RAPPORT DE STAGE

JULIEN ROTROU
DEA RESEAUX - 2002

ACCES INTELLIGENT AUX RESEAUX AMBIANTS

Encadrants : Sidi-Mohamed Senouci – Yacine Ghamri Doudane

Responsable de stage : Pr. Guy Pujolle

Introduction

Les réseaux de télécommunications mobiles sont en forte expansion et constante évolution. Le nombre de terminaux mobiles ne cessent d'augmenter jour après jour, pour prochainement dépasser le nombre de terminaux fixes, et dans le même temps, de nombreux réseaux mobiles et sans fil apparaissent. Le GSM, qui a connu un énorme succès, devrait à terme laisser sa place à l'UMTS, le réseau mobile de 3ème génération. Les réseaux sans fil ont aussi en ce moment même une énorme croissance, notamment la technologie IEEE 802.11.
C'est dans ce contexte que s'est développée la notion de réseaux ambiants, ou réseau de 4ème génération : fédérer les différents réseaux mobiles et sans fil (GSM, GPRS, UMTS, IEEE 802.11, Hiperlan, Bluetooth, HomeRF) en un système encastré et interconnecté. Le terminal possède plusieurs interfaces (ou une interface « Software Radio ») qui lui permettent d'utiliser ces différents réseaux d'accès. Ainsi l'utilisateur profite de plusieurs services offerts par des réseaux différents, et ce à partir d'un seul et même terminal et de façon transparente. Les avantages sont multiples et les moyens purement techniques sont déjà disponibles dans certains laboratoires.
Mais un problème majeur est rencontré lorsque l'on veut réaliser une architecture de ce type : le contrôle d'admission. Cette difficulté est développée dans la problématique et constitue le thème principal de mon stage. Il se déroule sous la responsabilité du professeur Guy Pujolle, au Laboratoire d'Informatique de Paris 6 (LIP6) -Université Pierre et Marie Curie et est encadré par Sidi-Mohamed Senouci et Yacine Ghamri Doudane. Après avoir observé les différents réseaux existants, le but du stage est de proposer une architecture adaptée aux objectifs de la 4ème génération, puis d'étudier la partie « contrôle d'admission » et en particulier la détermination dynamique du prix, aspect assez innovant dans ce domaine. Le plan de ce rapport suit ces étapes, après avoir défini plus précisément la problématique du sujet.

Problématique

Ce sujet entraîne plusieurs tâches : la définition d'une architecture en fonction des réseaux existant. En effet, il est important de préciser l'architecture du système avant de discuter de son contrôle d'admission. Il existe dans la littérature de nombreuses visions de systèmes 4G [3,5,13,25], qui diffèrent toutes plus ou moins et n'ont pas les mêmes exigences ni les mêmes buts. Notre vision correspond à une utilisation du contexte actuel des réseaux mobiles et sans fil, en minimisant l'ajout d'éléments nouveaux. Le terminal mobile est en contact direct avec les différents fournisseurs d'accès, déjà existants ou futurs, et les met en concurrence afin de trouver la meilleure offre. Il communique avec un seul réseau d'accès à la fois. L'architecture proposée est décrite dans la suite de ce document.
Cette architecture étant définie, elle nécessite la spécification de mécanismes de signalisation et de contrôle d'admission permettant la facturation du service en fonction de la qualité de celui-ci et de l'état des ressources du réseau. Le terminal doit être capable de choisir un accès parmi les réseaux ambiants disponibles de manière transparente à l'utilisateur, qui a tout de même la possibilité de spécifier le prix maximum ou la qualité minimum du service.
La signalisation a pour rôle de transporter ces différents paramètres tout en étant adaptée à l'environnement mobile : c'est à dire ne pas occuper trop de bande passante qui est une ressource limitée dans l'environnement radio.
Le contrôle d'admission doit permettre de s'adapter à ce contexte dynamique : afin d'être concurrentiel, il doit tenir compte de l'état des ressources et donc de la qualité du service pour proposer son prix. Mais en proposant dynamiquement le prix du service, la concurrence entraîne le phénomène de « guerre des prix » : un cycle se crée où les prix baissent légèrement chacun leur tour, pour ensuite remonter quand le prix atteint une valeur minimum.
Le but est donc de proposer un algorithme permettant d'arriver à une certaine stabilité des prix pour un contexte donné et de maximiser les revenus des différents fournisseurs d'accès, mais qui soit capable de s'adapter à un changement de contexte (une congestion faisant momentanément augmenter les prix par exemple). Mes encadrants ont pour cela proposé l'étude un algorithme du domaine de l'intelligence artificielle permettant un apprentissage par renforcement, le Q-learning.
Ces différents points (réseaux existants, architecture retenue, mécanismes de signalisation, contrôle d'admission et plus particulièrement le Q-learning, simulations) sont donc développés dans ce rapport.

1. Les réseaux ambiants

1.1. Introduction aux réseaux ambiants existants

Les réseaux ambiants existants regroupent principalement deux catégories de réseaux : les réseaux de mobiles et les réseaux sans fil. Si ces concepts sont proches et souvent imbriqués, il est important de faire la différence de signification de ces termes.
La définition de « mobile » la plus pertinente est peut-être celle ci :
« Un utilisateur mobile est défini théoriquement comme un utilisateur capable de communiquer à l'extérieur de son réseau d'abonnement tout en conservant une même adresse »[1]. Il s'agit par exemple des réseaux GSM, GPRS, et UMTS, où l'adresse de l'utilisateur reste la même quelle que soit sa localisation.
Les réseaux sans fil ont une portée moins étendue, et l'utilisateur a une mobilité plus restreinte. En contre partie, leur débit est souvent beaucoup plus élevé. Il s'agit typiquement de la norme IEEE 802.11. Les réseaux cellulaires tel que le GSM ou les réseaux satellites sont aussi sans fil, mais leur caractéristique de grande mobilité est mise en avant.
Les caractéristiques de ces réseaux ambiants sont donc différentes et les services associés à chacun le sont aussi. Après un bref aperçu de chacun, un tableau comparatif exposera les principales différences des ces réseaux.

1.1.1. Le GSM : réseau mobile de 2nde génération

Le Global System for Mobile communication est la référence des réseaux mobiles numériques: en une dizaine d'années, plus d'un français sur deux est devenu utilisateur de ce réseau mobile.
Le GSM est réseau en mode circuit destiné à la téléphonie. C'est un réseau cellulaire radio fréquence dans la bande des 900 et 1800 MHz et dont le débit utile est de 9.6 Kbps pour la transmission de données. Le service HSCSD permet d'augmenter le débit du service de données en allouant non plus un slot de la trame TDMA par utilisateur mais plusieurs.
L'architecture du réseau GSM est en quelque sorte la référence, ou l'existant, du fait de son succès. Mais le mode circuit du GSM est non adapté aux services autres que la voix : pour envoyer très peu de données dans le temps, on est obligé d'ouvrir un circuit et donc il y a un gaspillage de ressource.

1.1.2. Le GPRS : réseau mobile 2.5 génération

Un mode paquet est donc nécessaire: le GPRS répond à ce besoin. Le réseau General Packet Radio Service est une évolution du GSM. Il se superpose au réseau GSM : il tire profit de la même interface radio mais possède un nouveau réseau cœur adapté au transfert de paquet. De nouveaux terminaux compatible GPRS sont cependant nécessaire. L'idée est d'utiliser des slots libre de la trame TDMA du GSM pour transporter des paquets. Le débit théorique varie de 9.6 à 171.2 Kbps selon le nombre de slot dédié au mode paquet. Mais en pratique, pour différentes raisons, le débit utile est d'environ 10Kbps vers le réseau et de 30 à 40 Kbps en réception. Le service GPRS permet des connexions point à point en mode connecté ou non et multipoint en mode broadcast ou multicast. Il offre un accès standardisé à Internet ainsi qu'une taxation au volume de données échangées.
Le GSM et le GPRS peuvent être associé à EDGE : un nouveau système plus performant de modulation de l'interface radio. Les débits bruts sont alors trois fois supérieurs. Mais le déploiement de EDGE est plus coûteux : les modifications matérielles sont plus importantes et peuvent réduire la portée des BTS. Le E-GPRS est l'application principale d'EDGE : il respecte le cahier des charges de la troisième génération pour des coûts moindre que l'UMTS. Il se pose alors comme une solution de transition vers la 3ème génération.

1.1.3. L'UMTS : réseau mobile de 3ème génération

La 3ème génération de réseau mobile n'est pas encore définitivement définie. L'UIT a lancé un programme mondial sous forme d'un appel à proposition, l'IMT 2000, pour définir une norme mondiale. L'UMTS est une des propositions retenues, et est soutenue entre autres par l'Europe et le japon(3GPP).(Les Etats-Unis soutiennent eux le CDMA 2000.)
L'UMTS, dans sa version actuelle, utilise le réseau cœur du GSM/GPRS : cela correspond à un souci de compatibilité et de rentabilité. L'architecture reste globalement la même. Le domaine utilisateur reste composé d'un terminal avec une carte à puce (U-SIM). Les différences premières se trouvent dans le domaine radio : les antennes ou stations de base sont appelées Node B (au lieu de BTS) et les concentrateurs de Node B sont des RNC (BSC).
On ne standardise plus les services mais les outils qui permettent de les réaliser.

Les débits théoriques ont bien sûr augmenté : de 144 Kbps en campagne jusqu'à 2 Mbps pour une mobilité faible avec de bonnes conditions radio (en pratique ils sont inférieurs).
Mais cette version de l'UMTS, la release 99, est considérée comme trop lourde et les suivantes visent à réduire cette complexité. A terme, la troisième génération ne devrait plus se baser sur l'architecture de la 2nde génération mais progressivement sur une architecture tout-IP. Le Node B, la passerelle vers le RTC ainsi qu'un certain nombre de serveurs seront interconnectés directement sur un même réseau IP avec qualité de service.

1.1.4. Les réseaux sans fil : la technologie émergente

On peut décomposer ces réseaux en deux sous groupes : les WPAN et les WLAN [1].

Wireless Personal Area Network (IEEE 802.15): Bluetooth et HomeRF
Ce type de réseau, d'une portée d'une dizaine de mètres, permet la communication entre différents terminaux portables tels qu'un PC portable, un téléphone portable ou un PDA. Soutenue par plusieurs milliers de constructeurs, la technologie Bluetooth est en passe de devenir la norme des WPAN.
Une puce Bluetooth intègre tous les composants sur une surface de 9mm sur 9mm, et ce pour un coût très faible. Elle utilise des fréquences dans la bande sans licence des 2,4 Ghz. Les terminaux s'interconnectent pour former un piconet (maximum 8 terminaux), qui eux peuvent s'interconnecter pour former un scatternet.
La version 1.0 de Bluetooth permet une communication entre deux terminaux avec un débit maximum de 1 Mbps, ce qui peut être insuffisant pour certaines applications. Le groupe de travail 2.0 a pour but d'augmenter ce débit et de mettre en place des améliorations (handover, optimisation du routage, coexistence avec les autres réseaux de la bande des 2.4 Mhz).

Une autre technologie, plutôt issue de la domotique, concurrence Bluetooth : HomeRF. Une station de base communique avec différentes entités dans une portée de 50 mètres également dans la bande des 2.4 Ghz. Le débit de base est de 1.6 Mbps mais devrait augmenter. Les caractéristiques sont comparables mais elle est soutenue par beaucoup moins de constructeurs que Bluetooth.

Wireless Local Area Network: IEEE 802.11 et Hiperlan
L'IEEE 802.11 [2]est le premier standard des WLAN depuis 2001. Il connaît actuellement un grand succès grâce à sa facilité d'installation, ses performances et son coût concurrentiel. Il s'appuie sur la technique MAC de l'IEEE 802, avec toutefois une technique d'accès au canal différente (CSMA/CA). Dans la bande sans licence des 2.4 Ghz, ce réseau local sans fil a une architecture cellulaire. Un ensemble de terminaux qui communiquent directement entre eux forme un BSS. Une station de base ou point d'accès (AP) synchronise les différents terminaux de sa cellule et leur permet de communiquer vers l'extérieur de la BSS, vers un autre BSS ou

vers un quelconque autre réseau compatible. Les différents AP sont reliés entre eux par n'importe quel réseau filaire ou sans fil. Un mode ad hoc permet la communication entre deux entités 802.11 sans avoir besoin de point d'accès, mais ne permet que la communication directe sans intermédiaire ce qui est une différence capitale avec les réseaux ad hoc.
La version sur le marché est actuellement le 802.11b qui offre un débit de 11 Mbps. Elle ne propose pas de réel mécanisme de handover, puisqu'il n'est possible de changer de BSS qu'entre deux transmissions de données et non au milieu d'un dialogue. La sécurité est aussi un point faible : il est possible d'écouter les porteuses et ainsi d'intercepter tout trafic du réseau (des mécanismes de sécurité existent mais sont insuffisants). Mais cette technologie est assez récente et elle évolue rapidement vers des versions de plus en plus performantes.
Le 802.11a offre une communication jusqu'à 54 Mbps dans la bande des 5 Ghz, le 802.11e propose d'introduire de la qualité de service, une meilleure sécurité et de permettre des communications vers des terminaux hors de la BSS.
Le 802.11 est envisagé pour construire des réseaux de plus grande ampleur, à l'échelle d'un campus ou d'une ville. Les améliorations précédentes permettent en effet d'imaginer de tel réseau, appelé WMAN : Wireless Metropolitan Area Network. Des initiatives privées ont déjà été menées dans ce sens, notamment dans certaines villes américaines.

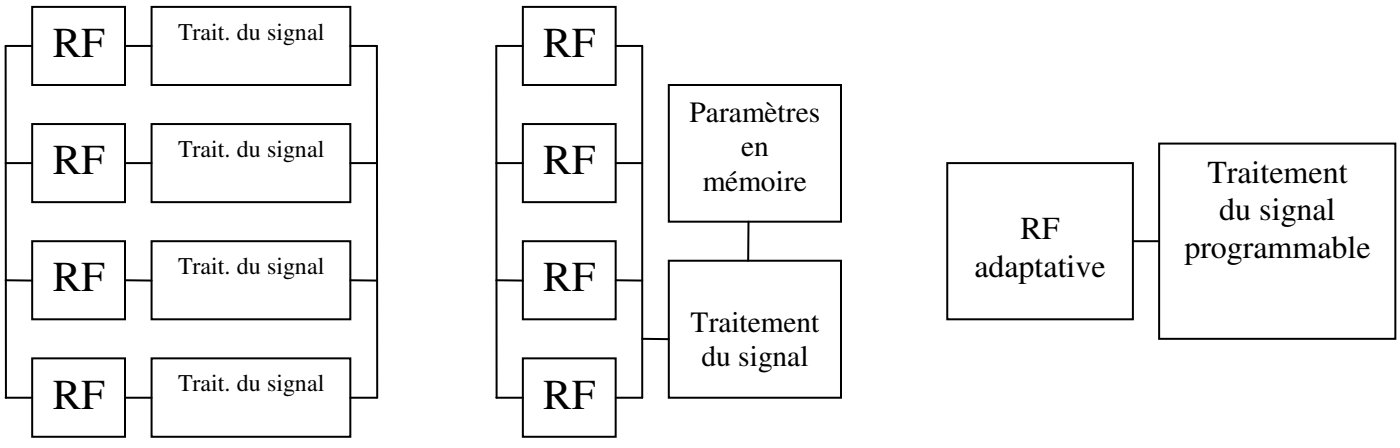
La technologie Hiperlan, d'origine européenne (ETSI) est souvent citée comme concurrent direct du 802.11, qui en a repris d'ailleurs certains aspects. Les normalisateurs ont privilégié des interfaces ATM ce qui explique peut être l'avantage pris par 802.11, basé sur Ethernet.
Il existe différents types d'Hiperlan, dont le type 1 et 2 qui correspondent à la notion de WLAN. Le type 2 améliorent le type 1, et propose un débit de 23 Mbps ainsi qu'une portée de 200 mètres. Mais l'Hiperlan n'a pas eu de succès économique, et a surtout été repris par l'IEEE, notamment le type 2 pour le 802.11a.

