélation entre ses arbres. On aboutit ainsi à des connaissances plus diversifiées, avec des règles distinctes. Lors de la phase d'inférence, deux DTs peuvent aboutir à des labels différents. Il s'agit alors d'agréger les différents résultats pour aboutir à une classification finale. Un vote est réalisé à cet effet. Les labels des différents DTs peuvent être pondérés avec leur estimation de probabilité.

Les principaux hyper-paramètres des classificateurs DT et RF sont donnés dans le tableau 5.1. Les deux dernières colonnes du tableau nous renseignent sur leurs valeurs par défaut dans *scikit learn*. Mis à part *max_features*, les hyper-paramètres des deux classificateurs ont généralement même valeur. Notons que les deux dernières lignes du tableau portent sur des paramètres spécifiques au RF. Dans l'ensemble, les valeurs renseignées sont intelligibles. Il convient toutefois d'en clarifier quelques-unes. Dans la première ligne, *gini* désigne le *coefficient de Gini*. Nous l'avons présenté quelques paragraphes plus tôt. Lorsque des hyper-paramètres sont fixés à *None*, cela signifie qu'ils n'exercent aucune contrainte. Ils sont donc ignorés ; c'est le cas pour *max_depth* et *max_leaf_nodes*. L'hyper-paramètre *max_features* est ignoré pour le DT, mais pas pour le RF. Soit $n\_features$ le nombre de caractéristiques prises en compte. Le champ *sqrt* signifie que le RF prendra en compte $max\_features = \sqrt{n\_features}$ pour chaque *split*.

Hyper-paramètre	Définition	Défaut : DT	Défaut : RF
criterion	Métrique sur la qualité d'un *split*	*gini*	*gini*
max_depth	Profondeur maximale de l'arbre	*None*	*None*
max_leaf_nodes	Nombre maximal de feuilles	*None*	*None*
max_features	Nombre de caractéristiques à considérer pour trouver θ	*None*	*sqrt*
min_impurity_decrease	Diminution minimale de l'impurité pour scinder un ensemble	*0.0*	*0.0*
min_samples_leaf	Nombre minimal d'échantillons dans une feuille	*1*	*1*
min_samples_split	Nombre minimal d'échantillons pour scinder un *noeud*	*2*	*2*
n_estimators	Nombre de DTs composant un RF	-	*100*
bootstrap	Utiliser des sous-ensembles aléatoires pour entraîner les DTs	-	*True*

TABLE 5.1 – Aperçu des hyper-paramètres

5.5 Étude des hyper-paramètres

Nous avons abordé quelques hyper-paramètres au cours des deux dernières sections. C'est un ensemble de valeurs qu'il convient de fixer au lancement d'un programme. Dans la présente

section, nous tâchons de leur spécifier une valeur. Le choix se base sur un compromis entre performance de la classification et la complexité du modèle qui influe sur le temps d'inférence. Parmi la multitude de métriques d'évaluation, nous considérons la précision (ou "*accuracy*"). Elle représente la proportion de bonnes prédictions par rapport à l'ensemble des classifications. Nous en rappelons la formule en équation 5.2.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.2}$$

Nous commencerons par nous intéresser à la taille des fenêtre temporelles. Ceci fait, nous étudierons les hyper-paramètres de nos différents modèles.

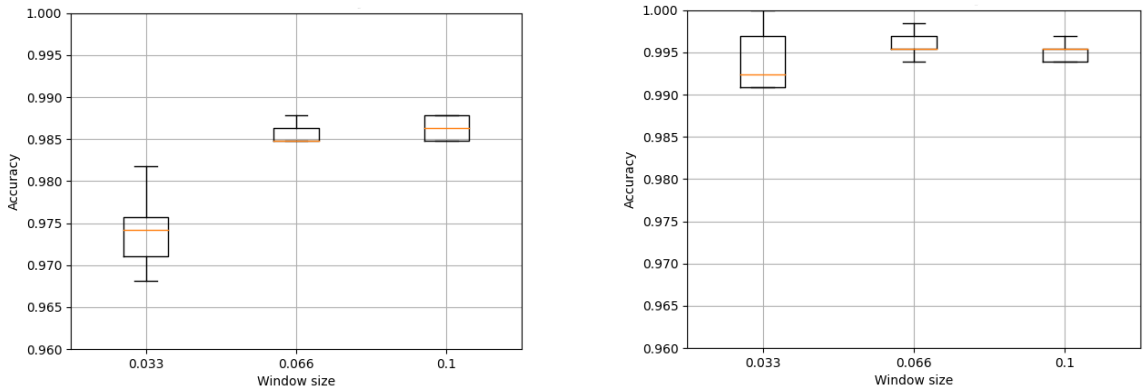
5.5.1 Taille de la fenêtre temporelle

Nous extrayons périodiquement les caractéristiques de chaque flux. Nous cherchons ici à évaluer quelle fréquence d'extraction serait la plus adaptée. Une grande taille de fenêtre ferait croître les délais de classification. De surcroît, nos données brutes (*fichiers* `pcap`) engendreraient moins d'échantillons. À l'inverse, une taille de fenêtre dérisoire pourrait ne pas suffire à capturer les motifs d'un flux permettant de le caractériser. Un compromis est donc à trouver entre l' *Accuracy* et la taille de la fenêtre.

Nous considérons au total trois tailles de fenêtre ; 33ms, 66ms et 100ms. Chacune de ces tailles donne lieu à un jeu d'entraînement. Nous avons fixé ces tailles de sorte à ce qu'elles puissent contenir une image complète. Pour une diffusion vidéo en ;

— 60 fps : une fenêtre de 33ms est suffisante pour contenir une image : $\frac{1}{60} \times 2 \approx 0.033$;
— 30 fps : une fenêtre de 66ms est suffisante pour contenir une image : $\frac{1}{30} \times 2 \approx 0.066$;

Une fois les jeux de données créés, nous étudions les résultats de classification des modèles DT et RF. Les hyper-paramètres de ces deux classifieurs sont alors laissés par défaut. Nous pouvons nous rapporter à la Table 5.1 pour les examiner. Par la suite, nous procédons à du K-FV (K-Fold Validation) sur les trois jeux de données. Le nombre d'itérations k est fixé à 5. La Figure 5.4 trace les résultats des deux modèles. Les résultats sont très bons ; on dépasse systématiquement les



(a) Classifieur DT

(b) Classifieur RF

FIGURE 5.4 – Fenêtre temporelle : impact de sa taille

96% d'*Accuracy*. On ne constate pas de grande différence entre une période d'extraction de 66ms et 100ms. L'écart avec une fenêtre de taille 33ms est un peu plus prononcé, mais reste négligeable. Il représente environ 1% de l'*Accuracy*. Nous décidons dès lors de retenir des fenêtres de taille 33ms.

5.5.2 Base de référence

Nous avons présenté notre base de référence dans la Section 5.4.1. Pendant la phase d'entraînement, le modèle est seulement exposé à du trafic CG. Il calcule les premier et troisième quartiles de chacune de ses caractéristiques. Ces seuils sont ensuite utilisés lors de la phase d'inférence. La classification dépend de λ, le nombre de caractéristiques se situant entre ses seuils respectifs. Nous étudions dès lors l'impact du seuil Thr ; la valeur minimale de λ pour attribuer le label CG.

Nous commencons par créer un jeu de données D_{train} exclusivement composé de CG. Nous en créons ensuite un deuxième, où trafic CG et NCG représentent 50% des données. Les catégories de service constituant les deux classes sont représentées équitablement. L'ensemble D_{test} résultant nous servira à évaluer le modèle. Chaque valeur de $Thr \in [1; 12]$ donne lieu à une évaluation. La

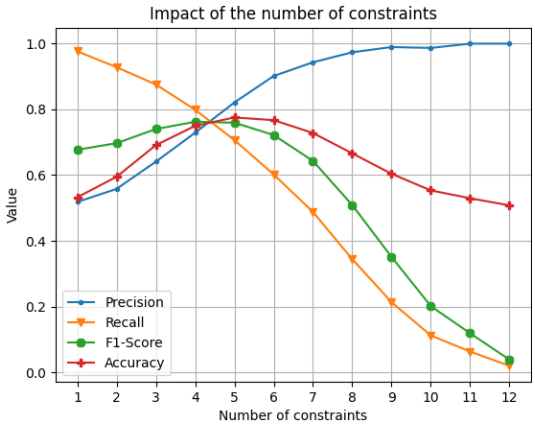


FIGURE 5.5 – Impact de Thr sur la classification

Figure 5.5 représente les résultats de classification pour chacune de ces valeurs. Quatre courbes sont représentées ; chacune correspond à une métrique d'évaluation. Un seuil trop élevé conduit à une bonne *Precision*, au détriment du *Recall*. Ces deux métriques sont rappelées dans les équations 5.3 et 5.4. Quand la contrainte est forte, le modèle attribue très rarement le label CG. En conséquence, on dénombre peu de *faux positifs*, ce qui profite à la *Precision*. À l'inverse, un grand nombre d'instances CG ne sont pas reconnues. Le taux consécutif de *faux négatifs* porte alors atteinte au *Recall*. On constate le phénomène inverse sur de faibles valeurs de Thr. Ici, le label CG est trop souvent attribué. La *Precision* tend alors vers la proportion d'instances CG dans le jeu de test. Sa valeur se situe ainsi à $\approx 50\%$.

$$\text{Precision} = \frac{TP}{TP + FP} \qquad (5.3)$$

$$\text{Recall} = \frac{TP}{TP + FN} \qquad (5.4)$$

Il convient dès lors de trouver un compromis entre *Precision* et *Recall*. Maximiser l'*Accuracy* revient à limiter le nombre de *faux positifs* et de *faux négatifs*. Cette métrique constitue ainsi un bon compromis, comme on peut le voir sur la Figure 5.5. Quand Thr est égal à `5`, l'*Accuracy* atteint sa valeur maximale. La *Precision* vaut alors $\approx 82\%$ et le *Recall* ≈ 70 %.

5.5.3 Decision Tree et Random Forest

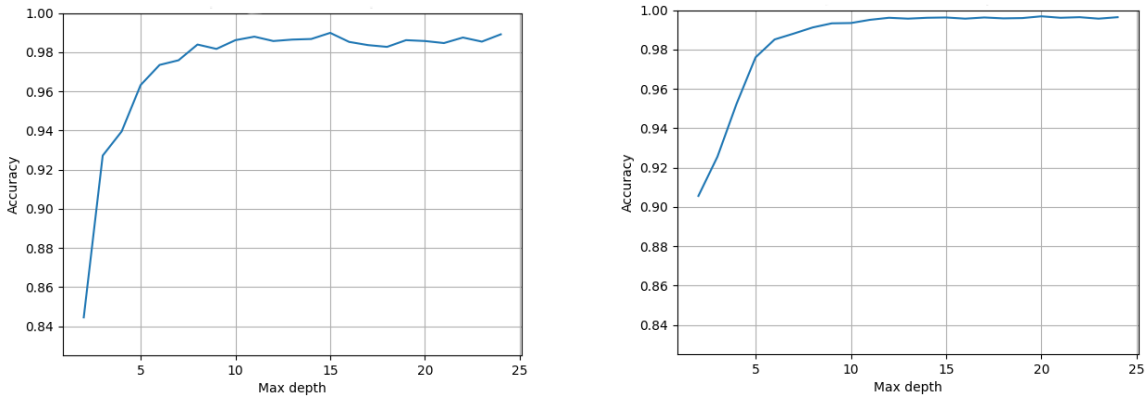
Nous avons présenté en Section 5.4.2 les hyper-paramètres des classificateurs DT et RF. Deux d'entre eux sont directement liés à la complexité des modèles ; *max_depth* et *n_estimators*. Comme son nom l'indique, *max_depth* permet de limiter la profondeur des arbres. Une arbre de plus petite taille réduit forcément le temps d'inférence. Du surcroît, sa simplicité limite les risques d'*over-fitting*. Une remarque connexe peut être établie au sujet de *n_estimators*. Le temps d'entraînement d'un classificateur RF croît avec le nombre de DTs le composant. Il en est de même pour le temps d'inférence. Il faut cependant souligner que *n_estimators* profite à la robustesse du modèle. Nous étudions deux autres hyper-paramètres ; *max_features* et *bootstrap*. Le premier, *max_features*, limite le nombre de caractéristiques prises en compte au moment d'un *split*. Le procédé permet de limiter les risques d'*over-fitting*. C'est la raison pour laquelle nous choisissons de l'étudier. Le dernier hyper-paramètre est un booléen ; il convient juste de choisir entre *True* et *False*. Son étude ne devrait donc pas alourdir notre analyse.

Suite à l'étude menée en Section 5.5.1, la taille des fenêtres temporelles est fixée à 33 ms. Nous employons la méthode *Grid Search* pour rechercher les hyper-paramètres optimaux. Cela revient à tester toutes les combinaisons possibles. Chaque combinaison donne lieu à une validation croisée (*K-Fold Validation*). Le principe consiste à entraîner le modèle sur $\frac{k-1}{k}\%$ des données et à l'évaluer sur les $\frac{1}{k}\%$ restants. Le procédé est répété k fois, de sorte à ce que chaque donnée ait appartenu au jeu de test.

Hyper-Parameter	DT	RF	Testing Range
max_depth	15	20	$[2; 24]$
$max\_features$	8	3	$[1; 12]$
$n\_estimators$	-	119	$[80; 119]$
$bootstrap$	-	$False$	$\{True; False\}$

TABLE 5.2 – DT & RF : Hyper-Parameters w=33ms

Le Tableau 5.2 retrace les résultats de l'étude. La dernière colonne précise l'étendue de la recherche. Eu égard des 4 hyper-paramètres étudiés, on aboutit à 22.080 combinaisons possibles. Le procédé est fastidieux ; nous n'avons donc pas considéré l'entièreté des caractéristiques. On constate tout d'abord que $max\_features$ est bien en deçà des 12 caractéristiques considérées. L'écart est d'autant plus prononcé pour le classificateur RF. On aboutit ainsi à de meilleurs résultats de classification en limitant $max\_features$. Cela est le signe d'un sur-apprentissage (*over-fitting*) des données d'entraînement. Selon le Tableau 5.2, la profondeur maximale d'un DT devrait être fixée à 15. Pour un RF, il faudrait la fixer à 20. Soucieux de classifier le trafic en temps réel, nous cherchons à davantage limiter cette valeur. Pour ce faire, nous étudions l'évolution de l'*Accuracy* en fonction de *max_depth*. Les hyper-paramètres étudiés sont fixés selon les résultats de la Table 5.2. Les autres sont laissés tels quels. La Figure 5.6 trace nos résultats pour les classificateurs DT (*à gauche*) et RF (*à droite*). Comme l'on s'en doutait, les deux courbes sont asymptotiques. Cela signifie que les gains sur l'*Accuracy* deviennent négligeables à partir d'une certaine profondeur. Les deux courbes semblent d'ailleurs se stabiliser à partir de $max\_depth = 10$. Nous retenons donc cette valeur pour les deux classificateurs. Dans le cas du RF, cela peut diviser par deux le nombre de tests.



(a) Classifieur DT (b) Classifieur RF

FIGURE 5.6 – Impact de *max_depth* sur l'*Accuracy*

5.6 Évaluation des modèles

Enfin, nous évaluons ici la capacité des modèles à s'acquitter de leur tâche. L'ensemble de leurs hyper-paramètres est fixé selon les résultats du tableau 5.2. Dans un second temps, nous évaluons leur capacité de généralisation. Pour ce faire, nous les entraînons sur des traces CG "*normales*" (CG). Nous vérifions ensuite leur capacité à reconnaître du trafic CG *perturbé* (CGP). Le lecteur peut se reporter à la Section 5.3.1 pour plus de détails. Nous ne ré-utilisons pas les données utilisées au cours de la Section précédente.

5.6.1 Traces CG *normales*

L'ensemble des données utilisées ici ont été collectées dans des conditions réseau *favorables*. Nous prenons garde de représenter équitablement les instances CG et NCG. Suite à cette manipulation, chacune des deux classes contient 79.284 échantillons. Nous procédons dès lors à une validation croisée *K-Fold*, avec $k = 5$.

Model	Accuracy	Precision	Recall	F1-Score
Thresholds	0.863	0.847	0.886	0.866
DT	0.968	0.997	0.939	0.967
RF	0.969	0.998	0.940	0.968

TABLE 5.3 – Classification du trafic CG, Conditions *normales*

Chacune des 5 itérations donne lieu à une évaluation portant sur $\frac{1}{5}$ des données. Nous moyennons le résultat de chaque métrique pour aboutir au Tableau 5.3. On constate un écart flagrant entre notre base de référence et les deux autres classificateurs. En ce qui concerne l'*Accuracy*, la différence s'élève à ≈10%. Le DT et le RF ont de très bons résultats, quasiment identiques, quelle que soit la métrique considérée. Tout ceci reflète la faible proportion de *faux positifs* et de *faux négatifs*.

5.6.2 Traces CG *perturbées*

Pour tester la capacité de généralisation de nos modèles, nous utilisons ici deux jeux de données. Le premier, le *jeu d'entraînement*, est constitué de traces CG prises en conditions réseau *normales*. Le *jeu de test* est constitué d'échantillons CG *perturbés* (CGP) tels que décrits dans la Section 5.3.1. Nous répartissons l'ensemble des échantillons NCG sur les deux jeux de données. Comme de coutume, nous représentons équitablement les deux classes. Au final, notre *jeu de test* comporte 39642×2 instances.

L'évaluation est différente que dans la Section 5.6.1. Il nous est en effet impossible de faire de la validation croisée. Une seule itération est ainsi accomplie. Les résultats sont consignés dans le Tableau 5.4.

Model	Accuracy	Precision	Recall	F1-Score
Thresholds	0.85	0.84	0.860	0.851
DT	0.925	0.996	0.853	0.919
RF	0.943	0.998	0.889	0.940

TABLE 5.4 – Classification du trafic CG *perturbé*

On constate une certaine détérioration entre la Table 5.3 et la Table 5.4. Le *Recall* est la métrique la plus impactée. Cela fait sens, car nous cherchons à reconnaître du trafic CG *perturbé*. L'adaptation des terminaux impacte en effet les caractéristiques de leur trafic. Les seuils calculés à partir de trafic *normal* (CG) peuvent donc être inadaptés. En corollaire, les classificateurs tendent à avoir un plus grand taux de *faux négatifs*. La reconnaissance du trafic CG n'est donc plus aussi bonne qu'en Section 5.6.1. Elle diminue d'environ 8% pour le DT. Le classificateur RF par construction est plus robuste ; ce qui se retrouve dans ses résultats qui surpassent tous les autres. Le DT reste tout de même assez performant dans ces conditions avec un F1-score seulement 2% inférieur au DT.

		DT		RF	
True Label	NCG	39.564 (**99.80%**)	78 (**0.20%**)	39.505 (**99.65%**)	137 (**0.35%**)
	CG	4410 (**11.12%**)	35.232 (**88.88%**)	5.835 (**14.72%**)	33.807 (**85.28%**)
		NCG	CG	NCG	CG
		Predicted label			

TABLE 5.5 – Classification du trafic CG *perturbé* ; matrice de confusion

Plus de détails sur ces résultats sont donnés en Table 5.5 à travers les matrices de confusion des classificateurs DT et RF. Le principe consiste à confronter les résultats de classification aux labels effectifs. Les lignes correspondent aux classes à prédire. Les labels attribués sont quant à eux représentés en colonnes. Par exemple, 33.807 échantillons CG ont été reconnus par le classificateur RF. Pour cette classe, le ratio de bonnes prédictions s'élève ainsi à 85.28%. Les métriques présentées en Table 5.4 découlent de ces valeurs.

Afin d'améliorer la robustesse de nos modèles par rapport au trafic CG perturbé, nous avons testé l'ajout de 10% d'instances de trafic CG perturbé à l'ensemble de données d'entraînement. L'évaluation selon un procédé similaire donne désormais des scores F1 de 97,5% pour le classificateur DT et de 98,8% pour le classificateur RF. Ainsi, aider le modèle à apprendre la plage d'adaptation du trafic CG réduit la diminution des performances de 6% à 1% lors de la classification d'un tel trafic CG perturbé.

Nos modèles parviennent à reconnaître du trafic CG capturé en conditions réseau normales ou perturbées. Dans l'ensemble, les classificateurs parviennent toujours à de très bons résultats. De surcroît, DT et RF aboutissent à des résultats très proches. Si l'on considère la complexité des modèles, le DT semble être le plus adapté à notre problème.

Des questions subsistent néanmoins plus largement quant à la capacité de généralisation de nos modèles. Dans [YCHL12], Lee & al. quantifient la sensibilité d'un jeu à la latence. Pour ce faire, ils observent les caractéristiques du trafic dans les deux directions. Nos modèles sont-ils donc en mesure de reconnaître un jeu jusqu'alors inconnu ? Cette question peut être généralisée aux autres plateformes de CG. Une fois la plateforme Luna [25] disponible en France, serait-on par exemple en mesure de la classifier correctement ? L'inconvénient du ML est que le *jeu d'entraînement* doit être exhaustif pour être représentatif. Pour bien reconnaître une classe, la pleine diversité du trafic associé doit ainsi être représentée. La section suivante s'intéresse à ces questions.

5.7 Généralisation à d'autres jeux et plateformes

En section 5.6, nous nous sommes questionnés sur la capacité de *généralisation* de nos modèles supervisés. Dans [YFJR21], Lixuan & al. évoquent les limites de l' *apprentissage supervisé* pour classifier du trafic inconnu. Dès lors, nous décidons d'introduire un modèle de DL ne se basant pas sur ce type d'apprentissage. Nous comparons ses résultats de classification à ceux du DT. Le *trafic inconnu* concerne :

— des jeux sur lesquels les modèles n'ont pas été entraînés ;
— des plateformes de CG sur lesquelles les modèles n'ont pas été entraînés.

Au même titre que dans la Section 5.6.1, nous évaluons aussi les classifieurs sur du *trafic connu*.

5.7.1 Un modèle non supervisé : USAD

USAD signifie *UnSupervised Anomaly Detection* [AMG+20]. Comme son nom l'indique, le modèle se base sur de l'apprentissage *non supervisé*. Lors de la phase d'entraînement, il est exposé à du trafic CG. Soit $X = \{x_1, ..., x_T\}$ ses données d'entrée. Chaque vecteur x_i comporte les 12 caractéristiques présentées en 5.3.2. Le but du modèle est de déclarer si un vecteur $\widehat{x_t}, t > T$ est *anormal* ou pas. Un vecteur est considérée comme *anormal* si il se différencie trop du jeu d'entraînement. Dans notre cas, une anomalie se rapporte à du trafic *NCG*.

Le modèle USAD est constitué d'un encodeur et de deux décodeurs. Comme nous pouvons l'apercevoir sur la Figure 5.7, l'entraînement se déroule en deux étapes. La première étape consiste à minimiser l'erreur entre l'encodage et le décodage. L'encodage revient à représenter un vecteur W par un vecteur de plus petite dimension. Le décodage est l'opération inverse. Soit AE la composée des fonctions d'encodage et de décodage ; $AE = D \circ E(W)$. Le problème revient à minimiser l'écart entre W et son image par AE. La fonction objectif peut s'écrire de la sorte ;

$$\mathcal{L}_{AE} = \|W - AE(W)\|_2 \tag{5.5}$$

Comme nous l'avons mentionné précédemment, le modèle USAD est composé de deux décodeurs. Il y a dès lors deux fonctions objectif à optimiser, soit une par décodeur.

L'étape 2 se base sur les *Generative Adversarial Network* (GAN) [GPM+14]. Pour ces modèles, un réseau G est chargé de générer des données réalistes. Un second réseau doit différencier

25. `https://luna.amazon.com`

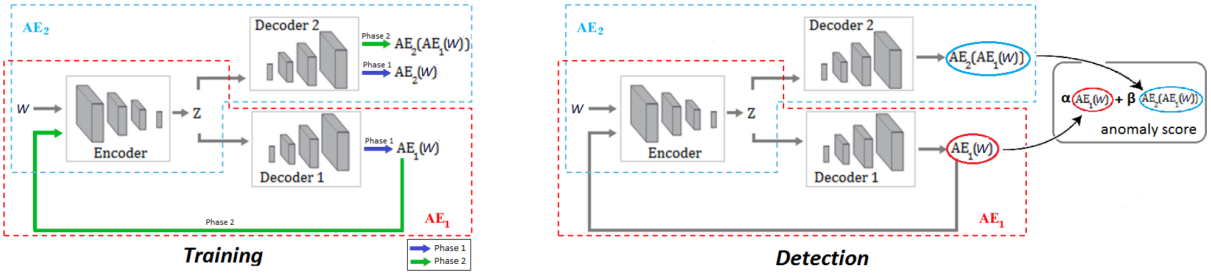


FIGURE 5.7 – Architecture du modèle USAD [AMG+20]

les *vraies* données de celles qui ont été générées par G. Chaque réseau a sa propre fonction objectif.

— G cherche à maximiser la probabilité que le second réseau se trompe ;
— D, le second réseau, cherche à minimiser ser erreurs de classification.

Le modèle USAD se différencie quelque peu des GAN. Dans notre cas, les auto-encodeurs AE_1 et AE_2 cherchent tous deux à minimiser leurs erreurs de reconstruction. L'expression a été donnée dans l'équation 5.7. L'adversité survient lorsque AE_2 prend pour entrée la sortie de AE_1. Dès lors, les deux auto-encodeurs n'ont plus la même fonction objectif.

$$\begin{cases} \mathcal{L}_{AE_1} = \frac{1}{n}\|W - AE_1(W)\|_2 \ + \ (1 - \frac{1}{n})\|W - AE_2(AE_1(W))\|_2 \\ \mathcal{L}_{AE_2} = \frac{1}{n}\|W - AE_2(W)\|_2 \ - \ (1 - \frac{1}{n})\|W - AE_2(AE_1(W))\|_2 \end{cases} \tag{5.6}$$

Nous pouvons nous reporter à l'équation 5.6. Soit $\mathcal{C}$ la composée des auto-encodeurs ; $\mathcal{C}(W) = AE_2 \circ AE_1(W)$. L'auto-encodeur AE_1 cherche à minimiser l'erreur entre W et $\mathcal{C}(W)$. À l'inverse, AE_2 cherche à maximiser cette même valeur. Cette dissemblance se manifeste par le changement de signe.

Intéressons-nous à présent à la phase d'inférence. Comment USAD parvient-il à reconnaître une anomalie ? Étant donné un vecteur $\widehat{W}$, le modèle lui associe un score d'anomalie. Ce score est donné par l'équation 5.7.

$$\mathcal{A}(\widehat{W}) = \alpha\|\widehat{W} - AE_1(\widehat{W})\|_2 \ + \ \beta\|\widehat{W} - AE_2(AE_1(\widehat{W}))\|_2 \tag{5.7}$$

Les paramètres α et β permettent de trouver un compromis entre *faux positifs* et *vrais positifs*. Lorsque $\alpha > \beta$, le modèle aura moins de *faux positifs*. L'inconvénient réside dans la baisse du taux de *vrais positifs*. On constate l'inverse dès lors que $\alpha < \beta$. Soit δ_1 l'écart entre $\widehat{W}$ et son image par AE_1. Nous définissons δ_2 comme la distance entre $\widehat{W}$ et $AE_2 \circ AE_1(\widehat{W})$. Nous avons vu dans l'équation 5.6 que AE_1 a été entraîné pour minimiser δ_1. À l'inverse, AE_2 a été entraîné pour maximiser δ_2. Lors de la phase d'inférence, δ_2 dépasse généralement δ_1. Accorder plus de poids à δ_2 (*via le paramètre* β) va irréfutablement augmenter le score d'anomalie. S'ensuit une hausse des *faux positifs* et des *vrais positifs*.

Nous ne nous sommes par livrés à une étude des hyper-paramètres du modèle. Nous les avons ainsi laissés tels quels. L'apprentissage s'est fait par le biais de l'*optimiseur Adam*. Son taux d'apprentissage est fixé à 10^{-3}. Au total, le modèle effectue 100 *epochs*. Chacune d'entre elles porte sur un lot de 128 données.

5.7.2 Comparaison

Dans le cadre d'une collaboration avec nos collègues d'Orange Labs, nous avons comparé les résultats de classification de leur modèle USAD entraîné sur les jeux de données que nous avons généré à notre DT. Plus précisément, nous les entraînons tous deux sur les mêmes échantillons de trafic CG. Pour son entraînement, le *DT* a de plus bénéficié de traces NCG. Quatre évaluations sont alors menées. Elles ont pour but d'évaluer la reconnaissance :

1. de jeux connus ;
2. de jeux inconnus ;
3. de plateformes inconnues ;
4. de trafic NCG.

