Architecture de gestion et de contrôle des ressources pour les applications multimédia dans le réseau local domestique

Maxime Louvel

THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE

sous le sceau de l'Université européenne de Bretagne

pour obtenir le titre de

DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE

Mention : Informatique

École Doctorale Santé, Information-Communications, Mathématiques, Matière (SICMA)

présentée par

Maxime Louvel

Préparée au Laboratoire d'Informatique des Systèmes Complexes (LISyC) et au sein d'Orange Labs

Architecture de gestion et de contrôle des ressources pour les applications multimédia dans le réseau local domestique

Thèse soutenue le 17 novembre 2011

devant le jury composé de :

Jean-Philippe Babau
Professeur à l'Université de Bretagne Occidentale, LISyC / directeur de thèse

Isabelle Demeure
Professeur à TELECOM Paris Tech, Institut Telecom / rapporteur - présidente

Didier Donsez
Professeur à l'Université Joseph Fourier, LIG / rapporteur

Eric Gressier-Soudan
Professeur au CNAM, CEDRIC / examinateur

Jérôme Hugues
Maître de Conférences à l'ISAE, DMIA/MARS / examinateur

Alain Plantec
Maître de Conférences à l'Université de Bretagne Occidentale, LISyC / co-encadrant

Jacques Pulou
Ingénieur de Recherche France Telecom R&D / examinateur

Remerciements

Je tiens tout d'abord à remercier les membres du jury pour le temps qu'ils ont consacré à évaluer mon travail de thèse. Je m'adresse tout particulièrement aux deux rapporteurs, pour leurs remarques intéressantes et constructives sur mon travail.

Je remercie ensuite mon directeur de thèse, Jean-Philippe, qui aura su m'initier à la recherche avec une bonne dose de critiques maîtrisées, toujours réfléchies et constructives. J'aurai particulièrement apprécié son franc-parler et son honnêteté. Mes remerciements vont aussi à Alain Plantec, pour le temps passé en relecture, conseils et écriture.

Cette thèse n'aurait pas vu le jour sans Jacques Pulou qui a rendu l'aventure possible à France Telecom. Grâce à lui j'ai pu faire mon chemin sur la berge industrielle de la recherche. Il est important de remercier aussi l'entreprise, France Telecom, qui accueille de nombreux autres thésards dans les différents Orange labs.

Je tiens ensuite à exprimer ma reconnaissance aux personnes qui ont travaillé avec moi, sans lesquelles je n'aurais pas eu les résultats scientifiques présentés dans ce manuscrit. Je remercie particulièrement Julien Tous pour son aide précieuse. Merci aussi à Pierre Bonhomme et Loïc Bith pour leurs développements. Enfin je remercie Laurent Lemarchand pour ses compétences mathématiques et algorithmiques.

J'ai eu la chance d'effectuer cette thèse dans une URD et un laboratoire composés de personnes très chaleureuses et accueillantes. Un grand merci à ces deux équipes. Merci à toute l'équipe Brestoise pour leur accueil lors de mes voyages réguliers à l'Ouest. Une thèse en industrie regorge parfois de surprises et d'embûches. Je tiens à remercier les deux managers que j'ai connu et qui ont su m'écouter et m'intégrer dans la R&D de France Telecom.

Trois ans de thèse contiennent toujours des hauts des bas. Durant ces différentes phases j'ai eu la chance d'avoir des oreilles attentives à mes élucubrations. Je pense particulièrement à la bande de thésards et aux personnes de mon équipe qui ont pris le temps de m'écouter. Il est ensuite important d'exprimer ma reconnaissance envers mes co-bureaux pour m'avoir supporté tous les jours. Un grand merci aussi à tous ceux qui m'auront permis de m'aérer durant ces trois ans, que ce soit à l'extérieur ou à l'intérieur, le midi ou autour du temps de midi.

Ce manuscrit n'aurait pas la même qualité d'expression sans les nombreuses relectures, merci à tous ceux qui y ont participé.

Sur le plan personnel, je remercie les amis qui ont été là durant ces trois années. Je remercie aussi ma famille et tout particulièrement mes parents qui ont toujours cru en moi.

Enfin j'exprime toute ma gratitude à ma femme qui m'a épaulé, soutenu et beaucoup aidé dans cette thèse.

Résumé

Le réseau local domestique est un environnement ouvert, hétérogène et distribué pour lequel il est primordial de garantir la qualité de service des applications multimédia. Des mécanismes de réservation de ressources (CPU, mémoire, réseau) et des architectures utilisant ces mécanismes existent. Les architectures et les mécanismes existants nécessitent de modifier les équipements ou les applications et ne prennent pas en compte l'hétérogénéité inhérente au domaine d'étude. Ces solutions ne sont pas adaptées au contexte du réseau local domestique.

Pour répondre au problème de l'hétérogénéité, ce travail propose de limiter le nombre de mesures à réaliser et à stocker en agrégeant les quantités de ressources. Cette agrégation est automatisée à l'aide d'algorithmes basés sur du *clustering* ou du *bin-packing*. Cette thèse propose ensuite un *framework* de gestion des ressources peu intrusif, reposant sur une architecture configurable en fonction des équipements présents. Cette architecture utilise des composants globaux qui délèguent, aux composants locaux, la gestion des ressources locales. Les composants locaux utilisent les mécanismes de réservation du système d'exploitation Linux pour garantir les quantités de ressources aux applications.

L'agrégation évaluée par simulation, réduit efficacement le nombre de mesures à réaliser et à stocker. De son côté, le *framework* de gestion des ressources est mis œuvre sur des équipements réels (des PCs, des ordinateurs portables et des équipements embarqués dédiés au multimédia), communiquant via des réseaux Wifi et Ethernet. Les évaluations du *framework* montrent que les réservations sont garanties même quand du bruit est généré sur les ressources utilisées, ce qui garantit aussi la qualité de service attendue.

Abstract

The home network is an open, heterogeneous and distributed environment where ensuring multimedia applications' quality of service is a main concern. Mechanisms to reserve resources (CPU, memory, network) and architectures using them already exist. However, they require modifying the devices or the applications and they do not take into account the heterogeneity of the home network.

To tackle the heterogeneity related issues, this work limits the number of measurements to make and to store by aggregating the quantities of resources. This aggregation is automated using clustering or bin-packing algorithms. Then a non-intrusive resource management framework is developed upon an architecture, customized for the actual devices. This architecture uses global components that delegate to local components the management of local resources. These components rely on the resource reservation mechanisms provided by the Linux operating system in order to guarantee the resources to the applications.

The aggregation, which we evaluated through simulation, efficiently reduces the number of measurements to carry and to store. The framework has been implemented on real devices (PCs, laptops and embedded multimedia devices), bridged with wireless and Ethernet networks. The evaluations of the framework show that reservations are guaranteed even if noise is generated on the resources, which also guaranties the expected quality of service.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Contexte

Ces dernières années, le réseau local domestique est passé de quelques PCs connectés à Internet, à un environnement ouvert qui interconnecte de nombreux équipements accédant aux contenus multimédia distribués dans ce réseau [63]. Historiquement, les opérateurs de télécommunication fournissent à leurs clients un accès à Internet et des services multimédia, via les *Set-Top-Boxes* (décodeurs TV). Ces services sont maintenant accessibles via des télévisons connectées ou des téléphones mobiles achetés auprès de fournisseurs d'équipements. En outre, de nouvelles applications de partage de contenus multimédia ont fait leur apparition dans le réseau local domestique. Cet environnement est aujourd'hui partagé entre trois acteurs principaux : les opérateurs de télécommunication, les fournisseurs d'équipements et les fournisseurs d'applications.

Dans ce contexte, les contenus multimédia sont visionnés sur de nombreux équipements [99]. En plus des équipements historiques (*Set-Top-Boxes*) et des télévisons, les utilisateurs veulent visionner leurs contenus sur des PCs, des tablettes, des téléphones portables, des consoles de jeux et sur les équipements futurs.

Or, pour ce genre d'applications, il est primordial de garantir la qualité de service perçue par l'utilisateur. Celle-ci étant subjective, des contraintes objectives sont définies sur les applications. Par exemple, un flux vidéo doit être affiché avec une fréquence de 25 images par seconde. Pour satisfaire ces contraintes, une application a besoin d'avoir assez de ressources (CPU, mémoire et réseau) pour s'exécuter correctement et fournir le niveau de qualité de service attendu. Cette thèse s'intéresse spécifiquement à la gestion des ressources et plus particulièrement à la gestion des ressources dans le contexte hétérogène et distribué du réseau local domestique.

Problématique

Une application multimédia distribuée utilise plusieurs ressources, parmi lesquelles on retrouve classiquement des ressources d'exécution, de stockage et de communication. Afin de garantir qu'une application possède assez de ressources pour fournir la qualité de service attendue, il est classique de réserver à l'avance une part de chaque ressource pour cette application [81, 97]. Pour ce faire, des mécanismes de

réservation ont été ajoutés aux systèmes d'exploitation temps réel. Un système temps réel est un système dans lequel les applications ont des contraintes temporelles.

Dans le domaine des systèmes temps réel, un effort d'intégration a été fait dernièrement pour offrir une gestion coordonnée des ressources pour des applications distribuées. Ces travaux se matérialisent par des *frameworks* s'articulant autour d'une architecture de gestion des ressources. Ces *frameworks* effectuent des réservations sur toutes les ressources utilisées fournissant ainsi un cadre d'exécution garantissant à l'application l'accès aux ressources. Cependant deux limites apparaissent pour utiliser ces *frameworks* dans le réseau local domestique :

- Tous d'abord ces *frameworks* supposent que les quantités de ressources requises par une application sont connues sur chaque ressource. Or, pour les applications multimédia, ces quantités dépendent du flux utilisé, des paramètres d'encodage du flux, des logiciels utilisés, de la ressource elle-même et des autres ressources utilisées. De ce fait, il n'est pas évident de connaître les quantités de ressources requises par une application.
- Le deuxième frein à l'utilisation de ces *frameworks* est que les mécanismes utilisés ne sont pas disponibles sur les équipements du réseau local domestique. En effet, ces travaux s'intéressent aux systèmes temps réel "classiques" tels les systèmes de contrôle de missions, de contrôle de véhicule ou de satellite. Des efforts d'intégration des mécanismes utilisés dans des systèmes d'exploitation généraux ont été menés. Pour ce faire, le système d'exploitation des équipements et l'application sont modifiés pour utiliser les mécanismes de réservation. Dans le contexte du réseau local domestique, les équipements appartiennent à différents acteurs. De ce fait, il est quasiment impossible d'imposer l'introduction d'un mécanisme sur un équipement. De plus, la tendance dans le réseau local domestique est à l'interconnexion automatique et transparente des équipements et des services. C'est pourquoi, une solution intrusive, nécessitant de modifier les équipements ou les applications, n'est pas adaptée.

Au vu de ces deux limites, les solutions de gestion des ressources existantes n'apparaissent pas directement utilisables avec les équipements du réseau local domestique. Pour répondre à ces limites, les contributions de la thèse visent à offrir un *framework* de gestion des ressources dans le contexte hétérogène du réseau local domestique, en étant le moins intrusif possible.

Contributions

Les contributions de cette thèse s'articulent autour de trois points. Les deux premiers correspondent aux deux limites exprimées sur les *frameworks* existants. Le dernier point porte sur la mise en œuvre de notre *framework* sur des équipements Linux.

Connaissance des quantités de ressources requises Au vu de l'hétérogénéité du réseau local domestique, connaître et stocker une valeur de quantité de ressources pour tous les cas possibles s'avère prohibitif. Pour répondre à ce problème, les valeurs similaires sont agrégées pour réduire la quantité d'information à mesurer et à stocker. Nous présentons une méthode d'agrégation et deux algorithmes automatisant cette agrégation.

Proposition d'un *framework* de gestion des ressources Dans cette thèse, nous proposons un *framework* de gestion des ressources, appelé ARMOR (ARMOR *is A Resource Management framewORk*) qui se base uniquement sur des mécanismes présents sur les équipements du réseau local domestique. ARMOR utilise son architecture pour masquer l'hétérogénéité des ressources du réseau local domestique. ARMOR est construit de manière à être le moins intrusif possible. Enfin, ARMOR garantit aux applications l'accès aux ressources à l'aide de mécanismes de réservation. Ces réservations sont coordonnées par les composants d'ARMOR et mises en œuvre en utilisant les mécanismes du système d'exploitation des équipements.

Mise en œuvre du *framework* Pour montrer sa faisabilité, nous avons mis en œuvre notre *framework* de gestion des ressources sur différents équipements du réseau local domestique. Nous nous sommes focalisés sur les équipements utilisant le système d'exploitation Linux. Notre *framework* a été implémenté et testé sur des PCs, des ordinateurs portables et des équipements embarqués dédiés au multimédia. La version actuelle de notre *framework* supporte les réseaux de type Wifi et Ethernet. Enfin, nous avons testé notre *framework* avec des flux aux standards *h.264* et *mpeg4-part2* qui sont classiquement utilisés aujourd'hui.

Plan

Le plan de ce rapport se divise en trois parties.

La première partie porte sur le réseau local domestique et fait un état de l'art des mécanismes de réservation disponibles aujourd'hui. Cet état de l'art présente aussi les *frameworks* de gestion des ressources existants, leurs intérêts et leurs limites dans le cadre du réseau local domestique.

La deuxième partie de ce rapport, divisée en deux chapitres, détaille les contributions de cette thèse. Le premier chapitre présente comment les quantités de ressources sont estimées et agrégées avant d'être stockées et réutilisées. Le deuxième chapitre détaille ARMOR, notre *framework* de gestion des ressources, son architecture et son fonctionnement.

La troisième partie de ce rapport est l'évaluation des contributions, divisée en deux chapitres. Le premier chapitre évalue l'estimation et la politique d'agrégation des quantités de ressources. Le deuxième chapitre évalue d'abord la capacité d'ARMOR à être mis en œuvre dans un réseau local domestique. Ensuite, ce chapitre montre que les réservations mises en place par ARMOR garantissent effectivement les quantités de ressources requises par les applications, ce qui garantit la qualité de service attendue.

Enfin, à la suite d'une conclusion, des perspectives sont élaborées.

Première partie

État de l'art

Chapitre 2

État de l'art

Sommaire

Afin de garantir la qualité de service de niveau utilisateur, pour des applications de diffusion de contenus multimédia, dans le cadre du réseau local domestique, cette thèse s'intéresse à maîtriser l'utilisation des ressources.

Dans une première partie, ce chapitre présente le domaine d'étude : le réseau local domestique et les applications de diffusion de contenus multimédia. Lorsqu'un utilisateur veut exécuter une application multimédia, il s'attend à un certain niveau de qualité de service. La qualité de service d'une telle application est fortement liée aux ressources utilisées. En effet, pour fournir la qualité de service attendue par l'utilisateur, l'application requiert une quantité de ressources, sur chaque ressource utilisée. Les notions relatives à la qualité de service et aux quantités de ressources sont définies par la suite.

Dans une deuxième partie, ce chapitre présente les approches permettant de gérer les quantités de ressources existantes dans la littérature. Traditionnellement, pour garantir les quantités de ressources requises sur chaque ressource, des mécanismes de réservation sont fournis par le système d'exploitation. C'est pourquoi nous présentons les mécanismes existants, pour les différentes ressources utilisées par les applications de diffusion de contenus multimédia (réseau, CPU et mémoire RAM). Pour garantir les quantités de ressources aux applications, un *framework* de gestion des quantités de ressources s'appuie sur des mécanismes de réservation. Nous présentons ensuite une étude des *frameworks* de gestion des quantités de ressources existantes.

Enfin, les limites des solutions de gestion des quantités de ressources existantes pour les applications de diffusion de contenus multimédia dans le réseau local domestique sont discutées.

2.1 Domaine d'étude

Un réseau local domestique se compose d'un ensemble d'équipements interconnectés au sein du foyer d'un utilisateur. Nous abordons d'abord le réseau local domestique, ses équipements et leur mode d'interconnexion, puis les applications de diffusion de contenus multimédia, démarrées à la demande de l'utilisateur. Lors de l'exécution de la demande d'un utilisateur, il est primordial de lui fournir le niveau de qualité de service attendu. Nous présentons comment la qualité de service d'une application de diffusion de contenus multimédia est liée au quantités de ressources. Enfin, une discussion met en évidence les enjeux d'un tel environnement.

2.1.1 Réseau local domestique

Un réseau local domestique est le réseau local d'un utilisateur. Il est nécessaire de comprendre les caractéristiques d'un réseau local, et de se pencher sur les spécificités du réseau local domestique.

2.1.1.1 Réseau Local

Un réseau local est un réseau informatique qui connecte un ensemble d'ordinateurs et autres équipements dans une zone géographique limitée telle une maison ou une entreprise. Les réseaux locaux sont apparus dans les années 1970 et ont connu un fort essor à partir des années 1980 avec l'émergence de la micro-informatique.

Pour permettre d'interconnecter les équipements de différents fournisseurs, des protocoles de communication standards ont été définis. Ces standards se structurent à l'aide du modèle *Open Systems Interconnexion* (OSI [55]). Outre ces standards, les équipements sont physiquement interconnectés, via la topologie du réseau. Les ressources d'un réseau local sont caractérisées par une capacité que nous définissons. Enfin, nous présentons les normes de qualité de service réseau, définissant des classes de trafic.

Modèle OSI Pour structurer et simplifier la définition des standards de communication, le standard OSI a été défini par l'organisme de standardisation *International Organization for Standardisation* (ISO). Ce standard définit 7 couches protocolaires. Les protocoles d'une couche s'appuient sur les protocoles de la couche inférieure, la couche la plus proche du matériel étant la couche 1. Chaque couche a un rôle spécifique dans la gestion des communications. Le Tableau 2.1 donne les sept couches du modèle OSI, leur nom et quelques exemples de protocoles pour chaque couche.

Les couches de niveaux 4 et supérieurs sont utilisées pour interconnecter les applications. La couche 3 permet d'adresser les différents équipements, grâce au protocole IP. La couche 2 définit les protocoles d'accès au medium de communication. Dans un réseau local, les équipements communiquent par l'intermédiaire des protocoles de la couche 2. Cette couche permet entre autre d'affecter les ressources

couche OSI		exemples de protocoles utilisés dans le réseau local domestique
numéro	**nom**	
7	Application	SIP
6	Présentation	SSL
5	Session	RTSP
4	Transport	UDP, TCP, RTP
3	Réseau	IP
2	Liaison	Ethernet, Wifi
1	Physique	100baseT, 802.11

TABLE 2.1 – Modèle OSI

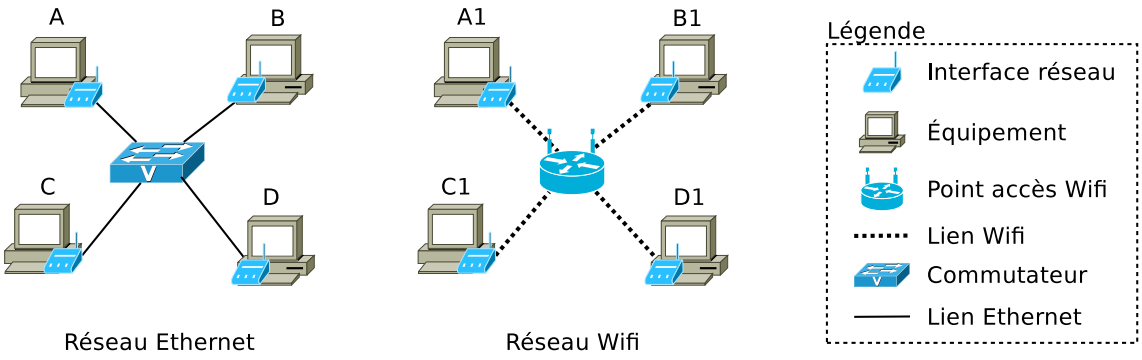


FIGURE 2.1 – Topologie de réseaux locaux en étoile

de communication aux applications. Les équipements se connectent au réseau local via une interface réseau et un protocole de la couche 2. Les réseaux locaux actuels utilisent les protocoles de couche 2 Ethernet pour les liaisons filaires et Wifi pour les liaisons sans fil.

Topologie La topologie du réseau local définit comment les équipements sont interconnectés physiquement. Les équipements communiquent par l'intermédiaire de protocoles de couche 2 du modèle OSI. Il existe différentes topologies qui ne sont pas détaillées ici car les réseaux locaux considérés par cette thèse utilisent une topologie en étoile. Dans cette topologie, tous les équipements sont connectés à un équipement réseau central via un lien, comme le montre la Figure 2.1. Si A veut envoyer des données à B, il les envoie au commutateur qui les transmet à B. Un commutateur permet d'interconnecter des équipements utilisant le protocole Ethernet. Pour le protocole Wifi, un point d'accès est utilisé. Si A1 veut envoyer des données à B1, il les envoie au point d'accès qui les transmet à B1.

Capacité Chaque ressource réseau est associée à une capacité :

- Pour un lien, la capacité définit la bande passante maximale que le lien peut fournir.
- Pour une interface réseau d'un équipement, la capacité définit la bande passante maximale d'envoi ou de réception. On parle de capacité d'émission et de capacité de réception.

protocole	standard	capacité d'un lien	capacité d'une interface	
			théorique	réelle
Ethernet	100 Base T	100 Mbit/s	100 Mbit/s	100 Mbit/s
	1000 Base T	1000 Mbit/s	1000 Mbit/s	1000 Mbit/s
Wifi	802.11a	54 Mbit/s	54 Mbit/s	27 Mbit/s
	802.11b	11 Mbit/s	11 Mbit/s	6 Mbit/s
	802.11g	54 Mbit/s	54 Mbit/s	25 Mbit/s
	802.11n	300 Mbit/s	300 Mbit/s	100 Mbit/s

TABLE 2.2 – Capacités des standards des protocoles de couche 2

– Une bande passante, est exprimée en kilobits par seconde ($kbits/s$) ou en megabits par seconde ($Mbits/s$).

Il existe plusieurs standards pour chaque protocole de couche 2. À chaque standard est associée une capacité.

Ethernet est un protocole *full-duplex*, c'est-à-dire qu'il est possible d'envoyer et de recevoir des données en même temps sur un lien. Sur ce type de lien la capacité est donc fournie pour chaque sens. Par exemple, la capacité du lien $A - commutateur$ dans la Figure 2.1 est de 100 Mbits/s dans le sens $A \rightarrow commutateur$ et de 100 Mbits/s dans le sens $commutateur \rightarrow A$. Le protocole Wifi est lui half-duplex, un seul équipement peut émettre à la fois. La capacité d'un lien Wifi est donnée pour toutes les transmissions. En effet, pour le Wifi il n'y a qu'un seul média de communication. Le Tableau 2.2 donne les capacités des standards utilisés actuellement pour les réseaux Ethernet et Wifi. Pour les interfaces réseau Wifi, la capacité réelle est différente de la capacité théorique. Les valeurs données dans le tableau sont une estimation de la capacité d'émission réelle attendue pour un standard Wifi.

Normes de qualité de service réseau Pour traiter en priorité certains paquets réseaux, des normes de qualité de service réseau (normes de QdS réseau) ont été définies. Ces normes se basent sur le marquage et la priorisation des paquets.

– Le marquage consiste à modifier un champ de l'en-tête d'un paquet, pour l'affecter à une classe de trafic.
– La priorisation consiste à traiter en priorité les paquets des classes de trafic les plus prioritaires.

Au niveau de la couche 2 du modèle OSI, il existe deux normes de QdS réseau. 802.1p est la norme pour le protocole Ethernet et 802.11e [39, 70] est la norme pour le protocole Wifi. Le tableau 2.3 donne les classes de trafic définies pour chaque norme.

L'utilisation des mécanismes de QdS réseau suppose que l'interface réseau émettant les paquets et le commutateur (ou le point d'accès Wifi) supportent la même norme de QdS réseau.

2.1.1.2 Réseau local domestique

Le réseau local domestique contient l'ensemble des équipements domestiques et des liens au sein de la maison d'un utilisateur. Par la suite, nous utilisons le terme équipement pour désigner un équipement domestique.

Prio	**classe**	**type**
max	7	contrôle
	6	contrôle inter-réseau
	5	voix
	4	vidéo
	3	applications critiques
	2	excellent effort
	1	best effort
min	0	background

(a) 802.1p

Prio	**classe**	**type**
max	AC_VO	voix
	AC_VI	vidéo
	AC_BE	best effort
min	AC_BK	background

(b) 802.11e

TABLE 2.3 – Classes de trafic pour les normes de QdS réseau

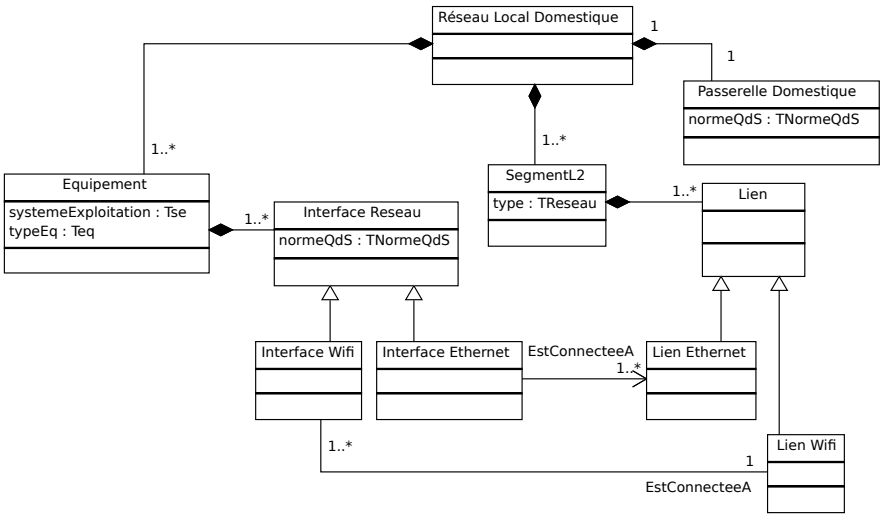


FIGURE 2.2 – Modèle du réseau local domestique

La Figure 2.2 présente le modèle du réseau local domestique en suivant le formalisme d'un diagramme de classe UML. Le réseau local domestique se compose de la passerelle domestique, de segments L2 et d'équipements. Nous détaillons ces différents éléments, puis nous présentons comment les équipements s'interconnectent, au niveau applicatif.

Passerelle domestique La passerelle domestique est un élément central qui fait office de commutateur (pour les liens Ethernet), de point d'accès Wifi, et de routeur (pour accéder au réseau extérieur à la maison). La passerelle domestique interconnecte les segments L2 de types différents (Ethernet et Wifi). Pour les opérateurs de télécommunication c'est un élément primordial. Il existe par exemple le consortium *Home Gateway Initiative* (HGI) [1] qui regroupe différents opérateurs de télécommunication, vendeurs de matériel et de logiciel. Ce consortium a pour objectif de définir les besoins et les spécifications des passerelles domestiques.

1. `http://www.homegatewayinitiative.org/`

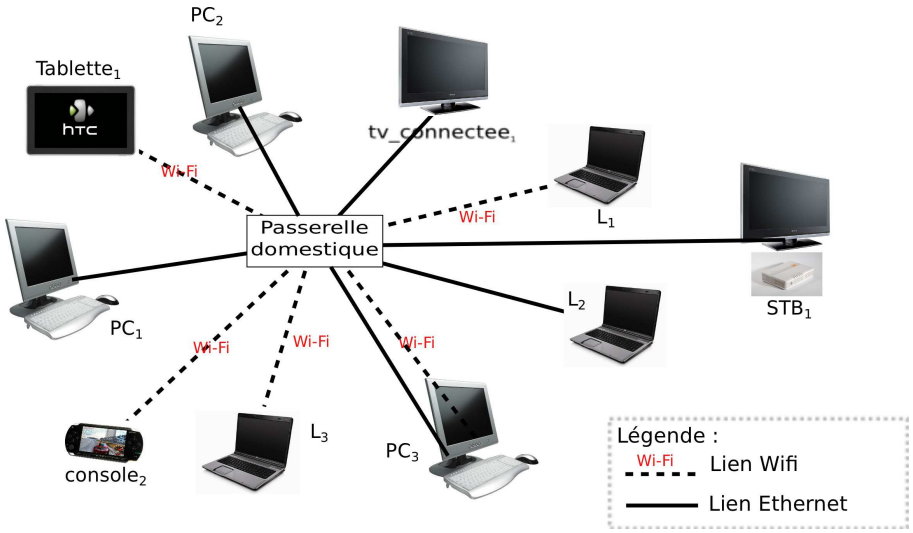


FIGURE 2.3 – Exemple de réseau local domestique

Enfin la passerelle domestique supporte ou non une norme de QdS réseau. L'attribut *normeQdS* indique la norme de QdS réseau supportée ou l'absence de support de QdS réseau.

Segment L2 Pour simplifier les problématiques d'adressage, tous les équipements du réseau local domestique sont dans le même réseau IP (couche 3 du modèle OSI). On parle de segments segment L2 pour définir les liens de même type. On retrouve un segment L2 par type de liens dans le réseau local domestique. L'attribut *type* contient le type du lien. Par exemple un segment L2, de type Ethernet, contient tous les liens Ethernet, et un segment L2, de type Wifi, contient le lien Wifi.

Équipements Un équipement se connecte au réseau local domestique, via un segment L2, par une interface réseau. Un équipement a une ou plusieurs interfaces réseau. Une interface Ethernet est connectée à un lien Ethernet, une interface Wifi est connectée à un lien Wifi. Une interface supporte ou non une norme de QdS réseau. L'attribut *normeQdS* indique la norme de QdS réseau supportée ou l'absence de support de QdS réseau.

Les équipements du réseau local domestique se sont d'abord limités aux équipements fournis par l'opérateur : la passerelle domestique et le décodeur TV ainsi qu'un ou deux ordinateurs appartenant à l'utilisateur. On observe ces dernières années une forte tendance à l'augmentation du nombre et de la diversité des équipements présents dans le réseau local domestique [63, 83, 99]. Ce dernier se caractérise aujourd'hui par une forte hétérogénéité des liens et des équipements.

La Figure 2.3 donne un exemple de réseau local domestique, contenant des équipements de l'opérateur et des équipements utilisateurs.

Nous présentons les équipements que l'on retrouve typiquement dans les réseaux locaux domestiques, en commençant par ceux fournis par l'opérateur.

Équipements de l'opérateur En plus de la passerelle domestique, les opérateurs fournissent à leurs clients un ou plusieurs décodeurs TV (*Set-Top-Box* ou STB).

Ces STBs permettent aux utilisateurs de regarder la télévision ou des contenus multimédia à la demande. Ces contenus sont acheminés par l'intermédiaire du réseau de l'opérateur et de la passerelle domestique jusqu'à une STB. Dans l'avenir, ces STB serviront à visionner des contenus utilisateurs provenant d'autres équipements du réseau local domestique. Un réseau local domestique contient une ou plusieurs STB, reliées à ce réseau via une interface Wifi ou Ethernet.

Équipements de l'utilisateur En plus des équipements de l'opérateur, le réseau local domestique comprend des équipements appartenant à l'utilisateur. Ces derniers sont de plusieurs types (*typeEq* sur le modèle) : PCs, ordinateurs portables, téléphones mobiles, consoles de jeux, télévisions connectées et tablettes (liste non exhaustive). Plusieurs équipements de même type peuvent être présents simultanément.

Interconnexion des équipements au niveau applicatif Les équipements du réseau local domestique sont physiquement connectés à l'aide de liens et d'interfaces réseau. Ces équipements hétérogènes provenant de fabriquants différents, communiquent à l'aide de standards d'interconnexion. Ces standards s'intègrent dans les couches 5 à 7 sept du modèle OSI. Ces standards permettent la détection de services et l'interconnexion des équipements au niveau applicatif. Ces standards résolvent aussi les problématiques d'adressage au sein du réseau local domestique en les masquant.

Parmi les standards les plus avancés à l'heure actuelle on retrouve le standard UPnP (*Universal Plug and Play*) [31,67,119], le standard DPWS (*Devices Profile for Web Services*) [21], et le standard DLNA (*Digital Living Network Alliance*) [32]. Par exemple, le profil standard UPnP AV [124] définit les profils *"media server"* et *"media renderer"* et leur mode d'interaction. Ainsi un équipement *"media server"* et un équipement *"media renderer"* sont interconnectables pour démarrer une application de diffusion de contenus multimédia.

2.1.2 Applications de diffusion de contenus multimédia

Les applications de diffusion de contenus multimédia suivent un modèle client-serveur et sont démarrées à la demande de l'utilisateur. Ces demandes ne sont pas connues à l'avance et tout équipement du réseau local domestique peut faire office de serveur et/ou de client.

Nous détaillons d'abord les caractéristiques des contenus multimédia utilisés. Ensuite, nous présentons le fonctionnement des applications de diffusion de contenus multimédia.

2.1.2.1 Contenu multimédia

Un contenu multimédia se compose d'un flux vidéo et d'un ou plusieurs flux audio. Ces flux sont compressés car les capacités de stockage des périphériques, les capacités d'envoi et de réception des interfaces réseau et les capacités des liens sont limitées. Un flux compressé est communément appelé flux encodé. Les principaux standards d'encodage utilisés pour les flux vidéo sont *h.264* [89] et *mpeg4-part2* [57]. Les principaux standards d'encodage utilisés pour les flux audio sont *mp3* [52] et

aac [53]. Pour former un contenu multimédia, les flux audio et vidéo sont intégrés dans un flux conteneur. Les principaux flux conteneurs sont avi, mp4 et mov (liste non exhaustive).

Un fois encodé, le contenu d'un flux est appelé flot binaire (*bitstream* en anglais). Le flot binaire est utilisé par l'application de diffusion de contenus multimédia et influe sur les quantités de ressources requises par l'application. Nous détaillons d'abord les flots binaires. Ensuite, nous présentons les paramètres utilisés lors de l'encodage d'un flux pour faire varier la complexité du flot binaire. Cette variation influence directement les quantités de ressources requises par une application.

Flot binaire Un flot binaire est caractérisé par un débit, exprimé en kilobits par secondes (kbits/s). Le processus d'encodage d'un flux est un compromis entre le débit du flot binaire (le plus petit possible pour minimiser la quantité de données à stocker et envoyer) et la complexité du flot binaire (plus le flot binaire est complexe, plus la quantité de ressources nécessaire au décodage sera grande). Ce compromis est différent pour chaque standard d'encodage et est modifié par les paramètres d'encodage.

Le débit du flot binaire est constant ou variable. Un flot binaire à débit constant (*Constant Bit Rate*, CBR en anglais) est plus facile à caractériser pour la transmission mais la qualité est réduite à certains endroits dans le flux (nécessitant plus d'information). À l'inverse, le débit du flot binaire n'est pas pleinement utilisé à d'autres endroits (nécessitant moins d'information). Au contraire un flot binaire à débit variable (*Variable Bit Rate*, VBR en anglais) maintient une qualité constante mais est plus difficile à caractériser pour la transmission. Dans les standards d'encodage actuels, le débit du flot binaire est généralement variable.

Paramètres d'encodage Lors de l'encodage d'un flux, plusieurs paramètres sont utilisés par l'encodeur.

Tout d'abord, des paramètres tels que la résolution (taille des images), la fréquence d'images ou la fréquence d'échantillonnage sont utilisés.

Certains paramètres sont liés à un standard. Par exemple, le standard *h.264* met à disposition de l'encodeur plusieurs techniques de compressions, dont la technique *Context-Adaptive Binary Arithmetic Coding* (*CABAC*) [76]. L'encodeur est paramétré pour utiliser ou non une technique de compression. D'autres paramètres sont liés à un encodeur. Par exemple, l'encodeur `x264` [2] définit un paramètre reflétant la qualité du flux encodé. Ce paramètre, appelé *crf*, prend une valeur entre 0 et 51, 0 donnant la meilleure qualité.

Certains paramètres limitent le débit maximal du flot binaire ou sa variation.

Le nombre de paramètres étant important, lors de l'encodage d'un flux, un profil et un niveau sont utilisés pour paramétrer l'encodeur.

- Un profil définit les fonctionnalités disponibles lors de l'encodage d'un flux. Par exemple, dans le profil de base (baseline) du standard *h.264* la technique de compression CABAC n'est pas disponible.

2. x264 est un encodeur *h.264* libre, disponible sur `http://www.videolan.org/developers/x264.html`

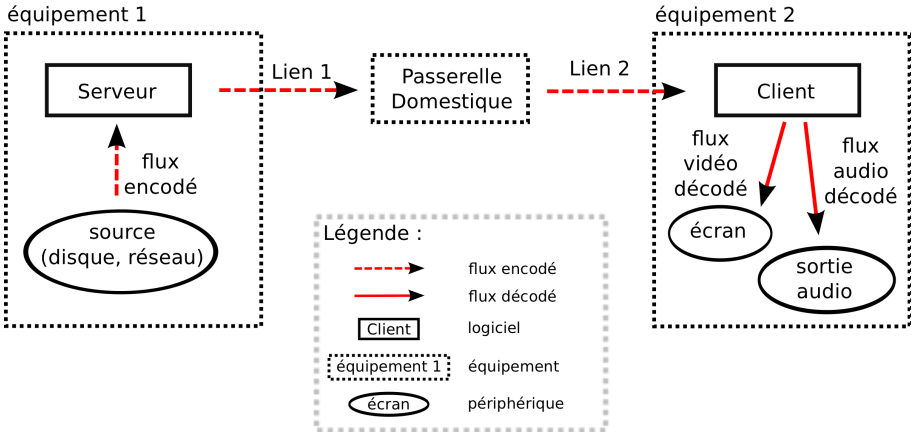


FIGURE 2.4 – Schéma d'une application de diffusion de contenus multimédia

– Un niveau définit des valeurs limites sur des caractéristiques du flot binaire. Par exemple, le niveau 3 du standard *h.264* limite le débit flot binaire à 10000 $kbits/s$ pour le profil de base.

D'autres paramètres que ceux décrits ici peuvent être utilisés et il n'existe pas à notre connaissance de liste exhaustive de tous les paramètres possibles, tout standards confondus. Pour avoir un ordre de grandeur du nombre de paramètres, l'encodeur `x264` utilise une centaine de paramètres. Par défaut, `x264` fournit 10 combinaisons de paramètres prédéfinies. Parmi les paramètres de `x264`, on retrouve le choix du profil et du niveau d'encodage. Le standard *h.264* définit 17 profils et 5 niveaux. Chaque niveau est découpé en 2 ou 3 sous-niveaux. Enfin `ffmpeg`[3] définit 29 tailles d'images prédéfinies. Pour un standard d'encodage, il existe donc plusieurs centaines de paramètres possibles lors de l'encodage.

2.1.2.2 Description des applications de diffusion de contenus multimédia

Dans cette thèse, une application de diffusion de contenus multimédia se compose d'un serveur et d'un client. Lorsque l'utilisateur demande à démarrer une application, il précise l'équipement serveur, l'équipement client et le contenu multimédia. Les interfaces réseau, les liens et les protocoles réseaux utilisés sont fournis par l'utilisateur ou inférés par le système. La Figure 2.4 donne un exemple d'application de diffusion de contenus multimédia.

Nous commençons par présenter les protocoles permettant de faire de la diffusion de contenu multimédia. Ensuite, nous détaillons comment cette diffusion est réalisée entre le serveur et le client. Enfin nous donnons les logiciels qui sont utilisés pour faire de la diffusion.

Protocoles de diffusion Pour faire de la diffusion de contenu multimédia, le protocole RTSP (*Real Time Streaming Protocol*) [45] de la couche 5 du modèle OSI est communément utilisé. Ce protocole s'appuie sur un protocole de transport (couche 4 du modèle OSI), notamment RTP (*Real-time Transport Protocol*) [108], UDP et TCP. Le choix du protocole de transport se fait lors du démarrage du

3. `ffmpeg` est un encodeur libre, supportant un grand nombre de standards d'encodage. `ffmpeg` est disponible à `http://www.ffmpeg.org/`

serveur. Le client se connecte ensuite au serveur en utilisant le protocole session RTSP. Deux modes de diffusions sont disponibles. Dans le premier (mode *pull*), le serveur attend une demande d'un client pour envoyer le contenu multimédia. C'est le mode qui est utilisé lors de la diffusion de contenu à la demande via le réseau de l'opérateur. Dans le deuxième mode de diffusion (mode *push*), le serveur commence à envoyer le contenu multimédia dès qu'il est démarré. Lors de la connexion d'un client, la diffusion continue sans interruption. Comme les applications considérées par cette thèse n'ont qu'un seul client, il n'y a pas de différence entre le mode *pull* et le mode *push* après le début de la diffusion. Dans les deux cas le flux conteneur est envoyé du serveur vers le client.

Processus de diffusion Lors de la diffusion du contenu, le serveur lit le flux conteneur sur un de ses périphériques (disque dur, clé USB, réseau extérieur). Ce flux est ensuite envoyé à un client qui décode les flux vidéo et audio, affiche le premier à l'écran et envoie le second au périphérique audio (cf. Figure 2.4). La diffusion se fait en continue, le client décode et affiche les flux en temps réel. Pour compenser une variation du temps de transmission des données sur le réseau, le client commence par stocker en mémoire le flux encodé. Le démarrage du décodage et de l'affichage sont ainsi décalés de quelques secondes. Le temps pendant lequel le client stocke, sans afficher, est configuré à l'initialisation.

Logiciels utilisés Il existe de nombreux logiciels permettant d'exécuter le serveur et le client sur les différents équipements du réseau local domestique. Par exemple il existe des logiciels libres : `vlc` [4] et `ffserver` [5] pour le serveur, client `gstreamer` [6], `mplayer` [7] ou `ffplay` [8] pour le client.

Après cette présentation des contenus et des applications de diffusion de contenus multimédia, nous détaillons les notions de qualité de service et de quantité de ressources.

2.1.3 Qualité de service et quantité de ressources

Il existe de nombreuses visions de la Qualité de Service, appelée QdS (*Quality of Service*, *QoS* en anglais). L'objectif de la QdS est de satisfaire l'expérience utilisateur.

Parmi les nombreuses définitions de la qualité de service, celle de l'*International Telecommunication Union* (ITU-T), donnée dans [54], nous semble bien adaptée à notre contexte :

> *"QoS has been defined as a collective effect of service and performances that determine the degre of satisfaction of the service."*

Cette définition présente la QdS comme une fonction du service et des performances de l'exécution du service. Cette fonction détermine le niveau de satisfaction du

4. `http://www.videolan.org/vlc/`
5. `http://www.ffmpeg.org/ffserver-doc.html`
6. `http://www.gstreamer.net/`
7. `http://www.mplayerhq.hu/`
8. `http://www.ffmpeg.org/ffplay-doc.html`

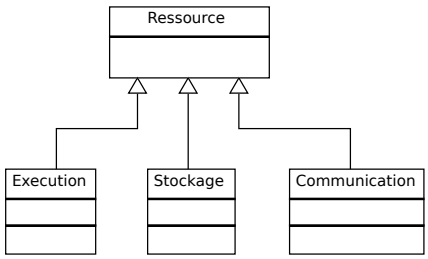


FIGURE 2.5 – Types de ressources

service. Dans notre contexte, le service est la diffusion de contenu multimédia. Pour fournir le niveau de satisfaction attendu par l'utilisateur, trois couches de QdS sont définies :

- couche utilisateur : Le niveau de satisfaction, appelé niveau de QdS utilisateur [121], est une perception subjective de l'utilisateur. Par exemple le niveau de QdS utilisateur est bon, passable ou mauvais.
- couche application : Pour fournir le niveau de QdS utilisateur attendu, les applications doivent respecter un ensemble de contraintes de QdS objectives [91]. Ces contraintes de QdS sont des bornes sur des valeurs quantifiables. Un exemple typique de contrainte de QdS est une fréquence d'images de 25 $images/s$ pour le décodage d'un flux vidéo.
- couche ressources : Pour satisfaire les contraintes de QdS, une application requiert un ensemble de Quantités de Ressources, appelées par la suite QdR. Par exemple pour décoder un flux f à 25 $images/s$ une application requiert 100 Mo de mémoire RAM.

Ainsi, pour satisfaire la QdS utilisateur, via une QdS applicative, une application requiert un ensemble de QdR sur les différentes ressources utilisées.

Nous commençons par définir les types de ressources qui sont classiquement considérés puis comment les ressources sont intégrées dans les équipements. Nous présentons ensuite comment les applications de diffusion de contenus multimédia utilisent les ressources des équipements. Par la suite, nous nous intéressons plus particulièrement à la définition des quantités de ressources (QdR). Les QdR sont fournies par les ressources des équipements et requises par les applications.

2.1.3.1 Types de ressources

Classiquement trois types de ressources sont définis et considérés : les ressources d'exécution, les ressources de stockage et les ressources de communication [96, 104, 105, 111], modélisés dans la Figure 2.5 par un diagramme de classe UML.

Ces ressources, dites physiques, sont gérées par le système d'exploitation. Les application n'accèdent pas directement aux ressources physiques mais aux abstractions faites par le système d'exploitation, pour gérer les ressources physiques. Les ressources d'exécution sont accessibles via un ordonnanceur, les ressources de stockage sont accessibles via un gestionnaire de mémoire et les ressources de communication sont accessibles via le pilote de l'interface réseau. Nous verrons dans la suite de ce chapitre comment les quantités de ressources sont exprimées en fonction du type de ressource.

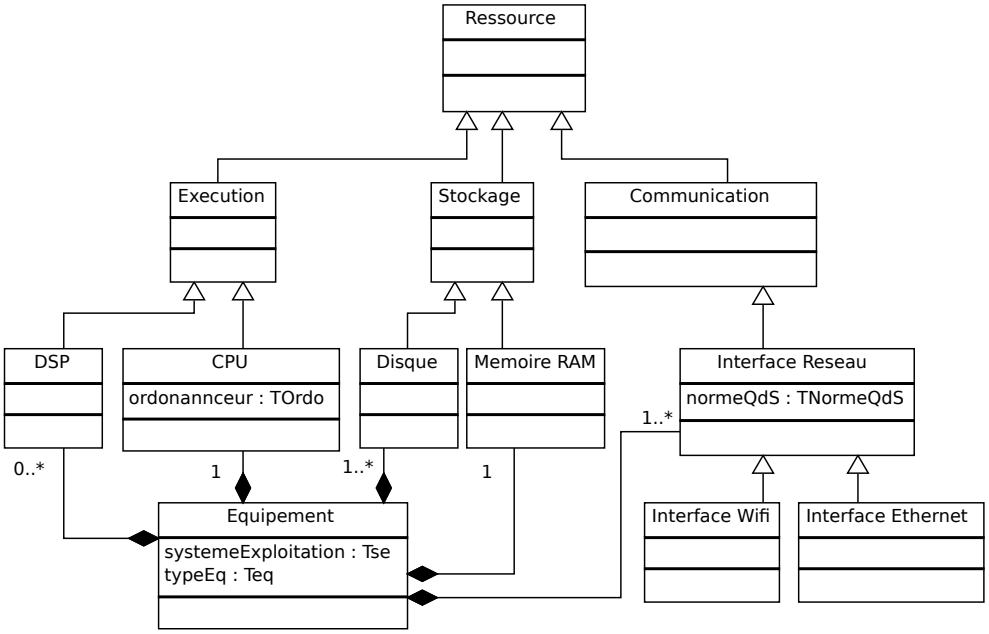


FIGURE 2.6 – Modèle des équipements du réseau local domestique considéré dans cette thèse

Dans la suite de ce document, nous utilisons le terme ressource pour désigner l'abstraction faite par le système d'exploitation de la ressource physique. Ces ressources sont regroupées et gérées dans des équipements.

2.1.3.2 Modèle des équipements

Un équipement est caractérisé par un ensemble de ressources, partagées par toutes les applications s'exécutant sur l'équipement. La Figure 2.6 présente le modèle des équipements, considéré dans cette thèse, à l'aide d'un diagramme de classe UML.

Dans notre étude, un équipement possède une seule ressource processeur (communément appelé *Central Processing Unit* (CPU)). Dans le cas des équipements ayant plusieurs cœurs symétriques (*Symetric Multi Processor* (SMP)), la ressource CPU est une abstraction englobant tous les cœurs. L'affectation des applications à un cœur est laissée au système d'exploitation et n'est pas considérée dans nos travaux. Le CPU d'un équipement est partagé entre les différentes applications par un ordonnanceur. Un équipement possède aussi une ou plusieurs ressources disques durs (`Disque` sur la figure), une ressource mémoire RAM et une ou plusieurs ressources interfaces réseau. Un équipement peut aussi posséder une ou plusieurs ressources *Digital Signal Processor* (DSP, processeur de signal numérique en français). Le rôle du DSP est ici de décoder des flux vidéo ou des flux audio.

Les ressources d'un équipement sont gérées par un système d'exploitation (*systemeExploitation* sur la figure). Les systèmes d'exploitation utilisés par les équipements du réseau local domestique sont des systèmes d'exploitation généraux, tels que Linux, Windows, Mac OS ou les versions pour les mobiles (Android, IOS, WindowsCE). Linux est très répandu parmi les équipements du réseau local domestique. De plus, il offre un accès aux sources du système, et est modifiable plus

facilement. Pour ces raisons, Linux est très utilisé dans la littérature. Enfin Linux est un système d'exploitation général et est représentatif des systèmes d'exploitation utilisés sur les équipements actuels. Ainsi nous nous sommes intéressés plus particulièrement à ce système d'exploitation.

2.1.3.3 Utilisation des ressources

Au travers des équipements, une application multimédia utilise une ou plusieurs ressources d'exécution. Le CPU est toujours utilisé sur l'équipement serveur et sur l'équipement client. Il est possible d'effectuer le décodage d'un flux, sur le client à l'aide d'un DSP. Lors du démarrage d'une application de diffusion de contenus multimédia, le client est configuré pour effectuer le décodage de chaque flux via le CPU ou via un DSP.

Une application multimédia utilise aussi des ressources de stockage. Le disque dur, ou autre périphérique de stockage, est utilisé par le serveur pour stocker le flux conteneur. Ensuite, la mémoire RAM est utilisée, par le serveur et par le client, lors de la diffusion. Si la mémoire RAM n'est pas suffisante, le système d'exploitation, via le gestionnaire de mémoire, utilise le disque comme mémoire tampon (*swap* en anglais). Le système d'exploitation permet ainsi aux applications de continuer à s'exécuter même s'il n'y pas assez de mémoire RAM pour satisfaire toutes les applications. Le temps d'accès à la mémoire tampon est beaucoup plus important que le temps d'accès à la mémoire RAM. Pour les applications de diffusion de contenus multimédia, utiliser de la mémoire tampon entraîne une dégradation de la QdS.

Enfin, les applications de diffusion de contenus multimédia, considérées dans cette thèse, sont distribuées et utilisent des ressources de communication :

- Interface serveur : le serveur envoie le flux via son interface réseau
- Liens : les liens font transiter les flux
- Interface client : le client reçoit le flux via son interface réseau.

2.1.3.4 Quantités de ressources fournies

Chaque ressource fournit des quantités de ressources aux applications l'utilisant. Chaque ressource est associée à une ou plusieurs caractéristiques de QdS [91]. La Figure 2.7 donne le modèle de QdS lié aux ressources, considéré dans cette thèse, sous la forme d'un diagramme de classe UML.

Toutes les ressources sont associées à une `capacité` (caractéristique de QdS), définissant la quantité maximale que la ressource est capable de fournir. Pour une ressource CPU, la `capacité` est la somme des capacités de tous les cœurs. Il existe différentes façons d'exprimer la capacité d'une ressource. qui ne sont pas détaillées ici [91]. Les ressources sont aussi associées à une `QdRdisponible` (caractéristique de QdS), définissant la quantité disponible, à un instant donné, sur la ressource.

Un lien réseau est une ressource de communication, partagée par les flux transitant par le lien. En plus de sa capacité, cette ressource possède des caractéristiques de QdS réseau : le délai, la gigue et le taux de perte. Le délai est le temps de transmission des données d'un bout à l'autre du lien. La gigue est la variation du délai. Enfin, le taux de perte donne le nombre de paquets perdus sur le lien. Il existe plusieurs façons d'exprimer le taux de perte. Pour garantir la QdS applicative il est important de garantir que le taux de perte du lien est en dessous d'une valeur

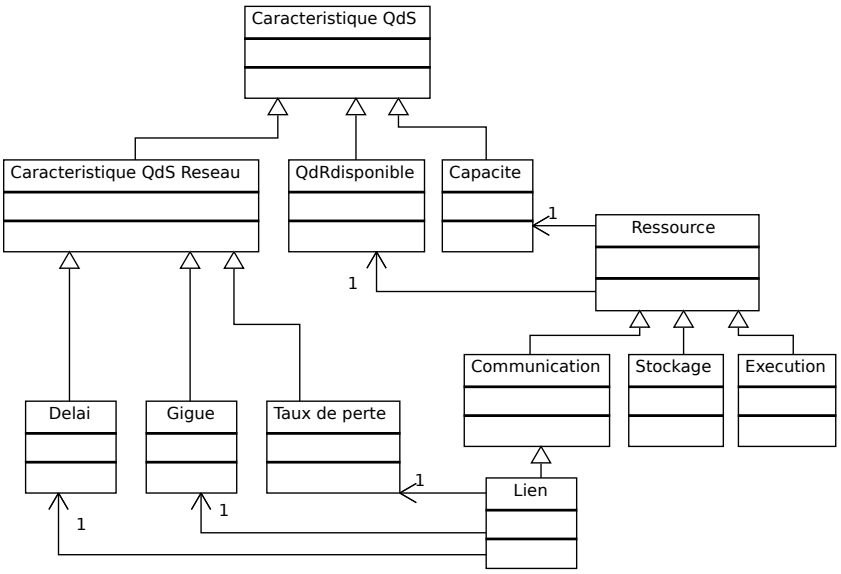


FIGURE 2.7 – Modèle des ressources

donnée pour l'application. La façon d'exprimer le taux de perte n'est qu'un moyen d'exprimer cette contrainte. Dans cette thèse, il est exprimé en pourcentage de paquets perdus sur une seconde. Par exemple un taux de perte de 1% signifie qu'un paquet sur 100 est perdu toutes les secondes en moyenne.

2.1.3.5 Quantités de ressources requises

Pour satisfaire les contraintes de QdS de la couche application, une application requiert, sur chaque ressource, une quantité de ressources. La QdR requise s'exprime à l'aide de valeurs et de contraintes.

Valeur de QdR requise Les QdR utilisées par une application de diffusion de contenus multimédia sur une ressource varient au cours du temps. Pour simplifier la gestion des QdR, une valeur représentative est souvent utilisée pour abstraire l'ensemble des QdR requises sur toute la durée de l'application [98, 100, 122]. Classiquement deux valeurs sont utilisées : le maximum de QdR requise, afin de garantir la QdS de l'application ou la moyenne de QdR requise, afin d'optimiser l'utilisation de la ressource.

Unités et contraintes des QdR requises Nous présentons maintenant comment les QdR sont exprimées et quelles sont les contraintes pour chaque type de ressources. Ensuite nous détaillons les paramètres qui influencent les QdR requises.

Ressources d'exécution : Le client doit afficher le flux vidéo selon sa fréquence d'images. Les applications, dédiées à la diffusion de contenu multimédia, sont dit périodiques. Classiquement, une application périodique est vue comme un ensemble de tâches périodiques. La quantité de ressources d'exécution requise par une tâche τ_i est exprimée par (C_i, T_i). C_i donne la quantité de ressources requise, T_i

est la période de la tâche, exprimée en millisecondes (ms). Autrement dit, la tâche τ_i requiert C_i unités de ressource d'exécution, toutes les T_i unités de temps.

C_i s'exprime classiquement en nombre d'instructions ou en temps CPU. Il est aussi possible d'exprimer C_i en pourcentage de la capacité de la ressource. Dans ce cas, la tâche requiert C_i pourcent du processeur toutes les périodes T_i. Par exemple, si la QdR requise d'une tâche est de $(25\%, 40ms)$, alors la tâche doit avoir le CPU pendant 10 millisecondes toutes les 40 millisecondes.

Ressources de stockage : Les quantités de ressources requises sur les ressources de stockage s'expriment par une quantité de mémoire en kilo octets (Ko) ou en mega octets (Mo). Il n'y a pas de paramétrage spécifique à effectuer sur la gestion des ressources de stockage.

Ressources de communication : Les quantités de ressources requises sur les ressources de communication, s'expriment par une bande passante en kbits/s ou en Mbits/s.

Sur les liens, des contraintes sur les QdS réseau sont exprimées. Ces contraintes sont des bornes supérieures sur le débit, la gigue et le taux de perte du lien.

Paramètres influant sur les QdR requises : La QdR requise sur une ressource d'exécution ou de stockage est fonction :

- des flux utilisés : le contenus du flux, exprimé sous la forme d'un flot binaire dépendant des paramètres d'encodage
- des logiciels utilisés (`vlc` ou `gstreamer` par exemple)
- de la ressource elle-même
- des autres ressources utilisées (utilisation ou non d'un DSP)
- des contraintes de QdS.

La QdR requise sur une ressource de communication est fonction :

- des flux utilisés (contenus, flots binaires, paramètres d'encodage)

Le tableau 2.4a décrit un scénario d'exécution d'une application de diffusion de contenus multimédia. Pour ce scénario, le flux conteneur contient uniquement un flux vidéo. Les colonnes 5 et 6 donnent les équipements utilisés. Les colonnes 7 et 8 donnent les ressources utilisées sur les équipements. Les contraintes de QdS liées au flux sont données dans la colonne 4 (paramètres du flux). Le Tableau 2.4b donne les QdR requises et les contraintes de QdS réseau pour ce scénario, sur chaque ressource, pour une QdS jugée acceptable par l'utilisateur. Le passage d'une QdS acceptable au niveau utilisateur à une QdR est hors du cadre de cette thèse. Les QdR requises représentent les QdR maximales requises par l'application.

Hétérogénéité des flux et des équipements Nous avons vu que le réseau local domestique contient des équipements hétérogènes (section 2.1.1.2). De plus, pour un contenu multimédia, plusieurs standards d'encodage et de nombreux paramètres sont utilisables (section 2.1.2.1).

Nous avons présenté précédemment les paramètres influant sur les QdR requises par une application de diffusion de contenus multimédia. Dans le contexte hétérogène du réseau local domestique, le nombre de QdR requises à connaître est très important. L'équation (2.1), donne la combinatoire du nombre de valeurs de QdR requises

S	Flux	encodeur	paramètres	équipements		ressources d'exécution	
				serveur	client	serveur	client
1	big buck bunny	*h.264*	profil : base taille images : CIF fréq. im. : 25 im/s	PC_1	$Tablette_1$	CPU	CPU

(a) Scénario

S	Serveur		client		réseau	
	CPU	mémoire	CPU	mémoire	QdR	contrainte QdS
1	(2.5%,40ms)	10 Mo	(50%,40ms)	50 Mo	12 Mbits/s	$delai < 200ms$ $gigue < 400ms$ $taux\ de\ perte$ $< 0.20\%$

(b) QdR requises

TABLE 2.4 – Exemple d'un scénario et des QdR requises associées

pour le CPU, sur l'équipement exécutant le client. Pour un contenu multimédia, composé de 2 flux, dans un réseau local domestique contenant 10 équipements, en considérant 100 paramètres d'encodage pour chaque flux, 6 logiciels clients possibles, 4 utilisations des ressources sur chaque équipement (avec ou sans DSP pour le flux audio et le flux vidéo) et 5 contraintes de QdS, 240000 valeurs de QdR requises doivent être connues. À ce nombre il faut rajouter les QdR des autres ressources et les QdR requises pour les autres contenus multimédia.

$$2 * 10 * 100 * 6 * 4 * 5 = 240000 \tag{2.1}$$

Pour un réseau local domestique contenant 100 contenus multimédia, plusieurs millions de QdR requises doivent être connues. Si les opérateurs de télécommunication veulent fournir un *framework* de gestion des QdR à leurs utilisateurs, alors le nombre de données à traiter chez l'opérateur devient très important.