Réseau	Débit	Fréquence	Mobilité	Portée	Multiplexage
GSM	9.6Kbps	900, 1800 Mhz	Haute	30 Km	FDMA+TDMA
GPRS	↑ 10Kbps ↓30Kbps	900, 1800 Mhz	Haute	30 Km	FDMA+TDMA
UMTS	144Kbps – 2Mbps théorique	2 GHz	Haute	20 Km	FDD (TDD)
802.11b	11Mbps	2.4 GHz	Basse	50m – 300m	FHSS , DSSS
802.11a	25 Mbps (max. 54Mbps)	5 Ghz	Basse	50m – 300m	OFDM
Hiperlan 2	23 Mbps (max. 54 Mbps)	5 GHz	Basse	200m	OFDM
Bluetooth	1.0 : 1 Mbps 2.0 : 10 Mbps	2.4 Ghz	Basse	qq. dizaines de mètres	TDMA
HomeRF	1.6 – 10 Mbps	2.4 GHz	Basse	50m	TDMA

Tableau 1 : Comparatif des principaux réseaux de mobiles et sans fil

1.2. Accès multi-interfaces et gestion de handover vertical

Comme on peut le voir dans le tableau comparatif précédent, tous ces différents réseaux ambiants forment une hiérarchie de cellules de tailles et de débits différents. En un point géographique, un ou plusieurs de ces réseaux seront disponibles. Ce concept de hiérarchie est apparu clairement pour la première fois dans l'article « the case for wireless overlay network » issu d'une conférence de janvier 1996 [3].
Puisque que fédérer ces différents réseaux apporte un meilleur service à l'utilisateur, celui ci a besoin d'un terminal ayant la capacité d'accéder à plusieurs réseaux : un terminal multi-interfaces, appelé aussi multimode.
Certains travaux [4] proposent de n'avoir qu'une seule interface parmi celles présentes qui communique à un instant donné, afin d'optimiser le temps de latence de changement de réseau. D'autres travaux plus récents traitent de la réception simultanée de données sur des interfaces différentes. Cette approche est donc techniquement possible, mais elle pose certains problèmes comme par exemple la consommation de la batterie. Nous n'envisagerons pas cette possibilité dans ce projet.
On trouve différentes architectures physiques pour réaliser ces terminaux multi-interfaces [5,6]. La plus simple consiste à disposer différentes interfaces d'un mode fixe en parallèle (a). Mais de cette façon, le terminal n'est pas du tout évolutif : il ne peut s'adapter à l'arrivée d'un nouveau réseau. A l'opposé, une seule interface programmable peut se connecter sur n'importe quelle fréquence avec n'importe quel traitement du signal reçu (c). Cette solution est évidemment la plus flexible. Une solution intermédiaire existe : un nombre fixe de récepteur de radio-fréquence est piloté par une unique unité de traitement du signal, configurée par les paramètres nécessaires (b). Ces différentes approches sont encore actuellement discutées selon des critères économiques, de performances et d'adaptabilité. Un critère capital est la consommation électrique, qui est primordiale dans les terminaux portables [7].



(a) Interfaces en parallèle (b) « Software Defined Radio » (c) « Software Radio »

Figure 1. Les différentes architectures multi-interfaces

On peut considérer un changement d'interface comme un nouveau type de handover : le handover vertical (figure 2). Le terminal change de réseaux d'accès sans forcément changer de position géographique comme c'est le cas dans le handover classique, dit horizontal. Ce changement doit s'effectuer de manière transparente à l'utilisateur, sans perturbation de ses services en cours.
Le premier article sur ce sujet [4] expose les différents problèmes à prendre en compte : temps de latence, problème d'énergie lié aux interfaces devant rester active, overhead… Cette étude considère le meilleur réseau comme le plus bas dans la hiérarchie, c'est à dire celui possédant le plus grand débit.
Mais le choix du meilleur réseau est plus complexe et doit prendre en compte d'autres critères. C'est l'objet d'un deuxième article [8] qui propose de différencier l'élection du meilleur réseau et le mécanisme de handover. Après avoir récolté des paramètres sur les différents réseaux disponibles et les préférences de l'utilisateur, le terminal exécute une fonction de coût pour chaque réseau et retient le meilleur. L'utilisateur intervient très peu : il privilégie seulement certains aspects comme le coût ou la performance. Mais les besoins des applications en terme de paramètres de qualité de service ne sont pas encore pris en compte et font l'objet de leurs futurs travaux.

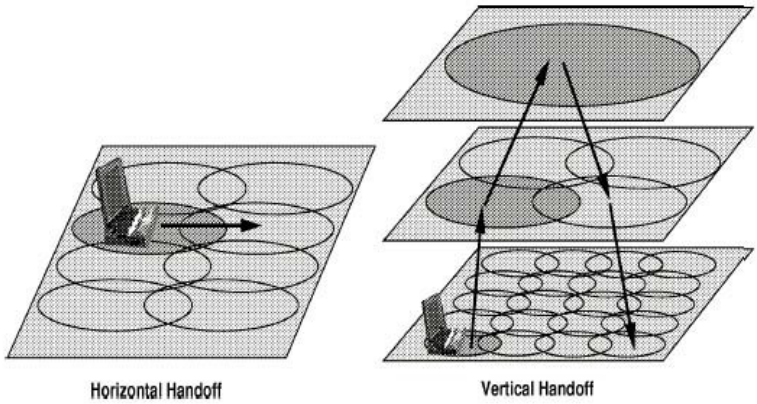


Figure 2. Handover horizontal et handover vertical

La gestion des interfaces multiples est souvent vue comme un cas particulier de Mobile IP [9,10] ou plus généralement comme un problème de mobilité. Au lieu de n'avoir qu'une adresse IP temporaire par terminal, on en a soit plusieurs, soit une seule adresse multicast temporaire. Ainsi un changement d'interface correspond à un changement d'adresse temporaire. Chaque article [4,8,11,12]définit son extension du protocole Mobile IP ou un système proche « Mobile IP-like ». Il est assez difficile de résumé la vision de chaque article vue la complexité et la diversité de celles ci, mais ce rapprochement avec Mobile IP et/ou Cellular IP est commun à tous : cela permet d'effectuer la gestion multi-interfaces et la gestion de la mobilité grâce au même mécanisme. Il s'agit dans un cas de la mobilité intra-système (handover horizontal) et dans l'autre de mobilité inter-système (handover vertical). Mobile IP permet de gérer la macro-mobilité, c'est à dire un changement peut fréquent de cellule tandis que Cellular IP[14], développé par l'IETF pour gérer la mobilité dans les systèmes de 3ème génération, est plus adapté à la micro-mobilité : des cellules plus petites où les handover sont fréquents. Certaines propositions allient ces deux mécanismes pour permettre de gérer au mieux la macro et la micro mobilité [13].

2. Perspectives et propositions pour un accès intelligent aux réseaux ambiants

2.1. Quelques travaux existants

Deux travaux sont particulièrement intéressant pour imaginer une proposition à la problématique du sujet : le contrôle d'admission par politique qui permet de définir des règles dynamiques pour la décision d'allocation de ressources, et le projet I3 qui définit une architecture globale pour l'Internet nouvelle génération. Nous allons les décrire afin de mieux comprendre l'intérêt de chacun.

2.1.1. Le contrôle d'admission par politiques

Avant de présenter le contrôle d'admission par politiques, il convient de définir le concept de réseau basé sur les politiques et le protocole de signalisation associé : COPS.
Common Open Policy Server [15] est un protocole de signalisation développé par l'IETF utilisé pour échanger des requêtes et des réponses entre un serveur de politique et des éléments du réseau. Cet échange est fiable (il utilise le protocole TCP) et les informations sont représentées sous forme d'objets à définir. Le serveur de politique est le point de décision du réseau, décision rendues en se basant sur des politiques. C'est un Policy Decision Point : PDP. Les éléments du réseau, Policy Enforcement Point (PEP), sont des clients qui envoie une requête au PDP lorsqu'ils ont besoin d'une décision pour effectuer une action. Le protocole COPS permet d'échanger ces informations concernant la politique à appliquer de manière bidirectionnelle et dynamique, tandis que les solutions classiques proposent généralement une diffusion de la politique du serveur vers le client.
Les politiques ne sont pas définies et restent totalement ouvertes. Elles peuvent correspondre à des buts complètement différents, s'adaptant selon le type du client ou d'autres paramètres. L'application de base est le contrôle d'admission fondé sur des politiques, principe de COPS-RSVP. Mais d'autres applications sont envisagées comme la configuration automatique de service Diffserv ou la gestion de la mobilité.
Il existe deux politiques de base : l'outsourcing et le provisionning. Le mode outsourcing correspond au modèle du contrôle d'admission : une requête est formulée par un PEP, par exemple une demande d'allocation de ressources, et le PDP décide s'il autorise cette demande en fonction de ses politiques. Le mode provisionning correspond plus à une sorte de configuration automatique du réseau selon des politiques du PDP. Celui ci prépare des règles de politiques qui seront envoyées aux PEP afin qu'ils les appliquent.

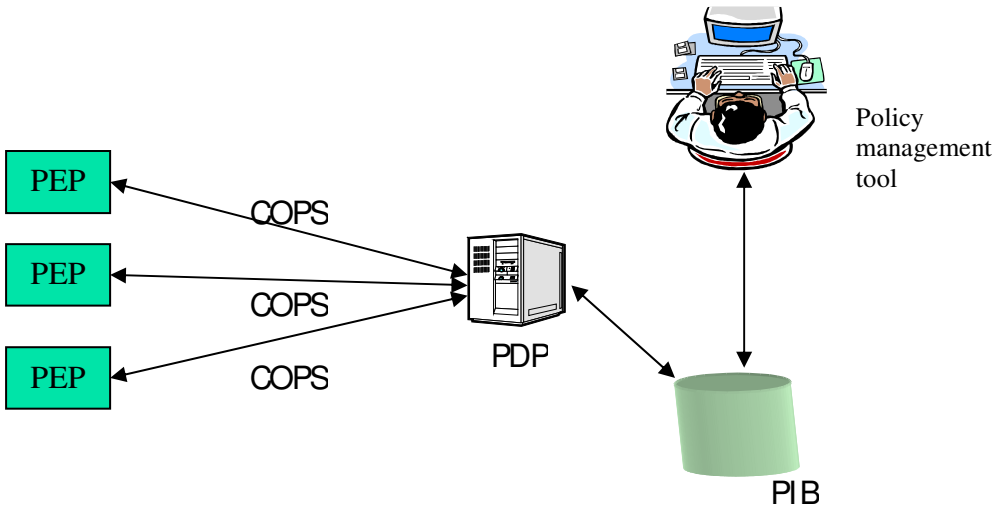


Figure 3. Architecture de base d'un réseau basé sur les politiques

Pour effectuer un contrôle d'admission dynamique et efficace dans un réseau multimédia, il est nécessaire de prendre en compte plusieurs paramètres comme les besoins de l'utilisateur effectuant une requête (paramètres de qualité de service, de sécurité, de mobilité, de coût…) ou l'état des ressources du réseau (monitoring). Un ensemble de règles sur ces paramètres doit ensuite s'appliquer pour décider l'acceptation ou le rejet.
L'architecture de réseau basé sur des politiques est une solution intéressante pour répondre à ce besoin. En effet, COPS permet d'échanger des objets contenant toutes sortes de paramètres pour exprimer les besoins : la requête peut être précise et contenir autant d'informations que nécessaire. Cela correspond au mode outsourcing où l'on utilise une signalisation pour demander l'autorisation d'accéder à des ressources. De plus, le contrôle d'admission nécessite la connaissance globale de l'état des ressources du réseau. L'architecture centralisée de COPS permet au PDP d'avoir cette vue globale puisque toute décision d'allocation passe par lui (ce qui peut aussi poser un autre problème à étudier : la scalabilité).

La proposition de l'IETF pour un contrôle d'admission par politique (RFC 2753) [16] présente une telle architecture. Elle est orientée vers l'utilisation de RSVP et est donc proche de COPS-RSVP[17], mais reste ouverte à d'autres mécanismes de QoS comme Diffserv. Son principe général est le suivant :

- le PEP qui a besoin d'une décision (suite à la demande de l'utilisateur ou à un événement local) crée une requête de contrôle d'admission qui contient les informations nécessaires.
- le PEP détermine ce qui peut être décidé localement (utilisation d'Access list par exemple) et le demande au LPDP.
- ensuite le PEP envoie la requête plus la décision locale au PDP (si le LPDP n'a pas déjà refusé la demande).
- Le PDP applique les politiques adaptées à cette requête et renvoie au PEP la décision finale.
- Dans tous les cas, le PEP doit envoyer un rapport sur la mise en œuvre de cette décision.

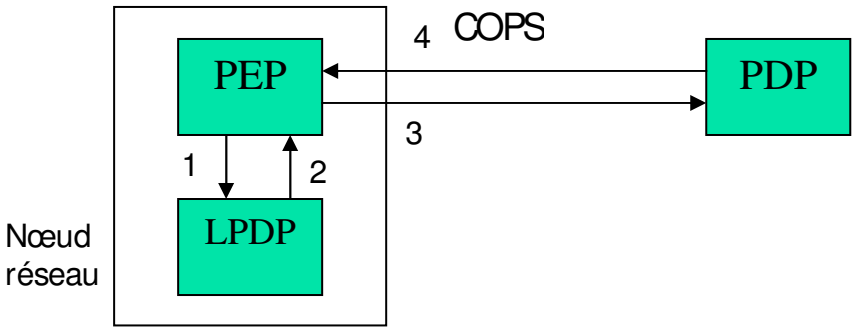


Figure 4. Principe du contrôle d'admission par politiques

Le RFC prévoit la possibilité de préemption pour permettre de modifier l'état actuel en faveur d'une nouvelle demande plus prioritaire. Le mécanisme doit de plus permettre le monitoring des états des politiques : le PDP doit avoir accès à certaines informations pour la facturation. Le mécanisme nécessite un certain niveau de sécurité pour éviter les attaques de type « denial of service » et pour pouvoir avoir la garanti de l'identité des différentes entités.