Dans les résultats représentés par la Figure 5.8, le DT surpasse le modèle USAD sur les types de données "*connus*". L'écart est un peu plus prononcé pour le trafic NCG. Comme mentionné en Section 5.3.1, nous considérons 4 catégories de services pour le NCG. Le DT a appris à les départager du trafic CG pendant sa phase d'entraînement. Ceci explique les bons résultats du *DT* par rapport à ceux du modèle *USAD*. Il convient cependant de nuancer ces résultats. Considérons du trafic NCG n'appartenant à aucune des 4 catégories susvisées. Dès lors, le *DT* est susceptible de commettre de nombreuses erreurs de classification. À l'inverse, *USAD* devrait surpasser les résultats de ce dernier. Nous avons délibérément sélectionné ce modèle pour ses capacités de généralisation. Nous pouvons d'ailleurs l'observer sur les colonnes 2 et 3 de la Figure 5.8. La reconnaissance d'une session CG sur un jeu *inconnu* est meilleure pour le modèle *USAD*. Son *Accuracy* dépasse celle du *DT* d'environ 20%. L'écart est encore plus fragrant lorsque l'on considère des plateformes *inconnues* de CG qui ne sont pour ainsi dire pas du tout identifiées comme du trafic CG par le modèle DT alors que le modèle USAD les identifie parfaitement. La

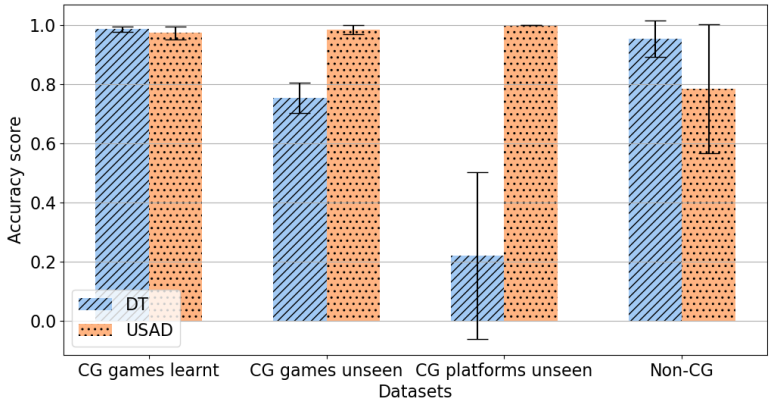


FIGURE 5.8 – Comparaison entre *DT* et *USAD*

présente étude nous a ainsi permis de montrer l'intérêt du modèle *USAD* sur le *DT*. Le recours à de l'apprentissage non supervisé nous a fait aboutir à un modèle plus *généraliste*. L'inconvénient du *DT* concerne le besoin d'exhaustivité des données d'entraînement, au moins concernant les différentes plateformes et si possible en variant les types de jeux. En effet, chaque classe de trafic doit *bien* être représentée. Il en est de même pour le trafic NCG, qui devrait inclure un panel représentatifs des services fonctionnant sur UDP. Nous aurons aussi l'occasion d'évaluer dans le prochain chapitre la capacité de chaque modèle à traiter du trafic en temps réel.

5.8 Conclusion

Le présent chapitre s'est intéressé à l'identification du trafic CG. Ce type de trafic est très sensible aux délais et aux pertes. Il est donc intéressant de lui appliquer un traitement adapté dans le réseau, mais cela nécessite son identification préalable.

Notre hypothèse est que le trafic CG dispose de caractéristiques intrinsèques dont les motifs sont identifiables à travers certaines statistiques sur les flux. Pour atteindre notre objectif, nous avons ainsi considéré 12 caractéristiques réseau, extraites périodiquement, pour chaque couple d'adresses IP. Ces caractéristiques permettent d'entraîner et de tester nos modèles. Les modèles considérés sont les classificateurs DT et RF que nous avons présentés. Tous deux parviennent à de très bons scores de classification. La reconnaissance de trafic CG en conditions réseau perturbées est plus difficile de prime abord mais devient une formalité dès lors que ces conditions de captures sont représentées dans le jeu d'entrainement. Bien que les résultats soient très satisfaisants, nous nous sommes interrogés sur la capacité de généralisation de nos modèles à d'autres dimensions (jeux et plateformes). Nous avons ainsi comparé le DT et USAD, un modèle d'apprentissage non supervisé. Alors que le premier ne reconnaît pas les nouvelles plateformes de CG, le second les identifie parfaitement. Cela montre qu'un modèle supervisé comme DT doit être entraîné sur un jeu de données représentatif.

Une fois les modèles entraînés, il convient de les mettre en production pour évaluer leur capacité à traiter du trafic à la vitesse du réseau. Nous pouvons dès lors nous questionner sur l'architecture à mettre en place. Le chapitre suivant traite de ces questions.

Chapitre 6

Évaluation des stratégies d'implantation

Sommaire

6.1 Introduction 95
6.2 Architecture à base de micro-services 96
6.2.1 Présentation des micro-services 96
6.2.2 Évaluation des performances 97
6.3 Programmabilité réseau grâce à P4 99
6.4 Classification en P4 logiciel depuis le *Data Plane* 102
6.4.1 Testbed 102
6.4.2 Extraction des caractéristiques en P4 102
6.4.3 Implantation d'un DT en P4 104
6.5 Approche de classification *Hybride* 105
6.5.1 Matériel et limitations 105
6.5.2 Impact des approximations 105
6.5.3 Architecture finale 107
6.6 Conclusion 108

6.1 Introduction

Nous avons présenté nos classificateurs au cours du chapitre précédent et montré qu'ils offrent des performances d'identification satisfaisantes. Une fois les modèles entraînés, il convient de classifier le trafic *à la volée*, c'est à dire à la vitesse où les données transitent en un point du réseau de l'opérateur. Ceci nécessite de mettre au point une architecture efficiente capable de tenir cette charge avec une utilisation raisonnable de ressources de calcul. Les architectures de traitement au niveau du réseau s'inscrivent en général sur deux plans ; le *plan de données* et le *plan de contrôle*. Le plan de données désigne les appareils réseau chargés de transmettre le trafic. Les éléments constitutifs du *plan de données* opèrent sur des paquets réseau. Le plan de contrôle désigne quant à lui la "*logique*" du réseau. Il dicte au reste des composants le comportement qu'il leur convient d'adopter.

Nous concevons et évaluons dans un premier temps une architecture purement logicielle basée sur les une *micro services*, en suivant le paradigme de virtualisation des fonctions réseau (NFV).

Celle-ci est présentée dans la section 6.2. Le principe consiste à répartir un traitement en plusieurs sous-services. Leur association porte le nom de *Service Function Chains* (SFC). Le procédé facilite la maintenabilité, le déploiement et le passage à l'échelle. De surcroît, chacun des composants est indépendant. Il est donc possible de les développer dans de langages de programmation distincts.

Dans un second temps, nous verrons dans les sections 6.3 et 6.4 comment le langage P4 pourrait permettre d'accélérer certaines fonctions de l'architecture mais aussi les limitations actuelles qui restreignent fortement son usage. Nous proposerons alors une architecture hybride tirant au mieux partie des forces des micro-services et du langage P4 dans la section 6.5.

6.2 Architecture à base de micro-services

Dans la présente section, nous présentons une architecture s'appuyant intégralement sur des fonctions réseau virtuelles (VNF). En d'autres termes, aucun traitement n'est appliqué depuis le *plan de données*. Nous commençons par présenter l'architecture globale et les micro-services la composant. Ensuite nous évaluons les performances de la solution. À noter que l'ensemble du code constituant cette architecture est open source et mis à disposition de la communauté [26].

6.2.1 Présentation des micro-services

Comme nous pouvons le voir sur la Figure 6.1, notre architecture est composée de 4 éléments. Tout à gauche figure la *sonde* qui fait face au trafic et qui a pour mission de calculer les statistiques des flux. Elle a été développée en langage `C++` [27] par Xavier Marchal. Il utilise deux *threads* : un d'extraction se chargeant de lire les paquets pour en extraire les valeurs brutes, et un de calcul se chargeant du calcul des statistiques sur la fenêtre temporelle considérée. Un rapport est envoyé périodiquement, pour chaque couple d'adresses IP rencontré. Il est composé des 12 caractéristiques (*features*) présentées en Section 5.3.2. Un champ permettant d'identifier le flux concerné vient le compléter. Chacun des rapports est transmis via le protocole `UDP` au prochain composant. Le "*Load Balancer*", est ainsi chargé de les réceptionner. Le but de ce composant est de répartir la charge sur les ressources de calcul disponibles. Plusieurs stratégies peuvent être implantées. Nous avons opté pour un simple algorithme du *tourniquet* (ou *Round-Robin*). Les "*Compute Node*" présentés en Figure 6.1 symbolisent les ressources de calcul. Chacune de ces instances est exécutée dans un processus à part. Ce composant exécute le modèle préalablement entraîné, tel le *DT* ou *USAD*. Chacune des instances prend en entrée un ensemble de caractéristiques, et lui associe un résultat de classification (CG ou NCG). Le résultat susvisé est acheminé vers le dernier composant ; l'"*Aggregator*". Ce dernier associe à chaque *flux* l'ensemble de ses prédictions, qui peuvent avoir été réalisées par différents nœuds de calcul. Il est possible de spécifier une certaine politique quant à la labellisation finale d'un flux. Par exemple, la décision peut attendre qu'au moins n rapports aient été reçus. Un seuil Thr peut de plus être défini quant au pourcentage minimum de rapports labellisés CG pour labelliser le flux en tant que tel. Enfin, un intervalle de temps minimum t peut être défini avant que le flux ne soit réévalué. Ceci permet de consacrer les ressources à la labellisation d'autres flux, et évite les changements intempestifs de label.

Dans notre cas, l'ensemble des composants de la Figure 6.1 est exécuté sur une même machine. L'interface locale `lo` est ainsi utilisée comme jonction entre la sonde et le serveur UDP. Des *pipelines* linux permettent de relier le reste des composants entre eux. L'architecture peut néan-

26. `https://github.com/mosaico-anr/CG_Classifier`
27. `https://github.com/Nayald/probe-CG-detection`

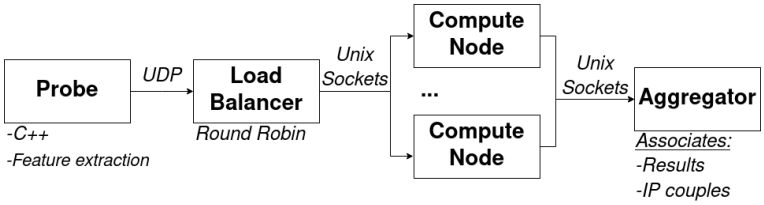


FIGURE 6.1 – VNFs : vue de l'architecture globale

moins être adaptée de sorte à ce que chaque composant soit exécuté sur une machine distincte. Il suffit alors de remplacer les *pipelines* par des *sockets*.

6.2.2 Évaluation des performances

Nous pouvons dès à présent nous intéresser aux performances de cette architecture. Les *performances* désignent ici la quantité de données pouvant être traitées par seconde. La précision des modèles a déjà été abordée lors de la section 5.6. Pour mener à bien notre tâche, nous commençons par évaluer les performances de la sonde qui fait face au trafic, car de sa capacité à extraire rapidement les statistiques dépendent les performances du reste de l'architecture. Ceci fait, nous porterons notre attention sur le reste des composants.

Performances de la sonde

Nous générons du trafic réseau de sorte à évaluer la rapidité de la sonde. Pour ce faire, nous avons recours à deux serveurs *Arch Linux*. Tous deux disposent de deux CPU *Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz*. Le lien réseau dispose quant à lui de 2x10 Gbps aggrégés. Il permet de faire transiter du trafic `iperf` entre les deux serveurs. Le trafic susvisé est bidirectionnel ; chaque appareil est donc émetteur et récepteur. La sonde est placée sur un des deux serveurs. Nous comptabilisons 30 Gbps de capture au niveau de la *socket* `PF_PACKET`. Le *thread* de capture utilise alors $\approx 60\%$ d'un coeur CPU. Notons que la taille des paquets n'a pas d'incidence sur les performances. En effet, seuls leurs 82 premiers octets sont capturés.

Au cours de nos expériences, nous générons systématiquement un débit de 10 Gbps. Une telle valeur est commune pour des réseaux d'accès. Nous commençons par considérer 1000 flux de 10 Mbps chacun. Ceci fait, nous générons 2000 flux de 5 Mbps. Les performances décroissent entre les deux expériences. Dans le premier cas, la sonde parvient à générer 29100 $rapports.s^{-1}$. Cette valeur est très proche de ce qui est attendu, i.e. $\frac{1}{0.033} \times 1000 \approx 30303$. Le *thread* d'extraction en est alors à $\approx 70\%$ d'utilisation CPU. Ce pourcentage se situe à $\approx 37\%$ pour le *thread* de calcul. Lorsque l'on double le nombre de flux, la quantité de rapports à produire double aussi. Le nombre de rapports générés par la sonde ne double cependant pas ; il se situe à ≈ 49800 $rapports.s^{-1}$. Ceci trahit l'existence d'un engorgement. Il n'est cependant pas aisé d'identifier le facteur limitant. Le *thread* d'extraction semble utiliser 100% de son cœur. Le *thread* de calcul atteint quant à lui les 70% d'utilisation. Malheureusement, les instances d'`iperf` qui servent à générer le trafic consomment également beaucoup de ressources ce qui ne permet pas de conclure facilement quant au facteur limitant. L'ensemble des cœurs leur ayant été attribués dépassent alors les 94% d'utilisation.

Évaluation du reste des composants

Une fois la sonde évaluée, nous nous intéressons au reste des composants. On peut dès lors se référer à la Figure 6.1. La "*Probe*" est remplacée par un générateur de rapports aléatoires. Il a pour consigne de générer 40000 rapports par seconde. Le débit supporté est calculé au niveau de l'"*Aggregator*". Soit $T = t_{Last} - t_{First}$ le laps de temps entre la réception du premier et du dernier rapport. Nous désignons par Nb_{rep} le nombre de classifications effectuées dans cette fourchette de temps. Le débit "*de sortie*" est donné par le rapport entre Nb_{rep} et T. Il doit se situer au plus près du débit *d'entrée*, i.e. 40000. L'évaluation prend place sur une tierce machine, disposant de 8 cœurs physiques. Le CPU est un *Intel(R) Core i7-9700 CPU @3.00 GHz*.

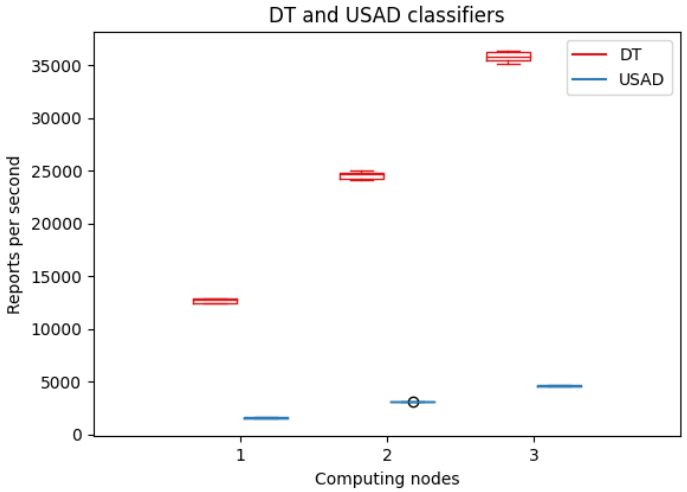


FIGURE 6.2 – VNFs : analyse de performances

La Figure 6.2 donne un aperçu de nos résultats. L'axe des ordonnées désigne le nombre de classifications effectuées à la seconde. Cette grandeur devrait idéalement valoir 40000. L'axe des abscisses désigne le nombre de nœuds de calcul instanciés. Ce sont les "*Compute Node*" présents au centre de la Figure 6.1. Les labels *DT* et *USAD* servent à déterminer le modèle utilisé pour classifier. Chaque expérience s'étale sur une durée de 30 secondes. Elles sont toutes répétées 5 fois, de sorte à vérifier la présence d'éventuelles fluctuations.

L'ajout de ressources de calcul améliore grandement les performances. Lorsqu'un seul nœud est déployé, le DT parvient à traiter $\approx 32\%$ des rapports. L'ajout d'un nœud supplémentaire fait doubler les performances de classification. Il en est de même pour le modèle USAD, passant d'environ 1623 à 3141 classifications par seconde. L'évolution des performances selon le nombre de nœuds semble bien linéaire, ce qui était attendu.

Les différences de performances entre DT et USAD sont très importantes. Le temps d'inférence est bien plus long pour le modèle d'apprentissage profond qui est beaucoup plus lourd à exécuter. En l'état, il y a un ordre de grandeur quant au nombre de rapports traités entre les deux modèles à ressources de calcul équivalentes. Le recours à une carte graphique pour accélérer les calculs (GP-GPU) pourrait cependant accroître les performances du classificateur USAD.

Nous avons vu que l'objectif des 10Gb/s initialement fixé en vue d'un déploiement de notre solution en périphérie de réseau est atteignable avec une approche purement logicielle à base de micro-services et un modèle de type DT. Le facteur limitant semble ici être la sonde qui fait face au trafic. Nous allons donc voir dans les sections suivantes si le fait de déporter une partie des

traitements dans le plan de données pourrait améliorer les performances.

Nous commençons par dresser un rapide état de l'art des études proposant des traitements grâce à P4. Nous y décrivons notamment des moyens permettant de classifier depuis le *plan de données*.

6.3 Programmabilité réseau grâce à P4

Comme nous l'avons souligné précédemment, le réseau peut être conceptuellement scindé en deux fonctions logiques. Nous distinguons dès lors le *plan de données* du *plan de contrôle*. Cette séparation est reprise par le paradigme SDN [NMN+14]. Ce dernier permet de reconfigurer dynamiquement les équipements réseau. De nombreux efforts ont été entrepris en vue de passer outre le problème de compatibilité. Nous pouvons citer le protocole OpenFlow [MAB+08], permettant la communication entre un contrôleur et le plan de données. L'avènement du langage P4 [BDG+13] étend encore davantage la programmabilité réseau. Il permet de déployer des programmes sur les appareils du *plan de données*. C'est un langage spécifique au domaine (DSL) du réseau et plus précisément au traitement de paquets. Il est ainsi possible de définir comment les équipements d'interconnexion (*switchs*) doivent parser les paquets réseau. Cette propriété permet de s'acquitter d'une quelconque dépendance vis-à-vis des protocoles réseau existants. En complément, un code P4 est indépendant de l'appareil devant l'exécuter. Les spécificités de la machine cible sont prises en compte lors de la compilation. On aboutit au final à un programme adapté à son environnement d'exécution.

De nombreux articles s'intéressent à l'élaboration de programmes *complexes* dans le *plan de données*. Des structures approximatives sont alors utilisées pour pallier les contraintes inhérentes à P4. Comme nous pouvons le voir sur la Figure 6.3, P4 a recours à des tables de type *Match Action*. Ces tables permettent d'associer des actions à des conditions. Au cours de son cheminement au travers du *switch*, un paquet traverse plusieurs étapes ; on parle de *stages*. Chacune de ces étapes est isolée des autres. Dans [SNR+16], Sivaraman & al. énumèrent trois limitations propres à P4. Pour chaque *stage* :

1. le temps d'exécution ne doit pas dépasser 1ns ;
2. les accès mémoire sont limités (*lecture/modification/écriture*) ;
3. la mémoire disponible est restreinte.

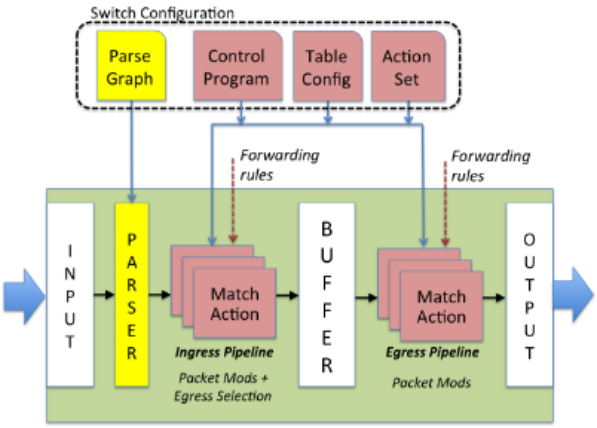


FIGURE 6.3 – P4 : vue d'ensemble [BDG+13]

Les références [SNR+16, dSJPG18, HCGR18] cherchent à détecter des *Elephant flows*. Ce sont des flux *massifs* qui tendent à encombrer la bande passante. Dans [SNR+16], les auteurs ont recours à un *pipeline* de tables de hachage. Chaque flux est alors représenté par un *hash*. Des registres répartis sur plusieurs *stages* font la correspondance entre un flux et son compteur. Il est possible que deux flux différents aient un *hash* identique. On parle alors de collision. Dans un tel cas, la marche à suivre dépend de l'étape courante. S'il s'agit de la première étape, le nouveau flux est inséré dans le registre. Le couple (clé, valeur) qui y était jusqu'alors stocké est inscrit dans les *métadonnées* du paquet. Le procédé permet de le propager à l'étape suivante. Dans le cas général, une collision se solde par une comparaison entre couples. Le flux le plus léger par rapport à son compteur est propagé vers l'étape suivante. De la sorte, les flux les plus légers sont expulsés au fur et à mesure des itérations. L'écueil de cette méthode est que les auteurs ne tiennent pas comte de la durée des flux. Da Silva & al. présentent donc une méthode considérant en plus la durée des flux [dSJPG18]. Chaque étape est associée à une fonction de hachage h_i. Un paquet se verra donc attribuer des indexs différents au cours des différents *stages*. Ces indexs permettent de mettre à jour trois registres :

— F_t : moment où le premier paquet a été réceptionné ;
— L_t : moment où le dernier paquet a été réceptionné ;
— S : somme cumulée des tailles des paquets.

Un flux est ainsi constitué de plusieurs états, répartis sur plusieurs *stages*. Il peut y avoir des divergences du fait de possibles collisions. Les auteurs se basent sur l'ensemble de ces états pour estimer la taille et la durée des flux. La taille finale est obtenue en faisant le minimum des sommes cumulées sur les différentes étapes. La durée du flux est calculée pour chaque étape, à partir du moment de réception du *premier* paquet (*registre* F_t). Dans le cas d'une collision, la durée ne peut qu'être supérieure à la durée effective du flux. Les auteurs retiennent donc la plus courte durée calculée sur l'ensemble des *stages*. Des seuils portant sur ces deux valeurs permettent de classifier le flux courant. Dans [HCGR18], Cai & al. ont aussi recours à des seuils. Leur approche s'appuie cependant sur plusieurs *switchs*, reliés à un coordinateur central. Chaque *switch* possède un seuil propre à chaque flux. La valeur $T_{i,k}$ désigne ainsi le seuil du flux k sur le *switch* i. Dès lors qu'un flux dépasse son seuil, le *switch* concerné en informe le contrôleur. Ces seuils sont adaptables, de sorte à ne pas surcharger le lien entre *plan de données* et *plan de contrôle*. Proportionnellement, certains *switchs* peuvent être plus exposés à un flux que les autres. Les *switchs* concernés adopteront ainsi un seuil plus élevé que la moyenne. Le contrôleur aggrège l'ensemble des rapports reçus. Cela lui permet d'estimer le compte total associé à chaque flux. Cette valeur est ensuite comparée à un seuil global $T_G(k)$, en vue de procéder à la classification.

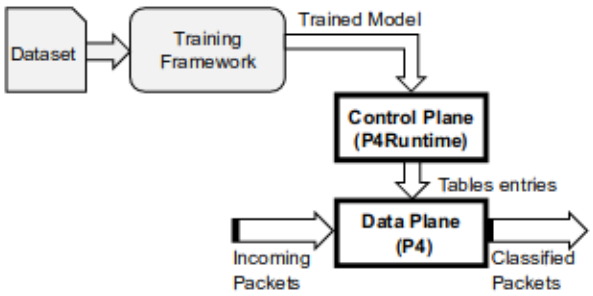


FIGURE 6.4 – P4 : mise à jour depuis le *plan de données* [XZ19]

Les références parcourues jusqu'alors présentent des traitements assez simples. Des études ont cependant cherché à faire du ML depuis le *plan de données* [SB18, SSB18, XZ19, KS21].

Des nombreuses raisons viennent conforter cette idée. Tout d'abord, la classification depuis le *plan de données* est moins énergivore [TDP+19] que d'autres alternatives. De surcroît, la phase d'inférence se fait très rapidement. Dans [SSB18], Sanvito & al. s'intéressent au ML distribué. Le principe consiste à répartir un réseau neuronal sur plusieurs noeuds. La latence les séparant constitue cependant un inconvénient. Dans cette optique, les auteurs proposent d'effectuer une partie des traitements sur les appareils réseau. Une simplification du modèle est réalisée pour répondre aux exigences de P4 [HCS+16]. Elle limite le nombre de bits utilisés par les couches *denses* sans trop détériorer l'*Accuracy*. Les auteurs se basent sur les travaux de Siracusano & al [SB18] pour implanter leur modèle. L'outil utilisé (`N2Net`) leur permet de configurer une puce étant donné un réseau neuronal. Les résultats intermédiaires transitent d'une machine à l'autre via les paquets réseau (*encapsulation*). Dans [XZ19], Xiong & al. cherchent à implanter des modèles de ML dans un *switch*. Les modèles considérés sont entraînés au préalable. Le *switch* doit *juste* pouvoir effectuer la phase d'inférence. Contrairement à l'étude menée en [SB18], les réseaux neuronaux ne sont pas pris en charge. La mise à jour des modèles se fait par le biais du contrôleur. Pour ce faire, les caractéristiques considérées doivent cependant rester les mêmes. Dans [KS21], Kamath & al. prennent en compte 4 caractéristiques pour classifier des flux. Ils considèrent respectivement :

— la taille du flux ;
— le nombre de paquets composant le flux ;
— les IATs entre paquets ;
— la durée du flux.

La classification a lieu depuis le *switch*. Le résultat est envoyé au *plan de contrôle* pour qu'il adapte les règles de routage. Celles-ci sont ensuite propagées à l'ensemble des *switchs* pour être mises en œuvre. Le modèle de classification considéré est le DT, du fait de sa simplicité. Les auteurs lui apportent cependant quelques modifications pour limiter son impact mémoire. Deux architectures sont ainsi proposées ; `SMASH-D1` et `SMASH-D2`. La classification se base sur une ou deux conditions selon l'architecture considérée. Dans le cas d'un modèle arborescent, les feuilles de l'arbre sont associées aux classes. Les conditions concernent alors le ou les noeuds précédant chaque feuille. Les seuils associés aux noeuds sont aussi adaptés, par souci de simplification. De nombreuses contraintes matérielles portent sur les équipements réseau. Ces mêmes contraintes entravent le déploiement de modèles de ML. Les programmes sont alors simplifiés de sorte à pallier le problème. Cela ne va pas sans occasionner une détérioration des performances de classification (typiquement, une baisse de l'*Accuracy*). De surcroît, le nombre de *caractéristiques* et de *classes* à prédire est fortement limité [XZ19].

Certaines approches relèguent les traitements les plus complexes au *plan de contrôle* [SCP20, GML+22]. Dans [SCP20], Simpson & al. cherchent à prédire l'algorithme de congestion régissant un flux TCP. La classification se base sur les temps d'inter-arrivées (IATs). Pour ce faire, l'horodatage du *plan de données* est mis à profit. D'après [KSB+20], les appareils situés dans le *plan de données* engendrent les horodatages les plus précis. Des histogrammes agrégeant ces informations sont envoyés vers le *plan de contrôle* pour procéder à la classification. Dans [GML+22], Gombos & al. cherchent à implanter deux AQMs dans un *switch* programmable. Les deux AQMs concernés sont DualPI2 [ADSB+19] et PI2 [SBTB16]. Les traitements les plus complexes sont encore une fois relégués au *plan de contrôle*. Le délai d'attente est monitoré en comparant les horodatage d'entrée et de sortie des paquets. Le *plan de contrôle* consulte périodiquement cette valeur pour pouvoir calculer p', la probabilité d'envoyer un signal de congestion. La complexité de la formule rend son calcul impossible depuis le *plan de données*. Comme mentionné dans [dSJPG18], les méthodes *hybrides* retardent l'identification des flux. Il s'agit donc de trouver un compromis entre délais et *Accuracy*.

6.4 Classification en P4 logiciel depuis le *Data Plane*

Comme nous avons vu d'après [KS21] qu'une classification par DT depuis le *plan de données* semble possible en P4, nous détaillons ici notre tentative visant à traduire une partie de notre architecture initiale en P4. Le code est d'ailleurs accessible sur *Github*[28]. L'intégralité de notre étude a été menée dans un réseau virtuel. Nous commençons par présenter notre *testbed*.

