Adaptation des applications distribuées à la Qualité de Service fournie par le réseau de communication

Fabien Michaut

CENTRE DE
RECHERCHE EN
AUTOMATIQUE
DE NANCY

CNRS UMR 7039

U.F.R. : ESSTIN

Ecole Doctorale : IAEM

Département de Formation Doctorale : Automatique et Production Automatisée

Thèse

présentée pour l'obtention du titre de

Docteur de l'Université Henri Poincaré, Nancy I

en Automatique, Traitement du Signal et Génie Informatique

par **Fabien MICHAUT**

Adaptation des applications distribuées à la Qualité de Service fournie par le réseau de communication

Thèse soutenue le 26 Novembre 2003

Membres du jury :

Président :	M. Michel ROBERT	Professeur, Université Henri Poincaré, Nancy I
Rapporteurs :	Mme Dominique GAITI	Professeur, Université de Technologie de Troyes
	M. Guy JUANOLE	Professeur, Université Paul Sabatier, Toulouse III
Examinateurs :	M. Francis LEPAGE	Professeur, Université Henri Poincaré, Nancy I *(Directeur de thèse)*
	M. Thierry DIVOUX	Professeur, Université Henri Poincaré, Nancy I
	M. Philippe FRAISSE	Maître de Conférence, Université de Montpellier II

Université Henri Poincaré, Centre de Recherche en Automatique de Nancy
Faculté des Sciences B.P. 239 – 54506 Vandoeuvre-lès-Nancy

Remerciements

Mes remerciements iront en premier lieu à mon directeur de thèse, le Professeur Francis Lepage pour m'avoir accueilli au CRAN et pour m'avoir encadré durant ces trois dernières années. Je le remercie donc pour ses précieux conseils, sa collaboration et sa grande convivialité.

Je tiens à exprimer ma profonde gratitude aux Professeurs Francis Lepage et Michel Robert pour leur soutien dans l'obtention de la bourse BDI dont j'ai bénéficié et sans laquelle je n'aurai pu mener ces travaux.

Je tiens à remercier de plus toutes les personnes avec qui j'ai pu échanger des idées lors de longues discussions (notamment dans notre *Brain Storming Room!*). Je ne saurais non plus oublier les personnes que j'ai côtoyées au quotidien au laboratoire pour le climat de travail et les bons moments de vie.

Ce travail n'est en aucun cas le fruit d'une recherche menée en solitaire et je tiens à remercier Thierry, Thomas, Pavol, Eric, Jean-Philippe, Jamal, Pascal et Christophe pour leur aide précieuse et leurs conseils.

J'adresse mes sincères remerciements aux Professeurs Dominique Gaïti de l'Université de Technologie de Troyes et Guy Juanole de l'Université Paul Sabatier de Toulouse III pour avoir accepté de rapporter sur ma thèse et pour leurs suggestions et remarques qui m'ont permis de mettre un point final à ce travail.

Je remercie également les autres membres du jury pour s'être intéressés à mes travaux.

J'associe à ces remerciements, mes parents et toute ma famille pour leur soutien durant mes années d'étude.

Pour finir, je remercie le CNRS et la Région Lorraine pour leur soutien financier.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Aujourd'hui, les échanges d'information prennent des dimensions géographiques plus importantes. De ce fait, les réseaux utilisés sont le plus souvent hétérogènes et non déterministes. Parallèlement, les informations transférées sont de plus en plus complexes (voix, vidéo, données robotiques, etc.) et contraintes (temps-réel, etc.). Ainsi, les réseaux de communication à commutation de paquets initialement conçus pour acheminer des trafics sans contraintes particulières (messagerie, transfert de fichiers, etc.) doivent à présent satisfaire les besoins d'applications distribuées "critiques".

Afin de permettre à ces nouvelles applications un fonctionnement correct et cohérent, il devient alors indispensable de prendre en compte la Qualité de Service (QdS) de bout en bout (incluant le réseau et les systèmes terminaux) et particulièrement des systèmes de communication.

Dans cette introduction, nous allons dans un premier temps présenter les nouvelles applications et leurs contraintes en terme de ressources de communication. Nous expliquerons ensuite les approches possibles pour la prise en compte de ces besoins. Nous préciserons alors les objectifs visés dans nos travaux pour finalement déboucher sur le plan du mémoire.

1.1 Applications et contraintes de Qualité de Service

Parmi les nouvelles applications évoquées précédemment, les plus nombreuses sont sans doute les applications multimédias. Celles-ci se distinguent des applications "classiques" par le fait qu'elles manipulent essentiellement des flux continus et non des fichiers[1] : par

[1] Le terme fichier est utilisé ici pour désigner un ensemble de données temporellement indépendantes les unes des autres.

exemple, les données audio sont des séquences d'échantillons sonores, les données vidéo des séquences d'images. De par leur nature, les flux multimédias sont contraints en terme de QdS. Par exemple, des contraintes temporelles fortes existent entre les différentes unités qui composent un flux. En effet, pour les flux tels que la vidéo et l'audio, les images ou les échantillons sonores doivent être produits, traités, présentés à une cadence régulière et imposée. De plus, quand une application manipule plusieurs flux, il existe souvent des contraintes temporelles "inter-flux" (synchronisation d'un flux audio et d'un flux vidéo, etc.). Les flux ont aussi des contraintes spatiales qui portent sur des aspects quantitatifs (taille des images, etc.), et des contraintes de fiabilité qui expriment la marge d'erreur acceptable pour la transmission des unités du flux (nombre maximum d'échantillons sonores non délivrés, etc.).

L'usage croissant d'Internet par des applications de télé-opération doit aussi être soulignée. Ces applications consistent à déporter le pilotage de systèmes automatisés. Elles permettent notamment l'utilisation de compétences très spécifiques à distance (télé-chirurgie, etc.). Pour ces applications, les informations échangées sont des ordres de commande et des mesures issues de capteurs et sont fortement contraintes (respect de la période d'échantillonnage, etc.). Ainsi, l'insertion d'un réseau dans la boucle de commande perturbe le fonctionnement du système, nuit à ses performances et peut menacer sa sécurité et celle de son environnement. Par exemple, un délai de transmission trop important peut causer l'instabilité d'un tel système.

Remarquons que le contrôle de systèmes distribués n'est pas un concept nouveau. Depuis plus de 20 ans, les réseaux locaux industriels permettent de commander et de surveiller de tels systèmes. De grands débats scientifiques ont eu lieu au sujet des spécifications de ces réseaux afin qu'ils respectent les contraintes temporelles imposées par les systèmes distribués. Ces réseaux relèvent totalement de leur exploitant et présentent une faible dispersion géographique. Ce dernier peut donc faire une évaluation préalable des ressources nécessaires puis architecturer et dimensionner son réseau en conséquence. L'exploitant a une maîtrise complète du réseau, ce qui n'est pas le cas lorsque le réseau employé est géré tout ou partie par un opérateur extérieur (réseau public grande distance).

Notons de plus, que selon les applications, les contraintes de QdS sont différentes. Par exemple, une application de téléphonie est surtout contrainte en terme de délai (un délai trop élevé entre deux interlocuteurs rend la conversation difficile, voire impossible) et nécessite peu de bande passante alors que le fonctionnement d'une application de diffusion vidéo est essentiellement conditionné par la bande passante disponible.

1.2 Prise en compte de la Qualité de Service

Pour prendre en compte les contraintes de QdS des applications, il existe plusieurs approches. Celles-ci peuvent se classer en deux catégories.

La première repose sur la gestion des ressources, par réservation ou par gestion de priorités :

- Le modèle de services intégrés de l'IETF (Integrated Services ou IntServ) [Int] ainsi que des architectures proposées dans la littérature [NS95a], [FRS00], [OMRW98] se fondent sur la réservation de ressources qui se déroule en deux phases : tout d'abord, durant le contrôle d'admission, on vérifie que les ressources demandées par l'application sont disponibles. Lorsque ce n'est pas le cas, l'application doit réviser ses besoins à la baisse et une négociation intervient entre celle-ci et le système de communication. Lorsque les ressources demandées sont disponibles, elles sont réservées et garanties pour l'application. L'avantage majeur de ce type d'approche est de fournir une garantie stricte de QdS (délai maximum par exemple). Son inconvénient principal réside dans la complexité et la lourdeur des mécanismes qu'elle nécessite. En effet, chaque noeud du réseau concerné par la réservation doit maintenir un état par flot. En conséquence, la réservation de ressource ne résiste pas au facteur d'échelle.
- En raison de la complexité de la réservation de ressource, une solution plus facile à déployer a été développée par l'IETF. Il s'agit de la différenciation de services (Differentiated Services ou DiffServ) [Dif]. Elle consiste à pré-classer les flots selon leur niveau de priorité dans le réseau par marquage direct des paquets. Les flots sont ainsi agrégés en un nombre limité de classes de service. Celles-ci sont exploitées par les routeurs pour fournir un traitement différencié à chaque paquet en fonction de son degré de priorité ("traitement express" par exemple). Il n'est alors plus nécessaire de maintenir des états dans les routeurs pour chacun des flots. Le marquage des paquets s'effectue uniquement aux points d'entrée sur le réseau. Les routeurs intermédiaires se contentent de le lire et n'ont plus à le modifier. Les mécanismes de QdS de ce type ne font qu'aménager le "best-effort" et sont incapables d'offrir de garantie stricte de QdS.

La seconde approche considère qu'aucune garantie ne peut être obtenue ni du système terminal, ni du réseau de communication. C'est alors à l'application de s'adapter à son environnement [DHT95]. Dans ce cadre, plusieurs applications adaptatives ont été proposées : les mécanismes habituels d'adaptation consistent à moduler le débit de l'appli-

cation en fonction du taux de pertes de paquets sur le réseau ou à adapter la taille d'une mémoire-tampon afin de compenser les variations de délai subies par le trafic [WS99] [VG96]. Malgré leur adaptabilité, ces applications ne fonctionnent pas dans une architecture à QdS intégrée complète au sens de Campbell [Cam96]. Par exemple, la surveillance des ressources de communication y est peu ou pas décrite. En outre, les deux techniques précédentes ne sont pas les seuls mécanismes adaptatifs possibles. D'autres approches ont été développées dans [CK99], [BCA+00], [KHP99] et [BG97]. Ces travaux sont focalisés sur les "mécanismes logiciels" utilisés pour l'adaptation et ne définissent pas non plus une architecture complète pour l'adaptation.