2.1.4 Discussion

Dans la première partie de ce chapitre, nous avons présenté le réseau local domestique, les applications de diffusion de contenus multimédia et les notions relatives aux quantités de ressources. La Figure 2.8 regroupe les différents modèles présentés au cours de ce chapitre pour donner le modèle du réseau local domestique considéré dans cette thèse.

Nous résumons les informations présentées dans ce chapitre sur le réseau local domestique, les applications multimédia et les quantités de ressources.

Réseau local domestique Le réseau local domestique se compose :
- de la passerelle domestique
- d'équipements
- de segments L2

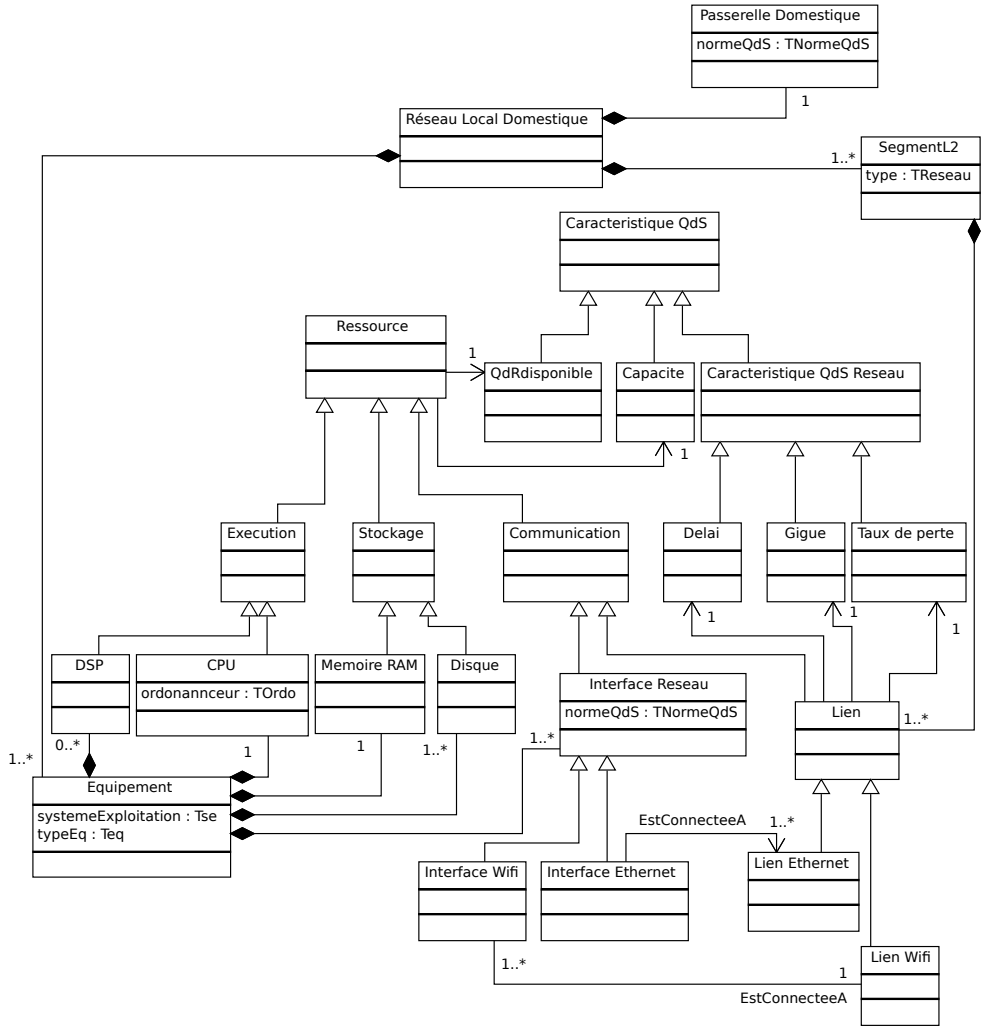


FIGURE 2.8 – Modèle du réseau local domestique

Les équipements sont connectés grâce à leurs interfaces réseau et des liens. Les liens de même type sont connectés selon une topologie en étoile et forment des segments L2, de type Ethernet ou Wifi.

Les équipements sont composés d'un ensemble de ressources, gérées par un système d'exploitation dit général. Dans cette thèse, nous nous sommes focalisés sur le système d'exploitation Linux.

Le réseau local domestique rassemble de nombreux acteurs (opérateurs de télécommunication, fournisseurs d'équipements et fournisseurs d'applications). Aucun des acteurs ne contrôle tous les équipements et toutes les applications du réseau local domestique. En outre, même s'il est possible de modifier un équipement, le coût de développement induit serait très vite prohibitif. Il est donc primordial de ne pas modifier les équipements existants.

Application multimédia Les applications de diffusion de contenus multimédia se composent d'un serveur qui envoie un contenu multimédia à un client. Les contenus multimédia sont pré-encodés avec différents standards. Lors de l'encodage, de nombreux paramètres sont à la disposition de l'encodeur. Les standards d'encodage actuels utilisent des flots binaires à débit variable (*Variable Bit Rate* (VBR)). Tous les équipements du réseau local domestique sont susceptibles d'exécuter une application de diffusion de contenus multimédia.

Il existe à ce jour de nombreux logiciels permettant de faire de la diffusion de contenus multimédia. Certains sont libres et pourraient être modifiés, mais cela ne va pas sans un certain coût. De plus, d'autres logiciels non libres doivent pouvoir être supportés. En outre, les standards d'interconnexion tels qu'UPNP permettent d'intégrer de nouveaux équipements et de nouveaux logiciels, sans les modifier. Il serait ainsi contre productif de devoir modifier les logiciels pour garantir les quantités de ressources requises.

Il est donc important d'être capable de gérer les quantités de ressources requises par ces logiciels, sans les modifier.

Quantité de ressources D'un point de vue économique, il est primordial de pouvoir satisfaire les attentes de qualité de service (QdS) de l'utilisateur. Nous avons vu que la QdS utilisateur est liée aux ressources utilisées par les applications de diffusion de contenus multimédia. En effet pour fournir le niveau de QdS attendu, une application requiert une quantité de ressources (QdR) pour chacune des ressources utilisées.

Les quantités de ressources d'exécution et de stockage requises, par une application de diffusion de contenus multimédia, dépendent des flux utilisés (contenus, flots binaires, paramètres d'encodage), des logiciels utilisés, de la ressource elle-même, des autres ressources utilisées et des contraintes de QdS. De plus le choix de la ressource d'exécution pour le décodage influe sur les quantités de ressources d'exécution et de stockage requises. Enfin, la quantité de ressources de communication requise dépend des flux utilisés.

Nous avons vu que l'hétérogénéité des équipements et des flux du réseau local domestique produit un grand nombre de valeurs de QdR requises à connaître et à gérer. Cette hétérogénéité complexifie la gestion des QdR, lors de la demande

de démarrage (non connue à l'avance) d'une application de diffusion de contenus multimédia.

Cette présentation du contexte de l'étude a mis en évidence le caractère fortement hétérogène et interconnecté du réseau local domestique. Les utilisateurs du réseau local domestique, souhaitent démarrer des applications de diffusion de contenus multimédia sur n'importe quel équipement. Lors de l'exécution d'une application de diffusion de contenus multimédia, il est primordial de fournir la qualité de service attendue, en garantissant les QdR requises par l'application.

Nous présentons dans la suite de ce chapitre un état de l'art des travaux s'intéressant à la gestion des quantités de ressources. Nous détaillons les solutions existantes puis nous nous intéressons à leur intégration dans le contexte présenté précédemment.

2.2 Gestion des quantités de ressources

Nous avons vu que pour fournir un certain niveau de QdS, une application requiert un ensemble de quantités de ressources. Pour garantir à une application une quantité de ressources, il est possible de passer un contrat, dit contrat de QdS, avec la ressource. Dans [17], 4 niveaux de contrats sont définis :

- niveau de base : les contrats de ce niveau spécifient les fonctions offertes et les entrées/sorties. Ces contrats sont purement fonctionnels
- niveau comportemental : les contrats de ce niveau spécifient des pré et post conditions.
- niveau synchronisation : les contrats de ce niveau spécifient des contraintes de synchronisation sur des objets partagés. Ces contrats sont utilisés pour réifier les dépendances cachées entre les composants telles des données partagées
- niveau qualité de service : les contrats de ce niveau spécifient des contraintes de QdS.

On retrouve la définition de contrat de QdS dans plusieurs travaux [17, 85, 91, 109]. Dans cette thèse, nous nous intéressons à la couche ressource. Les contrats de QdS sont vus au niveau de cette couche et sont appelés contrats de QdR. Un contrat de QdR porte sur la QdR requises sur une ressource et les contraintes liées à cette QdR (contraintes de QdS réseau, périodicité par exemple).

Nous présentons maintenant comment les contrats de QdR sont gérés, par négociation entre la ressource et l'application. Au cours du temps, ces contrats peuvent être renégociés. Enfin, nous présentons la programmation à composant, qui offre un cadre bien adapté pour intégrer les politiques de gestion des contrats de QdR, entre un composant client et un composant gérant l'accès à la ressource.

Négociation de contrat de QdR De nombreux travaux utilisent des contrats de QdR pour garantir les QdR requises par une application [12, 13, 28, 46, 48, 87, 91, 113, 122, 125]. Lorsqu'une application est démarrée, une phase de négociation est réalisée entre l'application et la ressource. Si la négociation réussit, alors un contrat de QdR est établi et la QdR est réservée pour l'application. Sinon, l'application n'est pas démarrée ou est démarrée sans garantie. L'application peut nécessiter différents niveaux de QdR, correspondants à différents niveaux de QdS. Un contrat de QdR

niveau de QdS utilisateur	QdS application (taille image)	QdR requise (RAM en Mo)
BON	1080p	150
MOYEN	720p	100
PASSABLE	352x288	75

TABLE 2.5 – Niveaux de contrats QdS et de QdR pour la ressource mémoire RAM de l'équipement client

porte alors sur un niveau de QdR. Le Tableau 2.5 donne un exemple avec trois niveaux de contrat de QdR, portant sur l'utilisation de la ressource mémoire RAM. Pour la QdS application, seule la contrainte de QdS portant sur la taille d'images est donnée.

Une application utilisant plusieurs ressources, un contrat de QdR est établi pour chaque ressource. Les contrats de QdR d'une même application sont ensuite "liés" pour assurer leur cohérence et l'intégrité du système. Cette "liaison" est faite par une entité centrale connaissant tous les contrats établis pour chaque application.

Renégociation de contrat de QdR Au cours du temps il peut être nécessaire de renégocier un contrat de QdR, soit pour adapter la demande de l'application, soit pour s'adapter à un changement de QdR disponible dans le système, soit pour admettre une nouvelle application [113, 122]. Deux types d'adaptation des contrats de QdR sont utilisés : changer le niveau du contrat de QdR [122], ou utiliser une borne supérieure et une borne inférieure pour le contrat de QdR [113].

Dans le premier cas, l'adaptation consiste à renégocier le contrat de QdR. S'il n'y pas assez de QdR disponible, le niveau du contrat de QdR est diminué. Dans le second cas, la QdR affectée à l'application est rapprochée de la borne inférieure du contrat. Si les QdR disponibles augmentent, le mécanisme d'adaptation inverse est réalisé (changement de niveau du contrat de QdR, rapprochement de la borne supérieure).

Programmation à base de composants La programmation à base de composants est particulièrement adaptée à la gestion des contrats de QdR. En effet, les composants communiquent via des interfaces bien définies. Un contrat de QdR peut donc être aisément défini sur une interface, pour assurer une composition correcte. Une définition communément admise de la notion de composant est donnée par Szyperski dans [120] :

> *"A component is an unit of composition with contractually specified interfaces and fully explicit context dependencies that can be deployed independently and is subject to third-party composition."*

Dans cette définition, un composant possède des interfaces clairement définies. Ces interfaces permettent de composer et de déployer ce composant. On retrouve dans cette définition la notion d'interface permettant d'intégrer les contrats et notamment les contrats de QdR. Les travaux portant sur des contrats de QdR [12, 28, 46, 48, 91, 113, 122, 125] se basent sur des composants.

Les composants gérant l'accès aux ressources sont communément appelés *broker* [86, 88, 122]. C'est le composant *broker* qui effectue la négociation de contrat de

QdR, l'admission de l'application et la réservation de la QdR. Un composant *broker* est associé à une ressource. On retrouve aussi communément des composants appelés manager de ressource qui gèrent les QdR pour les applications durant l'exécution de ces dernières.

Les mécanismes présentés précédemment permettent de contractualiser l'accès aux ressources. Établir un contrat de QdR présuppose de :

1. connaître la QdR requise par une application et la QdR disponible, sur chaque ressource
2. utiliser une table de mapping, pour associer un niveau de QdS à une quantité de QdR, sur une ressource.

Le Tableau 2.5 est un exemple de table de mapping pour la ressource mémoire RAM de l'équipement exécutant le client. Une table de mapping peut être vue comme une fonction $f(contraintes\ QdS, flux, resource) = QdR$. f associe une QdR requise à un ensemble de contraintes de QdS, un flux et une ressource.

Quand la négociation de contrat de QdR a été effectuée, les QdR sont réservées pour l'application. Dans la suite de ce chapitre, nous décrivons les mécanismes de réservation des quantités de ressources fournis par le système d'exploitation. Ensuite, nous présentons les solutions de gestion des quantités de ressources existantes, utilisant les mécanismes de réservation pour garantir les QdR aux applications.

2.2.1 Réservation des quantités de ressources

La réservation de quantités de ressources est communément utilisée pour garantir les QdR requises par une application et ainsi garantir la QdS de l'application [37, 82, 87, 97]. Par exemple, une réservation de mémoire RAM va garantir que l'application n'utilisera pas la mémoire tampon ce qui pourrait compromettre ses propriétés temporelles et donc la QdS utilisateur. Cette propriété est appelée isolation temporelle [90], ou protection temporelle.

Avant de détailler les mécanismes de gestion des QdR du système d'exploitation, nous définissons les éléments nécessaires à la réservation d'une quantité de ressources. Ensuite nous présentons les types de réservations utilisés. Enfin nous introduisons les *control groups*, fournis par Linux pour gérer les ressources.

Éléments nécessaires à la réservation de QdR Pour réserver une QdR, sur une ressource, il faut [65, 82, 116] :

1. Estimer la QdR requise par l'application et la QdR que la ressource peut fournir
2. Effectuer un contrôle d'admission pour admettre ou rejeter la demande de l'application
3. Garantir et limiter les réservations admises
4. Comptabiliser l'utilisation d'une réservation par une application.

Types de réservation Il existe trois types de réservations [97] :
- Dure (*Hard* en anglais) : l'application ne peut pas utiliser plus de QdR que celle établie lors de la réservation
- Ferme (*Firm* en anglais) : l'application peut utiliser plus de QdR que sa réservation, seulement si la ressource n'est utilisée par aucune autre application
- Molle (*Soft* en anglais) : l'application peut utiliser plus de QdR que sa réservation, dans la limite où ce dépassement ne perturbe pas les autres réservations. Contrairement à une réservation ferme, quand l'application a utilisé toute sa réservation, elle est gérée comme une application sans réservation.

Quelque soit le type de réservations, la QdR réservée est garantie à l'application. En pratique, seules les réservations dures et molles sont utilisées [2, 24, 59, 96]. Quand une ressource est partagée entre des applications sans réservation et des applications avec réservation, les applications sans réservation utilisent ce qui est laissé libre par les applications avec réservation.

Control groups Les *control groups* ou *cgroups* [79, 80] sont utilisés dans Linux pour abstraire les ressources physiques et gérer les allocations à un groupe de processus. Un *cgroup* peut contenir des processus ou d'autres *cgroups*. Tous les processus créés par un processus restent dans le même *cgroup*.

Le noyau Linux utilise actuellement 4 types de *cgroups* :
- CPU : pour la ressource d'exécution
- réseau (*network*) : pour la ressource de communication
- mémoire (*memory*) : pour la mémoire RAM
- *IO* : pour les accès aux disques.

Dans la suite de cette section, nous présentons les mécanismes de réservation des quantités de ressources pour les trois types de ressource (exécution, stockage et communication). Enfin nous présentons les mécanismes de réservation adaptative.

2.2.1.1 Réservation des ressources d'exécution

Pour les ressources d'exécution, la ressource CPU a majoritairement été traitée. Si la plupart des travaux peuvent être utilisés pour gérer les autres ressources d'exécution, peu de solutions ont été mises en œuvre sur d'autres ressources d'exécution, telles que les DSP. Nous focalisons notre étude sur la ressource CPU. Pour les ressources DSP, nous considérons qu'elles sont affectées à une seule application à la fois. Cette limitation permet de garantir l'accès au DSP même si le système d'exploitation de l'équipement ne fournit pas de mécanisme de réservation.

Nous présentons d'abord le modèle des applications classiquement utilisé pour les applications périodiques dans les systèmes temps réel. Le CPU est partagé entre les différentes tâches s'exécutant sur un équipement à l'aide d'un ordonnanceur. Nous présentons les algorithmes d'ordonnancement permettant de réserver des QdR CPU. Enfin nous présentons comment le contrôle d'admission est effectué, selon l'algorithme d'ordonnancement.

Modèle des applications Une application de diffusion de contenus multimédia est composée d'un serveur et d'un client. Le serveur et le client forment un ensemble

de tâches périodiques, définies par (C_i, T_i). C_i est le *Worst Case Execution Time* (WCET) de la tâche et T_i sa période d'activation.

Un système temps réel est un système dans lequel les tâches ont des échéances. On distingue classiquement les systèmes temps réel durs et les systèmes temps réel mous. Dans les premiers, le non respect d'une échéance peut avoir de lourdes conséquences (système de freinage, avionique). Dans les seconds, le non respect d'une échéance va dégrader la QdS de l'application, par exemple la suppression d'une image pour une application multimédia. Les applications cibles de cette thèse sont dites temps réel.

Lors de l'activation périodique de la tâche, elle doit effectuer son traitement (nécessitant C_i unités de CPU) avant sa prochaine activation. Lors de l'activation d'une tâche, l'échéance est alors fixée au maximum par défaut à $t = t + T_i$.

Algorithmes d'ordonnancement Le CPU est géré à l'aide d'un ordonnanceur qui choisit la tâche à exécuter. Dans les systèmes d'exploitation considérés dans cette thèse, les ordonnanceurs sont préemptifs. Une tâche en cours d'exécution peut être préemptée à tout instant, pour laisser le CPU à une autre tâche (plus prioritaire). Il existe de nombreux algorithmes d'ordonnancement, ayant différents objectifs et différentes propriétés. On distingue classiquement les algorithmes à priorités de ceux à partage de temps.

Deux types d'algorithme d'ordonnancement permettent de garantir l'isolation temporelle des applications, et de réserver des QdR CPU [7, 123]. Nous présentons tout d'abord les algorithmes basés sur un serveur [2, 10, 59, 68, 69, 77, 114, 118]. Ces algorithmes se basent sur des algorithmes d'ordonnancement, classiquement utilisés dans les systèmes temps réel, pouvant être à priorités fixes ou dynamiques. Ensuite nous présentons les algorithmes d'ordonnancement à parts équitables [15, 29, 42, 94, 117,126,128,129] qui sont traditionnellement utilisés dans les systèmes d'exploitation dits généraux.

Ordonnancement basé sur un serveur Pour faire de la réservation de QdR CPU, un des algorithmes les plus utilisés est le *Constant Bandwidth Server* (CBS), proposé par Abeni et al. [2,4] et pouvant être intégré dans le noyaux Linux à l'aide d'un *patch* [6,11]. Le *Constant Bandwidth Server* s'appuie sur des tâches périodiques ordonnancées via l'algorithme *Earliest Deadline First* (EDF). EDF exécute la tâche avec l'échéance la plus proche et garantit qu'un ensemble de n tâches peut être ordonnancé si le test (2.2) est vrai [71], dans le cas optimal de tâches indépendantes. Il peut être nécessaire de se baser sur un formule plus précise ou de prendre une marge de sécurité, au prix d'une moins bonne utilisation de la ressource CPU. Ce problème n'a pas été considéré dans cette thèse, nous nous basons sur l'équation (2.2).

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{2.2}$$

Pour des applications multimédia il est difficile voir impossible de définir une valeur garantie de C_i (WCET). En effet, cette valeur dépend des flux utilisés (contenus, flots binaires, paramètres d'encodage), des logiciels utilisés, des ressources utilisées et des contraintes de QdS. Il semble donc difficile de prouver analytiquement le WCET pour l'exécution d'une application de diffusion de contenus multimédia. Dans ce contexte, l'algorithme CBS permet d'ordonnancer des tâches qui ne sont

pas bornées par un WCET. Un serveur ayant les propriétés (C_i, T_i) s'exécute comme une tâche périodique. La tâche à servir utilise elle les C_i unités de temps offertes par le serveur pour s'exécuter. Si elle n'est pas terminée, elle s'exécute dans la période suivante du serveur. Le serveur est ordonnancé en se basant sur ses paramètres qui sont eux garantis.

Dans l'algorithme CBS, un serveur CBS est caractérisé par un budget c_s et la pair (Q_s, T_s) où Q_s est le budget maximal et T_s est la période du serveur. Lorsque le serveur exécute une tâche, le budget c_s du serveur est diminué d'autant. Quand $c_s = 0$, le budget du serveur est rempli de nouveau (à $c_s = Q_s$) et son échéance est fixée à $d_{s,k+1} = d_{s,k} + T_s$. À la création du serveur $d_{s,0}$ est égal 0. Dans sa version originale, l'algorithme CBS fournit des réservations molles. En effet, quand le budget du serveur arrive à 0, il est rempli de nouveau et peut être exécuté s'il est plus prioritaire (décision laissé à l'ordonnanceur EDF). Dans [5], les auteurs intègrent des réservations dures. Dans ce cas, le budget du serveur n'est rempli de nouveau qu'à l'expiration de l'échéance $d_{s,k}$. Une tâche servie par le serveur ne peut donc pas être exécutée avant la fin de sa période. Elle ne peut pas utiliser plus de QdR que sa réservation. L'algorithme CBS permet de garantir que l'exécution d'une tâche ne manquera pas d'échéance si les paramètres (Q_s, T_s) correspondent à ceux de la tâche. Il garantit aussi l'isolation temporelle des tâches (cf. preuve dans [2]).

D'autres algorithmes basés sur un serveur similaires au CBS ont été proposés dans la littérature. Certains se basent sur le CBS et améliorent l'utilisation de la QdR CPU non utilisée par un serveur, notamment dans le cas de réservations molles. Ces algorithmes ne sont pas détaillés ici. Le lecteur intéressé pourra se reporter aux papiers les présentant pour plus de détails. On peut citer les algorithmes *Greedy Reclamation of Unused Bandwidth* (GRUB) [69], HGRUB [10], *Idle-time Reclaiming Improved Server* (IRIS) [77] et *Flexible CBS* [59]. On peut aussi citer des algorithmes ne se basant pas sur CBS mais qui sont similaires : *Sporadic Server* [114], *Deferrable Server* [118] et *Dependable Server* [68]

Ordonnancement à parts équitables Une autre façon de garantir l'isolation temporelle est d'utiliser un algorithme d'ordonnancement à parts équitables (*Fair scheduler*) [117, 123]. L'objectif d'un tel algorithme est d'affecter à chaque instant une part équitable du CPU à chaque tâche, en fonction de leur poids. Le modèle *Generalized Processor Sharing* (GPS) [94] est un modèle idéal pour le partage équitable d'une ressource. Il a d'abord été introduit pour ordonnancer les paquets réseaux mais sert aussi de référence aux algorithmes d'ordonnancement à parts équitables utilisés pour la gestion du CPU. Dans ce modèle, chaque ressource est considérée comme un flot qui peut être partagé indéfiniment entre les applications. À chaque instant, une tâche τ_i aura une QdR égale à sa part F_i, calculée à l'aide de l'équation (2.3) (pour n tâches, où w_i est le poids de la tâche i).

$$F_i = \frac{w_i}{\sum_{j=1}^{n} w_j} \tag{2.3}$$

Il existe deux types d'ordonnanceurs à parts équitables. Le premier est appelé *ordonnanceur à partage proportionnel* (*proportional share scheduling*). Ici, chaque tâche τ_i est affectée un poids w_i. La part de CPU F_i, affectée à la tâche τ_i est donnée, pour n tâches, par l'équation (2.3). Les poids sont relatifs les uns par rapport

aux autres. Pour garantir que l'arrivée d'une nouvelle tâche ne va pas remettre en cause la propriété d'isolation temporelle, un mécanisme de contrôle d'admission doit être utilisé. En effet, il est possible de rajouter autant de tâches que l'on veut et l'équation (2.3) sera toujours vraie. Cependant si la somme des poids dépasse la capacité de la ressource CPU alors les QdR requises par les applications ne seront plus garanties. Par exemple, une application A requiert une QdR de 50% de CPU. Si deux autres tâches sont lancées avec un poids de 50, alors A aura 33% du CPU ($F_A = \frac{50}{50+50+50}$). Le test d'admission (2.4) est utilisé pour n tâches, où max_cpu est une valeur fixe, normalisée.

$$\sum_{i=1}^{n} w_i < max\_cpu \tag{2.4}$$

Dans l'exemple précédent, la troisième tâche n'est pas lancée car $\sum_{i=1}^{n} w_i = 50 + 50 + 50 > max\_cpu$, max_cpu étant ici fixé à 100%.

Le deuxième type d'ordonnanceur à parts équitables est appelé *ordonnanceur p-fair*. Ici un poids w_i est affecté à chaque tâche et le test d'admission (2.5) est utilisé, où n est le nombre de tâches et m est le nombre de CPUs. Les poids sont déjà normalisés et le contrôle d'admission consiste uniquement à vérifier que la somme des poids n'excède pas la capacité de la ressource d'exécution, exprimée par le nombre de CPU.

$$\sum_{i=1}^{n} w_i < m \tag{2.5}$$

Une propriété importante des algorithmes d'ordonnancement à parts équitables est qu'ils sont *"work-conserving"* [94]. Cela signifie que le CPU est inactif seulement si aucune tâche ne peut s'exécuter. Cette propriété n'est pas forcément vérifiée par les algorithmes basés sur un serveur notamment quand des réservations dures sont utilisées.

En pratique, à cause de la précision des ordonnanceurs, chaque tâche aura eu sa part du CPU après un temps fixé par le système. Les différents algorithmes d'ordonnancement à parts équitables se basent sur la notion de temps virtuel pour simuler le modèle GPS.

Par exemple dans l'algorithme *Completely Fair Scheduler* (CFS) [92,129], qui est l'algorithme standard dans le noyau Linux depuis la version 2.6.23, chaque *thread*[9] est associé à un temps virtuel : *vruntime*. Le *vruntime* représente combien de temps le *thread* s'est exécuté. Pour maintenir l'équité entre les *threads*, l'ordonnanceur maintient un arbre rouge-noir [44] ordonné sur le *vruntime* de chaque *thread*. L'ordonnanceur vérifie quel est le *thread* à exécuter tous les quantum de temps Q. Il exécute le *thread* le plus à gauche de l'arbre, celui qui a le plus petit *vruntime*. Pour intégrer le poids des *threads*, le *vruntime* est incrémenté de manière inversement proportionnelle au poids du *thread*, comme le montre l'équation (2.6), dans laquelle $vruntime_i$ est le *vruntime* du *thread* τ_i et $delta\_exec\_\tau_i$ donne le temps pendant lequel il vient de s'exécuter.

$$vruntime_i = vruntime_{i-1} + \frac{delta\_exec\_\tau_i}{w_i} \tag{2.6}$$

9. dans Linux une tâche est appelée *thread*

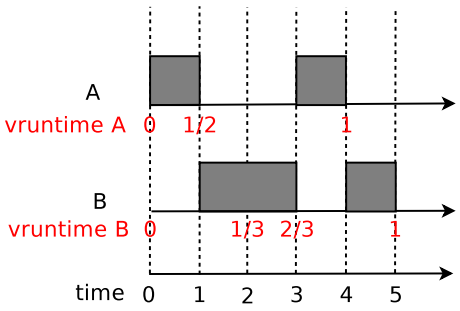


FIGURE 2.9 – Exemple d'ordonnancement avec l'algorithme CFS

Un *thread* avec un poids plus important sera exécuté plus longtemps car son *vruntime* augmente moins vite.

La Figure 2.9 donne un exemple d'ordonnancement, avec le CFS, de deux *threads* A et B ayant un poids respectif de 2 et 3. Le *vruntime* des *threads* est donné lorsque sa valeur change. Au début, le *vruntime* de chaque *thread* est égal à 0 et A est exécuté (décision arbitraire). À $t = 1$, le *vruntime* de A est incrémenté de $\frac{1}{2}$. Le *thread* B a toujours un *vruntime* de 0, il est donc exécuté. À $t = 2$, le *vruntime* de B est incrémenté de $\frac{1}{3}$. B est donc encore exécuté car il a toujours le plus petit *vruntime*. On peut vérifier qu'à $t = 5$, l'équation (2.3) est vérifiée : A a eu $\frac{2}{5}$ du CPU et B a eu $\frac{3}{5}$ du cpu.

L'algorithme CFS garantit que l'équation (2.7) est vérifiée pour tous les *threads* du système après une période donnée. F_i représente la part de CPU qui a été affectée au *thread*, exprimée en temps d'exécution. Ainsi l'équation (2.3), du modèle idéal GPS, est respectée à la fin de chaque période.

$$F_i = \frac{w_i}{\sum_{j=1}^{n} w_j} * period \tag{2.7}$$

Par défaut, la période de l'ordonnanceur CFS du noyau Linux est de 20 millisecondes. S'il y a trop de *threads* à ordonnancer, le noyau peut décider d'augmenter la période pour diminuer le surcoût de l'ordonnanceur.

D'autres algorithmes d'ordonnancement à parts équitables ont été proposés et ne sont pas détaillés ici. On peut notamment citer le *Weighted Fair Queuing* (WFQ) [29], *Start Fair Queuing* [42], WF^2Q [15], *Lottery scheduling* [126] et *Earliest Eligible Virtual Deadline First* (EEVDF) [117]. Le CFS utilise le même calcul pour le *vruntime* que les algorithmes WFQ et EEVDF.

Le système des *cgroups* CPU du noyau Linux utilise le CFS. De même que pour les *threads*, il est possible de définir un poids par *cgroup*. Le CFS ordonnance les *cgroups* en fonction de leur poids, de la même manière qu'il ordonnance les *threads* en fonction de leur poids à l'intérieur d'un *cgroup*.

Comparaison des deux types d'algorithme Les deux types d'algorithme présentés précédemment permettent de faire de la réservation de CPU. Les algorithmes basés sur un serveur semblent mieux adaptés pour des applications temps réel car ils se basent sur l'échéance des tâches. En revanche, ces algorithmes nécessitent de modifier le système d'exploitation de l'équipement. Or, la non modification des équipements est une contrainte forte dans le contexte de cette thèse. Les

équipements actuels du réseau local domestique utilisent des systèmes d'exploitation généraux. On retrouve sur ces équipements des algorithmes d'ordonnancement à parts équitables, par exemple le CFS dans le noyau Linux. Dans l'avenir, les algorithmes basés sur un serveur seront peut-être intégrés dans les systèmes d'exploitation des équipements du réseau local domestique. Il apparaît donc nécessaire de supporter les deux types d'algorithme d'ordonnancement abordés ci-avant.

Contrôle d'admission Les deux types d'algorithme permettent de garantir l'allocation de ressources CPU aux applications. Ajouter un contrôle d'admission permet de réserver la QdR, et de garantir l'isolation temporelle. Le contrôle d'admission décide si une tâche peut être exécutée sans perturber celles en cours d'exécution.

Les algorithmes basés sur un serveur et notamment le CBS permettent d'ordonnancer des tâches avec un ordonnanceur temps réel. Pour EDF, et les algorithmes similaires, un test d'admission peut être utilisé, tel que le test (2.2) pour EDF.

Les algorithmes d'ordonnancement à parts équitables ordonnancent les tâches en fonction de leur poids. Pour l'algorithme CFS, le test d'admission (2.4) est utilisé.

Le contrôle d'admission se base sur la QdR CPU requise et la QdR disponible. La QdR disponible est fixée par le test d'admission utilisé. La QdR requise est souvent considérée comme connue. L'application peut être démarrée une première fois pour connaître cette valeur [97]. Une autre solution consiste à utiliser une table de mapping pour associer un niveau de QdS à une QdR [111,122]. Dans ces travaux la table de mapping est considérée comme connue. Enfin, certains travaux s'intéressent à la prédiction de cette valeur pour des applications multimédia [58, 101, 102]. Dans ces travaux une fonction est générée par apprentissage. Cette fonction prend en entrée un ensemble de paramètres et donne le temps CPU requis pour le décodage de chaque image.

2.2.1.2 Réservation des ressources de communication

Comme nous l'avons vu à la section 2.1.3, les applications de diffusion de contenus multimédia utilisent trois ressources de communication : l'interface réseau du serveur, celle du client et les liens entre les deux. Nous commençons par présenter comment les paquets sont ordonnancés sur le serveur. Ensuite, nous présentons les éléments utilisés pour la réservation du lien. Pour garantir la QdR sur le lien, des mécanismes sont utilisés pour façonner le trafic émis, et définir la politique de gestion du trafic émis. Enfin un contrôle d'admission local (serveur) et global (liens) est utilisé pour garantir les réservations de QdR.

Ordonnancement du trafic émis Pour faire de la réservation de QdR réseau sur l'interface du serveur, des ordonnanceurs à parts équitables sont utilisés comme pour la ressource CPU. Comme nous l'avons vu à la section 2.2.1.1, ces algorithmes essaient d'approcher le modèle de flot idéal du *Generalized Processor Sharing* (GPS) [94, 95]. Le modèle théorique est ici limité par la taille des paquets réseaux à envoyer, car ils ne peuvent pas être divisés. On retrouve des approches similaires à celles utilisées pour le CPU tels les algorithmes WFQ [29], WF^2Q [15] ou *Stochastic Fairness Queueing* [78]. Comme pour la ressource CPU, ces algorithmes permettent de garantir une QdR équitable à chaque application souhaitant émettre sur le réseau, modulée par leur poids.

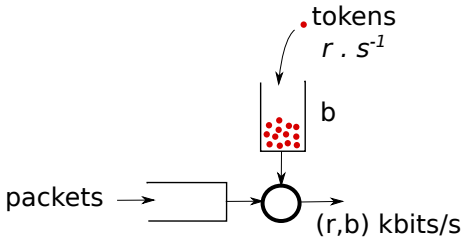


FIGURE 2.10 – Schéma du modèle à seau de jeton

Réservation du lien Dans le réseau local domestique il n'existe pas de mécanisme de réservation de bande passante, tel que le *Resource Reservation Protocol* (RSVP) [19,132], qui est classiquement utilisé dans les réseaux de plus grande taille. En effet ces mécanismes agissent au niveau de la couche 4 du modèle OSI. Dans un réseau local, les équipements communiquent directement par la couche 2 du modèle OSI.

Comme nous l'avons vu dans la section 2.1.1.1, il est parfois possible d'utiliser des standards de QdS réseau dans le réseau local domestique. La norme 802.1p est utilisée pour Ethernet et la norme 802.11e est utilisée pour le Wifi. Dans ces normes, les paquets marqués comme prioritaires sont envoyés d'abord et sont traités en priorité par la passerelle domestique. Ces mécanismes permettent de traiter en priorité les paquets de certaines applications. Le trafic des applications multimédia n'est donc pas perturbé par du trafic transitant sur le même lien. Les normes de QdS réseau ne garantissent pas le partage de trafic entre plusieurs applications de même priorité. Ce partage est fait en façonnant le trafic émis, grâce à une politique de gestion du trafic émis et à un contrôle d'admission local et global.

Façonnage du trafic émis Le façonnage du trafic permet de changer la forme du trafic émis par une application.

Deux modèles sont couramment utilisés : le modèle à seau de jeton (*token bucket*) et le modèle à seau percé (*leaky bucket*). La Figure 2.10 montre un schéma du modèle à seau de jeton. Dans ce modèle, les paramètres (r, b) sont utilisés. r correspond au rythme d'arrivée des jetons (en secondes), b correspond au nombre maximum de jetons pouvant être utilisés en une fois et à la capacité du seau. Si le seau contient déjà b jetons alors les nouveaux jetons sont jetés. Quand un paquet de taille L arrive, L jetons sont retirés du seau et le paquet est envoyé sur le réseau. S'il n'y pas assez de jetons ($L > nb$ jetons dans le seau), le paquet est placé en file d'attente. Ainsi, le débit d'envoi des paquets sur le réseau est caractérisé par un débit moyen r et un envoi en rafale maximum de taille b. Le modèle à seau percé utilise seulement le paramètre r pour l'arrivée des jetons, il n'autorise pas d'envoi en rafale. La Figure 2.11 schématise ce modèle. Le seau est ici aussi de taille de b mais est vidé avec un débit constant. Ainsi le débit d'envoi des paquets sur le réseau est caractérisé par un débit de r (débit d'arrivée des jetons).

Le façonnage du trafic émis permet de limiter la QdR réseau utilisée par une application. Le débit d'émission est régulé par les paramètres (r,b) pour le seau de jeton et par le paramètre (r) pour le seau percé. En introduisant du retard, il est possible de niveler les pics de consommation de bande passante et ainsi de maximiser l'utilisation des liens [49]. Le retard est introduit en stockant les paquets à émettre dans une file d'attente. Cette file d'attente est vidée quand le trafic émis a un débit

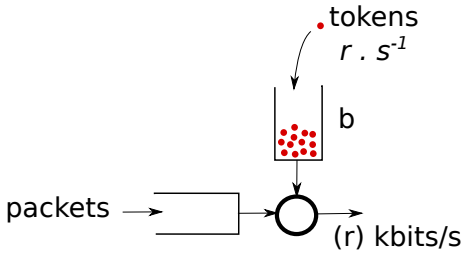


FIGURE 2.11 – Schéma du modèle à seau percé

inférieur à r. Le façonnage du trafic permet aussi de garantir qu'une application n'utilisera pas plus de bande passante que ce qu'elle a réservé [38, 60–62, 131]. Ainsi, il s'intègre dans une politique de gestion du trafic émis, en accord avec le contrôle d'admission comme nous allons le voir.

Politique de gestion du trafic émis La politique de gestion du trafic émis regroupe l'ordonnancement, le façonnage du trafic et la gestion du trafic excédentaire.

Certains algorithmes d'ordonnancement permettent de définir des classes de trafic. Ces classes permettent de traiter le trafic de plusieurs applications avec des priorités différentes. Il est possible de définir une hiérarchie de classes. Le trafic excédentaire d'une classe peut être réaffecté en priorité à une classe sœur (de même niveau) au lieu d'être réparti entre toutes les applications. On peut par exemple citer les ordonnanceurs *Hierarchical Packet Fair Queuing* (H-PFQ) [16] et *Class-based queueing* (CBQ) [36].

Dans le noyau Linux l'utilitaire *tc*[10] pour *"traffic control"* permet de définir la politique de gestion du trafic sortant. Cet utilitaire contient les éléments présentés précédemment (ordonnanceurs et façonnage) ainsi que des *Hierarchical Token Bucket* (HTB) [30]. Ces HTB fournissent un ordonnancement à parts équitables, avec la notion de hiérarchie. Cependant, les HTBs utilisent des valeurs de QdR réseau exprimées par des bandes passantes au lieu de poids. Les HTBs couplent les mécanismes d'ordonnancement avec le façonnage du trafic suivant un modèle à seau de jeton. Les HTBs sont par exemple utilisées dans [26] pour gérer la QdR réseau de différents types de liens dans un réseau local domestique.

D'autres politiques de gestion du trafic ont été proposées pour le réseau local domestique. Certaines solutions utilisent les normes de QdS réseau (802.11e et 802.1p) [26, 66, 131]. D'autres solutions s'intéressent aux réseaux locaux domestiques où les équipements ne supportent pas de norme de QdS réseau [60]. Dans les deux cas, grâce à la politique de gestion du trafic émis, les applications sont garanties d'avoir les QdR réseau requises.

Enfin, des travaux fournissent une politique de gestion du trafic en intervenant au travers de plusieurs couches du modèle OSI [62, 66]. Ces politiques permettent de mieux prendre en compte les spécificités du trafic multimédia.

La politique de gestion du trafic émis permet de réserver la QdR réseau sur l'interface réseau de l'équipement serveur. Il faut aussi garantir que la QdR au niveau du lien. Pour ce faire, un contrôle d'admission est réalisé au niveau lien.

10. `http://lartc.org/`

Contrôle d'admission Le contrôle d'admission compare la QdR réseau disponible et la QdR réseau requise.

La QdR disponible peut être estimée à l'aide de mesures actives, en envoyant des données sur le réseau. Une autre méthode consiste à utiliser un modèle pour estimer la QdR disponible [60, 131]. L'utilisation de mesure active introduit un surcoût, en termes de trafic. Cependant, les modèles d'estimation du trafic sont encore à l'état de recherche. Par exemple, une thèse est en cours à France Telecom, intitulée "mesure de bande passante pour les dispositifs hybrides", pour estimer les QdR disponibles dans le réseau local domestique.

La QdR requise est directement fonction du flot binaire et des contraintes de QdS. Il est possible d'analyser le flot binaire pour en déduire la QdR requise. Cela a par exemple été utilisé pour optimiser l'utilisation du lien [49].

Interface client Un équipement ne maîtrise pas les paquets arrivant sur son interface réseau. Il n'est donc pas possible de faire de la réservation du trafic entrant. Ce point doit être traité lors du contrôle d'admission.

2.2.1.3 Réservation des ressources de stockage

La gestion des ressources de stockage et plus particulièrement de la mémoire RAM a reçu moins d'attention de la part de la communauté. Du fait de la différence de temps d'accès entre la mémoire RAM et la mémoire SWAP, il est nécessaire de garantir que la QdR mémoire requise par une application de diffusion de contenus multimédia sera présente en RAM.

Dans les *cgroups* du noyau Linux, la mémoire RAM est gérée par *cgroup*[11]. L'algorithme de pagination (*swapping* en anglais) a été modifié pour fonctionner au niveau de chaque *cgroup* et non sur toute la mémoire RAM [20]. Le noyau Linux offre la possibilité de donner des limites dures ou molles par *cgroup* pour l'utilisation de la mémoire RAM. Ainsi quand un *cgroup* veut utiliser plus de mémoire que sa limite, l'algorithme de pagination réclame de la mémoire RAM appartenant à ce *cgroup*. S'il n'est pas possible de réclamer cette QdR, alors l'OOM *killer* est utilisé pour terminer un processus en cours d'exécution dans le *cgroup*. Une approche similaire pour la réservation de mémoire RAM a été proposée dans [33]. Ici la gestion de la mémoire est intégrée au *Resource Kernel* [97] qui contient aussi des algorithmes basés sur un serveur de CPU. Enfin dans Redline [130], une part de la mémoire RAM est réservée pour les applications interactives. Une application interactives est une application qui interagit avec l'utilisateur, une application multimédia par exemple. L'algorithme de pagination met ici en mémoire tampon en priorité les pages appartenant à des tâches non interactives.

2.2.1.4 Réservation adaptative des quantités de ressources

De nombreux travaux se sont intéressés à la réservation adaptative de CPU [3, 5, 25, 35, 111, 115]. L'idée est d'adapter la réservation d'une application en se basant sur une boucle de rétroaction. L'observation peut être faite, soit au niveau de l'utilisation de la ressource [1, 25], soit au niveau de la QdS de l'application [35, 111].

11. `http://lwn.net/Articles/268937/`

Si un algorithme basé sur un serveur est utilisé, la boucle de rétroaction peut être effectuée sur l'erreur d'ordonnancement [9, 25]. Cette erreur est la différence entre la fin de l'exécution de la tâche et la période du serveur. Dans [5, 25], à la fin de son traitement périodique, une tâche doit indiquer à l'ordonnanceur que son traitement est fini. Ainsi l'erreur d'ordonnancement peut être calculée. Cependant cet appel à l'ordonnanceur implique de modifier les applications existantes. Pour répondre à ce problème, un mécanisme d'observation de l'ordonnancement est utilisé pour savoir quand la tâche a fini de s'exécuter [8, 22, 23, 93]. Cette observation se base sur le fait que la tâche est exécutée par un serveur CBS. Dans un algorithme basé sur un serveur, un serveur (type CBS) est affecté un budget et une période. À la fin de la période, si le budget n'a pas été consommé cela signifie que la réservation est trop large. Au contraire, si le budget a été complètement utilisé alors la réservation n'est pas assez large. Pour observer la QdS de l'application, des métriques dépendantes de l'application sont utilisées, telles que le taux d'échéances manquées par exemple.

Après cette phase d'observation, les réservations sont adaptées. Si la somme des réservations ne dépasse pas la capacité de la ressource CPU, aucune action n'est nécessaire. Dans le cas contraire, plusieurs stratégies peuvent être adoptées. La première stratégie consiste à prioriser les applications les unes par rapport aux autres, pour favoriser certaines réservations. Une autre stratégie consiste à définir une réservation minimale pour chaque application. L'application peut tolérer d'avoir moins de QdR jusqu'à un certain point [75]. Cette stratégie doit être couplée avec un contrôle d'admission et suppose de connaître la QdR minimale requise par une application. Enfin une autre stratégie consiste à dégrader équitablement toutes les réservations [24].

La réservation adaptative a été utilisée de manière similaire pour la ressource de communication [35, 51, 75, 84, 111]. Pour la ressource mémoire RAM les QdR requises par les applications sont moins fluctuantes. Ainsi cette ressource a moins été traitée dans la littérature.

La réservation adaptative répond à trois objectifs : maximiser l'utilisation des ressources, avoir une connaissance moins précise des QdR requises ou s'adapter à un changement de QdR disponible dans l'environnement (dégradation de la qualité du lien, un équipement sur batterie par exemple). Dans tous les cas, une ou plusieurs applications peuvent ne pas avoir les QdR qu'elles requièrent. Ainsi la QdS fournie par une application, et donc la QdS perçue par l'utilisateur, peut temporairement être dégradée. Le cas d'un manque de ressources peut être géré par la renégociation d'un contrat de QdR (cf. introduction de la section 2.2). Dans le cadre de cette thèse, l'objectif est de garantir à priori la QdS perçue par l'utilisateur. L'approche adaptative ne semble pas adaptée. En outre, maximiser l'utilisation des ressources est ici moins important que garantir la QdS utilisateur.

2.2.2 Solutions de gestion des quantités de ressources

Nous avons présenté jusqu'ici les mécanismes utilisés pour gérer les contrats de QdR et réserver les QdR sur chaque ressource. Nous nous intéressons maintenant aux travaux intégrant ces mécanismes. La solution est une architecture qui coordonne la gestion des QdR requises pour chaque ressource.

Avant de commencer l'étude des solutions, nous définissons deux familles de solutions utilisées pour gérer les QdR. Ensuite, nous présentons les critères d'évaluation des solutions de gestion des QdR.

Familles de solution Tout d'abord une architecture de gestion des QdR peut être utilisée. Parmi les nombreuses définitions d'architecture, nous avons retenu celle de [14] :

> *"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them."*

Dans cette définition, une architecture définit les composants utilisés, leurs propriétés visibles et leurs relations. Une architecture de gestion des quantités de ressources contient les composants de gestion des QdR et leurs relations. Les solutions se focalisant sur la structure de gestion des QdR sont classées comme des architectures.

Une autre solution est d'utiliser un *framework* de gestion des QdR. Il est difficile de trouver une définition communément admise d'un *framework*. Dans le cadre de cette thèse nous utilisons la définition suivante :

> *Un framework d'exécution contient l'ensemble des services et des composants fournissant un cadre d'exécution aux applications.*

Ainsi un *framework* de gestion des QdR fournit un cadre d'exécution pour gérer les QdR des applications.

Il est possible de mixer les deux types de solutions, en définissant une architecture, utilisée pour fournir un cadre d'exécution aux applications.

Critères d'étude des solutions Pour chaque solution nous précisons si elle gère uniquement les ressources locales à un équipement ou si elle gère aussi des ressources globales. Nous donnons aussi le type de solution (architecture ou *framework*), les ressources gérées et les mécanismes employés pour garantir les réservations. Nous précisons si la solution utilise des réservations adaptatives, au niveau des contrats, de l'application ou de la ressource. Enfin, nous présentons comment des applications existantes peuvent être intégrées dans la solution et comment la solution s'intègre sur des équipements existants. Le tableau 2.6, page 46, résume chacun de ces points pour les différentes solutions.

Nous avons divisé en deux catégories les solutions des QdR, selon comment elles s'intègrent sur les équipements. Nous présentons d'abord les solutions se plaçant au niveau du système d'exploitation, puis nous nous intéressons à celles s'intégrant ou s'appuyant sur un intergiciel.

2.2.2.1 Gestion à l'aide du système d'exploitation

Les solutions présentées ici s'intègrent au niveau du système d'exploitation des équipements. Toutes ces solutions nécessitent de modifier le système d'exploitation pour intégrer des mécanismes de réservation.

AQuoSA AQuoSA [24] (*adaptive quality of service architecture*) est un *framework* permettant de faire de la réservation adaptative de QdR CPU sur un équipement. L'architecture d'AQuoSA est basée sur un noyau Linux modifié par le *Generic Scheduler Patch* [6] permettant d'avoir un ordonnanceur EDF. L'architecture est composée d'un module noyau de réservation et d'un module noyau de supervision. Le noyau Linux est modifié pour ajouter des *hooks* (pointeurs de fonction), permettant de déporter certaines décisions d'ordonnancement dans le module de réservation. Ce dernier implémente plusieurs algorithmes basés sur un serveur de CPU (CBS [2], IRIS [77] et GRUB [69]). Par défaut des réservations dures sont utilisées mais il est possible d'utiliser des réservations molles. L'architecture comprend également une librairie de réservation et une librairie de gestion de la QdS dans l'espace utilisateur. Les applications communiquent grâce à ces librairies avec les modules de réservation et de supervision. AQuoSA est une solution locale à un équipement.

Les auteurs fournissent une implémentation de la librairie de réservation pour créer un serveur CBS et lui affecter des tâches. Un serveur CBS ne peut exécuter qu'un seul *thread* à un instant donné, cependant plusieurs *threads* peuvent être attachés à un serveur CBS. Dans ce cas, ils sont exécutés avec une politique premier arrivé, premier servi. Le Listing 2.1 crée un serveur CBS avec les paramètres $(C_s, T_s) = (20000, 40000)$ et y attache le *thread* ayant l'id 830. Ici C_s et T_s sont exprimés en millisecondes. Lors de la création d'un serveur CBS, si le test d'admission d'EDF (test 2.2 page 29) n'est pas satisfait alors chaque serveur CBS voit sa réservation diminuée pour arriver à un ensemble de serveurs CBS ordonnançable. Cette stratégie maintient le système dans un état stable mais elle a l'inconvénient de ne pas garantir les réservations présentent dans le système lors de l'ajout d'une nouvelle réservation.

```
>qres -create -q 20000 -p 40000 -attach -tid 830
```

Listing 2.1 – création d'un serveur CBS et attachement d'un thread

En plus de la réservation de QdR CPU, l'architecture AQuoSA permet de faire de la réservation adaptative de QdR. L'adaptation est réalisée à l'aide d'une boucle de rétroaction, basée sur l'erreur d'ordonnancement. Pour utiliser les réservations adaptatives, les applications doivent utiliser l'API [12] de réservation. Quand une tâche commence à s'exécuter, et quand elle a fini son traitement périodique, elle l'indique à la librairie d'adaptation de la QdS application, via l'API. L'erreur d'ordonnancement pourra ainsi être calculée. L'utilisation de la librairie d'adaptation nécessite donc une modification des applications existantes. Il est possible d'effectuer une réservation pour une application existante, sans la modifier, mais la réservation ne sera pas adaptée. Comme AQuoSA propose des réservations adaptatives, les auteurs ne sont pas focalisés sur l'estimation les QdR requises.

Qinna Qinna [122] est une architecture de gestion de la QdS, basée sur des contrats de QdR. L'architecture contient un composant central appelé "QoSDomain" qui coordonne les réservations effectuées pour chaque application. Qinna est une solution locale à un équipement.

Qinna se base sur Think [34], un système d'exploitation implémenté à l'aide de composants clairement identifiés. Think permet de développer des systèmes d'-

12. *Application Programming Interface*

exploitation pour des équipements embarqués, ayant souvent de faibles capacités. Lorsqu'une application est démarrée, une chaîne de contrats de QdR est établie depuis les interfaces requises de l'application jusqu'aux interfaces offertes par les composants donnant accès aux ressources. Les réservations de QdR sont effectuées par des composants *"QoSComponentManager"* et garanties par des composants *"QoSComponentBroker"* .

Dans Qinna, le contrôle d'admission est fait lors de l'établissement de la chaîne de contrats de QdR. Les QdR requises pour un niveau de QdS sont obtenues à l'aide d'une table de mapping, considérée comme connue.

Qinna supporte la gestion des QdR pour les ressources CPU et mémoire RAM. Pour ces ressources, des composants *"QoSComponentBroker"* ont été développés. Pour le CPU différents algorithmes d'ordonnancement sont supportés. Pour gérer d'autres ressources, les composants *"QoSComponentBroker"* correspondant doivent être développés.

Enfin, Qinna permet de faire de l'adaptation des contrats de QdR grâce à des composants observant l'utilisation des ressources et la QdS des applications.

Redline Redline [130] est un ensemble de spécifications pour intégrer des tâches interactives (interagissant avec l'utilisateur) dans un système d'exploitation général. Cette solution est locale à un équipement. Un prototype a été implémenté en modifiant le noyau Linux. Cette implémentation fournit un *framework* de gestion des QdR. Redline a pour objectif de maximiser l'utilisation des ressources tout en permettant aux tâches interactives de s'exécuter correctement, à l'aide d'un ordonnanceur EDF. Si aucune tâche interactive n'est exécutable alors l'algorithme CFS est utilisé pour les autres tâches. Redline inclut aussi la gestion des ressources de stockage (disque et mémoire RAM) pour les tâches interactives.

Redline se différencie des autres approches sur le contrôle d'admission. Redline réserve une part du CPU pour l'ensemble des tâches interactives. Les tests d'admission se basent sur cette réservation et sur la charge courante du système et non sur les réservations effectuées pour chaque application. Les applications doivent fournir leur QdR CPU, disque et mémoire RAM. Le test d'admission consiste à vérifier que la nouvelle application ne va pas faire augmenter la QdR utilisée par les applications interactives au dessus du seuil réservé pour l'ensemble des applications interactives. Si le contrôle d'admission échoue, l'application est démarrée comme non interactive. Redline permet ainsi de maximiser l'utilisation des ressources, en se basant sur l'utilisation réelle des ressources lors du contrôle d'admission au lieu des QdR réservées par les tâches.

Comme Redline maximise l'utilisation des ressources, la charge du système peut devenir supérieure à la capacité CPU de l'équipement. Dans ce cas, une application interactive est dégradée en application non interactive. Quand la charge aura diminuée, l'application sera réintégrée comme interactive. L'optimisation de l'utilisation des ressources se fait ainsi au détriment de la QdS.

Enfin Redline utilise une API pour définir les besoins d'une application, ce qui permet d'inclure plus facilement des applications existantes. Cependant l'estimation des QdR requises par une application reste à faire manuellement. Redline ne fait pas de réservation adaptative.

Resource Kernel Le *Resource Kernel* [97] est un *framework* de gestion des QdR, implémenté dans le système d'exploitation, qui permet de garantir l'isolation temporelle des applications. L'architecture de gestion des QdR a été définie indépendemment du noyau Linux [90]. Cependant seule une implémentation modifiant le noyau Linux existe à notre connaissance. Le *Resource Kernel* offre une solution de gestion des QdR locale à un équipement.

Quand une application est démarrée, elle doit exprimer ses QdR requises pour chaque ressource. Les ressources CPU, disque, mémoire RAM [33] et les interfaces réseau [41] sont gérées. Un contrôle d'admission est effectué pour garantir les réservations. Le contrôle d'admission est réalisé sur toutes les ressources gérées. S'il échoue, l'application n'est pas démarrée. Le *Resource Kernel* utilise des réservations dures, fermes ou molles. Les réservations de CPU se font à l'aide d'un algorithme basé sur un serveur, utilisant des serveurs sporadiques [114] à priorités fixes. La mémoire RAM est gérée par réservation. Si une application utilise plus de mémoire RAM que sa réservation, elle ne peut pas prendre de la mémoire RAM à une autre application. Les réservations ne sont pas adaptées et sont garanties grâce au contrôle d'admission et aux mécanismes de réservation.

Effectuer une réservation pour une application existante nécessite de connaître ses QdR requises, mais ne nécessite pas de modification de l'application.

Distributed Resource Kernel Le *Distributed Resource Kernel* [64] est un *framework* permettant de faire des réservations de QdR CPU pour une application distribuée. Les réservations de CPU sont garanties par l'utilisation d'un *Resource Kernel* sur chaque équipement. Une application est modélisée par un graphe de tâche. Chaque tâche τ_i est caractérisée par (C_i, T_i) et est exécutée sur un équipement. Le contrôle d'admission est réalisé à deux niveaux, pour garantir les QdR requises sur tous les équipements et la QdS de l'application distribuée. Au niveau global, le contrôle d'admission vérifie que la somme des échéances de chaque tâche, du délai et de la gigue maximale des liens est inférieure à l'échéance de bout en bout de l'application. Au niveau local, sur chaque équipement, le contrôle d'admission est effectué par un *Resource Kernel*, avec les paramètres (C_i, T_i). Ces paramètres sont considérés comme connus pour chaque tâche. Pour intégrer une application existante il faut fournir son graphe de tâches distribuées.

Seule la ressource CPU est complètement considérée dans [64]. La ressource réseau, sur le lien, est considérée au niveau du contrôle d'admission mais aucune réservation n'est effectuée. Cette solution ne fournit pas de réservation adaptative.

2.2.2.2 Gestion à l'aide d'un intergiciel

Dans les systèmes distribués, il est courant d'utiliser un intergiciel pour masquer l'hétérogénéité des systèmes d'exploitation et abstraire la distribution des équipements. Des intergiciels tels que CORBA [43] sont classiquement utilisés. En général, ces intergiciels ne sont pas adaptés aux applications ayant des contraintes de QdS. Nous présentons les solutions existantes pour gérer les QdR dans des systèmes distribués.

End-to-End QoS management Dans [75] les auteurs présentent une architecture de gestion de la QdS de bout en bout. Ces travaux s'appuient sur le *framework*

QuO [125], qui permet de concevoir des applications distribuées, pouvant s'adapter à un changement de QdR disponible. Dans QuO, lors d'un appel distant, un composant *"delegate"* intercepte l'appel. Le *delegate* appelle le QuO *kernel* pour connaître l'état courant du système. Le *delegate* choisit le niveau de QdS utilisé puis l'appel distant est effectué. QuO fournit aussi des interfaces pour gérer les changements d'état dans le système. Les Qoskets [106] sont des composants QuO, de gestion de la QdS, dans les couches application et ressource, visant à être génériques et réutilisables.