Mais cette architecture, qui permet à un utilisateur d'entrer sur le réseau, ne prévoit pas la gestion de QoS de bout en bout. Le RFC 2753 prévoit la nécessité d'un dialogue inter-domaine mais précise que ce sujet n'est pas traité dans ce document (seule une proposition d'ajouter un objet à la requête RSVP est introduite).

On peut envisager une variation de cette architecture pour une gestion de bout en bout : l'utilisation de COPS-SLS[18]. Dans cette extension de COPS, l'utilisateur peut émettre lui-même sa requête : le PEP se trouve dans son terminal (ou même dans une carte à puce pour plus de sécurité) et peut négocier plusieurs services simultanément. Ainsi l'utilisateur négocie directement son service avec le PDP, qui à son tour a la possibilité de négocier avec l'utilisateur destinataire ou avec le PDP d'un autre domaine. Le service est ainsi assuré de bout en bout. Cette solution allie le contrôle d'admission et la qualité de service de bout en bout grâce à un point fort de COPS-SLS : pouvoir négocier de manière flexible un service entre utilisateur et PDP ou entres PDP (Figure 5).
Mais, puisque le PEP est dans le terminal portable, des études de performances sont nécessaires pour évaluer l'efficacité de COPS, qui fonctionne sur TCP, sur les liens radio-fréquences. En effet une négociation directe entre le terminal et le PDP implique l'échange d'un volume non négligeable de données sur le lien sans fil, dont les ressources sont limitées. Les perspectives face à ce problème seront plus détaillées dans la présentation de notre proposition.

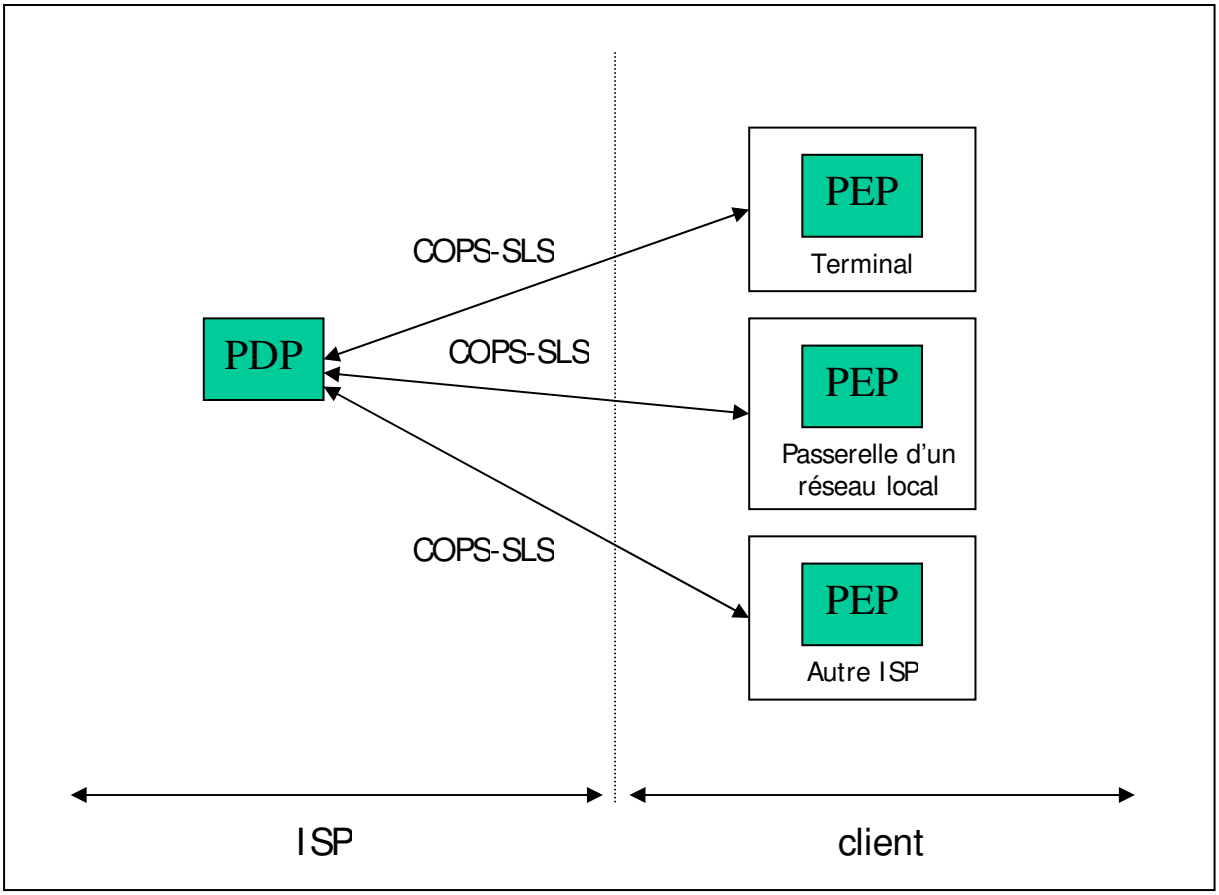


Figure 5. : Le modèle de COPS-SLS

2.1.2. Le projet I3

Ce projet [19] repose lui aussi sur une architecture basée sur les politiques et sur l'introduction d'intelligence dans le réseau. Le concept d'intelligence regroupe trois types d'action : apprendre, communiquer et raisonner. Des agents intelligents [21] sont utilisés pour effectuer ces actions. Ce projet propose de gérer le réseau Internet nouvelle génération grâce à ces agents qui communiqueront avec le protocole COPS et ses extensions. Cette homogénéité est un avantage de cette architecture : le même protocole de signalisation permet la gestion de la qualité de service, de la sécurité et de la mobilité.

La qualité de service est assurée de bout en bout par la négociation de SLS grâce à l'extension COPS-SLS, dont les informations sont représentées dans des Policy Information Base. Une extension de COPS est aussi prévue pour la gestion de la QoS dans les réseaux sans fil.
La gestion de la mobilité, ou plutôt du nomadisme, utilise une autre extension de COPS : COPS-MU. L'architecture est aussi basée sur des agents qui ont chacun un rôle, et qui communiquent avec COPS-MU. Dans le Terminal Home Agent, une association est mise à jour entre l'adresse temporaire du terminal et son adresse d'origine. Ce principe repris à Mobile IP permet de localiser le terminal. De même, on a une association dans le User Home Agent entre l'identificateur de l'utilisateur et l'adresse IP du terminal qu'il utilise. Ainsi l'utilisateur peut utiliser les services auxquels il est abonné à partir de n'importe quel terminal.
La portabilité des services est aussi traitée dans le projet I3 grâce au concept de politique. Le Terminal PEP communique avec le Foreign PDP, le User Home PDP et le Terminal Home PDP. Le but est de minimiser le trafic sur le lien sans fil et donc de faire le maximum de la négociation sur la partie fixe du réseau.
Pour implémenter cette architecture, il faut définir les nouveaux objets COPS : des objets de Mobile IP, des objets pour les paramètres du terminal, du réseau d'accès, des services et de comportement de mobilité. La PIB doit aussi être définie.
L'architecture fonctionnelle du projet comprend quatre grande parties :

- la gestion des contrats : souscription et invocation de SLA. Ces fonctions s'occupent de la souscription des clients et du contrôle d'admission par politique. Le principe du contrôle d'admission par politique présenté précédemment pourrait donc être appliqué, en utilisant COPS-SLS pour acheminer directement les requêtes du client vers le PDP.
- la gestion du réseau : responsable de la configuration du réseau (routage et allocation des ressources selon le routage) en fonction du contrat et de la requête du client.
- La gestion des politiques : stockage et administration des politiques
- Le monitoring : il est essentiel à la qualité de service. Il analyse l'état du réseau et rapporte les résultats au système de gestion des politiques. Le contrôle d'admission a besoin de ces résultats pour prendre des décisions.

La plate-forme permettant la mise en place d'une telle architecture est définie comme un système multi-agent [22] capable de réagir, de gérer et de contrôler la qualité de service offerte.
Les PEP sont des agents réactifs : ils réagissent à leur environnement et appliquent des décisions. Le PDP est un agent cognitif : une certaine intelligence lui est nécessaire pour jouer son rôle de décideur et de juge.
Le projet I3 répond au besoin de qualité de service du réseau Internet de nouvelle génération en utilisant deux concepts communs à notre proposition : un système multi-agent qui communiquent avec le protocole COPS et ses extensions, et dont la gestion générale utilise le concept de politique.

2.2. Architecture retenue

Notre proposition reprend certains éléments traités dans ce document pour permettre un accès intelligent dans les réseaux mobiles de 4ème génération, le but étant de reprendre le contexte existant des réseaux mobiles et sans fil en y ajoutant des entités. Elle utilise une architecture basée sur les politiques (Figure 6) où une certaine intelligence est introduite par une plate-forme multi-agent.
Un PDP (Policy Decision Point) est intégré à chaque réseau d'accès : il a pour rôle de gérer les règles de contrôle d'admission et de les appliquer. Il communique avec un PEP (Policy Enforcement Point) présent dans le terminal mobile qui à la charge de mettre en place concrètement la connexion choisie. Ces deux entités communiquent via le protocole COPS SLS pour échanger les paramètres nécessaires à ce mécanisme d'accès intelligent aux réseaux ambiants.
Ces entités, correspondant au modèle de la RFC 2753 présentée dans la section précédente, sont couplées avec un système multi-agent. Le PDP contient un agent cognitif implémentant l'algorithme de Q-learning. Le PEP contient lui un agent réactif qui a pour rôle la sélection du réseau approprié à partir des paramètres envoyés par les différents PDP et en fonction des préférences de l'utilisateur du terminal.
Ces différentes entités et leur rôle sont repris en détail dans la suite de ce document.

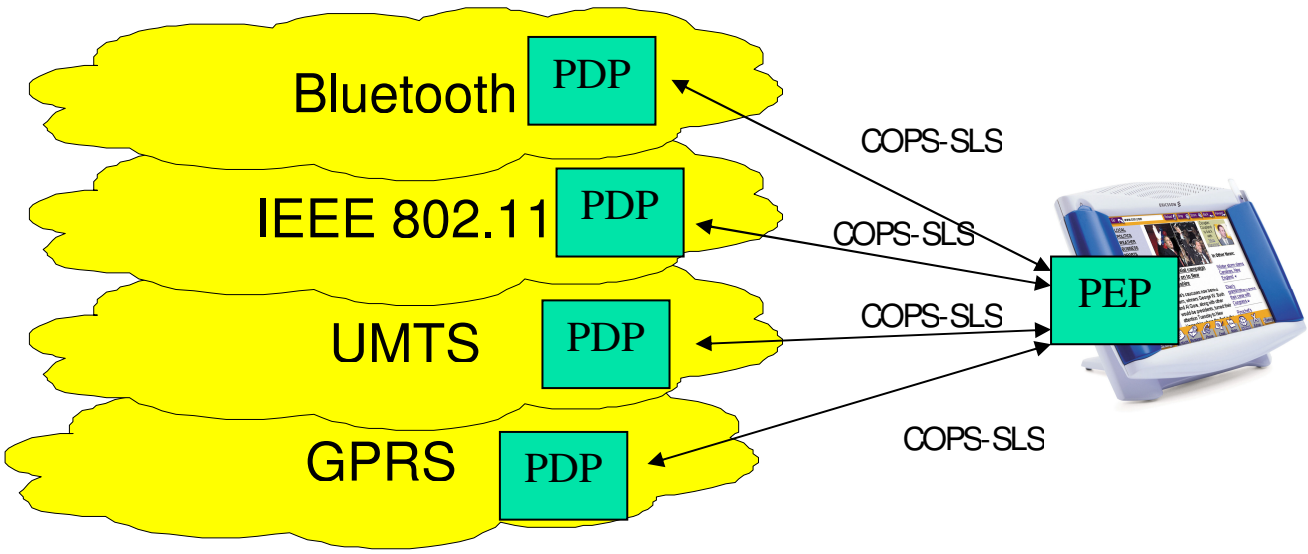


Figure 6. : L'architecture de notre proposition pour un accès intelligent

2.2.1. L'agent PEP

Comme dans le projet I3[19], le PEP se trouve dans le terminal. Il a pour rôle d'effectuer les actions nécessaires à la satisfaction des besoins de l'utilisateur de façon transparente celui-ci. Cette architecture permet au PEP d'interroger directement les PDP et de négocier les différentes possibilités d'accès que lui offre son environnement. Lorsque l'utilisateur veut accéder à un service, son PEP peut donc sélectionner le réseau le plus approprié à ce service grâce à une fonction de coût décrite ci-après. Bien sûr, l'utilisateur a la possibilité d'influencer ce choix par ses préférences (coût, qualité,…).

Le PEP est un agent réactif : il est à l'écoute de son environnement et réagit aux événements provenant de l'utilisateur ou des réseaux ambiants. Ses fonctions sont assez simples et ne nécessitent pas de capacités d'intelligence. Ses principales actions sont :

- écouter les besoins de l'utilisateur
- formuler ces besoins sous forme de requête et négocier avec les PDP des réseaux disponibles
- exécuter une fonction de coût afin de choisir le réseau le plus approprié.

Il est nécessaire d'effectuer certains choix :

- Le premier est de définir ce que le PEP demande à un PDP : quels paramètres il lui transmet pour définir le service demandé. Un compromis entre une définition parfaite du service demandé et une utilisation trop importante de la bande passante du lien radio doit être trouvé. Ce point est détaillé dans la section traitant du moyen de communication entre PDP et PEP.
- Il faut ensuite définir quand et à quels réseaux d'accès le PEP s'adresse. Doit-il « scanner » en permanence les réseaux disponibles dans sa zone ou le faire seulement pour une demande de service ? Est-il intéressant d'utiliser le GPS pour avoir une localisation qui permettrait d'améliorer la connaissance des réseaux disponibles dans une zone ? Cet aspect concerne beaucoup la couche physique : problème d'énergie dépensée, nombre d'interface radio dans un terminal, capacité à avoir plusieurs connexions simultanées pour la découverte de réseaux. Néanmoins les couches supérieures sont concernées par l'architecture et les possibilités de la couche physique. Ces questions ne sont pas traitées dans ce document car elles s'éloignent de la problématique de mon stage, même si les solutions proposées ici dépendent aussi de l'aspect physique. Nous supposerons que le PEP a la possibilité de communiquer avec les différents PDP de sa zone à chaque fois qu'il en a besoin, mais avec un seul à la fois.
- Pour finir, il est nécessaire de choisir une fonction de coût qui reflète au mieux les désirs de l'utilisateur. Les différentes possibilités sont envisagées dans la partie suivante, la sélection du réseau étant une partie majeure du système multi-agent.

La fonction de coût :
La sélection du réseau se fait par l'exécution d'une fonction de coût qui évalue les différents accès possibles, selon les offres reçues des PDP. Une offre contient un ensemble de paramètres la décrivant : les paramètres de QoS ou un code de classe avec un indice de qualité selon la solution de négociation retenue, le prix de cette offre, ainsi qu'éventuellement d'autres paramètres liés à des aspects tels que la mobilité, la stabilité de l'offre… Le but de cette fonction de coût est de transformer plusieurs de paramètres en une valeur unique reflétant l'intérêt que porte l'utilisateur sur cette offre.
Les valeurs de cette fonction sont pondérées par des coefficients reflétant les préférences de l'utilisateur. Ensuite le PEP sélectionne le réseau dont la fonction de coût donne la valeur la plus intéressante.