Notons que ces deux approches ne sont pas antagonistes mais complémentaires [BG98]. C'est le cas par exemple lorsqu'une réservation de ressources est effectuée et qu'un ajustement des demandes de l'application aux ressources disponibles est nécessaire.

1.3 Objectifs de la thèse

Les travaux présentés dans cette thèse s'inscrivent dans la deuxième catégorie d'approches : partant de l'hypothèse que la maîtrise du système de communication n'est pas totale ou est impossible, ces travaux visent à doter les applications distribuées de moyens leur permettant de pallier l'insuffisance du système de communication en adaptant leur exécution.

Dans ce contexte, nous définirons une architecture à QdS intégrée (ou architecture de QdS) dédiée à l'adaptation des applications aux fluctuations des ressources disponibles. Cette architecture comprendra des mécanismes spécifiques dont notamment un service de métrologie chargé de mesurer la QdS disponible. La métrologie de la QdS des réseaux de communication à commutation de paquets constituera d'ailleurs une partie essentielle de nos travaux.

En outre, l'architecture proposée sera suffisamment générique pour être utilisée par des applications multimédia et des applications de télé-opération. C'est dans ce dernier cas que nous l'évaluerons.

1.4 Organisation du document

Le deuxième chapitre présente succinctement la théorie de la commande adaptative en automatique. Il indique que les systèmes distribués ne sont pas propices à l'application

de cette théorie et qu'il n'existe pas de cadre conceptuel générique pour l'adaptation de ces systèmes. Par contre, plusieurs techniques d'adaptation spécifiques ont été proposées dans la littérature. Celles-ci sont alors passées en revue. Par ailleurs, nous expliquons que la bonne intégration de ces techniques dans les systèmes distribués requiert différents mécanismes particuliers que nous explicitons, au sein d'une architecture à QdS intégrée.

Le troisième chapitre est consacré à la métrologie réseau. Les paramètres essentiels de "QdS réseau" y sont clairement définis. Il s'agit de la bande passante totale, minimale et disponible, du délai unidirectionnel et aller-retour, de la variation de délai, des pertes de paquets et de la route. Pour chaque paramètre, un état de l'art des techniques et des outils de mesure associés est exposé. Il permet notamment de constater la diversité de ces derniers et l'intérêt que susciterait un environnement de métrologie intégré offrant la possibilité de mesurer plusieurs paramètres différents.

Dans le quatrième chapitre, nous proposons une nouvelle architecture à QdS intégrée dédiée à l'adaptation des applications aux performances du réseau. Au travers des concepts de "classes de QdS" et de "blocs d'adaptation", les applications spécifient leur politique d'adaptation dynamique aux variations de QdS. L'adaptation des applications est pilotée par une couche dédiée qui inclut un service de métrologie des performances courantes (la conception modulaire de ce dernier autorise l'implémentation de n'importe quelle technique de mesure exposée dans le chapitre précédent). Tout d'abord, les applications fournissent elles-mêmes les règles d'adaptation à cette couche. C'est ensuite la couche qui les applique en se basant sur les mesures du service de métrologie.

Le cinquième chapitre présente l'implémentation de l'architecture et de son service de métrologie de la QdS sur une plateforme de télé-opération d'un robot mobile au travers de deux applications : la première est une application "récréative" de télé-pilotage du robot mobile et a une vocation démonstrative. La deuxième concerne l'asservissement en position du robot mobile à distance. L'expérimentation montre l'intérêt de notre démarche : l'architecture permet d'adapter l'asservissement à la QdS du réseau et évite ainsi les dépassements du système. Les risques de collisions du robot avec son environnement sont alors limités.

Chapitre 2

Adaptation dans les systèmes distribués

2.1 Adaptation en Automatique : commande adaptative (d'après [LD86])

La "Commande adaptative" est un ensemble de techniques utilisées pour l'ajustement automatique en ligne et en temps réel des régulateurs des boucles de commande afin de réaliser ou maintenir un certain niveau de performances quand les paramètres du procédé à commander sont soit inconnus soit/et varient dans le temps.

Un système de commande est sujet à deux types de perturbations :

- perturbations agissant sur les variables à réguler,
- perturbations (paramétriques) agissant sur les performances du système de commande.

La contre-réaction est essentiellement utilisée dans les systèmes de régulation conventionnels pour réduire (ou éliminer) l'effet des perturbations agissant sur les variables à réguler. Pour la réaliser, on mesure les variables, on les compare aux valeurs désirées et les différences sont appliquées à l'entrée du régulateur qui engendre la commande appropriée. Un système de commande à contre-réaction conventionnelle va donc réduire l'effet des perturbations agissant sur les variables à réguler mais ses performances dynamiques vont varier sous l'effet des perturbations paramétriques.

Une approche conceptuellement similaire peut être considérée pour le problème du

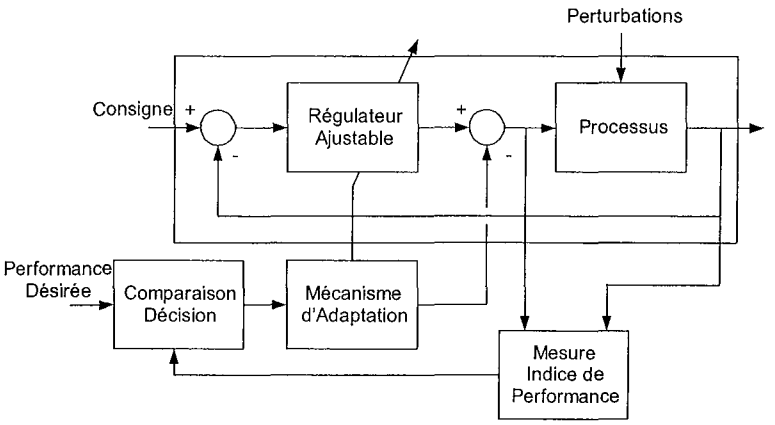


FIG. 2.1 – Commande adaptative

maintien des performances désirées d'un système de commande en présence de perturbations paramétriques. Il faut définir d'abord un indice de performance (I.P.) du système qui est une mesure des performances du système (ex : facteur d'amortissement pour des systèmes caractérisés par une fonction de transfert du 2ème ordre). Il faut ensuite mesurer cet I.P. et le comparer avec l'I.P. désiré. L'écart entre l'I.P. désiré et l'I.P. mesuré est traité par un "mécanisme d'adaptation". La sortie du "mécanisme d'adaptation" agit sur les paramètres du régulateur ou directement sur le signal de commande afin de modifier d'une manière appropriée les performances du système. Ainsi, un système de commande adaptative contient en plus d'une boucle de commande à contre-réaction ayant un régulateur à paramètres ajustables, une boucle supplémentaire qui agit sur les paramètres du régulateur afin de maintenir les performances du système en présence de variations des paramètres du procédé. Cette boucle supplémentaire a aussi une structure à contre-réaction où la variable contrôlée est la performance du système de commande proprement dit. Le principe des systèmes de commande adaptative est illustré sur la figure 2.1.

Les techniques de commande adaptative utilisées en pratique sont la commande auto-ajustable et la commande adaptative à modèle de référence. Leur développement repose sur l'hypothèse fondamentale suivante : pour toutes les valeurs possibles des paramètres du procédé on suppose qu'il existe un régulateur de structure donnée qui peut assurer la réalisation des performances désirées. Le rôle de la boucle d'adaptation est uniquement limité à trouver les bonnes valeurs des paramètres de ce régulateur dans chaque cas.

Le principe de calcul d'un régulateur suppose la connaissance du modèle dynamique

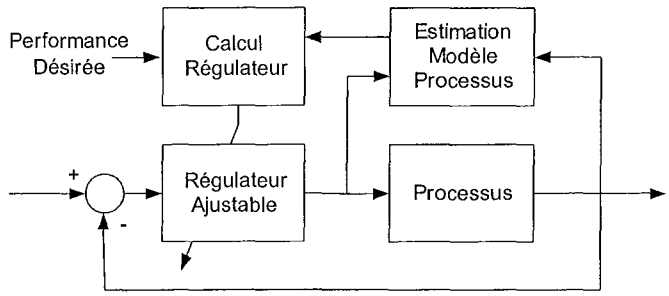


FIG. 2.2 – Commande auto-ajustable

du procédé et des performances désirées. Dans des nombreuses situations les performances désirées du système de commande à contre-réaction peuvent être spécifiées en termes de "fonction de transfert". Dans ces cas le régulateur est calculé de telle sorte que pour un modèle de procédé donné, le système en boucle fermé soit caractérisé par une "fonction de transfert" spécifiée a priori.

Dans le cas de la commande auto-ajustable, le modèle du procédé servant pour le calcul est remplacé par un modèle estimé (identifié) en temps réel à partir des entrées et des sorties du procédé (cf. figure 2.2). Les paramètres du procédé sont estimés en utilisant un prédicteur de la sortie du procédé. Ce sont ces paramètres qui, à chaque pas, sont utilisés pour le calcul du régulateur.

Dans le cas de la commande adaptative avec modèle de référence explicite (explicit MRAC), la fonction de transfert désirée du système en boucle fermée est connue. Une réalisation de cette fonction de transfert, appelée "modèle de référence" est utilisée. La différence entre la sortie du procédé et la sortie du modèle de référence est une mesure de la différence entre la performance réelle et la performance désirée. Cette information est utilisée par le mécanisme d'adaptation pour ajuster automatiquement les paramètres du régulateur (cf. figure 2.3).