L'architecture présentée dans [75] comprend un composant *"system resource manager"* qui interagit avec des *"local resource manager"* (présents sur chaque équipement) pour faire de la gestion des QdR. Un contrôle d'admission global est réalisé par le *"system resource manager"*. Le contrôle d'admission local étant laissé à la charge de chaque *"local resource manager"*. Les réservations de QdR réseau sont effectuées par un composant *broker*, le *"DiffServ qosket"* [27]. Ce composant utilise la norme de QdS réseau DiffServ. DiffServ est un protocole de couche 3 dans le modèle OSI qui permet de marquer et prioriser les paquets dans un réseau IP. Un composant réalisant du façonnage de trafic est aussi utilisé pour garantir que l'application n'utilisera pas plus de QdR réseau que sa réservation. La QdR CPU est réservée à l'aide d'un composant *"CPU Broker qosket"* [50]. Les réservations de CPU sont garanties par l'utilisation de TimeSys Linux [13] fournissant un ordonnanceur basé sur un serveur. Les deux composants *brokers* fournissent des réservations adaptatives. Le *"system resource manager"* coordonne l'adaptation de l'application de bout en bout. L'adaptation consiste ici à faire varier le niveau de compression des images envoyées, pour une application de surveillance militaire. L'architecture de gestion des QdR mise en place dans ces travaux nécessitent que l'application soit conçue à l'aide de composants qoskets, en se basant sur CORBA et que les équipements utilisent le système d'exploitation TimeSys Linux.

RACE RACE [111] (*Resource Allocation and Control Engine*) est un *framework* de gestion des QdR. RACE propose une approche similaire aux travaux présentés dans le paragraphe précédent. Cependant, dans RACE, la gestion des QdR se base sur CIAO [127], un intergiciel à composants prenant en compte la QdS. CIAO se base sur TAO [107], une implémentation temps réel de CORBA. Dans ces travaux, les auteurs s'intéressent au développement d'applications distribuées temps réel embarquées, pouvant s'adapter à une variation de QdR disponible. L'adaptation est réalisée à l'aide d'une boucle de rétroaction. Cette boucle de rétroaction est effectuée par des composants observant les QdR disponibles et utilisées (réseau et CPU) et un composant central permettant de coordonner les décisions. L'observation peut se faire sur des métriques génériques, telles que le délai de transmission ou sur des métriques spécifiques à l'application [110], fournies pas le développeur. La QdR réseau est réservée grâce à DiffServ. Le CPU et la mémoire RAM sont réservés en utilisant un système d'exploitation temps réel, tel que RT-Linux offrant un ordonnanceur EDF. Par rapport aux travaux autour de QuO, CIAO peut être configuré à la demande et les contraintes de QdS des applications peuvent être exprimées dans un formalisme abstrait, transformé automatiquement par RACE pour chaque équipement. RACE adresse principalement des problèmes liés à la conception d'applications temps réel embarquées.

13. `http://www.timesys.com/`

QARMA QARMA [35] (*Quality-based Adaptive Resource Management Architecture*) est un *framework* de gestion des QdR, distribué. Les auteurs se sont intéressés à la mise en œuvre d'une architecture de gestion des QdR permettant d'adapter des applications distribuées à un changement de QdR disponible. QARMA se base sur QuO. On retrouve dans QARMA, un composant central (*Decision maker*) qui collecte les informations récupérées par différents composants observateurs (*Monitors et Detectors*). Les décisions d'adaptation sont ensuite réalisées par des *enactors*, présents sur chaque équipement. Enfin QARMA utilise un *System Repository Service* pour stocker toutes les informations utiles à la gestion des QdR (spécifications, QdR requises, adaptations). QARMA permet de gérer le CPU, la mémoire RAM et le réseau. Les réservations sont garanties par l'utilisation de système d'exploitation temps réel (non détaillé dans les travaux). QARMA à besoin de modifier les applications existantes pour gérer leur QdR.

FRSH/FORB FRSH/FORB [112, 113] est un *framework* de gestion des QdR qui a pour objectif de gérer différents types de ressources. Quatre couches sont définies : *couche application*, *couche indépendante des ressources*, *couche dépendante des ressources* et *couche système d'exploitation*. La *couche application* fournit une API de négociation des contrats de QdR aux applications. Lorsqu'une application est démarrée, un contrat de QdR est négocié et une réservation est créée en cas de succès. En cas d'échec, l'application n'est pas démarrée. La *couche indépendante des ressources* fait le lien entre les applications et les *brokers* de ressource de la *couche dépendante des ressources*. La *couche indépendante des ressources* s'occupe aussi de renégocier un contrat de QdR, un contrat de QdR étant basé sur une borne supérieure et inférieure. La couche *dépendante des ressources* contient les managers et les *brokers* de ressource (appelés ici *allocator*). Un contrat de QdR est représenté par une structure de données. Un contrat est toujours identifié par un ID. La structure contient ensuite le type et l'identifiant de la ressource utilisée. La structure contient ensuite des champs communs à toutes les réservations tels que le budget (QdR requise). Pour chaque type de ressources, des champs spécifiques peuvent être ajoutés à la fin de la structure, comme les contraintes de QdS réseau pour les ressources réseau par exemple. Cela permet d'avoir une gestion plus uniforme des différents types de ressources. Ces champs servent aussi à exprimer des contrats dans les différentes couches du *framework*.

Chaque manager effectue le contrôle d'admission pour une ressource. Les ressources gérées sont le CPU, le disque et le réseau. Pour la ressource réseau, il y a un seul manager et plusieurs *brokers* (un pour chaque interface réseau). Pour les ressources CPU et disque, il y a un manager et un *broker* sur chaque équipement. Enfin la *couche système d'exploitation* offre une abstraction du système d'exploitation sous la forme d'une API. FRSH/FORB a été implémenté sur différents systèmes d'exploitation. Cependant, tous les systèmes d'exploitation utilisés fournissent un ordonnanceur EDF.

Les réservations de QdR CPU sont garanties par le système d'exploitation, AQuoSA dans [113]. Les réservations de QdR réseau sont garanties par la norme de QdS réseau 802.11e et le façonnage de trafic. L'estimation des QdR requises est obtenue en effectuant un *benchmark* manuel. L'application est démarrée plusieurs fois et des statistiques sont collectées pour définir les paramètres à utiliser pour l'établissement

d'un contrat de QdR. Enfin, FRSH/FORB se base sur un intergiciel CORBA pour résoudre les problématiques liées aux systèmes distribués.

QualMan Dans QualMan [85], la gestion des QdR est découpée en quatre couches : *application*, *intergiciel*, *système d'exploitation* et *réseau*. QualMan est un *framework* contenant des composants *brokers* pour gérer les QdR des ressources CPU et mémoire RAM de chaque équipement et le réseau. Lors du démarrage d'une application multimédia, un contrat de QdR est négocié entre l'application et les *brokers*. Si la négociation réussit, les réservations sont réalisées par les différents *brokers*. Les QdR requises sont considérées comme connues. Les réservations de CPU et de mémoire RAM sont garanties en utilisant un système d'exploitation général avec une extension temps réel (offrant un ordonnanceur basé sur un serveur. La QdR réseau est garantie grâce au protocole ATM [56] (*Asynchronous Transfer Mode*). ATM est un protocole de couche 2 du modèle OSI qui n'est plus utilisé dans les réseaux locaux actuels. Nous ne détaillons pas son fonctionnement ici. QualMan peut supporter des applications existantes sans les modifier. L'adaptation est gérée au niveau des contrats de QdR dans QualMan.

2.2.2.3 Tableau de comparaison des solutions existantes

Le Tableau 2.6 résume les solutions de gestion des quantités de ressources présentées dans ce chapitre. La colonne 1 (solution) contient le nom et les références de la solution. La colonne 2 (locale/dist) rappelle si la solution est locale à un équipement ou distribuée. La colonne 3 (type) donne le type de la solution (Architecture et/ou *Framework*). Les colonnes 4 (ressources gérées) et 5 (garantie) donnent les ressources gérées par la solution et comment les réservations de QdR sont garanties. *OS* signifie l'utilisation d'un système d'exploitation spécifique (Think pour Qinna par exemple) ou une modification importante du noyau Linux. Pour AQuoSA, le *patch* du noyau est relativement léger, c'est pourquoi une différence est faite. Les autres termes donnent les standards utilisés pour garantir les réservations. Dans distRK [64], le contrôle d'admission (*CA*) permet de garantir que l'application distribuée respectera ses échéances. La colonne 6 (niveau solution) donne le niveau de la solution : intégrée dans le système d'exploitation (*OS*), en utilisant une API ou intégrée dans un intergiciel. La colonne 7 (adaptation) précise comment l'adaptation est prise en compte :

- *Contrat* : les mécanismes de gestion des contrats de QdR assurent la renégociation d'un contrat de QdR
- *Res* : une réservation adaptative est utilisée
- *App* : l'application est adaptée sans utiliser une renégociation d'un contrat de QdR
- *non* : l'adaptation n'est pas supportée.

Enfin les colonnes 8 (appli existante) et 9 (intrusivité) donnent la qualité de la solution vis-à-vis de l'intégration des applications existantes et de son intrusivité :

- *Appli. existante :* Cette colonne donne la difficulté pour intégrer des applications existantes dans la solution.
  - * → modifications importantes
  - ** → modifications légères
  - *** → pas de modification nécessaire

- *intrusivité* : Cette colonne donne le niveau d'intrusivité de la solution. Cela comprend la nécessité de modifier les applications, de modifier le système d'exploitation et/ou d'utiliser un intergiciel spécifique. Vu le domaine ciblé, la meilleure solution ne serait pas intrusive.
  - * → très intrusive : utilisation d'un OS spécifique
  - ** → intrusive : modification du noyau Linux
  - *** → peu intrusive : suppose l'existence de mécanismes de gestion des QdR non standard dans Linux

solution	**locale / dist**	**type**	**ressources gérées**	**garantie**	**niveau solution**	**adaptation**	**appli existante**	**intrusivité**
aquosa [24]	locale	A/F	CPU	*patch* Linux + module RR	OS (patch) + API	Res / App	***	**
qinna [122]	locale	A	CPU, mem	OS	OS	Contrat	*	*
Redline [130]	locale	F	CPU, mem, disque	OS	OS	non	***	**
Resource Kernel [33, 41, 90, 97]	locale	F	CPU, réseau mem, disque	OS	OS	non	***	**
distRK [64]	dist	F	CPU	*Resource Kernel* + CA	OS	non	**	**
E-to-E QoS management [75]	dist	A/F	CPU, réseau	TimeSys DiffServ, Façonnage	intergiciel	Res / App	*	*
RACE [110, 111]	dist	A/F	CPU, réseau mem	TAO, DiffServ RT-Linux	intergiciel	Res / App	*	*
QARMA [35]	dist	A/F	CPU, réseau mem	QuO RTOS	intergiciel	App	*	*
FRSH/FORB [113]	dist	F	CPU, réseau disque	AQuoSA, 802.11e	intergiciel, OS	Contrat	***	***
QualMan [85]	dist	F	CPU, réseau mem	OS + extension RT	intergiciel	contrat	***	**

TABLE 2.6 – Comparaison des solutions de gestion des QdR. *(dist (distribuée). Type : A (Architecture), F (Framework). mem : mémoire RAM, Garantie : CA (Contrôle d'admission). Adaptation : Res (niveau ressource), App (niveau application), Contrat : contrat de QdR, non : pas d'adaptation. *,**,*** → qualité.)*

2.3 Discussion

Dans ce chapitre nous avons présenté un état de l'art de la gestion des quantités de ressources (QdR). Pour garantir l'ensemble des contraintes de QdS de l'application et satisfaire la QdS utilisateur, un contrat de QdR est négocié sur chacune des ressources utilisées. Une fois la négociation établie, les QdR sont réservées pour l'application. Un contrat de QdR peut être renégocié au cours du temps.

Nous avons présenté, dans ce chapitre, les mécanismes de réservation des quantités de ressources, puis les solutions de gestion des QdR existantes intégrant ces mécanismes de réservation pour garantir l'ensemble des contrats de QdR négociés pour chaque application.

Mécanismes de réservation des quantités de ressources Pour les ressources d'exécutions, principalement le CPU, deux types d'algorithme d'ordonnancement peuvent être utilisés : les algorithmes basés sur un serveur et les algorithmes d'ordonnancement à parts équitables. Les algorithmes basés sur un serveur se basent sur des algorithmes d'ordonnancement temps-réel, tels que EDF, sur la QdR requise et la période des applications. Les algorithmes d'ordonnancement à parts équitables utilisent des poids relatifs les uns par rapport aux autres, la période d'affectation des QdR CPU est fixée par l'ordonnanceur. Ces deux types d'algorithme, couplés avec un contrôle d'admission adapté, garantissent à chaque application la QdR CPU requise.

Pour les ressources de communication, nous avons présenté les mécanismes d'ordonnancement, de façonnage du trafic et la politique de gestion du trafic émis au niveau de l'interface réseau du serveur. Pour les liens, les normes de QdS réseau 802.1p et 802.11e peuvent être utilisées et doivent être couplées avec du contrôle d'admission. Enfin, pour l'interface client il faut se baser sur le contrôle d'admission. En effet, il n'existe pas de mécanisme permettant de réserver de la QdR réseau lors de la réception de trafic.

Pour les ressources de stockage, nous avons limité notre étude à la mémoire RAM. La mémoire RAM est réservée à l'aide de l'algorithme de pagination, qui garantit que la mémoire réservée par une application sera effectivement en mémoire RAM.

Il est possible d'adapter les réservations en fonction de l'utilisation des QdR par une application ou de la QdR disponible sur la ressource. Cette approche permet de maximiser l'utilisation des ressources mais au détriment de la QdS des applications. Une approche basée sur une réservation où les QdR requises sont connues à l'avance nous parait mieux adaptée.

Étude des solutions de gestion des quantités de ressources existantes Cette étude a montré l'importance du système d'exploitation dans la gestion des QdR. En effet, certaines solutions s'intègrent dans le système d'exploitation, à l'aide d'un *patch*, pour intégrer des mécanismes de réservation de QdR. Les solutions s'intégrant dans un intergiciel supposent l'existence de mécanismes de réservation dans les systèmes d'exploitation. Dans tous les cas, les réservations demandées par une solution de gestion des QdR sont garanties par les mécanismes de réservation des QdR, fournis par le système d'exploitation.

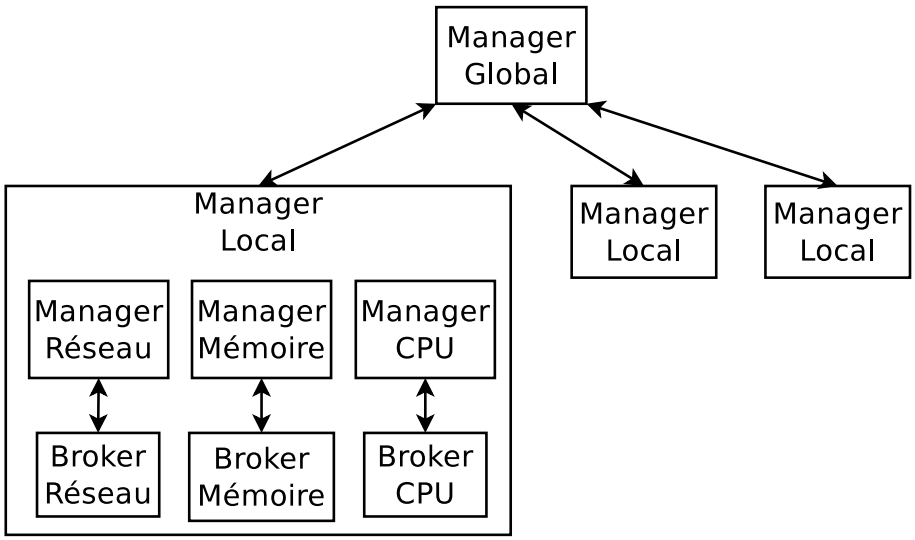


FIGURE 2.12 – Architecture de gestion des quantités de ressources

Cette étude a ensuite mis en évidence une architecture de gestion des quantités de ressources communément utilisée, présentée sur la Figure 2.12. Dans cette architecture on retrouve un manager global qui communique avec un ensemble de managers locaux, installés sur chaque équipement. Le manager global coordonne la gestion des QdR, effectuée localement par chaque manager local, et gère la QdR des ressources partagées, les liens dans notre contexte. Dans certaines solutions, le manager global est distribué en plusieurs composants [64, 113]. De son côté, le manager local est composé d'un couple "manager - *broker*" pour chaque ressource : CPU, mémoire RAM et interface réseau. Le *broker* effectue la négociation de contrat, l'admission de l'application et la réservation de la QdR. Le manager gère les QdR réservées pour les applications durant l'exécution de ces dernières (démarrage, arrêt, demande de renégociation de contrat). Selon les solutions, des noms différents sont utilisés mais, dans notre compréhension, les principes sont assez similaires.

Limites des solutions existantes L'utilisation des solutions existantes, dans le contexte de cette thèse, est rendue difficile par les contraintes du domaine d'étude. Il existe trois contraintes principales :

1. *Support des applications existantes :* Les solutions de gestion des QdR nécessitant de modifier les applications ne peuvent pas être utilisées directement (contrainte de non modification des applications de diffusion de contenus multimédia existantes). Pour certaines solutions, aucune modification des applications n'est nécessaire, cependant, elles ne fournissent pas un cadre d'exécution garantissant l'ensemble des QdR requises à une application de diffusion de contenus multimédia.
2. *Support d'équipements existants :*
   (a) Les solutions de gestion des QdR sont toutes intrusives, mais à des niveaux différents. Une modification du noyau Linux, même légère, peut être coûteuse à mettre en œuvre voire impossible sur certains équipements. Or, toutes les solutions présentées nécessitent de modifier le système d'exploitation ou d'utiliser un système d'exploitation spécifique.

(b) Aucun intergiciel standard ne s'est imposé dans le réseau local domestique. De plus, dans le réseau local domestique les problématiques d'adressage et d'interconnexion des équipements sont résolues à l'aide de standards tels que UPnP. Il n'est donc pas possible d'utiliser un *framework* basé sur un intergiciel spécifique, tel qu'une implémentation temps réel de CORBA. Pour gérer les QdR, il est nécessaire d'interagir avec le système d'exploitation des équipements.

3. *Support de l'hétérogénéité des équipements et des contenus multimédia :*

(a) Les équipements du réseau local domestique sont fortement hétérogènes. On retrouve par exemple différents types d'ordonnanceurs et différents types de liens. Une solution de gestion des QdR doit être capable de s'adapter à chaque équipement pour abstraire cette hétérogénéité. Or cette contrainte n'a pas ou peu été considérée dans les solutions de gestion des QdR.

(b) Les solutions présentées supposent que les QdR requises par les applications multimédia sont connues à l'avance. Pour les applications de diffusion de contenus multimédia, ces valeurs dépendent du flot binaire, des équipements utilisés et des logiciels utilisés, ce qui conduit à un nombre important de données. Dans ce contexte, connaître, accéder et stocker les QdR requises est un problème.

Au vu de ces limites, nous avons mis en œuvre un *framework* de gestion des QdR répondant aux contraintes du domaine d'étude. Nous détaillons cette contribution dans les deux chapitres suivants. Le chapitre 3 s'intéresse à l'estimation des QdR requises, dans le contexte hétérogène du réseau local domestique. Le chapitre 4 présente notre *framework*, qui garantit les QdR requises par les applications de diffusion de contenus multimédia démarrées dans le réseau local domestique.

Deuxième partie

Contributions

Chapitre 3

Estimation et stockage des quantités de ressources requises

Sommaire

Quand l'utilisateur veut démarrer une application de diffusion de contenus multimédia, il faut connaître les quantités de ressources (QdR) dont elle a besoin afin de garantir la Qualité de Service (QdS) attendue. Dans ce chapitre, nous détaillons les méthodes d'acquisition et le calcul des valeurs de QdR requises pour les ressources réseau, CPU et mémoire RAM. Les QdR CPU et mémoire RAM dépendent des flux utilisés (contenus et paramètres d'encodage), des logiciels utilisés, de la ressource elle-même et des autres ressources utilisées. L'ensemble des paramètres utilisés lors de la demande de l'utilisateur constitue un scénario. Au vu du nombre important de scénarios possibles nous proposons d'agréger les QdR requises afin de limiter la quantité d'information à stocker.

3.1 Acquisition des quantités de ressources requises

Il existe deux façons d'estimer les profils d'utilisation des ressources :

1. *par analyse du flux* : la QdR requise sur une ressource est donnée par une fonction prenant en paramètre le flux à diffuser
2. *par simulation* : Le scénario est exécuté et les QdR utilisées sont mesurées.

Si le temps d'analyse est raisonnable, il peut être fait en ligne, sinon les valeurs obtenues doivent être stockées et réutilisées lors du démarrage d'une application. Avec une simulation les valeurs obtenues sont stockées.

Pour les ressources réseau, les QdR requises dépendent du flux utilisé. Le flot binaire du flux est donc analysé pour connaître le profil d'utilisation et calculer la QdR requise.

Concernant les ressources CPU, l'estimation par analyse du flux fait l'objet de différentes recherches. Celles-ci (présentées lors de l'étude du contrôle d'admission à la section 2.2.1.1) s'intéressent à construire par apprentissage une fonction $F(flux)$ qui associe à un flux le temps de décodage de chaque image du flux. Une fonction est construite pour un seul équipement. Dans le contexte hétérogène du réseau local domestique, de nombreux équipements sont présents. Il apparaît difficile de générer une fonction pour tous les équipements. De plus, cette fonction permet de connaître le temps CPU utilisé pour décoder chaque image. Les mécanismes de réservation de QdR CPU existants utilisent des poids ou des "budgets". Il n'est pas évident de transformer un profil d'utilisation, contenant la QdR requise pour décoder chaque image, en une valeur de QdR utilisable par les mécanismes de réservation. En outre, ces fonctions d'estimation sont utilisées pour effectuer une adaptation plus efficace de QdS au niveau applicatif, et non pour réserver une QdR à une application.

Pour les ressources mémoire RAM, nous n'avons pas connaissance d'outils permettant d'estimer la QdR requise pour décoder un flux donné.

Ainsi, nous utilisons une estimation basée sur une mesure des profils d'utilisation pour les ressources CPU et mémoire RAM.

Nous détaillons maintenant comment les profils d'utilisation sont acquis, pour les différentes ressources. Ensuite, nous expliquons nos choix de valeurs statistiques pour exprimer la QdR requise.

3.1.1 Acquisition des profils d'utilisation des ressources

Pour appuyer l'explication de l'acquisition des profils d'utilisation, nous nous basons sur un exemple. L'équipement serveur, l'équipement client, les logiciels et le lien utilisés par le scénario sont donnés dans le Tableau 3.1a. L'équipement client utilise uniquement la ressource d'exécution CPU (res. exec.) pour décoder le flux. Sur l'équipement serveur, seul le CPU est utilisé. Les Tableaux 3.1c et 3.1d donnent les capacités et les caractéristiques de QdS des ressources utilisées dans cet exemple. Le Tableau 3.1b précise le flux et les paramètres d'encodage du flux vidéo. Dans cet exemple, le flux utilisé se compose uniquement du flux vidéo. Le flux utilisé dans ce scénario est le flux *big buck bunny*[1] qui a été réencodé à l'aide de l'encodeur `x264`. Les paramètres utilisés pour l'encodage sont la taille des images au format CIF (352x288 pixels), le profil de base du standard *h.264* et une fréquence d'images de 25 images par seconde. Nous avons vu à la section 2.1.2.1 qu'il existe un grand nombre de paramètres d'encodage. Les autres paramètres d'encodage sont laissés à leur valeur par défaut.

Pour garantir la QdS de l'application, il faut connaître les profils d'utilisation sur toutes les ressources utilisées. Dans cet exemple, les ressources utilisées sont le CPU et la mémoire RAM de l'équipement serveur et de l'équipement client. Les

1. disponible sur `http://www.bigbuckbunny.org/`

serveur			client			lien		
id	**logiciel**	**res. exec. utilisée**	**id**	**logiciel**	**res. exec. utilisée**	**id**	**type**	**protocole**
1	`vlc`	CPU	2	`mplayer`	CPU	1	eth	RTSP/UDP

(a) équipements

nom	**encodeur**	**paramètres**
big buck bunny	*h.264*	CIF, profil de base, 25 im/s

(b) flux

id	**nom**	**CPU (en %)**	**mémoire (en Mo)**	**interface réseau (en Mbits/s)**
1	PC_1	200	2000	100
2	BB_1	100	256	100

(c) Capacités des ressources des équipements

id	**capacité**	**caractéristiques de QdS réseau**		
		délai (en ms)	**gigue (en ms)**	**taux de perte**
1	100	10	15	0.05%

(d) Capacités et caractéristiques de QdS du lien

TABLE 3.1 – Scénario 1. *(res. exec. : ressource d'exécution, im. : images)*

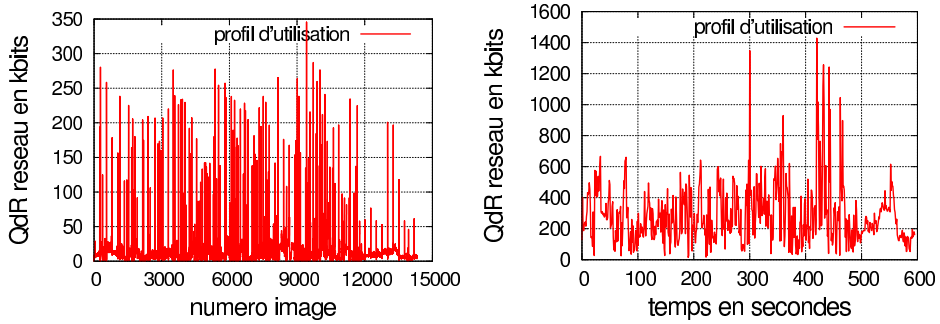
ressources de communication utilisées sont les interfaces réseau des équipements et les liens réseau entre les équipements. L'utilisation des ressources de communication est identique pour toutes les ressources. Pour cet exemple, il faut donc acquérir cinq profils d'utilisations : un profil d'utilisation des ressources réseau, un profil d'utilisation des ressources CPU et mémoire RAM de chaque équipement.

3.1.1.1 Ressources réseau

Pour les ressources réseau, le profil d'utilisation est estimé en analysant le flot binaire du flux du scénario. Cette analyse est réalisée à l'aide d'outils existants, qui donnent la taille des images du flux vidéo à diffuser. La taille des images détermine le profil d'utilisation des ressources réseau. La Figure 3.1a montre le profil d'utilisation ainsi calculé pour le scénario 1.

Dans cette thèse, nous nous intéressons au système d'exploitation Linux. Or, le noyau Linux gère l'envoi des données sur le réseau avec une granularité temporelle d'une seconde. Le profil d'utilisation réseau précédemment calculé est évalué image par image. Il doit donc être transformé sur une échelle d'une seconde. Pour passer d'une granularité d'une image à une granularité d'une seconde, la somme des tailles d'images sur une seconde est utilisée. L'équation (3.1) donne le calcul du profil P_{mean}, avec une granularité d'une seconde, pour un flux avec n images par seconde, à partir d'un profil P, avec une granularité d'une image.

$$P_{mean}(i) = \sum_{j=1}^{n} P(j) \tag{3.1}$$



(a) profil d'utilisation avec une granularité d'une image

(b) profil d'utilisation avec une granularité d'une seconde

FIGURE 3.1 – Profils d'utilisation avec différentes granularités

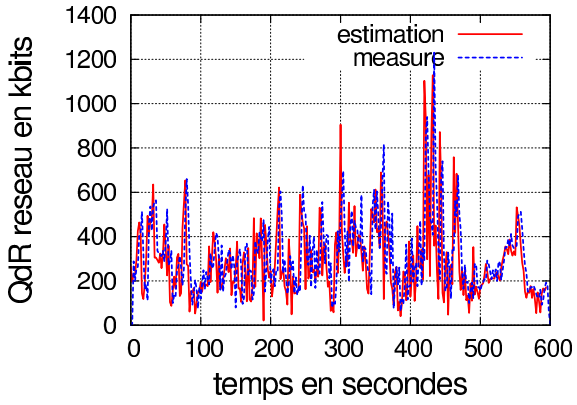


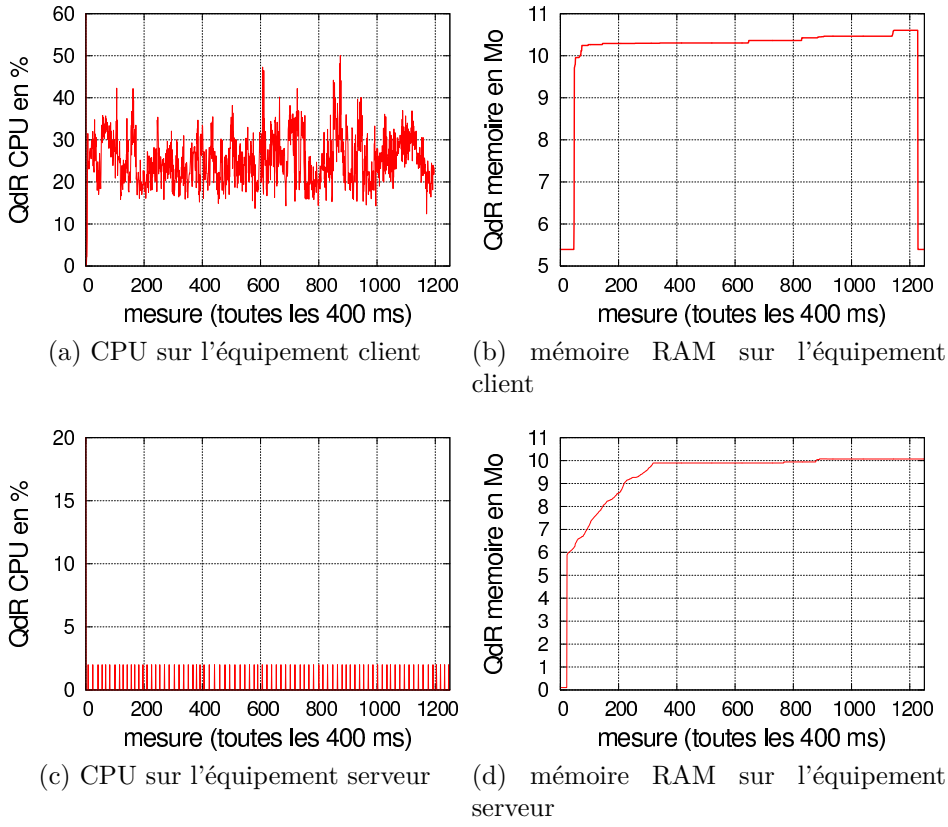
FIGURE 3.2 – Comparaison du profil d'utilisation estimé avec le profil d'utilisation mesuré

La Figure 3.1b donne le profil d'utilisation avec une granularité d'une seconde ainsi calculé. Sur les vidéos dont nous disposons, la taille du flux audio est en moyenne dix fois moindre que la taille du flux vidéo. Il n'est donc pas considéré dans le profil d'utilisation des ressources réseau.

Pour valider l'estimation par analyse du flux, nous avons mesuré les profils d'utilisation réels des ressources réseau et comparé ces profils avec ceux estimés. La Figure 3.2 montre cette comparaison pour le scénario 1. Pour mesurer le profil d'utilisation, le scénario 1 est exécuté et aucune autre application n'est exécutée sur les équipements. La courbe *"measure"* (bleue, pointillé), sur la Figure 3.2, donne la mesure des données émises par l'équipement serveur. La mesure des données reçues par l'équipement client corrobore ces valeurs. La courbe *"estimation"* (rouge, continue), sur la Figure 3.2, donne le profil d'utilisation estimé. On peut observer que les deux courbes sont confondues. Des résultats similaires ont été obtenus sur différents flux, notamment des flux contenant des flux vidéo et audio. Cette expérience valide l'estimation du profil d'utilisation de la ressource réseau par analyse du flux vidéo.

3.1.1.2 Ressources CPU et mémoire RAM

Pour les ressources CPU et mémoire RAM, le profil d'utilisation est mesuré par simulation. Le logiciel, l'équipement et les ressources utilisées sur l'équipement client



(a) CPU sur l'équipement client

(b) mémoire RAM sur l'équipement client

(c) CPU sur l'équipement serveur

(d) mémoire RAM sur l'équipement serveur

FIGURE 3.3 – Profils d'utilisation des ressources CPU et Mémoire RAM pour le scénario 1

n'influent pas les QdR requises sur l'équipement serveur. Réciproquement, le logiciel, l'équipement et les ressources utilisées sur l'équipement client n'influent pas les QdR requises sur l'équipement serveur. Les profils d'utilisation sur l'équipement client et sur l'équipement serveur sont donc mesurés indépendemment.

Nous avons vu au chapitre 2 que Linux permet de gérer la QdR d'un ensemble de processus grâce au système des *cgroups*. Linux fournit en plus des outils pour connaître l'utilisation CPU d'un processus et l'utilisation mémoire RAM d'un *cgroup*. Pour mesurer le profil d'utilisation, une réservation égale à la capacité de la ressource est effectuée sur chaque ressource utilisée par l'application. Les mesures fournies par ces outils sont cohérentes avec les mécanismes de réservation utilisés (détaillés au chapitre 4). Ces profils ne nécessitent donc pas de modification.

La Figure 3.3 donne les profils d'utilisation des ressources CPU et mémoire RAM, mesurés sur l'équipement client et sur l'équipement serveur, lors de l'exécution du scénario 1. L'intervalle de mesure de l'utilisation de chaque ressource est de 400 millisecondes. Les QdR CPU sont exprimées en pourcentage et les QdR mémoire RAM sont exprimées en méga-octets (Mo).

On peut noter que les profils d'utilisation du CPU sont fluctuants. Cette observation s'explique par la différence d'information à traiter entre différentes images. En effet, certaines images nécessitent plus de traitement pour être décodées et donc plus de CPU. Cette différence s'explique par la taille variable des images (images de référence de type I, images de type P et B non complètes, encodées à l'aide d'autres

images) et par un encodage différent des images. Sur le profil d'utilisation du CPU, on observe un pic d'utilisation lors de la première mesure (80% sur l'équipement client et 20% sur l'équipement serveur). Ce pic correspond au chargement du binaire. Si le CPU est sollicité par ailleurs, le binaire met quelques millisecondes de plus à se charger. Le début de l'application est retardé d'autant, ce qui n'impacte pas la QdS durant la diffusion. C'est pourquoi ce pic n'est pas considéré dans le profil d'utilisation.

Le profil d'utilisation de la mémoire RAM est plus stable du fait que le temps d'allocation de la mémoire est plus important que le temps d'accès. Pour optimiser le décodage vidéo, les logiciels clients évitent d'allouer la mémoire en permanence. Ainsi toutes les allocations ou presque sont effectuées quand le logiciel client commence à recevoir le flux.

3.1.2 D'un profil d'utilisation à une quantité de ressources requise

Pour simplifier la gestion des QdR, les profils d'utilisation sont abstraits par des valeurs statistiques représentant les QdR requises sur chaque ressource. L'abstraction des profils d'utilisation, estimés ou mesurés, par une QdR requise est un compromis entre la garantie de la QdS et la sur-réservation. Pour garantir la QdS, le maximum de QdR utilisé est réservé pour toute la durée de l'application. Cependant ce choix introduit de la sur-réservation, car l'application n'utilise pas toujours le maximum de QdR requise. La sur-réservation est la différence entre la réservation effectuée et l'utilisation réelle de la ressource. Afin de réduire la sur-réservation, d'autres approches sont possibles. Le choix d'une valeur de QdR requise est bien entendu fonction du type de ressources.

Pour la mémoire RAM, le profil d'utilisation est stable. La valeur maximale du profil d'utilisation est donc utilisée pour garantir la QdS, en introduisant une sur-réservation négligeable. Nous expliquons maintenant nos choix concernant les ressources réseau et CPU.

3.1.2.1 Ressources réseau

Pour diminuer la sur-réservation sur les ressources réseau, nous utilisons deux réservations pour une application de diffusion de contenus multimédia. Une réservation pour les pics d'utilisation qui est partagée avec d'autres applications, et une réservation propre à l'application de diffusion de contenus multimédia sans les pics d'utilisation. Pour les ressources réseau, la QdR requise s'exprime à l'aide de deux valeurs de bande passante, exprimées en $kbits/s$:

- `maxQoR` : le maximum du profil d'utilisation
- `statMaxQoR` : le maximum du profil d'utilisation, auquel sont retirés les pics.

Pour déterminer les pics d'utilisation, nous nous sommes basé sur l'inégalité de Bienaymé–Chebyshev [40] selon laquelle, quelque soit la loi de distribution, 89% des valeurs sont inférieures à $mean+3\sigma$; $mean$ étant la moyenne arithmétique et σ étant l'écart type des QdR du profil d'utilisation. L'inégalité de Bienaymé–Chebyshev est utilisée car nous ne faisons pas d'hypothèse sur la distribution des valeurs dans un profil d'utilisation des ressources réseau. `statMaxQoR` est donc donné par l'équa-

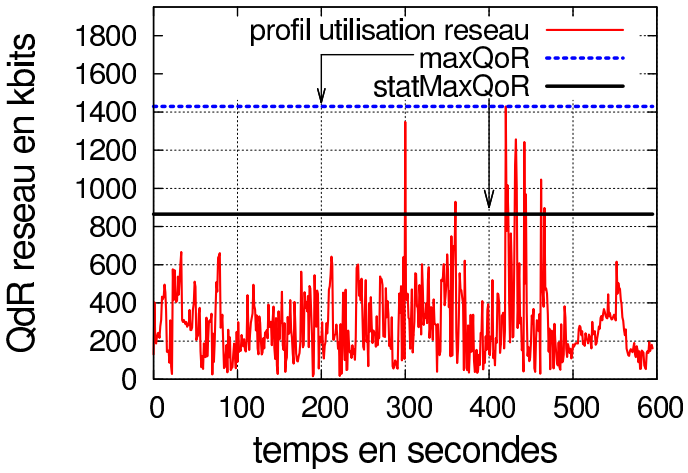


FIGURE 3.4 – Profil d'utilisation réseau et QdR requise

tion (3.2).

$$statMaxQoR = mean + 3\sigma \tag{3.2}$$

La Figure 3.4 montre le profil d'utilisation des ressources réseau pour le scénario 1 et les valeurs utilisées pour exprimer la QdR requise. La droite bleue, en pointillé, donne la valeur `maxQoR` et la droite noire continue donne la valeur `statMaxQoR`. Nous verrons au chapitre 4 comment ces deux valeurs sont utilisées pour réserver les ressources réseau.

En plus de la QdR requise sur un lien, des contraintes sont exprimées par des bornes sur les caractéristiques de QdS réseau (délai, gigue et taux de perte). Les contraintes sur le délai et le taux de perte du lien sont considérées comme connues. Par exemple, le délai doit être inférieur à 100 millisecondes selon [83]. La gigue peut être compensée par le logiciel client à l'aide d'un tampon mémoire. Dans le réseau local domestique, de nombreux logiciels peuvent être utilisés. De ce fait, nous ne faisons pas d'hypothèse sur le logiciel client. Nous considérons que la contrainte sur la gigue est aussi connue, par exemple deux fois la valeur du délai, soit 200 millisecondes. Dans ce cas, nous supposons que le logiciel client est capable de compenser une gigue d'au moins 200 millisecondes.

3.1.2.2 Ressources CPU

Contrairement aux ressources réseau, les mécanismes de réservation de Linux ne permettent pas d'exploiter la valeur `statMaxQoR` pour le CPU. De ce fait, seule la valeur `maxQoR` est utilisée.

Les QdR CPU sont exprimées à l'aide d'un QdR CPU et d'une période : $(maxQoR_i, T_i)$. Le logiciel client peut avoir un ou deux *threads* pour décoder le flux vidéo et le flux audio (si ce dernier est présent). Pour le flux vidéo, la période T_i est calculée en fonction de la fréquence d'images ($frame\_rate_i$) :

$$T_i = \frac{1}{frame\_rate_i} \tag{3.3}$$

Les contraintes de QdS sur le décodage du flux audio sont liées à la synchronisation entre le flux audio et le flux vidéo. Ces contraintes de QdS sont satisfaites si

S	CPU (en (%,ms))	mémoire RAM (en Mo)
1	(2,40)	10

(a) QdR requises sur l'équipement serveur

S	CPU (en (%,ms))	mémoire RAM (en Mo)
1	(45,40)	11

(b) QdR requises sur l'équipement client

S	`maxQoR` (en Kbits/s)	`statMaxQoR` (en Kbits/s)	Contraintes QdS réseau
1	1429	865	$délai < 100ms$ $gigue < 200ms$ $taux\ perte < 0.20\%$

(c) QdR réseau (liens et interfaces réseau)

TABLE 3.2 – QdR requises *(S. scénario)*

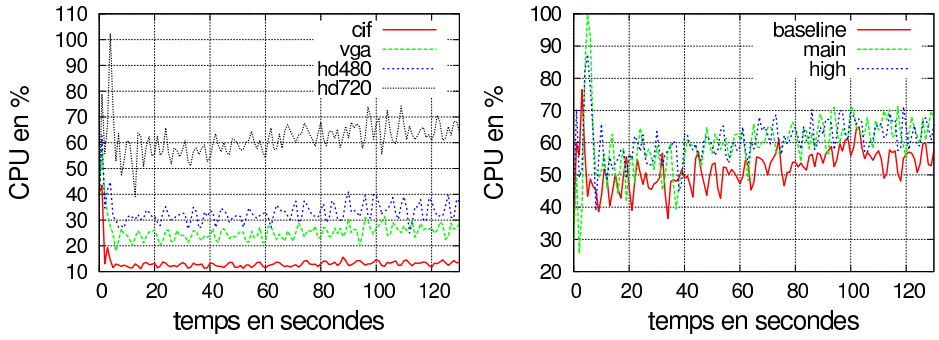
le logiciel client a assez de QdR CPU pour décoder les deux flux. La mesure du profil d'utilisation est réalisée au niveau du processus contenant le *thread* de décodage du flux vidéo et le *thread* de décodage du flux audio. La valeur `maxQoR` et la période de décodage du flux vidéo sont donc utilisées pour le processus client. Le processus serveur utilise la même période, pour garantir les contraintes de QdS sur l'équipement client.

Le Tableau 3.2 récapitule toutes les QdR requises et les contraintes de QdS réseau calculées pour l'exemple d'exécution du scénario 1.

3.2 Stockage et agrégation des QdR requises

La mesure et le stockage des QdR requises (CPU et mémoire RAM) sont rendus difficiles par l'hétérogénéité des flux et des équipements du réseau local domestique. Nous avons vu (section 2.1.3.5) que le nombre de scénarios possibles est très important dans le réseau local domestique. Pour utiliser un *framework* de gestion des QdR dans un tel environnement, il faut limiter le coût des mesures et du stockage. Il est difficile de chiffrer le coût d'une mesure. Ce coût est lié au temps de mesure et à la nécessité de disposer des équipements pour faire la mesure. Pour le stockage, le coût est lié à la quantité d'information à stocker. En effet, cette quantité influe sur la taille de la base de données utilisée et sur le temps d'accès à cette base de données.

Nous proposons d'agréger les QdR requises pour réduire la quantité d'information à stocker. Cette agrégation a aussi l'avantage de produire un nombre limité de valeurs "références" pour un flux donné. Nous montrons tout d'abord deux exemples de paramètres du scénario influant sur la QdR CPU requise. Nous détaillons ensuite le principe d'agrégation proposé, que nous illustrons sur ces exemples. Enfin, nous présentons deux algorithmes d'agrégation fournissant automatiquement le meilleur compromis d'agrégation, à partir de critères liés à la sur-réservation qui sont donnés en paramètres.



(a) taille des images (b) profil d'encodage

FIGURE 3.5 – Impact des paramètres d'encodage sur l'utilisation du CPU

3.2.1 Influence des paramètres d'encodage du flux et des ressources utilisées sur les QdR requises

Dans cette section, nous mesurons les profils d'utilisation pour différents paramètres d'encodage du flux et différentes ressources CPU. Ces exemples illustrent l'influence des paramètres d'un scénario sur le profil d'utilisation des ressources et donc sur les QdR requises.

3.2.1.1 Influence des paramètres d'encodage du flux

Pour illustrer l'influence des paramètres d'encodage, nous nous intéressons à deux paramètres : la taille des images et le profil du standard *h.264*.

La Figure 3.5 montre les profils d'utilisation du CPU, mesurés sur l'équipement client. Chaque courbe correspond à l'exécution du même scénario en changeant uniquement la taille des images du flux vidéo pour la Figure 3.5a et le profil d'encodage pour la Figure 3.5b. Quatre tailles d'images sont utilisées sur la Figure 3.5a (le profil d'encodage *main* est utilisé) :

- CIF : 352x288 pixels
- VGA : 640x480 pixels
- hd480 : 852x480 pixels
- hd720 : 1280x720 pixels.

Trois profils d'encodage du standard *h.264* classiquement utilisés sont mesurés sur la Figure 3.5b (la taille des images est hd720) :

- profil de base : *baseline*
- profil *main*
- profil de haute qualité : *high*.

Le profil *high* utilise des techniques de compression plus avancées que le profil *main*, lui-même utilisant des techniques de compression plus avancées que le profil de base (*baseline*).

Le Tableau 3.3 donne les QdR requises, calculées à partir des profils d'utilisation de la Figure 3.5. Seule la valeur `maxQoR` est donnée, car la période de l'application est identique (le flux a toujours la même fréquence d'images).

On observe sur ces deux exemples que si la valeur du paramètre augmente (images plus grandes, profil utilisant des techniques de compression plus avancées), alors la QdR requise augmente également. Pour la mémoire RAM, des observations

paramètre	**valeur**	**maxQoR (en %)**
taille images (profil *main*)	CIF	17
	VGA	35
	hd480	46
	hd720	76
profil (taille images hd720)	baseline	69
	main	76
	high	78

TABLE 3.3 – QdR CPU requise pour différents paramètres d'encodage

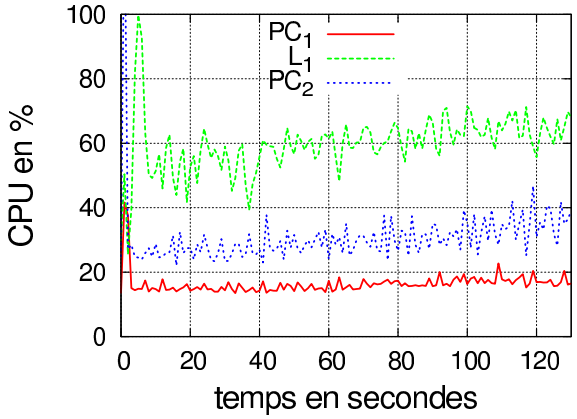


FIGURE 3.6 – Profils d'utilisation CPU pour différents équipements clients

identiques sont faites. Cette expérience a été réalisée sur plusieurs flux et les mêmes résultats ont été observés.

3.2.1.2 Influence des ressources utilisées

La Figure 3.6 donne les profils d'utilisation pour trois équipements clients différents. Entre chaque mesure (une courbe donne une mesure), seul l'équipement client change (le même flux avec les mêmes paramètres d'encodage est utilisé). Le Tableau 3.4 donne les QdR CPU requises sur l'équipement client, calculées à partir des profils d'utilisation présentés sur la Figure 3.6.

Contrairement aux paramètres, il est plus difficile de fournir une relation entre l'équipement utilisé et la QdR requise.

équipement	**maxQoR CPU (en %)**
PC_1	26
PC_2	54
L_1	69

TABLE 3.4 – QdR CPU requises sur l'équipement client

3.2.1.3 Conclusion

Les mesures présentées ici ont mis en évidence que la QdR requise sur les ressources CPU et mémoire RAM dépend :

– des ressources utilisées
– des paramètres d'encodage du flux.

Dans l'exemple détaillé, pour quatre tailles d'images, trois profils et trois équipements, $4 * 3 * 3 = 36$ valeurs doivent être connues. Ces valeurs fluctuent entre 17 et 78% pour la ressource CPU de l'équipement client.

Des expériences similaires ont mis en évidence que la QdR requise dépend aussi du contenu du flux et des ressources utilisées. Pour un même flux, de nombreuses valeurs de QdR requises doivent donc être connues.

3.2.2 Principe d'agrégation

Dans cette thèse, nous avons fait l'hypothèse qu'une base de données, contenant toutes les QdR requises pour tous les scénarios existe. Cette base de données est interrogée lorsqu'un utilisateur veut démarrer une application de diffusion de contenus multimédia. Cette base de données se compose de tables, appelées *tables de mapping* qui associent à un scénario, une QdR requise, sur une ressource (cf. chapitre 2).

Du fait du caractère hétérogène du réseau local domestique, il faut limiter le coût des mesures et du nombre d'information d'une telle base de données. Pour ce faire, nous proposons d'agréger les tables de mapping, en regroupant les QdR requises "similaires" afin de former des tables de mapping agrégées de tailles inférieures. Dans ces tables, les QdR requises "similaires" sont remplacées par une seule valeur appelée `aggMaxQoR`, représentative de l'ensemble des valeurs.

L'agrégation est utilisée pour une QdR requise s'exprimant par `maxQoR`, contenant la valeur maximale du profil d'utilisation. Pour conserver cette propriété, la valeur `aggMaxQoR` est fixée au maximum de toutes les QdR requises.

Nous détaillons d'abord le compromis introduit par l'agrégation des tables de mapping. Ensuite, nous illustrons son intérêt sur deux exemples : une agrégation sur les ressources CPU des équipements clients et une agrégation sur les paramètres d'encodage du flux.

3.2.2.1 Compromis d'agrégation

L'objectif de l'agrégation est de réduire la taille des tables de mapping. Cependant, regrouper les QdR requises et utiliser `aggMaxQoR` à la place de `maxQoR` introduit une "perte" de QdR disponible. On parle de perte car la QdR ne peut pas être réservée et elle n'est pas utilisée par l'application.

La perte sur une ressource i, introduite par une table de mapping agrégée A_k, se calcule à l'aide de l'équation (3.4), où $aggMaxQoR_{ik}$ est la valeur utilisée comme QdR requise sur la ressource i, dans la table de mapping agrégée A_k.

$$perte_{ik} = aggMaxQoR_{ik} - maxQoR_i \tag{3.4}$$

La perte engendrée par une table de mapping agrégée A_k contient l'ensemble des pertes engendrées sur chaque ressource. L'équation (3.5) donne la perte pour une

table de mapping agrégée A_k, regroupant n QdR requises.

$$perte_{A_k} = \cup_{i=1}^{n}(perte_{ik}) \tag{3.5}$$

L'agrégation d'une table de mapping fournit un compromis entre la taille de la table de mapping agrégée et la perte qu'elle génère. Pour évaluer la perte générée par une table de mapping agrégée A_k, trois critères sont utilisés :

- la moyenne des pertes ($mean(perte_{A_k})$) : les demandes de démarrages sont considérées comme équiprobables, la moyenne des pertes introduites pour chaque démarrage est donc utilisée ;
- la perte maximale ($max(perte_{A_k})$) : la perte maximale permet de borner la perte autorisée pour une table de mapping agrégée ;
- le nombre de faux négatifs introduits par A_k ($nbFaNeg(A_k)$) : un faux négatif correspond à un échec du contrôle d'admission en utilisant `aggMaxQoR` au lieu de `maxQoR`. Si le test d'admission avait été réalisé avec `maxQoR`, il aurait réussi. Comme `aggMaxQoR` est le maximum des `maxQoR`, aucun faux positif n'est généré. Si le test d'admission réussit avec `aggMaxQoR`, il réussit aussi avec `maxQoR`.

3.2.2.2 Illustration sur l'agrégation des ressources utilisées

Le Tableau 3.5 illustre le principe d'agrégation sur les ressources CPU utilisées sur l'équipement client. Ce tableau donne une table la mapping initiale pour la ressource CPU, avec 5 équipements clients et deux tables de mapping agrégées construites à partir de la table initiale. Seul l'équipement client change entre les 5 scénarios. La table de mapping initiale (3.5a) contient la QdR CPU requise sur chaque équipement client. Les tableaux 3.5b et 3.5c présentent deux tables de mapping agrégées possibles (A_1 et A_2), construites à partir de la Table 3.5a. Les Tableaux 3.5b et 3.5c donnent les QdR agrégées (`aggMaxQoR`) qui sont utilisées comme QdR requises lors d'une demande de démarrage. Par exemple, la QdR requise sur le PC_1 est de 26% si la table A_1 est utilisée, et de 69% si la table A_2 est utilisée.

La table de mapping agrégée A_2 a une taille inférieure à la table A_1, car la stratégie d'agrégation est plus agressive : une seule `aggMaxQoR` est utilisée. En revanche, la perte introduite par la table A_2 est plus importante. Le Tableau 3.6 compare les deux tables de mapping agrégées avec les critères d'évaluation précédemment définis. La colonne perte donne la perte introduite sur chaque ressource. La taille représente le nombre d'`aggMaxQoR` à stocker.

L'agrégation sur les ressources utilisées diminue le nombre de QdR requises à stocker mais pas le nombre de QdR requises à mesurer. De plus l'information donnant la ressource utilisée est conservée dans la table de mapping agrégée.

3.2.2.3 Illustration sur l'agrégation des paramètres d'encodage

Le Tableau 3.7 illustre le principe d'agrégation sur le paramètre taille d'images. Comme précédemment, ce tableau contient la table de mapping initiale (3.7a) et deux tables de mapping agrégées possibles (A_3 et A_4). La comparaison des tables de mapping agrégées étant identique à l'exemple 3.2.2.2 elle n'est pas détaillée à nouveau.

équipement client	`maxQoR` (en %)
STB_1	12
PC_1	26
PC_2	54
$Tablette_1$	25
L_1	69

(a) Table de mapping initiale pour les QdR CPU requises sur l'équipement client

équipement client	`aggMaxQoR` (en %)
$STB_1, PC_1, Tablette_1$	26
L_1, PC_2	69

(b) table de mapping agrégée A_1

équipement client	`aggMaxQoR` (en %)
$STB_1, PC_1, PC_2,$ $Tablette_1, L_1$	69

(c) table de mapping agrégée A_2

TABLE 3.5 – Exemple d'une table de mapping et de tables de mapping agrégées

table	**perte**	**max** ($\mathbf{perte_{A_k}}$)	**mean** ($\mathbf{perte_{A_k}}$)	**nbFaNeg** ($\mathbf{A_k}$)	**taille**
A_1	$\{14, 1, 0, 14, 0\}$	14	5.8	0	2
A_2	$\{56, 42, 14, 43, 0\}$	56	31	0	1

TABLE 3.6 – Comparaison des tables de mapping agrégées

taille images	`maxQoR` (en %)
CIF	17
VGA	35
hd480	46
hd720	76

(a) Table de mapping pour les QdR CPU requises sur l'équipement client

taille images	`aggMaxQoR` (en %)
VGA	35
hd720	76

(b) table de mapping agrégée A_3

taille images	`aggMaxQoR` (en %)
hd720	76

(c) table de mapping agrégée A_4

TABLE 3.7 – Exemple d'une table de mapping et de tables de mapping agrégées

Cependant, contrairement à l'exemple 3.2.2.2, les valeurs des paramètres sont ordonnées ($CIF < VGA < hd480 < hd720$). C'est pourquoi les tables de mapping agrégées ne contiennent qu'une seule valeur de paramètre, associée à une `aggMaxQoR`. Lors de la phase de démarrage d'une application, la QdR requise est celle de la valeur supérieure la plus proche dans la table. Par exemple, pour la table A_3, la valeur 35 est utilisée pour la taille d'images CIF ($CIF < VGA$). Cette propriété diminue plus efficacement la taille de la table de mapping agrégée.

En outre, l'agrégation sur les paramètres d'encodage réduit le nombre de mesures à réaliser. En effet, en se basant sur l'ordre des paramètres, seules les mesures supérieures sont nécessaires. Par contre, si une valeur est utilisée sans avoir été mesurée, puis agrégée, la perte introduite et les éventuels faux négatifs générés ne peuvent pas être connus.

3.2.3 Algorithmes d'agrégation

Pour automatiser la création de tables de mapping agrégées, à partir des tables de mapping existantes, nous proposons deux algorithmes d'agrégation. L'objectif de ces algorithmes est de trouver la meilleure agrégation possible en fonction de paramètres donnés par un fournisseur de contenu ou d'équipement. Pour évaluer les tables de mapping agrégées, des bornes sur chaque critère d'évaluation sont données. Les algorithmes utilisent la table de mapping à agréger et les trois paramètres suivants :

- *boundMaxLossQoR* : borne sur la perte maximale sur une ressource, $boundMaxLossQoR \geq max(perte_{A_k})$
- *boundMeanLossQoR* : borne sur la perte moyenne d'une table de mapping agrégée, $boundMeanLossQoR \geq mean(perte_{A_k})$
- *boundNbFaNeg* : borne sur le nombre de faux négatifs générés par une table de mapping agrégée, $boundNbFaNeg \geq nbFaNeg(A_k)$.

Le nombre de faux négatifs générés par la table de mapping agrégée dépend des réservations qui sont en place lors d'une demande de démarrage. Ces réservations ne sont pas connues au moment de l'agrégation. Les algorithmes peuvent cependant vérifier que l'`aggMaxQoR` utilisée pour une ressource n'est pas supérieure à la capacité maximale offerte par la ressource. Dans ce cas, un faux négatif serait toujours généré.

Une agrégation est une partition de l'ensemble des QdR requises. Le nombre de partitions dans un ensemble est connu comme le nombre de Bell et croit exponentiellement [103]. Les algorithmes d'agrégation ne peuvent donc pas évaluer toutes les agrégations possibles pour choisir la meilleure.

Les deux algorithmes proposés se différencient par les concepts mathématiques qu'ils utilisent pour modéliser l'agrégation : le *clustering* et le *bin-packing*.

3.2.3.1 Algorithme basé sur le *clustering*

Le *clustering* est l'affectation d'un ensemble d'observations dans un sous-ensemble (appelé *cluster*) de telle manière que les observations dans un *cluster* soient similaires. Dans le cadre de l'agrégation proposée ici, les tables de mapping agrégées peuvent être vues comme des *clusters* et les QdR requises comme des observations. La similarité entre deux QdR requises est la différence absolue entre ces deux valeurs.

Notre premier algorithme d'agrégation [73], exploite l'algorithme de *clustering k-means* [47]. L'algorithme *k-means* prend en entrée le nombre de *clusters* à produire.

Le nombre de *clusters* correspond à la taille de la table de mapping agrégée produite par l'algorithme. Or, c'est ce critère que l'on cherche à minimiser. L'utilisation de *k-means* permet d'orienter l'algorithme de *clustering* pour trouver la solution la plus efficace en lui donnant en entrée le nombre désiré de *clusters*.

Le Listing 3.1 présente l'algorithme d'agrégation écrit en python [2].

```
def testCriteriaMet(A,boundMaxLossQoR,boundMeanLossQoR,boundNbFaNeg):
  return (A.getMaxLoss()<=boundMaxLossQoR
          and A.getMeanLoss()<=boundMeanLossQoR
          and A.getNbFaNega()<=boundNbFaNeg)

def findBestAggMapTable(boundMaxLossQoR,boundMeanLossQoR,boundNbFaNeg,
    req_qor_set):
  cl_kmeans = KMeansClustering(req_qor_set)
  # create an aggregated mapping table with only one aggMaxQoR
  nbCluster=1
  clusters=getAllRequiredQoR()
  A=Aggregated_map_table("A"+str(nbCluster),clusters)
  nbCluster=2
  while (not testCriteriaMet(A,boundMaxLossQoR,boundMeanLossQoR,
      boundNbFaNeg)):
      nbCluster=nbCluster+1
      clusters = cl_kmeans.getclusters(nbCluster)
      # build a new aggMapTable
      A=Aggregated_map_table("A"+str(nbCluster),clusters)
```

Listing 3.1 – Algorithme d'agrégation basé sur le *clustering* (écrit en python)

L'algorithme commence par générer une table de mapping agrégée contenant toutes les QdR requises. Si cette table ne satisfait par les critères d'entrée, l'algorithme génère une nouvelle table de mapping agrégée, de plus grande taille, jusqu'à trouver une table qui correspond aux critères d'entrée (lignes 13 à 17). Les tables de mapping agrégées sont générées à l'aide de l'algorithme *k-means* [3], comme le montre la ligne 15.