6.4.1 Testbed

Notre programme est exécuté dans un conteneur *Docker*. Le programme *Mininet* y est installé, de sorte à y émuler un réseau. La topologie est spécifiée lors du lancement du programme. Elle contient un hôte et un *switch* logiciel P4 *bmv2*. Un tel *switch* est bien moins performant qu'un *switch* P4 matériel. Nous nous en contentons cependant pour développer et tester notre code P4. Un contrôleur est chargé de déployer des tables *match-action* sur le *switch*. Ces tables permettent d'associer une condition à une action précise. On peut par exemple définir le port de sortie d'un paquet selon l'adresse IP du destinataire. On parle alors de règles de routage. D'autres types de règles peuvent néanmoins être déployés. Nous nous y pencherons en section 6.4.3.

L'hôte est chargé de générer du trafic réseau. Nous y installons `tcpreplay` en vue de rejouer des traces `pcap`. Une fois exposé à du trafic, le *switch* virtuel pourvu de notre classificateur en P4 commence à classifier. Il génère un fichier de `log`, détaillant l'ensemble des opérations effectuées. Ce même fichier permet notamment de consulter les résultats de classification.

6.4.2 Extraction des caractéristiques en P4

Pour pouvoir classifier, il faut tout d'abord extraire les 12 caractéristiques d'un flux qui ont été présentées en section 5.3.2. On y retrouve notamment des moyennes et des variances. Les limitations du langage P4 compliquent cependant leur calcul. Pour commencer, aucun type ne permet de représenter les nombres flottants. L'opération de division n'est par conséquent pas supportée. Une autre difficulté concerne la mémoire. Le calcul d'une moyenne ou d'une variance porte sur plusieurs valeurs. Il n'est cependant pas possible de toutes les garder en mémoire. Des travaux de recherche traitent justement de cette problématique [Jos23]. Nous présentons ici quelques pistes permettant de contourner ces limitations.

Division

L'opération de division n'est pas supportée par P4. On peut néanmoins s'en rapprocher en décalant des bits (*shifting*) vers la droite ou la gauche. Dans [Jr.13], Warren expose quelques manipulations bas niveau. Elles permettent de réaliser des opérations complexes à partir d'opérations sur des bits. La division et la multiplication s'y inscrivent par exemple. Dans le cas d'une division, on fait usage de puissances inverses de 2. L'équation 6.1 exprime le rapport de X par Y.

$$\frac{X}{Y} \approx X \times \sum_{\alpha \in E_Y} \frac{1}{2^\alpha} \approx \sum_{\alpha \in E_Y} X >> \alpha \tag{6.1}$$

L'ensemble E_Y contient les puissances permettant d'approximer $\frac{1}{Y}$. Le nombre de décalages et d'additions est cependant limité [SKA+17]. Il convient donc de limiter E_Y et la taille des éléments le composant ce qui fait perdre en précision. Dans notre cas, nous nous limitons à trois additions. Les règles d'approximation sont déployées au lancement du programme. C'est le contrôleur qui

28. `https://github.com/mosaico-anr/P4-classifier`

s'en charge, au moyen de tables *match-action*. Elles permettent d'associer chaque dénominateur Y à son approximation E_Y.

Calcul de moyenne

Eu égard des contraintes de mémoire, nous calculons la moyenne *à la volée*. Certains travaux ont d'ailleurs recours à une moyenne mobile exponentielle (EWMA) [NSN+17]. La formule est donnée dans l'équation 6.2.

$$\bar{x}_t = (1 - \alpha) \times \bar{x}_{t-1} + (\alpha \times x_t) \tag{6.2}$$

La valeur $\bar{x}_t$ désigne la moyenne au cours de la $t^{\text{ème}}$ itération. Tout à droite, x_t désigne la $t^{\text{ème}}$ valeur prise en compte dans le calcul de la moyenne. L'inconvénient de cette approche est que les échantillons x_i n'ont pas tous le même poids. Nous avons ainsi opté pour une approche différente. Les calculs de moyenne y sont effectués à l'aboutissement d'une fenêtre temporelle. Pour y parvenir, nous surveillons l'évolution du nombre de paquets. Nous différencions les paquets montant et descendant, comme précisé en Section 5.3.2. L'IAT moyen est donné par le rapport entre la taille d'une fenêtre (*33 ms*) et le nombre de paquets la composant. Le calcul de la taille moyenne nécessite de sommer les tailles des paquets. Au final, nous employons deux registres par direction. Un registre supplémentaire permet de stocker le *timestamp* d'ouverture t_{start} de la fenêtre courante. L'ensemble de ces valeurs est ré-initialisé dès lors qu'un paquet dépasse la fenêtre. Il suffit de comparer son *timestamp* à t_{start} pour s'en assurer.

Calcul de variance

Calculer une variance nécessite de connaître au préalable la moyenne. Par conséquent, nous ne pouvons que produire une estimation imparfaite ici. L'algorithme de Welford [29] permet d'aboutir à une estimation. Elle est donnée par l'équation 6.3.

$$\begin{cases} \sigma_n^2 = \dfrac{M_{2,n}}{n-1} \\ M_{2,n} = M_{2,n-1} + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n) \\ \bar{x}_n = \bar{x}_{n-1} + \dfrac{x_n - \bar{x}_{n-1}}{n} \end{cases} \tag{6.3}$$

La valeur $\bar{x}_n$ désigne la moyenne glissante lors de la $n^{\text{ème}}$ itération. Chaque itération correspond à la réception d'un paquet réseau. La variable n est ré-initialisée dès que la fenêtre temporelle prend fin. Ici, le nombre de divisions augmente linéairement avec le nombre de paquets. Un programme P4 est cependant limité dans son nombre d'opérations. Nous choisissons donc de procéder autrement. Pour ce faire, nous nous basons sur la dernière moyenne connue. La somme des carrés des écarts à la moyenne est mise à jour à chaque paquet. La division n'est nécessaire qu'en fin de fenêtre temporelle. De nombreux registres sont cependant nécessaires :

— pour stocker la dernière moyenne des IATs (*resp. tailles de paquets*) ;
— pour stocker le *timestamp* du dernier paquet reçu (*pour le calcul d'IAT*) ;
— pour stocker la somme des carrés des écarts à la moyenne (*IATs et tailles de paquets*).

Cela représente 5 registres par direction. Il convient d'y rajouter les 5 registres mentionnés dans le paragraphe précédent. On aboutit ainsi à un total de 15 registres. La portabilité sur un *switch* matériel reste à tester.

29. `https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance`

6.4.3 Implantation d'un DT en P4

L'implantation d'un algorithme de ML n'est pas aisée étant données les limitations de P4. Nous nous basons sur les travaux de Xiong & al. [XZ19] pour déployer notre DT sur le *switch*. Le principe d'un DT consiste à se déplacer le long d'un arbre. Un ensemble de conditions permet de déterminer le chemin à emprunter. L'inférence prend fin dès lors qu'une feuille est atteinte ; on peut alors apposer un *label*. Une feuille peut être perçue comme un ensemble de conditions la séparant de la racine. Étant donné une feuille l, nous notons $\mathcal{C}_l$ l'ensemble des conditions permettant d'y parvenir. Une même caractéristique peut y apparaître plusieurs fois ; il convient alors d'intersecter ses contraintes. Le procédé permet de réduire le nombre de tests ; on se rapporte à un intervalle par caractéristique.

Nous notons $th_{i,j}$ l'ensemble des seuils concernant 'i'. Un couple d'indices (α_i, β_i) permet de spécifier l'emplacement de x_i par rapport à ses seuils. Cela revient à écrire $th_{i,\alpha_i-1} < x_i \leq th_{i,\beta_i}$. La correspondance entre seuils et indices de la caractéristique 'i' est illustrée dans la Figure 6.5. Les valeurs $Th_{i,min}$ et $Th_{i,Max}$ représentent les bornes inférieures et supérieures des seuils de 'i'. La contrainte $x_i \leq Th_{i,1}$ revient à dire que $Th_{i,min} < x_i \leq Thi, 1$. En termes d'indices, elle s'écrirait alors ainsi ; $(1, 1)$. Des *feature rules* se basant sur des tables *match-action* servent à calculer ces indices. Chaque caractéristique 'i' se voit alors associer un couple (α_i, β_i). Pour ce faire, il est nécessaire de déployer autant de tables *match-action* que de caractéristiques. L'ensemble des couples $\{(\alpha_i, \beta_i), i \in [1; N]\}$ situe 'x' par rapport aux contraintes du modèle.

La dernière étape consiste à associer 'x' à une feuille. C'est le rôle des *action rules*. Chaque feuille 'l' donne ainsi lieu à une règle. Une règle est composée de un ou plusieurs tests portant sur les couples (α_i, β_i). Seules les caractéristiques apparaissant dans $\mathcal{C}_l$ sont prises en compte. En cas de *match*, le paquet se voit apposer le *label* associé à 'l'. Les données transitent d'une étape à l'autre via les *métadonnées* du paquet. Le résultat des *feature rules* et des *action rules* y est ainsi inscrit.

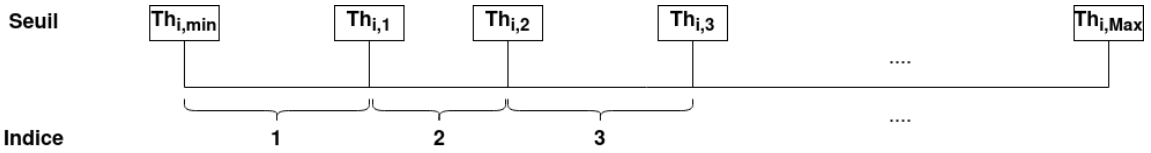


FIGURE 6.5 – Correspondance entre seuils et indices pour la caractéristique 'i'

L'ensemble des méthodes exposées ici ont été testées avec succès dans un environnement virtuel sur un *switch* logiciel *bmv2*. Nous ne sous sommes cependant pas livrés à une analyse des performances. En réalité, le langage P4 garantit un temps maximum de traitement par paquet afin de garantir les performances globales. C'est pourquoi les opérations *complexes* sont proscrites, à l'image de la division. Tout programme passant l'étape de la compilation sur un *switch* matériel respecte forcément le budget de latence imparti et peut s'exécuter à la vitesse nominale des interfaces prévues par le switch. Un *switch* logiciel *bmv2* n'a cependant pas les mêmes contraintes qu'un *switch* matériel. Nous nous en sommes rendus compte en essayant de déployer le code sur un *switch matériel P4 Tofino*. Malgré toutes les adaptations déjà effectuées et décrites dans cette section pour implanter notre classificateur en P4, les contraintes supplémentaires rajoutées par le switch Tofino empêchent toute compilation et nous ont donc poussé à adopter une approche *hybride* plus agile.

6.5 Approche de classification *Hybride*

La présente section présente une approche de classification *hybride*. Le principe général consiste à reléguer les opérations les plus complexes au *plan de contrôle* afin de contourner les contraintes les plus limitantes inhérentes au *switch matériel P4 Tofino*. Ainsi nous proposons d'implanter la sonde réalisant l'extraction des caractéristiques des flux à la vitesse de la ligne en P4, et de garder l'exécution des modèles sous forme de VNFs dans le plan de contrôle.

Nous commencerons par nous intéresser aux limitations rencontrées. Pour y faire face, nous avons été contraints d'effectuer un certain nombre d'approximations. Nous allons donc étudier leur impact sur les résultats de classification. Pour finir, nous présenterons notre solution.

6.5.1 Matériel et limitations

Nous considérons un *switch Edgecore Wedge 100BF-32X* équipé d'une puce *Intel Tofino P4*[30]. L'appareil est constitué de 32 ports supportant chacun un débit de 100 $Gbit.s^{-1}$. Les performances affichées justifient d'un placement en cœur de réseau. L'aggrégation de ses 32 ports permettrait de supporter jusqu'à 12.8 $Tbit.s^{-1}$.

Le *switch* est composé de deux parties distinctes. La première d'entre elles est dédiée au traitement des paquets réseau. Son comportement est spécifié par un programe P4. Elle correspond à la partie "*Switch Tofino P4 Chipset*" de la Figure 6.8. On distingue ensuite le "*Switch Linux OS*", chargé de déployer des programmes dans le *plan de données*. Le nombre d'opérations est cependant limité, de sorte à satisfaire les contraintes de débit. Dans notre cas, la puce *Tofino 1* permet d'effectuer au plus 12 *stages match-action* par *pipeline*. Les *stages* sont définis par le compilateur et dépendent des traitements effectués. Chaque *stage* a sa propre mémoire, ce qui complique l'usage des registres. Leur mise à jour doit ainsi être immédiate, de sorte à tenir dans un même *stage*. Notons que le *switch* met une unité mathématique "*MathUnit*" à disposition. Elle a pour but d'effectuer des opérations mathématiques *complexes*, à l'image de la racine carrée. De nombreuses contraintes compliquent cependant son utilisation, rendant même certains calculs impossibles.

6.5.2 Impact des approximations

Comme nous l'avons souligné, de nombreuses approximations sont faites lors du calcul des caractéristiques d'un flux. Ces approximations peuvent mettre à défaut nos modèles de classification tels que conçus et évalués précédemment sur des caractéristiques proprement calculées. Nous commençons donc par évaluer l'impact des caractéristiques approximées sur notre modèle USAD. Ceci fait, nous vérifions si les caractéristiques *problématiques* peuvent être ignorées.

Impact des approximations sur la classification

La Figure 6.6 représente l'impact des approximations sur la classification. L'axe des ordonnées constitue les performances de classification du modèle USAD. La reconnaissance des flux STD devient difficile dès lors que les caractéristiques sont calculées en P4. À l'inverse, les plateformes XC et PSN ne semblent pas être trop impactées. Une hypothèse serait que leur reconnaissance ne se base sur aucune des caractéristiques approximées. Notons que les *boîtes à moustache* sont plus étendues pour des caractéristiques *P4*. En d'autres termes, les performances de classification fluctuent beaucoup plus. La variance doit être la caractéristique la plus problématique. Elle est

30. `https://www.edge-core.com/productsInfo.php?cls=1&cls2=5&cls3=181&id=335`

définie comme la somme des carrés des écarts à la moyenne. Mais son calcul en P4 se base sur la dernière moyenne connue, comme mentionné en section 6.4.2. Les erreurs d'approximation de la moyenne apparaissent ainsi dans le calcul de la variance. De surcroît, elles sont élevées au carré et sommées. Il peut être intéressant de ré-entraîner les modèles en faisant abstraction de la variance. La classification se baserait alors sur 8 caractéristiques au lieu de 12.

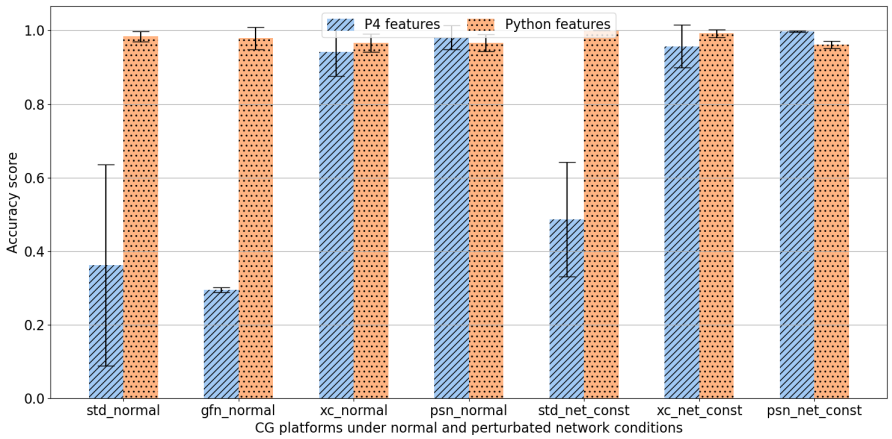


FIGURE 6.6 – Performances de classification selon le calcul des caractéristiques (modèles USAD)

Importance de la variance dans la classification

Nous limitons le nombre de caractéristiques du fait des contraintes de P4. Les nombreuses approximations nécessaires au calcul de la variance nous incitent à l'ignorer. Nous choisissons donc de ré-entraîner nos modèles sans variance. La figure 6.7 présente l'impact de cette caractéristique sur les performances de classification. Pour y aboutir, nous entraînons et testons nos modèles avec 8 (*resp. 12*) caractéristiques. Les performances sont à peu près similaires, quel que soit le *jeu de test*. De plus, le programme P4 est bien plus simple lorsque l'on s'acquitte des calculs de variance. Plus que 6 *stages* sont alors nécessaires pour traiter un paquet.

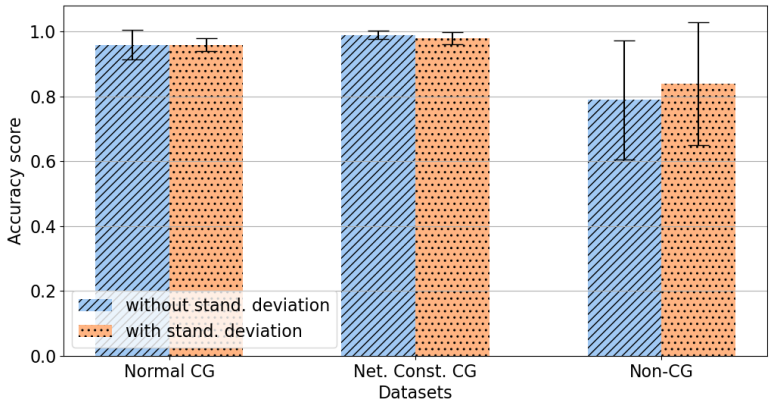


FIGURE 6.7 – Impact de la variance sur les performances de classification (modèle USAD)

6.5.3 Architecture finale

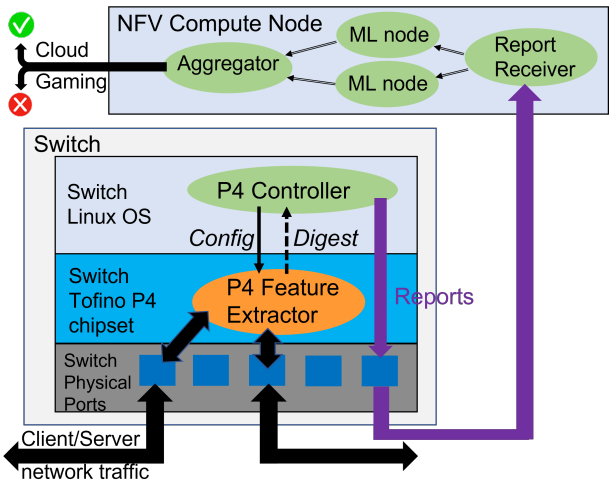


FIGURE 6.8 – Architecture proposée

Notre architecture finale est donnée par la Figure 6.8. Dans notre approche, le *plan de données* associe une fenêtre temporelle à chaque flux rencontré. Il envoie un message *digest* au contrôleur dès lors qu'une fenêtre prend fin. Le message contient les informations nécessaires au calcul des caractéristiques. Sa taille est cependant limitée à 48 octets. Il convient d'en retrancher 8 pour l'envoi des adresses IP concernées. Les 40 octets restants ne suffisent pas à acheminer 12 caractéristiques. Nous choisissons d'y apposer la somme des tailles, des IATs, et le nombre de paquets par direction. Le contrôleur calcule ensuite les moyennes, n'étant pas assujetti aux limitations de P4. Il fait parvenir les 8 caractéristiques aux VNFs de calcul, chargées de classifier les *rapports*.

Notons que le programme P4 ne peut être déclenché que par des évènements, comme la réception d'un paquet, mais pas par des *timers*. Une fenêtre temporelle ayant dépassé les 33ms ne peut donc pas être traitée immédiatement. Il convient donc d'attendre le prochain paquet pour pouvoir réagir. L'inconvénient peut se faire ressentir en présence de trafic sporadique. De grands intervalles de temps peuvent en effet séparer deux paquets consécutifs. Pour remédier à ce problème, le contrôleur se charge d'envoyer des *rapports* vides pour chaque fenêtre intermédiaire. Le nombre de rapports à envoyer est donné par l'équation 6.4.

$$Nb_{rep} = \frac{Tim_{new} - (Tim_{old} + 33)}{33} \tag{6.4}$$

Ici, Tim_{new} désigne le moment de réception du dernier *digest*. Il concerne la fenêtre temporelle appartenant à l'intervalle $[Tim_{old}; Tim_{old} + 33]$. Le paquet à l'origine de Tim_{new} déclenche la prochaine fenêtre temporelle. Notons que $Tim_{new} - Tim_{old}$ n'est pas forcément un multiple de 33. Un décalage va donc se former entre fenêtres *réelles* et fenêtres *calculées*.

Le recours au contrôleur du *switch* pourrait cependant poser des problèmes de scalabilité car, contrairement au code P4, ses performances ne sont pas garanties par la compilation.

Une solution serait de recourir à de l' *In-band Network Telemetry* (INT) [TSZ+21]. Le principe consiste à utiliser le *plan de données* pour faire remonter des informations réseau. Ces informations sont rajoutées dans les métadonnées des paquets en transit. Il convient alors de cloner le paquet clôturant une fenêtre temporelle. Le clone est démunit de sa charge utile (*payload*) et on

lui joint ensuite les informations nécessaires à la classification. Il n'y a ainsi plus d'intermédiaire entre le *plan de données* et les VNFs. Un nouveau module P4 de INT doit cependant compléter la Figure 6.8 en lieu et place de la flèche violette.

6.6 Conclusion

Dans le présent chapitre, nous avons implanté et cherché à déployer au mieux nos modèles de classification du trafic CG dans le réseau.

Une première approche à base de 4 micro-services différents a déjà permis d'obtenir de très bonnes performances en atteignant l'objectif initial de 10 $Gb.s^{-1}$ qui est nécessaire pour un déploiement en périphérie du réseau. Les VNFs calculatoires s'occupant de la classification ont montré de bonnes capacités de passage à l'échelle. Il est cependant à noter que le modèle USAD est trop coûteux pour être déployé sans accélération matérielle alors que le DT soutient sans problème le débit ciblé sur un ordinateur classique. Le goulot d'étranglement semble être la sonde faisant face au trafic pour en extraire les caractéristiques. Bien que son optimisation soit tout à fait possible, notamment en utilisant la technologie *Intel Data Plane Development Kit* (DPDK), nous avons plutôt souhaité explorer les possibilités du langage P4 pour un traitement directement au sein du plan de données.

Nous aboutissons alors à une deuxième approche, entièrement située dans le *plan de données*. Le langage P4 y est mis à l'oeuvre pour configurer des *switchs* programmables. Les limitations matérielles contraignent cependant les possibilités du langage. L'absence des nombres flottants et de certains opérateurs arithmétiques en est un exemple. Nous avons ainsi été contraints d'approximer certains calculs. L'instanciation d'un DT en langage P4 y est aussi présentée. Elle repose essentiellement sur des registres *match-action*. L'entièreté du programme a été testée sur un *switch* logiciel *bmv2*. Le portage vers un *switch* matériel nous expose à de nombreuses contraintes supplémentaires. Nous décidons donc d'adapter une approche *hybride*.

La méthode *hybride* consiste à répartir les traitements sur les deux *plans*. Les opérations les plus complexes de classification y sont reléguées au *plan de contrôle*. Le *plan de données* envoie des *rapports* pour chaque flux. Les caractéristiques les constituant sont néanmoins simplifiées ; on n'y trouve plus que des sommes, sans porter toutefois préjudice à la précision des résultats de classification si les modèles sont entraînés en prenant en compte ces contraintes. Cette approche hybride permet le déploiement de la partie sonde sur un *switch matériel P4 Tofino* pouvant atteindre jusqu'à 12.8 $Tbit.s^{-1}$ et levant ainsi le goulot d'étranglement de l'approche purement logicielle.

Nous pouvons conclure qu'une industrialisation est possible au niveau d'un opérateur, que ce soit en périphérie de réseau avec une approche purement logicielle, ou en cœur de réseau avec du matériel compatible P4 et les ressources calculatoires dimensionnées en conséquence.

La prochaine étape consiste à exploiter nos résultats de classification du trafic CG pour lui appliquer un traitement plus adapté dans le réseau. Le prochain chapitre présente et évalue plusieurs solutions possibles.

Chapitre 7

Priorisation du trafic Cloud Gaming

Sommaire

7.1 Introduction 109
7.2 Plateforme expérimentale de Cloud Gaming 110
7.2.1 Streaming de jeu vidéo 110
7.2.2 *Plugin* SCReAM 112
7.3 Évaluation du CCA SCReAM appliqué au trafic CG 114
7.3.1 Sur réseau cellulaire 114
7.3.2 Face à du trafic concurrent 116
7.4 Priorisation du trafic CG avec la discipline de file d'attente HTB 118
7.4.1 Face à du trafic Cubic 119
7.4.2 Face à du trafic BBR 119
7.4.3 Premier bilan et limites de l'approche 121
7.5 Priorisation du trafic CG avec l'AQM DualPI2 122
7.5.1 Face à du trafic Cubic 122
7.5.2 Face à du trafic BBR 123
7.6 Conclusion 124

7.1 Introduction

À l'issue des deux précédents chapitres, nous avons montré qu'il est possible d'identifier avec précision le trafic CG transitant dans un réseau et avons proposé plusieurs stratégies d'implantation répondant aux contraintes de performances d'un opérateur. La classe de trafic CG pouvant désormais être définie, le but du présent chapitre est d'en améliorer la QoS grâce à une prise en charge plus adaptée par le réseau, en particulier concernant les problèmes de cohabitation avec les CCA concurrents et de délais induits pas les buffers trop grands ("*bufferbloat*") quand le réseau est chargé.

Nous commençons par présenter en section 7.2 notre propre plateforme de CG qui fut développée par Xavier Marchal dans le cadre du projet ANR MOSAICO afin de pouvoir maîtriser l'ensemble de la chaîne : du serveur [31] au client [32], en passant par les aspects réseau [33]. Cela nous

31. `https://github.com/Nayald/game-stream-server`
32. `https://github.com/Nayald/game-stream-client`
33. `https://github.com/Nayald/game-stream-scream-proxy`

donne notamment la possibilité de tester un nouvel algorithme de contrôle de congestion pour ce cas d'usage, *SCReAM* [Joh14], dont l'un des intérêts est de supporter *ECN*. Ainsi, un équipement réseau compatible peut indiquer son niveau de congestion aux terminaux. Fort de ces informations, l'algorithme de contrôle de congestion (CCA) peut spécifier un débit cible à l'encodeur. Comme précédemment, l'ensemble du code produit est mis à disposition de la communauté.

Pour autant que nous le sachions, *SCReAM* n'a jamais été appliqué à des flux CG. Nous allons donc étudier dans la section 7.3 son comportement nominal dans le cadre du transport de trafic CG au travers de plusieurs scénarios. Nous le confronterons notamment à d'autres CCAs pour apprécier son équité vis-à-vis du trafic concurrent. Nous considérons ensuite les deux mécanismes de gestion de file d'attente différenciéss retenus à la fin du chapitre 2, à savoir la discipline de file d'attente HTB et l'AQM DualPI2 qui bénéficie d'une file L4S. Nous évaluons leur apport respectif au transport du traffic CG dans les sections 7.4 et 7.5.

7.2 Plateforme expérimentale de Cloud Gaming

Le *streaming* suit un débit spécifié au lancement du programme. Aucun mécanisme de contrôle de congestion n'est implanté par défaut. Nous nous intéressons ensuite au déploiement du CCA *SCReAM* sur la plateforme à travers un module d'extension. Cette modularité permet de départager le *streaming* du contrôle de congestion ce qui présente l'avantage de pouvoir changer de CCA indépendamment du reste de la plateforme.

7.2.1 Streaming de jeu vidéo

Une session de *streaming* englobe nécessairement un serveur et un client. Tous deux s'y échangent des données de différentes natures. Nous commençons par nous intéresser au fonctionnement du serveur en présentant son architecture et les modules la composant. Nous présenterons ensuite le client, de façon analogue.

Le serveur

La plateforme est développée pour un système d'exploitation GNU/Linux et fonctionne de manière analogue à une session de bureau à distance (RDP). Un client se connecte au serveur et y effectue des actions. Il reçoit en retour des flux audio et vidéo. Ces flux résultent de captures faites sur le serveur. Le système de fenêtrage `X11` est mis à profit pour générer la vidéo. Il permet en effet de capturer une zone de l'écran. On se retrouve alors avec des *frames* vidéo brutes `AVFrame`. L'ensemble de ces composants est représenté au bas de la Figure 7.1. On constate qu'un traitement est appliqué juste avant la phase d'encodage. L'objectif est d'adapter les *frames* au format attendu par l'encodeur. Nous avons retenu deux encodeurs : `H.264` et son successeur, `H.265`. Tous deux sont des standards largement répandus. L'encodage vidéo peut en outre être configuré pour utiliser *NVENC* afin de tirer partie de l'encodage matériel des GPUs Nvidia.

L'audio est traitée de façon analogue à la vidéo. Sa capture est réalisée par le biais du logiciel `ALSA`, partie intégrante du noyau Linux. Il met à disposition une API permettant de contrôler les pilotes de la carte son. Le *grabber* audio engendre ainsi des *frames* audio, directement transmises à l'encodeur. L'encodage se fait par le biais de l'encodeur `Opus`. Très efficient, il convient à l'encodage des communications interactives.