Une équipe de **Berkeley** [8] propose une telle fonction basée sur trois paramètres : le débit disponible B, la consommation d'énergie E et le coût du réseau C. Il propose alors la formule suivante pour calculer la valeur de la fonction de coût f(n) du réseau n :

$$f(n) = w_b \cdot \ln \frac{1}{B_n} + w_e \cdot \ln E_n + w_c \cdot \ln C_n \qquad \text{avec } \sum w_i = 1$$

La fonction logarithme est utilisée pour mieux refléter l'impact des valeurs des paramètres sur la fonction de coût. Par exemple, si deux réseaux ont un débit important mais que l'un en possède 10 fois plus que l'autre alors pour l'utilisateur ne verra pas forcément un facteur de 10 dans ses préférences. La fonction ln(.) permet alors un lissage et reflète mieux la logique réelle de l'utilisateur.
Les coefficients w_i permettent de pondérer l'importance de chaque paramètre. On peut alors privilégier l'aspect économique, qualitatif,… Si l'on veut avoir le plus de bande passante possible sans considération de prix alors $w_b = 1$ et les autres valent 0.

Mais cette solution n'est pas parfaitement adaptée à nos besoins. Cette fonction ne prend pas en compte tous les paramètres de QoS nécessaire. En plus de la bande passante , il faut par exemple tenir compte du délai ou de la gigue pour un service temps réel. Il est donc indispensable d'intégrer d'autres paramètres, fournis par le PDP.
De plus, il n'est pas forcément aisé de mélanger le coût avec les autres paramètres de base. Notre souci étant de facturer un service en fonction de la QoS fournie, il est plus intéressant de pouvoir mettre face à face un indice global de qualité de service et un indice de prix. Ainsi on peut mettre en place plusieurs fonctions, chacune adaptée à son rôle.
Nous proposons donc d'avoir une fonction par type de service qui a pour rôle d'évaluer la qualité pour un service donné. A partir des paramètres de QoS reçus et éventuellement d'autres paramètres tels que l'énergie nécessaire, la fonction correspondant au type de service demandé par l'utilisateur rend un indice de qualité globale de ce service. Ainsi la fonction, appelée « fonction de qualité », permet d'évaluer la qualité d'un réseau pour un type de service donné en rendant un indice de qualité.
Pour obtenir un indice de prix, deux possibilités peuvent être envisagées. Soit tous les fournisseurs d'accès facturent leur service selon la même unité, par exemple le temps d'utilisation d'un service, soit il faut mettre une « fonction de prix » qui rend les prix comparables. Nous faisons l'hypothèse qu'une comparaison des prix des services sera possible par une méthode ou une autre.
Il est ensuite possible de prendre en compte de manière équitable la qualité et le coût. Une fonction de coût « finale » ne prenant en compte que l'indice de qualité et l'indice de prix est utilisé.

Dans ce but, nous nous sommes intéressés aux travaux du domaine du commerce électronique, notamment à ceux du centre de recherche IBM[24,20], qui s'intéressent à une problématique similaire. Différents vendeurs proposent des services (ou produit) à un niveau de qualité fixe, mais qui peut être différent d'un vendeur à l'autre. Ils proposent une « utility function » qui a le même but que notre fonction de coût : permettre à un agent acheteur de prendre une décision et de sélectionner le meilleur produit en fonction de sa qualité et de son coût.

$$u = [\alpha(q - \overline{q}) + (1 - \alpha)(\overline{p} - p)]\Theta(\overline{p} - p)\Theta(q - \overline{q})$$

avec $\begin{cases} \Theta(x) = 1 \text{ si } x > 0 \\ \Theta(x) = 0 \text{ sinon} \end{cases}$

Cette fonction est exploitable dans notre architecture. Le paramètre α, compris entre 0 et 1, permet de spécifier la préférence de l'utilisateur entre la qualité et le coût d'un service. $\overline{q}$ représente le niveau de qualité minimal que l'utilisateur désire et $\overline{p}$ le prix maximum qu'il veut payer. Par exemple, un utilisateur avec $\alpha = 1$ est seulement attentif à la qualité d'un service et est donc indifférent à son prix. Les deux fonctions indicatrices $\Theta(x)$ permettent de rendre nul le résultat de la fonction dans les cas inintéressants pour l'utilisateur comme un prix supérieur au prix maximum fixé.

Remarques :

- Attention, dans le cas où l'on désirerait échanger des indices de qualité q afin de les comparer, un problème d'homogénéité se pose : le calcul de q est effectué par une fonction qui doit être la même chez chaque « acteur ». On peut imaginer que chacun possède les mêmes fonctions de qualité mais cela nécessite une coordination…Par contre, si on ne se sert que localement des valeurs q, alors chacun peut utiliser sa façon de juger la qualité. Dans ce cas l'indice est calculé localement, en fonction des paramètres envoyés par le PDP.)

- Ce type de fonction de coût est utilisé dans notre proposition, en utilisant des paramètres différents encore à définir. Plus on utilise de paramètres et plus on peut affiner le système en répondant très précisément aux besoins utilisateurs. Mais la multiplication des paramètres a une limite : c'est le PDP qui doit décider de ceux ci et la complexité de l'algorithme augmente avec le nombre de paramètres.

2.2.2. L'agent PDP :

Chaque réseau ambiant possède un PDP qui décide du service que le réseau peut offrir à un utilisateur en fonction de politiques, des paramètres issus du monitoring du réseau.
L'agent PDP est un agent cognitif : il doit prendre des décisions en faisant appel à des mécanismes plus ou moins complexes et faire preuve de capacité d'apprentissage pour s'adapter à un environnement dynamique.
Il contrôle l'accès à son réseau par un contrôle d'admission par politique, dont le modèle suit celui de la RFC 2753, mais où le protocole de négociation est COPS-SLS. Celui-ci présente certains avantages (protocole flexible, négociation configurable et gestion de bout en bout). Grâce à ce protocole, le PDP a la possibilité d'accepter un appel ou de proposer une dégradation de service au PEP si le réseau est congestionné (dans ce cas la facturation devra en prendre compte). Cette capacité de négociation est très intéressante dans le contexte d'un réseau paquet multiclasse où chaque classe peut accepter une certaine dégradation de service. On a donc un contrôle d'admission dynamique qui ne se contente pas d'accepter ou de refuser une requête : soit il propose d'accepter la demande en proposant de fournir un service avec une certaine qualité et un certain prix, soit il refuse s'il n'a pas les ressources nécessaires.

Mais cette flexibilité est à double tranchant : elle permet une facturation au plus juste et une meilleure concurrence, mais elle induit aussi un phénomène de « guerre des prix » connue dans le domaine économique. Les différents PDP se font concurrence et les prix deviennent alors instables : les coûts des services sont à chaque fois un peu inférieurs pour finalement atteindre une valeur critique. Les prix sont alors remis à une valeur initiale et le cycle recommence. Ce phénomène est appelé dans la littérature « cycle de la guerre des prix ».

Il est alors intéressant de mettre en place un algorithme de contrôle d'admission distribué entre les différents réseaux comme l'apprentissage par renforcement ou Q-learning [20,23]. Cet algorithme, qui sera distribué sur les PDP, a déjà été étudié pour réaliser un contrôle d'admission et permet de réduire la probabilité de blocage à partir d'une simple spécification des préférences. Dans ce cadre, il présente de bonnes capacités d'adaptation et de généralisation pour un environnement dynamique. De plus, il ne demande que peu de ressources pour le calcul.

Mais dans notre cas il s'agit de plusieurs agents ou PDP implémentant le Q-learning. Or s'il est garantit que le Q-learning trouve la politique optimale dans un environnement stationnaire, il n'existe pas de preuve d'une convergence vers la politique optimale dans un environnement où plusieurs éléments implémentent le Q-learning et donc où l'environnement est non stationnaire et dépend de l'historique. Nous allons donc nous intéresser dans la suite du document à un contexte où seul deux fournisseurs d'accès implémentent le Q-learning. Une fois cet objectif spécifié et simulé, il sera alors intéressant de discuter la généralisation de cette architecture.

2.2.3. La communication entre agents

L'utilisation de COPS-SLS pour la communication entre le terminal et un réseau nous paraît intéressante pour les différentes raisons évoquées auparavant.
Mais son utilisation implique un problème d'occupation de la bande passante des réseaux ambiants. En effet, pour permettre une négociation, de nombreux objets doivent être échangés. La phase de configuration de COPS-SLS et les objets décrivant le service constituent un ensemble de données assez lourd sur un lien sans-fil, et encore plus dans ce contexte où chaque terminal communique plusieurs fois avec plusieurs PDP.
Ce problème n'est pas spécialement celui de COPS-SLS, mais plutôt celui de la négociation entre le terminal et le réseau. Plus celle ci est précise et flexible et plus elle sera lourde.
On peut donc envisager deux adaptations pour optimiser l'utilisation des ressources radios :

- adapter COPS-SLS pour alléger le nombre de bit nécessaire à la négociation. On aurait alors un protocole de signalisation adapté au lien sans fil.
- spécifier un nombre limité de paramètres de négociation. Il serait par exemple intéressant de définir un certain nombre de classes de service communes et adaptées à un type de service : téléphonie, visioconférence, téléchargement, best effort… A chaque classe on attribue un prix de base, modifiable selon le niveau de qualité atteint. Ainsi, les paramètres de négociation peuvent se réduire à la qualité globale et au prix.

Mais pour avoir un système réparti, où chaque fournisseur d'accès gère indépendamment ses ressources, le PEP doit transmettre un maximum de paramètres plutôt que d'avoir une classification commune des services. Ainsi le PDP interprète comme il veut ceux-ci, et gère ses ressources selon ses désirs. Il a la possibilité de créer autant de classe de service qu'il le désire en interne, chacune associée à un nombre de ressources occupées. De même lorsque les PDP proposent une offre, le PEP concerné pourra comparer avec plus de précision si les offres sont détaillées. Cela correspond plus à notre vision de vendeurs concurrents et indépendants.
L'idée d'une classification commune des services est donc intéressante pour optimiser l'utilisation du lien radio mais est peu compatible avec les principes de la problématique : elle empêche la flexibilité du système et l'indépendance des fournisseurs. On peut par contre imaginer que chaque fournisseur envoie une fois pour toutes les différents types de services qu'il propose au terminal. Ainsi le mapping entre les paramètres et la classe de service correspondante est effectuée par le PEP et on optimise alors le nombre de bits nécessaire à la négociation. Ainsi le fournisseur garde une certaine flexibilité pour élaborer ses différentes classes.

La communication entre les différents PDP ne nécessite pas forcément COPS-SLS et pourrait se faire en utilisant un langage de communication entre agent comme par exemple KQML.
Elle est nécessaire pour pouvoir mettre en place un algorithme distribué de contrôle d'admission décrit dans la partie suivante.

2.2.4. Scénario de fonctionnement de l'architecture :

Le contrôle d'admission est donc basé sur un système multi-agent constitués des éléments décrits précédemment et dont le scénario de fonctionnement est le suivant :

- L'utilisateur demande un service à partir de son terminal. Le PEP du terminal reçoit cette requête et détermine à quelle classe de service elle correspond. Ces classes sont à déterminer en fonction de la finesse du contrôle d'admission que l'on veut effectuer et de la complexité que cela entraîne.
- Le PEP communique à chaque PDP des réseaux accessibles la classe de service désirée grâce à COPS-SLS ou une version allégée.
- Chaque PDP exécute l'algorithme de contrôle d'admission selon la méthode décrite dans la partie concernée.
- Les PDP renvoient ensuite leur proposition de service : la classe de service ainsi que le prix et la qualité de ce service. La réponse peut être négative si les ressources d'un réseau sont insuffisantes.
- Le PEP doit alors faire son choix parmi ces différentes offres. Il exécute alors une fonction de coût sur chacune des offres et le réseau élu sera celui obtenant la valeur la plus adaptée (minimum ou maximum selon la façon de procéder). L'utilisateur pourra alors bénéficier du service sur le réseau qui lui convient le mieux.

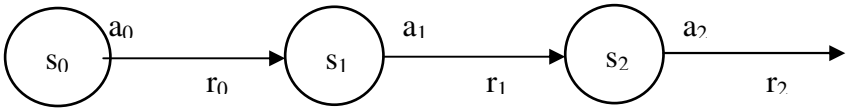
3. Le contrôle d'admission

Dans cette partie, nous présenterons d'abord généralement l'algorithme de Q-learning et son utilisation pour effectuer un contrôle d'admission classique pour ensuite proposer une manière d'intégrer celui-ci dans le PDP pour permettre un contrôle d'admission intégrant le facteur prix. Nous nous intéresserons à une architecture où deux agents PDP sont en concurrence dans le but de spécifier au mieux le système.

3.1. Q-learning et contrôle d'admission

3.1.1. Apprentissage par renforcement : le Q-learning

Le Q-learning[23] est une des méthodes d'apprentissage par renforcement. Il consiste à apprendre par l'expérience pour déterminer l'action à effectuer en fonction de l'état actuel. L'agent implémentant le Q-learning interagit directement avec son environnement.
Le problème est formulé comme un SMDP (Semi Markov Decision Process) :



Les états s_t décrivent l'état du système par un ou plusieurs paramètres constituant un ensemble d'états fini S. L'agent implémentant le Q-learning peut effectuer une action parmi un ensemble fini A. En appliquant une action a_t à un état s_t, on arrive à un autre état, s_t'. En passant d'un état s_t à un état s_t', par une action a_t, l'agent reçoit instantanément un certain revenu r_t. Ces revenus r_t sont définis statiquement en fonction de l'état et de l'action et reflètent les préférences que l'on a pour une action dans un certain contexte.

Le but de l'agent est d'apprendre une politique $\pi : S \rightarrow A$ permettant de choisir l'action à accomplir dans l'ensemble A.

Le principe du Q-learning est le suivant :

- on associe une valeur Q(s,a), la Q-valeur, à un état s et une action a. Cette valeur est l'espérance de gain évaluant l'intérêt de l'action a quand on est dans l'état s.
  Elle est égale à la somme de l'espérance de gain immédiat r(s,a) et des espérances de gain des autres états, atteints en menant l'action a, pondérées par la probabilité d'atteindre chacun d'eux. (On considèrera plutôt l'espérance de gain maximum parmi ces différentes espérances.)
- cette fonction Q, appelée Q-Function, est apprise au fur et à mesure du déroulement de l'algorithme selon la fonction récursive :

$$Q_{t+1} = \begin{cases} Q_t(s,a) + \alpha_t \left[r_t + \gamma \max_b \left[Q_t(s'_t, b) \right] - Q_t(s,a) \right], & \text{Si } s=s_t \text{ et } a=a_t \\ Q_t(s,a), & \text{Sinon} \end{cases}$$

Le facteur $\gamma \in [0,1]$ est le taux d'apprentissage et $\alpha_t = \frac{1}{1+visit_t(s_t,a_t)}$ est le pas d'apprentissage, où $visit_t(s_t,a_t)$ est le nombre de fois où l'action a_t a été choisie ou visitée lorsque le système est dans l'état s_t.