Les tables de mapping agrégées sont évaluées par la fonction *testCriteriaMet*, ligne 1. Cette fonction calcule la perte maximale et moyenne et le nombre de faux négatifs générés par la table de mapping agrégée. Si un des critères n'est pas satisfait, la fonction *testCriteriaMet* retourne `faux` ce qui déclenche la génération d'une nouvelle table de mapping agrégée.

3.2.3.2 Algorithme basé sur le *bin-packing*

Un problème de *bin-packing* est classiquement défini de la manière suivante :

> *Combien de bins (sacs) sont nécessaires pour stocker un ensemble d'éléments, sachant que la contenance des sacs est limitée par une constante et que chaque élément a son volume propre.*

En suivant ce principe, un autre algorithme d'agrégation a été implémenté. L'algorithme est donné à la Figure 3.7.

2. `http://www.python.org/`
3. l'implémentation python du *k-means*, disponible à `http://sourceforge.net/projects/python-cluster/` a été utilisée

$algoBinPacking(req\_qor\_set, boundMaxLossQoR, boundMeanLossQoR, boundNbFaNeg)$

$k \leftarrow 1$
trier req_qor_set dans l'ordre croissant des valeurs $reqQoR_i$
$\forall e_i \in req\_qor\_set$
  inclure e_i dans B_k
  calculer $aggMaxQoR(B_k) = max_{e_i \in B_k}\{reqQoR_i\}$ et vérifier les contraintes :
    $\forall i, 1 \leq i \leq n, \quad loss_i \leq boundMaxLossQoR$
    $\forall k, 1 \leq i \leq capacité_i, \quad \frac{1}{|B_k|} \sum_{e_i \in B_k} loss_i \quad \leq \quad boundMeanLossQoR$
    $| \{e_i \in B_k \ \wedge \ aggMaxQoR(B_k) > capacité_i\} | \leq boundNbFaNeg$
  si l'une d'elle n'est pas satisfaite :
    retirer l'élement e_i de B_k
    $k \leftarrow k + 1$
    inclure e_i dans B_k
  finsi
fin $\forall$

FIGURE 3.7 – algorithme d'agrégation basé sur le *bin-packing*

Dans cet algorithme, les tables de mapping agrégées sont vues comme des sacs. L'objectif de l'algorithme est de remplir toutes les QdR requises dans un sac, en utilisant le moins de sac possible. Les paramètres d'évaluation d'une table de mapping agrégée modélisent la contenance des sacs.

L'algorithme commence par trier la table de mapping pour qu'elle soit ordonnée sur les QdR requises. L'ensemble req_qor_set contient un couple $(reqQoR_i, capacité_i)$ pour chaque scénario à agréger. $reqQoR_i$ est la QdR requise sur la ressource i et $capacité_i$ est la capacité de la ressource i. L'algorithme ajoute ensuite une nouvelle QdR requise au sac (appelé B_k) en court de remplissage. L'algorithme vérifie que les bornes autorisées de perte et de faux négatifs pour une table de mapping agrégés ne sont pas dépassés. Pour cela l'algorithme calcule la perte générée par le sac appelée *loss*. L'algorithme vérifie ensuite que :

- la perte est inférieure à la borne $boundMaxLossQoR$ pour les n éléments du sac
- la moyenne des pertes est inférieure à la borne $boundMeanLossQoR$. La moyenne des pertes est calculée uniquement sur les éléments ne générant pas de faux négatifs
- le nombre de faux négatifs est inférieur à la borne $boundNbFaNeg$.

Si l'ajout de cette QdR requise ne permet plus de respecter les seuils autorisés alors la QdR requise est retirée du sac et un nouveau sac est rempli.

3.2.3.3 Comparaison des deux algorithmes

L'évaluation des deux algorithmes sera détaillée dans le chapitre 5. Pour illustration, nous donnons les résultats obtenus, pour l'exemple détaillé dans le Tableau 3.5. Trois tests sont réalisés pour chaque algorithme.

Le premier test utilise les paramètres :

- $boundMaxLossQoR = 200$
- $boundMeanLossQoR = 200$

– $boundNbFaNeg = 5$

Ces paramètres autorisent une agrégation très agressive : les capacités des équipements sont de 100 ou 200 et il y a cinq équipements. Les deux algorithmes retrouvent la table de mapping agrégée A_2 contenant une seule `aggMaxQoR` donnée dans le Tableau 3.5c.

Le deuxième test utilise les paramètres :

– $boundMaxLossQoR = 20$

– $boundMeanLossQoR = 10$

– $boundNbFaNeg = 0$

Ces paramètres autorisent moins de perte mais permettent tout de même d'agréger les QdR requises. En effet, les deux algorithmes retrouvent la table de mapping agrégée A_1 contenant deux `aggMaxQoR` (donnée dans le Tableau 3.5b)

Le dernier test utilise les paramètres :

– $boundMaxLossQoR = 0$

– $boundMeanLossQoR = 0$

– $boundNbFaNeg = 0$

Ces paramètres n'autorisent aucune perte. Les deux algorithmes donnent une table de mapping agrégée contenant cinq `aggMaxQoR`, c'est-à-dire sans aucun regroupement.

3.3 Conclusion

Dans ce chapitre, nous avons montré comment les QdR requises par une application de diffusion de contenus multimédia sont connues. Les QdR requises sur les ressources réseau sont obtenues en analysant le flux vidéo à diffuser. Pour les ressources CPU et mémoire RAM une mesure est réalisée pour connaître les QdR requises sur chaque ressource. Les valeurs de QdR requises retenues sont des bornes supérieures sur l'utilisation des ressources d'une application, dans le but de garantir la QdS de cette application.

Les QdR requises sur les ressources CPU et mémoire RAM dépendent de nombreux paramètres. Effectuer une mesure et stocker la QdR requise pour chaque valeur de paramètre peut s'avérer prohibitif dans le réseau local domestique. De ce fait, nous avons proposé dans ce chapitre, d'agréger les QdR requises similaires afin de réduire la quantité d'information à mesurer et à stocker. Pour automatiser ce processus, nous avons développé deux algorithmes d'agrégation. Ces algorithmes utilisent des concepts différents (*clustering* et *bin-packing*) mais ont la même fonction. Ils génèrent la table de mapping agrégée la plus petite possible respectant des bornes de perte passées en paramètres.

Les propositions de ce chapitre seront évaluées dans le chapitre 5. Avant cela, dans le chapitre suivant, nous regardons comment les QdR requises par une application de diffusion de contenus multimédia sont réservées afin de garantir la QdS utilisateur.

Chapitre 4

ARMOR, un *framework* de gestion des quantités de ressources

Sommaire

Dans ce chapitre, nous présentons ARMOR, un *framework* de gestion des quantités de ressources (QdR). ARMOR fournit un cadre d'exécution, garantissant les QdR requises par des applications de diffusion de contenus multimédia.

Une fois les composants d'ARMOR initialisés, des applications de diffusion de contenus multimédia peuvent être démarrées. Pour garantir la Qualité de Service (QdS) de ces applications, ARMOR s'assure préalablement que les QdR disponibles sur les ressources utilisées sont suffisantes. Si c'est le cas, ARMOR met en place une réservation sur chaque ressource. Ces réservations, fournies à l'aide des mécanismes du système d'exploitation, garantissent l'ensemble des QdR à l'application. Ainsi, grâce à ARMOR la QdS utilisateur est satisfaite.

Dans la première partie de ce chapitre, nous détaillons les composants d'ARMOR et comment ils s'interconnectent. Puis, nous montrons la pertinence de ces composants pour le domaine d'étude de cette thèse : le réseau local domestique. Nous décrivons ensuite les phases d'initialisation, de démarrage d'une application multimédia et de suppression des réservations effectuées pour une application multimédia.

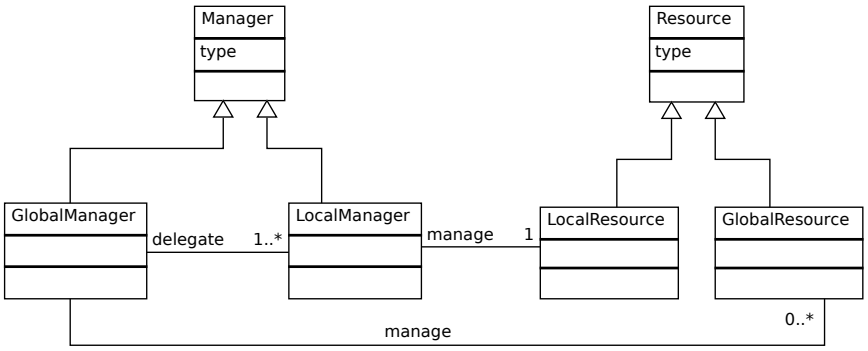


FIGURE 4.1 – Modèle conceptuel d'ARMOR

4.1 Architecture d'ARMOR

ARMOR utilise une architecture avec deux niveaux : global et local. Les composants du niveau global gèrent les QdR des ressources partagées par les équipements et délèguent la gestion des QdR sur les ressources locales. Les composants du niveau local effectuent les traitements, demandés par le niveau global, pour gérer les QdR des ressources d'un équipement. Nous détaillons d'abord le modèle et les composants d'ARMOR, puis les interfaces de ces composants. Ensuite, nous montrons comment ARMOR prend en compte les caractéristiques du réseau local domestique.

4.1.1 Modèle et composants d'ARMOR

La Figure 4.1 présente le modèle conceptuel d'ARMOR sous la forme d'un diagramme de classes UML. Une ressource a un `type` et est locale à un équipement (`LocalResource`) ou est partagée entre plusieurs équipements (`GlobalResource`). Les QdR d'une ressource sont gérées par un manager. Un manager a un `type` et est soit local (`LocalManager`), soit global (`GlobalManager`). Un manager local gère une ressource locale du même type. Un manager global coordonne la gestion de plusieurs ressources locales du même type. Pour ce faire, il délègue la gestion de chaque ressource au manager local correspond. Un manager global peut aussi gérer des ressources globales du même type.

Pour implémenter ce modèle, ARMOR se base sur des composants logiciels. Un composant de gestion des ressources locales (`Local Resources Manager` LRM) gère toutes les ressources d'un équipement. Un LRM est un composite contenant un `LocalManager` par type de ressources. ARMOR utilise une architecture centralisée. Le point central est un composite contenant tous les `GlobalManager`, ce composant est appelé `Global Resources Manager` (GRM). Une instance unique du composant GRM est installée dans le réseau local domestique. Toutes les demandes faites à ARMOR passent par le composant GRM. Ces demandes sont effectuées par un composant console. L'instance du composant GRM étant le point central de l'architecture, elle doit être accessible en permanence par un composant console. Il doit donc y avoir au moins une instance de composant console. Si plusieurs instances sont présentes, les demandes sont traitées, au niveau du GRM, selon la politique premier arrivé, premier servi.

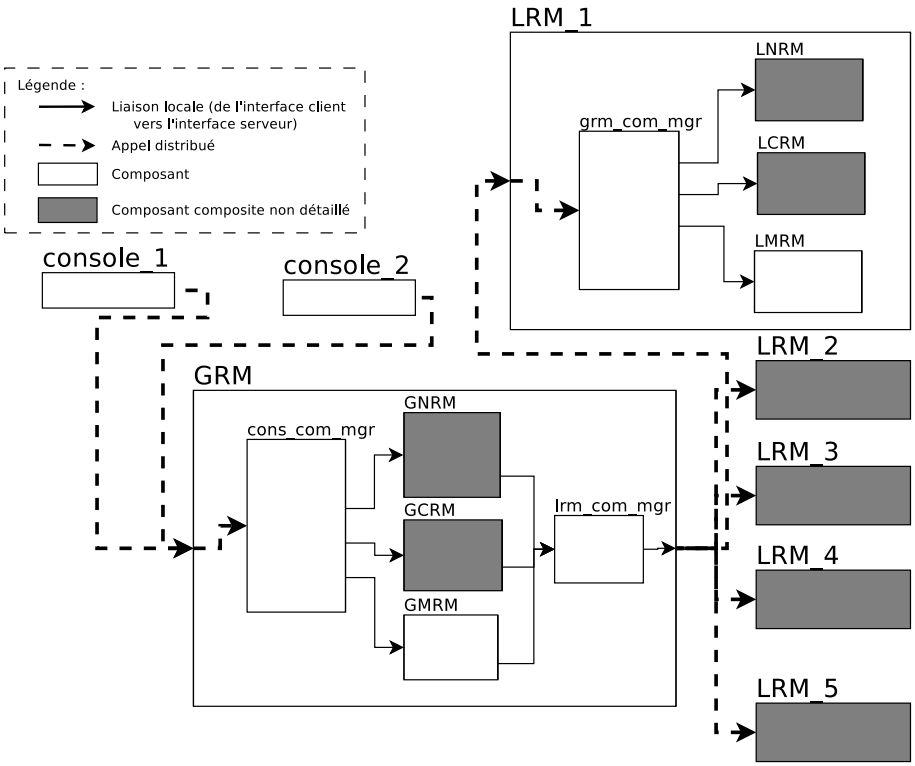


FIGURE 4.2 – Exemple d'une instance d'ARMOR

La Figure 4.2 donne une instance d'ARMOR pour un réseau local domestique contenant cinq équipements. Une instance du composant LRM est installée sur chaque équipement (LRM_1 à LRM_5). L'instance unique du GRM peut être installée sur n'importe quel équipement. Enfin, deux instances de composant console sont installées sur deux équipements (console_1 et console_2), pour communiquer avec ARMOR via le GRM. Les éléments grisés sur la figure sont des composites qui sont détaillés à la section 4.2.

Dans sa version actuelle, ARMOR supporte la gestion des types de ressources CPU, mémoire RAM et réseau. Pour chaque type de ressources, ARMOR utilise un composant manager global et un composant manager local sur chaque équipement :

- les ressources CPU sont gérées par les composants `Global CPU Resources Manager` (GCRM) et `Local CPU Resource Manager` (LCRM) ;
- les ressources mémoire RAM sont gérées par les composants `Global Memory Resources Manager` (GMRM) et `Local Memory Resource Manager` (LMRM) ;
- les ressources réseau sont gérées par les composants `Global Network Resources Manager` (GNRM) et `Local Network Resources Manager` (LNRM).

Les ressources CPU et mémoire RAM sont locales à un équipement. Les composants GCRM et GMRM coordonnent la gestion de ces ressources, en délégant aux managers locaux. Pour les ressources réseau, les liens sont partagés entre les équipements et les interfaces sont locales à un équipement. Le composant GNRM assure la gestion des liens, et coordonne en plus la gestion des interfaces réseau, en la délégant aux managers locaux.

	initialisation	**controleAdmission**	**réservation**	**suppression**
réseau	capacité	$maxQoR$ $statMaxQoR$	$maxQoR$ $statMaxQoR$	idReservation
CPU	capacité, ordonnanceur	$(maxQoR, T)$	$(maxQoR, T)$	idReservation
mémoire RAM	capacité	$maxQoR$	$maxQoR$	idReservation

TABLE 4.1 – Paramètres des méthodes des composants locaux de gestion des QdR

4.1.2 Interfaces des composants d'ARMOR

ARMOR utilise trois phases de fonctionnement :
- initialisation : cette phase configure tous les composants d'ARMOR ;
- démarrage d'une application : cette phase traite les demandes de démarrage d'applications en deux étapes :
  - contrôle d'admission : vérifie que les QdR requises sont disponibles sur toutes les ressources utilisées ;
  - réservation : réserve les QdR requises, sur toutes les ressources utilisées.
- suppression : cette phase supprime toutes les réservations effectuées pour une application.

Ces trois phases sont déclenchées par une demande transmise à ARMOR via un composant console. Afin de masquer l'hétérogénéité des types de ressources, chaque composant manager global offre une interface générique. Cette interface est composée de quatre méthodes, invoquées selon le traitement en cours dans ARMOR :
- `initialisation()` : initialise le composant global et coordonne l'initialisation des composants locaux du même type ;
- `controleAdmission()` : vérifie les QdR requises, sur les ressources du type correspondant ;
- `réservation()` : réserve les QdR requises, sur les ressources du type correspondant ;
- `suppression(int idReservation)` : supprime les réservations effectuées pour une application, sur les ressources du type correspondant. L'ensemble des réservations effectuées pour une application est identifié par un numéro unique : `idReservation`.

Lorsqu'une méthode est invoquée au niveau global, elle invoque la méthode correspondante au niveau local. Lors de la phase d'initialisation, la méthode `initialisation()` est invoquée pour chaque instance de composant LRM existante. De même, lors d'une demande de démarrage les méthodes `controleAdmission()` et `réservation()` sont invoquées pour les instances de composant LRM installées sur les équipements utilisés par l'application. Quand l'application est terminée, la méthode `suppression(int idReservation)` est invoquée pour les instances de composant LRM installées sur les équipements utilisés par l'application.

Les méthodes des composants managers locaux sont spécifiques à chaque type de ressources. Le Tableau 4.1 donne les paramètres formels pour chacune des méthodes locales. Tous les paramètres sont des entiers. Le composant du niveau global sait exprimer les QdR requises pour le type de ressources qu'il gère. Par exemple pour la réservation de QdR CPU, le composant du niveau global GCRM invoque

la méthode `réservation(int maxQoR, int T)` du composant local LCRM avec les valeurs $(maxQoR, T)$ pour exprimer la QdR requise et la période de l'application.

L'invocation de méthodes entre les composants globaux et les composants locaux, situés sur des équipements distants, est assurée par les composants `lrm_com_mgr` (composant du GRM) et `grm_com_mgr` (composant du LRM). Le composant `lrm_com_mgr` effectue un appel de type *Remote Procedure Call* (RPC) en rajoutant le type de ressources concerné. Quand le `grm_com_mgr` est invoqué, l'invocation est déléguée au composant de gestion des QdR local du type correspondant. Les demandes de démarrage d'une application de diffusion de contenus multimédia et de suppression des réservations effectuées pour une application, provenant d'un composant console, sont traitées au niveau du GRM par le composant `cons_com_mgr`. Ce composant invoque séquentiellement, pour chaque type de ressources, la méthode `controleAdmission()` puis la méthode `réservation()` pour le démarrage d'une application. Pour la suppression des réservations effectuées pour une application, ce composant invoque séquentiellement, pour chaque type de ressources, les méthodes `suppression()`.

4.1.3 Utilisation d'ARMOR dans le réseau local domestique

Dans le réseau local domestique, il existe plusieurs types de ressources réseau (Wifi ou Ethernet), supportant ou non une norme de QdS réseau. Pour les ressources CPU, plusieurs types d'ordonnanceurs existent, gérant la QdR différemment. Afin de masquer cette hétérogénéité, ARMOR utilise les attributs des composants locaux pour les configurer lors de la phase d'initialisation. Les services offerts par ces composants sont ensuite identiques pour toutes les configurations. Les attributs des composants locaux sont aussi utilisés pour représenter les caractéristiques de QdS (`capacité`, `QdRdisponible` et caractéristiques de QdS réseau) des ressources du réseau local domestique (cf. Figure 2.8, page 23). Dans un réseau local domestique, il n'y a qu'un seul lien Wifi et plusieurs liens Ethernet. Pour simplifier la gestion des liens Ethernet, ARMOR se base sur des liens logiques. Enfin, pour les ressources réseau, ARMOR utilise une caractéristique de QdS appelée `QdRpartagée`.

Attributs architecturaux Dans la programmation à base de composants, les attributs architecturaux sont exposés par un composant et permettent notamment de le configurer. Dans ARMOR, les caractéristiques de QdS de chaque ressource sont intégrées dans un attribut du composant gérant les QdR de cette ressource. Les composants LCRM et LMRM ont donc un attribut `QdRdisponible` et `capacité`. Pour les ressources CPU, le LCRM a en plus un attribut `ordonnanceur`, identifiant l'ordonnanceur utilisé sur l'équipement. Sur chaque équipement, le composant LNRM gère toutes les interfaces réseau. Les attributs `QdRdisponible` et `capacité` sont intégrés dans les sous-composants du LNRM (détaillés à la section 4.2.1). Ces sous-composants ont aussi un attribut `normeQdS` identifiant la norme de QdS réseau supportée par l'interface.

Le Listing 4.1 donne un extrait du composant LMRM décrivant ses attributs, à l'aide du langage de description architecturale (*Architecture Description Language* ADL) du projet Mind[1]. Ce composant a deux attributs : *capacity* (`capacité`) et

1. `http://mind.ow2.org/`

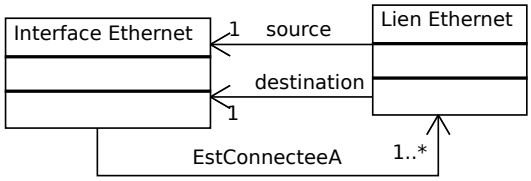


FIGURE 4.3 – Modèle de liens logiques

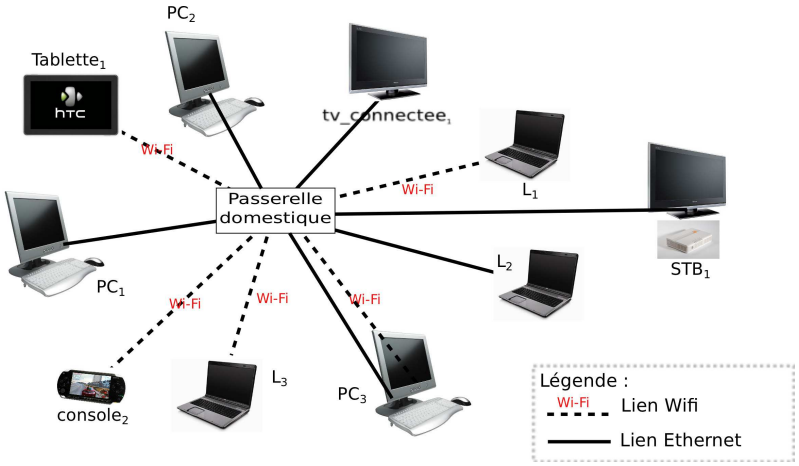


FIGURE 4.4 – Exemple de réseau local domestique

availableQoR (`QdRdisponible`).

```
composite LRM.LMRM {                                    1
  provides LMRM as mem_manager;
                                                         3
  attribute int capacity;
  attribute int availableQoR;                            5

  ...                                                    7
}
```

Listing 4.1 – extrait d'ADL du composant LMRM

Au niveau global, pour simplifier le composant GNRM, ce dernier gère tous les liens sans utiliser de sous-composants. Le GNRM connaît donc la `capacité` et la `QdRdisponible` de chaque lien.

Liens logiques Dans ARMOR les liens de type Ethernet sont représentés par des liens logiques entre deux équipements. Un lien logique a une `source` et une `destination`, comme le montre la Figure 4.3. Les éléments `Interface Ethernet` et Lien Ethernet, de cette figure, proviennent du modèle du réseau local domestique, détaillé sur la Figure 2.8, page 23. Par exemple, sur la Figure 4.4 le lien logique $PC_1 \rightarrow STB_1$ a pour source l'interface Ethernet de l'équipement PC_1 et pour destination l'interface Ethernet de l'équipement STB_1. Il n'y a qu'un seul lien de type Wifi, dans un réseau local domestique, qui est partagé par tous les équipements.

QdRpartagée Pour la gestion des ressources réseau, ARMOR utilise une caractéristique de QdS appelée `QdRpartagée`. Cette caractéristique de QdS permet de faire une réservation pour les pics de consommation (détails à la section 4.3.1.1). Chaque lien et chaque interface réseau est associé à une caractéristique de QdS `QdRpartagée`. Comme les autres caractéristiques de QdS, la `QdRpartagée` est intégrée dans les composants d'ARMOR, à l'aide d'un attribut.

4.2 Initialisation

Pour garantir les QdR requises par une application de diffusion de contenus multimédia, ARMOR vérifie que la QdR disponible sur chaque ressource utilisée est suffisante. La QdR disponible dépend de la capacité de la ressource (la QdR maximale que la ressource peut fournir) et des réservations effectuées.

La phase d'initialisation configure la capacité des ressources, initialise les mécanismes de réservation du système d'exploitation de chaque équipement et configure les composants d'ARMOR.

Cette phase d'initialisation est dirigée par le composant `Global Resources Manager` (GRM), connaissant tous les équipements et tous les liens présents dans le réseau local domestique. Cette initialisation est réalisée au niveau local par le composant `Local Resources Manager` (LRM), démarré sur chaque équipement.

Nous détaillons maintenant la phase d'initialisation des ressources réseau, des ressources CPU et des ressources mémoire RAM.

4.2.1 Initialisation des ressources réseau

La phase d'initialisation des ressources réseau (interfaces réseau et liens) estime les capacités des interfaces réseau et des liens. En plus de leur capacité, des contraintes sont exprimées sur les caractéristiques de QdS réseau des liens (délai, gigue et taux de perte). Ces caractéristiques sont donc aussi initialisées. Ensuite, la phase d'initialisation met en place les mécanismes de réservation utilisés par ARMOR. Enfin, la phase d'initialisation configure les composants de gestion des ressources réseau d'ARMOR.

4.2.1.1 Estimation de la capacité et des caractéristiques de QdS réseau

Les capacités réelles des interfaces réseau Wifi sont inférieures aux capacités théoriques du standard. Par exemple, une interfaces Wifi de norme G a théoriquement une capacité de 54 Mbits/s. Cependant la capacité réelle varie le plus souvent entre 15 et 25 Mbits/s. Pour avoir une meilleure estimation de capacité des interfaces réseau, ARMOR effectue une mesure, dite active, lors de l'initialisation. Une mesure active consiste à envoyer des données sur le réseau pour mesurer ses caractéristiques.

Dans le cas d'ARMOR, on cherche à connaître la capacité des interfaces réseau. La mesure active est réalisée en envoyant 300 Mbits pendant une seconde. Cette valeur correspond à la plus grande capacité théorique des interfaces Wifi actuelles du réseau local domestique (300 Mbits/s). Pour effectuer la mesure, le logiciel `iperf` [2] est classiquement utilisé dans Linux. Ce logiciel fonctionne avec une partie cliente

2. `http://iperf.sourceforge.net/`

qui envoie des données vers une partie serveur. ARMOR démarre la partie serveur du logiciel `iperf` sur l'équipement exécutant le GRM. Lors de l'initialisation de chaque interface réseau, ARMOR démarre la partie cliente du logiciel `iperf` sur l'équipement en cours d'initialisation. La capacité et les caractéristiques de QdS réseau (délai, gigue, taux de perte) varient en fonction des classes de trafic. Une classe de trafic définit la priorité d'un flux, selon une norme de QdS réseau. Si l'interface réseau de l'équipement supporte une norme de QdS réseau alors la mesure est effectuée avec la même classe de trafic que les futures applications multimédia. Si l'interface réseau de l'équipement ne supporte pas de norme de QdS réseau, la mesure est faite comme s'il s'agissait de trafic par défaut.

Quand la mesure est terminée, la partie serveur du logiciel `iperf` donne la bande passante mesurée et les caractéristiques de QdS réseau observées. La capacité de l'interface réseau est initialisée avec la valeur de bande passante mesurée. Les caractéristiques de QdS réseau du lien Wifi sont mesurées plusieurs fois lors de l'initialisation de toutes les interfaces. ARMOR effectue une moyenne pour chaque caractéristique de QdS réseau et initialise ensuite les caractéristiques de QdS réseau du lien Wifi.

La capacité d'un lien Wifi est difficile à mesurer car les capacités des interfaces varient. De ce fait, ARMOR initialise la capacité du lien à la valeur théorique.

Pour les interfaces Ethernet, la capacité réelle des interfaces est proche de la capacité théorique. La capacité théorique est donc utilisée pour initialiser la capacité des interfaces et des liens. Dans un réseau local domestique, les distances sont faibles. Les caractéristiques de QdS réseau (délai, gigue et taux de perte) des liens Ethernet sont proches de 0. Elles sont donc initialisées à 0.

4.2.1.2 Initialisation des mécanismes de réservation

Une application de diffusion de contenus multimédia utilise trois ressources réseau : l'interface réseau de l'équipement serveur, l'interface réseau de l'équipement client et le lien entre les deux. L'équipement client ne maîtrise par l'arrivée des paquets sur son interface réseau. De ce fait, il n'est pas possible de réserver la QdR réseau sur l'interface cliente. La gestion des QdR réseau s'effectue donc sur deux ressources : l'interface réseau de l'équipement serveur et le lien entre les équipements. Le lien est partagé par les équipements, il est géré au niveau global. L'interface réseau de l'équipement serveur est gérée au niveau local.

Au niveau global, deux configurations existent dans le réseau local domestique :

- une norme de QdS réseau est supportée : la passerelle domestique différencie le trafic multimédia du trafic par défaut (émis par n'importe quel équipement du réseau local domestique) et le traite en priorité ;
- aucune norme de QdS réseau n'est supportée : la passerelle domestique ne différencie pas le trafic multimédia du trafic par défaut. Le trafic par défaut doit être limité au niveau local.

Dans les deux cas, la passerelle domestique ne différencie pas le trafic de deux applications de diffusion de contenus multimédia, ayant la même priorité. Si une application utilise plus de QdR réseau que sa réservation, les autres réservations peuvent être compromises. Le trafic émis par une application doit être limité au niveau local.

Au niveau local, ARMOR met en place un politique de gestion du trafic émis pour assurer la réservation de QdR requise par l'application. De plus, cette politique marque les paquets multimédia pour qu'ils utilisent une norme de QdS réseau et

soient traités en priorité au niveau global (si la norme de QdS réseau est supportée). La politique de gestion du trafic émis est aussi utilisée pour pallier le manque de contrôle sur les ressources globales :

- limitation du trafic par défaut quand aucune norme de QdS réseau n'est supportée ;
- limitation du trafic émis par une application à hauteur de sa réservation.

Lors de la phase d'initialisation, aucune réservation n'est effectuée. La politique de gestion du trafic émis, installée par ARMOR, permet de différencier le trafic multimédia du trafic par défaut et de limiter le trafic par défaut si besoin. Nous détaillons la politique mise en place par ARMOR, sur chaque interface réseau. Nous détaillons ensuite quand et comment ARMOR limite le trafic par défaut pour pallier l'absence de support de QdS réseau.

Politique de gestion du trafic émis Les équipements cibles de nos travaux utilisent Linux. La politique par défaut, du noyau Linux, est d'offrir un service équitable entre toutes les applications émettant sur un équipement. ARMOR modifie la politique de gestion du trafic émis, sur l'équipement serveur, pour différencier le trafic multimédia du trafic par défaut. Le trafic multimédia est traité en priorité.

Dans Linux, une politique de gestion du trafic émis est construite à l'aide d'un arbre dans lequel sont orientés les paquets à émettre, depuis la *qdisc* (*queuing discipline*) racine jusqu'aux feuilles. L'arbre contient trois types d'éléments :

- *leaf qdisc* : feuille de l'arbre, c'est une file d'attente dans laquelle les paquets à émettre sont placés ;
- *class* : une *class* est une *qdisc* contenant d'autres *qdisc*. Une *class* définit comment le trafic est affecté à ses *qdisc* filles. Un *filter* (filtre), appartenant à une *class*, est utilisé pour affecter le trafic à une *class* fille selon le critère choisi ;
- *classfull qdisc* : élément structurant, contenant des *class*.

La Figure 4.5 montre la politique mise en place, par ARMOR, lors de la phase d'initialisation, sur chaque interface réseau. Lorsqu'une application émet sur le réseau, les paquets à envoyer (*Packets to send*) sont aiguillés dans la politique de gestion du trafic émis (de gauche à droite sur la figure). Cet aiguillage s'arrête quand une *leaf qdisc* feuille (la plus à droite) est rencontrée. Le paquet à envoyer est alors stocké dans cette *qdisc*. Le noyau Linux gère ensuite l'envoi des paquets sur le lien réseau, selon le type et les paramètres de la *qdisc*.

Linux impose de commencer la politique (à gauche de la figure) par une *qdisc*. ARMOR crée une *qdisc* avec l'identifiant `1:`. ARMOR met en place la politique pour différencier le trafic multimédia du trafic par défaut. Pour ce faire il utilise une *class* de type *Hierachical Token Bucket* (HTB), avec l'identifiant `1:1`. Linux impose d'utiliser le même majeur (`1:`) pour une *class* que sa *qdisc* mère. La *qdisc* `1:` est aussi une HTB, avec les mêmes paramètres que la *class* `1:1`. Quand un paquet arrive à la *class* `1:1`, il est aiguillé selon le résultat du filtre 1. Si c'est un paquet d'une application de diffusion de contenus multimédia alors il est envoyé à la *class* `1:3`, sinon il est envoyé à la *class* `1:2`. ARMOR affecte une priorité supérieure à la *class* `1:3`. Le trafic multimédia est donc envoyé en priorité, sur le lien réseau, par rapport au trafic par défaut. La gestion du trafic multimédia (réservation et limitation), à partir de la *qdisc* `3:`, est détaillée par la suite (dans la section 4.3). La *qdisc* feuille

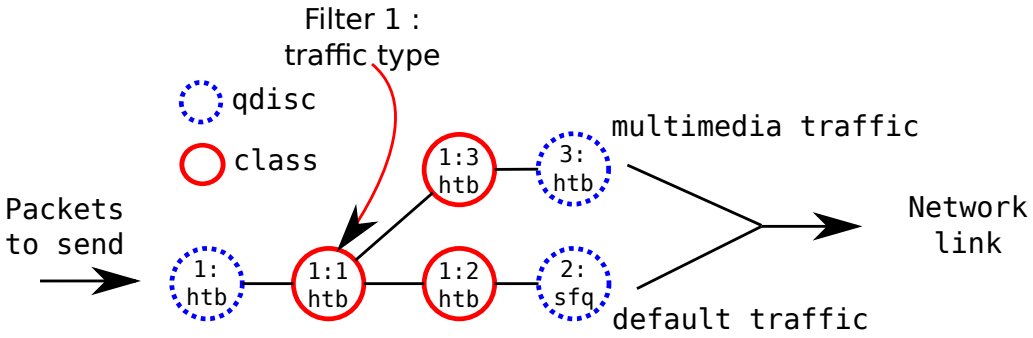


FIGURE 4.5 – Politique de gestion du trafic émis

du trafic par défaut (2:) est de type *Stochastic Fairness Queueing* (SFQ) pour être équitable entre tous les flux par défaut.

Pour différencier le trafic multimédia du trafic par défaut, le filtre 1 (*Filter 1 : traffic type*) utilise la valeur du champ DSCP, de l'en-tête IP, du paquet à émettre (couche 3 du modèle OSI). ARMOR modifie la valeur du champ DSCP des paquets des application de diffusion de contenus multimédia, avant qu'ils ne soient envoyés à la *qdisc* 1:. Cette technique est appelée marquage, où les paquets sont marqués pour être différenciés. Pour reconnaître les paquets d'une application, ARMOR se base sur l'adresse IP destination (*IP_DEST*) et le port destination (*PORT_DEST*) utilisés par l'application. Les paquets sont marqués selon la norme de QdS réseau supportée par l'interface réseau. Si aucune norme de QdS réseau n'est supportée par l'équipement, les paquets sont tout de même marqués pour être différenciés. Pour la norme 802.11e (Wifi), la valeur de la classe de trafic *AC_VI* (vidéo) (cf. section 2.1.1.1) est utilisée. Pour la norme 802.1p (Ethernet), la valeur de la classe de trafic *4* (vidéo) est utilisée. Le noyau Linux se charge ensuite d'utiliser la bonne valeur pour l'en-tête de la couche 2 du modèle OSI. Les paquets du trafic par défaut ont la valeur par défaut du champ DSCP (0). Ainsi le trafic multimédia est marqué comme prioritaire, selon la norme de QdS réseau utilisée, et est traité en priorité au niveau global.

Limitation du trafic émis Pour différencier le trafic multimédia du trafic par défaut, au niveau global, la même norme de QdS réseau doit être supportée par l'interface réseau de l'équipement serveur et par la passerelle domestique. Si ce n'est pas le cas, ARMOR limite le trafic émis sur certains équipements, afin d'empêcher le trafic par défaut de perturber une application multimédia. Trois cas sont possibles :

1. la passerelle domestique ne supporte pas de norme de QdS réseau : limitation du trafic émis par tous les équipements du réseau local domestique ;
2. la passerelle domestique supporte une norme de QdS réseau et un équipement ne supporte pas de norme QdS réseau : limitation du trafic émis par tous les équipements appartenant au même segment L2 [3] ;
3. La passerelle domestique et toutes les interfaces d'un segment L2 supportent la même norme de QdS réseau : pas de limitation du trafic émis sur ce segment L2.

3. un segment L2 contient l'ensemble des liens de même type (Ethernet ou Wifi)

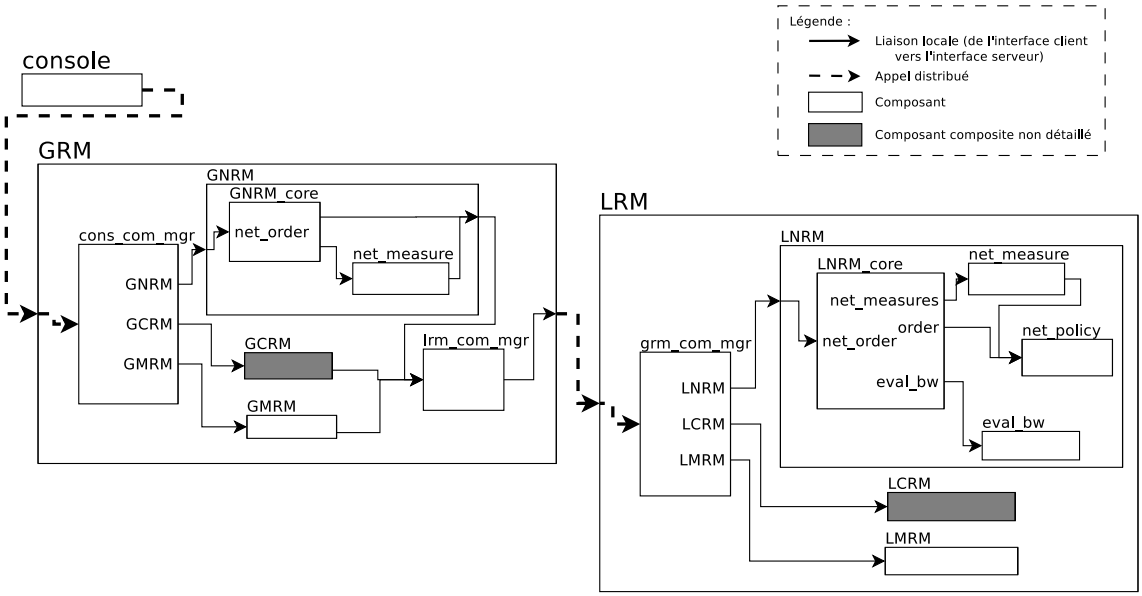


FIGURE 4.6 – Composants de gestion des QdR réseau

Pour limiter le trafic par défaut d'un équipement (tous les équipements dans le cas 1 et les équipements du segment L2 limité dans le cas 2), ARMOR le façonne en se basant sur le modèle à seau de jeton. Ce modèle utilise deux paramètres (r,b) définissant un envoi sur le réseau avec un débit de r kbits/s, et un envoi en rafale de b kbits au maximum. Les classes HTB utilisent deux couples de paramètres ($rate$,b_{rate}) et ($ceil$,b_{ceil}) correspondant chacun aux paramètres (r, b) du modèle à seau de jeton. Les paramètres ($rate$,b_{rate}) sont utilisés pour réserver la QdR réseau, les paramètres ($ceil$,b_{ceil}) sont utilisés pour limiter la QdR réseau.

Dans la politique de gestion du trafic émis, installée par ARMOR (Figure 4.5), le trafic par défaut est limité par la *class* `1:2` (gérant le trafic par défaut), de type HTB. Cette *class* façonne le trafic pour qu'il ne dépasse pas une valeur, fixée par défaut à $500kbits/s$ $(0,5\ megabits/s)$. Cette valeur semble raisonnable au vu des normes réseau actuelles, où les interfaces réseau ont des capacités de plusieurs dizaines de megabits par seconde. Nous verrons dans le chapitre 6 portant sur les évaluations, que cette valeur permet de garantir les QdR réseau aux applications multimédia. Pour limiter le trafic par défaut, en le façonnant, le paramètre *ceil* de la classe `1:2` est fixé à $500kbits/s$, le paramètre b_{ceil} est laissé à sa valeur par défaut.

Quand ARMOR ne limite pas le trafic par défaut des équipements (cas 3 et équipements des segments L2 non limités dans le cas 2), le paramètre *ceil* de la *class* `1:2` est fixé à la capacité de l'interface. Le paramètre b_{ceil} de cette *class* est laissé à sa valeur par défaut. Ainsi le trafic par défaut utilise ce qui est laissé libre par les applications de diffusion de contenus multimédia.

4.2.1.3 Configuration des composants d'ARMOR

ARMOR gère les QdR réseau, à l'aide des composants `Global Network Resources Manager` (GNRM) et `Local Network Resources Manager` (LNRM) [72]. La Figure 4.6 détaille ces composants.

La GNRM coordonne les mesures précédemment présentées pour initialiser les

caractéristiques de QdS de tous les liens et de toutes les interfaces réseau du réseau local domestique. Les mesures actives sont effectuées, par les composants `net_mesure`. La partie serveur, du logiciel `iperf`, est démarrée par le composant `net_mesure` du composant GNRM. La partie cliente, du logiciel de `iperf`, est démarrée par le composant `net_mesure` du composant LNRM.

Au niveau global, le composant GNRM gère tous les liens, via le composant `GNRM_core`. Ce composant initialise la `capacité`, la `QdRdisponible` et la `QdRpartagée` des liens. Lors de l'initialisation aucune réservation n'est faite. La `QdRdisponible` des liens est fixée à la capacité de la ressource et la `QdRpartagée` des liens est fixée à 0.

Au niveau local, le composant `LNRM_core` traite les demandes venant du niveau global. Le composant LNRM a une instance de composant `net_policy` par interface réseau. Chaque composant `net_policy` a un attribut `capacité`, `QdRdisponible` et `QdRpartagée` qui sont initialisés comme ceux des liens. La politique de gestion du trafic émis est installée sur chaque interface par le composant `net_policy`. Le composant `eval_bw` permet d'évaluer la bande passante requise pour une application (détaillé à la section 4.3.1).

4.2.2 Initialisation des ressources CPU

Les ressources CPU sont gérées uniquement au niveau local, par le système d'exploitation des équipements. Nous présentons comment la capacité des ressources CPU est initialisée. Ensuite, nous détaillons l'initialisation des mécanismes de réservation du système d'exploitation. Enfin, nous détaillons l'initialisation des composants de gestion des ressources CPU d'ARMOR.

4.2.2.1 Initialisation de la capacité

Pour ARMOR, chaque équipement a une seule ressource CPU. Dans le cas des multiprocesseurs SMP [4], la ressource CPU regroupe l'ensemble des processeurs. L'affectation des *threads* à un cœur n'est pas considérée par ARMOR ; elle est laissée à la charge du système d'exploitation. La capacité de la ressource CPU est fixée par l'équation (4.1). Pour les multiprocesseurs SMP, la capacité représente la puissance de calcul totale de l'équipement.

$$capacité = nb\ cpu * 100\% \tag{4.1}$$

Les QdR CPU requises s'expriment en % (cf. chapitre 3). Dans le cas des multiprocesseurs SMP, 100% correspond à l'utilisation totale d'un cœur. Dans le cas d'un multiprocesseurs SMP avec deux cœurs, 200% correspond à l'utilisation totale de la ressource CPU de l'équipement. La formule (4.1), d'initialisation de la capacité de la ressource CPU, permet d'utiliser directement les QdR CPU requises, lors d'une réservation.

4.2.2.2 Initialisation des mécanismes de réservation

Le système d'exploitation gère la ressource CPU, d'un équipement, à l'aide d'un ordonnanceur. Il existe deux types d'ordonnanceurs, permettant de faire de la réser-

4. *Symetric Multi Processor*

vation : les ordonnanceurs à parts équitables et les ordonnanceurs basés sur un serveur. ARMOR supporte un ordonnanceur de chaque type : l'ordonnanceur *Completely Fair Scheduler* (CFS), à parts équitables et l'ordonnanceur *Constant Bandwidth Server* (CBS), basé sur un serveur. Ces ordonnanceurs gèrent la ressource CPU différemment. Nous détaillons comment ARMOR initialise les mécanismes de réservation pour chaque ordonnanceur.

Ordonnanceur CFS Avec cet ordonnanceur, Linux offre la possibilité de gérer les processus par groupe, à l'aide des *control groups* (appelés *cgroups*). Un *cgroup* contient des processus ou des *cgroups*. Dans ce cas, l'ordonnanceur affecte équitablement le CPU à chaque *cgroup*. Cette "équité" peut être modifiée en affectant des poids, relatifs les uns par rapport aux autres, aux différents *cgroups*. Par exemple, si deux *cgroups* A et B ont respectivement un poids de 1 et 2 alors le *cgroup* B aura deux fois plus de CPU que le *cgroup* A.

Pour garantir la QdR CPU d'une application multimédia, ARMOR effectue une réservation en créant un *cgroup* dédié à l'application. Tous les autres processus sont répartis en deux *cgroups* : *user_group* et *system_group*. Lors de l'initialisation, ARMOR crée ces deux *cgroups*. Le *cgroup user_group* contiendra tous les processus lancés par l'utilisateur (par exemple un navigateur Internet). Le *cgroup system_group* contiendra tous les processus systèmes (par exemple des processus de mise à jour). À l'intérieur de ces deux *cgroups*, les processus ont tous le même poids et sont traités équitablement.

Comme les poids des *cgroups* sont relatifs, les uns par rapport aux autres, la somme des poids ne doit pas dépasser la capacité de la ressource CPU. Si c'est le cas, les réservations ne sont plus garanties. Par exemple si deux *cgroups* ont un poids de 75, et que la capacité de la ressource CPU est de 100, alors chaque *cgroups* aura 50% du CPU. Pour permettre de démarrer le maximum d'applications multimédia, les poids des *cgroups user_group* et *system_group* sont fixés à $\frac{1}{100} * capacité$. Cette valeur, volontairement faible, permet de laisser un peu de ressource CPU aux processus utilisateurs et systèmes. Ainsi l'utilisateur pourra garder la main sur l'équipement, par exemple pour arrêter des applications. L'algorithme CFS fournit uniquement des réservations molles. Quand une application a une réservation molle, elle peut utiliser plus que sa réservation. À condition bien entendu que l'application ne perturbe pas d'autres réservations. Les *cgroups user_group* et *system_group* utilisent donc la QdR CPU laissée libre par les applications de diffusion de contenus multimédia. Effectuer une petite réservation pour ces *cgroups* ne limite pas l'utilisation de l'équipement.

Une fois les *cgroup* créés, il faut affecter les processus en cours d'exécution au bon *cgroup*. Les processus fils du gestionnaire de fenêtre sont affectés au *cgroup user_group*, tous les autres processus sont affectés au *system_group*. Lors de sa création, un processus est affecté au *cgroup* de son processus père. Tous les processus lancés par l'utilisateur, via le gestionnaire de fenêtre, seront donc dans le *cgroup user_group*.

Ordonnanceur CBS L'ordonnanceur CBS utilise des serveurs CBS avec les paramètres (C, T) (budget,période) pour exécuter les applications. Les serveurs sont

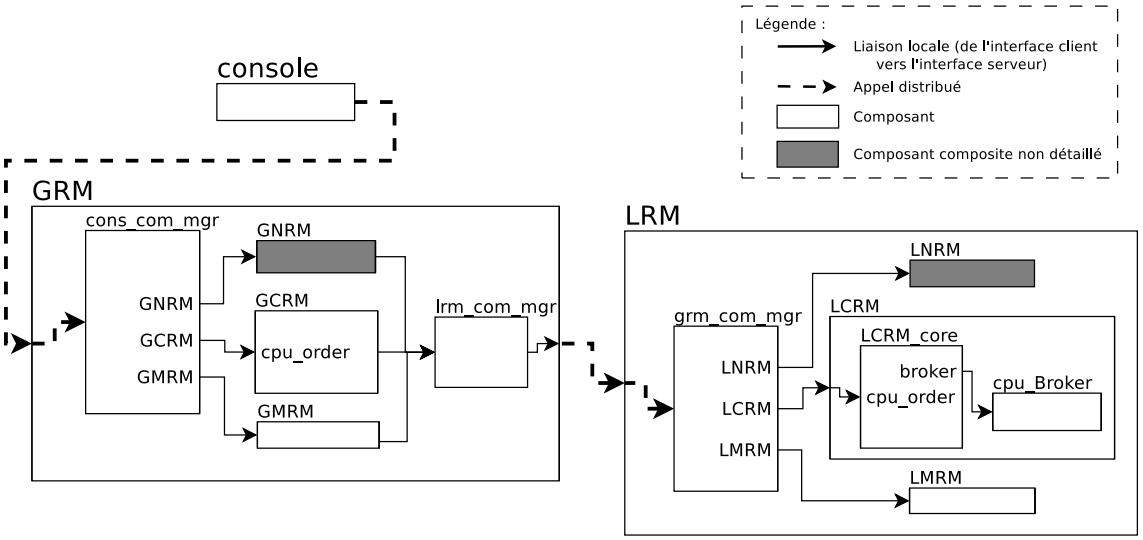


FIGURE 4.7 – Composants de gestion des QdR CPU

ordonnancés à l'aide de l'algorithme *Earliest Deadline First* (EDF) qui affecte le CPU au serveur ayant l'échéance la plus proche.

Pour garantir la QdR CPU d'une application multimédia, ARMOR effectue une réservation en créant un serveur CBS dédié à l'application.

Les processus qui ne sont pas exécutés par un serveur CBS sont ordonnancés quand aucun serveur CBS n'est exécutable. C'est pourquoi, aucune réservation n'est effectuée pour les processus utilisateurs ou systèmes.

4.2.2.3 Configuration des composants d'ARMOR

ARMOR gère les QdR CPU à l'aide des composants `Global CPU Resources Manager (GCRM)` et `Local CPU Resource Manager (LCRM) [74]`, détaillés sur la Figure 4.7. Le composant GCRM coordonne la gestion des QdR CPU, effectuée localement par le LCRM, démarré sur l'équipement.

Lors de la phase d'initialisation, ARMOR configure le composant LCRM, en fonction de l'ordonnanceur de l'équipement. Ce composant est formé de deux sous-composants : le `LCRM_core` et le `cpu_Broker`. Lors de la phase d'initialisation, les attributs `ordonnanceur` de ces deux composants sont initialisés en fonction de l'ordonnanceur de l'équipement. Par la suite les services offerts par les composants locaux de gestion des ressources CPU sont indépendants de l'ordonnanceur de l'équipement.

Le composant `LCRM_core` contient la logique de gestion du CPU. Le composant `cpu_Broker` accède aux mécanismes de réservation fournis par le système d'exploitation de l'équipement et connaît la QdR CPU disponible à tout instant. Le composant `cpu_Broker` a un attribut `capacité` (capacité de la ressource CPU) et un attribut `QdRdisponible` (QdR actuellement disponible sur la ressource). Lors de l'initialisation l'attribut `capacité` est initialisé à la capacité de la ressource CPU. L'attribut `QdRdisponible` est initialisé différemment selon le type d'ordonnanceur.

Ordonnanceur CFS Lors de l'initialisation, aucune réservation n'est effectuée. La QdR disponible est donc égale à la `capacité` moins la QdR réservée pour les processus du *user_group* et du *system_group*. L'attribut `QdRdisponible` du composant

`cpu_Broker` est donc initialisé par l'équation (4.2).

$$QdRdisponible = capacité - user\_group - system\_group \quad (4.2)$$

Ordonnanceur CBS Pour l'ordonnanceur CBS, aucune réservation n'est effectuée pour les processus systèmes et utilisateurs. L'attribut `QdRdisponible` du composant `cpu_Broker` est donc initialisé à la capacité de la ressource CPU.

4.2.3 Initialisation des ressources mémoire RAM

Les ressources mémoire RAM sont gérées uniquement au niveau local, par le système d'exploitation des équipements. La capacité des ressources mémoire RAM est initialisée à la taille de la mémoire RAM de l'équipement. Nous détaillons l'initialisation des mécanismes de réservation du système d'exploitation pour les ressources mémoire RAM. Ensuite, nous détaillons l'initialisation des composants de gestion des ressources mémoire RAM d'ARMOR.

Initialisation des mécanismes de réservation Le système d'exploitation gère la mémoire RAM, d'un équipement, à l'aide de l'algorithme de pagination. La mémoire RAM est plus rapide mais aussi plus onéreuse que le disque. Le système d'exploitation utilise le disque quand la mémoire RAM ne suffit pas. L'algorithme de pagination gèrent la mise en mémoire tampon (sur le disque) des processus. Comme pour le CPU, Linux permet de gérer la mise en mémoire tampon au niveau des group de processus (*cgroup*). Si un processus d'un *cgroup* utilise trop de mémoire RAM alors c'est la mémoire RAM de ce *cgroup* qui sera mise en mémoire tampon en priorité.

Le noyau Linux offre aussi la possibilité de limiter l'utilisation mémoire RAM d'un *cgroup*. Cette limite peut être dure (non dépassable) ou molle (dépassable si cela ne perturbe pas les autres *cgroups*). Dans le cas d'une réservation molle, si un *cgroup* utilise plus de mémoire RAM que sa limite, alors il sera mis en mémoire tampon si besoin. En revanche le noyaux Linux ne permet pas de réserver de la mémoire RAM à un *cgroup*. Pour pallier ce manque ARMOR crée, comme pour les ressources CPU, deux *cgroups* : *user_group* (pour les processus utilisateurs) et *system_group* (pour les processus systèmes). Pour garantir la QdR mémoire RAM requise par une application multimédia, ARMOR limite l'utilisation mémoire RAM des *cgroups user_group* et *system_group*.

Quand une limite dure est imposée sur un *cgroup* et qu'un processus de ce *cgroup* veut utiliser plus de mémoire RAM, le noyau Linux refuse. Ce refus peut se traduire par une erreur du fait du manque de mémoire. Les processus systèmes sont des processus important (sécurité, mise à jour). ARMOR affecte donc une limite molle au *cgroup system_group*. Un processus système peut utiliser plus de mémoire RAM si nécessaire. Pour laisser de la place aux applications multimédia et aux processus utilisateurs, la limite imposée sur le *system_group* est fixée à $\frac{1}{100} * capacité$.

Lors de l'initialisation, aucune réservation n'est faite. ARMOR affecte tout le reste de la mémoire au *cgroup user_group*. Lorsqu'une réservation est faite pour une application multimédia, ARMOR diminue la limite du *cgroup user_group*. Si une limite molle est imposée sur le *cgroup user_group*, alors un processus de ce *cgroup*

pourra utiliser la mémoire RAM d'une application multimédia. Si l'application réclame la mémoire RAM lui appartenant, l'algorithme de pagination va mettre en mémoire tampon la mémoire du processus du *user_group*. Cette mise en mémoire tampon prend du temps, ce qui peut poser des problèmes de QdS pour une application multimédia. Pour empêcher ce phénomène, ARMOR impose une limite dure sur le *cgroup user_group*.

Ainsi seuls les processus du *system_group* peuvent prendre de la mémoire RAM non utilisée par une application multimédia. Dans ce cas l'application reste prioritaire sur l'accès à la mémoire RAM. Les processus systèmes ont une fréquence d'exécution faible (par exemple des processus de mise à jour). ARMOR fait l'hypothèse qu'il ne perturberont pas les applications multimédia.

Configuration des composants d'ARMOR ARMOR gère les QdR mémoire RAM à l'aide des composants `Global Memory Resources Manager (GMRM)` et `Local Memory Resource Manager (LMRM)`. Le composant GMRM coordonne la gestion des QdR mémoire RAM, effectuée localement par le LMRM démarré sur l'équipement.

L'attribut `capacité` du composant LMRM est initialisé à la capacité de la mémoire RAM. L'attribut `QdRdisponible` du composant LMRM est initialisé en retranchant la QdR réservée pour le *system_group* (4.3).

$$QdRdisponible = capacité - system\_group \tag{4.3}$$

4.3 Démarrage d'une application de diffusion de contenus multimédia

Lorsqu'un utilisateur veut démarrer une application de diffusion de contenus multimédia, une demande est faite à ARMOR. ARMOR vérifie que les QdR requises par l'application, sont disponibles, sur chacune des ressources utilisées. Cette phase est appelée contrôle d'admission. Si un test d'admission échoue, l'application n'est pas démarrée. Si tous les tests d'admission réussissent, ARMOR effectue les réservations sur toutes les ressources utilisées à l'aide des mécanismes de réservation du système d'exploitation. Dans la suite de ce chapitre nous détaillons ces deux phases.

4.3.1 Contrôle d'admission

Le contrôle d'admission, pour une ressource, vérifie que la QdR disponible est supérieure ou égale à la QdR requise.

La QdR disponible sur chaque ressource est connue, à tout instant, par les composants d'ARMOR. Pour les ressources réseau, les QdR requises sont obtenues par une analyse du flux (détaillée à la section 3.1), effectuée lors de la phase de contrôle d'admission. Pour les ressources CPU et mémoire RAM, les QdR requises sont connues au travers d'un *manifest*, contenant les QdR CPU et mémoire RAM requises sur toutes les ressources utilisées.

Nous détaillons maintenant la phase de contrôle d'admission pour chaque ressource, en commençant par les ressources réseau.

4.3.1.1 Contrôle d'admission pour les ressources réseau

Dans le réseau local domestique, les équipements communiquent via des liens connectés à la passerelle domestique. La passerelle domestique a une capacité supérieure à celle des liens. ARMOR fait l'hypothèse que la capacité de la passerelle domestique n'est jamais dépassée. Une application de diffusion de contenus multimédia utilise trois ressources réseau : l'interface réseau de l'équipement serveur, l'interface réseau de l'équipement client et le lien entre les deux. ARMOR fait l'hypothèse que la capacité de réception de l'équipement client est suffisante. Nous verrons dans le chapitre 6 que ces hypothèses sont réalistes pour garantir les QdR réseau requises. Le contrôle d'admission, pour les ressources réseau, se focalise donc sur deux ressources : l'interface de l'équipement serveur et le lien. La QdR requise est la même sur toutes les ressources réseau utilisées. ARMOR commence par estimer la QdR réseau requise, en analysant le flux à diffuser. En plus de la QdR requise, des contraintes sur les caractéristiques de QdS réseau (délai, gigue et taux de perte) sont exprimées pour les liens. Ces contraintes sont renseignées dans le *manifest*.

ARMOR vérifie ensuite que la QdR disponible sur les deux ressources réseau utilisées, connue à l'aide de ses composants, est suffisante. Ce contrôle d'admission est dit "théorique". Ce test tient compte des réservations effectuées et de la capacité de la ressource réseau. Cette capacité a été calculée lors de la phase d'initialisation. Les caractéristiques de QdS réseau du lien sont mesurées à nouveau pour vérifier les contraintes de QdS réseau.

La capacité d'une interface Wifi varie dans le temps. Par conséquent, la QdR disponible sur une interface Wifi varie aussi. Si le contrôle d'admission "théorique" réussit, pour l'interface réseau et pour le lien, ARMOR effectue un test réel de bande passante. Ce test vérifie que la QdR supposée disponible par ARMOR sur l'interface réseau l'est réellement. Le test de bande passante envoie des données entre l'équipement serveur et l'équipement client. Ce test vérifie aussi que les caractéristiques de QdS réseau observées satisfont les contraintes exprimées dans le *manifest*.