Commande adaptative et système distribué

Le cadre pour l'adaptation donné par la théorie de la commande adaptative est rarement utilisé pour implanter des techniques d'adaptation dans les systèmes distribués. En effet, ce type de système est peu propice à l'application de cette théorie : D'abord, le choix et la mesure d'un indice de performance est difficile. Ensuite, la modélisation des systèmes de

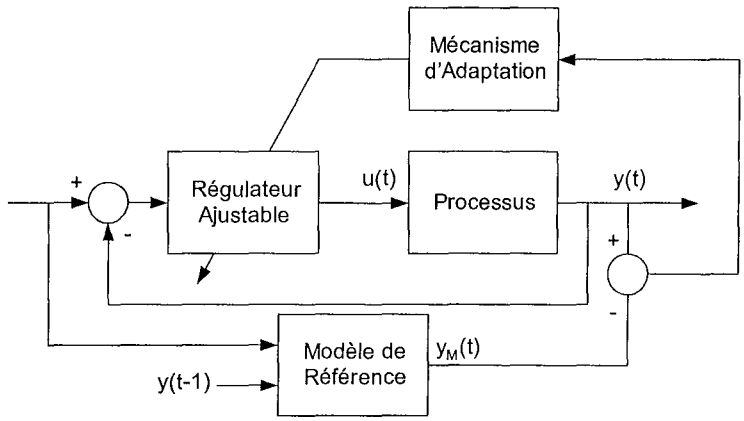


FIG. 2.3 – Commande adaptative avec modèle de référence

bout en bout par des modèles continus est très rarement réaliste. Enfin, **la nature distribuée des systèmes concernés ne permet pas de disposer d'une vue globale instantanée du système.**

Il n'existe donc pas de cadre conceptuel générique pour l'adaptation des systèmes distribués. Par contre, plusieurs techniques spécifiques ont été proposées. Ces techniques d'adaptation se caractérisent par le niveau du système auquel elles sont implantées [FDBC99]. Le paragraphe suivant les présente en les classant selon le niveau auquel elles interviennent.

2.2 Techniques d'adaptation

2.2.1 Adaptation au niveau système (système terminal et système de communication)

2.2.1.1 Contrôle de flux

La technique d'adaptation de niveau transport la plus connue est celle mise en oeuvre dans le protocole TCP où la source module son débit d'émission en fonction des pertes de paquets (pertes inhérentes à la gestion des files d'attente dans les routeurs). Le rôle de cette adaptation du service de transport est d'assurer le contrôle de flux. Le contrôle de flux est un mécanisme dont l'objectif est d'éviter la congestion du réseau. Il est indispensable sur un réseau IP. Notons de plus que l'algorithme de retransmission de TCP est aussi adaptatif. Les valeurs de temporisation utilisées sont déterminées en fonction du délai

aller-retour de la connexion concernée.

2.2.1.2 Choix et/ou adaptation du protocole

Une technique d'adaptation simple consiste à choisir le protocole le plus adapté à chaque communication. Ce choix peut être statique ou dynamique selon l'horizon temporel de travail désiré. Dans le premier cas, le choix est effectué au début et pour toute la durée de la communication. Des exemples de ce type d'adaptation sont donnés dans [Wu97] et [Sou03] où la pile protocolaire est choisie (statiquement) selon les besoins de l'application. Dans le deuxième cas, le choix du protocole peut être remis en cause périodiquement s'il est nécessaire de s'adapter sur des horizons plus courts. Le système pourrait ainsi être capable de changer dynamiquement de protocole selon les variations de QdS du réseau. Certains systèmes d'exploitation traitent les protocoles "en bloc". D'autres sont capables d'agir sur les différents éléments d'un protocole de manière individuelle. La capacité de reconfiguration est donc variable selon les systèmes [FDBC99].

Par exemple, le débit utile d'une connexion peut être amélioré aux dépens de sa fiabilité. Une solution consiste à basculer du protocole TCP au protocole UDP [FG99]. Il est aussi possible d'employer des connexions à fiabilité partielle, ce qui permet de disposer d'une plus grande flexibilité entre les connexions à fiabilité nulle (par exemple UDP) et celles à fiabilité "totale" (par exemple TCP). La connexion est alors dite d'ordre partiel, ou POC (Partial Order Connection) [LL00] [RC00] [CAC94]. Dans [LL00] par exemple, la fiabilité requise pour le transport des paquets est spécifiée par un marquage direct sur les paquets. Cette technique permet alors au récepteur de déceler la fiabilité escomptée pour un paquet perdu et ainsi adopter la réaction la plus adaptée (demande de retransmission ou non,etc.).

2.2.1.3 Ordonnancement des paquets

Une autre stratégie consiste à adapter les techniques d'ordonnancement des paquets de données aux réseaux utilisés : lors de l'emploi de réseaux haut débit pour la transmission continue de flux à forte contrainte temporelle, l'ordonnancement des paquets est directement lié aux risques de congestion et à la capacité de maintenir une QdS. Il est effectivement impératif de transmettre les données dans l'ordre afin d'éviter les stockages en file d'attente. Ceux-ci sont nécessaires aux traitements de remise en ordre des paquets [FDBC99] et pour assurer la continuité du flux. De plus, à haut débit, les buffers se remplis-

sent à grande vitesse. Dans le cas de réseaux RTC, le trafic peut être organisé en utilisant des techniques de *batching* afin de réduire les temps de latence et d'optimiser le coût de la connexion (les données peuvent être ordonnancées pendant les périodes d'inactivité) [SKSZ99]. Finalement, dans les systèmes pour lesquels la bande passante est extrêmement limitée (typiquement les réseaux sans fil), il peut être nécessaire de remettre en ordre les données en fonction de notions de priorité, d'urgence ou de délai d'expiration [FDBC99].

2.2.1.4 Taille des paquets

Un protocole de communication peut aussi s'optimiser par le choix de la taille des unités de données à transférer selon le réseau utilisé [FDBC99]. En effet, la taille des paquets influe directement sur les temps de mise en file d'attente et de traitement par les routeurs. Modifier la taille des paquets est une opération très délicate car selon le principe d'ALF[1] [CT90] l'application doit organiser les données en unités logiques de traitement. Le contenu des paquets impose donc des contraintes relatives à la taille de ceux-ci.

2.2.1.5 Choix du réseau

Dans une configuration où plusieurs réseaux sont disponibles (GSM, WAN, RTC), il peut être intéressant de choisir le(s) réseau(x) le(s) plus adapté(s) aux contraintes QdS associées aux données. Les informations à transmettre sont démultiplexées dans le cas où plusieurs réseaux sont utilisés à la fois, ce qui impose une classification des données pour déterminer celles à envoyer sur chaque réseau [FDBC99].

Les adaptations de niveau protocole sont compliquées à mettre en place : une relation très étroite entre l'application et les couches réseau est impérative. Le protocole doit prendre en compte la nature des informations pour s'adapter. Quand plusieurs technologies sont employées simultanément, une couche supplémentaire est requise afin d'assurer un fonctionnement cohérent des différents protocoles en parallèle.

2.2.1.6 Gestion de réseau

L'adaptation peut aussi intervenir dans le cadre de la gestion de réseau. C'est le cas lorsque les équipements sont reconfigurés en fonction d'indices de performance mesurés sur le réseau (routage dynamique en fonction d'une métrique spécifique par exemple).

[1] Application Level Framing

2.2.1.7 Ordonnancement dans les systèmes d'exploitation

Certaines approches ont proposé des techniques d'ordonnancement des tâches permettant l'adaptation des applications aux ressources disponibles du système terminal (processeur, mémoire, etc.) [OMRW98] [NL97] [JRR97] [BNBH98] [Dem02]. Dans [BNBH98], les applications sont supposées proposer différents niveaux d'exécution. Un gestionnaire de QdS décide du niveau d'exécution qu'une application doit utiliser en fonction de l'utilisation courante du processeur. Dans [JRR97], une analyse du respect des contraintes temporelles de l'application est réalisée a priori. Les applications en sont notifiées et peuvent alors adapter leur comportement en conséquence. Dans ces travaux, l'adaptation n'est pas réalisée dans le système d'exploitation mais au niveau applicatif. Par contre, le système d'exploitation offre des fonctionnalités propices à l'adaptation de l'application.

2.2.2 Adaptation au niveau *middleware*

Les techniques dites de niveau *middleware* consistent à implémenter des services entre les applications et le système. Il existe trois approches majeures [FDBC99]. La première est le filtrage : un service intercepte les données applicatives avant transmission et les transcode ou augmente leur niveau de compression. Cette technique est particulièrement adaptée aux flux continus pour lesquels les compressions avec pertes ou le filtrage des données non essentielles peuvent être utilisés, et permettent de réduire les besoins en bande passante de l'application. Cette technique est utilisée dans [SKSZ99] et [SLA+99] et le concept est déjà proposé dans [DS97]. Dans [FG99] une technique similaire est utilisée, mais les compressions et transcodages sont réalisés par des proxys sur le réseau.

La deuxième approche possible est d'implémenter des services qui stockent et placent dans un cache des données pendant les périodes de bonne connectivité (techniques de *caching*). Les données sont ensuite réordonnées au moment de leur utilisation et pendant les périodes de mauvaise connectivité [SKSZ99].

Enfin, la troisième approche consiste à intégrer au client un proxy local capable d'émuler le serveur dans un mode de fonctionnement dégradé pendant les périodes de mauvaise connectivité [SKSZ99].

2.2.3 Adaptation au niveau application

Dans la plupart des cas, ce sont les applications elles-mêmes qui sont les plus aptes à réagir pour faire face aux changements de QdS du réseau. L'adaptation consiste alors à

ajuster les besoins des applications aux possibilités offertes par le réseau. Ce type d'adaptation est une solution valable et utile lorsqu'une maîtrise complète de la QdS du système sous-jacent de bout en bout est impossible. Cette approche ne fournit pas une QdS aussi élevée à l'utilisateur que les techniques basées sur une garantie totale des ressources.

2.2.3.1 Changement de nature de l'information

Une technique d'adaptation simple consiste à changer la nature de l'information. Si la transmission d'une photographie n'est pas possible, il est peut être envisageable de transmettre une description de la photographie sous forme textuelle (données ASCII, etc.).

2.2.3.2 Modulation du débit

Les techniques d'adaptation les plus connues reposent sur la modulation du volume d'information émise sur le réseau. Elles sont particulièrement adaptées aux cas des médias continus (vidéo, audio) pour lesquels le débit peut être facilement modulé en modifiant les paramètres du média : par exemple, dans le cas d'une application de vidéoconférence, il est possible de baisser le niveau de qualité de l'image au profit de la qualité du son, de diminuer sa taille, de réduire le rafraîchissement, voir même de supprimer l'image et de ne conserver que le son.

Moins radicalement, il est possible de réduire les besoins en bande passante de ce type d'application en modifiant le taux de compression des codecs utilisés [Dio95] [BDS95] [Sis97] ou en changeant de codec [FG99] en fonction de la qualité de réception.