- lorsque l'on est dans l'état s, la meilleure politique sera d'effectuer l'action a pour laquelle la valeur Q(s,a) sera maximale.

3.1.2. Utilisation du Q-learning pour un contrôle d'admission classique

Il existe différentes manières d'implémenter cet algorithme pour réaliser un contrôle d'admission. Dans ces solutions l'état s exprime l'état du système, par exemple le nombre de connexions en cours, ainsi que l'événement demandant une prise de décision. Les revenus ont une valeur fixe correspondant à l'importance de l'acceptation d'un type d'appel ou d'un autre (en fonction de la classe de service de l'appel et de l'événement « arrivée » : nouvel appel ou handoff). On pourra par exemple donner une valeur de 50 à un appel en handover et seulement 5 pour un nouvel appel. A partir de cette simple spécification des préférences, une politique optimale est exprimée par la Q-fonction (la politique optimale est apprise en un temps infini : à partir d'un certain temps, une politique est efficace et exploitable est apprise).
Les différences se font ensuite dans l'implémentation de la manière de calculer cette Q-fonction. Soit on utilise un tableau contenant tous les couples (s,a) possibles et on a alors un calcul exact pour ces états, soit on met en place une technique d'approximation de cette Q-fonction, à l'aide des réseaux de neurones ou des arbres de régression[20].

L'algorithme TQ-CAC (Table Q-learning Call Admission Control) utilise un tableau représentant sur chaque ligne une valeur différente du couple (s,a) ainsi que la valeur Q(s,a). Après un certain temps, le tableau est « appris » et on peut alors l'utiliser pour choisir une action a lorsqu'on est dans un état s : la plus grande valeur de Q(s,a) indique quelle action à mener (acceptation ou refus). Nous reprendrons TQ-CAC pour nos simulations.
NQ-CAC (Neuron Q-learning CAC) fait appel aux réseaux de neurones et donne des approximations de Q(s,a), tous comme la technique d'arbres de régression.

3.2. Intégration du Q-learning dans notre contexte :

Ce paragraphe se réfère directement à l'algorithme de Q-learning et plus particulièrement au TQ-CAC, tous deux rapidement décrits précédemment. Il traite de l'intégration du Q-learning dans les agents ou PDP, qui sont dans un premier temps au nombre de deux. En effet, dans notre contexte, plusieurs agents doivent implémenter l'apprentissage par renforcement. Nous commençons par définir un mécanisme pour deux agents afin de simplifier le problème.
Chaque PDP possède le même mécanisme implémentant TQ-CAC pour pratiquer un contrôle d'admission dans son réseau. Afin de permettre à cet algorithme d'apprentissage de lutter contre la « guerre des prix », l'agent ou PDP doit aussi prendre en compte l'autre . Il pourra ainsi en tenir compte pour évaluer l'intérêt d'une action, c'est à dire calculer la table des valeurs Q(s,a).
Le PDP doit choisir une action en fonction de l'état et de la politique définit par la « Q-Function ». Nous proposons de mettre dans le Policy Repository du PDP les informations d'états s et les actions a, sous la forme : si s alors a.
Pour un état donné, on aura alors plusieurs règles de ce type avec le même état s et des actions différentes. Pour choisir la bonne action, il faut associer à chaque règle sa valeur Q(s,a). On a alors deux possibilités :

- intégrer cette valeur dans le Policy Repository dès que la Q-Function est apprise et la mettre à jour régulièrement.
- Insérer un lien vers la table de TQ-CAC ou directement vers les valeurs Q(s,a) correspondant à la règle.

Dans les deux cas, le calcul de la Q-Function de TQ-CAC est fait en dehors du Policy Repository et les valeurs d'évaluations du Q-learning sont prises en compte par le PDP pour le choix de l'action à accomplir.
Il faut donc définir les paramètres représentant un état, la nature des différentes actions à mener ainsi que l'algorithme de Q-learning dans ce contexte.

Afin de réduire le nombre de paramètres, il est important de commencer avec un contexte simplifié : nous nous intéresserons donc à l'aspect « détermination dynamique du prix d'un service ». En supposant un environnement de plusieurs vendeurs proposant des services de catégorie identique, comme c'est souvent le cas sur marché de consommation importante comme celui-ci, le prix est alors l'élément déterminant pour un terminal cherchant un accès. Dans un premier temps, il n'est donc pas nécessaire de représenter l'état des ressources, ou d'autres paramètres classiques de contrôle d'admission. De plus, le PDP peut d'abord vérifier qu'il dispose d'assez de ressource pour accepter la classe de service demandée. Si c'est le cas, il applique ensuite le mécanisme décrit ici. L'utilisation du Q-learning vise à gérer la décision de choix du prix d'un service plutôt que la gestion des ressources. Il est cependant envisageable de l'intégrer ultérieurement. Nous supposons aussi dans un premier temps que les PDP proposent un niveau de qualité constant et donc que seul le prix varie.
Nous allons donc définir la mise en pratique de l'algorithme de Q-learning, c'est à dire les états, les actions ainsi que le principal facteur d'une convergence vers une bonne politique : le revenu d'une action.

Les états :

Les états sont représentés par plusieurs paramètres :

- la classe de service demandée C représentée par un nombre entier x. Le nombre de classe n'est pas limité, mais il influe sur la taille mémoire nécessaire pour représenter la table de Q-learning.
- le dernier prix pratiqué pour cette classe de service, afin d'avoir une base pour fixer son prix. Ce prix est diffuser par le terminal lorsqu'il décide de choisir un PDP et représente donc un prix réaliste puisque accepté par un client.

Après simulation, nous nous sommes rendu compte de l'indépendance de chaque les unes par rapport aux autres. Nos simulations ont donc ensuite été faites avec un paramètre : le dernier prix pratiqué.
D'autres paramètres pourront être ensuite ajoutés pour permettre des améliorations : par exemple la congestion du réseau pour adapter la qualité du service ou la distinction de demande de ressources pour un nouvel appel ou pour un handoff.

Les actions :
D'après les hypothèses faites, les actions à mener concernent donc seulement le prix du service à proposer. Ainsi pour chaque état (une classe de service et un prix pratiqué), différentes valeurs de prix sont disponibles. Il s'agit de savoir quel prix pratiqué dans une situation donné. Le nombre d'action est donc fonction de la granularité de l'indice de prix. Plus la granularité est fine et plus le nombre de prix à proposer, donc d'actions, est grand.
Pour représenter le prix par une variable continue, l'utilisation d'une table pour l'apprentissage n'est plus possible et des solutions d'approximation seront intéressantes (réseaux de neurones…).
Une autre extension serait de permettre une qualité variable, adaptée à la demande. En effet, si à un instant t les utilisateurs privilégie le prix plutôt que la qualité du service alors il serait intéressant d'intégrer des actions représentant une baisse de la qualité, et donc du prix proposé.

Les revenus :
La détermination des revenus est un point capital pour converger vers une bonne politique. Puisque nous nous intéressons à la gestion des prix des services, ils doivent refléter le profit du fournisseur de service. Le revenu immédiat, pour un état s et une action a, correspond au prix proposé dans l'action moins le coût d'un service de cette classe, qui est une valeur fixe puisque nous avons supposé une qualité constante.
On a donc :
$r_{immédiat} = p_a - c(x)$

Dans le cadre d'un seul agent « apprenti » prenant une décision, un revenu immédiat suffit à définir r dans la formule de Q-learning. Mais quand plusieurs entités doivent prendre une décision, comme dans notre contexte de guerre des prix, alors il faut redéfinir le revenu. En effet, pour être significatif et efficace, le revenu doit symboliser le but à atteindre, spécifier les préférences. Dans notre cas, pour éviter le phénomène de guerre des prix, le revenu associé à une action a et un état s doit correspondre au profit de l'ensemble des entités proposant des prix pour des services. Puisque nous avons deux agents, le revenu est le revenu immédiat obtenu suite à une action a, ajouté au revenu induit par l'action « réponse » de l'autre agent. L'état s' correspond donc à l'état obtenu en partant de l'état s et en appliquant l'action a ainsi que l'action réponse de l'autre entité. Le but des simulations sera justement de définir un revenu permettant de minimiser la guerre des prix.

Utilisation de l'algorithme

Il existe deux phases dans l'utilisation de l'algorithme : l'apprentissage de la table et l'exploitation de la table. Comme il a été dit dans la section architecture, l'apprentissage de la table est fait à part, et non directement dans le PDP. Cet apprentissage, c'est à dire le calcul de la Q-fonction Q(s,a), est réalisé grâce à l'algorithme TQ-CAC : toutes les valeurs des couples (s,a) sont représentées dans les lignes d'un tableau. Chaque ligne possède la Q-valeur associée, initialisée à la valeur de revenu immédiat. Ensuite pendant l'apprentissage, chaque est visitée pour calculer la nouvelle Q-valeur selon la formule récursive de Q-learning décrite précédemment.
Il est important de visiter toutes les lignes un grand nombre de fois pour explorer toutes les possibilités et converger vers une bonne politique. Une sélection aléatoire uniforme des lignes est par exemple envisageable. Après avoir obtenu aléatoirement une ligne à explorer, on explore la ligne correspondant à l'état s', obtenue par l'effet d'une action a choisie pour sa valeur Q maximum. On répète ainsi un nombre de fois fixe l'exploration de la « chaîne » d'états, pour ensuite tirer à nouveau une autre ligne aléatoirement.

Une fois la période d'apprentissage passée, il est possible de continuer pendant l'exploitation de la Q-Function. L'apprentissage peut être fait en continu et exploité les informations du système (dernier prix pratiqué) : ainsi la politique s'adapte aux variations de l'environnement (bien que sous les hypothèses prises, l'intérêt principal est la lutte contre le cycle de guerre des prix). En ajoutant la gestion du prix et de la qualité en fonction des ressources, l'adaptabilité devient de plus en plus intéressante.

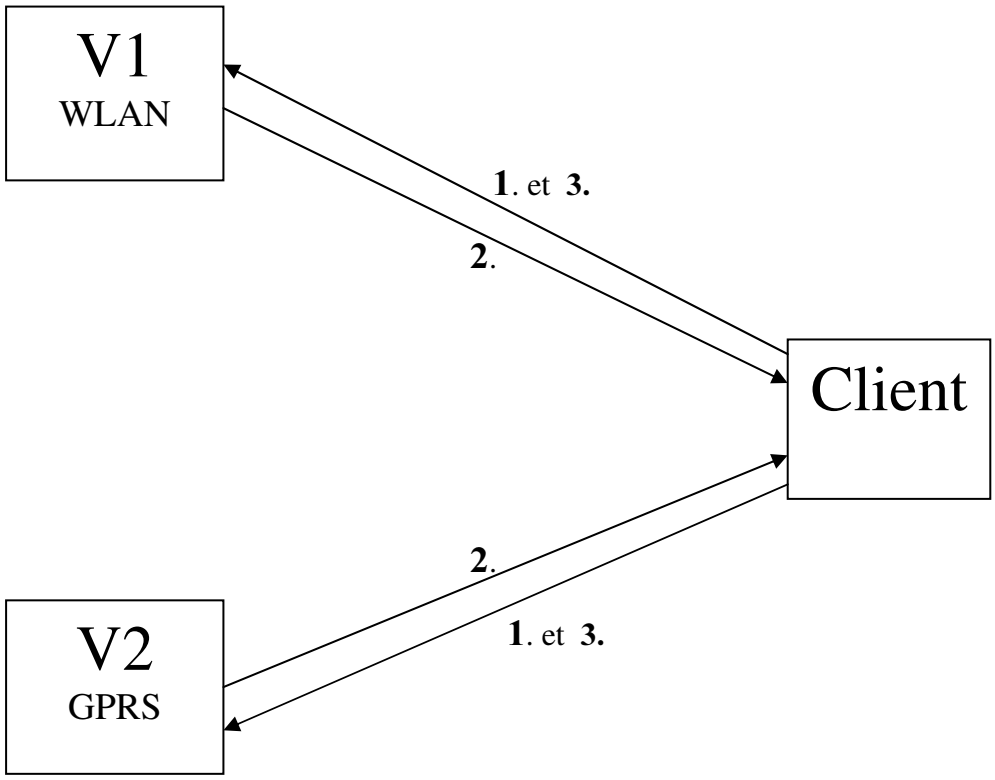
Le résultat de cet apprentissage est ensuite exploité dans le PDP. A une règle du type « si s alors a » est associée la Q-valeur Q(s,a). Comme il a été dit dans la partie architecture, deux possibilités sont envisageables pour cette association : mettre à jour à intervalle régulier les Q-valeurs ou insérer un pointeur dans chaque règle vers la bonne Q-valeur. L'exploitation de cette valeur est très simple : pour un état s donné (suite à une demande de service, on est dans un état s) le PDP applique la règle qui a la plus grande Q-valeur parmi les règles dont la condition est aussi s.

Cet algorithme d'auto apprentissage peut être utiliser pour de nombreux aspects, paramètres du contrôle d'admission. Comme nous l'avons plusieurs précisé dans ce document, nous nous sommes focalisé sur l'aspect « pricing » de notre contexte : la concurrence engendre une guerre des prix cyclique qui entraîne une instabilité des prix. Cela ne présente pas d'intérêt que ce soit pour le vendeur ou pour l'utilisateur. Nous avons donc simulé ce contexte, décrit précédemment, afin de comprendre comment le Q-learning peut minimiser cette guerre des prix, tout en maximisant le profit de chaque vendeur. La partie suivante présente rapidement la simulation, pour ensuite dégager quelques résultats.

4. Simulations et résultats

4.1. Environnement de simulation

Les simulations ont été effectuées en reproduisant notre contexte de système multi-agent : deux vendeurs et une multitude de client, l'ensemble s'échangeant des messages. Pour cela, nous avons utilisé Oris[26,27], un langage permettant de simuler un système multi-agent. Oris a été développée par Fabrice Harrouet comme thème de sa thèse sur la réalité virtuelle. Comme dans le cas de la réalité virtuelle, dans les systèmes multi-agents, il s'agit de faire agir des entités indépendantes selon des lois qui leur soient propres.
Les agents sont donc décrits, dans un langage proche du C++, comme des objets « actifs » possédant des attributs et des méthodes (voir Annexe). Ils sont actifs puisque qu'une fois lancés, il tourne tous en parallèle, gérés par l'ordonnanceur interne de Oris, et exécutent leurs tâches. L'utilisation de ce langage est intéressant pour sa simplicité de programmation ainsi que pour la réutilisation du code : une fois le contexte construit, il est aisément modifiable et peut simuler de multiples situations différentes.
On a alors le système suivant :



1. le client effectue sa demande de service auprès de chaque réseau.
2. chaque vendeur propose un prix au client.
3. le client choisi le réseau qui lui convient le mieux et informe chaque réseau du prix du service qu'il a choisi.