Estimation de la QdR réseau requise La QdR requise est calculée par ARMOR, en analysant le flux vidéo. Cette analyse est faite localement, sur l'équipement serveur, comme expliqué à la section 3.1. L'analyse du flux dépend du standard d'encodage. Plusieurs outils sont à la disposition d'ARMOR, pour les différents standards existants. Pour masquer cette hétérogénéité, le composant `eval_bw` (composant du `Local Network Resources Manager`) se charge d'utiliser les bons outils et de fournir la QdR réseau requise. La QdR requise est ensuite envoyée au niveau global, pour effectuer le contrôle d'admission théorique.

Contrôle d'admission théorique Le contrôle d'admission théorique est réalisé au niveau global, pour le lien, et au niveau local, pour l'interface réseau de l'équipement serveur. Le test est identique pour les deux ressources.

Sur les ressources réseau, la QdR requise s'exprime à l'aide de deux valeurs $maxQoR_i$ et $statMaxQoR_i$. $maxQoR_i$ correspond à la QdR maximale requise par l'application durant toute son exécution et $statMaxQoR_i$ correspond à la QdR maximale sans les pics d'utilisation. Pour diminuer la sur-réservation (la différence entre la QdR réservée et la QdR utilisée), ARMOR effectue une réservation dédiée à l'application avec $statMaxQoR_i$. Ensuite ARMOR effectue une réservation pour les pics

d'utilisation de toutes les applications utilisant une même ressource réseau. Pour une application, la QdR requise pour les pics est appelée $reqSharedQoR_i$ et est calculée à l'aide de l'équation (4.4).

$$reqSharedQoR_i = maxQoR_i - statMaxQoR_i \tag{4.4}$$

ARMOR fait l'hypothèse que deux pics, de deux applications, n'auront pas lieu en même temps. Si deux pics ont lieu au même instant alors la QdR réseau n'est garantie pour aucune des applications de diffusion de contenus multimédia ayant un pic d'utilisation. Cette hypothèse "optimiste" permet de diminuer la sur-réservation des ressources réseau. En effet, la réservation dédiée à une application est plus faible puisqu'elle est limitée à $statMaxQoR$.

Le contrôle d'admission pour une ressource réseau consiste donc à vérifier que la QdR disponible (la capacité moins les réservations effectuées) sur la ressource est suffisante pour :

- étendre la réservation partagée (`QdRpartagée`) pour prendre en compte les pics de l'application ($reqSharedQoR_i$). La valeur de `QdRpartagée` étendue, pour prendre en compte les pics de l'application est donnée par l'équation (4.5).
- créer une réservation dédiée à l'application (de taille $statMaxQoR_i$)

$$QdRpartagée = max(reqSharedQoR_i, QdRpartagé) \tag{4.5}$$

Ces deux points sont vérifiés par ARMOR par le test d'admission (4.6). Ce test consiste à vérifier que la capacité de la ressource réseau est supérieure à la somme des n réservations existantes ($statMaxQoR_j$), la QdR requise de l'application sans les pics ($statMaxQoR_i$) et de la réservation pour la bande passante partagée (`QdRpartagée`). Si la nouvelle application est admise, la bande passante partagée est étendue à la taille du plus grand pic d'utilisation (4.5). Pour le test d'admission (4.6), ARMOR utilise la valeur donnée par (4.5), qui prend en compte la nouvelle application.

$$capacité \geq \sum_{j=1}^{n} statMaxQoR_j + statMaxQoR_i + QdRpartagée \tag{4.6}$$

La Figure 4.8 schématise le contrôle d'admission pour une ressource réseau. Dans l'état courant, une application de diffusion de contenus multimédia est en cours d'exécution. Une nouvelle demande de démarrage est effectuée : *"nouvelle requête"*. Cette demande est acceptée, car la `QdRpartagée` peut être étendue et une réservation peut être créée pour l'application sans les pics.

Test réel de bande passante Le contrôle d'admission théorique se base sur la capacité de l'interface. La capacité des interfaces Ethernet ne varient pas, contrairement aux interfaces Wifi. La capacité de chaque interface Wifi est estimée lors de la phase d'initialisation. Pour savoir si l'interface réseau peut effectivement fournir la QdR requise par l'application, ARMOR effectue un test de bande passante.

Le test de bande passante consiste à envoyer pendant une seconde $maxQoR_i$ kbits de l'équipement serveur vers l'équipement client. Comme pour l'estimation de la capacité d'une interface réseau (cf. section 4.2.1), le logiciel `iperf` est utilisé à l'aide des composants `net_mesure`. Si une norme de QdS réseau est supportée,

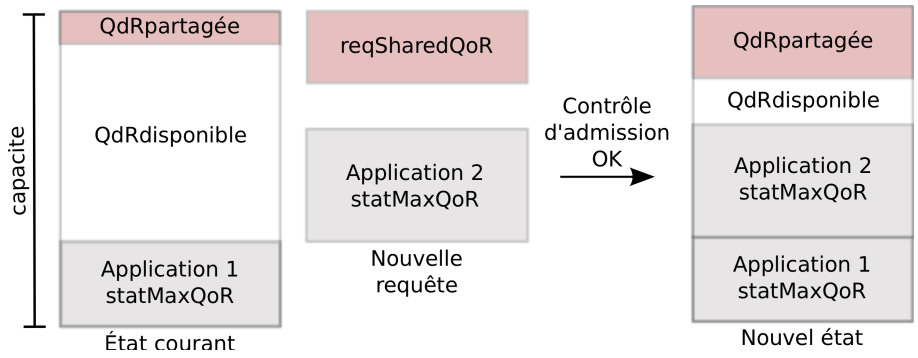


FIGURE 4.8 – Contrôle d'admission sur une ressource réseau

alors `iperf` l'utilise. La mesure donne la bande passante réellement obtenue et les caractéristiques de QdS du lien observées.

Si la bande passante obtenue est égale à la QdR requise ($maxQoR_i$), ARMOR vérifie les contraintes de QdS réseau. ARMOR vérifie que chaque caractéristique de QdS réseau (délai, gigue, taux de perte) est inférieure à la contrainte donnée dans le *manifest*. Si le test des contraintes de QdS réseau réussit, le contrôle d'admission pour les ressources réseau est validé.

4.3.1.2 Contrôle d'admission pour les ressources CPU

Pour les ressources CPU, la QdR requise s'exprime sur chaque équipement par les paramètres $(maxQor_{ik}, T_i)$ soit le maximum de QdR requise pour l'application i, sur l'équipement k et la période de l'application. La vérification des QdR CPU disponible est différente selon l'ordonnanceur utilisé sur chaque équipement. Nous détaillons le contrôle d'admission pour les deux ordonnanceurs supportés par ARMOR.

Ordonnanceur CFS L'ordonnanceur CFS affecte le CPU, aux différents *cgroups*, en fonction de leur poids. Il garantit que chaque *cgroup* aura une QdR CPU, proportionnelle à son poids, après une période T_{sched}. Par défaut, le noyau Linux fixe cette période à 20 millisecondes. Si le nombre de processus devient trop grand, le noyau Linux peut décider d'augmenter la période T_{sched}, pour limiter le temps passé à ordonnancer les processus. Cependant les équipements du réseau local domestique ont tendance à exécuter un nombre relativement faible d'applications. ARMOR fait l'hypothèse que le noyau Linux n'augmentera pas la période T_{sched}.

Les poids des *cgroups* sont relatifs les uns par rapport aux autres. Le contrôle d'admission consiste à vérifier que la somme des poids, des *cgroups*, ne dépasse pas la capacité de la ressource CPU. Le composant `cpu_Broker` d'ARMOR (composant du `Local CPU Resource Manager`, cf. Figure 4.7, page 84) connaît la QdR actuellement disponible sur la ressource. Le test d'admission (4.7) est donc utilisé. Ce test est effectué par le composant `LCRM_core`. La valeur $maxQoR_{ik}$, exprimée en %, est utilisée pour le poids de l'application lors de la réservation.

$$QdRdisponible_k \geq maxQoR_i \tag{4.7}$$

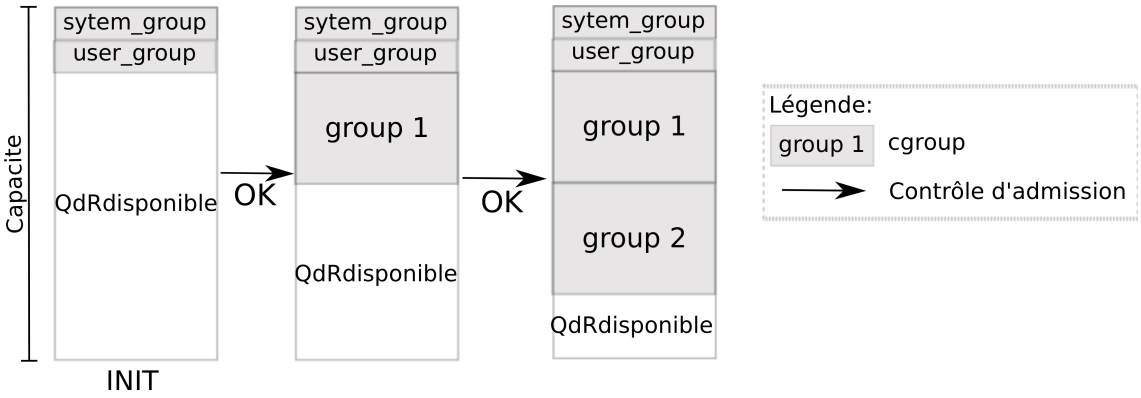


FIGURE 4.9 – Contrôle d'admission pour la ressource CPU, gérée par les *cgroups* et l'ordonnanceur CFS

L'équation (4.8) donne le calcul de la `QdRdisponible` après n réservations.

$$QdRdisponible_k = capacité_k - \sum_{i=1}^{n} maxQoR_{ik} - user\_group - system\_group \quad (4.8)$$

La Figure 4.9 illustre deux contrôles d'admission réalisés avec succès, pour deux applications de diffusion de contenus multimédia. Le premier test d'admission intervient après la phase d'initialisation. Aucune réservation n'a été faite sur cette ressource. Seuls les *cgroups user_group* et *system_group* existent. Pour le deuxième test d'admission, une réservation a été faite, pour laquelle ARMOR a créé le *cgroup group 1*.

Ordonnanceur CBS L'ordonnanceur CBS utilise des serveurs avec les paramètres (C,T) (budget,période) pour exécuter les applications. Ces serveurs CBS sont ordonnancés à l'aide de l'algorithme *Earliest Deadline First* (EDF). Avec cet algorithme, un ensemble de n *threads* indépendants est ordonnançable si l'équation (4.9) est vérifiée.

$$\sum_{j=1}^{n} \frac{C_{jk}}{T_j} \leq 1 \quad (4.9)$$

ARMOR vérifie qu'avec l'ajout d'un nouveau serveur CBS, l'équation est toujours vraie. Pour cela, ARMOR utilise le test (4.10), intégrant la nouvelle application. Le composant `cpu_Broker` connaît les réservations effectuées sur l'équipement, en interrogeant les mécanismes de réservation. Ce test est réalisé par le composant `LCRM_core`.

$$\sum_{j=1}^{n} \frac{C_{jk}}{T_j} + \frac{C_{ik}}{T_i} \leq 1 \quad (4.10)$$

Dans ce test C_{ik} est exprimé en millisecondes. Pour obtenir C_{ik} en millisecondes à partir de $maxQoR_{ik}$, en %, le composant `LCRM_core` utilise l'équation (4.11).

$$C_{ik} = \frac{maxQoR_{ik}}{capacité_k} * T_i \quad (4.11)$$

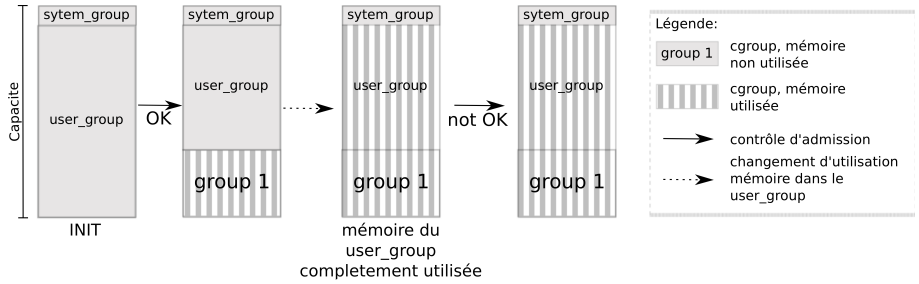


FIGURE 4.10 – Contrôles d'admission pour la ressource mémoire RAM

4.3.1.3 Contrôle d'admission pour les ressources mémoire RAM

Pour les ressources mémoire RAM, la QdR requise s'exprime à l'aide de $maxQoR_{ik}$, pour l'application i, sur l'équipement k. Le contrôle d'admission est réalisé localement (sur chaque équipement) par le composant LMRM. Ce composant vérifie que la QdR disponible est suffisante pour la nouvelle application (4.12).

$$QdRdisponible_k \geq maxQoR_{ik} \tag{4.12}$$

Nous avons vu que lors de la phase d'initialisation, ARMOR affecte une partie de la mémoire RAM au *cgroup system_group* et tout le reste au *cgroup user_group*. L'affectation du *cgroup user_group* est ensuite réduite à chaque fois qu'une nouvelle application est démarrée. La QdR disponible est donc donnée par l'équation (4.13), après n réservations.

$$QdRdisponible_k = capacité_k - \sum_{i=1}^{n} maxQoR_{ik} - system\_group \tag{4.13}$$

Si le test (4.12) réussit, la mémoire du *cgroup user_group* est diminuée de $maxQoR_{ik}$ lors de la phase de réservation. Cependant, les mécanismes de gestion de la mémoire RAM ne permettent pas de prendre de la mémoire à un *cgroup* si celui-ci l'utilise. Le contrôle d'admission doit donc s'assurer qu'il y a au moins $maxQoR_{ik}$ de mémoire RAM non utilisée dans le *cgroup user_group*. Si ce n'est pas le cas, un avertissement est remonté à l'utilisateur et le contrôle d'admission échoue.

La Figure 4.10 schématise deux contrôles d'admission. Le premier réussit car la mémoire RAM du *cgroup user_group* n'est pas utilisée. Ensuite, la mémoire RAM du *cgroup user_group* est utilisée. De ce fait, le deuxième test d'admission échoue.

4.3.1.4 Atomicité de la séquence de démarrage

Lors du traitement d'une demande de démarrage d'une application, la QdR disponible sur chaque ressource ne doit pas changer entre le premier test d'admission et la dernière réservation. Cette contrainte garantit l'atomicité du traitement des demandes de démarrage.

Toutes les demandes de démarrage sont faites au `Global Resources Manager` (GRM), dont une seule instance est présente dans le réseau local domestique. Le GRM traite les demandes de démarrage selon la politique premier arrivé, premier servi. Deux réservations ne peuvent pas se "mêler" et compromettre l'atomicité.

Pour les ressources réseau et CPU, grâce aux mécanismes de réservation mis en place par ARMOR, seules les réservations déjà effectuées sont considérées. Pour les

ressources réseau, le trafic émis par d'autres applications est traité après celui des applications multimédia. Pour le CPU, les applications sans réservation utilisent ce qui est laissé libre par les applications de diffusion de contenus multimédia. Pour ces ressources, le test d'admission est valable durant toute la phase de démarrage.

En revanche, pour la mémoire RAM, l'atomicité peut être compromise si l'utilisation de la mémoire RAM dans le *cgroup user_group* augmente entre les deux phases. En effet, la réservation peut échouer, alors que le test d'admission a réussi. Pour résoudre ce problème, ARMOR effectue la réservation de la mémoire RAM au moment du contrôle d'admission. De cette manière, une augmentation de l'utilisation mémoire RAM dans le *cgroup user_group* n'empêche pas la réservation. Si un autre test d'admission échoue, la réservation mémoire RAM est supprimée pour ramener le système dans un état cohérent.

4.3.2 Réservation des quantités de ressources

Une fois le contrôle d'admission effectué sur toutes les ressources utilisées par l'application, la phase de réservation est démarrée. Les réservations sont comptabilisées pour les prochains contrôle d'admission. Les réservations sont faites à l'aide des mécanismes du système d'exploitation de chaque équipement. Quand toutes les réservations sont effectuées, l'application est effectivement démarrée. Les QdR réservées pour l'application sont garanties grâce aux mécanismes de réservation. Une fois les réservations effectuées, les composants d'ARMOR n'interviennent plus pour cette application.

Nous présentons pour les ressources réseau, CPU et mémoire RAM, comment ARMOR effectue et garantit les réservations sans modifier les applications. Nous montrons aussi comment les applications utilisent les réservations.

4.3.2.1 Réservation des ressources réseau

Pour effectuer une réservation, ARMOR commence par mettre à jour ses composants de gestion des QdR réseau. Lors des demandes de démarrage futures, la nouvelle réservation est ainsi prise en compte. ARMOR met ensuite à jour la politique de gestion du trafic émis sur l'interface de l'équipement serveur. Cette politique réserve la QdR réseau sur l'interface serveur et garantit les réservations sur le lien [72]. Nous présentons ensuite comment ces réservations sont garanties, grâce à la politique mise en place par ARMOR. Enfin nous revenons sur la prise en compte de l'hétérogénéité des ressources réseau par ARMOR.

Mise à jour des composants d'ARMOR Les QdR requises pour une application de diffusion de contenus multimédia s'expriment à l'aide de $maxQoR_i$ (maximum d'utilisation) et de $statMaxQoR_i$ (maximum d'utilisation sans les pics).

Pour les deux ressources (interface réseau de l'équipement serveur et lien), les attributs du composant gérant la ressource sont mis à jour de manière identique. L'attribut `QdRdisponible` est diminué pour prendre en compte la réservation dédiée à l'application, comme le montre l'équation (4.14).

$$QdRdisponible = QdRdisponible - statMaxQoR_i \tag{4.14}$$

L'attribut `QdRpartagée` est étendu pour prendre en compte les pics de l'application, comme le montre l'équation (4.15). Pour une application i, la QdR requise pour les pics d'utilisation s'expriment par : $reqSharedQoR_i = maxQoR_i - statMaxQoR_i$. La `QdRpartagée` de la ressource est mise à jour si un pic de la nouvelle application est plus grand que la `QdRpartagée` actuelle de la ressource.

$$QdRpartagée = max(QdRpartagée, reqSharedQoR_i) \tag{4.15}$$

Pour gérer le liens Ethernet, ARMOR utilise des liens logiques. Un lien logique a une source et une destination. Un lien logique est donc composé de deux liens physiques : le lien physique entre l'équipement source et la passerelle domestique et le lien physique entre la passerelle domestique et l'équipement destination. Le premier lien physique est géré par l'interface de l'équipement serveur. Le deuxième lien physique est partagé par plusieurs liens logiques. Par exemple, sur la Figure 4.4 (page 76), les liens $PC_1 \rightarrow STB_1$ et $PC_2 \rightarrow STB_1$ ont la même destination et utilisent le lien physique *passerelle domestique* $\rightarrow STB_1$. ARMOR met donc à jour, de la même façon, les `QdRdisponible` et les `QdRpartagée` de tous les liens logiques ayant la même destination.

Pour les liens Wifi, il n'y qu'un seul lien physique. La réservation effectuée pour l'application a déjà modifié les `QdRdisponible` et `QdRpartagée` de ce lien.

Mise à jour de la politique de gestion du trafic émis Lors de la phase d'initialisation, ARMOR installe une politique de gestion du trafic émis. Cette politique est mise à jour quand une application est démarrée. La Figure 4.11 montre la politique installée lors de la phase d'initialisation et mise à jour par ARMOR, pour deux applications. La politique installée, lors de la phase d'initialisation, différencie le trafic multimédia du trafic par défaut. Cette différence est faite à l'aide de la *class* 1:1 et du filtre *Filter 1 : traffic type*. Le trafic multimédia est traité en priorité par rapport au trafic par défaut. Grâce à cette politique le trafic multimédia est aiguillé jusqu'à la *qdisc* 3:.

Lorsqu'une nouvelle application est démarrée, ARMOR met à jour la politique de gestion du trafic émis pour prendre en compte la nouvelle application. Pour ce faire, ARMOR rajoute une *class* fille à la *qdisc* 3:. La réservation mise en place sur l'interface de l'équipement serveur a deux objectifs :

1. garantir la QdR requise par l'application
2. limiter la QdR utilisée par l'application. Au niveau du lien, il n'est pas possible de différencier le trafic de deux applications multimédia. ARMOR s'assure donc qu'une application n'émettra pas plus que sa réservation. La réservation au niveau du lien, basée uniquement sur le contrôle d'admission, est donc garantie.

Pour répondre à ces deux objectifs, ARMOR crée une *class* de type *Hierachical Token Bucket* (HTB) pour chaque application. Par exemple, pour l'application 2, ARMOR crée la *class* 3:2.

Pour répondre au premier objectif, ARMOR utilise les paramètres *rate* des HTB. Le paramètre *rate*, d'une *class* (ou d'une *qdisc*) de type HTB, garantit que la *class* aura au moins *rate* QdR réseau. Pour faire une réservation dédiée à l'application 2, ARMOR fixe le paramètre *rate* de la *class* 3:2 à $statMaxQoR_2$. Il faut ensuite

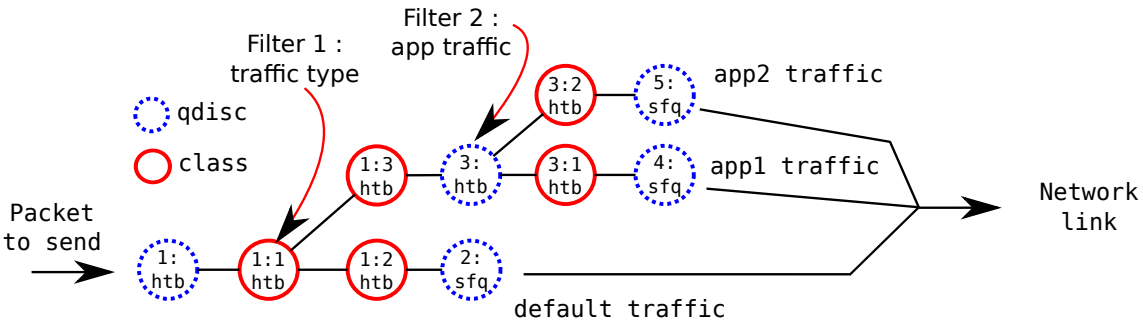


FIGURE 4.11 – Politique de gestion du trafic émis, installé par ARMOR, pour deux applications multimédia

s'assurer que l'application aura la QdR nécessaire pour ses pics d'utilisation. Les HTB sont hiérarchiques et allouent le trafic non utilisé par une *class* à une *class* sœur en priorité. ARMOR utilise un *qdisc* de type HTB pour exploiter cette propriété : la *qdisc* 3:. ARMOR fixe le paramètre *rate* de la *qdisc* 3: selon l'équation (4.16) (donnée pour n applications).

$$rate_{3:} = \sum_{i=1}^{n} maxQoR_i \tag{4.16}$$

ARMOR garantit ainsi que toutes les applications multimédia, utilisant la même interface sur l'équipement serveur, pourront émettre jusqu'à leur plus grand pic d'utilisation ($maxQoR_i$).

Rappelons qu'ARMOR fait l'hypothèse que deux pics, de deux applications de diffusion de contenus multimédia, n'auront lieu en même temps. Le contrôle d'admission se base sur cette hypothèse pour réduire la sur-réservation. Si deux applications émettent un pic en même temps, ARMOR ne garantit pas les QdR réseau des applications. Cependant, il est possible que la capacité de l'interface de l'équipement serveur le permette. Dans ce cas, il vaut mieux garantir la QdR aux applications multimédia plutôt que de la laisser au trafic par défaut de l'équipement. ARMOR fixe donc le paramètre *rate* de la *qdisc* 3: pour permettre aux applications d'émettre leur pic si c'est possible. Si la capacité du lien est suffisante, les deux pics d'utilisation réussiront à être émis. Si la capacité du lien n'est pas suffisante, un conflit aura lieu au niveau du lien. ARMOR traite toutes les applications multimédia avec la même priorité. Il est donc préférable d'avoir un conflit entre deux applications multimédia que de privilégier le trafic par défaut.

Pour répondre au deuxième objectif, ARMOR utilise les HTB pour façonner le trafic des applications multimédia, afin de le limiter. ARMOR fixe le paramètre *ceil* de la *class* 3:2 à $maxQoR_2$. L'application 2 ne peut pas émettre plus que son plus grand pic. Si une application essaye d'émettre plus que $maxQoR_i$ ses paquets sont stockés et envoyés avec du retard. Cependant, ARMOR se basant sur une estimation précise, cette situation ne devrait pas arriver. Le paramètre *ceil* d'une *qdisc* doit être supérieur ou égale à la somme des paramètres *ceil* de ses *class* filles. Le paramètre *ceil* de la *qdisc* 3: est donc fixé par l'équation (4.17).

$$ceil_{3:} = \sum_{i=1}^{n} maxQoR_i \tag{4.17}$$

Si deux applications multimédia émettent un pic en même temps, et si la capacité de l'interface réseau le permet, la politique de gestion du trafic émis n'introduit pas de retard. En effet, le paramètre *ceil* de la *qdisc* `3:` permet d'envoyer tous les pics.

Pour stocker les paquets d'une application multimédia, avant de les émettre sur le réseau, ARMOR utilise une *qdisc* de type SFQ. Cette *qdisc* ne limite pas et ne façonne pas le trafic. Le trafic de l'application 1 est envoyé à la *qdisc* `4:`. Le trafic de l'application 2 est envoyé à la *qdisc* `5:`.

Nous avons montré comment ARMOR réserve et garantit la QdR réseau, sur l'interface de l'équipement serveur, pour chaque application de diffusion de contenus multimédia. Pour différencier le trafic de deux applications multimédia, ARMOR utilise le filtre 2 (*Filter 2 : app traffic*) appartenant à la *qdisc* `3:`. Lorsqu'une réservation est faite, ARMOR modifie le filtre 2 pour prendre en compte la nouvelle application. Ce filtre se base sur l'adresse IP destination (*IP_DEST*) pour identifier le trafic de chaque application multimédia. Il n'est pas possible d'utiliser aussi le port destination (*PORT_DEST*) dans le filtre car les paquets envoyés sont fragmentés par le protocole UDP et les fragments ne contiennent pas le port destination. Cette limitation empêche le filtre de différencier le trafic de deux applications de diffusion de contenus multimédia ayant le même équipement serveur et le même équipement client. Cette limite pourrait être levée en utilisant les *cgroups* pour identifier les paquets réseau d'un *cgroup*[5]. Dans ce cas, le trafic émis par un *cgroup* serait affecté à la bonne *class*. Cependant, le support réseau des *cgroups* n'est pas encore disponible en standard sur les équipements utilisés. Dans sa version actuelle, ARMOR se base uniquement sur l'adresse IP destination.

Garantie des réservations Sur l'interface de l'équipement serveur, ARMOR utilise des *qdisc* et des *class* de type HTB pour garantir la QdR réseau réservée et la limiter (par façonnage).

ARMOR connaît la capacité de chaque lien. Les QdR réseau sont garanties au niveau des liens par le contrôle d'admission et les limites imposées sur l'interface de l'équipement serveur. La politique de gestion du trafic émis, installée lors de la phase d'initialisation, utilise une norme de QdS réseau si elle est disponible. Dans ce cas les paquets des applications de diffusion de contenus multimédia sont traités en priorité par la passerelle domestique. Si aucune norme de QdS réseau n'est disponible, la politique installée limite le trafic par défaut.

Les réservations de QdR réseau sont ainsi garanties pour toutes les ressources réseau utilisées par les applications de diffusion de contenus multimédia.

Prise en compte de l'hétérogénéité Les mêmes éléments de la politique de gestion du trafic émis sont utilisés pour les interfaces réseau Wifi et Ethernet, supportant ou non une norme de QdS réseau.

Lors de la phase d'initialisation, le composant `net_policy`, gérant l'interface réseau est configuré. Les différences de type (Wifi ou Ethernet) et la norme de QdS réseau sont masquées par ce composant. En dehors de ce composant, les QdR réseau sont gérées de la même manière par ARMOR.

5. `http://lwn.net/Articles/291161/`

Au niveau des liens, le composant GNRM traite les liens Wifi et Ethernet de la même façon. La seule différence porte sur les liens Ethernet physiques qui peuvent être partagés par plusieurs liens logiques.

4.3.2.2 Réservation des ressources CPU

ARMOR réserve localement les QdR CPU, sur chaque ressource utilisée. Les QdR CPU requises sont exprimées par les paramètres $(maxQoR_{ik}, T_i)$, pour l'application i sur l'équipement k (un équipement a une seule ressource CPU). Nous présentons comment les réservations sont effectuées et garanties pour les deux ordonnanceurs supportés par ARMOR. Ensuite, nous expliquons comment les composants d'ARMOR sont mis à jour. Puis, nous montrons comment ARMOR fait le lien entre l'application et les réservations. Enfin, nous revenons sur la prise en compte de l'hétérogénéité des ordonnanceurs par ARMOR.

Réservation de QdR CPU Les ressources CPU sont gérées localement par l'ordonnanceur de l'équipement. Nous présentons comment les réservations sont réalisées, pour les différents ordonnanceurs supportés par ARMOR.

Algorithme CFS L'ordonnanceur CFS se basent uniquement sur le poids des *cgroups*. Pour cet ordonnanceur, la phase de contrôle d'admission garantit, pour chaque ressource concernée, que la somme des poids des *cgroups*, avec la nouvelle application, est inférieure à la capacité de la ressource CPU ($\sum_{i=1}^{n} w_i + maxQoR_{ik} < capacité_k$).

ARMOR crée ensuite un nouveau *cgroup*, pour chaque équipement concerné, avec un poids de $maxQoR_{ik}$. Seul l'application multimédia est exécutée dans ce *cgroup*.

Algorithme CBS Pour l'ordonnanceur CBS, les deux paramètres (C_{ik}, T_i) sont utilisés. La phase de contrôle d'admission vérifie que l'ensemble des serveurs CBS, plus celui pour la nouvelle application, est ordonnançable.

Comme lors de la phase de contrôle d'admission, ARMOR transforme $maxQoR_{ik}$, exprimé en %, en C_{ik} exprimé en millisecondes. ARMOR crée ensuite un nouveau serveur CBS, sur chaque équipement, avec les paramètres (C_{ik}, T_i).

Mise à jour des composants d'ARMOR Pour l'ordonnanceur CFS, la `QdRdisponible` du composant `cpu_Broker` (composant du `Local CPU Resources Manager`) est diminuée de la réservation effectuée selon l'équation (4.18), sur l'équipement k.

$$QdRdisponible_k = QdRdisponible_k - maxQoR_{ik} \tag{4.18}$$

Pour l'ordonnanceur CBS, le composant `cpu_Broker` interroge directement les mécanismes de réservation, pour connaître tous les serveurs CBS créés sur un équipement. C'est pourquoi aucune mise à jour des attributs n'est nécessaire pour cet ordonnanceur.

Garantie des réservations

Ordonnanceur CFS Les poids des *cgroups* sont relatifs. ARMOR garantit que l'application a au moins $maxQoR_{ik}$ (exprimé en %) grâce au contrôle d'admission qui vérifie que la somme des poids est inférieure ou égale à la capacité de la ressource CPU [74]. Quand la somme des réservations est égale à la capacité de la ressource CPU, chaque *cgroup* a exactement sa part de CPU. Par contre, si la somme des réservations est inférieure à la capacité alors chaque *cgroup* a plus que sa réservation. L'objectif d'ARMOR est de garantir la QdR requise par une application. Cet objectif est atteint si l'application a plus que sa QdR requise.

Par exemple, si une réservation de 45% est faite pour une application sur l'équipement client, et si la ressource CPU de cet équipement a une capacité de 100% ; alors le *cgroup user_group* a un poids de 1%. Si aucune autre réservation n'est faite, sur la ressource CPU, l'ordonnanceur CFS donnera 45 fois plus de CPU à l'application multimédia qu'aux processus du *cgroup user_group* soit 97%. L'ordonnanceur CFS ne permet pas d'imposer de limite dure sur l'utilisation d'un *cgroup*. Cette répartition peut rendre l'équipement inutilisable si l'application de diffusion de contenus multimédia utilise plus de CPU que sa réservation. Pour limiter ce problème, ARMOR se base sur une estimation précise de la QdR requise (mesure et agrégation contrôlée). À l'inverse si des processus du *user_group* tentent d'utiliser 100% du CPU, l'application multimédia conserve la QdR requise grâce à la réservation effectuée.

Ordonnanceur CBS Pour l'ordonnanceur CBS, les réservations sont garanties par le test d'admission qui vérifie que l'ensemble des serveur CBS créés par ARMOR est ordonnançable [74].

Lien entre application et réservation Pour faire le lien entre l'application et la réservation, ARMOR utilise le PID [6] de l'application.

Pour l'algorithme CFS, nous avons développé un script pour attacher une processus à un *cgroup*. Le processus de l'application, sur l'équipement client et sur l'équipement serveur est attaché au bon *cgroup*, grâce à son PID. Le script attache tous les *threads* du processus au *cgroup* précédemment créé.

Pour l'algorithme CBS, ARMOR utilise un script, fourni dans le projet AQuoSA [7], pour attacher un *thread* à un serveur CBS. Tous les *threads* du processus, identifié par leur PID, sont attachés au serveur CBS précédemment créé.

Prise en compte de l'hétérogénéité À partir d'une seule valeur de QdR requise $(maxQoR_{ik}, T_i)$, exprimée en (%, millisecondes), ARMOR effectue des réservations pour deux ordonnanceurs. Ces deux ordonnanceurs, de types différents (ordonnanceur à parts équitables et ordonnanceur basé sur un serveur), gèrent la QdR différemment.

Lors de la phase d'initialisation, les composants locaux de gestion des QdR CPU (`LCRM_core` et `cpu_Broker`) sont configurés pour utiliser l'ordonnanceur de

6. identifiant du processus
7. `http://aquosa.sourceforge.net/`

l'équipement. Les services offerts par ces composants sont indépendants de l'ordonnanceur utilisé.

Les ordonnanceurs CFS et CBS sont des exemples d'ordonnanceurs, représentatifs des deux types d'ordonnanceurs étudiés. Pour supporter un autre ordonnanceur, du même type, dans ARMOR, il suffit de rajouter une configuration pour les composants de gestion locale de CPU.

4.3.2.3 Réservations des ressources mémoire RAM

ARMOR réserve localement les QdR mémoire RAM, sur chaque ressource utilisée. Les QdR mémoire RAM requises sont exprimées par $maxQoR_{ik}$, pour l'application i sur l'équipement k (un équipement a une seule ressource mémoire RAM).

Lors de la phase d'initialisation, ARMOR crée un *cgroup* pour les processus système (*system_group*) et un *cgroup* pour les processus utilisateur (*user_group*). ARMOR limite la mémoire RAM que chaque *cgroup* peut utiliser. Pour réserver la QdR requise par une application sur un équipement, ARMOR diminue la limite du *cgroup user_group*. L'équation (4.19) donne la nouvelle limite du *cgroup user_group*.

$$limite_{user\_group} = limite_{user\_group} - maxQoR_{ik} \tag{4.19}$$

ARMOR crée ensuite, sur chaque équipement, un *cgroup* dédié à l'application multimédia. ARMOR impose une limite dure (non dépassable) sur ce *cgroup* de $maxQoR_{ik}$. L'application n'utilise donc pas plus de mémoire RAM que $maxQoR_{ik}$.

L'algorithme de pagination utilise les limites des *cgroup* lors de la mise en mémoire tampon. La QdR mémoire RAM requise par l'application est garantie grâce aux limites imposées sur les autres *cgroup* (*user_group*, *system_group* et les *cgroups* créés pour les autres applications multimédia).

L'algorithme de pagination du noyau Linux utilise une variable appelée *swappiness* pour chaque *cgroup*. Cette variable définit à quel moment l'algorithme de pagination commence à mettre en mémoire tampon la mémoire d'un *cgroup*. La valeur du *swappiness* est comprise entre 0 et 100. Une valeur de 0 signifie que la mémoire tampon est utilisée seulement s'il n'y a plus de place en mémoire RAM ou que la limite du *cgroup* est atteinte. Une valeur de 100 définit une politique de mise en mémoire tampon très agressive. Par défaut le *swappiness* d'un *cgroup* est de 60. Pour les applications de diffusion de contenus multimédia, la QdR requise et réservée est le maximum de mémoire que l'application va utiliser. Pour prévenir la mise en mémoire tampon d'une partie de l'application de diffusion de contenus multimédia, ce qui introduit inévitablement un temps d'accès à la mémoire plus important, ARMOR fixe la variable *swappiness* du *cgroup* à 0. ARMOR se base sur une estimation précise de la QdR requise, l'application n'a donc jamais besoin d'utiliser la mémoire tampon. Pour le *user_group* et le *system_group*, la valeur par défaut de 60 est utilisée.

Pour faire le lien entre les applications et les réservations, nous avons utilisé le même script que pour les ressources CPU, qui affecte les *threads* d'un processus, identifié par son PID, à un *cgroup*.

4.4 Suppression des réservations effectuées

Pour supprimer toutes les réservations effectuées pour une application ARMOR utilise un identifiant global. Cet identifiant est généré par incrémentation unitaire par le composant `cons_com_mgr` (composant du `GRM`, cf. Figure 4.2, page 73). Lorsque la première application est démarrée, l'identifiant global généré porte le numéro 1. Lorsque l'application suivante est démarrée, l'identifiant global généré porte le numéro 2 et ainsi de suite.

L'identifiant global est transmis aux composants manager locaux (`LRM`) lors de l'étape de réservation. Chaque `LRM` associe les réservations effectuées à l'identifiant global. De cette manière, lors de la suppression d'une réservation, le composant local retrouve les réservations locales effectuées pour une application et peut les supprimer.

Lors de la suppression des réservations, ARMOR met à jour ses composants et modifie les mécanismes du système d'exploitation utilisés. Nous détaillons la suppression des réservations pour les ressources réseau, CPU et mémoire RAM.

Ressources réseau Pour une application, deux réservations sont effectuées sur les ressources réseau. Au niveau global une réservation est effectuée sur les liens utilisés. Au niveau local une réservation est effectuée sur l'interface de l'équipement serveur.

Le composant global de gestion des ressources réseau (le `GNRM`) effectue la mise à jour au niveau global. Le composant local de gestion des ressources réseau (le `LNRM`) effectue la mise à jour au niveau local. La mise à jour des composants d'ARMOR est identique au niveau global et au niveau local. Sur chaque ressource réseau, deux réservations sont effectuées par application : une réservation dédiée à l'application sans tenir compte de ses pics ($statMaxQoR_i$) et une réservation pour les pics de l'application ($reqSharedQoR_i$). Cette dernière est partagée par toutes les applications utilisant la même ressource réseau. Pour supprimer les réservations associées à une application, il faut supprimer la réservation dédiée et mettre à jour la réservation partagée. L'équation (4.20) donne la mise à jour effectuée sur les composants d'ARMOR, pour ne plus prendre en compte la réservation dédiée à l'application ($statMaxQoR_i$).

$$QdRdisponible = QdRdisponible + statMaxQoR_i \tag{4.20}$$

La réservation partagée (`QdRpartagée`) est fixée au plus grand pic sans tenir compte de l'application supprimée, comme le montre l'équation (4.21).

$$QdRpartagée = max(\cup_j^n reqSharedQoR_j) \tag{4.21}$$

Les réservations sur les ressources réseau sont mise en place à l'aide de la politique de gestion du trafic émis de l'interface serveur (cf. section 4.3.2.1). Lors de la phase de suppression, la `class` et la `qdisc` dédiées à l'application sont supprimées de la politique. Les paramètres `rate` et `ceil` de la `class` gérant le trafic multimédia (`class 3:`) sont modifiés pour ne plus prendre en compte l'application comme le montre les équations (4.22) et (4.23).

$$rate_{3:} = rate_{3:} - maxQoR_i \tag{4.22}$$

$$ceil_{3:} = ceil_{3:} - maxQoR_i \tag{4.23}$$

Ressources CPU Pour chaque application, une réservation est effectuée sur chaque CPU utilisé, par le composant manager local gérant les ressources CPU (le `LCRM`).

Deux ordonnanceurs sont actuellement supportés par ARMOR : l'ordonnanceur CFS (*Completely Fair Scheduler*) et l'ordonnanceur CBS (*Constant Bandwidth Server*). Pour l'ordonnanceur CFS, le composant `LCRM` utilise un attribut `QdRdisponible` pour connaître la quantité de CPU actuellement disponible. Cet attribut est mis à jour selon l'équation (4.24) pour ne plus prendre en compte la réservation. Pour l'ordonnanceur CBS, le composant `LCRM` utilise directement les mécanismes du système d'exploitation pour connaître la QdR disponible. Pour cet ordonnanceur, il n'y pas de mise à jour des composants d'ARMOR.

$$QdRdisponible_k = QdRdisponible_k + maxQoR_{ik} \tag{4.24}$$

Les réservations de CPU sont effectuées à l'aide d'un *cgroup* dédié pour l'ordonnanceur CFS et à l'aide d'un serveur CBS dédié pour l'ordonnanceur CBS. Dans les deux cas, cette réservation dédiée est supprimée. Pour identifier le *cgroup*, son nom est identique à l'identifiant de réservation global. Pour identifier le serveur CBS, le composant `LCRM` stocke l'identifiant du serveur CBS créé pour l'associer à l'identifiant de réservation global.

Ressources mémoire RAM Pour chaque application, une réservation est effectuée sur chaque ressource mémoire RAM utilisée, par le composant manager local gérant les ressources mémoire RAM (le `LMRM`).

Le composant `LMRM` met à jour l'attribut `QdRdisponible` comme le montre l'équation (4.25), pour ne plus prendre en compte la réservation.

$$QdRdisponible_k = QdRdisponible_k + maxQoR_{ik} \tag{4.25}$$

Les réservations mémoire RAM sont garanties en limitant la mémoire utilisable par le *cgroup* utilisateur et en créant un *cgroup* dédié pour l'application. Lors de la suppression d'une réservation, la limite d'utilisation du *cgroup* utilisateur est augmentée comme le montre l'équation (4.26). Le *cgroup* dédié à l'application est ensuite supprimé. De même que pour le CPU, le nom du *cgroup* dédié à une application est identique à l'identifiant de réservation global.

$$limite_{user\_group} = limite_{user\_group} + maxQoR_{ik} \tag{4.26}$$

4.5 Conclusion

Ce chapitre a présenté ARMOR, le *framework* de gestion des QdR mis en œuvre dans cette thèse.

Nous avons présenté l'architecture d'ARMOR, offrant deux niveaux de gestion des QdR : global et local. Les composants du niveau global gèrent les QdR partagées par plusieurs équipements et coordonnent la gestion des QdR, effectuée par les composants du niveau local sur chaque équipement. Nous avons montré qu'au travers de son architecture, ARMOR intègre les caractéristiques des ressources du réseau local domestique. Nous avons aussi montré que par son architecture, ARMOR maîtrise l'hétérogénéité des types de ressources et des équipements.

Nous avons ensuite détaillé les deux phases de fonctionnement d'ARMOR. Dans une phase d'initialisation, tous les composants d'ARMOR sont configurés et les mécanismes de réservation sont initialisés sur chaque équipement. Après cette initialisation, nous avons montré comment ARMOR garantit les QdR requises par une application de diffusion de contenus multimédia, sur l'ensemble des ressources réseau, CPU et mémoire RAM utilisées. ARMOR effectue un contrôle d'admission sur chaque ressource utilisée. Si tous les tests d'admission sont réussis, alors ARMOR utilise les mécanismes de réservation du système d'exploitation des équipements pour garantir les QdR requises. Les réservations effectuées par les composants d'ARMOR sont associées à une application à l'aide d'un identifiant global unique. Ces réservations peuvent ainsi être supprimées quand l'application est terminée.

Une étude montrant la capacité d'ARMOR à être mis en œuvre dans un réseau local domestique est détaillée dans le chapitre 6. Ce chapitre évalue aussi la capacité d'ARMOR à garantir les QdR requises par une application de diffusion de contenus multimédia, afin que la QdS utilisateur soit satisfaite.

Troisième partie

Évaluations

Chapitre 5

Évaluation de l'estimation et du stockage de quantités de ressources

Sommaire

Dans ce chapitre, nous évaluons d'abord les méthodes d'estimation des quantités de ressources (QdR) requises et leur faisabilité dans le cadre de notre étude. Étant donnée la quantité d'information à traiter dans le contexte hétérogène du réseau local domestique, nous avons proposé une politique d'agrégation. Nous montrons ensuite l'intérêt de cette politique. Enfin, nous évaluons les deux algorithmes d'agrégation développés pour automatiser l'agrégation.

5.1 Estimation des quantités de ressources requises

Pour estimer les QdR requises, il faut au préalable connaître les profils d'utilisation des ressources. Une fois ces profils acquis, il est possible de calculer les QdR requises.

5.1.1 Acquisition des profils d'utilisation

Dans cette thèse, nous avons utilisé deux méthodes d'acquisition des profils d'utilisation : par analyse du flux pour les ressources réseau et par simulation pour les ressources CPU et mémoire RAM. Lors d'une simulation, un scénario, contenant le flux et les paramètres d'encodage, les équipements et les ressources, est exécuté et les profils d'utilisation sont mesurés sur chaque ressource. Après avoir discuté l'intérêt des méthodes proposées, nous évaluons les outils utilisés pour montrer que les méthodes sont utilisables.

5.1.1.1 Intérêt des méthodes proposées

La méthode d'acquisition du profil d'utilisation d'une ressource réseau consiste à analyser le flux vidéo pour connaître la taille de chaque image. Ce profil est ensuite transformé pour être compatible avec les mécanismes de réservations. Sur les flux dont nous disposons, la taille du flux audio est en moyenne dix fois moindre que la taille du flux vidéo. De ce fait, le flux audio n'est pas considéré dans le profil d'utilisation des ressources réseau.

Les flux sont encodés avec des standards et des paramètres d'encodage différents. Estimer le profil d'utilisation en analysant directement le flux encodé masque ces différences. En effet, pour connaître le profil d'utilisation des ressources réseau pour un flux, il suffit de disposer d'un outil étant capable de lire le flux et d'en donner la taille des images. La connaissance du standard et des paramètres d'encodage du flux n'est donc pas nécessaire.

Le profil d'utilisation des ressources CPU et mémoire RAM est fonction des flux utilisés (contenu, standard et paramètres d'encodage), des logiciels utilisés, de la ressource elle-même et des autres ressources utilisées. Il est difficile de construire une fonction isolant chaque paramètre d'encodage. Pour ces ressources, il n'est pas possible d'analyser le flux ou ses paramètres d'encodage pour connaître le profil d'utilisation. L'estimation par simulation et mesure permet de connaître avec précision les QdR requises sur chaque ressource. Là encore, il n'est pas nécessaire de connaître tous les paramètres d'encodage du flux. La simulation est faite avec un flux déjà encodé. Le profil d'utilisation donne la QdR utilisée pour ce flux, visionné dans les conditions de la simulation.

5.1.1.2 Outils utilisés

Analyse du flux vidéo Pour analyser le flux, nous avons utilisé des outils existants. Pour les flux encodés dans un conteneur mov ou mp4, `mp4trackdump` est utilisé. Pour les flux encodés dans un conteneur avi, `avidump` est utilisé. Ces outils sont fournis par `mpeg4ip` [1], qui est un paquet standard des principales distributions Linux. Ces outils parcourent un flux conteneur et donnent la taille de toutes les images du flux vidéo et la taille des échantillons audio.

Mesure des profils d'utilisation CPU et mémoire RAM Pour mesurer les profils d'utilisation CPU et mémoire RAM, nous avons utilisé deux outils standards du noyau Linux. Tout d'abord, `top` mesure la QdR CPU utilisée par un processus.

1. `http://sourceforge.net/projects/mpeg4ip/`

id	CPU	mémoire
L_1	Intel(R) Core(TM) 2 Duo 1.06GHz	2000 Mo
PC_1	Intel(R) Core(TM) 2 Duo 3.00 GHz	2000 Mo
PC_2	AMD Athlon(tm) XP 2800+	2000 Mo

TABLE 5.1 – Configuration matérielle des équipements utilisés

`top` fonctionne avec un intervalle de temps et donne la QdR CPU utilisée par un processus durant le dernier intervalle.

`top` fournit aussi une valeur de QdR mémoire RAM utilisée. Cependant, cette valeur n'est pas très précise et il est difficile de savoir à quoi elle correspond vraiment. Les réservations mémoire RAM sont mises en place à l'aide des *control groups* (*cgroups*) fournis par le noyau Linux. Linux permet de connaître l'utilisation mémoire RAM courante d'un *cgroup*. Lors de la mesure, un *cgroup* contenant toute la mémoire RAM et toute la capacité du CPU est créé pour l'application, sur chaque équipement. Seule l'application est exécutée dans ce *cgroup*. Pour obtenir le profil d'utilisation mémoire RAM, nous avons développé un script qui récupère périodiquement la mémoire RAM utilisée par un *cgroup*. Les valeurs ainsi obtenues sont cohérentes avec les mécanismes de réservation utilisés.

Les méthodes d'acquisition utilisent uniquement des outils standards sur les équipements, ne sont pas intrusives et ne font pas d'hypothèse sur les logiciels utilisés. En effet, l'analyse du flux encodé ne dépend pas des logiciels utilisés pour le diffuser. La simulation et les mesures peuvent être faites pour n'importe quel logiciel, l'application étant identifiée par son PID [2] sur un équipement. Ces méthodes respectent ainsi les contraintes de non modification des équipements et des applications imposées par le domaine d'étude.

5.1.2 Calcul des QdR requises

Les QdR requises par une application sont calculées à partir du profil d'utilisation de chaque ressource.

Pour les ressources réseau, une fois le profil d'utilisation estimé, deux valeurs sont utilisées comme QdR requise :

- `maxQoR` : utilisation maximale
- `statMaxQoR` : utilisation maximale sans compter les pics d'utilisation.

Le temps d'analyse du flux et de calcul des valeurs de QdR requises est inférieur à une seconde en moyenne sur l'ordinateur portable L_1 (détaillé dans le Tableau 5.1). L'estimation peut donc être faite en ligne, lorsqu'une application est démarrée. Les QdR réseau n'ont pas besoin d'être stockées.

Pour les ressources CPU et mémoire RAM, le maximum du profil d'utilisation est utilisé pour la QdR requise (`maxQoR`). Une fois calculée, la QdR requise est stockée pour être utilisée lors du démarrage d'une application.

Nous verrons dans l'évaluation de notre *framework* de gestion des QdR, donnée dans le chapitre 6, que ces valeurs de QdR requises garantissent la QdS des applications de diffusion de contenus multimédia.

2. identifiant du processus

5.2 Agrégation

Les QdR requises sont stockées dans des tables, appelées tables de mapping. L'agrégation d'une table de mapping fournit un compromis entre la taille de la table de mapping agrégée produite et la perte que cette table engendre. Nous cherchons tout d'abord à évaluer l'intérêt du principe d'agrégation. L'agrégation proposée offre un accès aux données plus efficace si une relation d'ordre existe au sein de la table de mapping initiale. Nous regardons ensuite l'existence possible d'une relation d'ordre entre les ressources. Enfin une discussion est proposée.

5.2.1 Intérêt de l'agrégation

L'agrégation proposée dans le cadre de cette thèse regroupe les QdR requises similaires. Ces QdR sont remplacées par une valeur `aggMaxQoR` qui est le maximum des QdR requises, afin de garantir la QdS. Cette agrégation permet de réduire la quantité d'information à stocker et dans certains cas de diminuer le nombre de mesures nécessaires. Pour illustrer ces deux points, nous reprenons deux exemples d'agrégations (détaillés à la section 3.2.2).

Le Tableau 5.2 donne deux exemples d'agrégations possibles pour les ressources CPU. La table de mapping initiale 5.2a contient cinq valeurs de QdR requises, une par ressource CPU pouvant être utilisée. À partir de cette table de mapping, deux tables de mapping agrégées sont proposées : la table A_1 contenant deux valeurs d'`aggMaxQoR` et la table A_2 contenant une seule valeur d'`aggMaxQoR`. En utilisant la table de mapping A_1, deux valeurs `aggMaxQoR` doivent être stockées contre cinq en utilisant la table de mapping initiale. Pour chaque `aggMaxQoR` il faut cependant conserver l'information de la ressource utilisée. Pour la table A_1, il faut ainsi stocker 2 valeurs `aggMaxQoR` et 5 valeurs donnant la ressource CPU utilisée, soit 7 valeurs au total. Avec la table de mapping initiale, 10 valeurs doivent être stockées (5 QdR requises et 5 valeurs donnant la ressource CPU utilisée). Le gain sur le stockage est donc réduit. Ensuite, pour accéder à la bonne QdR requise il faut potentiellement regarder l'`aggMaxQoR` associée à toutes les ressources stockées. La lecture est donc en $O(n)$, n étant le nombre de ressources stockées ou la taille de la table de mapping initiale. De plus il est nécessaire d'effectuer la mesure des profils d'utilisation pour toutes les ressources potentiellement utilisables, au nombre de cinq dans notre exemple.

Dans le cas des paramètres d'encodage, on peut supposer l'existence d'une relation d'ordre. Ainsi, plus la valeur du paramètre d'encodage augmente (par exemple une taille d'images plus importante), plus la QdR requise augmente. Dans ce cas, il n'est pas nécessaire de conserver toutes les valeurs de paramètres d'encodage. Par exemple, les tables de mapping A_3 et A_4, agrégeant la table initiale 5.3a pour le paramètre d'encodage taille d'images, ne contiennent qu'une valeur de paramètre d'encodage par `aggMaxQoR`. Avec la table A_3, si une application veut diffuser le flux avec une taille d'images CIF, l'`aggMaxQoR` correspondant à la valeur VGA est utilisée ($CIF < VGA$). Pour la table A_3 quatre valeurs doivent être stockées (deux paramètres d'encodage et deux `aggMaxQoR`) contre dix en utilisant la table initiale (cinq paramètres d'encodage et cinq QdR requise). Pour accéder à la bonne valeur de QdR requise, les paramètres d'encodage étant triés, l'accès à la valeur est en

équipement client	`maxQoR` (en %)
STB_1	12
PC_1	26
PC_2	54
$Tablette_1$	25
L_1	69

(a) Table de mapping initiale pour les QdR CPU requises sur l'équipement client

équipement client	`aggMaxQoR` (en %)
$STB_1, PC_1, Tablette_1$	26
L_1, PC_2	69

(b) table de mapping agrégée A_1

équipement client	`aggMaxQoR` (en %)
$STB_1, PC_1, PC_2,$ $Tablette_1, L_1$	69

(c) table de mapping agrégée A_2

Table 5.2 – Exemples d'agrégations des ressources CPU

$O(log(m))$, où m est la taille de la table de mapping agrégée ($m \leq n$), n étant la taille de la table de mapping initiale.

L'agrégation est donc plus efficace, du point de vue de l'accès aux QdR requises, sur les paramètres d'encodage du fait qu'une relation d'ordre existe. On peut s'interroger sur l'existence d'une relation similaire pour les ressources.

5.2.2 Relation d'ordre sur les ressources

Certains équipements utilisent des périphériques plus ou moins adaptés à certains paramètres d'encodage ou à certains standards. Par exemple, la QdR CPU requise sur un équipement e_1 avec un DSP permettant de décoder des flux au standard *h.264* est inférieure à la QdR requise sur un équipement e_2 n'utilisant pas de DSP. Par contre pour un flux au standard *mpeg4-part2*, le DSP de l'équipement e_1 n'étant pas utilisable, la QdR requise peut être supérieure à la QdR requise sur l'équipement e_2. Pour les ressources CPU et mémoire RAM il n'est pas évident d'établir une relation d'ordre.

Cependant nous pensons qu'il est possible d'établir des classes d'équipements similaires dans lesquelles une relation d'ordre existe. Cette intuition a été vérifiée dans nos expérimentations. Nous avons réalisé de nombreuses mesures sur trois équipements : deux PCs (PC_1, PC_2) et un ordinateur portable (L_1), détaillés dans le Tableau 5.1. Une expérience consiste à décoder et afficher le même flux, avec les mêmes paramètres d'encodage, sur un équipement client différent (PC_1, PC_2, L_1). Les QdR requises sont ensuite calculées et comparées. La même expérience a été réalisée en faisant varier la taille des images du flux, le profil d'encodage, et un paramètre d'encodage spécifique à l'encodeur `x264` (paramètre appelé *crf* définissant une valeur relative de qualité, comprise entre 0 et 51). Les flux utilisés proviennent de `http://media.xiph.org`, de bandes annonces disponibles sur `http://trailers.apple.com/` et de `http://www.bigbuckbunny.org/`. Dans tous les cas, les QdR

taille images	`maxQoR` (en %)
CIF	17
VGA	35
hd480	46
hd720	76

(a) Table de mapping pour les QdR CPU requises sur l'équipement client

taille images	`aggMaxQoR` (en %)
VGA	35
hd720	76

(b) table de mapping agrégée A_3

taille images	`aggMaxQoR` (en %)
hd720	76

(c) table de mapping agrégée A_4

TABLE 5.3 – Exemples d'agrégations de paramètres d'encodage

requises sur l'équipement PC_1 sont inférieures aux QdR requises sur l'équipement PC_2, qui sont elles mêmes inférieures aux QdR requises sur l'équipement L_1. La relation d'ordre $PC_1 < PC_2 < L_1$ semble donc exister. De même que pour les paramètres d'encodage, on pourrait utiliser cette relation d'ordre lors de l'agrégation des ressources CPU ou mémoire RAM de ces trois équipements.

5.2.3 Discussion

L'agrégation proposée dans cette thèse réduit la taille des tables de mapping et le coût lié à ces tables (stockage et temps d'accès). Si l'agrégation est utilisée sur des paramètres d'encodage, alors le nombre de mesures à réaliser est aussi réduit car une relation d'ordre existe. Dans ce cas, l'`aggMaxQoR` correspondant à la valeur supérieure la plus proche du paramètre d'encodage est utilisée. Pour les ressources CPU et mémoire RAM, il est moins évident d'établir une relation d'ordre même si intuitivement cela paraît possible. Cette intuition serait à vérifier sur un nombre plus important d'équipements.

Si une `aggMaxQoR` est utilisée sans avoir préalablement mesuré et agrégé la QdR requise, la perte introduite ne peut être connue. Par exemple si le flux est diffusé avec une taille d'images x où $VGA < x < hd720$, et que la table A_3 est utilisée, la valeur `aggMaxQoR` de 76 est réservée. Dans ce cas on ne connaît pas la perte introduite. Cependant on sait que $VGA < x$, donc la QdR requise pour la taille d'images x est supérieure ou égale à 35. De même $x < hd720$, la QdR requise pour la taille d'images x est donc inférieure ou égale à 76. Ainsi, la perte introduite en utilisant la table A_3 pour une taille d'images x est comprise dans l'intervalle $[0, 41]$ ($[76 - 76, 76 - 35]$).

Cette information peut être prise en compte pour choisir quelles mesures effectuer. Par exemple si on considère qu'une perte de 45% est acceptable, alors il n'est pas nécessaire de mesurer les QdR pour les tailles d'images comprises entre VGA et $hd720$.