Nous pouvons dès lors nous intéresser à la gestion des connexions entrantes. La partie concernée se situe cette fois au centre de la Figure 7.1. Les clients commencent par se connecter au *socket_server* via `TCP`. Une *session* est alors attribuée à chacun d'entre eux. Les opérations

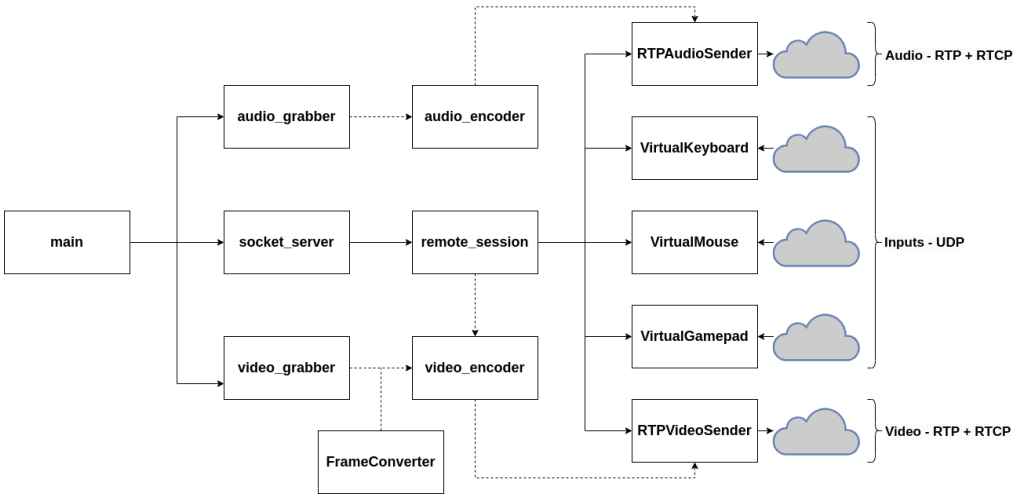


FIGURE 7.1 – Architecture du serveur de *streaming*

propres à chaque client sont gérées par leurs *sessions* respectives. Elles doivent notamment gérer les commandes de jeu et les commandes *applicatives*.
- Les commandes de jeu transitent au format *JSON*. Elles servent à exprimer les actions du *joueur*. Les trois modules chargés de les gérer sont représentés à droite de la Figure 7.1. On retrouve notamment ;

— *VirtualKeyboard* : objet chargé de gérer les actions sur le clavier ;
— *VirtualMouse* : objet chargé de gérer les actions sur la souris ;
— *VirtualGamepad* : objet chargé de gérer les actions sur la manette.

Tous trois éditent `/dev/uinput`, un module du noyau permettant de créer des appareils virtuels. Le flux concerné transite via le protocole `UDP`.
- Les commandes applicatives de la plateforme [34], c'est à dire de signalisation entre le client et le serveur, transitent quant à elles en `TCP`. Le *socket* utilisé est d'ailleurs le même que pour l'initialisation de la session. Des messages *SDP* [PHJ06] y circulent pour décrire les sessions multimédia. Ils informent notamment le client des ports vers lesquels transitent les flux audio (*resp. vidéo*). Enfin les fonctions *RTP Sender*, destinataires du flux audio (*resp. vidéo*) encodés se chargent de les transférer vers les clients concernés grâce aux protocoles RTP+RTCP qui sont adaptés au transport de flux multimédia aux contraintes *temps réel* et représentatifs des plateformes commerciales qui en font également usage (Stadia, xCloud et GeForceNow sur client web).

Le client

Intéressons-nous à présent au client CG dont l'architecture est décrite en Figure 7.2. La gestion des connexions se fait par le biais de la classe *Command Socket*. C'est ici que sont gérées les *commandes applicatives*. Elles servent à paramétrer les *RTP Receiver*, directement liés aux décodeurs. Ceux-ci génèrent des *frames* à partir des données brutes lues dans le *buffer*. Les *frames* sont ensuite acheminées vers la classe *SDL Display*. SDL signifie *Simple DirectMedia Layer*. C'est une librairie offrant une couche d'abstraction sur les composants matériels. Elle permet notamment de jouer les contenus audio et vidéo.

34. À ne pas confondre avec les commandes de l'utilisateur

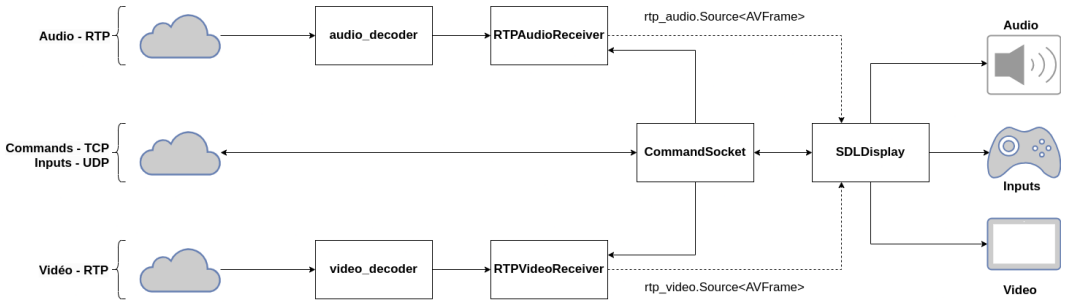


FIGURE 7.2 – Architecture du client de *streaming*

Le gestion des commandes de jeu se fait également via SDL. Un mécanisme d'interrogation (*polling*) permet d'intercepter les événements système. Il suffit de les filtrer selon leur type pour se limiter à l'essentiel. Un fichier *JSON* est ainsi créé pour chaque événement capturé. L'envoi sur le réseau est assuré par le biais de la *Command Socket*. Le protocole sous-jacent est UDP.

7.2.2 *Plugin* SCReAM

À présent que nous avons une plateforme opérationnelle, il ne nous reste plus qu'à la munir d'un CCA pour qu'elle puisse adapter son trafic aux conditions réseau. Nous avons choisi SCReAM que nous avions présenté dans la section 2.2.5. Pour y parvenir, nous faisons usage de deux proxys locaux, situés respectivement côté serveur et côté client. Le serveur de CG et son proxy sont exécutés sur une seule et même machine. Il en est de même côté client, de sorte à ne pas induire de délais supplémentaires. Nous commençons par présenter le côté serveur où réside toute l'intelligence de SCReAM.

Côté serveur

Le proxy côté serveur est de loin le plus complexe. Nous pouvons nous rapporter à la Figure 7.3 pour en avoir une illustration. Le serveur de *streaming* est représenté à gauche de la figure. Il interagit avec les composants du proxy au travers de la *loopback* (*interface* `lo`).

Les modules disposant de *sockets* réseau sont en charge de réceptionner l'ensemble des flux d'une session qui sont au nombre de 6. Ils sont représentés aux extrémités du proxy. On en distingue quatre types :

— *UDP Socket* : chargé d'émettre et de recevoir des segments UDP ;
— *TCP Client* : chargé d'émettre et de recevoir des segments TCP ;
— *TCP Server* : idem que *TCP Client*, mais n'initie pas la connexion TCP ;
— *SCReAM* : l'intermédiaire entre le CCA et les données brutes / la plateforme ;

Le type de module utilisé dépend du protocole de transport des flux. Le module *SCReAM* constitue néanmoins une exception ; il ne traite que le flux vidéo, le seul volumineux et demandant à être adapté en fonction de la congestion. Les numéros de port utilisés sont pré-définis pour chaque flux.

Mis à part la vidéo, les flux sont simplement transférés d'un module à son homologue. Un module de réception récupère la charge utile (*payload*) des paquets reçus. Il la transfert ensuite vers un module d'émission. Les modules sont reliés entre eux par des files d'attente, représentées sous forme de pointillés. Prenons l'exemple du flux audio. Initialement émis par le serveur de *streaming*, il parvient au proxy via l'interface `lo`. Il est réceptionné par le module *UDP Socket* sur le port `10000`. La charge utile du paquet reçu est transférée à un second module, chargé de

l'émettre sur le réseau. Un nouveau paquet est alors créé, à destination du client. Il est émis à partir du port `30000`. Notons que certains modules peuvent à la fois recevoir et émettre sur le réseau. C'est notamment le cas du module TCP gérant le trafic de signalisation.

Le module *SCReAM* est aussi bilatéral. Sa liaison vers le client lui permet d'envoyer des données et de recevoir des *feedbacks*. Les paquets RTP lui provenant sont immédiatement mis en file d'attente. Il s'agit d'une file *FIFO*, vidée selon les conditions du réseau. Cette file n'est pas représentée sur le schéma ; elle s'inscrit à l'intérieur du module *SCReAM*. Chaque envoi est conditionné par le CCA, dont le comportement est régi par un programme tiers[35] dont la spécification est connue [Joh14]. Il base ses décisions sur la taille de la file RTP, mais aussi sur les *feedbacks* RTCP reçus. Ces derniers lui permettent d'estimer les délais réseau, mais aussi de déceler d'éventuelles pertes. Ces informations sont transmises au CCA, qui émet alors une instruction pour l'encodeur. Nous distinguons deux types d'instruction :

— *Bitrate Request* : nouveau débit cible pour l'encodeur ;
— *I_Frame Request* : une *I_Frame* est nécessaire suite à une perte sur le réseau, afin d'éviter la propagation des artefacts visuels.

Ces instructions sont formatées de sorte à être comprises par l'encodeur. C'est le module *Bitrate Converter* qui s'en charge. Une fois les consignes formatées, le module *TCP Client* les émet sur la *loopback*. Elles sont traitées par la classe *remote_session*, visible sur la Figure 7.1 qui pilote l'encodeur vidéo.

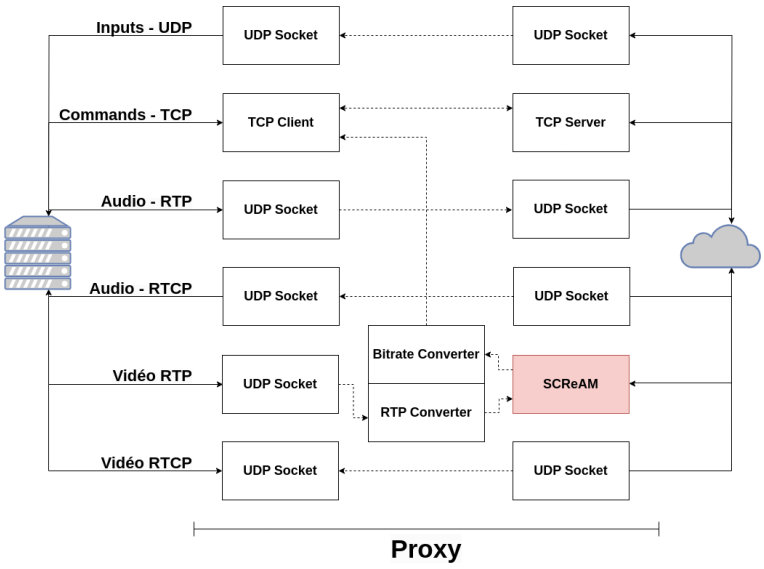


FIGURE 7.3 – Architecture du proxy "*serveur*"

Côté client

Au même titre que pour le proxy serveur, de nombreux flux sont tout simplement transférés. Seule la vidéo subit un traitement plus complexe. Lors de la réception d'un paquet, le module *SCReAM* transfère son *payload* UDP au module *RTP Converter*. Le procédé permet de transférer le contenu vidéo au client, identiquement à ce qui a été fait côté serveur. Les données sont traitées en parallèle par le CCA client. Il tient à jour un ensemble de statistiques sur le flux

35. `https://github.com/EricssonResearch/scream.git`

vidéo reçu.Un rapport RTCP contenant ces informations est régulièrement émis. Y figurent notamment les valeurs des bits `ECN-CE` des derniers paquets RTP reçus. Cette information permet à l'algorithme de contrôle de congestion prendre en compte le marquage ECN effectué par le réseau. Le module *SCReAM* est chargé de remonter l'ensemble des ces rapports au *proxy serveur* via l'émission régulière de rapports RTCP qui constituent les *feedbacks* client mentionnés auparavant. Un schéma du proxy client est donné en Annexe A.1.

7.3 Évaluation du CCA SCReAM appliqué au trafic CG

Bien que conçu pour être un CCA adapté aux média interactifs temps réel sur RTP, notamment vidéo, SCReAM n'a pour autant jamais été testé avec du trafic CG, ce qui est l'objet de cette section. Nous commencerons par nous intéresser aux réseaux cellulaires. Un scénario de mobilité sera considéré, de sorte à mettre en exergue l'adaptabilité de SCReAM qui a été notamment conçu pour les réseaux mobiles. Nous continuons cette étude par une confrontation face à des flux tcp `iperf` concurrents régis par les principaux CCA actuels à savoir CUBIC et BBR.

7.3.1 Sur réseau cellulaire

Notre banc de test se rapproche de ce qui a été présenté dans le chapitre 4. Un routeur séparant le serveur du client est chargé d'émuler les conditions réseau. Le trafic ne transite cependant pas sur Internet ; un seul *hop* sépare les deux terminaux. Deux captures réseau sont réalisées de part et d'autre du routeur, constituant le *goulot d'étranglement*. En plus du débit effectif, elles permettent d'estimer le temps d'attente et le taux de pertes. Les traces `txops` utilisées sont les mêmes que pour nos expériences antérieures. Dans un souci de concision, nous nous concentrons sur le scénario de mobilité, *i.e* la trace *Highway*, que nous avions identifié comme étant la plus difficile. Les résultats obtenus pour les autres traces sont disponibles en Annexe C. Chacune des expériences porte sur les 10 premières minutes de chaque capture.

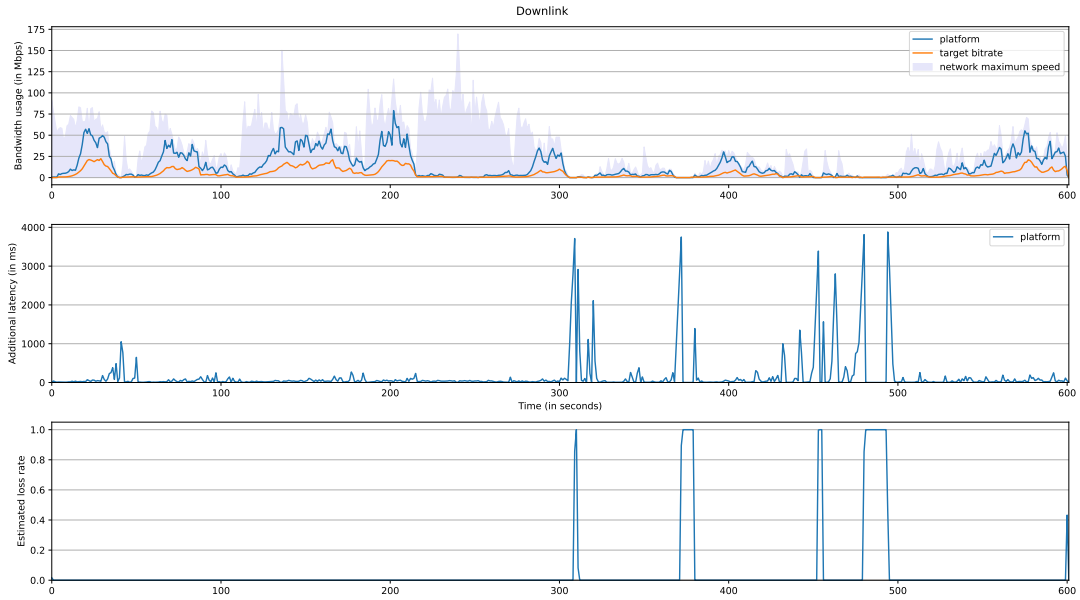


FIGURE 7.4 – SCReAM sur un réseau cellulaire : trace `txops` *Highway*

La Figure 7.4 retrace l'évolution du débit, des délais et du taux de pertes. La courbe orange correspond au débit cible, *i.e* la consigne de débit communiquée par SCReAM à l'encodeur. On constate que le débit effectif se situe généralement au-dessus du débit préconisé. L'encodeur utilisé (`H.264`) en tient compte sans s'y conformer entièrement.

On constate une brusque chute des capacités du lien aux alentours de la seconde 30. Il s'ensuit une hausse des délais, déclenchant une réaction de SCReAM. Débit cible et débit effectif sont immédiatement abaissés, évitant ainsi toute perte de données. Des pertes surviendront plus tard, en dépit de la réactivité de SCReAM. La perte intermittente de la bande passante les rend d'ailleurs inévitables sur cette trace.

Jusqu'à la seconde 300, on constate que les délais sont très bien contenus malgré la fluctuation de la bande passante disponible, signe que SCReAM arrive très bien à adapter son débit. Une remarque peut être toutefois soulevée quant au temps de récupération. Il semble que SCReAM ait parfois du mal à se remettre des périodes de carence, par exemple à la seconde 215. On y constate une brusque baisse des opportunités de transmission. Cette baisse est néanmoins passagère ; il s'ensuit une prompte résurgence de la bande passante. SCReAM tarde cependant à rehausser son débit ; il faut attendre la seconde 280 pour le voir augmenter. La QoE est impactée par une baisse de la qualité vidéo due au débit sous-dimensionné.

Entre les secondes 300 et 500, on peut constater plusieurs pics de latence concomitants avec des périodes où le taux de perte est de 100%. Ceci est causé par une perte de connectivité dans la trace *Highway*. Notons que les délais d'attente sont calculés au moment d'émission des paquets. Nous ne pouvons donc pas les afficher durant les périodes *de vide* (sans aucun `txop` disponible). Cela explique par exemple l'écart de temps séparant les deux pics entre 475s et 500s et où le taux de pertes atteint justement les 100%.

	CCA : néant	CCA : Scream
Latence moyenne	551.78ms	**159.21ms**
Pourcentage >10ms	87.87%	**78.24%**
Pourcentage >20ms	81.73%	**61.63%**
Pourcentage >50ms	58.47%	**30.90%**
Pourcentage >100ms	46.35%	**17.94%**

TABLE 7.1 – Étude des délais : avec/sans CCA sur *Highway*

Le Tableau 7.1 synthétise les statistiques des délais mesurés sur la plateforme avec ou sans CCA SCReAM. La colonne de gauche correspond au trafic CG sans algorithme de contrôle de congestion. Le serveur y maintien son débit à 30 Mbps, indépendamment des conditions réseau. La colonne de droite correspond aux délais affichés en Figure 7.4. Les deux expériences ont été réalisées dans les mêmes conditions.

On constate que les délais moyens sont trois fois moins élevés avec SCReAM. On passe d'une latence moyenne de 551.78ms à 159.21ms. Comme nous l'avons vu précédemment, les délais peuvent être sujets à de fortes hausses en cas de perte du lien. Des valeurs aberrantes viennent donc impacter la moyenne. Dans le cas de SCReAM, seul 17.9% de paquets dépassent le seuil fatidique des 100ms. Ce pourcentage s'élève à 46.35% en l'absence de CCA. SCReAM permet donc bien de limiter les délais, mais aussi le taux de pertes et d'améliorer significativement la QoS, même dans des conditions particulièrement difficiles.

Il ne peut bien évidemment pas faire de miracle quand la connectivité 4G est perdue. Pour cause des mauvaises conditions réseau, nous n'avons pu achever aucune session de jeu. L'image s'est par moments retrouvée bloquée, rendant impossible toute action du joueur.

7.3.2 Face à du trafic concurrent

Nous étudions à présent le comportement de SCReAM face à un flux concurrent. Un flux TCP `iperf` est alors utilisé pour induire de la charge réseau. Nous le munissons à tour de rôle des CCAs *Cubic* et *BBR*. Tous deux sont très répandus, et implantés par défaut dans le noyau Linux. Un aperçu de notre *testbed* est illustré par la Figure 7.5. On constate que l'ensemble des flux transite par un routeur intermédiaire qui permet de configurer dynamiquement les propriétés du lien, par le biais de règles `tc`. Le routeur constitue ainsi le *goulot d'étranglement*. Sa file d'attente est dépourvue de tout gestionnaire de file. Il s'agit donc d'une simple file "*droptail*", que nous limitons à 250 paquets. Une même interface réseau lie le routeur aux *entités* génératrices de débit. Les entités suvisées sont le client `iperf` et le serveur de CG. Toutes deux sont représentées à droite de la Figure 7.5.

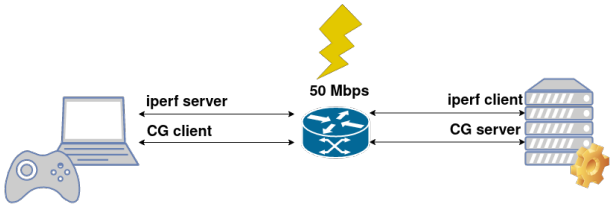


FIGURE 7.5 – Testbed : SCReAM face à du trafic concurrent

Nous décidons de rajouter des délais, de sorte à ce que nos tests soient représentatifs d'une vraie plateforme opérant sur Internet. Nous optons pour un RTT de 30ms. Pour ce faire, nous rajoutons 15ms aux deux interfaces réseau concernées en leur appliquant une règle `tc-netem` visant à retarder l'envoi des paquets du serveur vers le routeur et du routeur vers le serveur.

Nous choisissons de limiter la bande passante à 50 Mbps. Cette valeur est proche du débit moyen des réseaux 3G/4G français. D'après le dernier rapport de l'ARCEP, il se situerait à environ 63 Mbps [ARC23a].

Dans nos expériences, le flux TCP `iperf` se voit attribuer des consignes de débit croissantes. Chaque consigne s'étale sur une minute, de sorte à ce que les débits aient le temps de converger. Elles prennent à tour de rôle les valeurs suivantes ; 10Mbps, 20Mbps, 30Mbps, 40Mbps, 45Mbps. Avant d'enclencher les flux concurrents, nous observons le trafic CG pendant une minute. À l'issue des expériences, nous extrayons comme à l'accoutumée des statistiques portant sur le débit, les délais de mise en file d'attente et les pertes. Seules les 5 dernières minutes sont prises en compte dans le calcul des statistiques, c'est à dire en présence de trafic concurrent. Nous les présentons en Table 7.2.

Face à du trafic Cubic

La Figure 7.6 affiche les résultats obtenus en opposant SCReAM à *Cubic*. Les traits rouges verticaux sont tracés toutes les minutes. Ils correspondent aux changements de consigne de `iperf`. On voit que le lien commence à être saturé à partir de la seconde 120. Le flux TCP a alors un débit de 20 Mbps, tandis que le trafic CG a été légèrement baissé. Dans la suite, TCP subtilise progressivement la bande passante. Chaque hausse de son débit se fait au détriment du trafic CG. L'étude du troisième graphe trahit l'absence totale de pertes. C'est pour cette raison que `iperf` respecte scrupuleusement ses consignes de débit. En l'absence de signal de congestion, *Cubic* peut émettre sans problème. À l'inverse, la hausse des délais de *queuing* fait réagir SCReAM, qui baisse alors son débit jusqu'au minimum en dégradant la vidéo, ce qui engendre une QoE déplorable.

FIGURE 7.6 – SCReAM face à du trafic *Cubic*

		No CCA vs Cubic	SCReAM vs Cubic	SCReAM vs BBR
Trafic CG	Débit moyen	29.65 Mbps	16.11 Mbps	20.57 Mbps
	Premier quartile	27.59 Mbps	5.73 Mbps	16.1 Mbps
	Deuxième quartile	30.1 Mbps	16.18 Mbps	17.82 Mbps
	Troisième quartile	32.04 Mbps	26.02 Mbps	25.85 Mbps
Trafic TCP	Débit moyen	15.09 Mbps	28.74 Mbps	24.46 Mbps
	Premier quartile	10.52 Mbps	19.41 Mbps	19.7 Mbps
	Deuxième quartile	15.58 Mbps	29.81 Mbps	29.26 Mbps
	Troisième quartile	18.38 Mbps	42.37 Mbps	31.8 Mbps
Délais	Latence moyenne	19.02 ms	38.93 ms	29.51 ms
	Variance	7.26	25.25	13.18
	Pourcentage > 10ms	90.67%	87%	91.67%
	Pourcentage > 20ms	41.67%	74%	70.33%
	Pourcentage > 50ms	0%	27.67%	5.67%
	Pourcentage > 100ms	0%	0%	0%
Pertes	Pourcentage de pertes	0.35%	0.02%	0%
Lien	Pourcentage d'utilisation	89.47% (59.3% CG + 30.17% TCP)	89.69% (32.22% CG + 57.47% TCP)	90.06% (41.13% CG + 48.93% TCP)

TABLE 7.2 – Étude des délais : SCReAM face à des flux concurrents

Les délais entre les deux flux sont identiques car ils se partagent la même file d'attente. On se situe en moyenne à 38.93ms, comme nous pouvons le voir sur le Tableau 7.2.

Nous avons conduit la même expérience en désactivant SCReAM. Les graphes résultant de cette expérience sont illustrés dans la Figure 7.7. Les délais sont moindres en moyenne car en l'absence d'adaptation du flux CG, le taux de perte très élevé oblige *Cubic* à réduire sa fenêtre de congestion. Cette situation n'est pourtant pas enviable car le taux de perte subit par le flux CG non régulé est prohibitif et nuit encore davantage à la QoE que le débit résiduel laissé par Cubic à SCReAM.

Face à du trafic BBR

Intéressons-nous à présent à la cohabitation entre SCReAM et *BBR* qui devrait être a prioi meilleure. La Figure 7.8 trace l'évolution de nos trois métriques. Nous constatons immédiatement le contraste avec l'expérience précédente. On voit que le partage de la bande passante est bien

FIGURE 7.7 – Trafic CG sans CCA face à du trafic *Cubic*

plus équitable face à *BBR*. En effet, même si celui-ci prend le dessus, il laisse davantage de place à SCReAM que Cubic. D'après le Tableau 7.2, le premier quartile du débit vaut 16.1 Mbps. C'est presque trois fois plus que ce que l'on avait mesuré au cours de l'expérience précédente. Les débits semblent d'ailleurs se stabiliser dès le 3ème palier. La consigne de 40 mbps (*seconde 240*) engendre une dernière augmentation du débit de TCP. Elle est cependant dérisoire ; il ne s'agit que de quelques Mbps. Cette légère augmentation suffit néanmoins à accroître les délais réseau. Malgré l'absence de pertes, l'accroissement des délais empêche *BBR* d'accroître davantage son débit. Le temps d'attente des paquets passe à ainsi à $\approx$ 40ms, ce qui, ajouté aux 30ms de RTT et au temps mis pour encoder et décoder les trames vidéos outrepasse le budget de 100ms visé pour maintenir une expérience réactive.

En pratique, nous ne pouvons pas contrôler la nature des flux concurrents. Le partage de la bande passante dépend des caractéristiques du *bottleneck* et des CCAs employés. Comme nous l'avons vu sur la Figure 7.6, le partage des ressources peut s'avérer injuste et se fait au détriment des flux CG dans les deux cas. C'est ainsi que Turkovic & al. préconisent l'instauration de classes de trafic [TKU19]. Selon eux, les applications à contrainte faible latence doivent être isolées du reste des flux. Dans la prochaine section, sous décidons par conséquent d'étudier l'apport de la discipline de file d'attente *HTB*.

7.4 Priorisation du trafic CG avec la discipline de file d'attente HTB

Notre configuration de `HTB` est basée sur les travaux [Jan17] décrits dans la section 2.3.4 de l'état de l'art. Au même titre que Janczukowicz, nous avons créé deux classes de trafic ; CG et BE. Les résultats de cette même autrice ont inspiré la configuration de ces deux classes. Nous garantissons ainsi un débit *minimal* de 10 Mbps au trafic CG. Cette valeur est bien en-dessous du débit de référence de plateformes commerciales. Elle ne leur empêche toutefois pas d'opérer. Pour finir, nous accordons la priorité à la classe CG, en accord avec la configuration retenue par

FIGURE 7.8 – SCReAM face à du trafic *BBR*

l'autrice. En ce qui concerne les files d'attente, nous avons recours à deux mécanismes *classless* ; `SFQ` et `pFIFO`. Nous attribuons `SFQ` à la classe CG (*resp. pFIFO à la classe BE*). Soucieux des remarques soulevées par Janczukowicz, nous prenons soin de limiter leurs tailles respectives. Nous limitons ainsi la file `pFIFO` à 250 paquets. Des limitations sont aussi appliquées pour la classe CG, de sorte à limiter ses délais. Nous fixons la taille de sa file à 63 paquets. Notre *testbed* est tel que présenté sur la Figure 7.5. La *seule* différence par rapport à la section 7.3.2 réside dans la gestion des files d'attente selon HTB à l'intérieur du routeur. Étudions à présent les résultats obtenus.