2.2.3.3 Mémoires-tampon de taille variable

Dans le cas des médias continus et plus particulièrement des applications audio, il est souvent impératif de recevoir les informations de façon régulière. La périodicité de réception des données peut être perturbée par la variation du délai (ou gigue) des paquets. Pour s'affranchir des problèmes de gigue, des mécanismes de compensation de délai sont mis en place au niveau du récepteur : les données sont stockées dans des mémoires-tampons avant d'être interprétées [BLLS02]. Ces mémoires-tampon (buffers de play-out) sont dimensionnées de façon à fluidifier le flux d'information et ainsi éliminer les problèmes de gigue. Ce type de mécanismes d'adaptation de mémoires-tampon à la gigue est proposé par exemple dans les applications d'audioconférence vat [Jac93] et Free Phone [VG96].

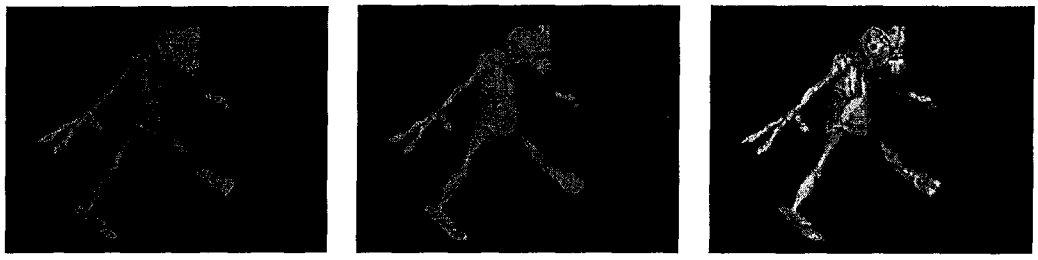


FIG. 2.4 – Codage hiérarchique

2.2.3.4 Codage hiérarchique

Les applications multimédias peuvent utiliser des techniques de codage hiérarchique pour s'adapter : dans ce cas, l'information est codée en plusieurs couches. Une couche de base contient l'information codée à basse qualité. Les autres couches sont facultatives, et complètent la couche de base en lui ajoutant des détails (enhancement layer). Quand le réseau n'est pas chargé, toutes les couches sont transmises, et l'information est reconstruite avec un bon niveau de qualité. Dans le cas contraire, les couches additionnelles sont éliminées du réseau. Par exemple, dans le cas de la transmission d'une scène en trois dimensions, il est possible d'utiliser trois couches (cf. figure 2.4) : la première correspond à la représentation filaire de la scène. La seconde ajoute les volumes à la précédente (représentation "3D simple"). Enfin, la troisième complète les précédentes en ajoutant des textures aux volumes. Quand la bande passante est très limitée sur le réseau, seule la représentation filaire parvient au récepteur, les deux autres couches sont éliminées. Dans des conditions moins extrêmes, seule la troisième couche (les textures) est éliminée et le récepteur dispose d'une représentation en 3D simplifiée. Dans le meilleur des cas, les trois couches sont transmises, et la représentation qui parvient au récepteur est complète (3D en texture). Les techniques d'adaptation basées sur le codage hiérarchique sont particulièrement bien adaptées au multicast. Dans ce contexte, chaque couche correspond à un groupe multicast. Le récepteur peut alors choisir la qualité de réception qu'il désire en rejoignant les groupes correspondants.

2.2.3.5 Techniques plus générales

D'autres approches plus générales proposent des bibliothèques de composants génériques. Un composant générique est une entité logicielle destinée à un traitement particulier (compresseur vidéo, etc.). Les composants incluent des interfaces dédiées à leur paramétrage.

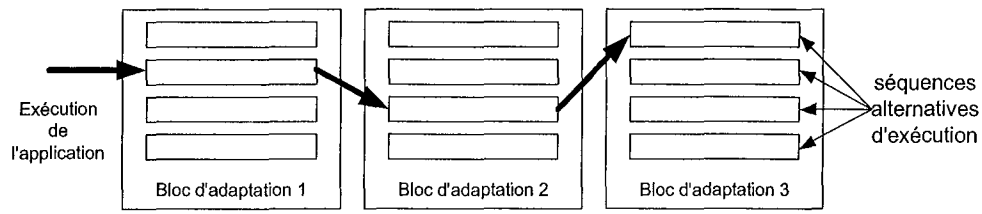


FIG. 2.5 – Blocs d'adaptation

Le développeur construit alors l'application avec ces composants. Le comportement de l'application peut alors être modifié en changeant la configuration de chaque composant (choix des paramètres d'un compresseur vidéo par exemple) [CK99] [BCA+00] [KHP99] [KLM97] [OMRW98] [HvB98].

Le code de l'application peut aussi se décomposer en plusieurs blocs. Un bloc correspond à une tâche particulière de l'application et est associé à un niveau donné de QdS. Il est alors possible de modifier le comportement de l'application en choisissant dynamiquement les blocs à exécuter. L'application propose ainsi plusieurs chemins d'exécution (cf. figure 2.5) [MK98] [BG97] [CK99] et doit fournir les fonctions de transition permettant de basculer d'un chemin d'exécution à un autre [CK99].

Remarque : un mécanisme de ce type est décrit en détail dans le chapitre 4.

Les différentes configurations des composants et les chemins d'exécution correspondent à des profils d'exécution de l'application et sont prévus pour être utilisés selon des profils prédéfinis d'utilisation des ressources [CK99].

2.2.3.6 Stratégie de distribution des tâches

Enfin, dans le cas d'applications distribuées coopératives, la distribution géographique des tâches conditionne l'utilisation des différents chemins employés et des ressources locales des sites distincts [KHP99]. Une technique d'adaptation consiste à modifier la distribution géographique des tâches qui s'exécutent sur les différents systèmes terminaux pour s'adapter à la disponibilité des ressources sur les liens et les systèmes terminaux concernés (une redistribution peut par exemple "alléger" le trafic sur certains tronçons plus sensibles et/ou répartir au mieux la charge des différents systèmes terminaux). Cette adaptation de l'application modifie l'architecture logique de l'infrastructure de communication (arbre multicast par exemple).

2.2.3.7 Remarque

Dans tous les cas, les mécanismes d'adaptation au niveau applicatif doivent être prévus dès la conception de l'application. Mais il s'agit des techniques qui offrent l'adaptabilité la plus souple [FDBC99].

2.2.4 Adaptation au niveau utilisateur

Le niveau final d'adaptation est l'adaptation du comportement de l'utilisateur. L'utilisateur doit être conscient de l'impact de ses actions sur le fonctionnement du système terminal et du système de communication [FDBC99]. Par exemple, lors d'un transfert de fichier, l'affichage d'une barre de progression permet à l'utilisateur de juger s'il souhaite ou non continuer l'opération.

Le système doit être chargé d'informer l'utilisateur des problèmes rencontrés et lui proposer une (des) configuration(s) alternative(s) plus adaptée(s) que celle(s) définie(s) dans ses préférences [DS97].

2.3 Implications

La mise en place des techniques d'adaptation dans le contexte de systèmes distribués implique de nombreuses contraintes. Par exemple, l'objectif étant de s'adapter aux ressources disponibles, il est indispensable de définir des mécanismes de mesure et de surveillance de la disponibilité des ressources. De plus, le comportement de l'application doit convenir aux attentes de l'utilisateur (niveau ultime de QdS). La QdS perçue par celui-ci résulte de la coopération de tous les niveaux du système distribué (application, système d'exploitation et réseau). Il faut donc tenir compte de la QdS à chaque niveau du système, c'est-à-dire des ressources utilisées jusqu'aux préférences de l'utilisateur. Les adaptations des différentes applications qui s'exécutent sur le même terminal doivent être coordonnées, contrairement aux approches les plus courantes, où chaque application réalise individuellement des mesures de QdS et pilote sa propre adaptation. L'absence de coordination peut par exemple poser des problèmes de stabilité des mécanismes d'adaptation ou de compétition dans l'utilisation des ressources partagées (processeur, énergie, etc.).

L'implantation de mécanismes d'adaptation de l'application requiert ainsi une architecture à QdS intégrée. Dans ce type d'architecture, la QdS est prise en compte et spécifiée à tous les niveaux (de l'utilisateur aux couches les plus basses). L'architecture doit éviter à

l'utilisateur et au développeur d'applications de se préoccuper de la complexité de la gestion intégrée de la QdS. L'architecture pilote globalement les adaptations des différentes applications.

L'objectif de cette partie est de passer en revue les contraintes qu'une architecture de QdS doit respecter et les mécanismes qu'elle doit inclure.

2.3.1 La spécification de la QdS

La QdS exprime les niveaux de performance requis par les systèmes distribués. Les performances globales du système dépendent des performances individuelles de chaque niveau du système. Par exemple, les performances du réseau ne déterminent pas à elles seules les performances du système de bout en bout. Dès lors, il est nécessaire de spécifier les besoins et les performances à chaque niveau de l'architecture [NS95b] [OMRW98].

Les architectures proposées dans la littérature ne sont pas toutes "découpées" selon les mêmes niveaux. Dans la suite, on décrit les paramètres pour les niveaux utilisateur, application, système d'exploitation et système de communication.

2.3.1.1 Paramètres utilisateur

L'utilisateur a besoin de connaître les différentes configurations disponibles de l'application. A ce niveau, la QdS est spécifiée en terme de préférences utilisateur. Ces préférences sont un ensemble de paramètres compréhensibles et interprétables par l'utilisateur. Ces paramètres expriment ainsi la QdS perçue par l'utilisateur par une combinaison de paramètres objectifs (taille de l'image) et subjectifs (qualité de l'image). L'utilisateur doit pouvoir aussi préciser ses préférences relatives à l'adaptation. Par exemple, il peut souhaiter privilégier la qualité du son sur celle de l'image dans le cas d'une vidéoconférence si la qualité de la session doit être dégradée. L'utilisateur peut agir sur le système pour l'informer s'il juge le niveau de qualité de la session inacceptable, et si, en conséquence, une nouvelle adaptation est nécessaire.

La nature de ces paramètres étant souvent subjective, la diversité des utilisateurs, des services et des perceptions très grande, la saisie des paramètres utilisateur n'est pas facile et nécessite une interface appropriée.