5.3 Algorithmes d'agrégation

Les deux algorithmes d'agrégation que nous avons développés visent à minimiser la taille de la table de mapping agrégée A_k, en respectant les bornes supérieures données en paramètres sur chaque critère. Pour évaluer la perte générée par une table de mapping agrégée A_k, trois critères sont utilisés :

- la perte maximale ($max(perte_{A_k})$)
- la moyenne des pertes ($mean(perte_{A_k})$)
- le nombre de faux négatifs introduits ($nbFaNeg(A_k)$)

Respectivement, les trois bornes passées en paramètres aux algorithmes d'agrégation sont :

- *boundMaxLossQoR* : $boundMaxLossQoR \geq max(perte_{A_k})$
- *boundMeanLossQoR* : $boundMeanLossQoR \geq mean(perte_{A_k})$
- *boundNbFaNeg* : $boundNbFaNeg \geq nbFaNeg(A_k)$.

Les deux algorithmes se basent sur des concepts mathématiques différents. Le premier est basé sur le *clustering*, le second est basé sur le *bin-packing*. Nous commençons par comparer la complexité et les performances des deux algorithmes d'agrégation. Puis une discussion portant sur l'utilisation des algorithmes d'agrégation et sur le choix des bornes est donnée.

5.3.1 Complexité des algorithmes

L'algorithme d'agrégation basé sur le *clustering* utilise l'algorithme *k-means*. Cet algorithme a une complexité en $O(kn)$ où k est le nombre de *clusters* et n le nombre d'éléments. Dans notre cas, les *clusters* sont les `aggMaxQoR`, k représente la taille de la table de mapping agrégée et n représente la taille de la table de mapping initiale. L'algorithme d'agrégation appelle le *k-means* en incrémentant unitairement le nombre de *clusters*, jusqu'à produire une table de mapping agrégée satisfaisant les paramètres d'entrée. Si ces paramètres ne permettent aucune agrégation, l'algorithme *k-means* est appelé n fois. La complexité de l'algorithme d'agrégation est en $O(nkn)$ ou $O(kn^2)$.

L'algorithme d'agrégation basé sur le *bin-packing* commence par trier la table de mapping initiale. Ce tri est en $O(nlog(n))$, n étant la taille de la table de mapping initiale. Chaque QdR requise n'est lue qu'une seule fois, l'algorithme principal est donc en $O(n)$.

L'algorithme basé sur le *bin-packing* a une complexité plus faible que l'algorithme basé sur le *clustering*. Cette différence s'explique car l'algorithme basé sur le *bin-packing* construit la table de mapping agrégée au fur et à mesure. En effet dans cet algorithme, la table de mapping est incrémentée d'une QdR requise à chaque itération. Au contraire, l'algorithme basé sur le *clustering* reconstruit la table de mapping agrégée à chaque itération.

5.3.2 Performances des algorithmes

Pour évaluer les performances des algorithmes, une table de mapping est générée aléatoirement. Cette table est ensuite agrégée avec différents paramètres pour chaque algorithme. Les deux algorithmes sont comparés en utilisant la taille des tables de mapping agrégées produites et le temps d'exécution.

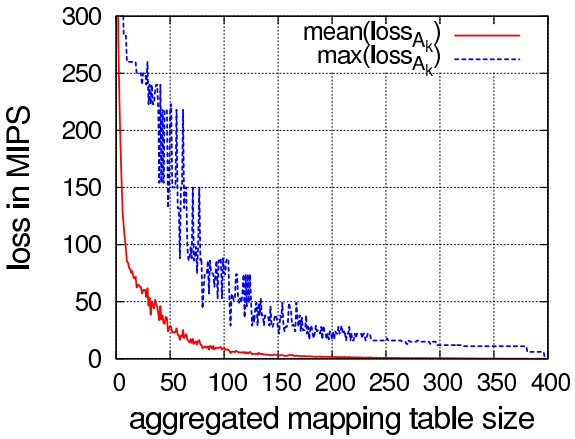


FIGURE 5.1 – Évaluation des tables de mapping générées

Dans cette évaluation, nous reprenons les données utilisées pour évaluer l'algorithme d'agrégation basé sur le *clustering* dans [73]. Les valeurs de QdR requises et les capacités des ressources CPU sont exprimées en *Million Of Instructions Per Second* (MIPS) [3].

Pour évaluer les algorithmes d'agrégation 400 QdR requises sont générées dans l'intervalle $[100, 2000]$. La capacité correspondante des équipements est générée dans l'intervalle $[500, 2500]$. Si la QdR requise est supérieure à la capacité de l'équipement alors la QdR requise est fixée à la capacité de l'équipement. Cette expérience simule l'agrégation de 400 ressources CPU. Étant donné qu'un équipement a une seule ressource CPU, cette agrégation se porte sur 400 équipements. Ce nombre peut paraître important pour un réseau local domestique. Cependant l'agrégation peut être utilisée par les opérateurs de télécommunication, pour garantir la QdS à leurs clients. Dans ce cas l'opérateur garantit la QdS sur 400 équipements qu'il aura préalablement testés.

Nous présentons d'abord les résultats obtenus avec l'algorithme basé sur le *clustering*. L'algorithme basé sur le *bin-packing* lui est ensuite comparé en utilisant les mêmes données.

5.3.2.1 Algorithme basé sur le *clustering*

La Figure 5.1 montre la perte introduite par une table de mapping agrégée (*loss in MIPS*) en fonction de sa taille (*aggregated mapping table size*). La perte maximale ($max(loss_{A_k})$) et la perte moyenne ($mean(loss_{A_k})$) sont données. Comme attendu, plus la taille de la table de mapping augmente, plus la perte introduite diminue.

Le Tableau 5.4 évalue 6 tables de mapping agrégées, générées par l'algorithme d'agrégation, pour 6 valeurs de bornes d'entrée passées en paramètres. Les trois premières colonnes du tableau donnent les bornes passées en paramètres à l'algorithme ($boundMeanLossQoR, boundMaxLossQoR, boundNbFaNeg$). Les colonnes quatre à six évaluent les tables de mapping générées par l'algorithme selon les critères d'évaluation $mean(perte_{A_k}), max(perte_{A_k})$ et $nbFaNeg(A_k)$. La colonne sept (taille) donne le nombre d'`aggMaxQoR` dans la table. Enfin, la dernière colonne (temps de

3. Pour passer d'une QdR ou d'une capacité exprimée en MIPS à une valeur en %, la valeur en MIPS est divisée par la fréquence du CPU

bornes passées en paramètres			tables de mapping agrégées				temps de calcul (en s)
boundMean LossQoR	**boundMax LossQoR**	**boundNb FaNeg**	**mean** ($\mathbf{perte_{A_k}}$)	**max** ($\mathbf{perte_{A_k}}$)	**nbFaNeg**($\mathbf{A_k}$)	**taille**	
2000	2000	2000	740	1880	164	1	0.3
200	600	100	190	416	22	5	13
200	400	20	145	394	14	6	18
100	500	100	85	260	1	11	52
200	400	2	37	218	1	48	515
0	0	0	0	0	0	374	6000

TABLE 5.4 – Résultats de l'algorithme basé sur le *clustering*

calcul) donne le temps en secondes mis par l'algorithme pour produire la table de mapping agrégée.

Ce tableau montre que des tables de mapping agrégées intéressantes sont trouvées par l'algorithme d'agrégation en un temps raisonnable. Par exemple à la ligne trois, pour les bornes passées en paramètres $boundMeanLossQoR = 200$ (soit 10% de la QdR requise maximale), $boundMaxLossQoR = 400$ (soit 20% de la QdR requise maximale) et $boundNbFaNeg = 20$ (soit 5% du nombre de lignes dans la table de mapping initiale) alors l'algorithme génère une table de mapping agrégée de taille 6 ($1,5\%$ de la taille initiale) en 18 secondes. Ce résultat montre que si les critères d'entrée sont bien choisis alors l'algorithme d'agrégation peut être utilisé, hors ligne, sur des tables de mapping de cette taille.

Par contre si les paramètres ne permettent pas une agrégation suffisante, le temps mis par l'algorithme devient prohibitif. Par exemple, pour la dernière ligne, l'algorithme met 6000 secondes à trouver la table de mapping agrégée. En effet, l'algorithme doit évaluer un grand nombre de table de mapping avant de trouver la bonne. De plus, plus la table à construire est grande, plus l'algorithme *k-means* est long à trouver une solution. On peut observer que la table de mapping agrégée contient seulement 374 éléments et non 400 (une ligne par QdR requise) comme on aurait pu le supposer quand il n'y a aucune agrégation. Cependant, les QdR sont générées aléatoirement et des doublons existent. De ce fait une `aggMaxQoR` est utilisée pour plusieurs QdR requises sans introduire de perte.

5.3.2.2 Algorithme basé sur le *bin-packing*

L'algorithme basé sur le *bin-packing* construit la table de mapping agrégée au fur et à mesure. Il n'est donc pas possible, contrairement à l'algorithme basé sur le *clustering*, de connaître les tables de mapping de taille 1 à n, n étant la taille de la table de mapping initiale. C'est pourquoi nous ne donnons pas la perte introduite par une table de mapping agrégée en fonction de sa taille.

Pour évaluer l'algorithme basé sur le *bin-packing*, les mêmes paramètres sont utilisés pour agréger la table de mapping initiale que pour l'évaluation de l'algorithme basé sur le *clustering*. Le Tableau 5.5 compare les résultats fournis par les deux algorithmes. Les 3 premières colonnes du tableau donnent les bornes passées en paramètres à chaque algorithme ($boundMeanLossQoR, boundMaxLossQoR, boundNbFaNeg$). Les colonnes 4 et 6 donnent la taille des tables de mapping agrégées produites par chaque algorithme.

bornes passées en paramètres			*clustering*		*bin-packing*	
boundMean LossQoR	**boundMax LossQoR**	**boundNb FaNeg**	**taille**	**temps (en s)**	**taille**	**temps (en s)**
1000	1000	1000	1	0,3	2	ϵ
200	600	100	5	13	5	ϵ
200	400	20	6	18	8	ϵ
100	500	100	11	52	10	ϵ
200	400	2	48	515	44	ϵ
0	0	0	374	6000	374	ϵ

TABLE 5.5 – Comparaison des résultats des deux algorithmes d'agrégation

Enfin, les colonnes 5 et 7 donnent le temps de calcul des tables de mapping agrégées. Les résultats pour l'algorithme de *clustering* sont ceux détaillés sur la Tableau 5.4. La perte et le nombre de faux négatifs introduits par les tables de mapping agrégées étant similaires pour les deux algorithmes, ils ne sont pas donnés dans le tableau.

On observe que les tailles des tables de mapping sont similaires pour les deux algorithmes. Par contre les temps de calcul de l'algorithme basé sur le *bin-packing* sont nettement inférieurs. En effet l'algorithme basé sur le *bin-packing* trouve une solution en un temps ϵ (inférieur à quelques millisecondes). Cette différence s'explique par la complexité inférieure de cet algorithme d'agrégation.

Afin d'évaluer les performances de l'algorithme basé sur le *bin-packing*, nous avons effectuée la même expérience avec 400000 QdR requises. Dans ce cas l'algorithme met 24 secondes. Pour un million de QdR requises, l'algorithme met 154 secondes plus 2 secondes pour le tri initial.

5.3.3 Discussion

Nous commençons par regarder dans quels cas les algorithmes d'agrégation proposés peuvent être utilisés. Les résultats des deux algorithmes dépendent des bornes passées en paramètres. Nous discutons ensuite le choix de ces valeurs.

Utilisation des algorithmes Les deux algorithmes peuvent être utilisés sur des tables de mapping contenant jusqu'à 400 QdR requises, à condition que les paramètres ne soient pas trop restrictifs. Si c'est le cas, seul l'algorithme basé sur le *bin-packing* donne une réponse en un temps acceptable. La complexité de l'algorithme basé sur le *clustering* pourrait être réduite en augmentant le nombre d'`aggMaxQoR` de manière plus importante dans les premières itérations (pour aller vite) et de revenir à une augmentation unitaire à la fin (pour être plus précis et générer une table de mapping agrégée de taille inférieure). L'algorithme basé sur le *clustering* devrait ainsi avoir de meilleures performances. Cependant cette amélioration n'a pas été implémentée.

Les deux algorithmes peuvent donc être utilisés pour agréger un paramètre d'encodage ou une ressource. Dans ce cas le nombre de QdR requises est relativement faible pour chaque table. L'algorithme basé sur le *bin-packing* est cependant plus rapide et le temps pour traiter l'ensemble des tables est plus faible avec cet algorithme. Vu le nombre de paramètres d'encodage, un grand nombre de tables existent.

Le temps de traitement de toutes les QdR requises peut ainsi varier fortement entre les deux algorithmes d'agrégation.

L'algorithme basé sur le *bin-packing* traite un million de QdR requises en un temps raisonnable. De ce fait, il peut aussi être utilisé pour agréger plusieurs paramètres d'encodage ou plusieurs flux en même temps. Cependant il n'existe pas de relation d'ordre entre plusieurs paramètres d'encodage. Par exemple, pour les paramètres d'encodage taille d'images et profil du standard *h.264*, les relations d'ordre suivantes existent :

- taille d'images : $CIF < VGA < hd480 < hd720$
- profil : $baseline < main < high$.

A partir de ces deux relations, on ne sait pas dire si la QdR requise pour un flux encodé avec une taille d'images VGA et le profil *main* est supérieure ou inférieure à la QdR requise pour le même flux encodé avec une taille d'images $hd480$ et le profil *baseline*. De ce fait, une agrégation sur plusieurs paramètres d'encodage ne permet pas de réduire le nombre de mesures et réduit moins efficacement la quantité d'information à stocker.

Choix des bornes passées en paramètres Les bornes passées en paramètres aux algorithmes d'agrégation expriment le compromis entre l'agrégation et la perte introduite. Il est difficile de donner une bonne valeur pour ces paramètres, de manière absolue. Nous donnons quelques indications pour le choix de ces valeurs.

Ces paramètres peuvent se baser sur les capacités des ressources. Par exemple pour agréger les paramètres d'un flux sur une ressource, on peut spécifier à l'algorithme que la perte moyenne doit être inférieure à 10% de la capacité de la ressource et que la perte maximale doit être inférieure à 20%.

L'utilisation d'`aggMaxQoR` au lieu de la QdR requise génère des faux négatifs lors du contrôle d'admission. Le nombre de faux négatifs, introduits par une `aggMaxQoR` supérieure à la capacité de la ressource, est contrôlé par la borne passée à l'algorithme d'agrégation ($boundNbFaNeg$). Par contre, les faux négatifs générés car plusieurs applications utilisent la même ressource ne sont pas pris en compte par les algorithmes. On peut cependant fixer les bornes de perte en fonction du nombre d'applications qui vont utiliser une même ressource. Par exemple, si une application est toujours seule à utiliser une ressource, alors une perte de 50% de la capacité de la ressource n'aura pas d'impact sur les autres applications. Au contraire, si une ressource est partagée par cinq applications, une perte de 50% de la capacité générera de nombreux faux négatifs lors du contrôle d'admission.

Ainsi, le choix des bornes, passées en paramètres aux algorithmes d'agrégation, est motivé par l'utilisation des ressources.

5.4 Conclusion

Dans ce chapitre nous avons évalué les méthodes d'estimation des QdR requises préconisées dans cette thèse. Cette évaluation a montré que ces méthodes sont utilisables dans le réseau local domestique.

Nous avons ensuite discuté l'intérêt de l'agrégation et évalué les algorithmes d'agrégation développés. Les résultats trouvés par ces deux algorithmes sont comparables. Cependant le temps de calcul de l'algorithme basé sur le *bin-packing* est

bien plus faible. Cet algorithme est donc mieux adapté à notre problème. Les algorithmes d'agrégation utilisent des paramètres, pour lesquels le choix de leur valeur est fonction de l'utilisation future des ressources.

L'agrégation proposée fonctionne sur une table de mapping existante et complète. Cependant, ce genre de table a vocation à évoluer au cours du temps. Par exemple, lors de l'ajout d'un nouvel équipement ou d'un nouveau paramètre d'encodage. On peut aussi vouloir construire une table de mapping avec les mesures actuellement connues et l'étoffer par la suite au lieu de reconstruire complètement la table de mapping agrégée. Les deux algorithmes d'agrégation construisent la table de mapping agrégée différemment mais elle est ensuite stockée de la même manière. L'ajout d'une nouvelle valeur est donc indépendant de l'algorithme d'agrégation utilisé.

Il est toujours possible de rajouter une nouvelle valeur de QdR requise dans une table de mapping agrégée existante. L'ajout le moins efficace au regard de la taille de la table est de créer une `aggMaxQoR` spécifique à la nouvelle QdR requise. Cet ajout n'augmentera pas la perte ou le nombre de faux négatifs introduits par la table de mapping agrégée. Cependant, cet ajout augmente la taille de la table. Pour éviter cela, il est possible d'utiliser une `aggMaxQoR` existante pour la QdR requise. Pour ce faire, il est nécessaire de conserver l'évaluation de la table de mapping agrégée ($(max(perte_{A_k}), mean(perte_{A_k}), nbFaNeg(A_k))$ et les bornes d'évaluation ($boundMeanLossQoR, boundMaxLossQoR, boundNbFaNeg$) afin de vérifier que l'ajout ne rendra pas la table invalide. Si ces valeurs sont connues, trois cas sont possibles pour ajouter la nouvelle QdR requise :

- augmenter une valeur `aggMaxQoR` existante à la QdR requise
- utiliser une valeur `aggMaxQoR` existante pour la QdR requise
- créer une nouvelle valeur `aggMaxQoR` spécifique à la QdR requise.

Le choix entre ces trois solutions est déterminé par l'évaluation de la table de mapping agrégée, après ajout de la nouvelle QdR requise. Entre les deux premières solutions, le choix générant le moins de perte est choisi. L'évaluation de la table de mapping agrégée ($(max(perte_{A_k}), mean(perte_{A_k}), nbFaNeg(A_k))$) est ensuite mise à jour. Si l'ajout avec un `aggMaxQoR` existante ne satisfait pas les bornes d'évaluation de la table de mapping agrégée alors une nouvelle `aggMaxQoR` est créée (troisième solution).

L'agrégation proposée permet de connaître les QdR requises par une application multimédia démarrée dans le réseau local domestique. Dans le chapitre suivant, nous nous intéressons à la gestion de ces QdR sur les ressources du réseau local domestique afin de garantir la QdS attendue.

Chapitre 6

Évaluation d'ARMOR

Sommaire

ARMOR est un *framework* de gestion des quantités de ressources (QdR). L'objectif d'ARMOR est de garantir les QdR requises par les applications de diffusion de contenus multimédia, distribuées dans le réseau local domestique. Pour ce faire, lors du démarrage d'une application, ARMOR vérifie que les QdR requises sont disponibles sur l'ensemble des ressources utilisées. Si c'est le cas, ARMOR met en place des réservations en utilisant les mécanismes de réservation fournis par le système d'exploitation des équipements.

Il est difficile d'évaluer formellement un tel *framework*. Dans ce chapitre, nous commençons par évaluer une mise en œuvre d'ARMOR. Cette évaluation, vise à montrer que l'implémentation d'ARMOR est réalisable, en respectant les contraintes du domaine d'étude : la non modification des applications et des équipements existants et le support de l'hétérogénéité des équipements et des contenus multimédia. Dans une deuxième partie de ce chapitre, nous évaluons la capacité d'ARMOR à garantir les QdR requises par les applications de diffusion de contenus multimédia. Pour ce faire, des applications sont démarrées et du bruit est généré sur les différentes ressources. Cette évaluation a pour objectif de montrer que les réservations mises en place par ARMOR garantissent effectivement les QdR requises par les applications, et donc la Qualité de Service (QdS) utilisateur attendue.

6.1 Mise en œuvre d'ARMOR

Dans cette section, nous évaluons la mise en œuvre d'ARMOR réalisée, dans le cadre de cette thèse. Nous commençons par expliquer comment l'architecture d'ARMOR a été mise en œuvre. Puis, nous regardons comment ARMOR intègre les caractéristiques du réseau local domestique. Ensuite, nous montrons que l'implémentation proposée répond aux contraintes du domaine d'étude. Nous nous intéressons ensuite aux temps pris par les opérations d'ARMOR. Enfin, nous évaluons la surréservation introduite.

6.1.1 Implémentation d'ARMOR

Pour mettre en œuvre ARMOR, nous avons utilisé le langage à composants *Mind*[1]. *Mind* est une implémentation en C du modèle à composant Fractal [18]. *Mind* a pour cible principale le développement de systèmes embarqués à composants et est donc bien adapté aux couches basses du logiciel. Le modèle à composant Fractal est hiérarchique : les composants peuvent contenir d'autres composants. ARMOR étant construit autour d'une architecture à composants, l'utilisation du modèle Fractal est donc appropriée.

ARMOR interagit, au travers des composants de gestion des QdR, avec le système d'exploitation des équipements. L'implémentation *Mind* est donc bien adaptée pour l'implémentation d'ARMOR. De plus, le compilateur *Mind* conserve les composants à l'exécution. Grâce à cette propriété, les composants de l'architecture d'ARMOR sont présents sur les équipements, lorsque que le *framework* est démarré.

Pour utiliser les mécanismes de réservation du système d'exploitation, des scripts *shell* ont été développés.

Nous donnons un chiffrage évaluant le coût de développement d'ARMOR.

Chiffrage de l'implémentation Pour chiffrer l'implémentation d'ARMOR, les métriques choisies sont le nombre de développeurs impliqués dans le projet, le nombre de composants du *framework* et le nombre de lignes de code.

Trois développeurs ont participé au développement d'ARMOR : un doctorant, un stagiaire M2 et un prestataire de service.

ARMOR se compose de 30 composants *Mind*. Au plus haut niveau hiérarchique, se retrouvent les composants `console`, `GRM` et `LRM` détaillés dans le chapitre 4. Le deuxième niveau de hiérarchie, c'est à dire les sous-composants directs des composants `GRM` et `LRM`, contient 21 composants. Sur ces 21 composants, 6 composants sont dédiés à la gestion des QdR, comme expliqué au chapitre 4 : les composants de gestion des ressources réseau, CPU et mémoire RAM au niveau global (`GNRM`, `GCRM`, et `GMRM`) et au niveau local (`LNRM`, `LCRM`, et `LMRM`). 3 composants gèrent la communication entre les composants distribués : les composants `lrm_com_mgr`, `grm_com_mgr` et `cons_com_mgr` (détaillés à la section 4.1). Ensuite 12 composants *wrappers* sont utilisés. Ces composants enveloppent des fonctionnalités spécifiques. Par exemple un *wrapper* existe pour l'accès aux fichiers XML, un autre existe pour l'accès aux *sockets* Linux. Ces dernières sont utilisées pour les invocations distantes. Enfin au plus bas niveau hiérarchique, le composant de gestion local des ressources réseau (le

1. `http://mind.ow2.org/`

`LNRM`) est composé de 4 composants (détaillés à la section 4.2.1) et le composant de gestion local des ressources CPU (le `LCRM`) est composé de 2 composants (détaillés à la section 4.2.2).

ARMOR contient 7500 lignes de codes. Cependant il est pertinent de noter que seulement 1000 lignes de codes sont utilisées pour les composants de gestion des ressources globales et 1000 lignes de codes sont utilisées pour les composants de gestion des ressources locales. La gestion des ressources au niveau global est réalisée entièrement en *Mind*. La gestion des ressources au niveau local est réalisée en *Mind* et à l'aide de script *shell*. Le reste du code est consacré à des fonctionnalités annexes telles que la communication entre les composants distribués ou l'accès aux fichiers XML. Ainsi le coût de développement d'ARMOR, notamment sur le cœur du *framework* : la gestion des ressources, se révèle faible.

L'ensemble du code d'ARMOR est disponible sous licence GPL [2] version 2 à l'adresse `https://sites.google.com/site/mlouvel/software`.

6.1.2 Connaissance du réseau local domestique

Les composants d'ARMOR connaissent tous les équipements du réseau local domestique, leurs caractéristiques et leur configuration. Dans l'implémentation actuelle d'ARMOR, cette connaissance provient d'un fichier XML, appelé "home file". Lors de la phase d'initialisation, les composants globaux d'ARMOR lisent ce fichier pour connaître les caractéristiques des équipements. Un exemple de ce fichier est donné en annexe B.1. Ce fichier contient les informations permettant d'accéder aux équipements (adresse IP, *hostname*), la capacité de chaque ressource, le type de ressources réseau, la norme de QdS supportée et l'ordonnanceur utilisé sur chaque équipement. Pour les liens et les interfaces réseau, le fichier contient les capacité théoriques. Enfin, le fichier précise les normes de QdS réseau supportées par la passerelle domestique.

Lors de la phase d'initialisation, le composant `Global Resources Manager` (GRM) parcourt ce fichier et initialise les données correspondantes sur chaque équipement rencontré. Le GRM lance l'initialisation de chaque composant local. Pour simplifier l'architecture d'ARMOR, le composant de gestion des ressources réseau du niveau global (`Global Network Resources Manager` GNRM) gère tous les liens, en accédant et modifiant directement les caractéristiques de QdS des liens dans le fichier "home file".

Les capacités réelles des interfaces Wifi sont différentes des capacités théoriques du standard. Lors de la phase d'initialisation, ARMOR effectue une mesure active, c'est-à-dire en envoyant des données sur le réseau, pour estimer la capacité réelle de chaque interface Wifi. Cette mesure est réalisée depuis l'interface qu'ARMOR initialise, vers l'équipement sur lequel est démarré le composant GRM. Si le composant GRM est démarré sur un équipement avec une interface ayant une capacité de réception limitée, alors les mesures actives des LRMs donneront des valeurs inférieures à la capacité réelle des interfaces. Pour régler ce problème, il faut installer le GRM sur la passerelle domestique ou sur l'équipement avec la plus grande capacité de réception.

2. *General Public License*

6.1.3 Respect des contraintes liées au domaine d'étude

Les contraintes du domaine d'étude (détaillées à la section 2.3) sont :

1. *Support des applications existantes*
2. *Support des équipements existants*
3. *Support de l'hétérogénéité des équipements et des contenus multimédia*

Nous présentons comment ARMOR satisfait ces contraintes et quelles sont les limites dues à ARMOR ou à des choix d'implémentation.

6.1.3.1 Support des applications existantes

ARMOR ne nécessite aucune modification des applications de diffusion de contenus multimédia. Pour connaître les QdR requises par une application, ARMOR analyse le flux vidéo (pour les QdR réseau) et utilise un *manifest* (pour les QdR CPU et mémoire RAM).

Avant le démarrage d'une application, ARMOR effectue les réservations nécessaires, sur les ressources que l'application va utiliser. L'application doit ensuite être liée aux différentes réservations. Pour les ressources réseau, ce lien se fait à l'aide de l'entête IP des paquets émis (champs *IP_DEST* et *PORT_DEST*). Pour les ressources CPU et mémoire RAM, ce lien se fait à l'aide du PID [3] du processus du logiciel serveur et du logiciel client de l'application. Nous avons développé un script *shell* associant un processus à une réservation CPU et mémoire RAM. Ce script appelé `attach_pid_to_cgroup.sh`, donné en annexe D, prend en argument le PID du processus et le *cgroup* dans lequel l'attacher.

6.1.3.2 Support des équipements existants

Dans sa version actuelle, ARMOR supporte uniquement les équipements avec un système d'exploitation Linux. Pour ces équipements, ARMOR utilise uniquement des fonctions standards du noyau, ne nécessitant pas de modification ni de re-compilation du noyau. En outre, ARMOR supporte aussi des équipements modifiés, pour intégrer un ordonnanceur *Constant Bandwidth Server* (CBS).

Nous présentons la configuration logicielle dont ARMOR a besoin sur un équipement. Ensuite, nous présentons comment un nouvel équipement peut être intégré à ARMOR.

Configuration logicielle requise sur un équipement Contrairement aux solutions existantes de gestion des QdR, présentées au chapitre 2, ARMOR ne suppose pas l'existence d'un intergiciel spécifique. ARMOR utilise directement les mécanismes de gestion des QdR, mis à disposition par le système d'exploitation de chaque équipement.

Pour gérer les ressources CPU et mémoire RAM, ARMOR se base sur les *cgroups* et l'ordonnanceur standard du noyau Linux : le *Completely Fair Scheduler* (CFS). À l'heure où nous écrivons ce manuscrit, la dernière version stable du noyau Linux est la version 2.6.39. Nous avons testé l'implémentation d'ARMOR sur la distribution Ubuntu [4]. Cette distribution est classiquement utilisée sur les PCs et ordinateurs

3. identifiant du processus
4. `http://www.ubuntu.com/`

portables actuels. La version du noyau fourni avec Ubuntu, utilisée dans nos évaluations, est la version 2.6.35-25. Le système de gestion des *cgroups* pour le CPU a été intégré dans la version standard du noyau Linux depuis la version 2.6.27. Le système de gestion des *cgroups* pour la mémoire RAM a été intégré dans la version standard depuis la version 2.6.35. Dans la configuration standard de la distribution Ubuntu, les *cgroups* pour la gestion du CPU et de la mémoire RAM sont activés par défaut. Pour les utiliser, il n'est pas nécessaire de recompiler le noyau Linux.

Pour les équipements ayant un ordonnanceur basé sur un serveur, de type CBS, l'implémentation actuelle d'ARMOR s'appuie le projet de recherche AQuosA [5]. Ce projet fournit un *patch* pour intégrer un ordonnanceur CBS dans le noyau Linux. La version actuelle d'ARMOR utilise le *patch* du noyau Linux fourni pour une version 2.6.32 du noyau Linux.

Pour gérer les interfaces réseau des équipements, ARMOR utilise deux outils : `iptables` [6] et `tc` (*traffic control*) [7]. `iptables` permet de masquer les paquets des applications multimédia, en modifiant la valeur du champ DSCP de l'entête IP. `tc` permet d'installer et modifier la politique de gestion du trafic émis sur l'interface de l'équipement serveur. Ces deux outils sont disponibles dans les principales distributions Linux.

Enfin, pour évaluer les capacités des interfaces réseau et les caractéristiques de QdS d'un lien, ARMOR utilise `iperf` [8], disponible dans les principales distributions Linux.

Intégration d'un nouvel équipement Pour intégrer un nouvel équipement dans ARMOR, l'équipement doit avoir la configuration logicielle précédemment présentée.

Il faut ensuite installer le composant de gestion des ressources locales (`Local Resources Manager` (LRM)) sur l'équipement. Pour supporter un nouvel équipement, il est possible de compiler le composant LRM sur cet équipement, d'utiliser directement une version binaire ou de l'intégrer dans un paquet Ubuntu, permettant son installation via le gestionnaire d'application de l'équipement.

Dans son implémentation actuelle, ARMOR se base sur un fichier XML "home file" pour connaître les équipements et les liens du réseau local domestique. Pour intégrer un nouvel équipement, il suffit de mettre à jour ce fichier. Si un nouvel équipement apparaît, dans le réseau local domestique, la phase d'initialisation doit être effectuée de nouveau. Cette limitation est purement liée au choix d'implémentation d'utiliser un fichier XML. Ce choix a été fait pour simplifier les problématiques de découverte automatique des équipements, des liens et de leurs caractéristiques. Ces problématiques sont partiellement résolues par l'utilisation de standard tel qu'UPnP. Cependant la découverte automatique des caractéristiques des équipements et des liens fait encore l'objet de recherche.

Une fois le LRM installé, démarré et initialisé sur l'équipement, des applications de diffusion de contenus multimédia peuvent être démarrées sur cet équipement. Pour connaître les ressources CPU et mémoire RAM requises, des mesures doivent

5. `http://aquosa.sourceforge.net/`
6. `http://www.netfilter.org/projects/iptables/index.html`
7. `http://lartc.org/`
8. `http://iperf.sourceforge.net/`

avoir été réalisées avec cet équipement et être renseignées dans le *manifest* utilisé par ARMOR.

6.1.3.3 Support de l'hétérogénéité

La gestion des QdR est spécifique à chaque type de ressources. De plus, un même type de ressources est géré différemment sur des équipements différents. Nous présentons tout d'abord comment ARMOR gère l'hétérogénéité des types de ressources. Ensuite, nous analysons comment ARMOR met en place des réservations sur des équipements hétérogènes. Enfin, l'estimation des QdR requises par une application de diffusion de contenus multimédia est rendue difficile par l'hétérogénéité des flux et des équipements. Nous montrons comment ARMOR répond à ce problème.

Hétérogénéité des types de ressources Pour prendre en compte l'hétérogénéité des types de ressources, ARMOR utilise un composant global et des composants locaux par type de ressources. Une instance du composant global (le GCRM, pour le CPU, par exemple) coordonne la gestion des QdR, effectuées par les composants corresponds au niveau local (les instances de LCRM, pour le CPU, par exemple). Les composants globaux offrent une interface indépendante du type de ressources. Les composants locaux offrent une interface spécifique. Cependant les méthodes de cette interface sont appelées uniquement par le composant correspondant au niveau global qui sait exprimer les QdR pour ce type de ressources.

Pour intégrer de nouveaux types de ressources dans ARMOR, le disque par exemple, un composant du GRM et un composant du LRM doivent être fournis. Les deux composants doivent respectivement implémenter les méthodes d'initialisation, de contrôle d'admission et de réservation. Le composant du GRM peut, comme pour les ressources réseau, effectuer des actions au niveau global en plus de la coordination des actions locales.

Hétérogénéité des mécanismes de réservation ARMOR supporte deux types d'ordonnanceurs. Tout d'abord, l'ordonnanceur CFS est un ordonnanceur à parts équitables qui se base sur le poids des processus. Ensuite, l'ordonnanceur CBS est un ordonnanceur basé sur un serveur utilisant les échéances des *threads*. ARMOR configure les composants locaux de gestion des QdR CPU, lors de la phase d'initialisation pour utiliser l'ordonnanceur de l'équipement. Ces composants fournissent ensuite les mêmes services de contrôle d'admission et de réservation.

De même, pour une interface réseau, le composant local de gestion des QdR réseau est configuré, lors de la phase d'initialisation, pour utiliser ou non une norme de QdS réseau et pour gérer une interface de type Ethernet ou Wifi. La bonne valeur du champ de l'adresse IP est ensuite utilisée par les composants d'ARMOR pour marquer les paquets. Au niveau des liens, ARMOR supporte les liens de type Wifi et Ethernet. Il n'y a qu'un seul lien Wifi qui est partagé par tous les équipements et plusieurs liens Ethernet. Pour gérer d'autres types de liens, ceux-ci doivent être classés dans une de ces deux catégories. Par exemple, les liens de type Courant Porteur en Ligne (CPL) seraient gérés par ARMOR comme des liens partagés. En effet, le courant est commun à tous les liens.

Avec l'implémentation actuelle d'ARMOR, nous avons montré sa capacité à masquer l'hétérogénéité des mécanismes de gestion des QdR fournis par le système d'exploitation. Pour supporter d'autres systèmes d'exploitation, une nouvelle configuration est ajoutée à chaque composant de gestion locale des QdR. Chaque composant est ensuite modifié pour utiliser les mécanismes de réservation fournis par le système d'exploitation. Les composants pour les ressources CPU, mémoire RAM et interface réseau sont modifiés.

Hétérogénéité des flux et des équipements Les contenus multimédia sont encodés avec différents logiciels, respectant différents standards. Lors de la phase de contrôle d'admission, ARMOR estime les QdR requises pour les ressources réseau en analysant le flux.

Pour masquer cette hétérogénéité, ARMOR utilise un composant dédié à l'estimation des QdR réseau. Le composant `eval_bw` est un sous-composant du composant de gestion locale des ressources réseau (LNRM). Lors de la phase de contrôle d'admission, le composant `eval_bw` est interrogé et analyse le flux vidéo. Pour faire cette analyse, le composant utilise les outils disponibles en fonction du standard d'encodage du flux. Ces outils analysent directement les images du flux encodé. Les paramètres d'encodage sont donc pris en compte automatiquement. Le composant `eval_bw` offre une interface indépendante du flux utilisé (conteneur, encodeur audio, encodeur vidéo)

Pour les ressources CPU et mémoire RAM, ARMOR se base sur un *manifest*. Les QdR CPU et mémoire RAM dépendent des flux utilisés, des logiciels utilisés, de la ressource et des autres ressources utilisées (utilisation ou non d'un DSP). Ce *manifest* est construit à l'aide de mesures et d'agrégations des QdR. L'hétérogénéité est ici masquée par l'agrégation détaillée au chapitre 3 et évaluée au chapitre 5 qui réduit le nombre de QdR requises à mesurer et stocker.

6.1.4 Temps des opérations d'ARMOR

ARMOR commence par configurer ses composants. Ensuite lorsqu'une application est démarrée, un contrôle d'admission et une réservation sont effectués sur chaque ressource utilisée. Nous analysons le temps nécessaire pour effectuer ces différentes opérations.

Pour un réseau local domestique avec les trois équipement PC_1, L_1 et BB_1 (détaillés dans le Tableau 6.3, page 128), la phase d'initialisation d'ARMOR met plus d'une minute en moyenne. Ce temps peut sembler très long mais la phase d'initialisation ne doit être faite qu'une seule fois, dans un contexte qui peut accepter ce délai. Lorsqu'une application est démarrée, le temps moyen (sur trois essais) pour effectuer tous les contrôles d'admission et toutes les réservations est de 11 secondes. Le démarrage de l'application est donc décalé de 11 secondes en moyenne. Ce temps apparaît trop long pour un réseau local domestique. Cependant ce temps est en grande partie dû à l'implémentation non optimisée d'ARMOR. En effet, les appels entre les composants distribués sont faits à l'aide de composants de communication qui, n'étant pas au centre de notre problématique, n'ont pas été optimisés.

Le Tableau 6.1 donne les temps des deux phases d'ARMOR au niveau local, sur chaque équipement. Pour la séquence d'initialisation le temps d'initialisation locale

	temps (en secondes)				
	initialisation		réservation		
équipement	**CPU,mem**	**réseau**	**CPU**	**mémoire RAM**	**réseau**
PC_1	50	1.1	0.2	0.2	0.1
L_1	23	1.1	0.6	0.6	0.2
BB_1	20	1.2	1	1	0.4

TABLE 6.1 – Temps des réservations locales

des ressources CPU et mémoire RAM et d'une interface réseau sont donnés. Dans l'implémentation actuelle d'ARMOR, les ressources CPU et mémoire RAM sont initialisées avec un seul script *shell*, le temps donné dans le tableau correspond à l'initialisation pour ces deux ressources. Pour la séquence de démarrage, le tableau détaille le temps nécessaire pour effectuer le contrôle d'admission et la réservation pour chaque ressource. Pour les ressources réseau, il faut rajouter le temps du test réel de bande passante, effectué pendant une seconde et le temps d'analyse du flux (pour connaître la QdR requise). Dans nos expériences, le temps d'analyse du flux et du calcul de la QdR requise est inférieur à une seconde sur l'ordinateur portable L_1.

Le temps de la phase d'initialisation est très important. L'initialisation des *cgroups* déplace tous les processus existants dans le *system_group* et le *user_group*. Pour déplacer un processus dans un autre *cgroup* mémoire, il faut mettre à jour la table des pages mémoire utilisées par le *cgroup*. Cette mise à jour explique le temps passé dans la phase d'initialisation. Le temps est plus faible sur l'équipement BB_1. Cet équipement exécute moins de processus qu'un PC classique, c'est pourquoi le temps est plus faible. Le temps est plus important sur l'équipement PC_1 car cet équipement possède un processeur 64 bits. La taille mémoire utilisée par page est donc plus importante. La phase d'initialisation n'étant effectuée qu'une seule fois, sa durée ne nous apparaît pas prohibitive.

Les temps de réservation sur chaque ressource sont beaucoup plus faibles. Sans compter la communication entre les composants d'ARMOR, le temps d'une réservation sur une ressource CPU ou mémoire est la somme des contrôles d'admission et des réservations. Pour le réseau, il faut rajouter le temps d'estimation de la QdR requise et le temps du test réel de bande passante. Le temps total de la phase de démarrage est fonction des équipements utilisés. Le Tableau 6.2 détaille le calcul du temps total pour trois scénarios, avec les trois équipements décrits dans le Tableau 6.1. La première colonne précise le scénario (équipement serveur → équipement client). Les colonnes deux et trois (contrôles d'admissions et réservations) détaillent respectivement le temps des contrôles d'admission et des réservations pour toutes les ressources d'un équipement. La colonne quatre donne le temps d'estimation des QdS réseau requise et du test réel de bande passante. Ce temps est toujours de deux secondes. La dernière colonne donne le temps total pour l'ensemble des contrôles d'admission et des réservations.

Le temps de la phase de démarrage est donc compris entre 3.9 et 5.8 secondes, pour ces équipements, ce qui paraît raisonnable.

Pour réduire ce temps, trois solutions sont envisageables :

1. optimiser les mécanismes de réservation pour les ressources CPU et mémoire

scénario	contrôles d'admission et réservations (s)		estimation QdR réseau, test réel (s)	temps total (s)
	serveur	client		
$PC_1 \rightarrow BB_1$	$0.2 + 0.2 + 0.1 = 0.5$	$1 + 1 + 0.4 = 2.4$	2	4.9
$PC_1 \rightarrow L_1$	$0.2 + 0.2 + 0.1 = 0.5s$	$0.6 + 0.6 + 0.2 = 1.4s$	2	3.9
$L_1 \rightarrow BB_1$	$0.6 + 0.6 + 0.2 = 1.4s$	$1 + 1 + 0.4 = 2.4s$	2	5.8

TABLE 6.2 – Temps total des réservations

RAM, le temps serait ainsi réduit sur chaque équipement ;

2. effectuer l'analyse de la QdR réseau requise hors ligne et la stocker avec les autres QdR requises, le temps total serait ainsi réduit d'une seconde ;
3. utiliser un modèle d'estimation des QdR réseau disponibles. Le test réel de bande passante ne serait pas nécessaire et le temps total serait ainsi réduit d'une seconde.

6.1.5 Sur-réservation

ARMOR utilise des valeurs statistiques pour abstraire les profils d'utilisation. Pour garantir la QdS des applications, les valeurs statistiques représentent des bornes supérieures aux profils d'utilisation. De ce fait, une sur-réservation est introduite car l'application n'utilise pas toujours la totalité de sa réservation. Il est difficile d'analyser formellement la sur-réservation introduite par ARMOR. En effet, elle dépend de la loi de distribution des profils d'utilisation qui est spécifique à chaque application. Pour illustrer ce phénomène de sur-réservation, nous nous basons sur l'exécution d'un scénario. Un scénario contient l'ensemble des paramètres utilisés lors de l'exécution d'une application de diffusion de contenus multimédia.

Pour illustrer la sur-réservation introduite par ARMOR, nous analysons la sur-réservation introduite pour le scénario 1, décrit dans le Tableau 6.4 page 129. Ce scénario est représentatif des autres expériences que nous avons réalisées. L'objectif est ici de donner un ordre d'idée de la sur-réservation introduite par ARMOR. Pour ce scénario, les profils d'utilisation sont mesurés et comparés aux réservations faites par ARMOR.

Réseau Pour le réseau, deux valeurs sont utilisées par ARMOR : `maxQoR` (QdR maximum utilisée) et `statMaxQoR` (QdR maximum sans les pics d'utilisation). La sur-réservation introduite par ARMOR est donnée par l'équation (6.1). Comme les pics sont partagés entre les applications, ils sont retirés du profil d'utilisation pour le calcul de la sur-réservation.

$$sur\ reservation = statMaxQoR - profil\ utilisation\ sans\ pic \tag{6.1}$$

La Figure 6.1 montre la QdR utilisée sur les ressources réseau pour le scénario 1 et les deux valeurs de QdR requises, estimées par ARMOR. Pour ce scénario, la sur-réservation introduite est en moyenne de 594 kbits (avec un écart type de 156), la médiane est de 622 kbits. La moyenne du profil d'utilisation est de 287 kbits/s, avec un écart de 193 kbits. La sur-réservation introduite est donc importante.

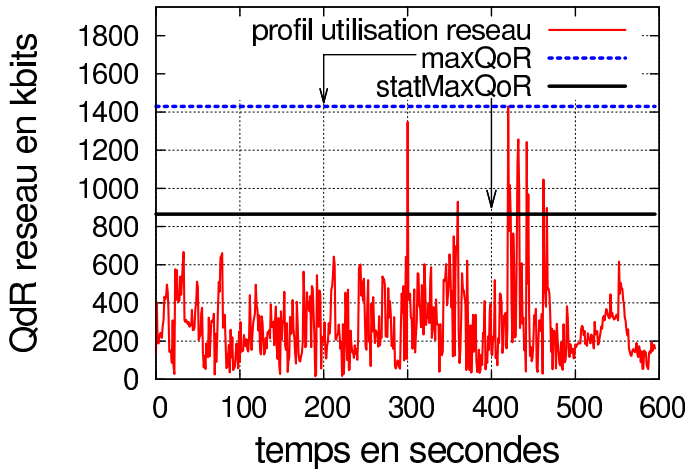
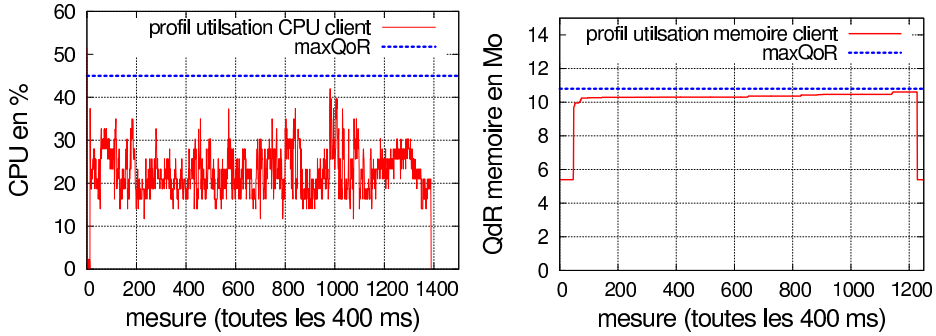


FIGURE 6.1 – Profil d'utilisation réseau et QdR requise



(a) CPU (b) mémoire RAM

FIGURE 6.2 – profils d'utilisations et QdR requises

La valeur `statMaxQoR` est fixée à $mean + 3\sigma$, selon l'inégalité de Bienaymé–Chebyshev. Cette inégalité dit que 89% des valeurs sont en dessous de ce seuil, et ce, quelque soit la distribution des données. Pour le scénario 1, le profil d'utilisation contient 596 valeurs. Le profil d'utilisation sans les pics (en retirant toutes les valeurs supérieures à `statMaxQoR`) contient 585 valeurs, soit 11 pics ($596 - 585$). Autrement dit seul $\frac{11}{596} \approx 2\%$ des valeurs sont au dessus de `statMaxQoR`. Si la loi de distribution du profil d'utilisation était considérée, ARMOR pourrait utiliser une formule plus adéquate et mieux limiter la sur-réservation. Par exemple, une valeur de $mean + 1.6\sigma$ dans le cas de la loi normale (où environ 89% des valeurs sont inférieures $mean + 1.6\sigma$).

CPU Pour le CPU, ARMOR utilise la valeur `maxQoR` pour toute la durée de l'application. La sur-réservation introduite par `maxQoR` est donnée par l'équation (6.2).

$$sur\ reservation = maxQoR - profil\ utilisation \tag{6.2}$$

La Figure 6.2a montre la QdR CPU utilisée et réservée (`maxQoR`) sur l'équipement client, pour le scénario 1. Pour ce scénario, la sur-réservation introduite est en moyenne de 22,5% (avec un écart type de 5,4%), la médiane est de 24%. Là encore, la sur-réservation est importante.

Mémoire RAM La Figure 6.2b montre la QdR mémoire RAM utilisée et réservée (`maxQoR`) sur l'équipement client, pour le scénario 1. Pour la mémoire RAM, ARMOR utilise la valeur `maxQoR` pour toute la durée de l'application. La sur-réservation introduite est ici négligeable car le profil d'utilisation mémoire RAM est stable pour la durée de l'application de diffusion de contenus multimédia.

6.2 Garantie des quantités de ressources et de la qualité de service

Pour évaluer la capacité d'ARMOR à garantir les quantités de ressources (QdR) requises, des applications de diffusion de contenus multimédia sont exécutées avec et sans ARMOR. Du bruit est ensuite généré sur les ressources utilisées. Les QdR utilisées par l'application sont mesurées et la QdS est évaluée à l'aide de mesures objectives et subjectives. Pour chaque évaluation, trois exécutions de l'application sont réalisées :

1. exécution de référence : seule l'application de diffusion de contenus multimédia est exécutée sur les équipements
2. bruit + application : l'application de diffusion de contenus multimédia est exécutée et du bruit est généré sur une ressource
3. bruit + application + ARMOR : l'application de diffusion de contenus multimédia est démarrée avec ARMOR (qui réserve les QdR nécessaires) et du bruit est généré sur une ressource.

Avant d'analyser les résultats des différentes expériences nous détaillons les équipements et les scénarios utilisés, puis nous détaillons les méthodes d'évaluation de la QdS.

Équipements utilisés Le Tableau 6.3 détaille les équipements utilisés dans nos évaluations. Pour chaque équipement, le tableau présente la configuration matérielle (CPU, mémoire RAM, interface réseau) et logicielle (distribution Linux, noyau, ordonnanceur et norme de QdS réseau supportée). La deuxième colonne donne les caractéristiques de quatre ordinateurs portables avec une configuration identique. La troisième colonne donne les caractéristiques d'un PC. La dernière colonne donne les caractéristiques d'une *beagleboard*[9] (BB_1). La *beagleboard* est un équipement embarqué, conçu pour faire du multimédia, et supporté par une communauté open source. Sa configuration matérielle est assez proche de celle d'une Set-Top-Box (STB)[10] ou d'un téléphone mobile. C'est pourquoi cet équipement apparaît pertinent pour valider la capacité d'ARMOR à être utilisé dans un réseau local domestique. Pour la *beagleboard*, deux configurations du système d'exploitation sont disponibles. *c1*, avec un noyau Linux standard et un ordonnanceur CFS et *c2*, avec un noyau Linux modifié grâce au *patch* du projet AQuoSA et un ordonnanceur CBS.

La passerelle domestique est un asus wl700ge. Cet équipement a les fonctions d'un routeur, d'un commutateur Ethernet et d'un point d'accès Wifi de norme G.

9. `http://beagleboard.org/` Les tests réalisés ici utilisent une version C4 de la *beagleboard*
10. Une STB est un équipement fourni par l'opérateur de télécommunication pour décoder des flux et les afficher sur une télévision.

	$\mathbf{L_1, L_2, L_3, L_4}$	$\mathbf{PC_1}$	$\mathbf{BB_1}$	
configuration	***c***	***c***	***c1***	***c2***
CPU	Intel(R) Core(TM) 2 Duo 1.06GHz	Intel(R) Core(TM) 2 Duo 3.00 GHz	ARM Cortex-A8 700 Mhz	
ordonnanceur	CFS	CFS	CFS	CBS
capacité mem RAM (en Mo)	2000	2000	256	
interface	Wifi G	Wifi G	Wifi G	
QdS réseau	802.11e	non	non	
distribution Linux	Ubuntu maverick	Ubuntu maverick	Ubuntu maverick	
noyau	standard 2.6.35-25	standard 2.6.35-25	standard 2.6.35-25	AQuosA 2.6.32

TABLE 6.3 – Caractéristiques et configurations des équipements (*mem* : mémoire)

La fonction point d'accès possède deux configurations : une supportant la norme de QdR réseau 802.11e et une ne supportant pas la norme 802.11e.

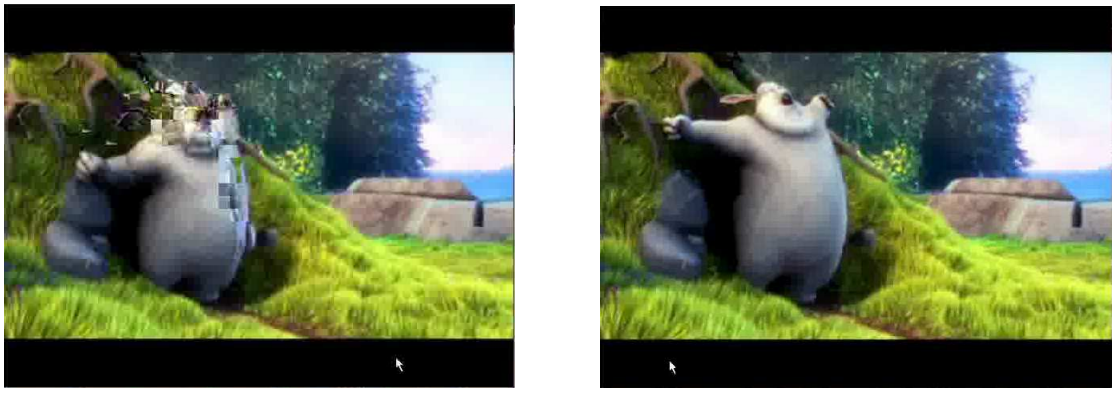
Scénarios utilisés Le Tableau 6.4 décrit les scénarios utilisés. Sur l'équipement serveur, le logiciel `vlc` est utilisé. Sur l'équipement client les logiciels `mplayer` et `vlc` sont utilisés. La diffusion est réalisée en mode *push*, où le logiciel serveur commence à envoyer le flux dès qu'il est démarré. Par défaut, `mplayer` stocke 8192 kbits avant de commencer à afficher le flux et `vlc` utilise un cache de 3 secondes. Deux flux sont utilisés :

- *big buck bunny* (scénario 1 et 2), d'une durée de 596 secondes. Deux versions du flux *big buck bunny* sont utilisées :
  - celle du scénario 1 : ici le flux a été réencodé à partir du fichier original [11] à l'aide de l'encodeur `x264`. Les paramètres utilisés pour l'encodage sont la taille des images au format CIF (352x288 pixels), le profil de base et le niveau 1.3 du standard *h.264* et une fréquence d'images de 25 images par secondes. Seul le flux vidéo est présent dans le flux conteneur.
  - celle du scénario 2 : le fichier original encodé au standard *h.264*, avec une taille d'images hd480 (852x480 pixels), le profil *main* et le niveau 3 du standard *h.264* et une fréquence d'images de 25 images par secondes. Le flux audio est encodé avec le standard *aac*.
- la bande annonce du film *Harry Potter 6* (scénario 3), d'une durée de 102 secondes : le fichier original [12], encodé au standard *h.264*, avec une taille d'images de 640x276 pixels, le profil *main* et le niveau 2.1 du standard *h.264* et une fréquence d'images de 25 images par secondes. Le flux audio est encodé avec le standard *aac*.

11. disponible à `http://www.bigbuckbunny.org/`
12. disponible sur `http://trailers.apple.com/` Le flux décrit ici est la version HP6_Large.mov sur le site

S	Flux	encodeur	paramètres	équipements		ressources	
				srv	clt	srv	clt
1	big buck bunny	*h.264*	profil base niveau 1.3 taille images : CIF fréq. im. : 25 im/s	PC_1	*beagleboard*	CPU	CPU
2	big buck bunny	*h.264*	profil : main niveau 3 taille images : hd480 fréq. im. : 25 im/s	L_3	L_1	CPU	CPU
3	Trailer Harry Potter 6	*h.264*	profil : main niveau 2.1 taille images : hd480 fréq. im. : 25 im/s	L_3	L_1	CPU	CPU

TABLE 6.4 – Scénarios d'évaluation



(a) mauvaise QdS utilisateur (b) bonne QdS utilisateur

FIGURE 6.3 – Exemple d'évaluation subjective de la QdS utilisateur

Méthode d'évaluation de la Qualité de Service Pour évaluer la QdS utilisateur, le flux est visionné sur l'équipement client et la qualité perçue est mesurée subjectivement. La QdS utilisateur peut être bonne ou mauvaise. La Figure 6.3 donne un exemple, d'une image dans une diffusion, où la QdS utilisateur est mauvaise et un exemple où la QdS utilisateur est bonne. Sur la Figure 6.3a, la qualité perçue est mauvaise car la tête du lapin est fortement pixelisée. Sur la Figure 6.3b, la qualité perçue est bonne car l'image est complète, sans pixelisation.

Pour évaluer la QdS de la couche application, la trace d'exécution du logiciel client est utilisé. Les deux logiciels clients utilisés lèvent une erreur lorsqu'une image de référence n'a pas pu être décoder à temps :

```
"number of reference frames exceeds max (probably corrupt
input), discarding one"
```

Pour évaluer la QdS de la couche application, le nombre d'erreurs levées est mesuré.

Nous détaillons maintenant les expériences réalisées pour chaque type de ressource, en commençant par les ressources réseau.

6.2.1 Ressources réseau

Le bruit sur les ressources réseau est généré à l'aide de l'outil `iperf`. Cet outil génère du trafic depuis un client `iperf` vers un serveur `iperf`. Dans les expériences décrites ici, chaque client `iperf` essaie d'envoyer 100 megabits/s, en utilisant le protocole UDP. Les paquets envoyés ont une taille de 1470 octets, correspondant à la taille maximale de paquets UDP appelée *Maximum Transmission Unit* (MTU). ARMOR supporte deux types de ressources réseau : Wifi et Ethernet. Cependant, avec les équipements du réseau local domestique, il est difficile de générer un bruit assez important pour perturber le trafic des applications de diffusion de contenus multimédia, sur des liens Ethernet. De ce fait, cette évaluation se focalise sur les liens et les interfaces réseau Wifi.

La QdR réseau utilisée par l'application, est mesurée à l'aide de `vnstat` [13]. Cet outil mesure la QdR réseau émise et reçue sur une interface. Les mesures présentées dans cette évaluation correspondent aux données reçues sur l'interface de l'équipement client. Pour mesurer uniquement la QdR de l'application de diffusion de contenus multimédia, elle est seule à émettre sur l'équipement serveur et aucune donnée n'est envoyée à l'équipement client.

En plus de la QdR réseau requise, les applications de diffusion de contenus multimédia sont sensibles au délai et à la gigue réseau. Pour évaluer les réservations faites par ARMOR, le délai est mesuré à l'aide de l'outil `ping`. Cet outil envoie un paquet vers un équipement et mesure le temps aller-retour en millisecondes. Pour estimer le délai des paquets d'une application de diffusion de contenus multimédia, un `ping` est effectué avec une période de 0.5 seconde entre l'équipement serveur et l'équipement client. Les valeurs collectées sont divisées par deux pour obtenir le délai. La surcharge introduite est considérée comme négligeable du fait de la faible taille et fréquence des paquets envoyés.

ARMOR est utilisable dans un réseau local domestique supportant ou non une norme de QdS réseau. Ainsi nous détaillons l'évaluation, sur les QdR réseau de type Wifi, avec et sans support de QdS réseau. Nous commençons par les réseaux locaux domestiques où aucune norme de QdS réseau n'est supportée.

6.2.1.1 Wifi sans QdS réseau

Dans cette expérience, l'interface réseau de l'équipement serveur ne supporte pas de norme de QdS réseau et la passerelle domestique est configurée pour ne pas utiliser de norme de QdS réseau. ARMOR limite donc le trafic de tous les équipements du réseau local domestique. La Figure 6.4 montre l'environnement expérimental. Une instance de composant LRM est démarrée sur chaque équipement du réseau local domestique. L'instance du composant GRM est démarrée sur l'équipement L_2 (il n'est pas possible de l'installer sur l'équipement faisant office de passerelle domestique). Le trafic de l'application de diffusion de contenus multimédia est représenté, sur la figure, par la flèche continue, le bruit généré est représenté par les flèches en pointillé.