7.4.1 Face à du trafic Cubic

La Figure 7.9 retrace les résultats de l'expérience. On constate que le trafic CG maintient son débit à 30 Mbps tout en conservant des délais négligeables. D'après le Tableau 7.3, on atteint une moyenne de 1.25 ms. Les délais subis par le flux TCP sont quant à eux bien plus importants (88ms en moyenne), ce qui n'est toutefois pas dérangeant puisque le service n'est pas sensible à la latence. Ils s'accentuent d'autant plus lorsque le lien est saturé, *i.e* à partir du $2^{\text{ème}}$ palier. Le débit du flux TCP semble alors se stabiliser à $\approx$ 20 Mbps. On constate quelques *pics* de latence, ne dépassant généralement pas 150 ms. Cette valeur est cohérente avec le débit moyen observé et la taille de la file. Rappelons que la file BE est limitée à 250 paquets, soit environ 3 Mb. Étant donné un débit de 20 Mbps, il ne suffit que de 150 ms pour que la file se draine. Les délais observés nous permettent ainsi de conclure que la file `pFIFO` est engorgée. Les *pics* de latence susvisés aboutissent donc à des pertes de paquets. Nous ne les apercevons pas sur le troisième graphe car *Cubic* y réagit rapidement. Son taux de pertes moyen est très bas ; il vaut environ 0.05%.

FIGURE 7.9 – Classes de trafic et HTB : SCReAM face à du trafic *Cubic*

7.4.2 Face à du trafic BBR

Intéressons-nous à présent à la cohabitation entre SCReAM et *BBR*. La Figure 7.10 trace l'évolution de nos trois métriques. On constate que le flux CG maintient son débit initial pendant toute la durée de l'expérience. Comme souligné auparavant, ses délais de *queuing* restent très faibles et les pertes quasiment inexistantes. Il en résulte une excellente QoE malgré le trafic concurrent. Qu'il soit en concurrence avec *Cubic* ou *BBR*, HTB permet effectivement de protéger SCReAM. Les statistiques dérivées du trafic CG sont très semblables entre les deux expériences. *BBR* maîtrise cependant mieux les délais (88ms en moyenne pour Cubic vs 72ms pour BBR), et présente moins de pics de latence dépassant les 100ms.

L'étude des courbes de débit nous laisse entrevoir quelques fluctuations. Notons que le trafic CG est impacté par deux facteurs. On distingue d'une part la consigne du CCA, dépendante de la charge réseau. S'y rajoute la nature du flux vidéo à *streamer*. La complexité scénique a en effet un impact sur le débit. Les trois grandes altérations du débit (*secondes 225, 255, 300*) peuvent donc être attribuées à l'encodeur vidéo. On voit d'ailleurs que `iperf` ne tarde pas à les exploiter. Une diminution du trafic CG fait que la file `pFIFO` peut être en partie vidée. Il s'ensuit une diminution des délais et un accroissement du débit, dûs au drainage de la file. L'amélioration soudaine des conditions réseau font que *BBR* augmente ses émissions. Ces raisons peuvent expliquer la brusque recrudescence du débit de `iperf`. À ces quelques occasions, le débit va jusqu'à dépasser la consigne initiale. Dès lors que le CG reprend son débit originel, les délais de `iperf` croissent soudainement. *BBR* est alors contraint de revoir son débit à la baisse.

Intéressons-nous à présent au taux de pertes, frôlant parfois les 5%. Il peut être étonnant que *BBR* présente plus de pertes que *Cubic*. Une étude a cependant comparé la réaction des deux CCAs en présence de pertes [CCG+17]. Les auteurs montrent que la réaction de *Cubic* est bien plus intense que celle de *BBR*. Il suffirait de 0.1% de pertes pour que *Cubic* divise son débit par 10. À l'inverse, *BBR* peut soutenir le débit maximal jusqu'à 5% de pertes. Nos résultats ne sont donc pas si suprenants. Entre les deux CCAs, c'est justement *BBR* qui bénéficie du plus haut taux d'utilisation. D'après le Tableau 7.3 il serait de 36.03% contre 35.41% pour *Cubic*.

FIGURE 7.10 – Classes de trafic et HTB : SCReAM face à du trafic *BBR*

		Cubic	BBR
Trafic CG	Latence moyenne	1.25 ms	1.86 ms
	Variance	1.09	1.48
	Pourcentage > 10ms	0%	0%
	Pourcentage > 20ms	0%	0%
	Pourcentage > 50ms	0%	0%
	Pourcentage > 100ms	0%	0%
	Débit moyen	28.88 Mbps	28.22 Mbps
	Premier quartile	27.29 Mbps	26.47 Mbps
	Deuxième quartile	29.03 Mbps	28.96 Mbps
	Troisième quartile	30.8 Mbps	31.5 Mbps
	Pourcentage de pertes	0%	0%
Trafic TCP	Latence moyenne	88.87 ms	72.09 ms
	Variance	35.98	43.14
	Pourcentage > 10ms	99.34%	99.34%
	Pourcentage > 20ms	98.68%	94.06%
	Pourcentage > 50ms	82.18%	67%
	Pourcentage > 100ms	39.27%	19.47%
	Débit moyen	17.7 Mbps	18.02 Mbps
	Premier quartile	15.49 Mbps	12.9 Mbps
	Deuxième quartile	18.78 Mbps	18.28 Mbps
	Troisième quartile	20.94 Mbps	21.31 Mbps
	Pourcentage de pertes	0.05%	0.48%
Lien	Pourcentage d'utilisation	93.16% (57.75% CG + 35.41% TCP)	92.47% (56.44% CG + 36.03% TCP)

TABLE 7.3 – Étude des délais : Scream face à du trafic concurrent sur HTB, configuration 1

7.4.3 Premier bilan et limites de l'approche

Le recours à des classes de trafic a grandement favorisé l'expérience utilisateur (QoE) du CG. Quelle que soit la nature des flux concurrents, on aboutit à de très bons indicateurs de QoS pour le flux CG : le débit est élevé et stable, les délais très faibles et stables et le taux de pertes est négligeable. Les résultats sont d'ailleurs très similaires entre les Sections 7.4.1 et 7.4.2. Grâce à la mise en place d'HTB, SCReAM ne semble pas souffrir comme initialement des flux concurrents. L'ingénierie de trafic proposée est pleinement concluante pour notre cas d'étude, et nous pourrions nous arrêter là.

L'approche présentée exhibe néanmoins quelques inconvénients à considérer. Le premier d'entre eux est juridique. La régulation européenne impose en effet la *neutralité des réseaux*

[UE15]. D'après le règlement, il est possible de différencier le trafic selon ses exigences techniques pour former des classes de trafic. Il ne doit cependant pas y avoir de discrimination entre des acteurs ayant les mêmes besoins. Cela signifie que le classificateur doit reconnaître toutes plateformes de CG, existantes et futures. Le classificateur devient ainsi un composant critique dont les manquements éventuels seraient attaquables en justice. Sa maintenance, impliquant un ré-entraînement périodique doit donc être fait avec rigueur. En outre, le classificateur consomme des ressources de calcul qui doivent être déployées dans le réseau. Ce déploiement constitue un coût supplémentaire à la charge de l'opérateur.

La paramétrisation de `HTB` peut s'avérer délicate pour les FAIs. Les garanties de débit portent sur des classes de trafic et non sur des flux. Il leur revient donc de connaître la proportion de chaque classe de trafic présente dans leur réseau.

Le recours à l'AQM DualPI2 (*présenté en Section 2.3.5*) semble résoudre les problèmes susvisés. Seules deux catégories de trafic sont considérées ; le trafic *classique* et le trafic *scalable*. La distinction se fait de manière déterministe au niveau protocolaire (*selon le positionnement des 2 bits ECN*), ce qui est beaucoup plus simple pour un opérateur. Chacune de ces deux classes bénéficie de sa propre file d'attente. Une des deux files (L4S) permet de limiter les délais et le taux de pertes. Les flux concernés bénéficient en prime d'une meilleure notification du niveau de congestion du *bottleneck*. Il est donc intéressant de faire transiter le trafic CG par cette file. Une exigence porte néanmoins sur le CCA employé ; il doit être *scalable*. C'est le cas de SCReAM, qui répond aux exigences de L4S. La section qui suit s'intéresse à la plus-value de l'AQM DualPI2 pour le trafic CG.

7.5 Priorisation du trafic CG avec l'AQM DualPI2

Les travaux existants décrits en section 2.3.5 promeuvent l'usage de CCAs *scalables* pour des applications à contrainte faible latence. Leur rapidité de convergence permet d'atteindre de faibles délais tout en garantissant un bon débit. Les CCAs *classiques* et *scalables* n'interprètent pas les signaux de congestion de la même façon. Il est donc nécessaire de les séparer, car il s'ensuivrait une *famine* de flux *classiques*. Cela justifie l'usage d'une file dédiée au trafic *scalable* ; L4S. Nous avons vu que SCReAM était conforme aux *exigences de Prague*[36]. Nous décidons donc de l'appliquer au trafic CG, et munissons le *bottleneck* de l'AQM DualPI2. Notre *testbed* est identique au schéma de la Figure 7.5. La seule différence est que nous munissons le routeur de l'AQM susvisé. Nous limitons la taille maximale des deux files à 1000 paquets, soit environ 12 Mb. Le reste des paramètres est laissé par défaut. La Section 7.5.1 oppose SCReAM à *Cubic*. Nous remplacerons ensuite Cubic par BBR.

7.5.1 Face à du trafic Cubic

Comme on peut le voir sur la Figure 7.11, le trafic CG n'est pas impacté par le flux concurrent. Il maintient son débit nominal et conserve de faibles délais, valant en moyenne 2.14 ms. Ceux-ci sont assez stables, au même titre que le débit de la plateforme. L'écart avec les délais de `iperf` n'est, d'autre part, pas flagrant. D'après le Tableau 7.4, on se situe en moyenne à 5.9 ms. Il faut préciser qu'un paramètre protège la file *classique* du trafic L4S *non réactif*. Le principe consiste à limiter l'écart entre les délais de *queuing* des deux files. Le délai d'attente maximal de la file *classique* est ainsi limité. Sa borne supérieure est égale à 1.9 fois le délai maximal de la file L4S.

36. `https://l4steam.github.io/PragueReqs/Scream_L4S_requirements_Compliance_and_Objections.pdf`

Quelques remarques peuvent être formulées par rapport aux pertes observées. On constate que `iperf` subit des pertes alors même que la bande passante est suffisante (*secondes 60 à 120*). Leur fréquence d'apparition est d'ailleurs assez régulière. Si l'on met en rapport le débit et les délais d'attente de `iperf`, on se rend compte que sa file n'est pas surchargée. Sa taille moyenne vaudrait 11.94 Mbps $\times$ 5.89 ms $\approx$ 70.33 kb. On est bien loin de sa taille maximale, fixée à 12 Mb. Les pertes observées sont en fait des signaux volontaires de congestion (la partie "*active*" de l'appelation AQM). Dans le cas de la file classique, ils sont envoyés selon une probabilité p_c. La file L4S est quant à elle soumise à une probabilité de marquage p_l. Les deux probabilités dépendent des délais d'attente. On peut d'ailleurs constater une corrélation entre l'évolution des délais et du taux de pertes.

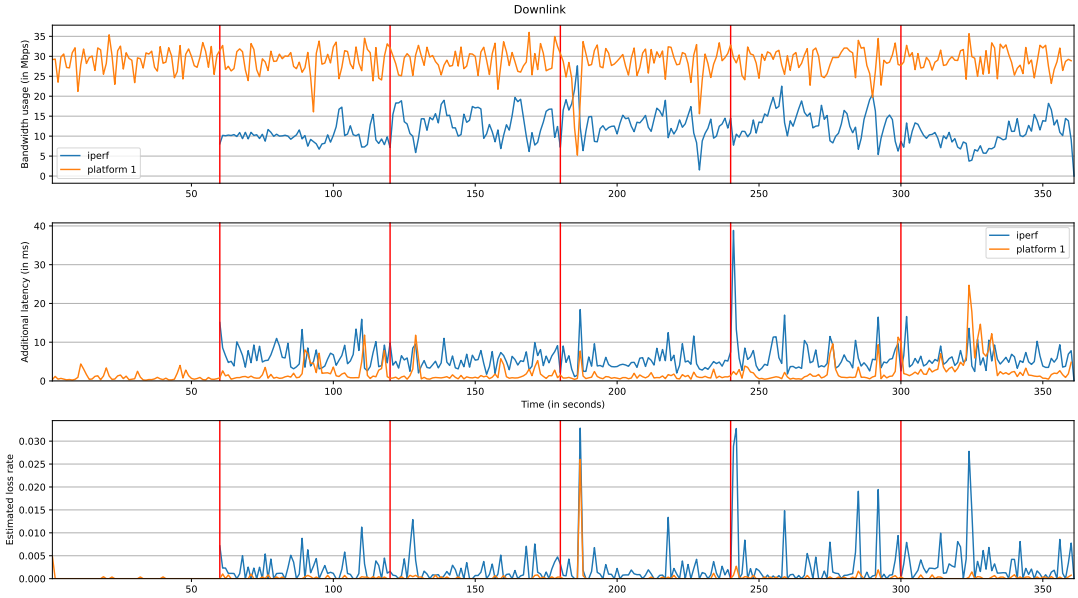


FIGURE 7.11 – DualPI2 : SCReAM face à du trafic *Cubic*

Le comportement du trafic `iperf` dépend des mécanismes de gestion des files. Au cours de la Section 7.4.1, *Cubic* présentait de très forts délais. Son débit s'articulait autour de 20 Mbps, soit la quantité de bande passante inutilisée par le CG. Les résultats de la présente expérience sont tout autres. Les délais moyens de TCP sont tout d'abord 15 fois plus faibles qu'avant. On déplore néanmoins un plus faible taux d'utilisation (*de 35.41% à 23.88%*). Les pertes de paquets causées préventivement expliquent ces diparités. On se retrouve donc en présence d'un compromis entre débit et délais. Il peut être intéressant de réaliser la même étude avec *BBR*. Le CCA susvisé est bien moins sensible aux pertes de paquets. Le partage des ressources peut donc différer.

7.5.2 Face à du trafic BBR

Intéressons-nous à la Figure 7.12. On y remarque que SCReAM baisse son débit dès que le lien est saturé (*seconde 120*). Cette diminution se fait cependant avec parcimonie. À partir de là, le débit des deux flux est relativement équitable, ce qui est une première parmi l'ensemble des expériences menées sur la cohabitation des CCAs. Le débit moyen du CG vaut $\approx$ 24 Mbps entre la seconde 120 et la fin de l'expérience. Durant ce même laps de temps, TCP atteint une moyenne de 22 Mbps. On peut se questionner sur les raisons expliquant la baisse du débit de SCReAM. Ses délais frôlent en effet la milliseconde pendant toute la durée de l'expérience. On

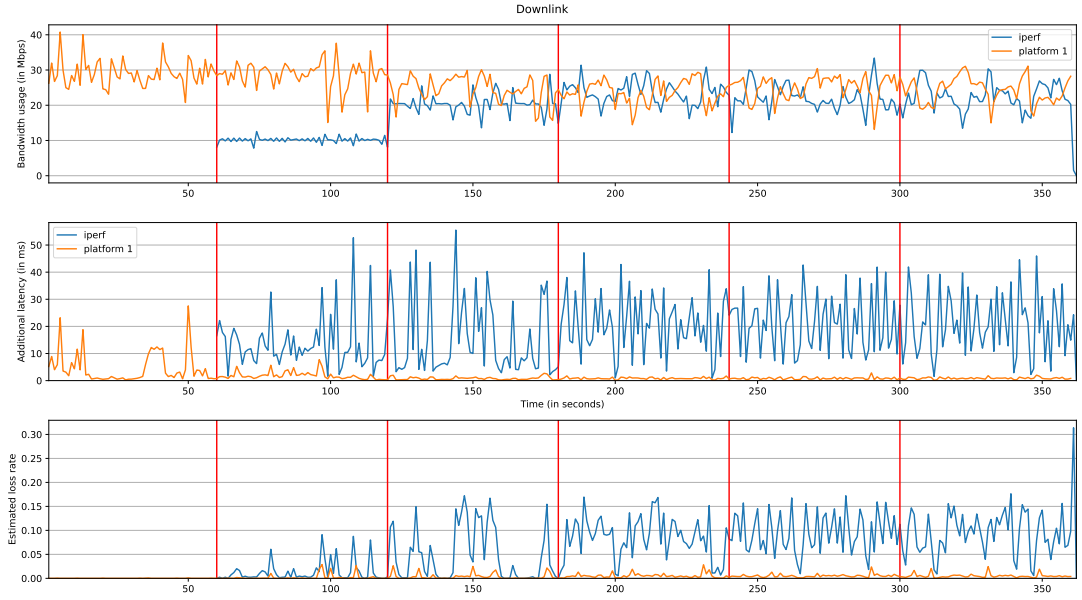


FIGURE 7.12 – DualPI2 : SCReAM face à du trafic *BBR*

en déduit qu'un autre type de signal de congestion en est à l'origine. Il s'agit vraisemblablement du marquage `ECN-CE` de ses paquets.

Concentrons-nous à présent sur les délais et les pertes de TCP. Tous deux sont bien plus importants que lors de la précédente expérience. Un aperçu de la Figure 7.12 suffit à constater l'ampleur des pertes. Le pourcentage de paquets égarés atteint plusieurs fois le seuil des 15%. La fréquence de ces signaux de congestion est due aux délais d'attente. Un seuil *target* constitue d'ailleurs une limite *à ne pas dépasser*. Sa valeur est fixée par défaut à 15ms. Passé cette limite, la probabilité d'envoyer un signal de congestion est accrue. Nous avons mesuré un taux de pertes moyen de 6.8 % pour *BBR*, ce qui est considérable. Son indifférence vis-à-vis des pertes lui permet de bénéficier d'un meilleur débit que *Cubic*. Il s'expose par contre à des délais plus élevés (*3 fois plus en moyenne*).

7.6 Conclusion

Nous avons ici cherché à favoriser la QoS d'une classe de trafic *CG* pré-existante. Des mécanismes ont été mis en œuvre dans les réseaux pour réduire ses délais de mise en file d'attente. Une attention a de plus été accordée à la cohabitation entre trafic CG et flux concurrents.

Nous avons basé notre étude sur SCReAM, un CCA présenté en Section 2.2.5, que nous appliquons à un flux CG. Pour ce faire, nous avons mis au point notre propre plateforme expérimentale de CG, dont nous maîtrisons chaque aspect. Une fois notre plateforme présentée, nous évaluons le comportement de SCReAM sur réseau cellulaire. Nous considérons des conditions radio particulièrement difficiles. SCReAM a permis de limiter les délais et les pertes en dépit de la mauvaise qualité du réseau. Ce CCA a d'ailleurs été conçu pour opérer dans de telles conditions.

Nous nous intéressons ensuite à la cohabitation entre SCReAM et d'autres CCAs majeurs, que sont *Cubic* et *BBR*. Dans les deux cas SCReAM réduit son débit en deçà du flux concurrent et subit de forts délais, dégradant la QoE. Le trafic CG a été le plus impacté lors de sa confrontation à *Cubic*. On aboutit à deux débits totalement opposés ; 45 Mbps pour `iperf` contre 5 Mbps pour

		Cubic	BBR
Trafic CG	Latence moyenne	2.14 ms	1.05 ms
	Variance	2.74	0.85
	Pourcentage > 10ms	2.67%	0%
	Pourcentage > 20ms	0.33%	0%
	Pourcentage > 50ms	0%	0%
	Pourcentage > 100ms	0%	0%
	Débit moyen	28.8 Mbps	25.3 Mbps
	Premier quartile	27.1 Mbps	22.86 Mbps
	Deuxième quartile	29.04 Mbps	25.74 Mbps
	Troisième quartile	31.09 Mbps	28.18 Mbps
	Pourcentage de pertes	0.03%	0.43%
Trafic TCP	Latence moyenne	5.89 ms	18.41 ms
	Variance	3.41	11.87
	Pourcentage > 10ms	8.28%	70.3%
	Pourcentage > 20ms	0.33%	38.94%
	Pourcentage > 50ms	0%	0.66%
	Pourcentage > 100ms	0%	0%
	Débit moyen	11.94 Mbps	19.63 Mbps
	Premier quartile	9.57 Mbps	16.66 Mbps
	Deuxième quartile	11.26 Mbps	20.59 Mbps
	Troisième quartile	14.32 Mbps	23.46 Mbps
	Pourcentage de pertes	0.24%	6.72%
Lien	Pourcentage d'utilisation	81.49% (57.61% CG + 23.88% TCP)	89.86% (50.6% CG + 39.26% TCP)

TABLE 7.4 – Étude des délais : Scream face à du trafic concurrent sur L4S

le CG. Ceci est dû au fait que *Cubic* est "*loss based*" ; il ne réagit qu'aux pertes de paquets. Il remplit donc complètement la file d'attente, ce qui engendre des délais supplémentaires.

Comme montré dans d'autres études et confirmé ici avec SCReAM, le partage des ressources entre CCAs peut s'avérer inéquitable en terme de débit. Les applications à faible latence peuvent de plus se retrouver pénalisées. Nous décidons donc d'exploiter les classes de trafic grâce à la discipline de file d'attente HTB. Chaque classe est ainsi associée à une file d'attente spécifique, ce qui permet de cloisonner leurs temps d'attente. La classe associée au trafic CG bénéficie en plus d'un traitement privilégié et d'une petite taille de file. D'après nos expériences, quel que soit le CCA régissant le flux TCP, le trafic CG est préservé. Son débit et ses délais restent les mêmes, indépendamment de la charge réseau. Ce n'est pas le cas du trafic `iperf` qui subit une latence assez importante durant la congestion, mais sans qu'elle soit problématique. La répartition de la bande passante pourrait cependant être plus équitable, le trafic CG prenant 3/5 et `iperf` 2/5. Le recours à des classes de trafic a donc clairement préservé la QoS du CG, ce qui était notre objectif initial. Cette approche comporte néanmoins quelques inconvénients, comme la conformité à la *neutralité d'internet*. La règle susvisée impose un traitement parfaitement *équitable* entres services d'une même classe. Cela fait peser une forte contrainte sur le classificateur.

L'AQM DualPI2 résout ce problème tout en permettant un déploiement plus simple nécessitant moins de configuration et moins de ressources de calcul. Deux classes de trafic sont considérées : le trafic *classique* et le trafic *scalable*, qui bénéficie de la file L4S garantissant de faibles délais et un faible taux de pertes. La distinction se fait selon la positionnement des bits ECN dans l'en-tête IP. À l'instar d'HTB, DualPI2 permet de préserver la QoS du trafic CG en cas de congestion et de concurrence avec d'autres flux. À regarder de plus près, la cohabitation n'est cependant pas parfaite. *Cubic* parvenait à une meilleure utilisation de la bande passante avec HTB (*35,41% contre 23,88% avec DualPI2*). Le taux de pertes subit par *BBR* est de plus considérable. Il n'est cependant pas possible de conclure définitivement quant à l'approche offrant les meilleures performances globales. Il faudrait réaliser davantage de tests axés sur la paramétrisation fine de chaque technologie (SCReAM, HTB, L4S).

En conclusion, nous avons ici deux solutions qui répondent à notre problématique. Chacune est très satisfaisante concernant la QoS des flux Cloud Gaming, mais elles s'accompagnent de contraintes différentes. La classification du trafic CG est à priori plus lourde à mettre en place

pour les FAIs. À l'inverse, la compatibilité avec la file L4S de DualPI2 demande aux fournisseurs de plateformes de CG de revoir leur CCA. Au delà des aspects techniques, la solution adoptée dépendra de la volonté de chacun des acteurs de faire cet effort, mais l'histoire montre que l'innovation dans les réseaux se fait traditionnellement à leur périphérie. D'ailleurs, au moment de boucler ce manuscrit, Nvidia [37] et Apple [38] ont conformé leurs CCAs aux *exigences de Prague*.

37. https://l4steam.github.io/PragueReqs/GeforceNow_L4S_requirements_Compliance_and_Objections.pdf
38. https://l4steam.github.io/PragueReqs/Apple_L4S_requirements_Compliance_and_Objections.pdf

Conclusion Générale

Rappel de la problématique

Le Cloud Gaming est un service en plein essor proposé aujourd'hui par de grands acteurs de l'industrie et qui a le potentiel de devenir l'un des principaux trafic d'Internet dans les années à venir. On lui confère de nombreux avantages, pouvant profiter aux joueurs comme aux éditeurs de jeu. Grâce à ce paradigme, il est par exemple possible de jouer depuis n'importe quel appareil ou d'accéder en quelques secondes à de nouveaux jeux. Le seul prérequis est de bénéficier d'une *bonne* connexion Internet. Il convient de souligner que le trafic CG est atypique et particulièrement exigeant pour les réseaux car il requiert à la fois une forte bande passante descendante et une très faible latence, tout en limitant également les pertes et la gigue. La moindre altération des conditions réseau peut ainsi négativement impacter l'expérience de jeu (QoE).

Nous avons vu que les nouvelles technologies déployées pour les réseaux d'accès, fixes comme mobiles, peuvent supporter de très hauts débits, mais la maîtrise de latence reste encore un défi aujourd'hui. Le trafic CG doit donc faire face à plusieurs difficultés. Il y a d'une part le phénomène du *bufferbloat* [GN11]. Il consiste en un accroissement des délais, résultant du remplissage de *buffers* surdimensionnés dans le réseau, le plus souvent en périphérie, afin de maximiser l'utilisation du lien. Se rajoute le problème de la coexistence avec d'autres flux, notamment ceux régis par des algorithmes de contrôle de congestion (CCA) *basés pertes*. Ceux-ci ne baissent leur débit que lorsqu'ils subissent des pertes de paquets. Ils ont donc tendance à remplir les files d'attente, nuisant au passage à la QoS des applications *temps réel*.

Étant donné ce contexte, le but de cette thèse était ainsi de favoriser la QoS du trafic CG en tirant partie des nouveaux leviers permis par la programmabilité des réseaux.

Travail Réalisé

Nous avons commencé par nous intéresser aux technologies et aux défis du Cloud Gaming. Une attention a été portée sur les plateformes existantes et les différents protocoles qu'elles utilisent. On a vu qu'il était possible d'estimer la qualité d'expérience à partir de mesures concrètes sur les flux, ce qui permet d'établir un lien clair entre QoE et QoS réseau. La latence du serveur vers le client est particulièrement impactante, et dépend de la congestion sous-jacente. Le flux multimédia transporté contribuant à l'engorgement du lien, les plateformes cherchent à adapter leur fonctionnement aux conditions réseau avec plus ou moins de réussite. Il manque surtout une évaluation exhaustive des plateformes modernes face à différentes perturbations réseau.

Nous avons ensuite étudié les mécanismes permettant de maîtriser la latence dans le réseau. D'abord en décrivant cinq algorithmes de contrôle de congestion orientés délais. Nous avons retenu SCReAM pour sa capacité à réguler des flux multimédia sur RTP, comme le trafic CG. Il est de plus compatible avec la fille basse latence (L4S) de l'AQM DualPI2. Nous avons vu

en outre que la taille des buffers influent sur la cohabitation entre les différents flux, de grands buffers profitant aux CCA basés pertes. Les deux problèmes (*bufferbloat* et cohabitation) sont donc liés.

Nous avons ensuite décrit cinq mécanismes de gestion de file d'attente. Deux technologies ont retenu notre attention. L'AQM DualPi2 permettant aux équipements réseau de signifier proactivement un début de congestion, permettant ainsi d'éviter les délais dans la file basse latence. L'AQM émet alors un signal, pouvant être de plusieurs natures. Cela peut être une perte volontaire de paquet (*drop*), ou un marquage de paquet `ECN-CE`. Nous avons également retenu un mécanisme permettant de gérer des classes de trafic, HTB, qui permettrait de prioriser le trafic CG et de lui adjoindre une file de taille appropriée.

Nous avons cherché à caractériser le trafic Cloud Gaming en environnement réseau contraint afin de bien appréhender ses spécificités et limitations, notamment en termes d'adaptabilité du trafic aux conditions réseau. Nous avons ainsi considéré 4 plateformes commerciales de CG. Plusieurs scénarios ont été étudiés, impactant tour à tout bande passante, délais, taux de perte et gigue. Nous avons appliqué différents niveaux d'intensité, que ce soit préalablement à la session de jeu ou au cours de celle-ci. On remarque que les réactions des 4 plateformes sont assez distinctes. Certaines réagissent démesurément aux perturbations et ont du mal à stabiliser leur débit après l'application d'une contrainte. Le phénomène se poursuit parfois même après leur abolition, ce qui impacte négativement la qualité vidéo. Une des plateformes reste quant à elle impassible à l'augmentation du délais, pourtant critique pour la QoE, et ne semble réagir qu'aux pertes de paquets. Il convient dès lors de trouver le bon équilibre entre surréaction et passivité, les deux pouvant tout autant nuire à la QoE. Cela trahit les difficultés d'adaptation des approches uniquement de bout en bout (*end-to-end*). Une aide du réseau pourrait ainsi être bénéfique.