Les paramètres de QdS utilisateur sont différents selon les applications. Dans le cas d'une application industrielle (régulation d'un processus à distance par exemple), la QdS peut être exprimée par un indice de performance de la régulation (la marge de phase

par exemple). Pour les applications multimédias, la QdS s'exprime par le type de média et les spécifications de fiabilité, temporelles et spatiales du média. Dans les deux cas, la spécification de la QdS peut inclure le coût du service :

- Le type de média indique à l'utilisateur la forme employée pour communiquer l'information (texte, image, etc.).
- La fiabilité spécifie la marge d'erreur acceptable pour le service demandé. Elle s'exprime par exemple en terme d'un nombre maximum d'images non délivrées par unité de temps. Les spécifications temporelles et spatiales expriment des caractéristiques intrinsèques au média : les caractéristiques temporelles sont par exemple le rafraîchissement vidéo, le taux d'échantillonnage, etc. Les caractéristiques spatiales sont par exemple la résolution de l'image, la profondeur des couleurs (bits/pixel), le rafraîchissement (images/seconde), les couleurs (n&b, niveaux de gris, couleur). Des paramètres supplémentaires sont requis pour exprimer la synchronisation intra ou inter-médias. La synchronisation intra-média exprime les contraintes temporelles entre les données qui font partie d'un même média. La synchronisation inter-média exprime les contraintes temporelles entre les données de média différents (synchronisation du son et de l'image par exemple). Tous ces paramètres peuvent être exprimés de façon implicite au travers de critères ou configurations correspondantes à des ensembles de paramètres prédéfinis : intelligibilité, qualité audio (CD, téléphone), rafraîchissement (TV, rafraîchissement réduit, images fixes), etc.
- Finalement, la spécification doit inclure le prix que l'utilisateur est prêt à payer pour le service sinon le niveau maximum de celui-ci est toujours demandé [Cam96]. L'utilisateur doit donc être en mesure de maîtriser son budget. Aussi, si l'algorithme d'adaptation fait intervenir des facteurs coûts (choix entre plusieurs réseaux de communication tarifés, réservation de ressources, etc.), l'utilisateur doit pouvoir fixer des limites [SKSZ99], [SLA+99], [DS97]. Dans [CL99], Cheong indique qu'il est préférable de calculer le coût en fonction des paramètres de QdS et de le présenter à l'utilisateur pour acceptation. L'utilisateur a la possibilité de modifier les paramètres de QdS pour maîtriser le coût. Ceci est plus simple que de demander à l'utilisateur de spécifier un coût et permet ainsi d'éviter la saisie de paramètres contradictoires (qualité maximale et coût minimum).

Saisie des paramètres utilisateur

La saisie des préférences de l'utilisateur et la présentation de la QdS disponible à celui-ci

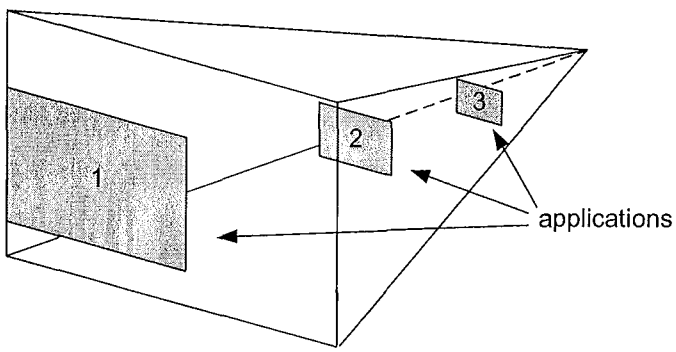


FIG. 2.6 – Interface Cockpitview

sont réalisées par l'intermédiaire d'une interface. Cette interface doit être simple, proposer un nombre limité de paramètres et éviter que son usage ne soit trop complexe [CL99]. Dans [Nah95], l'utilisateur spécifie de façon interactive ses préférences en écoutant/regardant des séquences audio/vidéo et en ajustant des curseurs de contrôle (glissières) pour sélectionner les caractéristiques requises (rafraîchissement par exemple). L'utilisateur ne saisit pas de valeurs numériques. Une interface originale est présentée dans [OMRW98]. Il s'agit de l'interface CockpitView. Elle permet à l'utilisateur de spécifier la QdS implicitement. L'interface CockpitView représente un paysage en trois dimensions. Les différentes applications y sont représentées par des objets. La position d'un objet dans le paysage exprime l'importance accordée par l'utilisateur à l'application concernée : un objet placé à l'horizon est moins important qu'un objet placé au premier plan et requiert donc en conséquence une QdS moins élevée (cf. figure 2.6). Par exemple, si un objet qui affiche une vidéo est placé à l'horizon, sa taille est plus petite et il est possible de réduire la résolution de la vidéo sans perturber la perception de l'utilisateur.

2.3.1.2 Paramètres application

La spécification de la QdS au niveau application est une traduction de la QdS utilisateur en termes quantitatifs. Elle permet d'exprimer les performances attendues de l'application et reçues par celle-ci. Notons que les performances attendues de l'application correspondent à la configuration de l'application. Il peut être difficile de distinguer les paramètres application et utilisateur. Par exemple, la valeur numérique correspondant à la position du curseur de contrôle dans l'interface présentée dans le paragraphe

précédent est un paramètre applicatif alors que sa position visuelle et son effet sur les séquences audio/vidéo données en exemple sont des paramètres utilisateur. Dans le cas d'une application de régulation à distance, le respect du théorème de Shannon (période d'échantillonnage) peut servir d'indice de performance. Pour ce même type d'applications, l'utilisateur peut être amené à configurer l'installation régulée en saisissant des paramètres tels que des valeurs de taux d'échantillonnage ou de gain. Ces paramètres peuvent être considérés comme des paramètres utilisateur et application à la fois.

Au niveau application, la spécification de la QdS correspond à une configuration applicative. Elle est réalisée par le développeur de l'application.

La QdS application est très souvent spécifiée par des caractéristiques du média manipulé par l'application (débit du flux, délai de bout en bout, synchronisation inter/intra média). C'est le cas dans [NS95a] par exemple. Cette spécification correspond plutôt à la QdS attendue du service de transport sous-jacent.

2.3.1.3 Paramètres système d'exploitation

A ce niveau, les ressources mises à contribution sont le(s) processeur(s), la mémoire, les bus d'entrées/sorties, et les périphériques multimédias. Comme la ressource la plus limitée est le processeur, la majorité des recherches sur la QdS dans les système d'exploitation a été effectuée sur les techniques d'ordonnancement des tâches sur le processeur [CL99].

La QdS peut être par exemple spécifiée par un taux d'occupation du processeur [BNBH98] ou une "bande passante processeur" [CL99] qui correspond au pourcentage de la ressource processeur totale. Il est aussi possible d'exprimer la QdS par le temps de traitement sur le processeur alloué périodiquement par un couple (c, t) où c est le temps processeur alloué chaque t seconde. Dans [NS95b], les besoins en ressources processeur d'une tâche et l'allocation des mémoires-tampons sont exprimés par la priorité, le début, la deadline, la durée, la période, etc. L'ordre des tâches est également précisé.

2.3.1.4 Paramètres système de communication

Les paramètres de QdS du système de communication utilisés communément sont la bande passante, le délai, la variation du délai et le taux d'erreurs :

Les applications requièrent la bande passante correspondante au débit du trafic qu'elles génèrent. Une bande passante insuffisante provoque une dégradation de la QdS.

Le délai de bout en bout tient compte des différents délais (codage, paquétisation, propagation, transmission, commutation, temps passé en file d'attente, etc.).

La variation du délai de bout en bout (ou gigue) cause des problèmes de synchronisation entre l'émetteur et le récepteur. La gigue résulte de la mise en file d'attente des paquets dans les nœuds intermédiaires du réseau.

La fiabilité du service de communication est exprimée par le taux de pertes de paquets.

Remarque : ces paramètres sont définis en détail dans le chapitre 3.

[NS95b] propose de spécifier le comportement en rafales du trafic en définissant un paramètre de *burstiness*.

La différence entre la QdS des services réseau et transport est rarement claire dans la littérature. En effet, la QdS de ces services s'expriment souvent par des paramètres identiques (c'est à dire les paramètres cités précédemment). Pourtant, à une valeur de bande passante du niveau transport correspond une valeur différente au niveau réseau. Il faut en effet tenir compte des en-têtes, des tailles des segments de transport et des datagrammes réseau. Il en est de même pour le délai de bout en bout qui tient compte des temps de traitement de chaque couche.

Les spécifications de ces niveaux peuvent aussi différer par les paramètres qu'elles intègrent. On pourra par exemple exprimer la fiabilité du service de transport requis : totalement fiable (TCP, etc.), non fiable (UDP, etc.) ou partiellement fiable (POC, etc.) sans spécifier de fiabilité au niveau réseau (IP).

La figure 2.7 illustre un exemple de spécification de la QdS aux niveaux utilisateur, application, transport et réseau pour une application de vidéoconférence. Ici deux canaux de transport sont utilisés pour l'audio et la vidéo : à un ensemble de paramètres applicatifs correspond deux ensembles de paramètres de transport et de réseau.