4.2. Simulations et résultats

Nous avons simulé trois cas : un vendeur implémentant le Q-learning avec un vendeur appliquant une politique plus classique, puis le cas de deux vendeurs « Q-learner » afin de les comparer avec un modèle sans Q-learning. Chaque vendeur implémentant le Q-learning possède donc une table de ce type, beaucoup plus grande en réalité (les valeurs sont à titre d'exemple) :

Etat s p1 : dernier prix pratiqué	Action a p2 : Prix à proposer	Q(s,a)
p=0.25	…	…
p=0.25	0.98	0.6235
p=0.25	0.99	0.865542
p=0.25	1	0.54256
p=0.26	0.5	0.32351
p=0.26	0.51	0.32659
p=0.26	…	…

Le déroulement d'une simulation est le suivant:

- Période d'apprentissage des tables (dans le cas d'agent(s) Q-learner(s))
- Exécution du client qui envoie des requêtes de services aux vendeurs (1000 en général)
- Analyse des résultats générés dans un fichier sous la forme de la section 4.2.1.

Nous allons maintenant présenter les différents types de simulation ainsi que leurs résultats.

4.2.1. Sans Q-learning

Dans cette simulation, qui sert de base de comparaison de résultat mais aussi de vérification du fonctionnement du système multi-agent écrit sous Oris, deux vendeurs sont en concurrence pour répondre à un grand nombre de requête client. Pour des raisons pratiques, il est plus simple de ne lancer qu'un seul client qui effectue les requêtes les unes après les autres. Cela n' a pas d'incidence sur la simulation puisque le temps d'inter arrivées des requêtes n'influe pas sur la détermination du prix dans notre contexte. Ce ne serait pas le cas si la congestion était prise en compte par exemple. La partie client de la simulation est commune aux trois cas simulés (sauf bien sûr en ce qui concerne la gestion des résultats de simulation).

Les deux vendeurs ont donc la même politique, dite classique :

- Si j'ai vendu lors de la dernière requête, je garde le même prix.
- Si je n'ai pas vendu, je baisse mon prix de quelques centièmes (rappelons que les prix s'échelonne de 0 à 1 par as de 0.01).
- Je remonte mon prix à 1 lorsque j

Avec cette politique, la guerre des prix est maximale. Les vendeurs baissent leur prix à tour de rôle jusqu'au prix coutant (fixé à 0.5). Ils se partagent ainsi équitablement les clients et obtiennent un profit moyen identique à 0.25 : donc un prix moyen de ventes de 0.75, ce qui est logique puisque les prix oscillent cycliquement entre 0.5 et 1.

On obtient les résultats suivants :

```
**************************************************************************
SANS QLEARNING : Guerre des prix totale
nb client :1000
profit total : 249.15
profit du vendeur1 : 124.79
profit du vendeur2 : 124.36
nb_client du vendeur1 : 499
nb_client du vendeur2 : 500
 profit moyen par client du vendeur 1 : 0.250080160321
 profit moyen par client du vendeur 2 : 0.24872
**************************************************************************
```

4.2.2. Un agent Q-learner

Dans cette simulation, nous avons observé la réaction d'un agent vendeur implémentant le Q-learning face à une politique plus classique, comme précédemment. Le but est d'avoir un profit moyen par client supérieur à celui de la politique classique.
Pour cela, une première version a été simulée, où le revenu était égal à p2 – prix_coutant(vendeur) et ce quelles que soient les valeurs de p1 et de p2. Mais dans ce cas l'algorithme ne converge pas : les prix proposés sont incohérents et au final le vendeur ne remporte presque aucun « duel » face à la politique classique. Nous avons donc affiné la fonction de revenu, afin d'obtenir une meilleure politique.

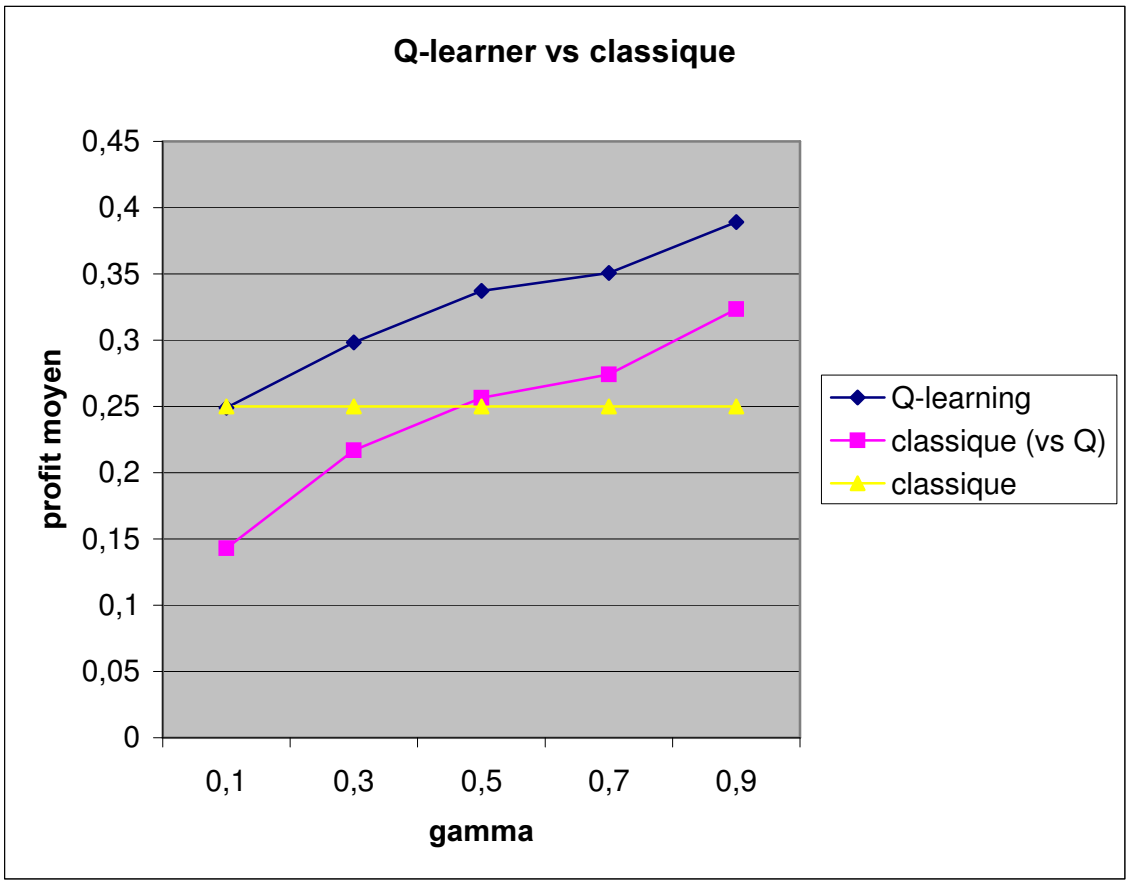
La fonction de revenu est alors définie en fonction de la politique « classique » du vendeur opposé (voir en annexe : fichier source de cette simulation). On a la fonction suivante :

- Si p2>p1, alors on est au-dessus du prix du vendeur classique donc revenu=0
- Si p2<=p1, alors le revenu est égal au revenu immédiat classique :
  Revenu= p2 – prix_coutant(vendeur)

De plus, on considère que si le vendeur classique vend trois fois de suite au même prix, il voudra maximiser ses revenus et remontera son prix à 1 (il pourrait être le seul à vendre pendant un instant et donc vouloir en profiter).

Dans ce cas les résultats sont satisfaisants : le vendeur Q-learner prendra l'initiative de remonter son prix avant d'atteindre le prix coutant. Une guerre de prix existera toujours, mais dans un intervalle plus réduit, ce qui permet d'augmenter le profit moyen de chaque vendeur. La guerre continue malgré tout puisque le vendeur classique baisse toujours ses prix : le vendeur Q-learner doit s'adapter et rester concurrentiel pour continuer à vendre ses services. Mais il abandonne plus vite la guerre des prix, quand le revenu d'une vente deviennent insuffisant. Cet événement n'est pas programmé, mais découle directement de la période d'apprentissage.

On obtient les résultats suivants :



On peut considérer que la modification de la fonction de revenu que l'on a effectuée « force » le vendeur Q-learner à adopter une politique ressemblant à la politique classique. Il baisse toujours ses prix par rapport au dernier prix pratiqué p1 : on a une guerre des prix plus courte qui optimise le profit moyen de chaque vendeur. Mais intéressons-nous au cas de 2 agents Q-learner.

4.2.3. Deux agents Q-learner

Dans le cas de deux agents implémentant le Q-learning, la période d'apprentissage s'applique à deux tables. On applique alors l'algorithme d'apprentissage sur chaque table, à tour de rôle. Comme précédemment, on sélectionne donc une ligne aléatoirement et on déroule les n états s', puis on effectue la même opération dans l'autre table.
La fonction de revenu de la section précédente n'est pas vraiment idéale dans le cas de deux agents Q-learner. Puisque la fonction de revenu ne dépend que de p1 et p2 de la table de l'agent, chaque agent Q-learner aura le même comportement. On aura alors une guerre des prix entre les agents, de la même façon que la solution sans Q-learning, mais avec une amplitude moindre. Concrètement, avec un prix coutant de 0.5, on aura une oscillation des prix dans un intervalle approximatif [0.65 ; 1] (selon le nombre d'itération de l'apprentissage et le hasard de l'algorithme) au lieu de [0.5 ; 1] du cas classique. De plus on observe un cas un peu spécifique dans le cycle : quand un agent remonte à 1, si rien est fait, l'autre agent continuera peut être à proposer à 0.64 et donc vendra indéfiniment à ce prix. Il faut donc rajouter un mécanisme (selon le nombre de ventes consécutives, ou une synchronisation des agents, ce qui est peu compatible avec l'idée de vendeur concurrent).

Une autre stratégie a donc été testée : additionner le revenu immédiat associé à p2 (celui précédemment utilisé) avec le revenu de l'action en réponse de l'autre table. Ainsi en additionnant les deux revenus dus à une action p2 et la réaction que celle ci produit, on désire maximiser les profits des deux vendeurs pour chaque apprentissage sur une ligne de la table.
Mais malgré de nombreuses tentatives avec des paramètres différents, cette stratégie ne converge pas. Les prix résultants des tables sont donc incohérents et ne produisent de résultats satisfaisants.

On a alors essayé avec une définition différente des revenus, qui tient compte de la vente ou non d'un service en fonction des prix proposés par les deux vendeurs. L'idée est de récompenser avec un revenu une proposition de prix qui sera inférieur à la proposition de l'autre. Le mécanisme est le suivant :

Pour un état s, on a sur la ligne en cours d'apprentissage une proposition de prix p2. On interroge alors l'autre table pour obtenir sa proposition de prix (en fonction de la valeur Q maximum) pour le même état s. On compare alors les deux propositions :

- si la proposition de la table en cours d'apprentissage est inférieure, alors on attribue le revenu immédiat r = p2 – prix_coutant,
- sinon le revenu r est égal à 0, puisque le service n'est pas concurrentiel et qu'il ne serait pas vendu.

E appliquant cette nouvelle définition de revenu, on constate que la table « converge » et donne des prix cohérents. Mais on obtient le contraire de l'objectif ! Au lieu de maximiser les profits, on les minimise car les prix proposés sont toujours très faibles : entre 0.5 et 0.6 environ. L'explication semble être la suivante : les prix bas ont de moins de chance statistiquement de se voir associer un revenu de 0, puisque lorsque l'on interroge l'autre table, elle donne un prix de manière aléatoire (au début la table n'a pas de Q valeur cohérente et donc le choix revient à une décision aléatoire). Plus la proposition est basse et plus le prix de l'autre a des chances d'être supérieur : les prix bas ont alors souvent un revenu différent de 0 alors que les propositions de prix hautes ont plus souvent un revenu de 0.
Donc au fil des itérations, les bas prix sont avantagés. On obtient des profits souvent inférieurs aux profits du cas classique, sans Q-learning. La définition de revenu ici décrite n'est donc pas adaptée à nos objectifs.

Au final, aucun résultat vraiment satisfaisant n'a été obtenu avec les différentes définitions de revenu décrites. Il faudrait définir d'une meilleure façon les revenus afin de voir les tables converger vers des valeurs intéressantes. De plus, un agent Q-learner s'adapte beaucoup mieux à un environnement stationnaire, ce qui n'est pas le cas lorsque l'on a deux agents Q-learner. Mais il existe beaucoup de possibilité pour définir un revenu adéquat ainsi qu'un mode d'apprentissage adapté qui permettraient d'obtenir de meilleurs résultats.

Conclusion

Le travail effectué durant ce stage n'est bien entendu pas complet, le sujet du contrôle d'admission étant trop vaste. Mais certains points, comme l'architecture utilisant au maximum l'existant ou le système multi-agent écrit sur Oris, seront intéressants et réutilisables pour poursuivre le travail sur ce sujet. Les objectifs purement « réseaux » sont atteints puisque l'architecture est définie et même implémentée dans un simulateur. Elle permet une négociation entre les clients et les vendeurs, basée sur la notion de politiques et de part sa structure décentralisée exploitant l'existant, elle ne nécessiterait pas de gros déploiement au niveau matériel.
Bien que les résultats obtenus ne soient pas idéaux, on constate tout de même une amélioration dans le cas d'un agent face à un environnement stationnaire, ce qui est encourageant. Dans le cas de deux ou plus de deux agents, l'utilisation du Q-learning est plus délicate à mettre en œuvre et il serait peut être plus aisé d'être familiarisé avec cet algorithme d'intelligence artificielle pour continuer l'aspect « contrôle d'admission » et obtenir de meilleurs résultats.
Il serait intéressant (et même indispensable!) de rajouter d'autres paramètres afin de permettre un contrôle d'admission plus ou moins précis. Notamment la qualité du service, qui entre en compte fortement dans la négociation, ou la congestion du réseau (on peut même imaginer un indice représentant la qualité à l'intérieur d'un type de classe). En plus de permettre une gestion plus intéressante des ressources du réseau, ces deux paramètres influent sur le prix du service et permettraient peut être de contribuer à une stabilisation des prix. En effet, si les services peuvent se différencier selon des paramètres comme la classe de service, la qualité, ou la mobilité, alors on aura peut être différentes « niches » de services différents, où la concurrence serait moins vive. Mais il sera toujours nécessaire de prendre en compte la complexité de l'algorithme de Q-learning : elle augmente fortement avec le nombre de paramètres.
Un autre axe intéressant serait de réfléchir à d'autres algorithmes pour effectuer le contrôle d'admission en utilisant la même architecture. On pourrait aussi se placer dans un contexte différent, correspondant plus à de la coopération qu'à de la concurrence (un vendeur qui aurait par exemple plusieurs technologies différentes à proposer) : la communication entre les différents points de contrôle d'admission permettrait sûrement d'imaginer des mécanismes plus performants.