Grâce à notre collaboration avec Orange, nous avons pu bénéficier de traces permettant d'émuler leur réseau 4G. Nous avons donc étendu notre étude aux réseaux mobiles, un environnement réseau réaliste mais toujours contraint. Ce type de réseau se distingue de par la forte instabilité de la bande passante disponible. Les plateformes doivent donc réagir très rapidement pour tantôt limiter la congestion et tantôt exploiter la bande passante. Au même titre que précédemment, nous avons pu dégager quelques tendances. On retrouve d'une part les plateformes qui surréagissent pour garantir de faibles délais. D'autres s'adaptent rapidement, mais présentent des changements intermittents de résolution et de qualité. Un autre plateforme semble insensible aux perturbations apportées, ne réagissant qu'aux pertes de paquets. Somme toute, la QoE s'est souvent retrouvée dégradée.

Afin d'améliorer la QoS du trafic CG, nous avons tout d'abord cherché à reconnaître ce type de trafic pour lui appliquer dans un second temps un traitement particulier. Nous avons donc décidé de mettre au point des modèles de *Machine Learning* (ML) considérant certaines caractéristiques statistiques des flux. Leur finalité est de distinguer le CG du reste du trafic. Les jeux de données destinés à entraîner (*ou tester*) ces modèles représentent deux classes. On distingue dès lors la classe CG de la classe non-CG (NCG), qui est constituée d'applications UDP à fort débit descendant. Nous avons commencé par réaliser de l'apprentissage *supervisé*. Les deux modèles considérés sont l'arbre de décision (DT) et le *Random Forest* (RF). Tous deux nous ont permis d'aboutir à de très bon résultats de classification y compris en considérant du trafic CG perturbé (F1-Score de 98%). Leur simplicité leur confère en plus un avantage sur d'autres techniques. Un inconvénient notoire doit néanmoins être souligné ; leur capacité de généralisation limitée qui implique d'avoir un dataset exhaustif, au moins en ce qui concerne les plateformes. Pour y remédier, nous avons considéré un troisième modèle non supervisé : USAD. Il répond à

la problématique susvisée au prix d'une plus grande complexité calculatoire.

Une fois nos modèles entraînés, nous avons proposé de les déployer au sein d'architectures adaptées aux contraintes opérationnelles. La première approche se base sur 4 fonctions réseau virtuelles (VNFs), les deux principales se chargeant d'extraire les caractéristiques des flux et de leur classification. La VNF de classification peut d'ailleurs être répliquée en fonction de la charge. Notons que nous avons soutenu un débit de 10 Gbps sur un modeste serveur, ce qui est compatible avec un déploiement en périphérie du réseau. La deuxième approche tire partit de P4, un langage de programmation réseau. Il permet d'exécuter des programmes directement dans le *dataplane*. Le temps d'exécution déterministe d'un programme compilé constitue ici un avantage majeur. Les contraintes inhérentes au langage ont néanmoins compliqué la phase de développement d'une classification entièrement dans le *dataplane*. Celles-ci furent surmontées, mais des contraintes supplémentaires venant du matériel ciblé nous ont poussés à adopter une approche *hybride*. Seul le calcul des caractéristiques est alors réalisé sur le *plan de données*. Nous les avons néanmoins simplifiées, eu égard des contraintes de P4. Les opérations les plus complexes sont quant à elles exécutées dans le *plan de contrôle*. Cette dernière approche nous permet de déployer le code P4 sur un switch Edgecore Wedge 100BF-32X (Intel Tofino, 32*100 $Gbit.s^{-1}$) et permet d'envisager un déploiement en cœur de réseau.

La dernière étape consiste à optimiser le transport des flux CG. N'ayant aucun contrôle sur les mécanismes de contrôle de congestion des plateformes commerciales, nous avons décidé d'implanter notre propre plateforme expérimentale de CG, et l'avons munie du CCA SCReAM. Nous avons montré que SCReAM joue bien son rôle en faisant transiter le trafic de la plateforme dans un environnement de réseau cellulaire difficile. Nous avons ensuite étudié les interactions entre SCReAM et d'autres CCAs. Il en ressort que les CCAs *basés pertes* tendent à monopoliser la bande passante. Dans tous les cas, le fait que les flux partagent une unique file d'attente induit des délais significatifs pour tous en cas de congestion. Cette observation nous incite à différencier les files d'attente sur le *bottleneck*, une des deux files étant alors dédiée à la classe de trafic CG, définie par nos résultats de classification. Nous lui garantissons un débit minimal grâce à `HTB`, une discipline de file d'attente, et limitons sa taille de file. Cette solution d'ingénierie de trafic s'avère être très concluante et capable de préserver la QoS du trafic CG. La législation européenne sur la *neutralité des réseaux* fait néanmoins peser une forte contrainte sur le classificateur qui ne doit léser aucune des plateformes. Nous avons étudié une seconde solution, reposant sur l'AQM DualPi2 composé de deux files d'attente. Le trafic est acheminé dans l'une ou l'autre de ses files selon le placement des bits ECN (`ECT(0)` ou `ECT(1)`), inscrits au niveau de l'en-tête IP, plutôt que sur une classe de trafic prédéfinie. Les CCAs dits *scalables* peuvent transiter via la file L4S, garantissant de faibles délais. C'est le cas de SCReAM qui répond aux *exigences de Prague* et permet de faire transiter le trafic CG dans la file L4S. Le trafic concurrent, régit par des CCAs *classiques*, transite quant à lui dans la seconde file. Notre évaluation a montré que cet AQM permet de préserver la QoS du trafic CG. Il est cependant difficile de décréter quelle approche serait la plus appropriée, car chacune s'accompagne d'avantages et d'inconvénients. Toutes demandent un effort d'intégration supplémentaire à des acteurs différents (opérateurs ou fournisseurs de services). La conformité récente d'Apple et de Nvidia aux *exigences de Prague* témoigne a minima d'un intérêt croissant pour la seconde approche.

Perspectives de recherche

Approfondir certains résultats par de nouvelles expériences

Les mécanismes que nous avons mis en place dans le dernier chapitre, à savoir SCReAM, DualPI2, et, dans une moindre mesure, HTB, contiennent une multitude de paramètres. Nous les avons laissés pour la plupart à leur valeur par défaut, suivant les recommandations de leurs concepteurs. Ceci nous a permis de garder une charge d'expérimentation compatible avec les échéances de la thèse. Notons que SCReAM a initialement été conçu pour transporter du trafic de viso-conférence. Il est donc possible que sa configuration ne soit pas optimisée au trafic CG. À titre d'exemple, les paquets RTP peuvent être retenus jusqu'à 100ms en file d'attente. Ces délais sont acceptables pour un flux de visio-conférence, mais trop longs pour notre cas d'étude. De même, DualPI2 et HTB sont des solutions généralistes qui peuvent être configurées selon les exigences de QoS. Une étude plus approfondie des principaux paramètres permettrait d'aboutir à une configuration optimisée pour le Cloud Gaming, améliorant d'autant plus sa QoS.

Lors de nos expérimentations, nous avons uniquement évalué SCReAM face à des flux concurrents TCP `iperf`. Il serait intéressant de l'opposer à des flux plus *représentatifs* des usages du réseau. Nous songeons à deux types de trafic en particulier : le *streaming* vidéo et la navigation web. Ceci demande de faire évoluer le *testbed* pour pouvoir y faire transiter du trafic venant d'Internet. Le trafic serait ainsi plus représentatif mais moins maîtrisé, ce qui pourrait limiter la reproductibilité des expériences.

Nous avons jusqu'à présent laissé le trafic SCReAM sans concurrence dans la file L4S. Il serait intéressant d'étudier la cohabitation entre SCReAM et un autre trafic *scalable* concurrent. On pourrait par exemple générer un flux *TCP Prague*, répondant par défaut aux exigences de L4S. Par analogie, nous pourrions également créer de la concurrence au sein de la file HTB dévolue au CG. On pourrait y faire transiter en plus du trafic issu des plateformes commerciales de CG.

L'ensemble des métriques exposées dans le Chapitre 7 sont axées QoS *réseau*. Nous songeons à enrichir notre étude par une analyse de la QoE du service CG. Pour ce faire, nous envisageons de comparer les *frames brutes* du serveur à celles du client pour mesurer l'impact des pertes et de la réduction du débit sur la qualité vidéo. Un *framework* [KRW03] pourrait d'ailleurs nous simplifier la tâche. Notons que notre plateforme n'emploie aucun mécanisme de redondance à ce stade. Nous pourrions par exemple évaluer le bénéfice de l'ajout de FEC sur la QoE. Pour être exhaustifs, il conviendrait également de mesurer la latence de bout en bout au niveau du service.

Opter pour un cas d'utilisation plus exigeant : le Cloud-VR

Dans un autre registre, nous pourrions faire évoluer notre cas d'usage vers celui de la réalité virtuelle dans le Cloud (Cloud-VR). Ceci augmenterait d'autant plus les exigences sur les réseaux. Les casques de réalité virtuelle se démocratisent progressivement. Le casque Meta Quest 2 s'est d'ailleurs vendu à 15 millions d'exemplaires. L'une des particularités de ce casque est de pouvoir exécuter des programmes directement sur son processeur. La VR est donc rendue accessible à des utilisateurs ne disposant pas du matériel nécessaire à rendre des scènes 3D complexes. Cette approche a cependant plusieurs limites. D'une part les modestes performances du processeur embarqué limitent la qualité graphique des modèles, ce qui nuit à l'immersion. D'autre part, ce même processeur est une source de chaleur inconfortable quand il est fortement sollicité, et réduit l'autonomie de la batterie du casque. Le Meta Quest 2 est par ailleurs équipé d'une liaison WiFi haut débit étant d'ores et déjà capable de streamer un jeu exécuté depuis un ordinateur du même réseau local. L'utilisation d'un service Cloud-VR, similaire au Cloud-Gaming dans son fonctionnement, pourrait permettre à des utilisateurs d'accéder à un rendu 3D de qualité, sans

devoir investir dans un ordinateur coûteux, tout en limitant la chauffe du casque et en améliorant son autonomie.

Du point de vue du réseau, ce cas d'usage impose de nouvelles contraintes. Le débit demandé est plus du double que celui du Cloud Gaming (*environ 80Mbps*), du fait d'une résolution supérieure (4128x2096) et d'un taux de rafraîchissement plus élevé (90Hz à 120Hz). Le budget de latence est encore plus réduit, et engendre un inconfort plus intense quand il est dépassé (nausées). John Carmack [39] recommande de ne pas dépasser 50ms de latence au total *"Motion to photon"*, ce qui est déjà quasiment la latence d'Air-Link sans intermédiaire réseau (45ms). Enfin, le protocole de transport utilisé actuellement est TCP, qui n'est pas adapté aux communications en temps réel sur Internet. Il serait donc intéressant d'évaluer dans quelle mesure les solutions que nous proposons dans cette thèse pourraient améliorer la QoS d'un service de Cloud-VR et comment celles-ci pourraient être encore améliorées vis-à-vis des nouvelles exigences.

Étudier une approche plus radicale avec BPP et Packet Wash

Big Packet Protocol (BPP) [LCC+18] est un nouveau format de paquet visant à améliorer la précision des communications en garantissant notamment les délais. L'une des fonctionnalités est *Packet Wash* [LMY+19], dont le principe consiste à *dropper* partiellement des paquets. Leur *payload* est alors découpée à des *offsets* spécifiques. Le procédé permet de soulager le réseau avec plus de granularité. En cas de congestion, les flux commencent par perdre les données les moins essentielles. Pour ce faire, la *payload* des paquets est divisée en plusieurs composants de différents niveaux d'importance, et éventuellement liés entre eux. Chaque paquet est de plus associé à un *budget de latence*, défini par l'application qui l'a émis. Ce budget de latence est décrémenté proportionnellement aux délais subits au niveau des équipements intermédiaires. C'est pertinent dans le cas d'une application *temps réel*. Une donnée ayant pris trop de retard peut alors tomber en désuétude. Le réseau peu dès lors détecter de tels paquets, les réduire ou les éliminer.

Dans [CS23], Clayman & al. présentent une technique permettant d'envoyer une vidéo `H264 SVC` sur BPP. L'acronyme `SVC` signifie *Scalable Video Coding*. De tels encodeurs envoient une vidéo sous forme de couches (similaire à du JPEG progressif). Chacune de couches apporte un gain de qualité par rapport à la couche initiale. Les auteurs ont donc fait le parallèle avec les *composants* de *Packet Wash*. Le réseau peut ainsi éliminer certaines couches, dégradant modérément la qualité vidéo pour préserver la latence. Cela se fait avec une réactivité inédite, n'impliquant aucun échange d'information entre le client et le serveur. Cette approche est donc parfaitement appropriée aux applications haut débit à contrainte faible latence. Les auteurs citent d'ailleurs pour exemple le CG, la réalité augmentée (AR) et la réalité virtuelle (VR).

Il serait donc intéressant d'appliquer *Packet Wash* au trafic CG qui semble être un cas d'usage idéal, puis d'évaluer le gain en QoE de cette approche plus radicale par rapport aux solutions que nous avons proposées.

39. `https://danluu.com/latency-mitigation/`

Productions

Publications

Les contributions présentées dans cette thèse ont pour la plupart d'entre elles été validées par la communauté scientifique à travers 4 publications scientifiques internationales sélectives (à comité de lecture), plus précisément 3 conférences et un journal :

1. Philippe Graff, Xavier Marchal, Thibault Cholez, Stéphane Tuffin, Bertrand Mathieu, Olivier Festor. An Analysis of Cloud Gaming Platforms Behavior under Different Network Constraints. HiPNet 2021 - 3rd International Workshop on High-Precision, Predictable, and Low-Latency Networking, IFIP-IEEE, Oct 2021, Izmir (Virtual), Turkey. pp.7, [GMC+21], correspond au chapitre 3.
   Lien : `https://inria.hal.science/hal-03421031/`.
2. Xavier Marchal, Philippe Graff, Joël Roman Ky, Thibault Cholez, Stéphane Tuffin, Bertrand Mathieu, Olivier Festor. An Analysis of Cloud Gaming Platforms Behaviour Under Synthetic Network Constraints and Real Cellular Networks Conditions. Journal of Network and Systems Management, 2023, Special Issue on High-Precision, Predictable and Low-Latency Networking, 31 (2), pp.39, [MGK+23], correspond aux chapitres 3 et 4.
   Lien : `https://inria.hal.science/hal-04050288/`.
3. Philippe Graff, Xavier Marchal, Thibault Cholez, Bertrand Mathieu, Olivier Festor. Efficient Identification of Cloud Gaming Traffic at the Edge. NOMS 2023 - 36th IEEE/IFIP Network Operations and Management Symposium, May 2023, Miami, United States. pp.10, [GMC+23], correspond au chapitre 5.
   Lien : `https://inria.hal.science/hal-04056607/`.
4. Joël Roman Ky, Philippe Graff, Bertrand Mathieu, Thibault Cholez. A Hybrid P4/NFV Architecture for Cloud Gaming Traffic Detection with Unsupervised ML. 28th IEEE Symposium on Computers and Communications (ISCC 2023), IEEE, Jul 2023, Gammarth, Tunisia. pp.733-738, [KGMC23], correspond au chapitre 6.
   Lien : `https://hal.science/hal-04130096/`.

Développement logiciel

— **Architecture à base de micro-services** pour la classification de trafic Cloud-Gaming composée (1) d'une sonde [40] extrayant et calculant les caractéristiques des flux par fenêtre temporelle, (2) d'un load-balancer, (3) de nœuds de calcul exécutant un arbre de décision pré-entraîné et (4) d'un agrégateur donnant le résultat de classification pour un couple d'adresse IP.
Lien : `https://github.com/mosaico-anr/CG_Classifier`.

40. développée par Xavier Marchal

— **Programme P4** pour la classification de trafic Cloud-Gaming : implantation en langage P4 compatible avec un *switch logiciel bmv2* de l'extraction et du calcul des caractéristiques de flux, puis de leur classification par un arbre de décision pré-entraîné.
Lien :`https://github.com/mosaico-anr/P4-classifier`.

Jeux de données

— **Captures réseau de trafic CG**, `pcap` représentant 4 plateformes commerciales (GFN, PSN, STD, XC), faisant varier résolutions, framerates, jeux, types de contrôles (35Go).
— **Captures réseau de trafic UDP à haut débit**, `pcap` incluant des services de visio-conférence, streaming vidéo, streaming vidéo live, navigation QUIC, et bureau à distance (17Go).
— **Captures réseau de trafic CG sur réseau à capacité fixe sous différentes contraintes**, `pcap` du trafic de 4 plateformes CG soumises à des contrainte de débit, latence, taux de pertes et gigue survenant avant la session de jeu (60Go) ou au cours de celle-ci (25Go).
— **Captures réseau de trafic CG sur réseau à capacité variable**, `pcap` du trafic de 4 plateformes CG soumises aux contraintes de 6 environnements de réseaux cellulaires 4G d'Orange (71Go).

Lien vers les différents datasets : `https://cloud-gaming-traces.lhs.loria.fr`

Table des figures

1 Illustration d'un service de Cloud Gaming . . . 3

1.1 Architecture de GamingAnywhere côté serveur [HHCC13] . . . 10

2.1 Architecture de GCC [HLC+16] . . . 23
2.2 Machine à états de GCC [CDCHM16] . . . 24
2.3 Visualisation du RTVL et des phases de l'algorithme C3G [AMC+19] . . . 25
2.4 Calcul de la taille de fenêtre selon la prédiction [WSB04] . . . 26
2.5 Architecture Scream [Joh14] . . . 27
2.6 Aperçu des deux classes utilisées par Janczukowicz [Jan17] . . . 34
2.7 Architecture de L4S [ADSB+19] . . . 35
2.8 Comparaison entre le débit de DCTCP et Cubic [OGBT20b] . . . 36
2.9 Comparaison entre le débit de Prague et Cubic [OGBT20a] . . . 37

3.1 Vue du testbed . . . 43
3.2 Débit de GFN, PSN, STD et XC en fonction du taux de pertes (service⇒client) . . . 44
3.3 Bitrate de Stadia pour 5% de pertes . . . 45
3.4 Débit de GFN, PSN, STD et XC en fonction de la bande passante disponible (service⇒client) . . . 46
3.5 Débit de GFN, PSN, STD et XC en fonction des délais (service⇒client) . . . 47
3.6 Débit de GFN, PSN, STD et XC en fonction des valeurs de gigue (service⇒client) 48
3.7 Variation relative des IAT et de la taille du Payload UDP selon les contraintes . . . 49
3.8 Perte soudaine de paquets . . . 51
3.9 Limitation soudaine de la bande passante . . . 52
3.10 Limitation soudaine de la bande passante : PSN et STD . . . 53
3.11 Ajout soudain de latence . . . 54
3.12 Ajout soudain de gigue . . . 54
3.13 Ajout soudain de gigue : GFN et STD . . . 55

4.1 Représentation du phénomène de *Handover* [TLL+04] . . . 58
4.2 Demande de transmission depuis l'UE [RJLHF22] . . . 59
4.3 Fonctionnement de l'outil *Saturator* . . . 60
4.4 Placement de *Mahimahi* sur un routeur intermédiaire . . . 62
4.5 Analyse des plateformes sous un scénario de mobilité (*Highway*) . . . 65
4.6 Comportement des plateformes de CG sous différentes conditions . . . 66
4.7 Lien entre débit et qualité vidéo . . . 68
4.8 Distribution cumulée des paquets avec plus de 100ms de délais . . . 70

5.1 Comparaison des motifs temporels du trafic CG avec d'autres applications [CB22] 78
5.2 Transformation de flux en images 2D 80
5.3 DT : arborescence du modèle - *splitting* de Q_m 84
5.4 Fenêtre temporelle : impact de sa taille 86
5.5 Impact de Thr sur la classification 87
5.6 Impact de max_depth sur l'*Accuracy* 89
5.7 Architecture du modèle USAD [AMG+20] 92
5.8 Comparaison entre DT et $USAD$ 93

6.1 VNFs : vue de l'architecture globale 97
6.2 VNFs : analyse de performances 98
6.3 P4 : vue d'ensemble [BDG+13] 99
6.4 P4 : mise à jour depuis le *plan de données* [XZ19] 100
6.5 Correspondance entre seuils et indices pour la caractéristique 'i' 104
6.6 Performances de classification selon le calcul des caractéristiques (modèles USAD) 106
6.7 Impact de la variance sur les performances de classification (modèle USAD) . . . 106
6.8 Architecture proposée 107

7.1 Architecture du serveur de *streaming* 111
7.2 Architecture du client de *streaming* 112
7.3 Architecture du proxy "*serveur*" 113
7.4 SCReAM sur un réseau cellulaire : trace `txops` *Highway* 114
7.5 Testbed : SCReAM face à du trafic concurrent 116
7.6 SCReAM face à du trafic *Cubic* 117
7.7 Trafic CG sans CCA face à du trafic *Cubic* 118
7.8 SCReAM face à du trafic *BBR* 119
7.9 Classes de trafic et HTB : SCReAM face à du trafic *Cubic* 120
7.10 Classes de trafic et HTB : SCReAM face à du trafic *BBR* 121
7.11 DualPI2 : SCReAM face à du trafic *Cubic* 123
7.12 DualPI2 : SCReAM face à du trafic *BBR* 124

A.1 Architecture du proxy "*client*" 141
A.2 Aperçu des différents flux 141

C.1 SCReAM sur réseau cellulaire : trace `txops` A 145
C.2 SCReAM sur réseau cellulaire : trace `txops` B 146
C.3 SCReAM sur réseau cellulaire : trace `txops` C 146
C.4 SCReAM sur réseau cellulaire : trace `txops` D 147
C.5 SCReAM sur réseau cellulaire : trace `txops` E 147

Liste des tableaux

1 Délais inhérents aux différentes technologies d'accès [ARC23a] 2

1.1 Caractéristiques nominales des flux multimédia des plateformes de CG 12
1.2 Protocoles de transports utilisés 14

2.1 Récapitulatif des principales caractéristiques des CCAs étudiés 30

3.1 Paramètres appliqués pour dégrader les conditions réseau 43
3.2 Paramètres appliqués pour dégrader temporairement les conditions réseau 50

4.1 Caractéristiques des traces `txops` sur le réseau mobile d'Orange 61
4.2 Statistiques nominales des 4 plateformes sur réseau FttH 63
4.3 Moyenne et variance du débit de chaque plateforme selon les traces `txops` 63
4.4 Utilisation de la bande passante ($\%avlb$) / Proportion du débit de référence $\%ref$ 64
4.5 Somme des variances normalisées des débits de chaque plateforme 66
4.6 Pourcentage de paquets UDP dont les délais dépassent 10,20,50 et 100ms 69

5.1 Aperçu des hyper-paramètres 85
5.2 DT & RF : Hyper-Parameters w=33ms 88
5.3 Classification du trafic CG, Conditions *normales* 89
5.4 Classification du trafic CG *perturbé* 90
5.5 Classification du trafic CG *perturbé* ; matrice de confusion 90

7.1 Étude des délais : avec/sans CCA sur *Highway* 115
7.2 Étude des délais : SCReAM face à des flux concurrents 117
7.3 Étude des délais : Scream face à du trafic concurrent sur HTB, configuration 1 . 121
7.4 Étude des délais : Scream face à du trafic concurrent sur L4S 125

C.1 Étude des délais : avec/sans CCA sur *Highway* 145

Annexes

Annexe A

Schémas complémentaires de la plateforme expérimentale de CG

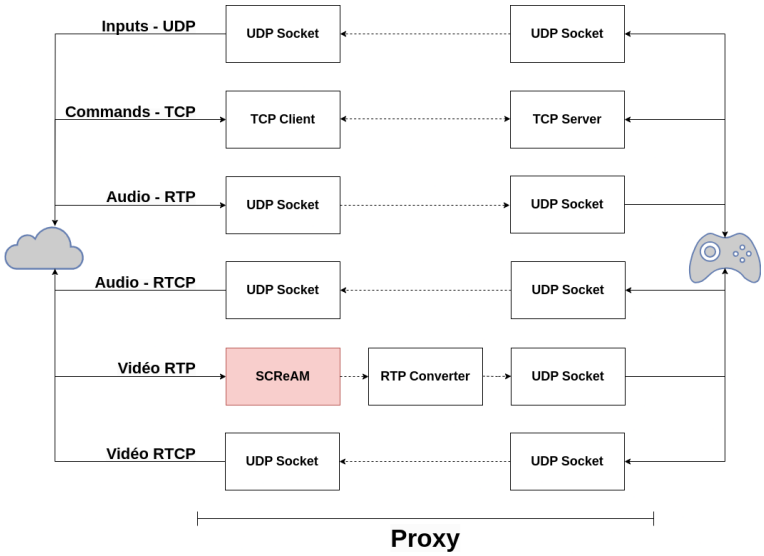


FIGURE A.1 – Architecture du proxy "*client*"

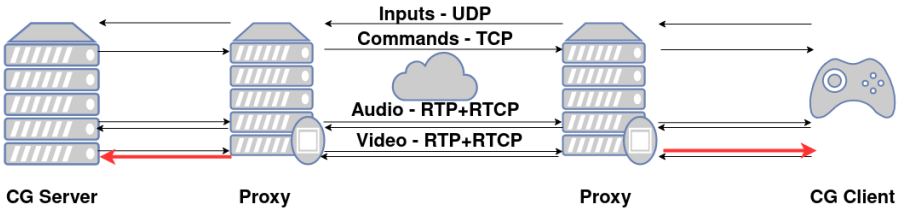
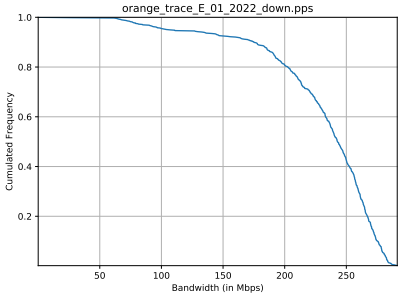
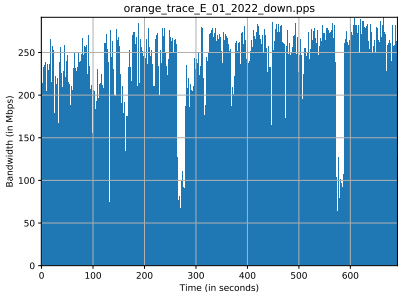
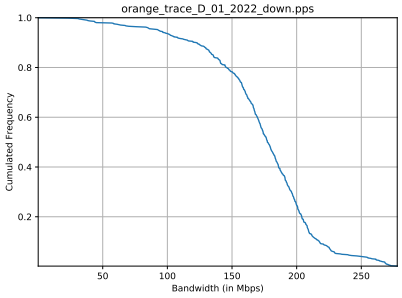
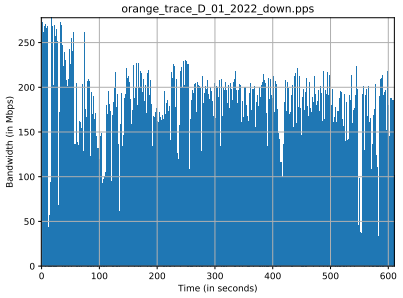
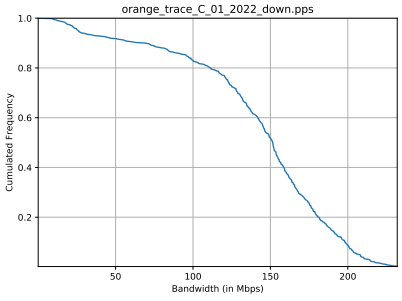
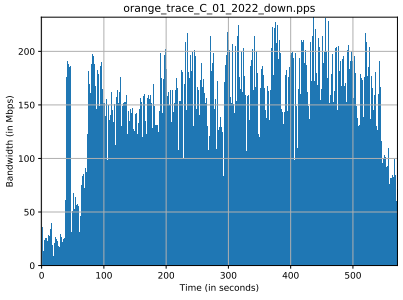
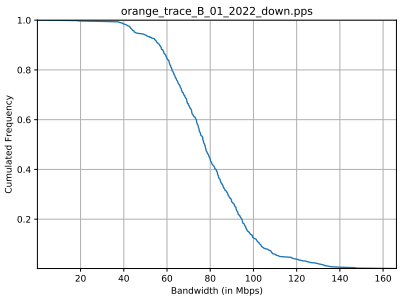
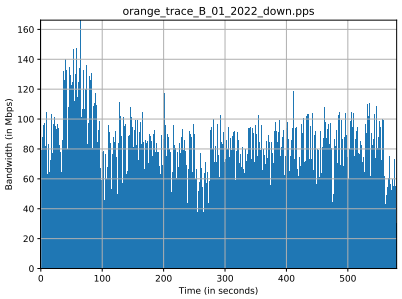
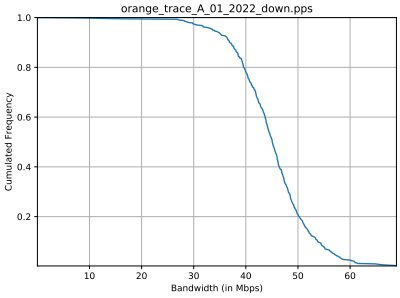
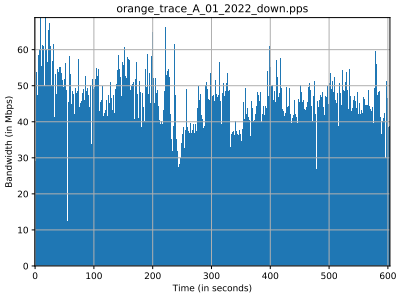
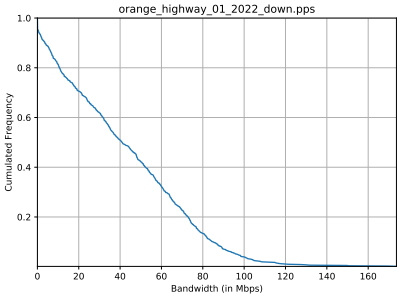
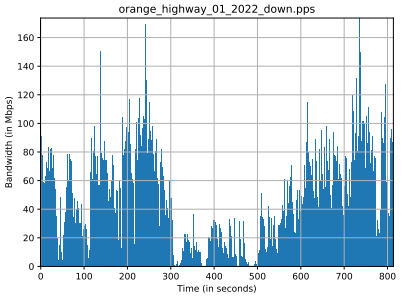