2.3.1.5 Remarques

Les applications (et leurs trafics associés) ont des caractéristiques et des contraintes de QdS différents. Les paramètres pertinents de QdS sont différents selon le "type" d'application et les mécanismes d'adaptation proposés. Cela est particulièrement évident au niveau applicatif et reste vrai à toutes les couches de l'architecture : le délai et sa variation sont à priori les paramètres les plus pertinents dans le cas d'une application d'audioconférence alors que la bande passante disponible est le paramètre primordial dans le cas d'une application de vidéo à la demande. Il est donc nécessaire d'adapter la spécification

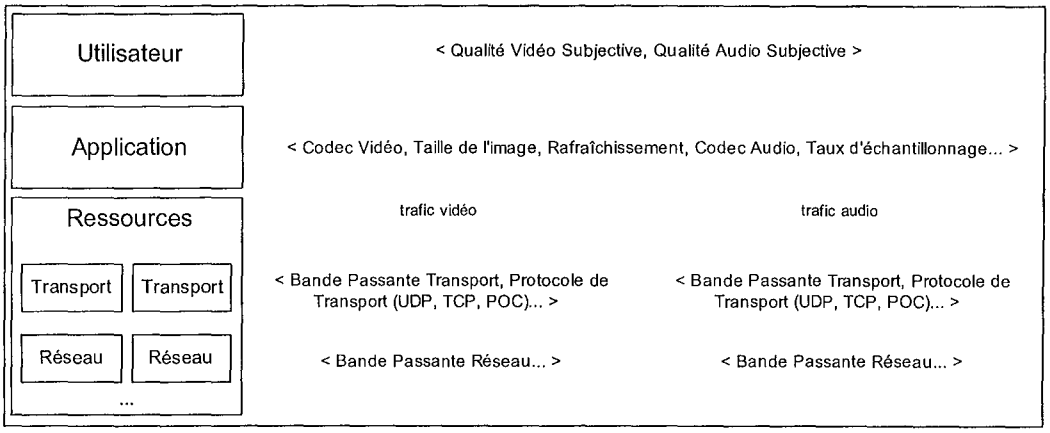


FIG. 2.7 – Exemple de spécification de la QoS pour une application de vidéoconférence

	Taille maximum des messages	Période d'envoi des messages	Taille moyenne des messages	Délai de bout en bout	Bande passante
isochrone	X	X	X	X	
trafic en rafale	X				X
délai faible	X			X	

TAB. 2.1 – Classification d'applications et paramètres associés

de la QdS à la "nature" de l'application. Dans l'hypothèse où on chercherait à fournir des techniques génériques de spécification de la QdS, une classification des applications et des flux est indispensable. Les rares travaux effectués en ce sens concernent les flux, ou du moins caractérisent les applications en fonction des flux qu'elles manipulent. [GP94] propose par exemple de classer les applications multimédias en applications isochrones (à média continu), application à trafic en rafale (transferts de données en rafale, envois non périodiques, blocs de données de tailles variées), et en applications à délai faible (et bande passante faible). Les paramètres de QdS sont différents pour chaque classe d'application (cf. tableau 2.1).

2.3.2 La traduction (translation) de la QdS

Comme expliqué dans le paragraphe précédent, il est nécessaire de spécifier les besoins à chaque niveau de l'architecture. Toutefois, les paramètres de QdS des couches basses (inférieures à la couche application) ne sont pas compréhensibles pour la majorité des utilisateurs et des développeurs. Les besoins de l'application sont souvent difficiles à évaluer pour le développeur [Bao98], [DS97]. Par exemple, les débits de la plupart des applications vidéo sont variables et la spécification d'un taux d'erreurs maximum acceptable est souvent impossible (niveau de redondance de l'information fonction du codec utilisé et du taux de compression potentiellement variable).

Dans ce contexte, il est plus simple d'obtenir suffisamment d'informations de la part de l'utilisateur et/ou de l'application (exprimées respectivement en termes utilisateur et applicatif) et de convertir cette spécification en paramètres de QdS pour les couches sous-jacentes. L'opération de conversion de la spécification d'un niveau à un autre est appelée traduction, *translation* ou *mapping* [Cam96] [NS95a].

Dans les cas les plus simples, cette conversion peut se limiter à copier directement des paramètres à une certaine couche (ex des paramètres à la fois utilisateur et application, cf. 2.3.1.2). Elle peut aussi consister à utiliser des procédures spécifiques (convertir le rafraîchissement, la taille de l'image et la résolution en valeurs de bande passante requise au niveau transport par exemple).

Le service de traduction doit être bidirectionnel [NS95b] : la conversion des paramètres du haut vers le bas de l'architecture permet de déterminer la QdS requise par l'utilisateur à tous les niveaux de l'architecture. Il est aussi nécessaire d'informer l'utilisateur de la QdS disponible (en terme utilisateur) en connaissant la QdS disponible aux niveaux bas de l'architecture. La propriété bidirectionnelle de la traduction est difficile à assurer car cette dernière est souvent ambiguë [CK99] [NS95a]. Par exemple, il est aisé de calculer la bande passante réseau nécessaire au transfert d'une séquence vidéo caractérisée par son rafraîchissement et la taille des images. Lorsque le calcul est difficile (compressions à débit variable par exemple), il est possible d'utiliser des tables de conversion obtenues empiriquement [OMRW98]. La traduction inverse est plus délicate car, à une valeur de bande passante, peut correspondre plusieurs couples de valeurs (taille des images, rafraîchissement). Une solution à ce problème consiste à utiliser des règles de conversion supplémentaires prédéfinies [NS95a].

La traduction utilisateur/application est souvent la plus simple [CL99]. La méthode

la plus commune est de stocker les valeurs qualitatives et quantitatives des paramètres de QdS dans une base de données qui sera utilisée par l'application pour déduire les valeurs quantitatives pour les paramètres utilisateur requis.

Etant donné la diversité des applications et des environnements d'exécution (systèmes d'exploitation et réseaux), il n'existe pas d'algorithme générique de traduction. L'approche proposée dans [CK99] consiste à mesurer les performances de l'application dans toutes les conditions possibles de disponibilité des ressources[2]. De cette façon, une base de données des performances par rapport aux ressources disponibles est construite. Elle permet par la suite d'en déduire les performances de l'application en se basant sur la mesure des ressources disponibles courantes et en interpolant les mesures stockées dans la base.

Un exemple concret de traduction de paramètres application en paramètres transport est donné dans [NS95a]. L'application concernée est une application de télécontrôle d'un robot. Le poste de contrôle déporté envoie des messages de contrôle au robot au travers d'un réseau ATM. Les paramètres de QdS au niveau application sont :

- taux d'échantillonnage R_A =100 Hz,
- taille d'un échantillon M_A =64 octets,
- taux de perte LR_A = 1 échantillon/min,
- délai de bout en bout maximum C_A =20 ms.

Sachant que la taille d'un paquet de transport est 48 octets, les paramètres réseau correspondant sont alors :

- fréquence d'envoi des paquets R_N =200 paquets/s (2 paquets pour un échantillon),
- temps inter-arrivée pour 2 paquets véhiculant le même échantillon inférieur à 5 ms,
- temps inter-arrivée pour 2 paquets véhiculant des échantillons différents supérieur ou égal à 5 ms,
- taux de perte LR_N =1 paquet/min (un échantillon est perdu dès qu'une de ces parties est perdue),
- délai de bout en bout C_N =19,05 ms sachant que le temps nécessaire à l'application pour lire un échantillon est de 0,148 ms, et le temps nécessaire pour lire est de 0,802 ms soit 0,95 ms au total.

[2]En pratique, les performances sont mesurées pour différents scénarios caractéristiques de disponibilité des ressources en utilisant un émulateur d'environnements d'exécution.

2.3.3 La surveillance (métrologie) de la QdS

L'objectif de l'architecture étant de permettre aux applications d'adapter leur exécution à la QdS disponible, des mécanismes de surveillance des ressources sont indispensables. Cette partie présente les niveaux de l'architecture auxquels ces mécanismes peuvent intervenir. Les techniques de surveillance utilisées dans les schémas adaptatifs proposés dans la littérature sont ensuite énumérés.

2.3.3.1 Niveau de surveillance

Les mesures peuvent être obtenues à tous les niveaux de l'architecture. Plus l'information est obtenue à un niveau élevée, moins elle est précise, et moins une coopération étroite entre les couches est nécessaire [BG98].

La surveillance des ressources au niveau application fournit une vision boîte noire du système de communication et du système d'exploitation puisque les mesures sont faites au niveau de l'application (débit, temps inter-arrivée des ADU, etc.). Pour cela, les systèmes terminaux peuvent par exemple s'échanger des informations régulièrement (par l'intermédiaire de RTP/RTCP par exemple, cf. paragraphe 2.3.3.2). Il est impossible de faire la différence des dégradations des performances dues aux ressources réseau de celles dues aux ressources système en utilisant cette approche (par exemple, un récepteur lent peut faire supposer à un émetteur qu'il y a des dégradations des performances du réseau).

Au niveau transport, les algorithmes de contrôle de congestion peuvent être mis à profit pour avoir des informations sur la bande passante puisqu'ils cherchent à adapter le débit le plus justement possible en évitant de sous-utiliser la bande passante et de provoquer des congestions. Ces algorithmes utilisent des informations implicites telles que les pertes de paquets, le délai, etc.

Au niveau réseau, les routeurs peuvent générer des messages pour informer de leur charge. Ces informations peuvent être traitées dans les systèmes terminaux pour déterminer la bande passante disponible.

De la même façon, les ressources disponibles sur le système terminal peuvent conditionner le fonctionnement des applications. Les ressources sont le processeur, la mémoire, les bus d'entrées/sorties, les périphériques multimédias et l'énergie (batterie). Dans [JRR97] et [BNBH98], la mesure des ressources processeur disponibles est utilisée pour notifier à l'application que les ressources requises ne sont pas disponibles et que cette dernière doit s'adapter en conséquence.

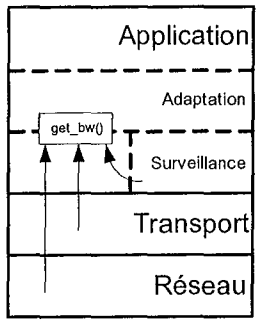


FIG. 2.8 – Mesure de la bande passante aux niveaux application, transport et réseau [BG98] (ici, l'adaptation et la surveillance font partie du niveau application)

2.3.3.2 Techniques de surveillance

Les protocoles Real Time Protocol (RTP) et Real Time Control Protocol (RTCP) [SCFJ96] sont utilisés dans de nombreux mécanismes d'adaptation pour s'informer de la QdS du réseau. C'est le cas des mécanismes d'adaptation du débit [Dio95] [BDS95] [Sis97] et d'adaptation des mémoires-tampon de lecture (vat [Jac93] et Free Phone [VG96]). Ces protocoles ont été conçus pour permettre la synchronisation des applications temps réel.