Dans cette expérience, le scénario 1 est exécuté (cf. Tableau 6.4). L'équipement client est ici la *beagleboard* BB_1, avec la configuration $c1$. Le logiciel `mplayer` est utilisé sur l'équipement client. Le Tableau 6.5 donne les QdR réseau requises de

13. `http://humdi.net/vnstat/`

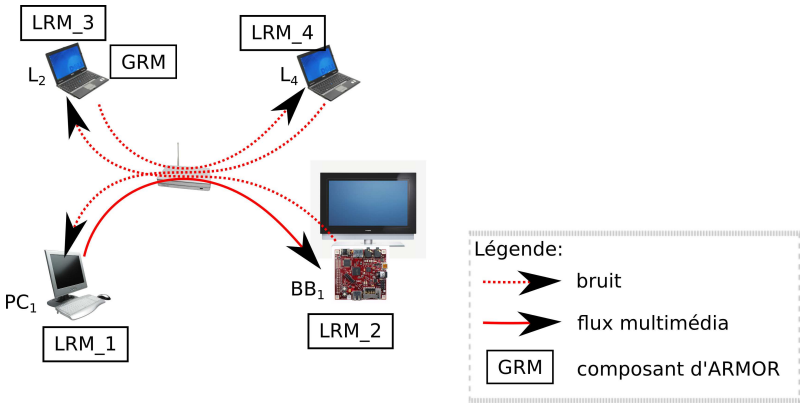


FIGURE 6.4 – Environnement expérimental, sans QdS réseau

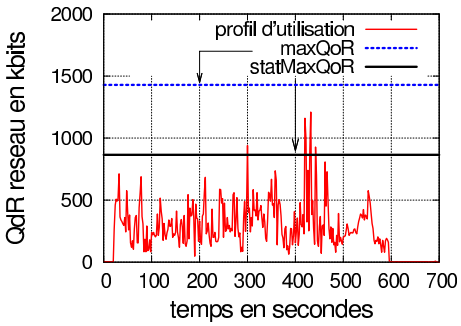
	estimation	valeur (en kbits/s)
QdR requise	`maxQoR`	1428
	`statMaxQoR`	865
	`reqSharedQoR`	563
	moyenne	286
lien	`capacité`	54000
	`QdRdisponible`	52572
	`QdRpartagée`	563
interface réseau serveur	`capacité`	5810
	`QdRdisponible`	4382
	`QdRpartagée`	563

TABLE 6.5 – Estimations faites par ARMOR des QdR requises et disponibles sur les ressources réseau utilisées

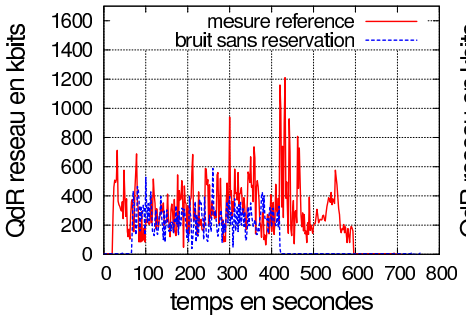
l'application de diffusion de contenus multimédia, les caractéristiques de QdS du lien et de l'interface réseau de l'équipement serveur (`capacité`, `QdRdisponible` et `QdRpartagée`) estimées par ARMOR. Ces valeurs sont données une fois la réservation effectuée pour l'application.

Nous regardons maintenant les QdR utilisées par l'application pour les trois exécutions. Nous nous intéressons ensuite à la Qualité de Service perçue lors des trois exécutions. Enfin, une discussion explique les observations.

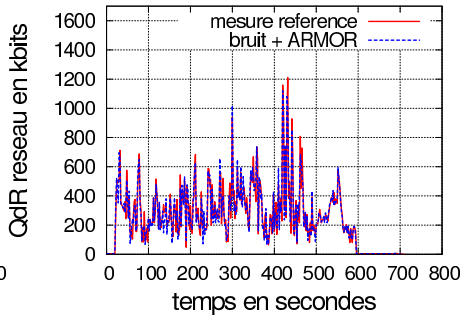
Observations des quantités de ressources La Figure 6.5a donne la courbe de référence mesurée, sans aucune autre application sur les équipements et la QdR requise estimée par ARMOR. On peut observer que la valeur `maxQoR`, estimée par ARMOR à 1428 kbits/s, est au dessus du maximum mesuré de 1290 kbits/s. Cette différence s'explique car la mesure, réalisée par `vnstat`, à une granularité de 2 secondes. Le profil d'utilisation des ressources réseau, calculé par ARMOR, a une granularité d'une seconde (pour être en accord avec les mécanismes de réservation du système d'exploitation). Or si l'on regarde les valeurs de ce profil, seules deux valeurs sont au dessus de 1290 : 1428 et 1347. Ces deux valeurs sont réparties dans le flux et sont donc "absorbées" lors la mesure qui est la moyenne de la QdR utilisée sur 2 secondes.



(a) référence et QdR requise



(b) pas de réservation



(c) utilisation d'ARMOR

FIGURE 6.5 – Mesures des QdR réseau utilisées

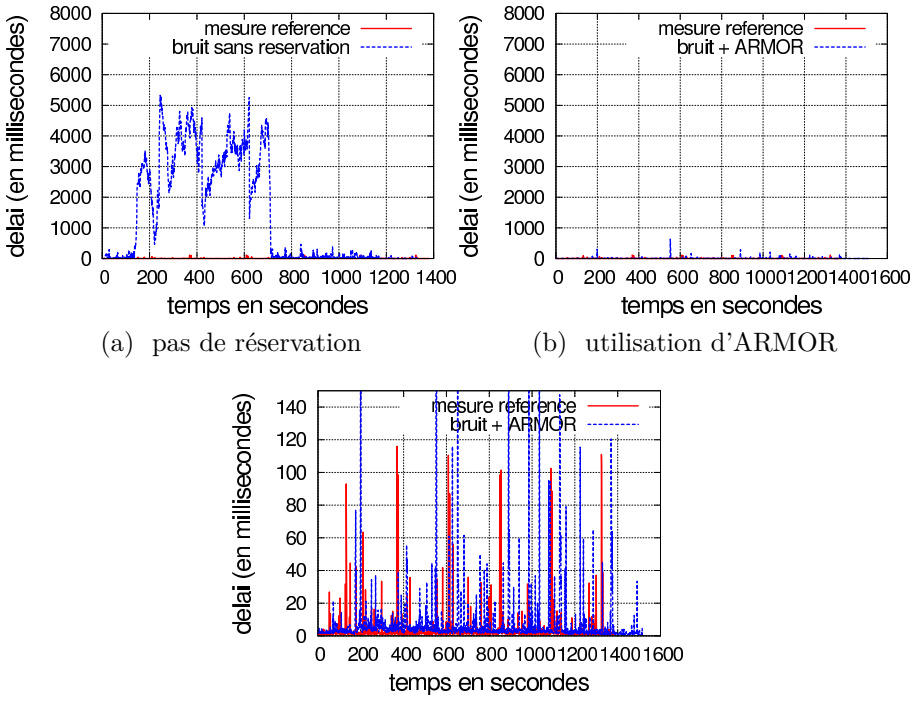
Les Figures 6.5b et 6.5c comparent respectivement la mesure de référence avec la mesure avec bruit sans ARMOR et avec la mesure avec bruit et ARMOR. Dans le cas où ARMOR n'est pas utilisé, on observe que l'application a moins de QdR réseau (la courbe bleue, en pointillé, est en dessous de la courbe de référence rouge et continue). En revanche, quand ARMOR fait une réservation pour l'application, les deux courbes sont presque confondues (Figure 6.5c).

Le Tableau 6.6 donne des statistiques sur les QdR utilisées par l'application de diffusion de contenus multimédia et le délai lors des trois exécutions. Les statistiques sont la valeur maximale mesurée, la moyenne et l'écart type des mesures. Pour la bande passante, la somme de toutes les données reçues sur l'équipement client est aussi fournie. Ces statistiques montrent que lorsqu'ARMOR est utilisé, les QdR réseau utilisées par l'application de diffusion de contenus multimédia sont comparables à la mesure de référence. En revanche, quand ARMOR n'est pas utilisé, l'application a environ deux fois moins de QdR réseau. En effet, la moyenne et la somme des données reçues est deux fois plus faible quand ARMOR n'est pas utilisé.

La Figure 6.6 compare le délai mesuré sur des paquets de `ping` avec la mesure de référence, lorsque du bruit est généré, avec et sans ARMOR. Sur la Figure 6.6a le délai est beaucoup plus important pour la mesure avec bruit et sans réservation (courbe bleue en pointillé). Le délai atteint plusieurs secondes par endroit. La Figure 6.6b donne le délai mesuré quand ARMOR est utilisé. Cette fois-ci des valeurs bien plus faibles sont mesurées. La Figure 6.6b utilise la même échelle que la Figure 6.6a, un zoom est donc réalisé pour comparer la mesure avec bruit et ARMOR à la courbe de référence. Ce zoom est donné sur la Figure 6.6c. On peut voir ici que les deux courbes sont comparables. La courbe bleue en pointillé, présente plus de pics mais la majorité reste en dessous de 100 millisecondes.

	valeur	référence	bruit sans ARMOR	bruit avec ARMOR
bande passante (kits/s)	max	1210	588	1120
	moyenne	257	116	236
	écart type	204	136	206
	somme	91600	43812	92100
délai (ms)	max	116	5320	632
	moyenne	3.82	1410	7.93
	écart type	11.1	1710	30.0

TABLE 6.6 – Statistiques d'utilisation des ressources réseau



(a) pas de réservation (b) utilisation d'ARMOR

(c) utilisation d'ARMOR (zoom)

FIGURE 6.6 – Mesures du délai

exécution	nombre d'erreurs	ratio $\frac{\text{nb erreurs}}{\text{nb images}}$
référence	65	0.005
bruit, sans ARMOR	823	0.05
bruit, avec ARMOR	92	0.013

TABLE 6.7 – Nombre d'erreurs levées par `mplayer`

Sur le Tableau 6.6, on observe que les valeurs de délai sont comparables à l'exécution de référence quand ARMOR est utilisé. À l'exception de la valeur maximale qui est supérieure, ce qui n'est pas représentatif et la moyenne reste comparable à la mesure de référence. Avec ARMOR un facteur deux est observé sur le délai mais ce dernier reste à une valeur bien inférieure à 100 ms en moyenne. Par contre, quand ARMOR n'est pas utilisé, un facteur d'environ 400 est observé sur le délai qui atteint plus d'une seconde en moyenne.

Observations de la qualité de service Du point de vue de la QdS utilisateur, il n'y pas de différence notable avec l'exécution de référence, sur la durée totale du film, quand ARMOR est utilisé. Quelques pixelisations supplémentaires sont observées mais cela reste raisonnable. En revanche quand ARMOR n'est pas utilisé, plus de pixelisations sont observées et la QdS utilisateur n'est pas très bonne.

Pour évaluer la QdS de la couche application, le Tableau 6.7 donne le nombre d'erreurs levées par le logiciel client lors des trois exécutions. La deuxième colonne donne le nombre d'erreurs, le flux utilisé dans cette expérience contenant 14316 images. La troisième colonne donne le ratio $\frac{nb\ erreurs}{nb\ images}$. On observe, avec ARMOR, un facteur de 1.5 sur le nombre d'erreurs alors que sans ARMOR un facteur de 12 est observé.

Discussion des observations

Quantités de ressources Cette expérience a montré que les réservations mises en place par ARMOR garantissent les QdR requises par l'application. En outre ces réservations garantissent aussi les contraintes de QdS réseau de délai (inférieur à 100 ms) et de gigue (inférieure à 200 ms). Enfin le taux perte (dernière contrainte de QdS réseau) est mesuré à l'aide des données reçues sur l'équipement. Comme aucune norme de QdS réseau n'est utilisée, le trafic par défaut et le trafic de l'application sont traités de la même manière. Quand `iperf` (bruit) et l'application cherchent à émettre en même temps, des collisions se produisent. Les paquets sont retransmis et jetés si des collisions continues de se produire. Le protocole UDP est utilisé, l'équipement client ne reçoit donc pas toutes les données envoyées par l'équipement serveur, ce qui augmente le taux de perte. Cette expérience a montré que les réservations d'ARMOR garantissent aussi cette contrainte de QdS réseau, en limitant le trafic par défaut des équipements.

Quand ARMOR n'est pas utilisé, l'application n'a pas assez de QdR réseau. De plus le délai atteint plusieurs secondes, ce qui n'est pas acceptable pour ce genre d'applications.

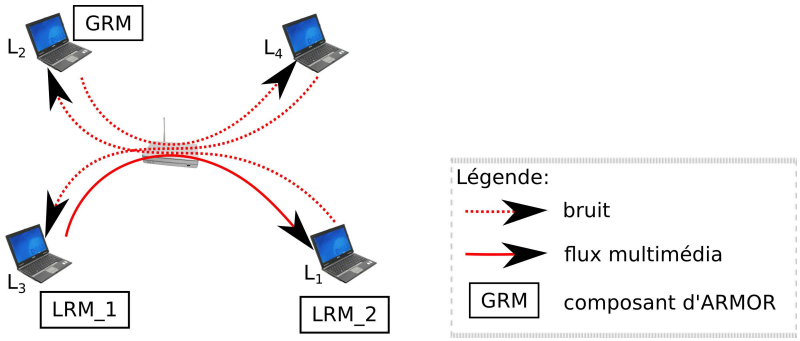


FIGURE 6.7 – Environnement expérimental, avec QdS réseau

Qualité de service Sur le délai, un facteur de 300 est observé entre en l'exécution avec bruit, sans ARMOR et l'exécution de référence. On pourrait donc s'attendre à un facteur similaire sur le nombre d'erreurs. Or, un facteur 12 est observé sur le nombre d'erreurs entre ces deux exécutions. La taille du flux envoyé est faible (286 kbits/s en moyenne). De plus, par défaut le logiciel client (`mplayer`) stocke 8192 kbits pour compenser la gigue du réseau. Or, le flux étant petit, le tampon de réception compense une gigue importante. De ce fait, la QdS de la couche application est moins bonne mais pas très mauvaise, lorsqu'ARMOR n'est pas utilisé. Cependant la QdS est meilleure quand ARMOR est utilisé.

6.2.1.2 Wifi avec QdS réseau

Dans ces expériences, les équipements utilisent la norme de QdS réseau Wifi 802.11e et la passerelle domestique est configurée pour utiliser cette norme. La Figure 6.7 donne l'environnement expérimental utilisé : les quatre ordinateurs portables (L_1, L_2, L_3, L_4) et la passerelle domestique (configurée avec la norme 802.11e). Comme les équipements utilisent la norme 802.11e, seuls les équipements exécutant une application de diffusion de contenus multimédia doivent avoir une instance de LRM installée et démarrée. L'instance du composant GRM est démarrée sur l'équipement L_2. Le trafic de l'application de diffusion de contenus multimédia est représenté, sur la figure, par la flèche continue, le bruit généré est représenté par les flèches en pointillé.

Dans cette évaluation nous présentons deux expériences. La première utilise le scénario 3. Pour ce scénario, les capacités des ressources réseau sont largement suffisantes pour émettre le flux. La deuxième expérience utilise le scénario 2. Cette fois-ci la QdR réseau requise par l'application est proche de la capacité de l'interface réseau de l'équipement serveur.

Exécution du scénario 3 Le Tableau 6.8 donne les QdR réseau requises de l'application de diffusion de contenus multimédia, les caractéristiques de QdS du lien et de l'interface réseau de l'équipement serveur (`capacité`, `QdRdisponible` et `QdRpartagée`) estimées par ARMOR, pour le scénario 3 (décrit dans le Tableau 6.4). Ces valeurs sont données une fois les réservations effectuées pour l'application.

Nous regardons maintenant les QdR utilisées par l'application pour les trois exécutions. Nous nous intéressons ensuite à la Qualité de Service perçue lors des trois exécutions. Enfin, une discussion explique les observations.

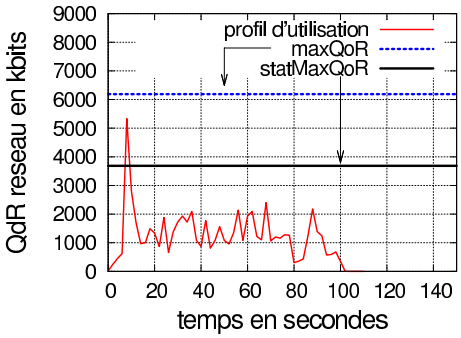
	estimation	valeur (en kbits/s)
QdR requise	`maxQoR`	6187
	`statMaxQoR`	3687
	`reqSharedQoR`	2501
	moyenne	1089
lien	`capacité`	54000
	`QdRdisponible`	47813
	`QdRpartagée`	2501
interface réseau serveur	`capacité`	25488
	`QdRdisponible`	19301
	`QdRpartagée`	2501

TABLE 6.8 – Estimations faites par ARMOR des QdR requises et disponibles sur les ressources réseau utilisées

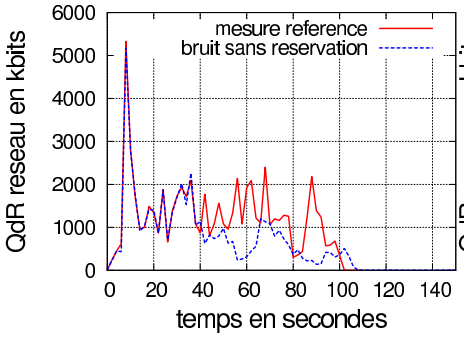
Observations des quantités de ressources La Figure 6.8a donne la courbe de référence mesurée, sans aucune application sur les équipements et la QdR requise estimée par ARMOR. Comme pour l'expérience détaillée à la section 6.2.1.1, la QdR requise estimée par ARMOR est supérieure au maximum du profil d'utilisation mesuré, du fait de la différence de granularité des profils. Les Figures 6.8b et 6.8c comparent respectivement la mesure de référence avec la mesure avec bruit sans ARMOR, et avec la mesure avec bruit et ARMOR. Sur la Figure 6.8b, les courbes sont confondues durant les 40 premières secondes. Ce temps correspond à l'initialisation du protocole d'évaluation. Le bruit sur le réseau ne commence réellement qu'à partir de 40 secondes. À partir de cet instant, la QdR utilisée par l'application est bien inférieure à la mesure de référence quand ARMOR n'est pas utilisé. Sur la Figure 6.8c, on peut noter que le pic de consommation du début de la diffusion est légèrement inférieur pour la mesure avec bruit et ARMOR. Cette observation s'explique par la différence de granularité. Par la suite, on observe que les deux courbes sont confondues, même quand le bruit est démarré.

Le Tableau 6.9 donne des statistiques sur les QdR utilisées par l'application de diffusion de contenus multimédia et le délai lors des trois exécutions. La bande passante moyenne lors de l'exécution avec du bruit et sans ARMOR est 3 fois plus faible que pour la courbe de référence. La somme des données reçues est inférieure d'environ 30%. Quand ARMOR est utilisé, la moyenne est plus faible que pour la courbe de référence (4110 au lieu de 5330). Cette différence s'explique car la mesure, avec ARMOR, n'a pas pris en compte le pic au début de la vidéo. Cependant la somme des données reçues est comparable avec ARMOR et la mesure de référence. De plus si on regarde les valeurs moyennes dans l'intervalle de temps [20-100] secondes, une différence de 10% est observée.

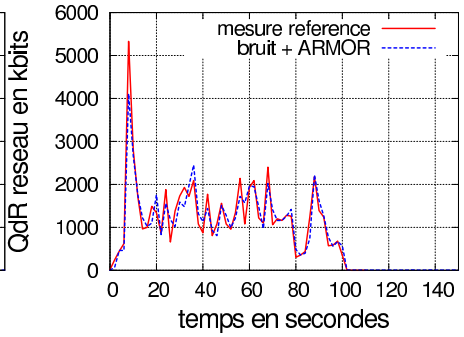
Pour refléter le délai des paquets de l'application de diffusion de contenus multimédia, les paquets de `ping` sont marqués pour utiliser la même classe de trafic que l'application de diffusion de contenus multimédia (sur l'équipement serveur et donc la même classe de trafic dans la norme 802.11e). La Figure 6.9 détaille les mesures de délai. Comme dans l'expérience précédente, quand ARMOR n'est pas utilisé le délai atteint plusieurs secondes. Quand ARMOR est utilisé, il est comparable à la mesure de référence.



(a) référence et QdR requise

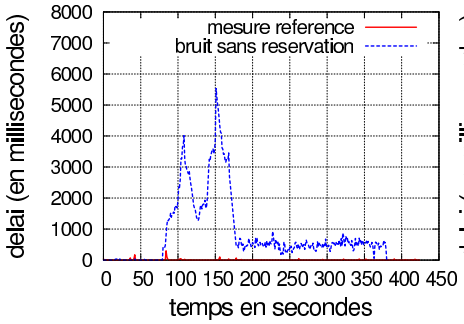


(b) pas de réservation

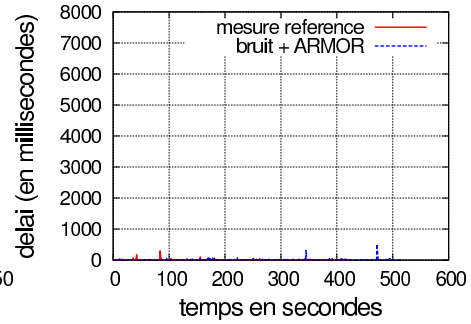


(c) utilisation d'ARMOR

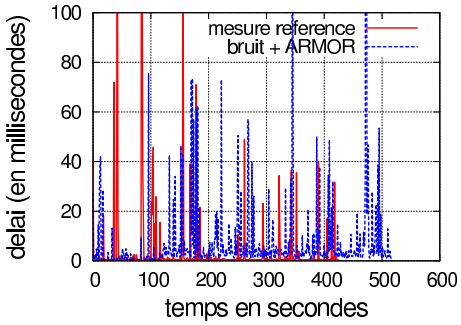
FIGURE 6.8 – Mesures des QdR réseau utilisées



(a) pas de réservation



(b) utilisation d'ARMOR



(c) utilisation d'ARMOR (zoom)

FIGURE 6.9 – Mesures du délai

	valeur	référence	bruit sans ARMOR	bruit avec ARMOR
bande passante (kits/s)	max	5330	5170	4110
	moyenne	1180	425	608
	écart type	881	728	833
	somme	66000	48000	68600
délai (ms)	max	298	5530	510
	moyenne	4.53	880	8.81
	écart type	20.2	1010	30.7

TABLE 6.9 – Statistiques d'utilisation des ressources réseau

exécution	nombre d'erreurs	ratio $\frac{\text{nb erreurs}}{\text{nb images}}$
référence	24	0.01
bruit, sans ARMOR	620	0.26
bruit, avec ARMOR	24	0.01

TABLE 6.10 – Nombre d'erreurs levées par `mplayer`

Comme dans l'expérience précédente, le Tableau 6.9 montre que le délai mesuré avec ARMOR est 2 fois supérieur à celui de la mesure de référence mais reste largement inférieur à 100ms. En revanche, quand ARMOR n'est pas utilisé, le délai atteint plusieurs secondes.

Observations de la qualité de service Du point de vue de la QdS utilisateur, il n'y a pas de différence notable entre l'exécution de référence et l'exécution avec bruit et ARMOR. Par contre la QdS utilisateur est mauvaise lorsqu'ARMOR n'est pas utilisé.

Le Tableau 6.10 donne le nombre d'erreurs levées par le logiciel client. Le flux utilisé dans ce scénario contient 2427 images. Il n'y a pas de différence entre l'exécution de référence et l'exécution avec du bruit et ARMOR. En revanche, lorsqu'ARMOR n'est pas utilisé, le nombre d'erreurs est 26 fois plus important.

Discussion des observations Comme précédemment, cette expérience a montré que les réservations mises en place par ARMOR garantissent les QdR requises et les contraintes de QdS réseau. Cette fois-ci, le nombre de paquets perdus du fait des collisions est limité par l'utilisation d'une norme de QdS réseau. Avec ARMOR, les paquets de l'application utilisent la norme de QdS 802.11e. Ils sont traités en priorité lors d'une collision. C'est donc uniquement le trafic par défaut qui est jeté.

Contrairement à l'expérience 6.2.1.1, le tampon de réception du logiciel client n'est pas suffisant pour compenser la gigue réseau. En effet, le flux diffusé ici a une bande passante moyenne de 1089 kibts/s alors que celui de l'expérience 6.2.1.1 avait une bande passante moyenne de 286 kbits/s. On observe cette fois-ci que la QdS de l'application est plus dégradée quand ARMOR n'est pas utilisé : 26 fois plus d'erreurs sont levées, contre 12 fois plus dans l'expérience précédente.

	estimation	**valeur (en kbits/s)**
QdR requise	`maxQoR`	10593
	`statMaxQoR`	8277
	`reqSharedQoR`	2316
	moyenne	2832
lien	`capacité`	54000
	`QdRdisponible`	43407
	`QdRpartagée`	2316
interface réseau serveur	`capacité`	12540
	`QdRdisponible`	1947
	`QdRpartagée`	2316

TABLE 6.11 – Estimations faites par ARMOR des QdR requises et disponibles sur les ressources réseau utilisées

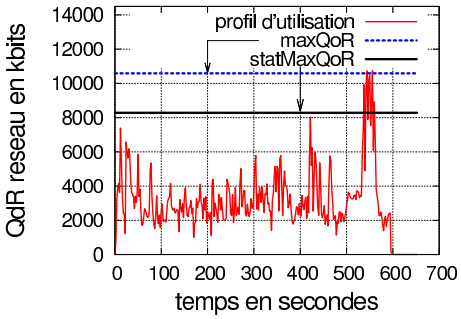
Exécution du scénario 2 Dans cette expérience, le scénario 2 décrit dans le Tableau 6.4 est exécuté. Le logiciel client est `vlc`. Le QdR réseau requise pour ce scénario est proche de la capacité de l'interface serveur. Quand ARMOR est utilisé, la demande de démarrage de l'application est parfois rejetée. Nous avons réalisé l'expérience avec ARMOR quatre fois, trois demandes ont été refusées par ARMOR. Le Tableau 6.11 donne les QdR requises de l'application de diffusion de contenus multimédia et les caractéristiques de QdS du lien et de l'interface réseau de l'équipement serveur (`capacité`, `QdRdisponible` et `QdRpartagée`). Ces valeurs sont données une fois la réservation effectuée pour l'application, dans le cas où l'application est acceptée par ARMOR.

Nous regardons maintenant les QdR utilisées par l'application pour les trois exécutions. Nous nous intéressons ensuite à la Qualité de Service perçue lors des trois exécutions. Enfin, une discussion explique les observations.

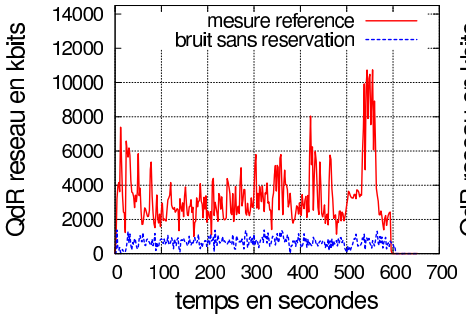
Observations des quantités de ressources La Figure 6.10a donne la courbe de référence mesurée, sans aucune application sur les équipements et la QdR requise estimée par ARMOR. Les Figures 6.10b et 6.10c comparent respectivement la mesure de référence avec la mesure avec bruit sans ARMOR et avec la mesure avec bruit et ARMOR. Dans le cas où ARMOR n'est pas utilisé, on observe que l'application a environ deux fois moins de QdR réseau (la courbe bleue, en pointillé, est en dessus de la courbe de référence en rouge, continue) et que les pics d'utilisation sont absents. En revanche, avec ARMOR les deux courbes sont presque confondues (Figure 6.10c). On peut noter que le pic de consommation est légèrement inférieur lors de l'exécution avec bruit et ARMOR.

Le Tableau 6.12 donne les statistiques d'utilisation des ressources réseau et du délai. On observe que la somme des données reçues avec ARMOR est comparable à l'exécution de référence. Une légère différence est observée sur les statistiques : la moyenne est inférieure de 5% et le maximum est inférieur de 10 %. Le bruit perturbe donc faiblement l'application. En revanche quand ARMOR n'est pas utilisé, la quantité d'information reçue par l'équipement client est cinq fois moindre au total et en moyenne sur chaque seconde.

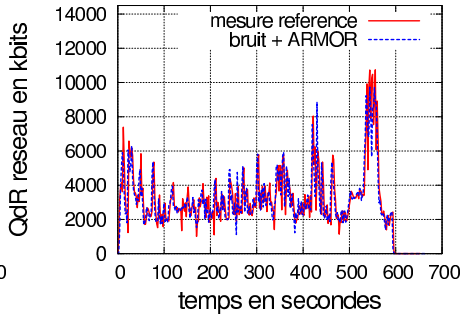
La Figure 6.11 montre le délai mesuré sur des paquets de `ping` envoyés dans les



(a) référence et QdR requise



(b) pas de réservation



(c) utilisation d'ARMOR

FIGURE 6.10 – Mesures des QdR réseau utilisées

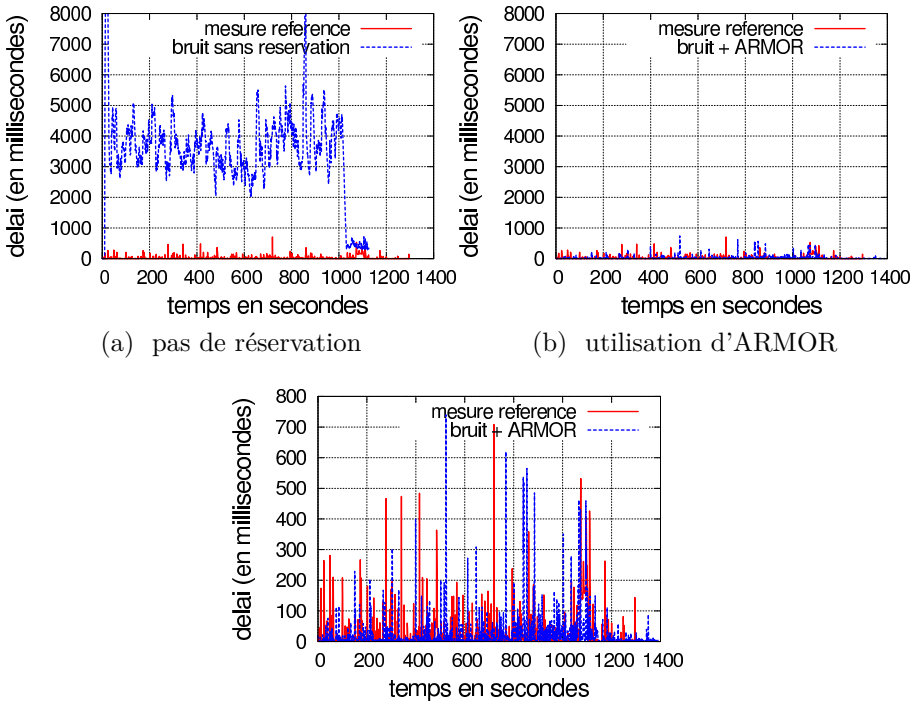
	valeur	**référence**	**bruit sans ARMOR**	**bruit avec ARMOR**
bande passante (kits/s)	max	10740	1380	9730
	moyenne	3040	621	2883
	écart type	1860	321	1830
	somme	995000	204000	980000
délai (ms)	max	708	9500	740
	moyenne	25	3500	23
	écart type	62	1390	60

TABLE 6.12 – Statistiques d'utilisation des ressources réseau

mêmes conditions que l'application de diffusion de contenus multimédia (en utilisant la même classe de trafic). Comme pour les expériences précédentes, le délai avec bruit et sans ARMOR est très important. Avec ARMOR il est comparable à celui de la mesure de référence.

Ces observations sont confirmées par le Tableau 6.12 donnant les statistiques de QdR réseau et du délai. Dans ce tableau, on voit qu'il n'y pas de différence entre le délai de la mesure de référence et celui de la mesure avec bruit et ARMOR. Alors que sans ARMOR, le délai observé atteint plusieurs secondes en moyenne.

Observations de la qualité de service Du point de vue de la QdS utilisateur, il n'y a pas de différence notable entre l'exécution de référence et l'exécution avec bruit et ARMOR. Dans les deux cas la QdS utilisateur n'est pas parfaite. Par contre la QdS utilisateur est très mauvaise lorsqu'ARMOR n'est pas utilisé.



(a) pas de réservation (b) utilisation d'ARMOR

(c) utilisation d'ARMOR (zoom)

FIGURE 6.11 – Mesures du délai

exécution	nombre d'erreurs	ratio $\frac{\text{nb erreurs}}{\text{nb images}}$
référence	223	0.02
bruit, sans ARMOR	11946	0.83
bruit, avec ARMOR	245	0.02

TABLE 6.13 – Nombre d'erreurs levées par `mplayer`

Le Tableau 6.13 donne le nombre d'erreurs levées par le logiciel client. Il n'y a pas de différence significative entre l'exécution de référence et l'exécution avec bruit et ARMOR. En revanche, lorsqu'ARMOR n'est pas utilisé, le nombre d'erreurs augment fortement. Le ratio est ici de 0.83, ce qui signifie que la QdS utilisateur est très fortement dégradée.

Discussion des observations Sur la Figure 6.10c, le pic d'utilisation à 550 secondes est atténué de 10% avec ARMOR comparé à la mesure de référence. Cette différence s'explique car la capacité de l'interface réseau est atteinte. De ce fait, le bruit généré sur le réseau perturbe légèrement l'application, même quand ARMOR est utilisé. Un léger retard est introduit et le pic est "absorbé" (10% de moins).

Contrairement aux expériences précédentes, le délai observé avec le bruit et ARMOR n'est pas supérieur à la mesure de référence. Cette différence s'explique car la QdR réseau requise est proche de la capacité de l'interface réseau de l'équipement serveur. Dans ce cas, même s'il n'y a pas de bruit sur le réseau, du retard est généré au niveau de l'interface serveur.

Du point de vue de la QdR, le ratio du nombre d'erreurs sur le nombre d'images, lors de l'exécution de référence ou lors de l'exécution avec bruit et ARMOR (0.02), est supérieur à celui de l'expérience 6.2.1.1 (0.005). Même lorsqu'aucun bruit n'est généré la QdS n'est ainsi pas parfaite.

Ces observations confirment que la capacité des ressources réseau est à peine suffisante. C'est pourquoi ARMOR refuse à juste titre 3 fois sur 4 le démarrage de l'application.

6.2.1.3 Discussion

Nous avons montré dans cette section qu'ARMOR garantit les QdR réseau requises par l'application, et ce, même quand du bruit est généré sur les liens. Dans nos expériences, quand ARMOR est utilisé la moyenne par seconde et la somme des QdR réseau reçues sur l'équipement client est comparable avec la mesure de référence. Quand ARMOR n'est pas utilisé, l'application n'a pas la QdR réseau requise.

La différence de QdR utilisée est due aux collisions perturbant le trafic. Nous avons vu qu'ARMOR limite ces collisions. Quand aucune norme de QdS réseau n'est supportée, le nombre de collisions est diminué en limitant le trafic de tous les équipements. Quand une norme de QdS réseau est supportée, le nombre de collisions des paquets multimédia est diminué. En outre, avec une norme de QdS réseau, ARMOR n'a pas besoin de démarrer une instance de composant LRM sur chaque équipement.

En plus de la QdR réseau requise, les applications de diffusion de contenus multimédia sont sensibles aux caractéristiques de QdS réseau (délai, gigue et taux de perte). Nous avons mesuré le délai et par conséquent la gigue, la variation du délai, à l'aide de `ping` envoyés dans les même conditions que l'application. Le taux de perte est estimé en comparant la somme des données reçues sur l'interface de l'équipement client. Nos expériences ont montré que les réservations d'ARMOR maintiennent ces valeurs à des seuils raisonnables, au vu des contraintes classiquement exprimées (100 ms pour le délai, 200 ms pour la gigue), et proches des valeurs de référence. La perte de paquets, sur un lien, peut subvenir lorsqu'un phénomène de collision apparaît ou si le lien est de mauvaise qualité. Les réservations d'ARMOR répondent au problème des collisions en les supprimant pour le trafic multimédia. Le protocole TCP supprime la perte en réémettant les paquets perdus. Avec ce protocole, les paquets seraient bien reçus mais avec un retard trop important. Cette solution n'est donc pas envisageable. La perte liée à la qualité du lien est traitée uniquement lors du contrôle d'admission où ARMOR mesure le taux de perte du lien.

Nous avons aussi mesuré dans nos expériences la QdS utilisateur et la QdS de la couche application lors de chaque expérience. Ces mesures ont montré que la QdS des applications de diffusion de contenus multimédia est garantie grâce aux réservations d'ARMOR.

Dans nos expériences, du bruit est généré sur le lien Wifi. Ce bruit simule du trafic entre d'autres équipements du réseau local domestique. Nous avons aussi réalisés des expériences où du bruit est généré uniquement sur l'interface serveur. Ces expériences ont montré que la réservation d'ARMOR garantit la QdR requise par les applications, sur l'interface réseau de l'équipement serveur. Ces expériences montrent que la politique de gestion du trafic émis sur l'interface serveur traite le trafic multimédia en priorité. Ce résultat étant déjà utilisé couramment pour gérer les

ressources réseau dans Linux, nous n'avons pas détaillé ces expériences. De plus, il apparaît plus probable que du bruit, sur les ressources réseau, provienne d'autres équipements que du trafic émis depuis l'interface réseau de l'équipement serveur.

Les capacités réelles des interfaces Wifi sont différentes des capacités théoriques. ARMOR estime la capacité réelle des interfaces Wifi, à l'aide de mesures actives, lors de la phase d'initialisation. ARMOR a donc une meilleure idée de la capacité réelle que s'il se basait sur la capacité théorique. Lors de la dernière expérience présentée, la QdR requise par l'application est proche de la capacité de l'interface de l'équipement serveur. Selon les tests réalisés, ARMOR refuse ou non l'application, du fait de la capacité de l'interface de l'équipement serveur. Cependant, s'il ne la refuse pas, la QdR réseau n'est pas totalement garantie. Les erreurs observées dans cette expérience proviennent de la capacité insuffisante de l'interface réseau de l'équipement serveur et non du bruit généré. Cette expérience met en évidence la limite d'utiliser une estimation par mesure active de la capacité des interfaces. Notamment dans le cas où la QdR requise est proche de la capacité des ressources réseau.

6.2.2 Ressources CPU

Pour garantir les QdR CPU d'une application de diffusion de contenus multimédia, ARMOR met en place une réservation sur chaque équipement utilisé. Les QdR requises sur les ressources CPU sont contenues dans un *manifest* et représentent la QdR maximum requise par l'application, sur chaque équipement. Le Tableau 6.14, donne les QdR requises sur les équipements serveur (PC_1) et client (*beagleboard* BB_1), lors de l'exécution du scénario 1. L'équipement client BB_1 a deux configurations (cf. Tableau 6.3, page 128). Dans la configuration *c1*, l'ordonnanceur CFS et un noyau Linux standard sont utilisés. Dans la configuration *c2*, l'ordonnanceur CBS est utilisé grâce à un noyau Linux modifié avec le *patch* du projet AQuosA.

Pour les ressources CPU, le bruit est généré sur la ressource CPU de l'équipement client à l'aide de l'outil `stress`[14], appelé avec l'option `-c 10` pour lancer 10 *threads*. Chaque *thread* utilise autant de CPU que possible, en exécutant le code donné dans le Listing 6.1, correspondant à un calcul de racine carrée. Pour mesurer la QdR CPU utilisée, l'outil `top` est exécuté avec la plus haute priorité pour ne pas être perturbé par le bruit. Pour ne pas perturber les applications de diffusion de contenus multimédia, `top` mesure la QdR CPU utilisée avec une période de 400 millisecondes.

```
int hogcpu (void)
{
  while (1){
    sqrt (rand ());
  }
  return 0;
}
```

Listing 6.1 – `stress` sur le CPU

Nous détaillons d'abord l'expérience exécutant uniquement le scénario 1. Celle-ci est faite pour les deux configurations de l'équipement client.

14. `http://weather.ou.edu/~apw/projects/stress/`

	serveur	client
capacité	200	100
QdR requise (en %,ms)	(20,40)	(45,40)

TABLE 6.14 – Capacités CPU et QdR requises CPU

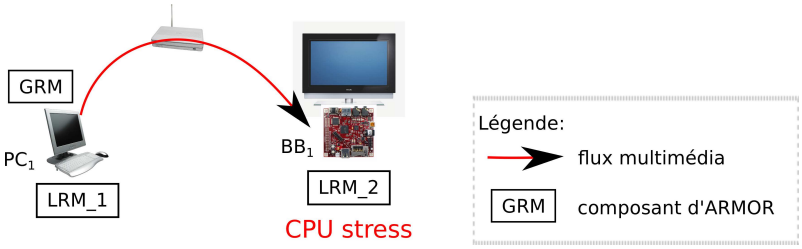


FIGURE 6.12 – Environnement expérimental

Pour affecter le CPU aux processus, l'ordonnanceur CFS utilise des poids, relatifs les uns par rapport aux autres. Pour cet ordonnanceur, ARMOR crée un *cgroup* dédié à l'application et lui affecte un poids égal à la QdR requise par l'application. Les autres processus sont regroupés en deux *cgroups* ayant un poids égal à $\frac{1}{100} * capacité$. Si aucune autre réservation n'est faite, lors de l'exécution du scénario 1, le logiciel client a donc 45 fois plus de CPU que les autres *cgroups*, c'est-à-dire une réservation de 98% . La réservation est donc bien supérieure à la QdR requise. Pour évaluer les réservations faites par ARMOR, pour l'ordonnanceur CFS, une deuxième évaluation est réalisée. Cette évaluation consiste à exécuter le scénario 1 et une application multimédia locale. Cette application décode le même flux et a aussi une QdR requise de 45%. Dans cette évaluation la somme des réservations (92%) est donc plus proche de la capacité de la ressource CPU (100%).

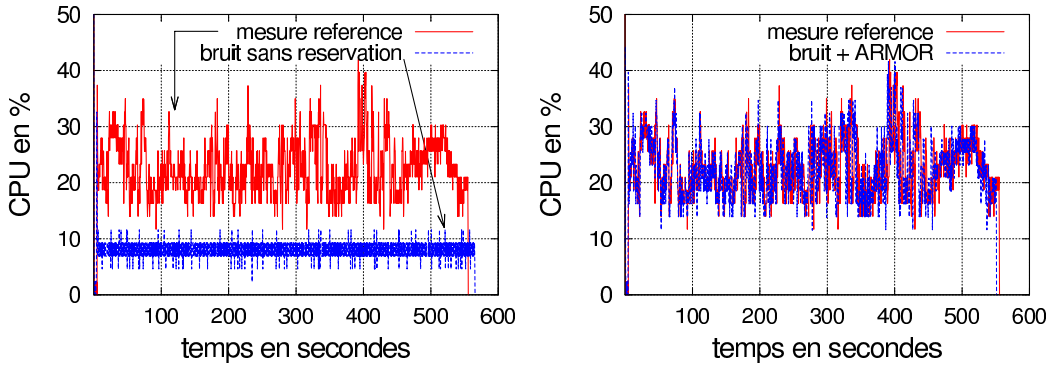
Pour affecter le CPU aux processus, l'ordonnanceur CBS utilise les échéances des *threads* qui ne sont pas relatives les unes aux autres. Cette deuxième évaluation n'est donc pas détaillée pour cet ordonnanceur. Nous commençons par détailler l'évaluation portant uniquement sur le scénario 1.

6.2.2.1 Évaluation sur le scénario 1

Le logiciel client utilisé est ici `mplayer`. L'expérience est réalisée pour la configuration *c1* de la *beagleboard*, avec un ordonnanceur CFS et pour la configuration *c2*, avec un ordonnanceur CBS. La Figure 6.12 donne l'environnement expérimental utilisé, avec un LRM installé sur chaque équipement utilisé. Le GRM est installé sur l'équipement PC_1.

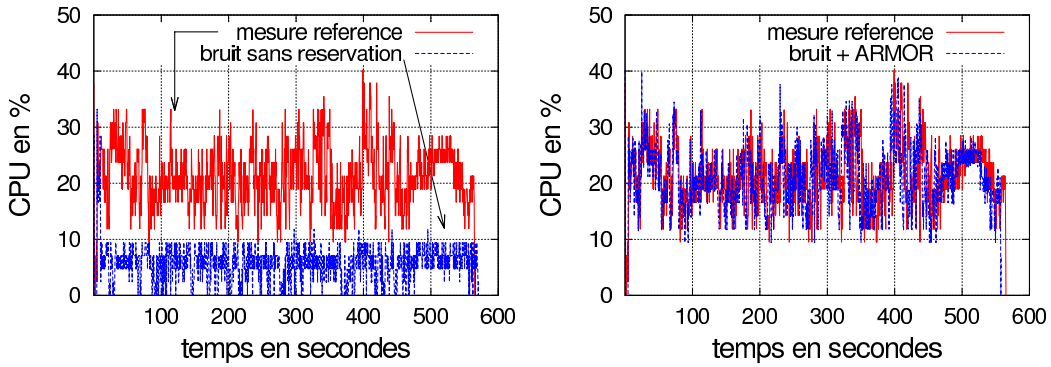
Nous observons d'abord les QdR utilisées par `mplayer` pour les différentes exécutions, puis nous observons la QdS de l'application. Enfin une discussion est donnée.

Observations des quantités de ressources La Figure 6.13 compare à l'exécution de référence, les QdR CPU utilisées par `mplayer`, lors des exécutions avec bruit, pour la configuration *c1* (ordonnanceur CFS). La Figure 6.13a compare les QdR CPU utilisées lors de l'exécution de référence (courbe rouge, continue) avec les QdR CPU utilisées lors de l'exécution avec bruit et sans ARMOR (courbe bleue, en pointillé). `mplayer` a ici moins de CPU que lors de l'exécution référence. En revanche,



(a) pas de réservation (b) utilisation d'ARMOR

FIGURE 6.13 – QdR CPU utilisées sur l'équipement client, pour la configuration *c1*



(a) pas de réservation (b) utilisation d'ARMOR

FIGURE 6.14 – QdR CPU utilisées sur l'équipement client, pour la configuration *c2*

lors de l'exécution avec bruit et ARMOR, comparée à l'exécution de référence sur la Figure 6.13b, les QdR CPU utilisées par `mplayer` sont les mêmes que pour l'exécution de référence.

La Figure 6.14 compare à l'exécution de référence, les QdR CPU utilisées par `mplayer`, lors des exécutions avec bruit, pour la configuration *c2* (ordonnanceur CBS). De même que pour la configuration *c1*, lorsqu'ARMOR n'est pas utilisé (Figure 6.14a), le bruit perturbe l'exécution de `mplayer`, qui n'a pas assez de QdR CPU. Quand une réservation est faite par ARMOR, `mplayer` a assez de CPU et n'est pas gêné par le bruit (Figure 6.14b).

Observations de la qualité de service Dans les deux configurations (*c1* et *c2*), la QdS utilisateur est mauvaise lors de l'exécution avec bruit et sans ARMOR. Quand ARMOR est utilisé, la QdS utilisateur est bonne.

Le Tableau 6.15 donne le nombre d'erreurs levées par `mplayer` pour chaque exécution ainsi que le ratio du nombre d'erreurs sur le nombre d'images du flux ($\frac{nb\ erreurs}{nb\ images}$). Le flux utilisé dans cette expérience contient 14316 images. Lorsqu'ARMOR n'est pas utilisé, le bruit perturbe fortement l'application de diffusion de contenus multimédia : le ratio $\frac{nb\ erreurs}{nb\ images}$ est de 0.43 c'est-à-dire 85 fois plus que lors de l'exécution de référence (ratio de 0.005). Avec ARMOR le ratio est de 0.006, ce qui est très proche de celui de l'exécution de référence à 0.005.

configuration	exécution	nombre d'erreurs	ratio $\frac{\text{nb erreurs}}{\text{nb images}}$
c1	référence	77	0.005
	bruit, sans ARMOR	6187	0.433
	bruit, avec ARMOR	90	0.006
c2	référence	83	0.006
	bruit, sans ARMOR	6204	0.434
	bruit, avec ARMOR	81	0.006

TABLE 6.15 – Nombre d'erreurs levées par `mplayer`

Discussion des observations Dans les deux configurations, quand ARMOR n'est pas utilisé, l'ordonnanceur partage équitablement le CPU entre tous les processus. De ce fait `mplayer` a environ 10% du CPU durant toute la durée de l'application. Les réservations d'ARMOR modifient cette politique, ce qui garantit les QdR requises par l'application.

Quand `mplayer` n'a pas assez de CPU, il n'a pas le temps de décoder toutes les images envoyées par le logiciel serveur, certaines sont écrasées et le flux est fortement pixelisé. Au contraire, lorsqu'ARMOR est utilisé, `mplayer` a assez de CPU et il n'y pas de différence du point de vue de la QdS utilisateur.

Dans les exécutions de référence et avec ARMOR, quelques erreurs sont levées pas `mplayer`. Ces erreurs viennent pour la plupart du temps de synchronisation au début de la diffusion, rendu nécessaire pas l'utilisation du mode *push*. On observe qu'avec l'ordonnanceur CFS, `mplayer` lève quelques erreurs de plus que l'exécution de référence ou avec l'ordonnanceur CBS. Cette légère différence montre que l'ordonnanceur CFS ne garantit pas complètement la QdR CPU. Cependant la différence du nombre d'erreurs est faible et jugée acceptable.

6.2.2.2 Évaluation sur le scénario 1 et une application locale

Dans cette expérience le scénario 1 est exécuté. La configuration *c1* de l'équipement client BB_1 est utilisé. La Figure 6.15 donne l'environnement expérimental. Un LRM est démarré sur l'équipement serveur et l'équipement client. Sur l'équipement client, du bruit est généré à l'aide de l'outil `stress`. Une application locale utilisant `mplayer` est aussi démarrée sur l'équipement client.

ARMOR met en place une réservation sur l'équipement client pour l'application de diffusion. Une réservation est faite manuellement pour l'application locale. Les deux réservations sont réalisées à l'aide d'un *cgroup* dédié à chaque application (diffusion et locale), ayant un poids de 45%. Tous les autres processus sont déplacés dans les *cgroups user_group* et *system_group* lors de la phase d'initialisation d'ARMOR, chacun de ces *cgroups* ayant un poids de 1%.

Observations des quantités de ressources Pour cette expérience, l'exécution de référence est mesurée lorsque l'application de diffusion et l'application locale sont seules à être exécutées sur les équipements. Ensuite, trois exécutions sont faites et sont comparées à l'exécution de référence :

– bruit, aucune réservation

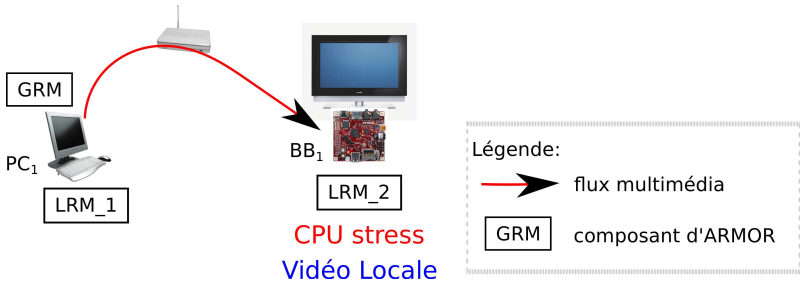


FIGURE 6.15 – Environnement expérimental

– bruit, réservation à l'aide d'ARMOR pour l'application de diffusion, pas de réservation pour l'application locale
– bruit, réservation à l'aide d'ARMOR pour l'application de diffusion, réservation manuelle pour l'application locale.

La Figure 6.16 compare les QdR utilisées, par les deux applications, lors de ces trois exécutions. Sur les Figures 6.16a et 6.16b, on observe que les deux applications ont moins de QdR CPU que lors des mesures de référence.

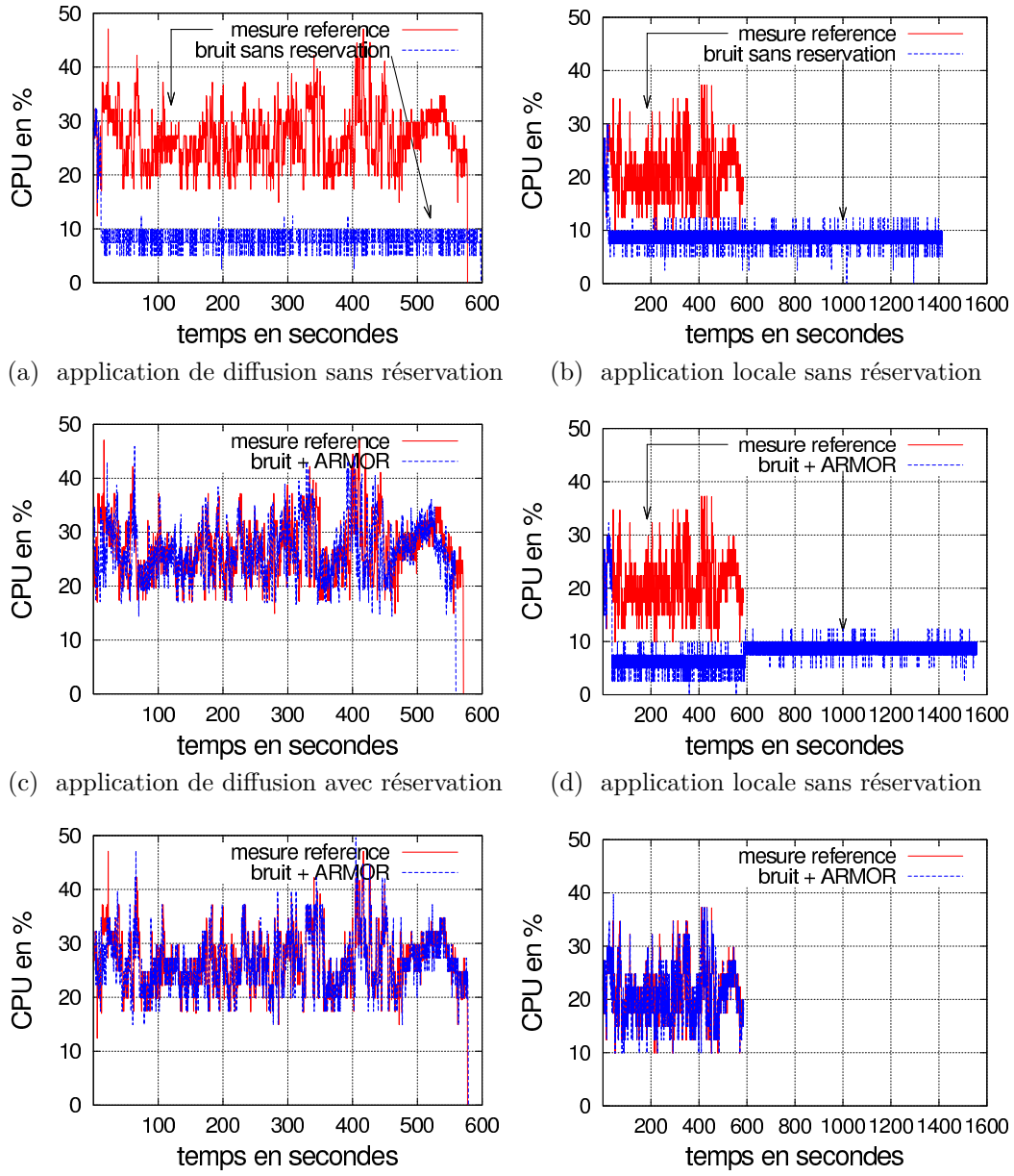
Les Figures 6.16c et 6.16d donnent les QdR CPU mesurées pour la deuxième exécution. Cette fois-ci, l'application de diffusion a les QdR CPU requises, grâce à la réservation mise en place par ARMOR. Les courbes de la Figure 6.16c sont confondues. En revanche, l'application locale n'a pas assez de CPU.

Les Figures 6.16e et 6.16f donnent les QdR CPU mesurées pour la troisième exécution. Cette fois-ci les deux applications ont les QdR CPU requises, grâce aux réservations. Sur les deux figures les courbes sont confondues.

Observation de la qualité de service La QdS utilisateur est mauvaise quand il n'y a pas de réservation. Comme dans les expériences précédentes, l'application de diffusion de contenus multimédia est fortement pixelisée lorsque du bruit est généré. Quand `mplayer` décode une vidéo localement, s'il n'a pas assez de CPU, le décodage est saccadé et dure plus longtemps. La QdS utilisateur est mauvaise, même s'il n'y a pas de pixelisation.

Le Tableau 6.16 donne les erreurs levées par le logiciel client de l'application de diffusion de contenus multimédia pour les quatre exécutions. La troisième ligne donne les statistiques pour l'exécution avec une réservation pour l'application de diffusion de contenus multimédia et pas de réservation pour l'application locale. La quatrième ligne donne les statistiques pour l'exécution avec une réservation pour l'application de diffusion de contenus multimédia et une réservation pour l'application locale. Comme pour l'expérience précédente, ces observations montrent que lorsqu'ARMOR est utilisé, la QdS de l'application de diffusion est garantie.

Pour l'application locale, aucune erreur n'est levée par `mplayer`, la QdS est dégradée car les images ne sont pas affichées à temps. Le Tableau 6.17 donne le temps pendant lequel l'application locale affiche la vidéo, pour les différentes exécutions. Le temps exprimé en seconde, est obtenu en comptant le nombre de mesures effectuées, une mesure étant effectuée toutes les 40 millisecondes. On observe que lorsqu'une réservation est faite pour les deux applications, le temps d'affichage est le même que celui de référence à 0.1% près. Cette légère différence est expliquée par la précision de `top` qui est plus faible quand le CPU de l'équipement est surchargé. En revanche



(a) application de diffusion sans réservation (b) application locale sans réservation

(c) application de diffusion avec réservation (d) application locale sans réservation

(e) application de diffusion avec réservation (f) application locale avec réservation

FIGURE 6.16 – QdR CPU utilisées sur l'équipement client, pour la configuration c1

configuration	**exécution**	**nombre d'erreurs**	**ratio** $\frac{\text{nb erreurs}}{\text{nb images}}$
c1	référence	66	0.005
	bruit, sans ARMOR	6691	0.47
	bruit, avec ARMOR, une réservation	67	0.005
	bruit, avec ARMOR, deux réservations	76	0.005

TABLE 6.16 – Nombre d'erreurs levées par `mplayer`

exécution	temps (en s)
référence	586
bruit, sans ARMOR	1403
bruit, avec ARMOR, une réservation	1478
bruit, avec ARMOR, deux réservations	587

TABLE 6.17 – Temps d'affichage de l'application locale

quand aucune réservation n'est faite pour l'application locale, le temps d'affichage est 2,5 fois plus long.

Discussion des observations Comme dans l'expérience précédente, quand ARMOR n'est pas utilisé, l'ordonnanceur affecte le CPU équitablement à tous processus. Pour les applications multimédia, quand du bruit est généré, cela se traduit par une dégradation de la qualité de service. Pour une application de diffusion, les images sont pixelisées. Pour une application locale, `mplayer` ne se rend pas compte qu'il est en retard. Les images sont saccadées et le temps d'affichage de la vidéo est 2.5 fois plus important.

Quand une réservation est faite, par ARMOR pour l'application de diffusion, ou manuellement pour l'application locale, les applications ont assez de CPU pour s'exécuter correctement et la QdS est satisfaite.

6.2.2.3 Discussion

Les expériences détaillées dans cette section ont montré que les réservations, mises en place par ARMOR garantissent les QdR CPU requises par les applications, et ce, même quand du bruit est généré. Quand aucune réservation n'est effectuée, le bruit généré sur l'équipement client perturbe fortement l'application de diffusion de contenus multimédia qui n'a pas la QdR CPU requise.

Les expériences présentées dans cette section ont aussi évalué la QdS des couches application et utilisateur. Ces expériences ont montré que les réservations faites par ARMOR garantissent la QdS dans ces deux couches.

La première expérience a été réalisée pour les deux ordonnanceurs supportés par ARMOR. Dans les deux cas, les QdR CPU et la QdS sont garanties par les réservations.

Enfin, une deuxième expérience a montré qu'ARMOR peut faire plusieurs réservations, notamment avec l'ordonnanceur CFS, même si la somme des réservations est proche de la capacité de la ressource CPU.