Références

[1] Al Agha K., Pujolle G. et Vivier G. « Réseaux de mobiles et sans fil » Edition Eyrolles 2001.
[2] Males D. « 802.11 » Rapport de stage de DEA.
[3] Katz R.H. and Brewer E.A. « The case for wireless overlay networks » in: Proc. SPIE Multimedia and Networking Conference (MMNC'96), San Jose, CA (January 1996).
[4] Katz R.H. and Stemm M. « Vertical Handoffs in wireless overlay networks » Mobile Networks and Applications, vol.3, no. 4, pp. 335-350, 1998.
[5] Aretz K., Haardt M., Konhäuser W. and Mohr W. « The future of wireless communications beyond the third generation » Computer Network 37 (2001), Elsevier.
[6] Konhäuser W. and Mohr W. " Access network evolution beyond third generation mobile communications" IEEE Communication Magazine (December 2000).
[7] Stemm M., Gauthier P., Harada D. and Katz R.H. "Reducing power consumption of network interfaces in hand-held devices" in: Proc. 3rd Workshop on Mobile Multimedia Communications (MoMuC-3, September 1996).
[8] Wang H.J., Katz R.H. and Giese J. "Policy-enabled handoffs across heterogeneous wireless networks".
[9] Perkins C. "IP Mobility Support" IETF RFC 2002, Oct. 1996.
[10] Johnson D. and Perkins C. "Mobility support in IPv6", work in progress, Jul 2001.
[11] Zhao X. and al. "Flexible network support for mobility" October 1998, MOBICOM 98.
[12] Cheshire S. and Baker M. "Internet Mobility 4x4" in ACM SIGCOMM August 1996.
[13] Wu G., Mizuno M. and Havinga P.J.M "MIRAI Architecture for Heterogeneous Network" IEEE Communication Magazine (February 2002).
[14] Campbell A.T. et al. « Design, Implementation and Evaluation of Cellular IP » IEEE Pers. Commun., vol. 7, no. 4, pp. 42-49, August 2000.
[15] "The COPS (Common Open Policy Server) Protocol", IETF RFC 2748, January 2000.
[16] « A Framework for Policy-based Admission Control », IETF RFC 2753, January 2000.
[17] « COPS Usage for RSVP » IETF RFC 2749, January 2000.
[18] Nguyen T.M.T., Pujolle G. et Boukhatem N. « COPS Usage for SLS Negociation (COPS-SLS)», Internet Draft, August 2001.
[19] Boukhatem N., Campedel B., Chaouchi H., Guyot V., Krief F., Nguyen T.M.T. et Pujolle G. "I3 – Une nouvelle génération intelligente de réseaux IP" GRES, Décembre 2001.
[20] Sridharan M. and Tesauro G. "Multi-Agent Q-learning and regression trees for automated pricing decisions" ICML'00, Stanford, CA, Juillet 2000.
[21] Boukhatem N., Gaïti D. et Pujolle G. « Agent-based Control for Policy-based Networking ».
[22] Ferber J. "Multi-agent Systems : Introduction to Distributed Artificial Intelligence" Addison Wesley, 1999.
[23] C.J.C.H. Watkins "Learning from delayed rewards." Ph D. thesis, Cambridge University, 1989
[24] Tesauro G. "Pricing in agent economies using neural networks and multi-agent Q-learning" IJCAI '99 , August 2, 1999, Stockholm
[25] IP-Based Access Network Infrastructure for Next-Generation - Ramjee, Porta et al. – 2000
[26] Fabrice Harrouet - « oRis : s'immerger par le langage pour le prototypage d'univers virtuels à base d'entités autonomes » Thèse – 2000 - EA2215 (UBO, ENIB)
[27] http://www.enib.fr/~harrouet/oris.html