FIGURE A.2 – Aperçu des différents flux

Annexe B

Caractéristiques des captures de réseau cellulaire

File	Capacity distribution	Capacity Profile
Trace E		
Trace D		
Trace C		

File	Capacity distribution	Capacity Profile
Trace B		
Trace A		
Highway		

Annexe C

SCReAM sur réseau cellulaire, autres traces

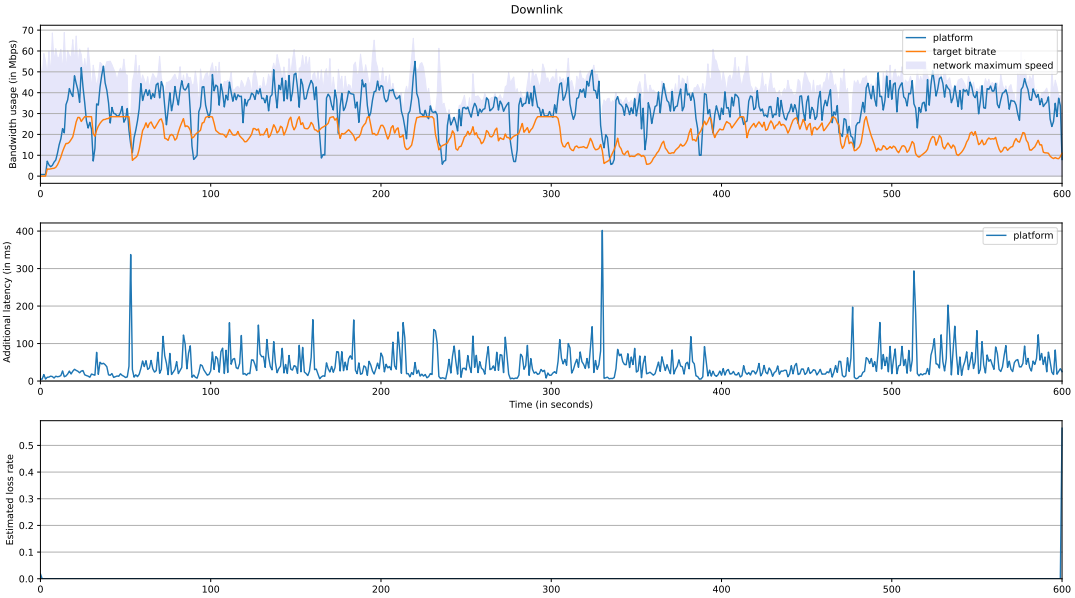


FIGURE C.1 – SCReAM sur réseau cellulaire : trace `txops` A

	`txops` A	`txops` B	`txops` C	`txops` D	`txops` E
Latence moyenne	42.41ms	25.83ms	7.17ms	9.66ms	4.89ms
Pourcentage >10ms	94.5%	68.69%	11.2%	11.8%	6.6%
Pourcentage >20ms	71.7%	51.73%	7.2%	9.6%	6.5%
Pourcentage >50ms	29%	11.59%	2.8%	5.3%	2%
Pourcentage >100ms	5.8%	1.4%	1.1%	3.2%	0.8%

TABLE C.1 – Étude des délais : avec/sans CCA sur *Highway*

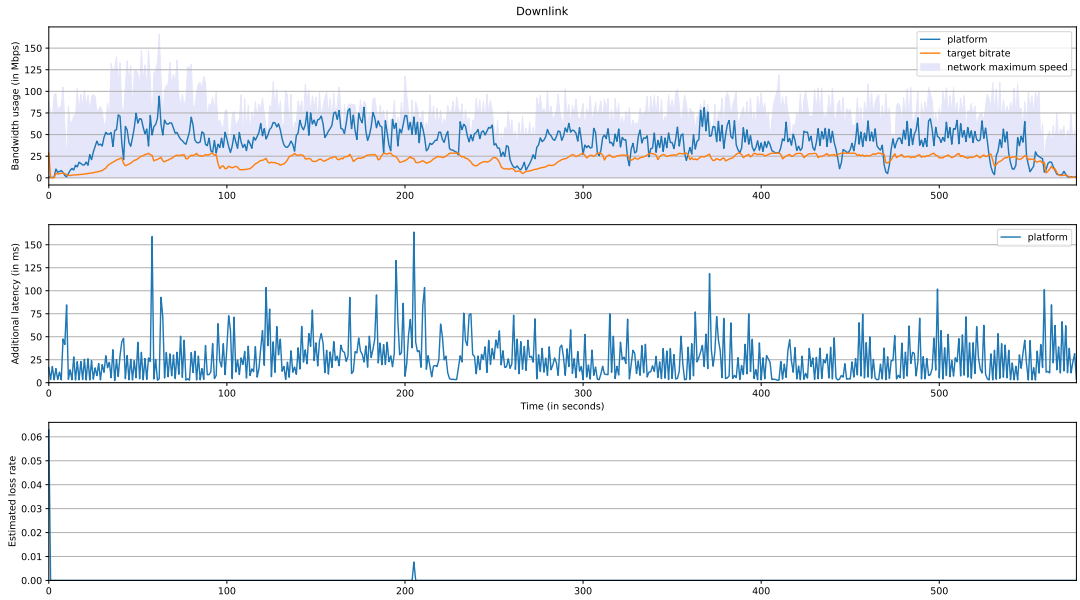


FIGURE C.2 – SCReAM sur réseau cellulaire : trace `txops` B

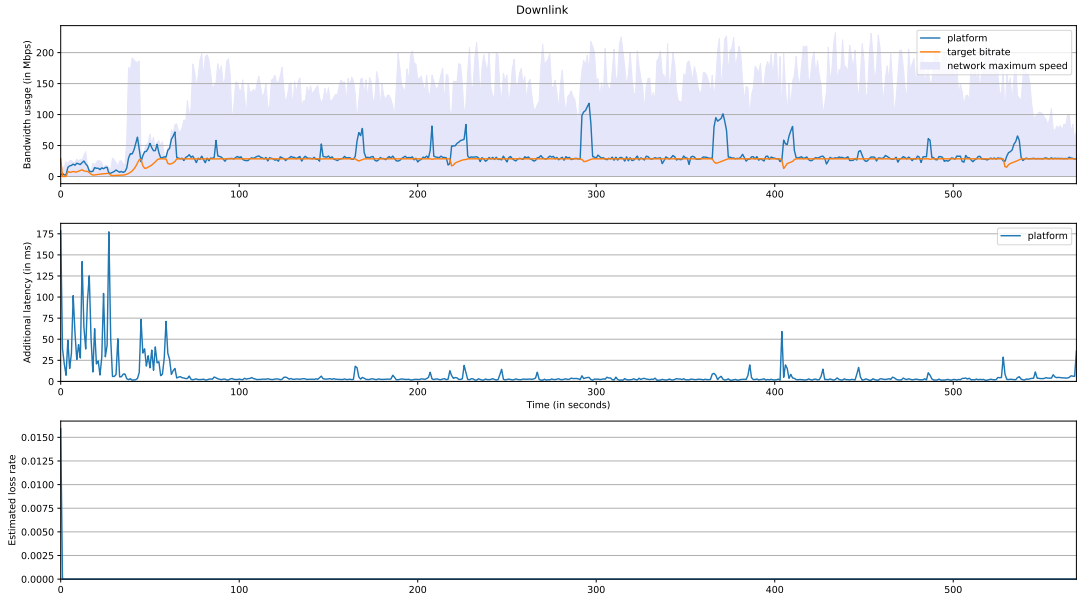


FIGURE C.3 – SCReAM sur réseau cellulaire : trace `txops` C

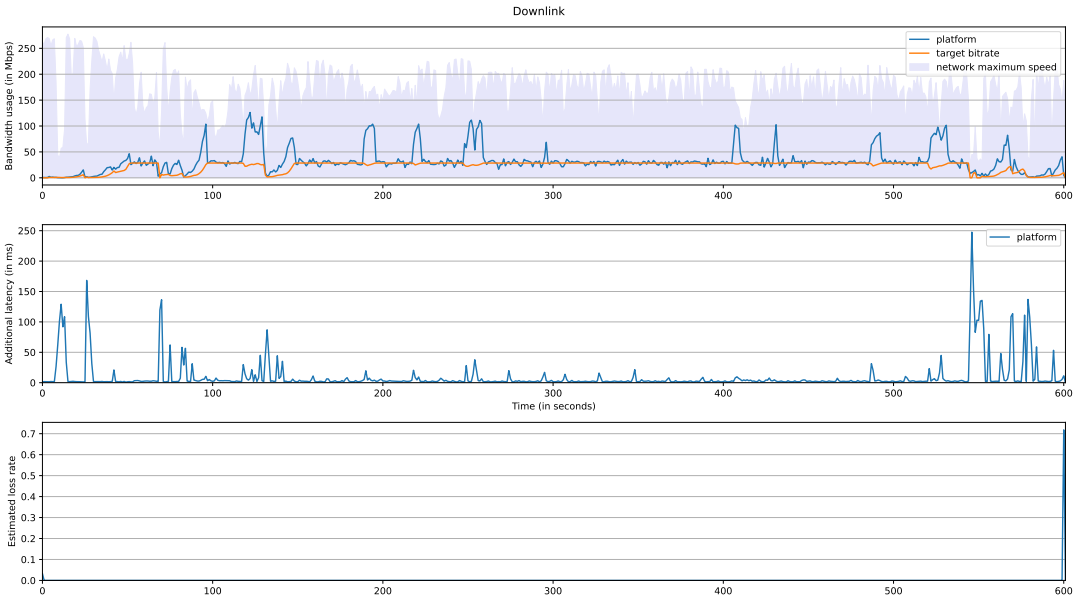


FIGURE C.4 – SCReAM sur réseau cellulaire : trace `txops` D

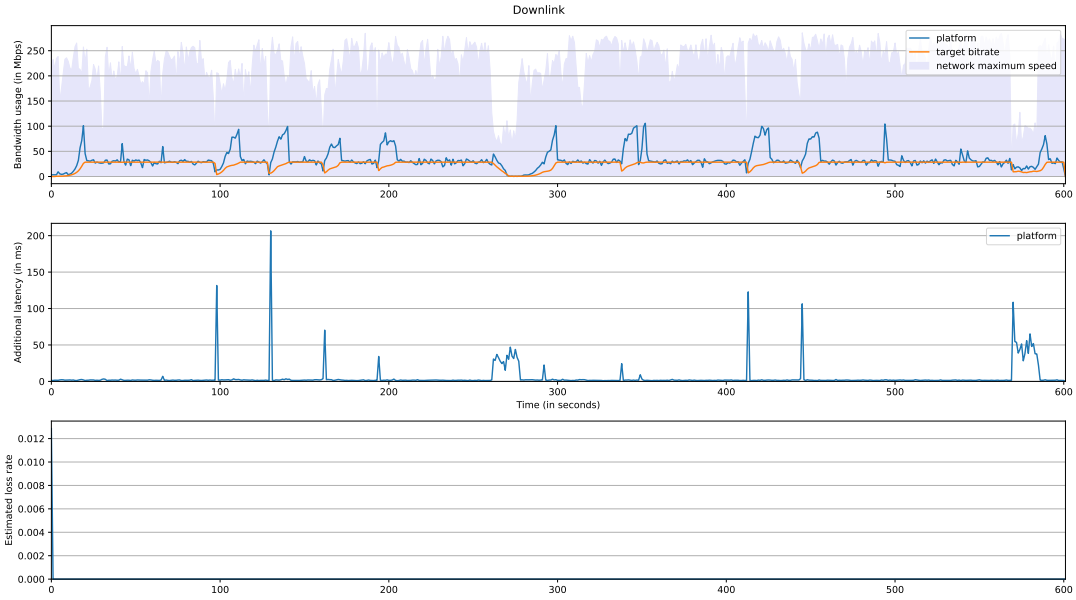


FIGURE C.5 – SCReAM sur réseau cellulaire : trace `txops` E

Glossaire

AQM Active Queue Management
API Application Programming Interface
BDP Bandwidth Delay Product
BS Base Station
BSR Buffer Status Report
CCA Congestion Control Algorithm
CG Cloud Gaming
CNN Convolutional Neural Network
DL Deep Learning
DT Decision Tree
ECN Explicit Congestion Notification
FAI Fournisseur d'Accès Internet
FEC Forward Error Correction
FIFO First In First Out
FPS Frames Per Second
FttH Fiber to the Home
GFN Nvidia GeForce Now
HTB Hierarchical Token Bucket
IAT Inter-Arrival-Time
KFV K-Fold Validation
L4S Low Loss Low Latency Scalable throughput
LSTM Long Short Term Memory
LTE Long Term Evolution
MAC Medium Access Control
ML Machine Learning
MTU Maximum Transmission Unit
NFV Network Function Virtualization
OWD One Way Delay
P4 Programming Protocol-Independent Packet Processors
PSN PlayStation Now
QoE Quality of Experience
QoS Quality of Service
RDP Remote Desktop Protocol
RLC Radio Link Control
RF Random Forest
RTCP RTP Control Protocol
RTP Real-time Transport Protocol
RTSP Real Time Streaming Protocol

RTT Round Trip Time
SDN Software Defined Networking
SDU Service Data Unit
SFC Service Function Chain
SSRC Synchronization Source
STD Stadia
TS Temporal Serie
UE User Equipment
UG Uplink Grant
VM Virtual Machine
VNF Virtual Network Function
VR Virtual Reality
XC Xbox Cloud Gaming

Bibliographie

[ADSB+19] Olga Albisser, Koen De Schepper, Bob Briscoe, Olivier Tilmans et Henrik Steen : DUALPI2 - Low Latency, Low Loss and Scalable (L4S) AQM. *In Proc. Netdev 0x13*, mars 2019.

[AGM+10] Mohammad Alizadeh, Albert G. Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta et Murari Sridharan : Data center TCP (DCTCP). *In* Shivkumar Kalyanaraman, Venkata N. Padmanabhan, K. K. Ramakrishnan, Rajeev Shorey et Geoffrey M. Voelker, éditeurs : *Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, New Delhi, India, August 30 -September 3, 2010*, pages 63–74. ACM, 2010.

[AKPC21] Seyoung Ahn, Jeehyeong Kim, Soo Young Park et Sunghyun Cho : Explaining deep learning-based traffic classification using a genetic algorithm. *IEEE Access*, 9:4738–4751, 2021.

[AMC+19] Alberto Alós, Francisco Morán, Pablo Carballeira, Daniel Berjón et Narciso García : Congestion Control for Cloud Gaming Over UDP Based on Round-Trip Video Latency. *IEEE Access*, 7:78882–78897, 2019.

[AMG+20] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti et Maria A. Zuluaga : Usad : Unsupervised anomaly detection on multivariate time series. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[ARC12] ARCEP : Rapport public d'activité de l'arcep. `https://www.arcep.fr/uploads/tx_gspublication/rapport-activite-2012.pdf`, 2012. Vu le 4 octobre 2023.

[ARC23a] ARCEP : L'etat d'internet en france. `https://www.arcep.fr/uploads/tx_gspublication/RA-2023-TOME3_etat-internet-france_juillet2023.pdf`, 2023. Vu le 4 octobre 2023.

[ARC23b] ARCEP : La régulation de l'arcep au service des territoires connectés. `https://www.arcep.fr/uploads/tx_gspublication/RA-2023_TOME2_territoires-connectes_juin2023.pdf`, 2023. Vu le 4 octobre 2023.

[ASV+21] Iman Akbari, Mohammad A. Salahuddin, Leni Ven, Noura Limam, Raouf Boutaba, Bertrand Mathieu, Stephanie Moteau et Stéphane Tuffin : A look behind the curtain : Traffic classification in an increasingly encrypted web. *Proc. ACM Meas. Anal. Comput. Syst.*, 5(1):04 :1–04 :26, 2021.

[Aut99] IANA Internet Assined Numbers Authority : Protocol registries. `https://www.iana.org/protocols`, 1999. Vu le 8 juin 2023.

[BBHH16] Anat Bremler-Barr, Yotam Harchol et David Hay : OpenBox : A software-defined framework for developing, deploying, and managing network functions. *In*

Proceedings of the 2016 ACM SIGCOMM Conference, SIGCOMM '16, pages 511–524. Association for Computing Machinery, 2016.

[BBP+16] Bob Briscoe, Anna Brunström, Andreas Petlund, David A. Hayes, David Ros, Ing-Jyh Tsang, Stein Gjessing, Gorry Fairhurst, Carsten Griwodz et Michael Welzl : Reducing internet latency : A survey of techniques and their merits. 18(3):2149–2196, 2016.

[BCB+02] Jean-Yves Le Boudec, William Courtney, Jon Bennett, Shahram Davari, Dimitrios Stiliadis, Kent Benson, Victor Firoiu, Dr. Bruce S. Davie et Anna Charny : An Expedited Forwarding PHB (Per-Hop Behavior). RFC 3246, mars 2002.

[BCS94] Robert T. Braden, Dr. David D. Clark et Scott Shenker : Integrated Services in the Internet Architecture : an Overview. RFC 1633, juin 1994.

[BDG+13] Patrick W. Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Daniel E. Talayco, Amin Vahdat, George Varghese et David Walker : P4 : programming protocol-independent packet processors. *Comput. Commun. Rev.*, 44:87–95, 2013.

[BDST+16] Olga Bondarenko, Koen De Schepper, Ing-Jyh Tsang, Bob Briscoe, Andreas Petlund et Carsten Griwodz : Ultra-low delay for all : Live experience, live analysis. *In Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, New York, NY, USA, 2016. Association for Computing Machinery.

[BFC+19] Pierre-Olivier Brissaud, Jérôme François, Isabelle Chrisment, Thibault Cholez et Olivier Bettan : Transparent and service-agnostic monitoring of encrypted web traffic. *IEEE Transactions on Network and Service Management*, 16(3):842–856, 2019.

[BJSOC21] Davide Brunello, Ingemar Johansson S, Mustafa Ozger et Cicek Cavdar : Low latency low loss scalable throughput in 5g networks. *In 2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, pages 1–7, 2021.

[BKS23] Bob Briscoe, Mirja Kühlewind et Richard Scheffenegger : More Accurate Explicit Congestion Notification (ECN) Feedback in TCP. Internet-Draft draft-ietf-tcpm-accurate-ecn-26, Internet Engineering Task Force, juillet 2023. Work in Progress.

[Bla18] David L. Black : Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation. RFC 8311, janvier 2018.

[BSA+18] Bob Briscoe, Koen De Schepper, Olga Albisser, Olivier Tilmans, Nicolas Kuhn, Gorry Fairhurst, Richard Scheffenegger, Michael Abrahamsson, Ingemar Johansson, Praveen Balasubramanian, David Pullen, Gabriel Bracha, Jonathan Morton et Dave Täht : Implementing the 'prague requirements' for low latency low loss scalable throughput (l4s). 2018.

[BSBW23] Bob Briscoe, Koen De Schepper, Marcelo Bagnulo et Greg White : Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service : Architecture. RFC 9330, janvier 2023.

[BWC+98] David L. Black, Zheng Wang, Mark A. Carlson, Walter Weiss, Elwyn B. Davies et Steven L. Blake : An Architecture for Differentiated Services. RFC 2475, décembre 1998.

[CAB+20] Shihabur Rahman Chowdhury, Anthony, Haibo Bian, Tim Bai et Raouf Boutaba : A disaggregated packet processing architecture for network function virtualization. 38(6):1075–1088, 2020.

[CB22] Marc Carrascosa et Boris Bellalta : Cloud-gaming : Analysis of google stadia traffic. *Computer Communications*, 188:99–116, apr 2022.

[CCG+16] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh et Van Jacobson : BBR : Congestion-Based Congestion Control : Measuring Bottleneck Bandwidth and Round-Trip Propagation Time. *Queue*, 14(5):20–53, octobre 2016. Place : New York, NY, USA Publisher : Association for Computing Machinery.

[CCG+17] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh et Van Jacobson : Bbr : Congestion-based congestion control. *Commun. ACM*, 60(2):58–66, jan 2017.

[CCG+18] N. Cardwell, Y. Cheng, C.S. Gunn, S.H. Yeganeh, I. Swett, V. Vassiliev, J. Iyengar, V.P.V. Jha, Y. Seung, K. Yang, M. Mathis et V. Jacobson : Bbrv2 : A model-based congestion control. `https://datatracker.ietf.org/meeting/102/materials/slides-102-iccrg-an-update-on-bbr-work-at-google-00`, 2018. Vu le 22 septembre 2023.

[CCH+14] K. Chen, Y. Chang, H. Hsu, D. Chen, C. Huang et C. Hsu : On the Quality of Service of Cloud Gaming Systems. *IEEE Transactions on Multimedia*, 16(2):480–495, 2014.

[CDCHM16] Gaetano Carlucci, Luca De Cicco, Stefan Holmer et Saverio Mascolo : Analysis and Design of the Google Congestion Control for Web Real-Time Communication (WebRTC). *In Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, New York, NY, USA, 2016. Association for Computing Machinery. event-place : Klagenfurt, Austria.

[CFGS12] M. Claypool, D. Finkel, A. Grant et M. Solano : Thin to win ? Network performance analysis of the OnLive thin client game system. *In 2012 11th Annual Workshop on Network and Systems Support for Games (NetGames)*, pages 1–6, 2012.

[CHL09] K. Chen, P. Huang et C. Lei : Effect of Network Quality on Player Departure Behavior in Online Games. *IEEE Transactions on Parallel and Distributed Systems*, 20(5):593–606, 2009.

[CJS+19] Yi Cao, Arpit Jain, Kriti Sharma, Aruna Balasubramanian et Anshul Gandhi : When to use and when not to use BBR : an empirical analysis and evaluation study. *In Proceedings of the Internet Measurement Conference, IMC 2019, Amsterdam, The Netherlands, October 21-23, 2019*, pages 130–136. ACM, 2019.

[CM12] Giovanna Carofiglio et Luca Muscariello : On the impact of tcp and per-flow scheduling on internet performance. *IEEE/ACM Transactions on Networking*, 20(2):620–633, 2012.

[Cor23] Intel Corporation : Data plane develoment kit. `https://www.intel.com/content/www/us/en/developer/topic-technology/networking/dpdk.html`, note = "Vu le 1er octobre 2023", 2023.

[CS23] Stuart Clayman et Müge Sayit : Low latency low loss media delivery utilizing in-network packet wash. 31(1):29, 2023.

[DAA10] Salih Dikbas, Tarik Arici et Yucel Altunbasak : Fast motion estimation with interpolation-free sub-sample accuracy. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(7):1047–1051, 2010.

[DAD15] Yogesh Dhote, Shikha Agrawal et Anjana Jayant Deen : A survey on feature selection techniques for internet traffic classification. *In 2015 International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 1375–1380, 2015.

[DDPT+21] Andrea Di Domenico, Gianluca Perna, Martino Trevisan, Luca Vassio et Danilo Giordano : A network analysis on cloud gaming : Stadia, geforce now and psnow. *Network*, 1(3):247–260, 2021.

[Dev99a] M Devera : Hierachical token bucket theory. `http://luxik.cdi.cz/~devik/qos/htb/manual/theory.htm`, 1999. Vu le 12 septembre 2023.

[Dev99b] M Devera : Htb linux queuing discipline manual - user guide. `http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm`, 1999. Vu le 12 septembre 2023.

[DGL+17] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch-Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin et Larisa Safina : Microservices : Yesterday, today, and tomorrow. *In* Manuel Mazzara et Bertrand Meyer, éditeurs : *Present and Ulterior Software Engineering*, pages 195–216. Springer, 2017.

[DKS89] Alan J. Demers, Srinivasan Keshav et Scott Shenker : Analysis and simulation of a fair queueing algorithm. *In* Lawrence H. Landweber, éditeur : *Proceedings of the ACM Symposium on Communications Architectures & Protocols, SIGCOMM 1989, Austin, TX, USA, September 19-22, 1989*, pages 1–12. ACM, 1989.

[dSJPG18] Marcus Vinicius Brito da Silva, Arthur Selle Jacobs, Ricardo Jose Pfitscher et Lisandro Zambenedetti Granville : Ideafix : Identifying elephant flows in p4-based ixp networks. *In IEEE Global Communications Conference (GLOBECOM 18)*, pages 1–6, 2018.

[ETS12] ETSI : Network functions virtualisation—introductory white paper. `https://portal.etsi.org/nfv/nfv_white_paper.pdf`, 2012. Vu le 30 septembre 2023.

[FCJ+14] D. Finkel, M. Claypool, S. Jaffe, T. Nguyen et B. Stephen : Assignment of games to servers in the OnLive cloud game system. *In 2014 13th Annual Workshop on Network and Systems Support for Games*, pages 1–3, 2014.

[FGS01] Sally Floyd, Ramakrishna Gummadi et Scott Shenker : Adaptive red : An algorithm for increasing the robustness of red's active queue management. 09 2001.

[FJ93] S. Floyd et V. Jacobson : Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[FR21] Julius Flohr et Erwin P. Rathgeb : Reducing End-to-End Delays in WebRTC using the FSE-NG Algorithm for SCReAM Congestion Control. *In 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–4, 2021.

[FRB01] Sally Floyd, Dr K. K. Ramakrishnan et David L. Black : The Addition of Explicit Congestion Notification (ECN) to IP, septembre 2001. Issue : 3168 Num Pages : 63 Series : Request for Comments Published : RFC 3168.

[GFN] How can i reduce lag or improve streaming quality when using geforce now ? https://nvidia.custhelp.com/app/answers/detail/a_id/4504/~/how-can-i-reduce-lag-or-improve-streaming-quality-when-using-geforce-now%3F. Vu le 4 mars 2023.

[GMC+21] Philippe Graff, Xavier Marchal, Thibault Cholez, Stéphane Tuffin, Bertrand Mathieu et Olivier Festor : An Analysis of Cloud Gaming Platforms Behavior under Different Network Constraints. *In HiPNet 2021 - 3rd International Workshop on High-Precision, Predictable, and Low-Latency Networking*, Workshops of the 17th International Conference on Network and Service Management (CNSM21), page 7, Izmir (Virtual), Turkey, octobre 2021. IFIP-IEEE, IEEE.

[GMC+23] Philippe Graff, Xavier Marchal, Thibault Cholez, Bertrand Mathieu et Olivier Festor : Efficient Identification of Cloud Gaming Traffic at the Edge. *In NOMS 2023 - 36th IEEE/IFIP Network Operations and Management Symposium*, page 10, Miami, United States, mai 2023.

[GML+22] Gergő Gombos, Maurice Mouw, Sándor Laki, Chrysa Papagianni et Koen De Schepper : Active queue management on the tofino programmable switch : The (dual)pi2 case. *In ICC 2022 - IEEE International Conference on Communications*, pages 1685–1691, 2022.

[GN11] Jim Gettys et Kathleen Nichols : Bufferbloat : Dark buffers in the internet : Networks without effective aqm may again be vulnerable to congestion collapse. *Queue*, 9(11):40–54, nov 2011.

[GPM+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville et Yoshua Bengio : Generative adversarial nets. *In* Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence et Kilian Q. Weinberger, éditeurs : *Advances in Neural Information Processing Systems 27 : Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014.