6.2.3 Ressources mémoire RAM

Pour garantir les QdR mémoire RAM d'une application de diffusion de contenus multimédia, ARMOR met en place une réservation sur chaque équipement utilisé. Ces réservations sont faites à l'aide des *cgroups* et de limites sur l'utilisation mémoire des *cgroups*.

La Figure 6.17 donne l'environnement expérimental utilisé. Comme pour les ressources CPU, une instance de composant LRM est installée sur chaque

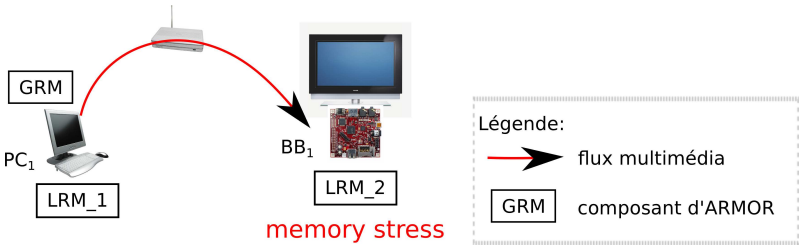


FIGURE 6.17 – Environnement expérimental

équipement et l'instance du composant GRM est installée sur l'équipement PC_1. Du bruit est généré sur l'équipement client à l'aide de l'outil `stress`. Cet outil est ici appelé avec l'option `-vm` pour utiliser de la mémoire. L'outil `stress` effectue des allocations mémoire et les utilise. Le code de stress exécuté sur la mémoire est donné en annexe C.

Pour tester les réservations mémoire RAM, un `stress` est lancé en comme le montre le Listing 6.2. Le `stress` essaie ici d'allouer 200Mo de mémoire RAM.

```
>stress --vm 1 --vm-bytes 200M
```

Listing 6.2 – `stress` avec un *thread*

L'équipement BB_1 a une capacité mémoire RAM de 256Mo et n'a pas de mémoire tampon. Pour exécuter le scénario 1, la QdR requise sur l'équipement client est de 10Mo. Dans ce cas, ARMOR limite la mémoire RAM du *cgroup user_group* à 243,4Mo : capacité (256Mo) moins la réservation pour le *system_group* ($\frac{1}{100} * capacité = 2,6Mo$) moins la réservation pour l'application (10Mo) ($256 - 2,6 - 10 = 243,6$). Les processus de l'équipement consomment 50Mo de mémoire RAM au moment où le `stress` est lancé. Il n'y a donc pas assez de mémoire pour tous les processus ($243,6 - 50 = 193,6 < 200$).

Quand ARMOR est utilisé, le noyau refuse l'allocation mémoire car elle est supérieure à la limite du *cgroup user_group*. Dans ce cas, le programme `stress` s'arrête. Si ARMOR n'est pas utilisé, l'allocation est autorisée et le noyau Linux termine ensuite un processus. Selon les tests réalisés, le processus terminé est le `stress`, le logiciel client ou un autre processus.

Un deuxième test est réalisé avec le code du Listing 6.3. Cette fois-ci, dix *threads* sont créés, utilisant chacun 20Mo.

```
>stress --vm 10 --vm-bytes 20Mo
```

Listing 6.3 – `stress` avec dix *threads*

Comme dans l'expérience précédente, lorsqu'ARMOR est utilisé le processus `stress` est terminé. Si ARMOR n'est pas utilisé, il peut s'agir d'un autre processus.

Nous avons réalisé les mêmes expériences en démarrant le logiciel client sur l'équipement L_1. Cet équipement a une capacité mémoire RAM de 2000Mo et une mémoire tampon de 500Mo. Quand ARMOR n'est pas utilisé, si le `stress` est lancé avec `-vm-bytes 2000` pour essayer d'utiliser toute la mémoire RAM, l'équipement n'est pas utilisable et doit être redémarré. Quand ARMOR est utilisé, si le `stress` est lancé dans le *cgroup user_group* le même résultat est observé. En effet ce *cgroup* contient les programmes utilisateurs qui sont rendus inutilisables par le `stress`. En

revanche, si un *cgroup* dédié au `stress` est créé alors l'équipement est toujours utilisable et l'application de diffusion de contenus multimédia s'exécute normalement.

Cette expérience montre la capacité des *cgroups* à gérer efficacement la mémoire RAM par *cgroup*. Ainsi, ARMOR peut s'appuyer sur ces mécanismes pour gérer les QdR mémoire RAM des applications multimédia.

6.3 Conclusion

Dans ce chapitre, nous avons montré qu'ARMOR peut être mis en œuvre dans un réseau local domestique. Les composants d'ARMOR utilisent uniquement des mécanismes standards dans les équipements Linux. Pour utiliser ARMOR sur un équipement du réseau local domestique, il suffit d'installer sur cet équipement les outils utilisés par ARMOR. Nous avons ainsi réalisé une implémentation réaliste d'ARMOR, respectant les contraintes du domaine d'étude. Nous avons aussi évalué les coûts liés à l'utilisation d'ARMOR, qui s'avèrent raisonnables. En effet, le temps des opérations ARMOR est acceptable dans un réseau local domestique. Enfin, ARMOR est capable de supporter des mécanismes de réservation hétérogènes : différents types de ressources réseau, support ou non d'une norme de QdS réseau, et différents ordonnanceurs.

Dans une deuxième partie de ce chapitre, nous avons montré que les réservations mises en place par ARMOR garantissent les QdR requises par les applications de diffusion de contenus multimédia. Ces QdR sont garanties même quand du bruit est généré sur les différentes ressources. Avec le même bruit, si ARMOR n'est pas utilisé, les applications de diffusion de contenus multimédia n'ont pas assez de QdR pour fournir la QdS attendue. Avec ARMOR, les applications ont suffisamment de ressources et la QdS des couches application et utilisateur est satisfaite. Nous avons aussi réalisé des expériences en générant simultanément du bruit sur les ressources réseau, CPU et mémoire RAM lors de l'exécution d'une même application de diffusion de contenus multimédia. Les mêmes résultats ont été obtenus : ARMOR garantit les QdR requises et la QdS est satisfaite.

Chapitre 7

Conclusion

Rappel de la problématique

L'objectif de cette thèse était de garantir la qualité de service des applications de diffusion de contenus multimédia dans le réseau local domestique en fournissant une gestion des ressources appropriée.

La garantie de la qualité de service d'une application repose généralement sur la réservation des ressources nécessaires à l'aide de mécanismes de réservation. Notre étude a montré que les équipements du réseau local domestique disposent effectivement de mécanismes de réservation pour chaque ressource (CPU, mémoire et réseau). Une application utilisant plusieurs ressources, nous nous sommes intéressés aux solutions gérant l'ensemble des ressources utilisées par une application.

Les solutions actuelles supposent l'existence de mécanismes ou d'intergiciels spécifiques. Pour intégrer ces mécanismes, les équipements et les applications doivent être modifiés. Or, le réseau local domestique est un environnement ouvert, hétérogène et distribué, dominé par plusieurs acteurs où il est primordial d'être le moins intrusif possible. Les solutions existantes ne se sont donc pas adaptées à ce domaine. En outre, elles supposent que les quantités de ressources requises par une application sont connues, ce qui n'est pas évident du fait de l'hétérogénéité inhérente au réseau local domestique.

Pour pallier ces limites, nous avons proposé trois contributions. La première contribution a porté sur l'estimation et le stockage des quantités de ressources requises par les applications de diffusion de contenus multimédia. La deuxième contribution est un *framework* de gestion des ressources, utilisable dans le réseau local domestique. La troisième contribution est la mise en œuvre de ce *framework* sur des équipements réels, typiquement utilisés dans un réseau local domestique.

Contributions

Estimation et stockage des quantités de ressources Les quantités de ressources requises par une application multimédia représentent des bornes supérieures sur le profil d'utilisation de chaque ressource. Nous avons utilisé deux méthodes d'estimation des profils d'utilisation. Pour les ressources réseau, les tailles des images du flux vidéo sont analysées avant le démarrage de l'application. Pour les ressources CPU et mémoire RAM, l'application est exécutée et les profils d'u-

tilisation sont mesurés sur chaque ressource. Les quantités de ressources requises sont ensuite stockées pour être réutilisées lors du démarrage de l'application avec les mêmes paramètres.

Au vu du nombre de paramètres possibles pour une application (flux, contenu, paramètres d'encodage, ressources et logiciels utilisés), les quantités de ressources CPU et mémoire RAM requises sont agrégées pour limiter la quantité d'information à stocker. En outre, quand une relation d'ordre existe sur le paramètre à agréger (ex. paramètre taille d'images), l'agrégation diminue aussi le nombre de mesures à réaliser. Dans ce cas, il est possible d'utiliser une valeur de quantité de ressources requise supérieure, à la place de la quantité correspondant exactement au paramètre spécifié.

Pour automatiser l'agrégation, nous avons développé deux algorithmes basés respectivement sur le *clustering* et le *bin-packing*. Dans notre évaluation, nous avons montré que ces algorithmes sont capables de traiter les quantités de ressources requises en un temps acceptable (3 minutes pour un million de valeurs). Les bornes, passées en paramètres aux algorithmes, orientent le compromis entre l'objectif d'agrégation et la perte engendrée (due à la sur-réservation). Nous avons montré que le choix de ces bornes est fonction de l'utilisation future des ressources. Enfin, nos évaluations indiquent que les algorithmes réduisent efficacement la quantité d'information, si les bornes sont bien choisies. Dans nos simulations, cette quantité est divisée par 10, pour une perte moyenne inférieure à 10% de la capacité des ressources.

***Framework* de gestion des ressources** La deuxième contribution de cette thèse est un *framework* de gestion des ressources appelé ARMOR, fournissant un cadre d'exécution garantissant les quantités de ressources requises par les applications de diffusion de contenus multimédia.

ARMOR utilise une architecture de gestion des ressources basée sur des composants managers globaux et locaux. Pour masquer l'hétérogénéité des équipements, ces composants sont configurables. Une fois configurés, lors d'une phase d'initialisation, les composants d'ARMOR offrent un service générique de gestion des ressources. Dans sa version actuelle, ARMOR supporte les ressources réseau de type Wifi ou Ethernet, avec ou sans support de QdS réseau. Pour les ressources CPU, ARMOR supporte actuellement deux types d'ordonnanceurs.

Afin de garantir la qualité de service des applications, ARMOR met en place des réservations sur l'ensemble des ressources CPU, mémoire RAM et réseau utilisées. Ces réservations sont effectuées par les composants d'ARMOR via les mécanismes de réservation des équipements. Les applications sont ensuite liées à ces réservations grâce à l'identifiant de leurs processus. ARMOR satisfait ainsi les contraintes du domaine d'étude de non modification des équipements et des applications.

Nous avons ensuite évalué les réservations mises en place par ARMOR. Pour ce faire, une application est démarrée avec les réservations nécessaires et du bruit est généré sur les différentes ressources utilisées. Ces expériences ont montré qu'avec ARMOR l'application n'est pas gênée par le bruit et que la qualité de service utilisateur est bonne. En revanche, quand ARMOR n'est pas utilisé, l'application n'a pas assez de ressources et la qualité de service utilisateur est mauvaise.

Mise en œuvre du *framework* La dernière contribution de cette thèse est la mise en œuvre des composants d'ARMOR sur des équipements avec un système d'exploitation Linux. Nos évaluations on été réalisées sur des PCs, des ordinateurs portables et des équipements multimédia embarqués. Dans tous les cas, les composants d'ARMOR ont été installés sans modifier le système d'exploitation des équipements.

Perspectives

Les travaux proposés dans cette thèse ouvrent de nombreuses perspectives. Nous avons classé ces perspectives en trois axes. Les deux premiers axes visent à étendre nos contributions. Le dernier axe s'intéresse à l'utilisation de nos travaux dans d'autres domaines.

Extension de l'agrégation

- Il serait pertinent de regarder si une relation d'ordre existe entre les ressources CPU et mémoire RAM des équipements. Dans ce cas, l'agrégation ferait l'hypothèse que les QdR requises sur les ressource CPU des équipements e_1, e_2, e_3 sont toujours ordonnées ($e_1 < e_2 < e_3$). L'agrégation pourrait ainsi, comme sur les paramètres d'encodage, diminuer le nombre de mesures à réaliser, diminuer plus efficacement la quantité d'information à stocker et diminuer le temps d'accès aux données.
- L'agrégation proposée sur les ressources des équipements regroupe les quantités de ressources requises connues sur tous les équipements. Il serait pertinent d'établir des groupes d'équipements pour lesquels les quantités de ressources requises sont similaires. Il serait ainsi possible d'intégrer un nouvel équipement sans faire de mesure puisqu'il suffirait de connaître le groupe auquel il appartient.
- L'agrégation proposée dans cette thèse peut être utilisée, comme nous l'avons fait, pour les quantités de ressources requises sur les ressources CPU et mémoire RAM. L'agrégation pourrait aussi être utilisée pour d'autres ressources, à condition que la quantité de ressources requise soit exprimée à l'aide d'une valeur `maxQoR`.
- Il serait intéressant de voir comment regrouper plusieurs agrégations, sur des paramètres d'encodage différents, ou encore de regrouper plus efficacement plusieurs paramètres. Pour ce faire, il faudrait regarder s'il est possible de construire des relations d'ordre sur plusieurs paramètres d'encodage. Par exemple, pour les paramètres taille d'images ($CIF, VGA, hd480$) et profil d'encodage ($baseline, main, high$) les relations d'ordre suivantes existent : $CIF < VGA < hd480$ et $baseline < main < high$. Coupler ces relations donnerait une seule relation, par exemple : $(CIF, baseline) < (CIF, main) < (VGA, baseline) < (VGA, main) < (CIF, high) < (hd480, baseline) < (VGA, high) < (hd480, main) < (hd480, high)$.

Extension d'ARMOR

- Dans cette thèse, nous avons utilisé uniquement le système d'exploitation Linux. Ce dernier est représentatif des systèmes d'exploitation que l'on retrouve dans le réseau local domestique. Cependant il faudrait vérifier que les

autres systèmes d'exploitation, notamment ceux pour les mobiles, fournissent des mécanismes équivalents à ceux utilisés par ARMOR.

- Dans sa version actuelle, ARMOR gère les ressources réseau, CPU et mémoire RAM. D'autres ressources sont utilisées par les applications multimédia telles que le disque dur ou des processeurs spécialisés. Il serait important d'intégrer l'ensemble des ressources utilisées. Les processeurs spécialisés peuvent être gérés en affectant la totalité du processeur à une seule application à la fois. Il faudrait regarder s'il est possible de partager plus finement ces ressources. Les principales limites de cette approche sont le manque de support des systèmes d'exploitation présents sur ces ressources et les fonctions matérielles (par exemple les pipelines) qui rendent difficile le partage plus fin de ces ressources.
- Les réseaux locaux domestiques actuels utilisent des liens Wifi et Ethernet connectés à l'aide d'une topologie en étoile. ARMOR se base sur cette topologie pour la gestion des ressources réseau. Il serait judicieux d'adapter ARMOR dans des réseaux utilisant une autre topologie, plus complexe. ARMOR utilise des liens logiques pour simplifier la gestion des liens physiques. Il faudrait adapter cette modélisation pour des topologies plus complexes.
- Dans cette thèse, nous avons considéré qu'une application de diffusion de contenus multimédia se compose d'un serveur et d'un client. Il existe en réalité des applications plus complexes. Par exemple, une application peut avoir plusieurs clients ou des éléments entre le serveur et le client qui transforment le flux envoyé. Il serait pertinent de regarder comment ARMOR peut être adapté pour prendre en compte ce genre d'application.
- ARMOR se préoccupe uniquement des composants logiciels des applications de diffusion de contenus multimédia. ARMOR garantit que les logiciels serveur et client de cette application ont accès à assez de ressources. Les composants partagés avec d'autres applications n'ont pas été considérés dans cette thèse. Or, si ces composants n'ont pas assez de ressources, la qualité de service utilisateur peut être dégradée. Un exemple typique est le gestionnaire de fenêtres. En effet, si ce dernier n'a pas assez de ressources, la qualité de service perçue par l'utilisateur est mauvaise même si la qualité de service fournie par l'application multimédia est satisfaite. Pour empêcher ce phénomène, deux approches seraient envisageables. Tout d'abord, la mise en place de réservations pour l'exécution du composant partagé en fonction des applications qui l'utilise. Par exemple une réservation, de 10% du CPU, serait effectuée au gestionnaire de fenêtre, pour afficher la vidéo de l'application 1. Cette approche nécessite de modifier le gestionnaire de fenêtre. Une autre approche serait d'effectuer un contrôle d'admission lorsqu'une application veut utiliser un composant partagé. Pour ce faire, il faudrait connaître le coût, en termes de ressources, de chaque méthode du composant partagé. Dans ce cas, l'application serait autorisée à utiliser la méthode d'affichage du gestionnaire de fenêtre puisqu'elle consomme 10% du CPU.
- ARMOR fait l'hypothèse que les quantités disponibles sur les ressources utilisées ne varient pas pendant la diffusion du flux. Cette hypothèse n'est pas exacte pour des équipements fonctionnant sur batteries ou pour des liens réseau dont la capacité varie. Pour les liens, ARMOR effectue un test réel de la quantité de ressources disponible avant de démarrer une application. Par contre,

une fois l'application démarrée aucun contrôle n'est effectué. Dans notre étude des solutions de gestion des ressources, nous avons vu que ces questions peuvent être traitées à l'aide de contrats portant sur l'utilisation des ressources. Il serait intéressant de voir comment les mécanismes de gestion dynamique des contrats peuvent être intégrés dans ARMOR.

– Dans la version actuelle d'ARMOR, quand il n'y pas assez de ressources pour une application, elle n'est pas démarrée. Cette approche peut ne pas être satisfaisante pour un utilisateur. Il serait pertinent de proposer d'autres solutions à l'utilisateur. Par exemple, l'application pourrait être démarrée sur un mode de fonctionnement consommant moins de ressources. Il serait envisageable de choisir un autre mode de fonctionnement, utilisant les ressources différemment ou d'autres ressources. Pour répondre à ces questions, une couche logicielle assurant un lien entre QdS et QdR, basée sur ARMOR, pourrait être rajoutée. Cette nouvelle couche proposerait différentes configurations qui seraient validées ou non par ARMOR. Dans le cas où un bouquet d'applications est disponible, il serait également intéressant de guider l'utilisateur vers le choix d'un sous-ensemble d'application acceptables, du point de vue de qualité de service.

Extension à d'autres domaines

– ARMOR a été développé pour répondre aux limites imposées par le domaine d'étude. ARMOR est capable de garantir les ressources requises par une application sans la modifier. La réutilisation d'applications existantes est une problématique qui dépasse le cadre de cette thèse. Il serait intéressant de voir comment les principes mis en avant dans ce travail, notamment via ARMOR, pourraient être appliqués à d'autres contextes. Par exemple, pour réutiliser des applications dans des systèmes de contrôle.

– Les composants d'ARMOR sont configurés pour supporter différents mécanismes de gestion des ressources. Il faudrait étudier si l'abstraction de ces mécanismes peut être réutilisée pour d'autres types d'applications ou dans d'autres domaines.

Bibliographie

[1] T.F. Abdelzaher, J.A. Stankovic, Chenyang L., Ronghua Z., and Ying L. Feedback performance control in software services. *Control Systems Magazine, IEEE*, 23(3) :74 – 90, june 2003.

[2] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 4–13, 1998.

[3] L. Abeni and G. Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*, pages 70 –77, 1999.

[4] L. Abeni and G. Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27 :123–167, 2004.

[5] L. Abeni, T. Cucinotta, G. Lipari, L. Marzario, and L. Palopoli. Qos management through adaptive reservations. *Real-Time Systems*, 29(2) :131–155, 2005.

[6] L. Abeni and G. Lipari. Implementing resource reservations in linux. In *Real-Time Linux Workshop*, 2002.

[7] L. Abeni, G. Lipari, and G. Buttazzo. Constant bandwidth vs. proportional share resource allocation. In *proceedings of the IEEE International Conference on Multimedia Computing and Systems*, volume 2, pages 107 –111 vol.2, july 1999.

[8] L. Abeni and L. Palopoli. Adaptive real-time scheduling for legacy applications. In *IEEE International Conference on Emerging Technologies and Factory Automation*, pages 583–590, 2008.

[9] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole. Analysis of a reservation-based feedback scheduler. In *proceedings of the 23rd Real-Time Systems Symposium*, pages 71 – 80, 2002.

[10] L. Abeni, L. Palopoli, C. Scordino, and G. Lipari. Resource reservations for general purpose applications. *Industrial Informatics, IEEE Transactions on*, 5(1) :12 –21, feb. 2009.

[11] L. Abeni, C. Scordino, G. Lipari, and L. Palopoli. Serving non real-time tasks in a reservation environment. In *proceedings of the 9th Real-Time Linux Workshop*, 2007.

[12] M. Aldea, G. Bernat, I. Broster, A. Burns, R. Dobrin, J.M. Drake, G. Fohler, P. Gai, M.G. Harbour, G. Guidi, J.J. Gutierrez, T. Lennvall, G. Lipari, J.M. Martinez, J.L. Medina, J.C. Palencia, and M. Trimarchi. Fsf : A real-time

scheduling architecture framework. In *In Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 113 – 124, april 2006.

[13] C. Aurrecoechea, A. T. Campbell, and L. Hauw. A survey of QoS architectures. *Multimedia Systems*, 6(3) :138–151, May 1998.

[14] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[15] J.C.R. Bennett and Hui Zhang. Wf2q : worst-case fair weighted fair queueing. In *proceedings of the 15th Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation*, volume 1, pages 120 –128 vol.1, March 1996.

[16] J.C.R. Bennett and Hui Zhang. Hierarchical packet fair queueing algorithms. *Networking, IEEE/ACM Transactions on*, 5(5) :675 –689, October 1997.

[17] A. Beugnard, J.-M. Jezequel, N. Plouzeau, and D. Watkins. Making components contract aware. *Computer*, 32(7) :38–45, 1999.

[18] G. Blair, T. Coupaye, and J.B. Stefani. Component-based architecture : the Fractal initiative. *Annals of Telecommunications*, 64(1) :1–4, 2009.

[19] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (rsvp) – version 1 functional specification. RFC 2205, Internet Engineering Task Force (IETF), 1997. `http://tools.ietf.org/html/rfc2205`.

[20] J. Corbet. Controlling memory use in containers. `http://lwn.net/Articles/243795/`, 2007.

[21] Microsoft Corp. Devices profile for web services. Technical report, http ://schemas.xmlsoap.org/ws/2006/02/devprof, 2006.

[22] T. Cucinotta, L. Abeni, L. Palopoli, and F. Checconi. The wizard of os : a heartbeat for legacy multimedia applications. In *proceedings of the IEEE/ACM/IFIP 7th Workshop on Embedded Systems for Real-Time Multimedia.*, pages 70 –79, oct. 2009.

[23] T. Cucinotta, F. Checconi, L. Abeni, and L. Palopoli. Self-tuning schedulers for legacy real-time applications. In *proceedings of the 5th European conference on Computer systems*, pages 55–68. ACM, 2010.

[24] T. Cucinotta, L. Palopoli, L. Marzario, and G. Lipari. AQuoSA—adaptive quality of service architecture. *Software : Practice and Experience*, 39(1) :1–31, 2009.

[25] T. Cucinotta, L. Palopoli, L. Marzario, G. Lipari, and L. Abeni. Adaptive reservations in a linux environment. In *proceedings of the 10th Real-Time and Embedded Technology and Applications Symposium*, pages 238 – 245, May 2004.

[26] F. Cuomo. An Architectural Model to Provide QoS in a Home Network and its Evaluation in a Real Testbed. *Journal of Networks*, 3(6) :44–53, jun 2008.

[27] B. Dasarathy, S. Gadgil, R. Vaidyanathan, K. Parmeswaran, B. Coan, M. Conarty, and V. Bhanot. Network qos assurance in a multi-layer adaptive resource management scheme for mission-critical applications using the corba middleware framework. In *proceedings of the 11th Real Time and Embedded Technology and Applications Symposium*, pages 246 – 255, march 2005.

[28] M.A. de Miguel, J.F. Ruiz, and M. Garcia. Qos-aware component frameworks. In *proceedings of the 10th IEEE International Workshop on Quality of Service*, pages 161 – 169, 2002.

[29] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *SIGCOMM Comput. Commun. Rev.*, 19 :1–12, August 1989.

[30] M. Devera. Hierachical token bucket theory. `http://luxik.cdi.cz/devik/qos/htb/manual/theory.htm`, 2002.

[31] M. Ditze, T. Bresser, and F. Berger. Resource adaptation for audio-visual devices in the UPnP QoS architecture. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications-Volume 2*, pages 543–547. IEEE Computer Society Washington, DC, USA, 2006.

[32] DLNA. Dlna home networked device interoperability guidelines v1.0. Technical report, Digital Living Network Alliance, 2004.

[33] A. Eswaran and R. Rajkumar. Energy-aware memory firewalling for qos-sensitive applications. In *proceedings of the 17th Euromicro Conference on Real-Time Systems*, pages 11 – 20, july 2005.

[34] J-P. Fassino, J-B. Stefani, J. L. Lawall, and G. Muller. Think : A software framework for component-based operating system kernels. In *proceedings of the Usenix Annual Technical Conference*, pages 73–86. USENIX Association, 2002.

[35] D. Fleeman, M. Gillen, A. Lenharth, M. Delaney, L. Welch, D. Juedes, and C. Liu. Quality-based adaptive resource management architecture (QARMA) : a CORBA resource management service. In *proceedings of the 18th International Parallel and Distributed Processing Symposium*, page 116, 2004.

[36] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Trans. Netw.*, 3 :365–386, August 1995.

[37] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Seventh International Workshop on Quality of Service (IWQoS)*, pages 27 –36, july 1999.

[38] A. Ganz, K. Wongthavarawat, and A. Phonphoem. Q-soft : software framework for qos support in home networks. *Computer Networks*, 42(1) :7 – 22, 2003. Contains papers of the Theme Issue 'Small and Home Networks'.

[39] D. Gao, J. Cai, and K.N. Ngan. Admission control in IEEE 802.11e wireless LANs. *IEEE network*, 19(4) :6–13, 2005.

[40] S. Ghahramani. *Fundamentals of probability(2nd Edition)* . Prentice Hall, 2000.

[41] S. Ghosh and R.R. Rajkumar. Resource management of the os network subsystem. In *proceedings of the Fifth IEEE International Symposium Object-Oriented Real-Time Distributed Computing*, pages 271 –279, 2002.

[42] P. Goyal, X. Guo, and H. M. Vin. A hierical cpu scheduler for multimedia operating systems. In *Proc. of the 2nd OSDI Symposium*, 1996.

[43] Object Management Group. The common object request broker : Architecture and specification. Technical report, 2.3 ed, 1999.

[44] L. J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *proceedings of the 19th Annual Symposium on Foundations of Computer Science*, pages 8 –21, oct. 1978.

[45] R. Lanphier H. Schulzrinne, A. Rao. Real time streaming protocol (rtsp). RFC 2326, Internet Engineering Task Force (IETF), 1998. `http://tools.ietf.org/html/rfc3260`.

[46] H. Hartig, S. Zschaler, M. Pohlack, R. Aigner, S. Gobel, C. Pohl, and S. Rottger. Enforceable component-based realtime contracts. *Real-Time Systems*, 35(1) :1–31, 2007.

[47] JA Hartigan and MA Wong. Algorithm AS 136 : A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1) :100–108, 1979.

[48] G.T. Heineman, J. Loyall, and R. Schantz. Component technology and qos management. *Component-Based Software Engineering*, pages 249–263, 2004.

[49] E. Hernández and J. Vila. A new approach to optimize bandwidth reservation for real-time video transmission with deterministic guarantees. *Real-Time Imaging*, 9(1) :11 – 26, 2003.

[50] E. Hide, T. Stack, J. Regehr, and J. Lepreau. Dynamic cpu management for real-time, middleware-based systems. In *proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 286 – 295, may 2004.

[51] Lei Huang, S. Kumar, and C.-C.J. Kuo. Adaptive resource allocation for multimedia qos management in wireless networks. *Vehicular Technology, IEEE Transactions on*, 53(2) :547 – 558, march 2004.

[52] ISO. Iso/iec 11172-3 coding of moving pictures and associated audio for digital storage media at up to about 1,5 mbit/s – part 3 : Audio. Technical report, ISO, 1993.

[53] ISO. Iso/iec 14496-3 coding of audio-visual objects – part 3 : Audio. Technical report, ISO, 1999.

[54] ITU-T. Terms and definitions related to quality of service recommendation e.800. Technical report, TU Telecommunication Standardization Sector (ITU-T), 1994.

[55] ITU-T. Recommendation x.224 (11/95) : Open systems interconnection - protocol for providing the connection-mode transport service. Technical report, ITU-T, 1995.

[56] ITU-T. B-isdn asynchronous transfer mode functional characteristics. Recomendation, Telecommunication Standardization Sector of ITU, 1999.

[57] ITU-T and Int. Standards Org./Int. Electrotech. Comm. (ISO/IEC) JTC 1. Generic coding of moving pictures and associated audio information–Part 2 : Video. *Recommendation H.262 and ISO/IEC 13 818-2 (MPEG-2 Video)*, 1994.

[58] H. KALVA, R. SHANKAR, T. PATEL, and C. CRUZ. Resource estimation methodology for multimedia applications. In *Proceedings of SPIE, the International Society for Optical Engineering*, pages 65040L–1. Society of Photo-Optical Instrumentation Engineers, 2007.

[59] S. Kato, R. Rajkumar, and Y. Ishikawa. Airs : Supporting interactive real-time applications on multicore platforms. In *22nd Euromicro Conference on Real-Time Systems (ECRTS)*, pages 47 –56, july 2010.

[60] D. Ko, S. Han, H. Cha, and R. Ha. A traffic control system for IEEE 802.11 networks based on available bandwidth estimation. *Wireless Communications and Mobile Computing*, 8(4) :407–419, 2008.

[61] C. Kohnen, C., V. Rakocevic, M. Rajarajan, and R. Jager. Qosilan - a heterogeneous approach to quality of service in local area networks. In *proceedings of Advances in Multimedia, International Conference on*, pages 109–112. IEEE Computer Society, 2010.

[62] Á. Kovács, I. Gódor, S. Rácz, and T. Borsos. Cross-layer quality-based resource reservation for scalable multimedia. *Computer Communications*, 33(3) :283 – 292, 2010.

[63] A. Ladbrook, G. Cottle, and R. Gallagher. Connected home : Beyond the pc. Technical report, informa telecoms and media, 2010.

[64] K. Lakshmanan and R. Rajkumar. Distributed resource kernels : OS support for End-To-End resource isolation. In *proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 195–204, 2008.

[65] R. M. Laverty. Robust open component based software architecture for configurable devices project. Technical report, Information Technology for European Advancement, 2003.

[66] G. Lazar, V. Dobrota, and T. Blaga. Practical evaluation of h.264 video streaming over ieee 802.11e devices by cross-layer signaling. In *proceedings of the 16th IEEE Workshop on Local and Metropolitan Area Networks*, pages 13 –18, 3-6 2008.

[67] H. Lee, S. Moon, and J. W. Kim. Enhanced upnp qos architecture for network-adaptive streaming service in home networks. *Consumer Electronics, IEEE Transactions on*, 53(3) :898–904, Aug. 2007.

[68] J. P. Lehoczky, L. Sha, and J. K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *IEEE Real-Time Systems Symposium*, pages 261–270, 1987.

[69] G. Lipari and S. Baruah. Greedy reclamation of unused bandwidth in constant-bandwidth servers. In *proceedings of the 12th Euromicro Conference on Real-Time Systems*, pages 193 –200, 2000.

[70] C. Liu and C. Zhou. Providing quality of service in ieee 802.11 wlan. In *Proceedings of the Advanced Information Networking and Applications*, pages 817–824, 2006.

[71] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a Hard-Real-Time environment. *J. ACM*, 20(1) :46–61, 1973.

[72] M. Louvel, P. Bonhomme, J-P. Babau, and A. Plantec. A network resource management framework for multimedia applications distributed in heterogeneous home networks. In *Poceedings of the IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 724 –731, march 2011.

[73] M. Louvel, J. Pulou, A. Plantec, and J-P. Babau. Quantity of resource aggregation for heterogeneous resource reservation for multimedia applications. In *Work In Progess session of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4, 2010.

[74] M. Louvel, J. Tous, A. Plantec, and J-P. Babau. Ensuring qos of multimedia applications in heterogeneous home networks : the cpu use case. In *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC)*, 2011. to be published available at.

[75] P. Manghwani, J. Loyall, P. Sharma, M. Gillen, and J. Ye. End-to-end quality of service management for distributed real-time embedded applications. In *Proceedings. 19th IEEE International Parallel and Distributed Processing Symposium*, page 138a. IEEE, 2005.

[76] D. Marpe, H. Schwarz, and T. Wiegand. Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, 13(7) :620 – 636, july 2003.

[77] L. Marzario, G. Lipari, P. Balbastre, and A. Crespo. Iris : a new reclaiming algorithm for server-based real-time systems. In *proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 211 – 218, May 2004.

[78] P.E. McKenney. Stochastic fairness queueing. In *proceedings of the Ninth Annual Joint Conference of the IEEE Computer and Communication Societies. 'The Multiple Facets of Integration'*, pages 733 –740 vol.2, June 1990.

[79] P. Menage. Cgroups. `http://www.mjmwired.net/kernel/Documentation/cgroups.txt`.

[80] P Menage. Adding generic process containers to the linux kernel. In *Proceedings of the Linux Symposium*, volume 2, page 45–57, 2007.

[81] C.W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves for multimedia operating systems. Technical report, School of Computer Science, Carnegie Mellon University, 1993.

[82] C.W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves : operating system support for multimedia applications. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 90 –99, May 1994.

[83] motorola. white paper : Home networking technologies. Technical report, motorola Inc, 2009.

[84] K. Nahrstedt, A. Arefin, R. Rivas, P. Agarwal, Z. Huang, W. Wu, and Z. Yang. Qos and resource management in distributed interactive multimedia environments. *Multimedia Tools and Applications*, 51 :99–132, 2011.

[85] K. Nahrstedt, H. Chu, and S. Narayan. QoS-aware resource management for distributed multimedia applications^{1}. *Journal of High Speed Networks*, 7(3) :229–257, 1998.

[86] K. Nahrstedt and J.M. Smith. The qos broker [distributed multimedia computing]. *Multimedia, IEEE*, 2(1) :53 –67, spring 1995.

[87] K. Nahrstedt and R. Steinmetz. Resource management in networked multimedia systems. *Computer*, 28(5) :52–63, 1995.

[88] K. Nahrstedt, Dongyan Xu, D. Wichadakul, and Baochun Li. Qos-aware middleware for ubiquitous and heterogeneous environments. *Communications Magazine, IEEE*, 39(11) :140 –148, November 2001.

[89] Joint Video Team of ITU-T and ISO/IEC JTC 1. Draft itu-t recommendation and final draft international standard of joint video specification (itu-t rec. h.264 | iso/iec 14496-10 avc). Technical report, Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVT-G050, march 2003.

[90] S. Oikawa and R. Rajkumar. Portable rk : a portable resource kernel for guaranteed and enforced timing behavior. In *Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium*, pages 111 –120, 1999.

[91] OMG. Umltm profile for modeling quality of service and fault tolerance characteristics and mechanisms. Technical report, OMG, 2004.

[92] C.S. Pabla. Completely fair scheduler. *Linux Journal*, 2009(184) :4, 2009.

[93] L. Palopoli and L. Abeni. Legacy real-time applications in a reservation-based system. *Industrial Informatics, IEEE Transactions on*, 5(3) :220 –228, aug. 2009.

[94] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks : the single-node case. *IEEE/ACM Trans. Netw.*, 1 :344–357, June 1993.

[95] A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks : the multiple node case. *Networking, IEEE/ACM Transactions on*, 2(2) :137 –150, April 1994.

[96] G. Parmer and R. West. Hires : a system for predictable hierarchical resource management. In *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 211 – 218, april 2011.

[97] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. Resource kernels : A resource-centric approach to real-time and multimedia systems. In *Conference on Multimedia Computing and Networking*, pages 150—164, 1998.

[98] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for qos management. In *proceedings of the 18th IEEE Real-Time Systems Symposium*, pages 298 –307, December 1997.

[99] T. Renowden and T. Cripps. Multi-screen video : matching demand to devices. Technical report, ovum, 2009.

[100] L. Rizvanovic and G. Fohler. The MATRIX - a framework for real-time resource management for video streaming in networks of heterogeneous devices. In *International Conference on Consumer Electronics, Digest of Technical Papers.*, pages 1–2, 2007.

[101] M. Roitzsch and M. Pohlack. Principles for the prediction of video decoding times applied to mpeg-1/2 and mpeg-4 part 2 video. In *Proceedings of the 27th IEEE Real-Time Systems Symposium*, 2006.

[102] M. Roitzsch and M. Pohlack. Video quality and system resources : Scheduling two opponents. *Journal of Visual Communication and Image Representation*, 19(8) :473–488, 2008.

[103] G.C. Rota. The number of partitions of a set. *The American Mathematical Monthly*, 71(5) :498–504, 1964.

[104] S.M. Sadjadi, S. Shimizu, J. Figueroa, R. Rangaswami, J. Delgado, H. Duran, and X. Collazo. A modeling approach for estimating execution time of long-running scientific applications. In *Proceedings of the Fifth High-Performance Grid Computing Workshop*. Citeseer, 2008.

[105] P. Sanchez, J. Barreda, and J. Ocon. Integration of domain-specific models into a mda framework for time-critical embedded systems. In *International Workshop on Intelligent Solutions in Embedded Systems*, pages 1 –15, july 2008.

[106] R. Schantz, J. Loyall, M. Atighetchi, and P. Pal. Packaging quality of service control behaviors for reuse. In *Proceedings. Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 375 –385, 2002.

[107] D. C. Schmidt, D. L. Levine, and S. Mungee. The design of the tao real-time object request broker. *Computer Communications*, 21 :294—324, 1998.

[108] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp : A transport protocol for real-time applications. RFC 3550, Internet Engineering Task Force (IETF), 2003. `http://tools.ietf.org/html/rfc3550`.

[109] B. Selic. A generic framework for modeling resources with uml. *Computer*, 33(6) :64 –69, June 2000.

[110] N. Shankaran, X.D. Koutsoukos, D.C. Schmidt, Yuan X., and Chenyang L. Hierarchical control of multiple resources in distributed real-time and embedded systems. In *18th Euromicro Conference on Real-Time Systems*, pages 10 pp. –160, 2006.

[111] N. Shankaran, N. Roy, D. C. Schmidt, X. D. Koutsoukos, Y. Chen, and C. Lu. Design and performance evaluation of an adaptive resource management framework for distributed real-time and embedded systems. *EURASIP J. Embedded Syst.*, 8(3) :1–20, 2008.

[112] M. Sojka and Z. Hanzalek. Modular architecture for real-time contract-based framework. In *IEEE International Symposium on Industrial Embedded Systems*, pages 66 –69, july 2009.

[113] M. Sojka, P. Písa, D. Faggioli, T. Cucinotta, F. Checconi, Z. Hanzálek, and G. Lipari. Modular software architecture for flexible reservation mechanisms on heterogeneous resources. *Journal of Systems Architecture*, 57(4) :366 – 382, 2011.

[114] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1 :27–60, 1989.

[115] D.C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Proceedings of the Third usenix- osdi. pub-usenix*, februray 1998.

[116] L. Steffens, G. Fohler, G. Lipari, and G. Buttazzo. Resource reservation in real-time operating systems - a joint industrial and academic position. In *International Workshop on Advanced Real-Time Operating System Services (ARTOSS)*, pages 25–30, 2003.

[117] I. Stoica, H. Abdel-Wahab, K. Jeffay, S.K. Baruah, J.E. Gehrke, and C.G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *proceedings of the Real-Time Systems Symposium*, page 288. Published by the IEEE Computer Society, 1996.

[118] J.K. Strosnider, J.P. Lehoczky, and Lui Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *Computers, IEEE Transactions on*, 44(1) :73 –91, January 1995.

[119] J. Sung, D. Kim, H. Song, J. Kim, S. Y. Lim, and J. S. Choi. Upnp based intelligent multimedia service architecture for digital home network. In *The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance.*, page 6 pp., april 2006.

[120] Clemens Szyperski. *Component software : beyond object-oriented programming*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1998.

[121] S. Tasaka and Y. Ishibashi. Mutually compensatory property of multimedia QoS. In *IEEE International Conference on Communications*, volume 2, 2002.

[122] J.C. Tournier, J.P. Babau, and V. Olive. Qinna, a component-based qos architecture. In *Component-Based Software Engineering, 8th International Symposium, St. Louis, USA*, pages 107–122. Springer, 2005.

[123] G. Fohler (TUKL), A. Neundorf (TUKL), K-E. Årzén (ULUND), C. Lucarz (EPFL), M. Mattavelli (EPFL), V. Noel (AKAtech), C. von Platen (Ericsson), G. Buttazzo (SSSA), E. Bini (SSSA), and C. Scordino (EVI). State of the art assessment. Technical report, ACTORS Adaptivity and Control of Resources in Embedded Systems, 2008.

[124] UPnP^{TM}. Upnp av architecture. Technical report, UPnP Forum Std. Rev. 0.83, 2002.

[125] R. Vanegas, J. A. Zinky, J. P. Loyall, D. Karr, R. E. Schantz, and D. E. Bakken. Quo's runtime support for quality of service in distributed objects. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, Middleware '98, pages 207–222, London, UK, 1998. Springer-Verlag.

[126] C. A. Waldspurger and W. E. Weihl. Lottery scheduling : flexible proportional-share resource management. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.

[127] N. Wang and C. Gill. Improving Real-Time system con.guration via a QoS-Aware CORBA component model. In *Proceedings of the 37th Annual Hawaii*

International Conference on System Sciences - Track 9 - Volume 9, page 90273.2. IEEE Computer Society, 2004.

[128] C.S. Wong, I. Tan, R.D. Kumari, and F. Wey. Towards achieving fairness in the Linux scheduler. *ACM SIGOPS Operating Systems Review*, 42(5) :34–43, 2008.

[129] C.S. Wong, I.K.T. Tan, R.D. Kumari, J.W. Lam, and W. Fun. Fairness and interactive performance of o(1) and cfs linux kernel schedulers. In *International Symposium on Information Technology*, volume 4, pages 1 –8, august 2008.

[130] T. Yang, T. Liu, E. D. Berger, S. F. Kaplan, and J. E. B. Moss. Redline : first class support for interactivity in commodity operating systems. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 73–86, Berkeley, CA, USA, 2008. USENIX Association.

[131] R. Yu, Y. Zhang, C. Huang, and R. Gao. Joint admission and rate control for multimedia sharing in wireless home networks. *Comput. Commun.*, 33(14) :1632–1644, 2010.

[132] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP : A new resource reservation protocol. *Network, IEEE*, 7(5) :8–18, 1993.

Quatrième partie

Annexes

Annexe A

Publications liées à la thèse

Pendant les travaux de thèse, l'auteur a participé à la rédaction des publications suivantes, présentées dans des conférences internationales :

- C. El Kaed, L. Petit, M. Louvel, A. Chazalet, Y. Denneulin, F-G. Ottogalli. INSIGHT : Interoperability and Service Management for the Digital Home. In *Proceedings of the 12th International Middleware Conference Industrial track*. to be published, Lisboa, Portugal, December 2011.
- M. Louvel, J. Tous, A. Plantec, J-P. Babau. Ensuring QoS of multimedia applications in heterogeneous home networks : the CPU use case. In *Proceedings of the IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC)*, to be published, Melbourne, Vic, Australia, October 2011.
- M. Louvel, P. Bonhomme, A. Plantec, J-P. Babau. A network resource management framework for multimedia applications distributed in heterogeneous home networks. In *Proceedings of the Advanced Information Networking and Applications (AINA)*, pages 724-731, Singapore, Singapore, March 2011.
- M. Louvel, J. Pulou, A. Plantec, J-P. Babau, Quantity of resource aggregation for heterogeneous resource reservation for multimedia applications. In *the Work In Progess session of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1-4, Bilbao, Spain, September 2010.
- M. Anne, H. Ruan, T. Jarboui, M. Lacoste, O. Lobry, G. Lorant, M. Louvel, J. Navas, V. Olive, J. Polakovic, M. Poulhies, J. Pulou, S. Seyvoz, J. Tous, T. Watteyne. Think : View-Based Support of Non-functional Properties in Embedded Systems. In *International Conference on Embedded Software and Systems (ICESS)*, HangZhou, Zhejiang, pages 147-156, China, May 2009.

Annexe B

Exemples de fichier xml

B.1 Fichier “home file”

```
<?xml version="1.0" encoding="UTF-8"?>
<home>
  <devices>
    <device>
      <device_id>0</device_id>
    <!-- Functionnal Properties of a device -->
      <fp>
    <hostname>device0</hostname>
    <communication_resources>
      <itf name="eth0">
      <itf_id>0</itf_id>
      <ip_address>192.168.0.1</ip_address>
      </itf>
    </communication_resources>
      </fp>
    <!-- Non Functionnal Properties of a device -->
      <nfp>
    <communication_resources>
      <itf>
      <itf_id>0</itf_id>
      <network_type>Ethernet</network_type>
      <capacity>100000</capacity>
      <availableQoR>100000</availableQoR>
      </itf>
    </communication_resources>
    <computing_resources>
      <cpu>
      <id>0</id>
      <capacity>200</capacity>
      <availableQoR>200</availableQoR>
      <scheduler>CFS</scheduler>
      </cpu>
    </computing_resources>
    <storage_resources>
      <mem>
      <id>0</id>
      <capacity>2000000</capacity> <!--in Kilo Bytes-->
      <availableQoR>2000000</availableQoR> <!--in Kilo Bytes-->
      </mem>
```

```
    </storage_resources>
      </nfp>
    </device>
    <device>
      <device_id>1</device_id>
      <fp>
    <hostname>device1</hostname>
    <communication_resources>
      <itf name="eth0">
      <itf_id>0</itf_id>
      <ip_address>192.168.0.2</ip_address>
      </itf>
      <itf name="wlan0">
      <itf_id>1</itf_id>
      <ip_address>192.168.0.3</ip_address>
      </itf>
    </communication_resources>
      </fp>
      <nfp>
    <communication_resources>
      <itf>
      <itf_id>0</itf_id>
      <network_type>Ethernet</network_type>
      <net_qos>no</net_qos>
      <capacity>100000</capacity>
      <availableQoR>100000</availableQoR>
      </itf>
      <itf>
      <itf_id>1</itf_id>
      <network_type>Wifi</network_type>
      <net_qos>802.11e</net_qos>
      <capacity>54000</capacity>
      <availableQoR>54000</availableQoR>
      </itf>
    </communication_resources>
    <computing_resources>
      <cpu>
      <id>0</id>
      <capacity>100</capacity>
      <availableQoR>100</availableQoR>
      <scheduler>CBS</scheduler>
      </cpu>
    </computing_resources>
    <storage_resources>
      <mem>
      <id>0</id>
      <capacity>2000000</capacity> <!--in Kilo Bytes-->
      <availableQoR>2000000</availableQoR> <!--in Kilo Bytes-->
      </mem>
    </storage_resources>
      </nfp>
    </device>
  </devices>
  <links>
    <link>
      <link_id>01</link_id>
      <type>Ethernet</type>
```

```
      <src>0</src>
      <dest>1</dest>
      <capacity>100000</capacity>
      <availableQoR>100000</availableQoR>
      <sharedQoR>0</sharedQoR>
      <loss_rate>0</loss_rate>
      <jitter>0</jitter>
      <delay>0</delay>
    </link>
    <link>
      <link_id>99</link_id>
      <type>wifi</type>
      <shared>1</shared>
      <capacity>54000</capacity>
      <availableQoR>54000</availableQoR>
      <sharedQoR>0</sharedQoR>
      <loss_rate></loss_rate>
      <jitter></jitter>
      <delay></delay>
    </link>
  </links>
  <homeGateway>
    <net_qos_wifi>802.11e</net_qos_wifi>
    <net_qos_ethernet>802.11e</net_qos_ethernet>
  </homeGateway>
</home>
```

Listing B.1 – Exemple de fichier “home”

B.2 Fichier décrivant un scénario

```
<scenario>
  <stream_name>/path/to/my/file/filename.mov</stream_name>
  <link>
  <link_id>01</link_id>
  <net_protocol>RTSP</net_protocol>
  <transport_protocol>UDP</transport_protocol>
  </link>
  <server>
  <device_id>0</device_id>
  <server_port>1234</server_port>
  <software>vlc</software>
  </server>
  <client>
  <device_id>1</device_id>
  <client_port_start>1234</client_port_start>
  <client_port_stop>1239</client_port_stop>
  <software>mplayer</software>
  <execution_resources>
    <exec_resource>CPU</exec_resource>
  </execution_resources>
  </client>
</scenario>
```

Listing B.2 – Exemple de fichier XML contenant un scénario

Annexe C

Stress mémoire RAM

```
int hogvm (long long bytes, long long stride, long long hang, int keep)
{
  long long i;
  char *ptr = 0;
  char c;
  int do_malloc = 1;

  while (1)
    {
    if (do_malloc)
        {
      dbg (stdout, "allocating %lli bytes ...\n", bytes);
      if (!(ptr = (char *) malloc (bytes * sizeof (char))))
            {
        err (stderr, "hogvm malloc failed: %s\n", strerror (errno));
        return 1;
            }
      if (keep)
        do_malloc = 0;
        }

    dbg (stdout, "touching bytes in strides of %lli bytes ...\n",
        stride);
    for (i = 0; i < bytes; i += stride)
      ptr[i] = 'Z';             /* Ensure that COW happens.  */

    if (hang == 0)
        {
      dbg (stdout, "sleeping forever with allocated memory\n");
      while (1)
        sleep (1024);
        }
    else if (hang > 0)
        {
      dbg (stdout, "sleeping for %llis with allocated memory\n", hang);
      sleep (hang);
        }

    for (i = 0; i < bytes; i += stride)
        {
      c = ptr[i];
      if (c != 'Z')
```

```
            {
        err (stderr, "memory corruption at: %p\n", ptr + i);
        return 1;
            }
        }

    if (do_malloc)
        {
      free (ptr);
      dbg (stdout, "freed %lli bytes\n", bytes);
        }
    }

  return 0;
}
```

Listing C.1 – *stress* sur la mémoire RAM

Annexe D

Scripts shells

```
#!/bin/bash

if [ $(id -u) -ne 0 ]
then
 echo "You must be 'root' to run $0"
 exit 0
fi

function usage {
    echo "usage : "
    echo "  $0 pid group_name"
}

if [ $# -lt 2 ]
then
    echo "missing args"
  usage $@
    exit 1
fi

#import variables that are shared between scripts
INSTALL_DIR=$(dirname $0)
. $INSTALL_DIR/variables.sh

#First argument is the PID
PID=$1
#Second argument is the cgroup name
if [ $2 = $NO_GROUP ]
then
  # we want to attach this pid to the root cgroup
  CGROUP_NAME=""
else
  CGROUP_NAME=$2
fi

#Attach the process to the cgroup
/bin/echo $PID > $CGROUP_MEMORY_DIR/$CGROUP_NAME/tasks
/bin/echo $PID > $CGROUP_CPU_DIR/$CGROUP_NAME/tasks

exit 0
```

Listing D.1 – script *attach_pid_to_cgroup.sh* attachant un processus à un *cgroup*

Glossaire

802.11e norme de QdS réseau pour le protocole Wifi. 10

802.1p norme de QdS réseau pour le protocole Ethernet. 10

aac standard d'encodage audio. 14

ADL *Architecture Description Language* . 75

aggMaxQoR *quantité de ressources requise agrégeant plusieurs maxQoR.* 63

API *Application Programing Interface.* 39

AQuoSA *Adaptive QUality Of Service Architecture* . 39

ATM *Asynchronous Transfer Mode* . 44

CABAC *Context-Adaptive Binary Arithmetic Coding* . 14

CBQ *Class-based queueing.* 35

CBR *Constant Bit Rate.* 14

CBS *Constant Bandwidth Server.* 29, 83, 85, 90, 96

CFS *Completely Fair Scheduler* . 31, 89, 96

cgroups *control cgroups.* 28

CIAO *Component-Integrated ACE ORB.* 42

CIF taille d'image de 352x288 pixels. 54, 61

clustering affectation d'un ensemble d'observations dans un sous-ensemble (appelé *cluster*) de telle manière que les observations dans un *cluster* soient similaires. 66

contrat de QdR un contrat de QdR est établi entre une application et une ressource. Le contrat porte sur la QdR requises, par l'application, sur cette ressource et les contraintes liées à cette QdR. 25

CORBA *Common Object Request Broker Architecture.* 41

CPU *Central Processing Unit.* 56, 82, 89, 96

DLNA *Digital Living Network Alliance* . 13

DPWS *Devices Profile for Web Services.* 13

DSP *Digital Signal Processor* . 18, 19

EDF *Earliest Deadline First.* 29

EEVDF *Earliest Eligible Virtual Deadline First.* 32

framework un *framework* d'exécution contient l'ensemble des services et des composants fournissant un cadre d'exécution aux applications. 38

GCRM *Global CPU Resources Manager*. 73

GMRM *Global Memory Resources Manager*. 73

GNRM *Global Network Resources Manager*. 73

GPL *General Public License*. 119

GPS *Generalized Processor Sharing* . 30

GRM *Global Resources Manager*. 72

GRUB *Greedy Reclamation of Unused Bandwidth*. 30

H-PFQ *Hierarchical Packet Fair Queuing* . 35

h.264 standard d'encodage vidéo. 13

hd480 taille d'image de 852x480 pixels. 61

hd720 taille d'image de 1280x720 pixels. 61

HGI *Home Gateway Initiative* . 11

HGRUB *Hard GRUB*. 30

HTB *Hierarchical Token Bucket*. 35, 79, 93

IO *Input / Output*. 28

IP *Internet Protocol*. 8

iperf logiciel de mesure de trafic réseau. 82

IRIS *Idle-time Reclaiming Improved Server*. 30

ISO *International Organization for Standardisation*. 8

ITU-T *International Telecommunication Union*. 16

LCRM *Local CPU Resource Manager*. 73

LMRM *Local Memory Resource Manager*. 73

LNRM *Local Network Resources Manager*. 73

LRM *Local Resources Manager LRM* . 73

maxQoR *quantité de ressources requise maximale*. 58

Mind implémentation en C du modèle Fractal, ciblant le développement de systèmes embarqués à composants. 75

MIPS *Million Of Instructions Per Second*. 112

mode *pull* mode de diffusion où le serveur attend une demande d'un client pour démarrer la diffusion. 16

mode *push* mode de diffusion où le serveur commence la diffusion dès qu'il est démarré, les clients se connectent à la diffusion en cours. 16

mp3 standard d'encodage audio. 13

mpeg4-part2 standard d'encodage vidéo. 13

MTU *Maximum Transmission Unit* . 130

nbFaNeg *nombre de faux négatifs introduits par une table de mapping agrégée.* 64

OOM *killer* *Out Of Memory killer, utilisé par le noyau Linux pour détruire un processus quand il n'y pas assez de mémoire disponible* . 36

ORB *Object Request Broker*. 181

OSI *Open Systems Interconnexion*. 8

QARMA *Quality-based Adaptive Resource Management Architecture.* 43

qdisc *queuing discipline.* 79

QdR Quantité de Ressources. 17, 20

QdS *Qualité de Service.* 16

QuO *Quality Objects.* 42

RACE *Resource Allocation and Control Engine* . 42

RAM *Random-Access Memory.* 56, 85, 91, 98

reqSharedQoR *quantité de ressources requise pour les pics d'utilisation.* 88

RPC *Remote Procedure Call* . 75

RSVP *Resource Reservation Protocol* . 34

RTP *Real-time Transport Protocol* . 15

RTSP *Real Time Streaming Protocol* . 15

segment L2 ensemble des liens physiques de même type dans un réseau local domestique. 12

SFQ *Stochastic Fairness Queueing.* 80

SMP *Symetric Multi Processor* . 18, 82

statMaxQoR *quantité de ressources requise maximale sans tenir compte des pics d'utilisation.* 58

STB *Set-Top-Box.* 12, 127

stress outil utilisé pour générer du bruit sur les ressources CPU et mémoire RAM. 143

TAO *The ACE ORB.* 42

tc *traffic control* . 35

UML *Unified Modeling Language.* 11

UPnP *Universal Plug and Play* . 13, 49

VBR *Variable Bit Rate.* 14, 24

VGA taille d'image de 640x480 pixels. 61

WCET *Worst Case Execution Time* . 29

WFQ *Weighted Fair Queuing.* 32

équipement équipement du réseau local domestique. 12

Architecture de gestion et de contrôle des ressources pour les applications multimédia dans le réseau local domestique

Résumé

Le réseau local domestique est un environnement ouvert, hétérogène et distribué pour lequel il est primordial de garantir la qualité de service des applications multimédia. Des mécanismes de réservation de ressources (CPU, mémoire, réseau) et des architectures utilisant ces mécanismes existent. Les architectures et les mécanismes existants nécessitent de modifier les équipements ou les applications et ne prennent pas en compte l'hétérogénéité inhérente au domaine d'étude. Ces solutions ne sont pas adaptées au contexte du réseau local domestique.

Pour répondre au problème de l'hétérogénéité, ce travail propose de limiter le nombre de mesures à réaliser et à stocker en agrégeant les quantités de ressources. Cette agrégation est automatisée à l'aide d'algorithmes basés sur du *clustering* ou du *bin-packing*. Cette thèse propose ensuite un *framework* de gestion des ressources peu intrusif, reposant sur une architecture configurable en fonction des équipements présents. Cette architecture utilise des composants globaux qui délèguent, aux composants locaux, la gestion des ressources locales. Les composants locaux utilisent les mécanismes de réservation du système d'exploitation Linux pour garantir les quantités de ressources aux applications.

L'agrégation évaluée par simulation, réduit efficacement le nombre de mesures à réaliser et à stocker. De son côté, le *framework* de gestion des ressources est mis œuvre sur des équipements réels (des PCs, des ordinateurs portables et des équipements embarqués dédiés au multimédia), communiquant via des réseaux Wifi et Ethernet. Les évaluations du *framework* montrent que les réservations sont garanties même quand du bruit est généré sur les ressources utilisées, ce qui garantit aussi la qualité de service attendue.

Mot-clefs

Réseau local domestique, qualité de service, gestion des ressources, multimédia, hétérogénéité

Resources management and control architecture for multimedia applications in home networks

Abstract

The home network is an open, heterogeneous and distributed environment where ensuring multimedia applications' quality of service is a main concern. Mechanisms to reserve resources (CPU, memory, network) and architectures using them already exist. However, they require modifying the devices or the applications and they do not take into account the heterogeneity of the home network.

To tackle the heterogeneity related issues, this work limits the number of measurements to make and to store by aggregating the quantities of resources. This aggregation is automated using clustering or bin-packing algorithms. Then a non-intrusive resource management framework is developed upon an architecture, customized for the actual devices. This architecture uses global components that delegate to local components the management of local resources. These components rely on the resource reservation mechanisms provided by the Linux operating system in order to guarantee the resources to the applications.

The aggregation, which we evaluated through simulation, efficiently reduces the number of measurements to carry and to store. The framework has been implemented on real devices (PCs, laptops and embedded multimedia devices), bridged with wireless and Ethernet networks. The evaluations of the framework show that reservations are guaranteed even if noise is generated on the resources, which also guaranties the expected quality of service.

Keywords

Home network, quality of service, resources management, multimedia, heterogeneity

Adrr : 20, avenue Le Gorgeu 29238 Brest