Annexe

Fichier source de simulation sous Oris : agent Q-learner vs agent classique.
Nous présentons ici un fichier parmi les différents fichiers écrits pour les simulations.
Bien que ce fichier soit un peu long, il représente une partie non négligeable du travail effectué pendant le stage et est donc intégré dans ce rapport.
Ce fichier source illustre la façon dont le simulateur a été implémenté. La plus grande partie, correspondant à l'architecture du système multi-agent, est reprise dans chaque simulation.
Le client est toujours plus ou moins identique, alors que les objets auront un code différent selon leur rôle.

```
/* MIXTE : cette version ne comprend pas de valeur de qos, un seul service,
 et 2 vendeurs dont 1 seul effectue du qlearning
-(on suppose que le prix coutant est le meme pour les deux vendeurs)*/
include "files.pkg"

replace NB_REQ_CLIENT by 1000;
replace NB_EXPLO_ALEA by 20;
replace NB_EXPLO_LIGNE by 20;
replace GAMMA by 0.7;

class Vendeur;
class table;


/*-----------------*/
/* COMMUNICATIONS */
/*-----------------*/

/*-------------------------------------------------------------------------*/
/*  Classe de message pour l'envoi de la demande de service
d'un client vers les vendeurs : broadcast */
class Diffservice : Message
{
int service;
void new(Object emitter,int s);
void delete(void);
};

void Diffservice::new(Object emitter,int s)
{
service=s;
}

void Diffservice::delete(void) {}

/*-------------------------------------------------------------------------*/

/* Classe de message pour la réponse du vendeur qui envoie son offre à un client,
suite à un message diffservice : unicast */
class Mess_offre : Message
{
float prix;
//float qos;
//void new(Object emitter,float p, float q);
void new(Object emitter,float p);
void delete(void);
};

//void Mess_offre::new(Object emitter,float p, float q)
void Mess_offre::new(Object emitter,float p)
{
prix=p;
//qos=q;
```

```
}

void Mess_offre::delete(void) {}

/*--------------------------------------------------------------------------*/


/*  Classe de message pour l'envoi du service choisi et de son prix
d'un client vers les vendeurs : broadcast */
class Diffservice_elu : Message
{
Vendeur vendeur_elu;
float prix;

void new(Object emitter,Vendeur v,float p);
void delete(void);
};

void Diffservice_elu::new(Object emitter,Vendeur v,float p)
{
vendeur_elu=v;

prix=p;
}

void Diffservice_elu::delete(void) {}

/*--------------------------------------------------------------------------*/



// la classe vendeur doit etre définie avant la classe client, afin que celle-ci
//puisse utiliser les attribut de la classe vendeur
// attention , ici nous sommes dans un cas mixte : chaque vendeur a les memes noms
// de fonctions sauf en ce qui concerne la politique de prix.
// Mais l'un aura une table de Q-learning pour trouver son prix,
//l'autre implementera la baisse de son prix de 0.01 s'il n'a pas vendu son dernier
prix.

class Vendeur
{
//int[] qos; // valeur de qos pour chaque service
//int ressource; // ressources total (pas ideal:a améliorer)
float prix_coutant;
float dernier_prix;
int nb_vent; /* nbr de ventes consécutives : permet au vendeur sans Qlearning de
detecter
              le moment ou il peut augmenter son prix*/
float prix_vent; // prix de lors des ventes consécutives : idem

float profit;
int nb_client;   //nombre de client auxquels le vendeur a vendu un service
table tab; // table de q-learning

bool fin_app;
bool type_Qlearn; // 1 si agent Qlearning , 0 sinon
bool jaivendu;

int b;

//void new(int q1,int q2,int q3,int q4);
void new(float p_c,bool Qlearn);
void delete(void);
void traiter_serv(Diffservice diffrecue); // réagit à une demande de service d'1
client
void traiter_elu(Diffservice_elu diffrecue); // réagit à une confirmation de choix
de service d'1 client
float offre(void); // trouve l'offre correspond a une demande et rend le prix
void main(void);
};
```

```
/*--------------*/
/* AGENT CLIENT */
/*--------------*/

/*-------------------------------------------------------------------------*/
class Offre_vendeur // un objet manipulé par le client pour retenir les différentes
offres qu'il a recues
{
Vendeur vend; //vendeur proposant l'offre
float prix;
//float qos;
//float cout;   // pas indispensable car calculable qd on veut : memoire vs cpu????
//void new(Vendeur v,float p,float q);
void new(Vendeur v,float p);
void delete(void);
};

//void Offre_vendeur::new(Vendeur v,float p,float q)
void Offre_vendeur::new(Vendeur v,float p)
{
vend=v;
prix=p;
//qos=q;
}

void Offre_vendeur::delete(void){}

/*-------------------------------------------------------------------------*/

/* CLIENT : OBJET ACTIF : il "tourne" toujours.
Il représente le client qui envoie une demande, recoit des offres, choisit parmi
ces offres
et renvoie son choix au vendeur pour qu'il adapte leur offre*/

class Client
{
int b,b1,b2;             // les b sont utilisés pour ne rentrer qu'une fois dans un
if...(technique)
int i;                // i indexe les mess_offre recus
//int service;          // Classe de service à demander
int nb_requete;

TextOutputFile fichier;


/* Paramètres de préférences utilisateurs prix/qualité : ATTENTION ici seul le prix
compte donc
pas besoin de comparer prix et qualité d'un service avec une fonction de cout pour
choisir une offre*/
//float wprix,wqos;
        // prix et qos proposé par un réseau
//float prix,qos;

// fonction de cout pour le choix du réseau
//float f_cout(float p, float q);


Offre_vendeur[] tab_offre; // pour retenir les offres recues afin de pouvoir
choisir la plus interessante

//void new(int s, float wp, float wq);
void new(void);
void delete(void);
```

```
void main(void);
};




//void Client::new(int s, float wp, float wq)
void Client::new(void)
{
//service=s
//wprix=wp;
//wqos=wq;
}

void Client::delete(void)
{}

/*
float Client::f_cout(float p, float q)
{
return(wprix*p + wqos*q);
}
*/

void Client::main(void) // la fonction qui tourne non stop pour chaque client
{

execute
  {

// ENVOI DU MESSAGE DE DEMANDE DE SERVICE AUX VENDEURS
  if(b==0)
  {
    Diffservice diff=new Diffservice(this,1);
    diff->broadcast();
    nb_requete+=1;
    b=1;
    println(this," envoie une demande de service");
  }



// TRAITEMENT DES REPONSES DES VENDEURS

  int nb=getNbMessages();

  Mess_offre msg=(Mess_offre)getNextMessage();

    // créer le tableau des offres au fur et a mesure des arrivées des messages
d'offres
  if(msg!=NONE)
  {
    Offre_vendeur o=new Offre_vendeur((Vendeur)msg->getEmitter(),msg->prix);
    tab_offre.pushBack(o); // ajoute une offre au tableau d'offres
    //println("je suis ",this," j'ai recu de ",tab_offre[i]->vend,"  prix: ",msg-
>prix);
    //println();
    //i=i+1;
  }

  // affichage des valeurs recues et calculées d'un client
  // ATTENTION on suppose ici que l'on a 2 vendeurs(0,1) !!!!!
  int taille;
  int j;
  taille=tab_offre.length();

/* if(taille==3 && b1==0)
  {
    println("CONTENU du tableau offre du client: ",this);
```

```
    for(j=0;j<=2;j++)
    {
      println("*****",tab_offre[j]->vend,"  ",tab_offre[j]->prix,"  ",tab_offre[j]-
>qos,"  ",tab_offre[j]->cout,"*****");
    }
    println();
    b1=1;
  }
*/


// CHOIX DU VENDEUR EN FONCTION DES VALEURS RECUES
  // ATTENTION on suppose ici que l'on a 2 vendeurs(0,1) :
  //on attend d'avoir les deux offres avant de choisir le vendeur
  if(taille==2 && b2==0)
    {
    Vendeur vendeur_elu=tab_offre[0]->vend;
             //float min=tab_offre[0]->cout;
    float p=tab_offre[0]->prix;
            //for (j=1;j<=2;j++)
            //{
            // if(tab_offre[j]->cout<min)
    if(tab_offre[1]->prix<p)
      {
      vendeur_elu=tab_offre[1]->vend;
            //min=tab_offre[j]->cout;
      p=tab_offre[1]->prix;
      }

    println(" le client ",this," a elu le vendeur  ",vendeur_elu," avec le prix
",p);
    b2=1;
    Diffservice_elu diff2=new Diffservice_elu(this,vendeur_elu,p);
 // ENVOI D'UN MESSAGE AUX VENDEURS POUR SIGNALER L'OFFRE CHOISIE ET SON PRIX
   diff2->broadcast();

/* REINITIALISATION DU CLIENT DE CLASSE 1 : quand le client envoie le message
de signalement de son choix aux vendeurs,
on considere qu'il a consommé le service et qu'il est pret a en demandé un
nouveau.*/
    b=0;// RELANCE UNE NOUVELE DEMANDE DE SERVICE.
    b2=0;// PERMET de choisir a nouveau un vendeur
    i=0; //?????? utilité àvérifier
    tab_offre.popBack();//vide le tableau des offres: 2 offres dc deux popback
    tab_offre.popBack();

    }



  /* DEBUG
  println("              service:",service,"    wprix=",wprix,"
wqos=",wqos,"nombre de messages : ",nb);
  println("              ",this);

  prix=0.15;
  qos=0.34;
  println("              valeur de la fonctionde cout:", f_cout(prix,qos));
  println("              ------------------------------------");
  */

  if(this->nb_requete==NB_REQ_CLIENT)
    {// une fois arrivé à un nombre de service consommé, on affiche les résultats
interessants: les profits des vendeurs


println("**************************************************************************
*********");
    println("nb client :",this->nb_requete);
```

```
    Vendeur vend1=(Vendeur)getObject("Vendeur.1");
    Vendeur vend2=(Vendeur)getObject("Vendeur.2");

    println("profit du vendeur1 : ",vend1->profit);
    println("profit du vendeur2 : ",vend2->profit);

    println("nb client du vendeur1 : ",vend1->nb_client);
    println("nb client du vendeur2 : ",vend2->nb_client);

    println("profit par client du vendeur1 : ",vend1->profit/vend1->nb_client);
    println("profit par client du vendeur2 : ",vend2->profit/vend2->nb_client);


    bool ouvert=1;
    fichier=new TextOutputFile("resultat_client.txt",1);
    ouvert=fichier->open(fichier->fileName());
    fichier->writeString("\n");
    fichier->writeString("simu avec 1 agent Qlearner et 1 agent bete");fichier-
>writeString("\n");fichier->writeString("RESULTATS : ");fichier->writeString("\n");
    fichier->writeString(" Nombre d'iteration : ");fichier-
>writeInt(NB_EXPLO_ALEA); fichier->writeString(" * "); fichier-
>writeInt(NB_EXPLO_LIGNE); fichier->writeString("\n");
    fichier->writeString(" GAMMA : "); fichier->writeFloat(GAMMA); fichier-
>writeString("\n");
    fichier->writeString("\n");
    fichier->writeString(" profit total : "); fichier->writeFloat(vend1-
>profit+vend2->profit); fichier->writeString("\n");
    fichier->writeString(" profit du vendeur 1 (avec Qlearning): "); fichier-
>writeFloat(vend1->profit); fichier->writeString("\n");
    fichier->writeString(" profit du vendeur 2 : "); fichier->writeFloat(vend2-
>profit); fichier->writeString("\n");
    fichier->writeString("\n");
    fichier->writeString(" nombre de client total :  "); fichier-
>writeInt(nb_requete); fichier->writeString("\n");
    fichier->writeString(" nombre de client du vendeur 1: "); fichier-
>writeInt(vend1->nb_client); fichier->writeString("\n");
    fichier->writeString(" nombre de client du vendeur 2: "); fichier-
>writeInt(vend2->nb_client); fichier->writeString("\n");
    fichier->writeString("\n");
    fichier->writeString(" profit moyen par client du vendeur 1 (avec Qlearning):
"); fichier->writeFloat(vend1->profit/vend1->nb_client); fichier-
>writeString("\n");
    fichier->writeString(" profit moyen par client du vendeur 2: "); fichier-
>writeFloat(vend2->profit/vend2->nb_client); fichier->writeString("\n");
    fichier->writeString("\n");
    fichier->writeString("-------------------------------------------------------------
---------------------------");

    fichier->close();
    this->suspend();
    }

  }
}



/*-------------------------------------------------------------------------*/

/*-------------------------------------*/
/* TABLE DU VENDEUR AYANT LE Q-LEARNING*/
/*-------------------------------------*/

class table
{
float[][] t; // tableau à 2 dim
float prix_c;
TextOutputFile toto;
```

```
void new(float p_c);
void delete(void);

void aff_tab(int a, int b);
void aff_tab_tout(void);

void impr_tab(int a, int b);

int nb_visit_min(int a,int b,int & ligne); // donne le nombre de visit de la ligne
la moins visitée
float trouve_action_max(float prix_autre);
int trouve_ligne_action_max(float prix_autre);
int trouve_ligne_action(float prix_autre);
float trouve_valeurq_max(float prix_autre);
void calcul_r(float a1,float a2,float & a_r,float & rev_commun);
float app_Q(int x);
}



void table::new(float p_c) // creation et initialisation de la table
{

float affect;
float alea;
int inc=0;

prix_c=p_c;


for(int l=0;l<=100;l++)
  {
  for(int m=0;m<=100;m++)
    {
    if( ((float)m/100 - prix_c <0) || ((float)l/100 - prix_c <0) )
    {}
    else
      {
      t.pushBack((float[]) []); // ajoute une ligne vide
      for(int c=0;c<5;c++)
        {

        if(c==0){affect=inc;}
        else if(c==1){ affect=(((float)l/100)); }
        else if(c==2){ affect=(((float)m/100)); }
        else if (c==4){ /*alea=?;*/ affect=0.0;}//avant:(float)m/100 - prix_c;
        else affect=0;


        t.back().pushBack(affect);
        }
      inc=inc+1;
      }
    }
  }
//pour finir le tableau, une ligne de plus facilite la recherche dans la table
t.pushBack((float[]) []); // ajoute une ligne vide que l'on remplie ensuite:
t.back().pushBack(inc);
t.back().pushBack(2.0);
t.back().pushBack(2.0);
t.back().pushBack(0);
t.back().pushBack(2.0);
}

void table::delete(void)
{}

void table::aff_tab(int a,int b)
```

```
{
println(" indice | prix autre | prix a proposer | visit | valeur Q  ",this);
if(this->t.length()<b){println("NE PEUT PAS AFFICHER PLUS QUE LA LONGUEUR DE LA
TABLE!!!!! longueur: ",this->t.length());}
else
  {
  for(int l=a;l<b;l++)
    {
    println("   ",this->t[l][0]," |     ",this->t[l][1]," |    ",this->t[l][2]," |
",this->t[l][3]," |    ",this->t[l][4]);
    }
  }
}


void table::aff_tab_tout(void)
{
  for(int l=0;l<this->t.length();l++)
    {
    println("   ",this->t[l][0]," |     ",this->t[l][1]," |    ",this->t[l][2]," |
",this->t[l][3]," |    ",this->t[l][4]);
    }
}

void table::impr_tab(int a, int b)
{
bool ouvert=1;
string str;
float f;

if(this->name()=="table.1"){str="t1.txt";}else{str="t2.txt";};
println("NOM du fichier:",str);


toto=new TextOutputFile(str,0);
ouvert=toto->open(toto->fileName());

if(this->t.length()<b){println("NE PEUT PAS AFFICHER PLUS QUE LA LONGUEUR DE LA
TABLE!!!!! longueur: ",this->t.length());}
else
  {
  for(int l=a;l<b;l++)
    {
    //println("   ",this->t[l][0]," |     ",this->t[l][1]," |    ",this->t[l][2],"
|    ",this->t[l][3]," |    ",this->t[l][4]);
    toto->writeString("   ");
    toto->writeFloat(t[l][0]);
    toto->writeString("   ");
    toto->writeFloat(t[l][1]);
    toto->writeString("   ");
    toto->writeFloat(t[l][2]);
    toto->writeString("   ");
    toto->writeFloat(t[l][3]);
    toto->writeString("   ");
    toto->writeFloat(t[l][4]);
    toto->writeString("\n");
    }
  }
toto->close();
}


int table::nb_visit_min(int a,int b,int & ligne)
{
int i,temp_i=a;
int temp_nb_visit=t[0][3];

for(i=a;i<=b;i++)
  {
  if(t[i][3]<temp_nb_visit)
```

```
    {temp_nb_visit=t[i][3];temp_i=i;}
  }
ligne=temp_i;
return(temp_nb_visit);
}


float table::trouve_action_max(float prix_autre)
{
int i=0;
float temp_Q;
float temp_action;

//i=prix_autre*100*101; calcul de l'indice impossible car suppression de lignes"à
pertes"
while(this->t[i][1]<prix_autre){i=i+1;}//scanne la table jusqu'a t[i][1]=prix_autre


temp_Q=this->t[i][4];
temp_action=this->t[i][2];


while (this->t[i][1]==prix_autre)
  {
  if(this->t[i][4] >= temp_Q)
    {
    temp_action=this->t[i][2];
    temp_Q=this->t[i][4];
    }
  i=i+1;
  }
return(temp_action);
}

int table::trouve_ligne_action_max(float prix_autre)
{
int i=0;
float temp_Q;
int temp_ligne;

//i=prix_autre*100*101; calcul de l'indice impossible car suppression de lignes"à
pertes"
while(this->t[i][1]<prix_autre){i=i+1;}//scanne la table jusqu'a t[i][1]=prix_autre


temp_Q=this->t[i][4];
temp_ligne=this->t[i][0];


while (this->t[i][1]==prix_autre)
  {
  if(this->t[i][4] >= temp_Q)
    {
    temp_ligne=this->t[i][0];
    temp_Q=this->t[i][4];
    }
  i=i+1;
  }

return(temp_ligne);
}

int table::trouve_ligne_action(float prix_autre)
{
int i=0;

//i=prix_autre*100*101;NON -> calcul de l'indice impossible car suppression de
lignes"à pertes"
while(this->t[i][1]<prix_autre){i=i+1;}//scanne la table jusqu'a t[i][1]=prix_autre
return(i);
```

```
}



float table::trouve_valeurq_max(float prix_autre)
{
int i;
float temp_Q;
float temp_action;

//i=prix_autre*100*101; calcul de l'indice impossible car suppression de lignes"à
pertes"
while(this->t[i][1]<prix_autre){i=i+1;}//scanne la table jusqu'a t[i][1]=prix_autre

temp_Q=this->t[i][4];
//temp_action=this->t[i][2];
i=i+1;

while (this->t[i][1]==prix_autre)
  {
  if(this->t[i][4] >= temp_Q)
    {
    //temp_action=this->t[i][2];
    temp_Q=this->t[i][4];
    }
  i=i+1;
  }

return(temp_Q);
}




void table::calcul_r(float a1,float a2,float & a_r,float & rev_commun)
{
/* le revenu est la somme ( ou seulement le revenu immédiat de a2 ):

- du revenu immediat (un prix - le prix coutant) de l'action choisie (un prix) dans
la table de Qlearning.

- du revenu immediat induit par l'action que l'autre vendeur fera, suite a la
premiere action.
Ici l'autre vendeur a une regle simple :
 quelle que soit l'action choisi par le premier agent (avec apprentissage),
 l'action réponse sera toujours inférieure de 0.02.
 Cette règle vient du fait que cet agent est sans apprentissage et donc a pour
 seule politique de baisser son prix pour etre compétitif */

float rev_imm_propre;
float rev_imm_autre;


if(a2<a1-0.02) // ICI A DEFINIR A PARTIR DE QD ON DONNE UN REVENU
{ rev_imm_propre=a2 - prix_c;}
//revenu immédiat de l'action de la ligne de la table de Q-learning : prix a
proposer-prix coutant
else {rev_imm_propre=0;}

// on cherche l'action réponse que l'autre vendeur(sans apprentissage) donnera.
// on retranche 0.02 pour avoir le prix a proposer de l'autre,
//sauf si cela donne un résultat inférieur au prix coutant.
if(a2-0.02 < prix_c){a_r=a2;}
else {a_r=a2-0.02;}

// rechercher aussi le revenu immédiat de l'autre pour cette action
// (on suppose que le prix coutant est le meme pour les deux vendeurs)
```

```
rev_imm_autre=a_r - prix_c;

// revenu commun d'un couple action et action réponse
rev_commun=rev_imm_propre;  //+rev_imm_autre; ON AJOUTE OU NON LE REVENU DE
L'ACTION DE L'AUTRE
//println("revenu imm propre et de l'autre: ",rev_imm_propre," ",rev_imm_autre);
//println("*****************************************************************");
}


/*float table::app_Q(int x) // pas utiliser dans ce prog
{
float new_Qval;

float Qval;
float alpha;
float gamma=GAMMA;
float r;

float action1;
float action2;
float action_rep;

this->t[x][3]+=1;

Qval=this->t[x][4];
action1=this->t[x][1];
action2=this->t[x][2];
calcul_r(action1,action2,action_rep,r);

alpha=1/ (1+ this->t[x][3]);
new_Qval=Qval + alpha*( r + gamma*(trouve_valeurq_max(action_rep))-Qval );

return(new_Qval);
*/
}


/*-------------------------------------------------------------------------*/

/*---------------*/
/* AGENT VENDEUR */
/*---------------*/

/*
class Vendeur : définie plus haut

*/


void Vendeur::new(float p_c, bool Qlearn)
{
dernier_prix=1;

setSensitivity("Diffservice","traiter_serv");
setSensitivity("Diffservice_elu","traiter_elu");

prix_coutant=p_c;
type_Qlearn=Qlearn ;// 1 si agent Qlearning , 0 si agent sans apprentissage

if(type_Qlearn)
  {
  tab=new table(prix_coutant);
  //yield();
  println(" TABLE DE Q-LEARNING de ",this,  "table de q : ",tab);
  }
}

void Vendeur::delete(void)
{}
```

```
void Vendeur::traiter_serv(Diffservice diffrecue)
{
float prix;
//println("*****je suis",this,"    j'ai recu : --",diffrecue->service,"-- du
client: ",diffrecue->getEmitter());

/* trouve le prix si l'offre est possible*/
println();
prix=offre();

/*envoie du message de réponse*/
Mess_offre mess=new Mess_offre(this,prix);
mess->sendTo(diffrecue->getEmitter());
}

void Vendeur::traiter_elu(Diffservice_elu diffrecue)
{
  // test du vendeur elu
  println(" ----------> diffelu :  ",diffrecue->vendeur_elu,"  vendeur qui
recoit",this);
  if(diffrecue->vendeur_elu==this)
  {
    // je suis le vendeur choisi : j'ai un profit
    println("le vendeur choisi est ",this," au PRIX ACCEPTé: ",diffrecue->prix);
    println("benef pour ",this," : ",diffrecue->prix-prix_coutant);
    nb_client=nb_client+1;
    profit=profit+(diffrecue->prix-prix_coutant);
    dernier_prix=diffrecue->prix;
    jaivendu=1; // indique le vendeur avréussi a vendre a ce dernier prix
  }
  else
  {
    // je ne suis pas le vendeur choisi : pas de profit
    println(" c'est PAS moi qui suis choisi",this," je mets donc mon dernier_prix
au prix accepté...");
    dernier_prix=diffrecue->prix;
    jaivendu=0; // indique le vendeur N'A PAS réussi a vendre a ce dernier prix
  }
}


float Vendeur::offre(void)
{
/* dans ce cas mixte le prix a trouver dépend du type de vendeur:
   - le vendeur apprenti utilise sa table
   - l'autre utilise son algo de prix sans apprentissage
*/
float prix_a_trouver;
/* d'abord controler la possibilité de fournir le service demandé*/
// faire du : ressource total - ressource demandéés par le service pour commencer

/* trouver le prix à proposer : consulter la table de Q-learning ou non selon le
type de vendeur*/
if(type_Qlearn)
  {
  // on utilise la table de Qlearning
  prix_a_trouver=tab->trouve_action_max(dernier_prix);
  println("dernier_prix= ",dernier_prix," prix trouvé par le Qlearner :
",prix_a_trouver);
  return (prix_a_trouver);
  }
else
  {  // cas sans apprentissage

  if(jaivendu)
    {
    nb_vent=nb_vent+1;
    if(nb_vent==1){prix_vent=dernier_prix;}
    if(nb_vent==3 && prix_vent==dernier_prix)
```

```
      {return(1.0);}
    else {return(dernier_prix);}
    }
  else{
      nb_vent=0;
      if(dernier_prix-0.02 < 0.5){return(dernier_prix);} // ici on passe en dessous
du prix coutant!!!
      else{return(dernier_prix-0.02);}
      }
  }
}


void Vendeur::main(void)
{
execute
  {

  }
}


/*-------------------------------------------------------------------------*/
/*---------------*/
/* PROG PRINCIPAL*/
/*---------------*/
execute
{
int i,j,x,aff;
float f;
bool a;

/* création de deux vendeurs : avec et sans apprentissage*/
println("Avec et sans app : mixte.ors\nmeme prix coutant pour les deux");
Vendeur v1=new Vendeur(0.50,1);//avec apprentissage

Vendeur v2=new Vendeur(0.50,0);//sans apprentissage


int long_table=v1->tab->t.length();


v1->tab->aff_tab(0,50);

int ligne,ligne_suiv;
float Qval;
float alpha;
float gamma=GAMMA;
float r;

float action1;
float action2;
float action_rep;

for(i=0;i<NB_EXPLO_ALEA;i++)
  {
  x=?(long_table-1);
  println("pour la ",i,"eme fois , on prends une ligne (ici : ",x," ) au hasard
dans la table");
  if(x!=2601)
    {
    ligne=x;
    for(j=0;j<NB_EXPLO_LIGNE;j++)
      {

      v1->tab->t[ligne][3]+=1; //incrementation du nombre de visite

      Qval=v1->tab->t[ligne][4];
      action1=v1->tab->t[ligne][1]; // dernier prix
```

```
      action2=v1->tab->t[ligne][2]; // action (donc prix) choisi par la table de
Qlearning

      v1->tab->calcul_r(action1,action2,action_rep,r);
      // donne a partir de l'action décidée par la table : réaction de l'autre et
revenu cumulé de l'autre

      //alpha=1/ (1+ v1->tab->t[ligne][3]);
      alpha=0.1;
      v1->tab->t[ligne][4]= Qval + alpha*( r + gamma*(v1->tab-
>trouve_valeurq_max(action_rep))-Qval );

      ligne=v1->tab->trouve_ligne_action(action_rep);
      // trouve la ligne ou l'action rep est la premiere fois presente en position
[ligne][1]
      v1->tab->nb_visit_min(ligne,ligne+50,ligne_suiv);
      ligne=ligne_suiv;

      //println(" action proposée : ",action,"  action en réponse : ",action_rep,"
prochaine ligne : ",ligne);
      }
    }

  }
int p;
println("nb_visit_min de table1 : ",v1->tab->nb_visit_min(0,v1->tab->t.length()-
2,p));


Client c=new Client();


}
```