[HCGR18] Rob Harrison, Qizhe Cai, Arpit Gupta et Jennifer Rexford : Network-wide heavy hitter detection with commodity switches. *Proceedings of the Symposium on SDN Research*, 2018.

[HCH+13] H. Hong, D. Chen, C. Huang, K. Chen et C. Hsu : QoE-aware virtual machine placement for cloud games. *In 2013 12th Annual Workshop on Network and Systems Support for Games (NetGames)*, pages 1–2, 2013.

[HCS+16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv et Yoshua Bengio : Binarized neural networks. *In* Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon et Roman Garnett, éditeurs : *Advances in Neural Information Processing Systems 29 : Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4107–4115, 2016.

[HHCC13] Chun-Ying Huang, Cheng-Hsin Hsu, Yu-Chun Chang et Kuan-Ta Chen : GamingAnywhere : An Open Cloud Gaming System. *In Proceedings of the 4th ACM Multimedia Systems Conference*, MMSys '13, pages 36–47, New York, NY, USA, 2013. Association for Computing Machinery. event-place : Oslo, Norway.

[HHT+15] H. Hong, C. Hsu, T. Tsai, C. Huang, K. Chen et C. Hsu : Enabling Adaptive Cloud Gaming in an Open-Source Cloud Gaming Platform. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(12):2078–2091, 2015.

[HJS+13] Mahdi Hemmati, Abbas Javadtalab, Ali Asghar Nazari Shirehjini, Shervin Shirmohammadi et Tarik Arici : Game as video : bit rate reduction through adaptive object encoding. *In NOSSDAV '13*, 2013.

[HLC+16] S. Holmer, H. Lundin, G. Carlucci, L. De Cicco et S. Mascolo : A google congestion control algorithm for real-time communication. `https://datatracker.ietf.org/doc/html/draft-ietf-rmcat-gcc-02`, 2016. Vu le 22 septembre 2023.

[HSRS12] Mahdi Hemmati, Shervin Shirmohammadi, Hesam Rahimi et Ali Asghar Nazari Shirehjini : Optimized game object selection and streaming for mobile devices optimized game object selection and streaming for mobile devices. 2012.

[Ins23] Fortune Business Insights : Market research report. `https://www.fortunebusinessinsights.com/cloud-gaming-market-102495`, 2023. Vu le 5 octobre 2023.

[IT21] Jana Iyengar et Martin Thomson : QUIC : A UDP-Based Multiplexed and Secure Transport. RFC 9000, mai 2021.

[Jan17] Ewa Czeslawa Janczukowicz : *QoS management for WebRTC : loose coupling strategies*. Thèse de doctorat, 2017. Thèse de doctorat dirigée par Bonnin, Jean-Marie Informatique Ecole nationale supérieure Mines-Télécom Atlantique Bretagne Pays de la Loire 2017.

[Joh14] Ingemar Johansson : Self-Clocked Rate Adaptation for Conversational Video in LTE. *In Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, CSWS '14, pages 51–56, New York, NY, USA, 2014. Association for Computing Machinery. event-place : Chicago, Illinois, USA.

[Jos23] Matthews Jose : *In-network real-value computation on programmable switches*. Thèse de doctorat, 2023. Thèse de doctorat dirigée par Festor, Olivier et François, Jérôme Informatique Université de Lorraine 2023.

[Jr.13] Henry S. Warren Jr. : *Hacker's Delight, Second Edition*. Pearson Education, 2013.

[JS17] I. Johansson et Z. Sarker : Self-clocked rate adaptation for multimedia. `https://www.rfc-editor.org/info/rfc8298`, décembre 2017.

[JSSH11] M. Jarschel, D. Schlosser, S. Scheuring et T. Hossfeld : An Evaluation of QoE in Cloud Gaming Based on Subjective Tests. *In 2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 330–335, 2011.

[KGMC23] Joël Roman Ky, Philippe Graff, Bertrand Mathieu et Thibault Cholez : A Hybrid P4/NFV Architecture for Cloud Gaming Traffic Detection with Unsupervised ML. *In 28th IEEE Symposium on Computers and Communications (ISCC 2023)*, pages 733–738, Gammarth, Tunisia, juillet 2023. IEEE, IEEE.

[KR01] D. Black K. Ramakrishnan, S. Floyd : The addition of explicit congestion notification (ecn) to ip. `https://datatracker.ietf.org/doc/rfc3168/`, 2001. Vu le 22 septembre 2023.

[KRW03] Jirka Klaue, Berthold Rathke et Adam Wolisz : Evalvid – a framework for video transmission and quality evaluation. *In* Peter Kemper et William H. Sanders, éditeurs : *Computer Performance Evaluation. Modelling Techniques and Tools*, pages 255–272, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[KS21] Radhakrishna Kamath et Krishna M Sivalingam : Machine learning based flow classification in dcns using p4 switches. *In 2021 International Conference on Computer Communications and Networks (ICCCN)*, pages 1–10, 2021.

[KSB+20] Ralf Kundel, Fridolin Siegmund, Jeremias Blendin, Amr Rizk et Boris Koldehofe : P4STA : high performance packet timestamping with programmable packet processors. *In NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*, pages 1–9. IEEE, 2020.

[LCC+15] Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman et Jason Flinn : Outatime : Using Speculation to Enable Low-Latency Continuous Interaction for Mobile Cloud Gaming. *In* Gaetano Borriello, Giovanni Pau, Marco Gruteser et Jason I. Hong, éditeurs : *MobiSys*, pages 151–165. ACM, 2015.

[LCC+18] Richard Li, Alexander Clemm, Uma Chunduri, Lijun Dong et Kiran Makhijani : A new framework and protocol for future networking applications. *In Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies*, NEAT '18, page 21–26, New York, NY, USA, 2018. Association for Computing Machinery.

[LFC+20] Abir Laraba, Jérôme François, Isabelle Chrisment, Shihabur Rahman Chowdhury et Raouf Boutaba : Defeating Protocol Abuse with P4 : Application to Explicit Congestion Notification. *In 2020 IFIP Networking Conference (Networking)*, Paris, France, juin 2020.

[LLM+09] Anna Larmo, Magnus Lindström, Michael Meyer, Ghyslain Pelletier, Johan Torsner et Henning Wiemann : The lte link-layer design. *IEEE Communications Magazine*, 47(4):52–59, 2009.

[LMY+19] Richard Li, Kiran Makhijani, Hamed Yousefi, Cedric Westphal, Lijun Dong, Tim Wauters et Filip De Turck : A framework for qualitative communications using big packet protocol. *In Proceedings of the ACM SIGCOMM 2019 Workshop on Networking for Emerging Applications and Technologies*, NEAT'19, pages 22–28. Association for Computing Machinery, 2019. event-place : Beijing, China.

[LPB+11] Weiyao Lin, Krit Panusopone, David M. Baylon, Ming-Ting Sun, Zhenzhong Chen et Hongxiang Li : A fast sub-pixel motion estimation algorithm for h.264/avc video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(2):237–242, 2011.

[LPYK15] Sung-Ho Lee, Jun-Sang Park, Sung-Ho Yoon et Myung-Sup Kim : High performance payload signature-based internet traffic classification system. *In 2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 491–494, 2015.

[LTC15] Y. Li, X. Tang et W. Cai : Play Request Dispatching for Efficient Virtual Machine Usage in Cloud Gaming. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(12):2052–2063, 2015.

[MAB+08] Nick McKeown, Thomas E. Anderson, H. Balakrishnan, Guru M. Parulkar, Larry L. Peterson, Jennifer Rexford, Scott Shenker et Jonathan S. Turner : Openflow : enabling innovation in campus networks. *Comput. Commun. Rev.*, 38:69–74, 2008.

[MAR+04] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco et Felipe Huici : ClickOS and the art of network function virtualization. *In 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 459–473. USENIX Association, 2014-04.

[MB02] David Meyer et Randy Bush : Some Internet Architectural Guidelines and Philosophy. RFC 3439, décembre 2002.

[McK90] Paul E. McKenney : Stochastic fairness queueing. *In Proceedings IEEE INFOCOM '90, The Conference on Computer Communications, Ninth Annual Joint Conference of the IEEE Computer and Communications Societies, The Multiple Facets of Integration, San Francisco, CA, USA, June 3-7, 1990*, pages 733–740. IEEE Computer Society, 1990.

[MFD12] Moreno Marzolla, Stefano Ferretti et Gabriele D'Angelo : Dynamic resource provisioning for cloud-based gaming infrastructures. *Computers in Entertainment*, 10(3):1–20, 2012.

[MGK+23] Xavier Marchal, Philippe Graff, Joël Roman Ky, Thibault Cholez, Stéphane Tuffin, Bertrand Mathieu et Olivier Festor : An Analysis of Cloud Gaming Platforms Behaviour Under Synthetic Network Constraints and Real Cellular Networks Conditions. *Journal of Network and Systems Management*, 31(2):39, janvier 2023.

[Mis20] Ayush Mishra : The great internet tcp congestion control census. `https://datatracker.ietf.org/meeting/109/materials/slides-109-iccrg-the-great-internet-tcp-congestion-control-census-00`, 2020. Vu le 6 octobre 2023.

[MSOY18] Kouto Miyazawa, Kanon Sasaki, Naoki Oda et Saneyasu Yamaguchi : Cycle and divergence of performance on TCP BBR. *In 7th IEEE International Conference on Cloud Networking, CloudNet 2018, Tokyo, Japan, October 22-24, 2018*, pages 1–6. IEEE, 2018.

[MUS+14] M. Manzano, M. Uruena, M. Suznjevic, E. Calle, Ja Hernandez et M. Matijasevic : Dissecting the protocol and network traffic of the OnLive cloud gaming platform. *Multimedia Systems*, 20(5):451–470, 2014. Publisher : SPRINGER.

[NGFL20] Szilveszter Nádas, Gergo Gombos, Ferenc Fejes et Sándor Laki : A congestion control independent L4S scheduler. *In ANRW '20 : Applied Networking Research Workshop, Virtual Event, Spain, July 27-30, 2020*, pages 45–51. ACM, 2020.

[NGHL18] Szilveszter Nádas, Gergo Gombos, Péter Hudoba et Sándor Laki : Towards a congestion control-independent core-stateless AQM. *In Proceedings of the Applied Networking Research Workshop, ANRW 2018, Montreal, QC, Canada, July 16-16, 2018*, pages 84–90. ACM, 2018.

[NGJ14] Aline Normoyle, Gina Guerrero et Sophie Jörg : Player perception of delays and jitter in character responsiveness. *In Proceedings of the ACM Symposium on Applied Perception*, SAP '14, page 117–124, New York, NY, USA, 2014. Association for Computing Machinery.

[NJ12] Kathleen Nichols et Van Jacobson : Controlling Queue Delay : A Modern AQM is Just One Piece of the Solution to Bufferbloat. *Queue*, 10(5):20–34, mai 2012. Place : New York, NY, USA Publisher : Association for Computing Machinery.

[NJMI18] Kathleen Nichols, Van Jacobson, Andrew McGregor et Jana Iyengar : Controlled Delay Active Queue Management, janvier 2018. Issue : 8289 Num Pages : 25 Series : Request for Comments Published : RFC 8289.

[NMN+14] Bruno Astuto A. Nunes, Marc Mendonca, Xuan Nam Nguyen, Katia Obraczka et Thierry Turletti : A survey of software-defined networking : Past, present, and future of programmable networks. *IEEE Commun. Surv. Tutorials*, 16(3):1617–1634, 2014.

[NSD+15] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens et Hari Balakrishnan : Mahimahi : Accurate record-and-replay for HTTP. *In* Shan Lu et Erik Riedel, éditeurs : *2015 USENIX Annual Technical Conference, USENIX ATC '15, July 8-10, Santa Clara, CA, USA*, pages 417–429. USENIX Association, 2015.

[NSN+17] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar et Changhoon Kim : Language-directed hardware design for network performance monitoring. *In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*, pages 85–98. ACM, 2017.

[OGBT20a] Dejene Boru Oljira, Karl-Johan Grinnemo, Anna Brunstrom et Javid Taheri : Guidelines and scripts to reproduce the results in this paper. `https://git.cs.kau.se/oljideje/l4s-validation-artifacts`, 2020.

[OGBT20b] Dejene Boru Oljira, Karl-Johan Grinnemo, Anna Brunström et Javid Taheri : Validating the sharing behavior and latency characteristics of the L4S architecture. *Comput. Commun. Rev.*, 50(2):37–44, 2020.

[PHJ06] Colin Perkins, Mark J. Handley et Van Jacobson : SDP : Session Description Protocol. RFC 4566, juillet 2006.

[PNP+13] Rong Pan, Preethi Natarajan, Chiara Piglione, Mythili Suryanarayana Prabhu, Vijay Subramanian, Fred Baker et Bill VerSteeg : PIE : A lightweight control scheme to address the bufferbloat problem. *In 2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*, pages 148–155, 2013.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot et E. Duchesnay : Scikit-learn : Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[REG14] K. Raaen, R. Eg et C. Griwodz : Can gamers detect cloud delay ? *In 2014 13th Annual Workshop on Network and Systems Support for Games*, pages 1–3, 2014.

[Res23] Ericsson Research : Scream - mobile optimised congestion control algorithm. `https://github.com/EricssonResearch/scream`, 2023.

[RJLHF22] Flavien Ronteix-Jacquet, Xavier Lagrange, Isabelle Hamchaoui et Alexandre Ferrieux : Rethinking buffer status estimation to improve radio resource utilization in cellular networks. *In 2022 IEEE 95th Vehicular Technology Conference : (VTC2022-Spring)*, pages 1–5, 2022.

[RKL20] Shahbaz Rezaei, Bryce Kroencke et Xin Liu : Large-scale mobile app identification using deep learning. *IEEE Access*, 8:348–362, 2020.

[RSS11] Hesam Rahimi, Ali Asghar Nazari Shirehjini et Shervin Shirmohammadi : Context-aware prioritized game streaming. *2011 IEEE International Conference on Multimedia and Expo*, pages 1–6, 2011.

[SB18] Giuseppe Siracusano et Roberto Bifulco : In-network neural networks. *CoRR*, abs/1801.05731, 2018.

[SB23] Koen De Schepper et Bob Briscoe : The Explicit Congestion Notification (ECN) Protocol for Low Latency, Low Loss, and Scalable Throughput (L4S). RFC 9331, janvier 2023.

[SBB+20] German Sviridov, Cedric Beliard, Andrea Bianco, Paolo Giaccone et Dario Rossi : Removing human players from the loop : AI-assisted assessment of Gaming QoE. *In IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1160–1165, 2020.

[SBSK+14] Mirko Suznjevic, Justus Beyer, Lea Skorin-Kapov, Sebastian Moller et Nikola Sorsa : Towards understanding the relationship between game type and network traffic for cloud gaming. *In 2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, pages 1–6, 2014.

[SBTB16] Koen De Schepper, Olga Bondarenko, Ing Jyh Tsang et Bob Briscoe : Pi^2 : A linearized AQM for both classic and scalable TCP. *In* Athina Markopoulou, Michalis Faloutsos, Vyas Sekar et Dejan Kostic, éditeurs : *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies, CoNEXT 2016, Irvine, California, USA, December 12-15, 2016*, pages 105–119. ACM, 2016.

[SCFJ03] Henning Schulzrinne, Stephen L. Casner, Ron Frederick et Van Jacobson : RTP : A Transport Protocol for Real-Time Applications. RFC 3550, juillet 2003.

[SCP20] Kyle A. Simpson, Richard Cziva et Dimitrios P. Pezaros : Seiðr : Dataplane assisted flow classification using ML. *In IEEE Global Communications Conference, GLOBECOM 2020, Virtual Event, Taiwan, December 7-11, 2020*, pages 1–6. IEEE, 2020.

[SHJ+14] M. Semsarzadeh, M. Hemmati, A. Javadtalab, A. Yassine et S. Shirmohammadi : A video encoding speed-up architecture for cloud gaming. *In 2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, pages 1–6, 2014.

[SKA+17] Naveen Kr. Sharma, Antoine Kaufmann, Thomas Anderson, Arvind Krishnamurthy, Jacob Nelson et Simon Peter : Evaluating the power of flexible packet processing for network resource allocation. *In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 67–82, Boston, MA, mars 2017. USENIX Association.

[SL91] S.R. Safavian et D. Landgrebe : A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3):660–674, 1991.

[SLNC13] R. Shea, J. Liu, E. C. Ngai et Y. Cui : Cloud gaming : architecture and performance. *IEEE Network*, 27(4):16–21, 2013.

[SNR+16] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan et Jennifer Rexford : Heavy-hitter detection entirely in the data plane. *Proceedings of the Symposium on SDN Research*, 2016.

[SPSR21] Zaheduzzaman Sarker, Colin Perkins, Varun Singh et Michael A. Ramalho : RTP Control Protocol (RTCP) Feedback for Congestion Control. RFC 8888, janvier 2021.

[SS21] Tal Shapira et Yuval Shavitt : FlowPic : A generic representation for encrypted traffic classification and applications identification. 18(2):1218–1232, 2021.

[SSB18] Davide Sanvito, Giuseppe Siracusano et Roberto Bifulco : Can the network be the AI accelerator ? *In* Xin Jin et Changhoon Kim, éditeurs : *Proceedings of the 2018 Morning Workshop on In-Network Computing, NetCompute@SIGCOMM 2018, Budapest, Hungary, August 20, 2018*, pages 20–25. ACM, 2018.

[SSH+23] Jangwoo Son, Yago Sanchez, Christian Hampe, Dominik Schnieders, Thomas Schierl et Cornelius Hellge : L4S congestion control algorithm for interactive low latency applications over 5g. *In IEEE International Conference on Multimedia and Expo, ICME 2023, Brisbane, Australia, July 10-14, 2023*, pages 1002–1007. IEEE, 2023.

[SSSK16] M. Suznjevic, I. Slivar et L. Skorin-Kapov : Analysis and QoE evaluation of cloud gaming service adaptation under different network conditions : The case of NVIDIA GeForce NOW. *In 2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6, 2016.

[TDP+19] Yuta Tokusashi, Huynh Tu Dang, Fernando Pedone, Robert Soulé et Noa Zilberman : The case for in-network computing on demand. *In* George Candea, Robbert van Renesse et Christof Fetzer, éditeurs : *Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, March 25-28, 2019*, pages 21 :1–21 :16. ACM, 2019.

[TKU19] Belma Turkovic, Fernando A. Kuipers et Steve Uhlig : Interactions between congestion control algorithms. *In* Stefano Secci, Isabelle Chrisment, Marco Fiore, Lionel Tabourier et Keun-Woo Lim, éditeurs : *Network Traffic Measurement and Analysis Conference, TMA 2019, Paris, France, June 19-21, 2019*, pages 161–168. IEEE, 2019.

[TLL+04] Zhaowei Tan, Yuanjie Li, Qianru Li, Zhehui Zhang, Zhehan Li et Songwu Lu : Supporting mobile VR in LTE networks : How close are we ? 2(1), 2018-04. Place : New York, NY, USA Publisher : Association for Computing Machinery.

[TSZ+21] Li-Zhuang Tan, Wei Su, Wei Zhang, Jianhui Lv, Zhenyi Zhang, Jingying Miao, Xiaoxi Liu et Na Li : In-band network telemetry : A survey. *Comput. Networks*, 186:107763, 2021.

[TWH+15] H. Tian, D. Wu, J. He, Y. Xu et M. Chen : On Achieving Cost-Effective Adaptive Cloud Gaming in Geo-Distributed Data Centers. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(12):2064–2077, 2015.

[UE15] UE : Journal officiel de l'union européenne. `https://www.arcep.fr/fileadmin/reprise/textes/communautaires/reglement-UE-2015_310-Net-Neutralite-251115.pdf`, 2015. Vu le 9 octobre 2023.

[VvvD15] Petr Velan, Milan Čermák, Pavel Čeleda et Martin Drašar : A survey of methods for encrypted traffic classification and analysis. *Netw.*, 25(5):355–374, 09 2015.

[WD10a] S. Wang et S. Dey : Addressing Response Time and Video Quality in Remote Server Based Internet Mobile Gaming. *In 2010 IEEE Wireless Communication and Networking Conference*, pages 1–6, 2010.

[WD10b] S. Wang et S. Dey : Rendering Adaptation to Address Communication and Computation Constraints in Cloud Mobile Gaming. *In 2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–6, 2010.

[Wei20] Guanglu Wei : Deep learning model under complex network and its application in traffic detection and analysis. *In 2020 IEEE 2nd International Conference on Civil Aviation Safety and Information Technology (ICCASIT*, pages 448–453, 2020.

[WSB04] Keith Winstein, Anirudh Sivaraman et Hari Balakrishnan : Stochastic forecasts achieve high throughput and low delay over cellular networks. *In 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 459–471. USENIX Association, 2013-04.

[WYC+15] J. Wu, C. Yuen, N. Cheung, J. Chen et C. W. Chen : Enabling Adaptive High-Frame-Rate Video Streaming in Mobile Cloud Gaming Applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(12):1988–2001, 2015.

[WZZ+17] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye et Yiqiang Sheng : Malware traffic classification using convolutional neural network for representation learning. *In 2017 International Conference on Information Networking (ICOIN)*, pages 712–717, 2017.

[XC22a] Xiaokun Xu et Mark Claypool : Measurement of cloud-based game streaming system response to competing tcp cubic or tcp bbr flows. *In Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, page 305–316, New York, NY, USA, 2022. Association for Computing Machinery.

[XC22b] Xiaokun Xu et Mark Claypool : Measurement of the responses of cloud-based game streaming to network congestion. *In Proceedings of the 32nd Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '22, pages 22–28. Association for Computing Machinery, 2022. event-place : Athlone, Ireland.

[XZ19] Zhaoqi Xiong et Noa Zilberman : Do switches dream of machine learning ? : Toward in-network classification. *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, 2019.

[YCHL12] Yeng-Ting Lee, K. Chen, Han-I Su et C. Lei : Are all games equally cloud-gaming-friendly ? An electromyographic approach. *In 2012 11th Annual Workshop on Network and Systems Support for Games (NetGames)*, pages 1–6, 2012.

[YFJR21] Lixuan Yang, Alessandro Finamore, Feng Jun et Dario Rossi : Deep learning and traffic classification : Lessons learned from a commercial-grade dataset with hundreds of encrypted and zero-day applications. *CoRR*, abs/2104.03182, 2021.

[YXLL14] Chenguang Yu, Yang Xu, Bo Liu et Yong Liu : "Can you SEE me now ?" A measurement study of mobile video calls. *In IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 1456–1464, 2014.

[ZHR+16] Wei Zhang, Jinho Hwang, Shriram Rajagopalan, K. K. Ramakrishnan et Timothy Wood : Flurries : Countless fine-grained NFs for flexible per-flow customization. *In Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, pages 3–17. Association for Computing Machinery, 2016.

[ZP13] Xiaoqing Zhu et Rong Pan : Nada : A unified congestion control scheme for low-latency interactive video. *In 2013 20th International Packet Video Workshop*, pages 1–8, 2013.

[ZSGJ19] Zhehui Zhang, Shu Shi, Varun Gupta et Rittwik Jana : Analysis of cellular network latency for edge-based remote rendering streaming applications. *In Proceedings of the ACM SIGCOMM 2019 Workshop on Networking for Emerging Applications and Technologies*, NEAT'19, pages 8–14. Association for Computing Machinery, 2019. event-place : Beijing, China.

Résumé

Caractérisation, identification dans le réseau et optimisation du transport de trafic à faible latence - le cas du Cloud-Gaming

Cette thèse s'intéresse au traitement du trafic à faible latence dans les réseaux, et en particulier aux services de type Cloud Gaming (CG) qui constituent notre cas d'étude. Ceux-ci ont fortement gagné en popularité et devraient représenter une part importante du trafic Internet dans les prochaines années. Or, les caractéristiques de leur trafic, à la fois très haut débit et à faible latence, est exigeant pour les réseaux et rend difficile le maintien d'une bonne qualité de service (QoS) en conditions de réseau dégradées.

Nous commençons par étudier le trafic de quatre plateformes de CG afin d'évaluer la capacité d'adaptation et la réactivité de leur algorithme de contrôle de congestion (CCA). Pour cela, nous dégradons synthétiquement les conditions réseau en impactant tour à tour le délais, la gigue, le taux de pertes et la bande passante disponible avec différentes intensités, et observons le trafic résultant. Dans un deuxième temps, nous étudions le trafic CG sur réseau cellulaire en reproduisant les conditions observées sur le réseau 4G d'Orange, y compris en situation de mobilité. Nous avons relevé une certaine disparité dans le comportement des plateformes. Certaines ne s'adaptent pas suffisamment aux différentes contraintes et s'exposent par conséquent à de forts délais, se soldant parfois par des pertes de paquets. D'autres surréagissent, au détriment de leur qualité vidéo. L'ensemble des données collectées a été mis à disposition de la communauté.

Constatant l'insuffisance des CCA de bout en bout, nous avons cherché dans un second temps à identifier le trafic CG dans le réseau pour lui faire bénéficier d'un traitement plus optimisé. Pour ce faire, nous avons mis au point un classificateur permettant de reconnaître avec une haute précision (98.5%) le trafic CG en s'appuyant sur un apprentissage automatique des propriétés statistiques du trafic qui sont calculées à la volée. Nous proposons une implantation de notre classificateur sous forme de plusieurs fonctions réseau virtuelles (VNF) pouvant traiter 10Gb/s. En partenariat avec Orange Labs, nous avons également étudié l'accélération de certaines fonctions sur un switch matériel programmable en P4 et en avons implanté certaines comme le calcul des caractéristiques, ce qui prouve qu'un déploiement réaliste au niveau d'un fournisseur d'accès est possible.

Enfin, nous proposons de faire bénéficier le trafic CG d'une architecture à double file d'attente afin d'éviter, d'une part de le faire transiter par des buffers surdimensionnés qui engendrent de la latence, et d'autre part la mauvaise cohabitation avec des flux régis par des CCA basé pertes. Nous évaluons ainsi l'apport d'une discipline de file d'attente HTB ou de l'AQM DualPI2 disposant d'un file "Low Latency, Low Loss, and Scalable Throughput" (L4S). Les plateformes actuelles de CG n'étant pas compatibles, nous avons développé notre propre plateforme expérimentale et l'avons munie de SCReAM, un CCA basé pertes et délais pour le protocole RTP et supportant la notification explicite de congestion (ECN). Nous évaluons alors le trafic face à différents flux concurrents soumis à un goulot d'étranglement. Nos résultats montrent que les deux approches permettent de préserver parfaitement la QoS du trafic CG.

Mots-clés: Cloud Gaming, Flux faible latence, Classification de flux, P4, Qualité de service

Abstract

Characterization, in-network identification and optimization of low-latency traffic transport - the case of Cloud-Gaming

This thesis focuses on the transport of low-latency traffic in networks, and in particular on Cloud Gaming (CG) services we selected as our case study. CG plaforms have gained much popularity recently and are expected to become a significant part of Internet traffic share in the upcoming years. However, the characteristics of their traffic, at the same time high bandwidth and low latency, are challenging for networks and make it difficult to maintain good quality of service (QoS) in degraded network conditions.

We first analyze the traffic of four CG platforms to evaluate the capacity of adaptation and responsiveness of their congestion control algorithm (CCA). We synthetically lower the network conditions by impacting in turn the delay, jitter, loss rate and available bandwidth with different intensities, and observe the resulting traffic. Then, we study CG traffic on cellular networks by reproducing the conditions observed on the Orange 4G network, including a user mobility scenario. We noted a some disparities in the platforms' behavior. Some do not adapt adequately to the different constraints and are therefore exposed to long queuing delays, sometimes resulting in packet losses. Others overreact, to the detriment of their video quality. All of the data collected was made available to the community.

Considering the limitations of end-to-end CCA, we propose to identify CG traffic in the network to make it benefit from an optimized traffic processing. We have developed a classifier able to recognize CG traffic with high precision (98.5%) by relying on a machine learning approach featuring the statistical properties of CG traffic which are calculated on the fly. We propose an implementation of our classifier as a set of virtual network functions (VNF) that can handle 10Gb/s. In collaboration with Orange Labs, we also considered the acceleration of some functions on a programmable hardware switch in P4 and implemented the extraction of flow statistical features, which proves that a realistic deployment at an ISP level is possible.

Finally, we propose to experiment the transport of CG traffic in a double queue architecture in order to avoid oversized buffers which generate latency and the poor cohabitation with flows driven by loss-based CCAs. We consider a HTB queuing discipline or the AQM DualPI2 with a "Low Latency, Low Loss, and Scalable Throughput" (L4S) queue. As current CG platforms are not compatible, we developed our own experimental platform coupled with SCReAM, a CCA based on loss and delays for the RTP protocol and supporting explicit congestion notification (ECN). We then monitor the traffic on the bottleneck against different competing flows. Our results show that both approaches perfectly preserve the QoS of CG traffic.

Keywords: Cloud Gaming, Low Latency Traffic, Flow Classification, P4,Quality of Service