RTP n'est pas un protocole de transport. En ce sens, il n'offre aucun mécanisme de fiabilité et n'inclut pas de notion de connexion. Il est implémenté comme une partie de l'application. Par contre, RTP fournit aux applications la capacité de distinguer les différents flux d'informations et leurs codages, et aussi de s'informer de la QdS de la session telle qu'elle est perçue par les autres membres de la session. Une session RTP combine deux flux d'informations : un flux de données (un flux audio par exemple) et un flux de paquets de contrôle. Les paquets de contrôle constituent le flux RTCP. Chaque participant à une session RTP diffuse périodiquement aux autres participants un rapport RTCP. Les émetteurs transmettent des rapports d'émission indiquant la quantité de données qu'ils ont envoyées, le type d'encodage utilisé, et précisent par l'intermédiaire d'une information d'horodatage la date à laquelle le rapport a été généré. Les récepteurs envoient pour chaque flux leur parvenant, un rapport de réception. Les rapports d'émission et de réception indiquent directement (ou permettent de déterminer selon le paramètre) le taux de perte, la gigue, le délai aller-retour.

Une autre approche consiste à doter les objets et librairies utilisés pour programmer

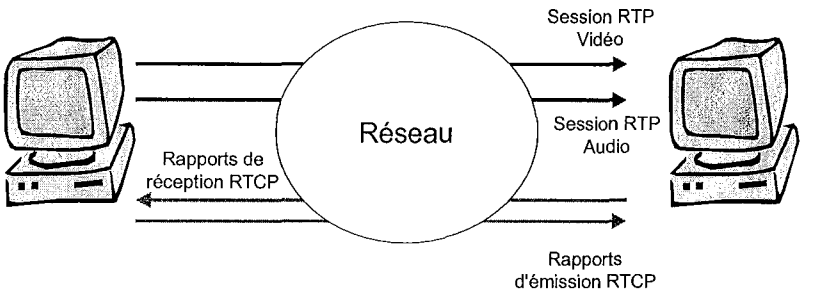


FIG. 2.9 – Sessions RTP / RTCP

les applications d'interfaces dédiées à la surveillance de la QdS. C'est le cas dans [BG98] où l'API des sockets a été étendue par une fonction get_bw() qui retourne une valeur de bande passante calculée en se basant sur la valeur de la fenêtre de congestion de TCP. Dans [CK99] et [KHP99], les objets ou composants utilisés pour le développement des applications incluent des métriques qui mesurent les performances de l'objet ou du composant. Ces métriques sont accessibles via des interfaces dédiées. Sur la figure 2.10, un composant exporte une interface (dite d'accord) qui permet d'accéder aux métriques du composant. Ici, les métriques sont des temps d'exécution internes du composant.

Lorsqu'un système de supervision est mis en œuvre sur le réseau, l'architecture peut obtenir des mesures en interrogeant directement le superviseur du système de supervision ou les agents des différents équipements présents sur la (les) liaison(s) concernée(s). Des règles de concaténation sont alors nécessaires pour déduire la QdS de bout en bout à partir des mesures de chaque liaison [HvB98].

Enfin, des mesures peuvent être réalisées par observation et analyse du trafic utile de l'application (mesure passive). Cette approche nécessite d'avoir une connaissance précise des caractéristiques du trafic de l'application. Lorsque ce n'est pas le cas, il est possible de réaliser des mesures en envoyant des "paquets-sonde" sur le réseau (mesure active). Ce flot de mesure circule sur le réseau entre les deux systèmes terminaux concernés et est supposé apprécier la Qualité de Service de bout en bout telle qu'elle est ressentie par l'application. A son arrivée, le flot est analysé et il est alors possible d'en déduire des métriques (délai, pertes de paquet, etc.). Une étude bibliographique des techniques de mesures actives est présentée dans le chapitre 3.

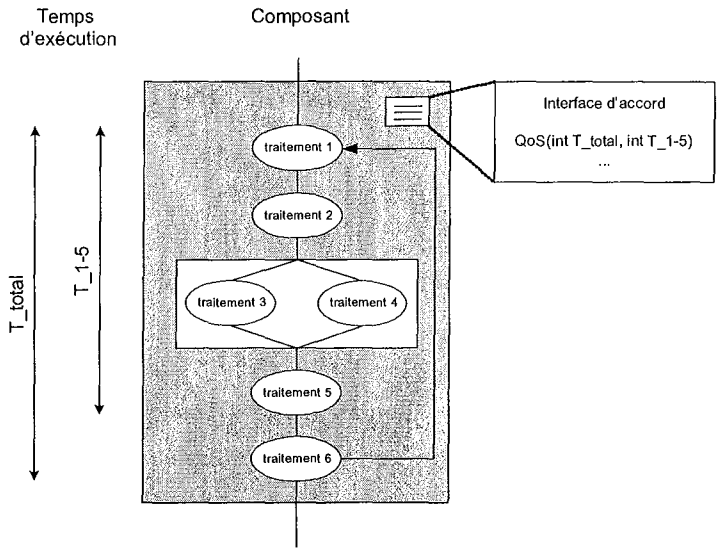


FIG. 2.10 – Une interface du composant permet d'accéder à ses métriques internes (ici des temps de réponse) [CK99]

2.3.3.3 Politique de surveillance

Quelles que soient les techniques de surveillance mises en oeuvre, il est nécessaire de définir la période avec laquelle les mesures sont réalisées. Ce paramètre est directement lié à la période du processus d'adaptation. De plus, selon les mécanismes d'adaptation et les applications, il peut s'avérer utile de définir des mécanismes d'alarme (pour notifier qu'un paramètre a dépassé un seuil important pour l'application).

2.3.4 Le pilotage de l'adaptation

2.3.4.1 Principe de transparence

La complexité de la gestion de la QdS doit être cachée à l'application : les mécanismes précédents de surveillance et de traduction de la QdS, ainsi que le pilotage de l'adaptation de l'application doivent lui être externes et délégués à l'environnement d'exécution (architecture).

En effet, la limitation de l'implémentation de ces fonctionnalités dans l'application a des avantages majeurs : tout d'abord, le travail du développeur s'en trouve facilité [CK99] [BCA+00] [BG97] [KLM97] [LY97]. Celui-ci n'ayant pas à se soucier des mécanismes parfois

complexes de la gestion de la QdS, il n'a par conséquent pas besoin d'avoir des connaissances poussées dans ce domaine. L'environnement de développement doit permettre au développeur de doter l'application de mécanismes pour déclarer ses différentes configurations possibles et les niveaux de QdS associés [CK99] [LY97] [Cam96] en terme applicatif. La traduction de la QdS est effectuée par l'architecture (cf. 2.3.2), le développeur spécifie la QdS uniquement en termes applicatifs sans tenir compte des implications de cette QdS aux niveaux inférieurs de l'architecture [KLM97] [Cam96].

L'application se limite à fournir des fonctions de transition qui permettent à l'architecture de faire basculer l'application d'une configuration à une autre [LY97] sans nécessairement préciser la politique d'adaptation. La politique d'adaptation est l'ensemble des règles qui définissent comment l'application doit s'adapter en fonction de la variation de la QdS. La politique d'adaptation est précisée dans [BG97] par l'intermédiaire de fonctions d'adaptation fournies par l'application. Dans cet exemple, les fonctions d'adaptation indiquent à l'architecture si l'exécution de l'application doit être modifiée ou non, suspendue ou arrêtée. Ces fonctions sont déclenchées dans le cas où la QdS attendue n'est pas disponible, ou au contraire lorsqu'il est possible de bénéficier d'une QdS plus élevée que nécessaire. Par contre, dans [CK99], l'architecture décide seule de la modification de la configuration applicative en se fondant sur une base de données qui l'informent des performances de l'application selon les différentes configurations applicatives et le niveau de QdS disponible.

Finalement, l'externalisation du contrôle de l'application permet à un contrôleur (ou superviseur) de QdS d'avoir une vision globale dans le cas où plusieurs applications s'exécutent sur le même système terminal et d'adapter de façon cohérente l'ensemble des applications. D'une part, les mécanismes d'adaptation peuvent interférer si plusieurs applications utilisent des ressources communes [LWNL98] et des problèmes d'instabilité peuvent se poser [DS97] (cf. 2.3.5). Une vision globale peut s'avérer aussi très utile dans le cas de systèmes distribués très coopératifs. L'exemple d'un tel système est présenté dans [KHP99]. Un gestionnaire assigne les tâches des applications aux différents noeuds de traitement en fonction de leurs ressources disponibles. Dans cet exemple, il est clair qu'une vision globale est nécessaire pour permettre une distribution optimale des tâches des différentes applications sur les nœuds de traitement. D'autre part, une gestion centralisée permet de prendre en compte l'importance relative qu'attache l'utilisateur aux différentes applications. La prise en compte des préférences inter-applications de l'utilisateur permet d'éviter l'utilisation inutile de ressources par des applications peu importantes aux yeux de l'utilisateur,

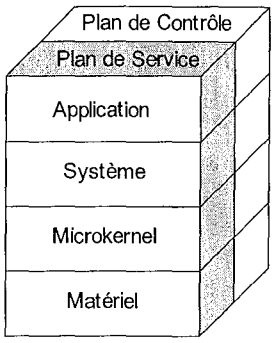


FIG. 2.11 – Principe de séparation dans Padma [KLM97]

ce qui permet à la fois de ne pas pénaliser des applications plus importantes et de limiter le coût dans le cas où les ressources sont tarifées [OMRW98] [LY97].

2.3.4.2 Principe de séparation

La transparence implique la séparation des mécanismes de l'architecture chargées de la gestion de la QdS et du traitement des médias [Cam96]. On retrouve ainsi la séparation des entités et des mécanismes logiciels en deux espaces distincts : un espace de contrôle et un espace applicatif. L'espace de contrôle contient toutes les entités dédiées au contrôle de la QdS. Ces entités sont par exemple des agents de surveillance, gestionnaires de QdS, gestionnaires de ressources, contrôleurs d'admission, etc. L'espace applicatif contient des entités souvent appelées composant ou objet "de flux" ou "de média" qui sont consacrées au traitement de l'information utile de l'application (flux vidéo, audio, etc.).

Par exemple, l'architecture padma [KLM97] définit deux plans : le plan de service (correspond à l'espace applicatif) et le plan de contrôle (cf. figure 2.11). Chaque plan inclut des opérations qui lui sont propres. Un exemple d'opération de service dans une application de transmission vidéo est "transmettre l'image suivante". Par contre, l'opération "changer le rafraîchissement" est une opération de contrôle.

L'interaction entre les deux espaces (ou plans) est réalisée au travers d'interfaces spécifiques des objets utilisés pour programmer l'application. Dans [KLM97], les objets ont deux types d'interface : l'une pour accéder à leurs fonctions de service (interface de base) et l'autre pour contrôler leur comportement (fournit l'accès aux paramètres de contrôle, aux métriques de QdS et aux fonctions de transition de l'objet par exemple). Cette dernière

interface est appelée interface de contrôle ou encore "interface d'accord" dans [CK99]. Dans [OMRW98], les objets ne font pas partie des deux espaces à la fois. Par contre, à chaque entité de l'espace applicatif correspond une entité dans l'espace de contrôle (cf. figure 2.12).

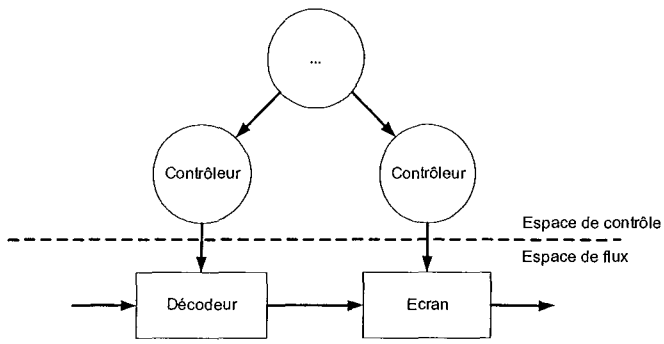


FIG. 2.12 – Les objets de l'espace de contrôle et de l'espace de flux inter-opèrent via des interfaces dédiées

En pratique, les entités de l'espace de contrôle sont implémentées par des librairies spécifiques liées au code de l'application [LY97]. Les entités et les interfaces de contrôle sont cachées au développeur. De cette façon, les développeurs d'applications n'utilisent que les objets et/ou les interfaces de l'espace applicatif alors que les concepteurs des librairies d'objets et de l'architecture doivent prévoir les deux types d'interface et/ou d'objets [KLM97].

2.3.5 Stabilité de l'adaptation

Comme tout processus dynamique, le mécanisme d'adaptation peut être confronté à des problèmes de stabilité [BNBH98]. Le cas d'une application audio est présenté dans [BK99]. Pour cette application, le choix du codage audio est conditionné par la bande passante disponible. Pour une bande passante inférieure à $48kb/s$, le codage "A" est utilisé. Lorsque la bande passante est supérieure à $48kb/s$, le codage "B" est choisi. Des problèmes d'instabilité peuvent survenir quand la valeur de la bande passante oscille autour du seuil utilisé pour le choix. Ce problème d'instabilité est illustré sur la figure 2.13 (R : bande passante disponible (b/s), flow rate : débit du flux généré par l'application, $Adm6\_p\_lo$: seuil de décision entre les deux codages). La bande passante disponible (courbe R) oscille

autour de $48kb/s$, l'algorithme ne cesse de commuter entre les deux codages (phénomène représenté par le débit en sortie de l'application, courbe en gras "flow rate") et l'oscillation est auto-entretenue : en commutant du codage B au codage A, l'application baisse son débit. De ce fait, la bande passante disponible augmente, ce qui a pour effet de refaire commuter l'application sur le codage B. En commutant sur le codage B, l'application augmente son débit, et la bande passante disponible diminue. L'application bascule à nouveau sur le codage A, etc.

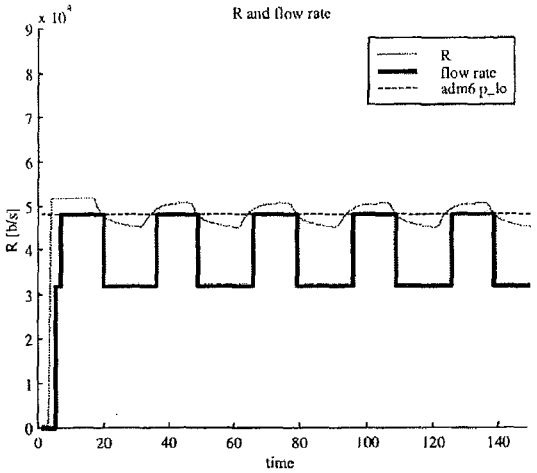


FIG. 2.13 – Instabilité dans le choix du codage audio [BK99]

D'autres problèmes peuvent survenir quand plusieurs applications adaptatives utilisent des ressources communes. C'est le cas des mécanismes d'adaptation du débit. Dans [DS97], il est montré qu'une concurrence entre les connexions adaptatives se produit et que cela conduit les débits utilisés par chaque application à osciller (une compétition entre ces connexions et des connexions TCP peut aussi se produire [Sis98]). Quand les applications adaptatives sont exécutées sur la même station, le pilotage des adaptations par un superviseur peut remédier à ce type de problèmes [LWNL98] en introduisant par exemple une politique de supervision des connexions.

2.3.6 Mécanismes de contrôle

L'objectif des mécanismes de contrôle de la QdS est de s'assurer que les flux générés par les applications respectent les spécifications annoncées, le plus souvent en terme de

débit. Ils sont surtout indispensables lorsque des réservations de ressources sont faites (ce qui n'est pas le cas dans nos travaux) car ces dernières sont garanties sous réserve de la conformité du flux à des spécifications précises de débit moyen et de débit crête (on parle alors de "traffic shaping").

Ces mécanismes ont aussi un intérêt dans les schémas adaptatifs. Dans le cas des mécanismes d'adaptation de débit par exemple, ils peuvent servir de boucle de retour pour contrôler le volume des flux générés en boucle fermée.

2.4 Conclusion

Dans ce chapitre, nous avons constaté qu'il n'existe pas de théorie générale pour l'adaptation des applications distribuées. Cependant, de nombreuses techniques d'adaptation propres ont été proposées dans la littérature, et nous les avons passées en revue. Par ailleurs, nous avons expliqué que la bonne intégration de ces techniques dans les systèmes distribués requiert différents mécanismes spécifiques que nous avons détaillés, au sein d'une architecture à QdS intégrée.

Parmi ces mécanismes, nous avons choisi d'approfondir l'étude de la métrologie de la QdS dans le chapitre suivant. Nous y définissons les paramètres essentiels de "QdS réseau" et dressons un état de l'art des techniques et des outils de mesure qui leur sont associés.

Chapitre 3

Méthodes et outils en métrologie réseau

3.1 Introduction

La métrologie réseau est la science de la mesure des performances du réseau. C'est une discipline récente. Les premiers travaux significatifs et aujourd'hui encore de référence ont été menés par Paxson [Pax97] au milieu des années 90 (des travaux avaient été conduits précédemment sur les lignes de télécommunications ou les réseaux à commutation de circuits et avaient pour objectif de déterminer des critères de performance similaires tels que le délai par exemple. Mais les différences conceptuelles et techniques entre ces réseaux et les réseaux à commutation de paquets impliquaient une nouvelle approche métrologique). Paxson a développé une infrastructure de mesure qui lui a permis de capturer puis d'analyser 20.000 traces de connexions TCP entre 35 sites repartis sur 9 pays. L'analyse des traces a été couplée à une analyse de routes de bout en bout (déterminées avec l'utilitaire traceroute, cf. 3.2.6). Ces études ont permis d'observer et de mieux comprendre pour la première fois le comportement dynamique du réseau : stabilité du routage, asymétrie des routes, comportement de TCP, livraison désordonnée des paquets, etc.

Depuis, différents projets et laboratoires de recherche ont vu le jour (Surveyor [Sur], NIMI[1] [NIM] [PAM00], RIPE[2] [RIP], Netsizer [Net], CAIDA[3] [CAI], SPRINT/IPMON

[1] National Internet Measurement Infrastructure

[2] Réseaux IP Européens

[3] Cooperative Association for Internet Data Analysis

[Owe01], METROPOLIS[4] [Rés01]). Certains projets n'ont pas d'objectif très précis (alimenter des bases de données de mesures, mesurer la croissance d'Internet, etc.). Pour les autres, les travaux sont axés notamment sur la caractérisation du trafic, sa modélisation, l'étude des matrices de trafic et la cartographie du réseau : la caractérisation du trafic consiste à déterminer le volume du trafic qui circule sur le réseau, sa variabilité en fonction du temps et sa nature. La nature du trafic est sa répartition par protocole (TCP, UDP, etc.) et par application (web, mail, multimédia temps-réel). On prend aussi en compte la taille des paquets, le nombre de flux simultanés, la taille des flux et leur composition. La modélisation du trafic a pour objet la détermination de modèles des arrivées des flux et/ou des paquets et des pertes de paquets. Les matrices de trafic sont utilisées pour étudier les routes empruntées par les paquets dans le réseau. Ces études s'intéressent par exemple à la dynamique du routage et à la symétrie des routes. Finalement, certains projets tentent de dresser des cartes de l'Internet en détectant tous les hôtes du réseau.

Toutes ces études ont des répercussions importantes [Owe01] :

- Pour l'opérateur : la caractérisation et la modélisation du trafic permettent un dimensionnement du réseau plus facile.
- Pour les équipementiers : la répartition de la taille des paquets influe sur la conception des routeurs par exemple.
- Pour la conception des protocoles : une meilleure connaissance de la dynamique des flux, des délais et des dépendances entre les pertes est indispensable à la conception de protocoles efficaces (amélioration de TCP, etc.).

3.1.1 Classes de mesures

Sur un réseau, les mesures peuvent être réalisées de façon active ou passive. Ces deux techniques diffèrent par leur principe de mise en œuvre, les problématiques qu'elles induisent et l'utilisation potentielle des résultats qu'elles fournissent. Les projets de recherche cités précédemment se classent dans l'une ou l'autre catégorie : Surveyor, NIMI, RIPE et Netsizer se basent sur des mesures actives, CAIDA et SPRINT/IPMON sur des mesures passives.

[4] METROlogie POur L'Internet et ses Services

3.1.1.1 Mesures passives

Les mesures passives sont effectuées en observant le trafic qui circule sur le réseau (trafic utile). Elles consistent à capturer les en-têtes des paquets et à analyser celles-ci. Elles n'ajoutent pas de trafic sur le réseau.

Les mesures passives peuvent être réalisées à deux niveaux :

- Au niveau *microscopique*, les mesures sont faites pour chaque paquet qui passe au point de mesure. La taille des paquets est une des informations qu'il est possible de collecter.
- Au niveau *macroscopique*, les mesures sont réalisées pour des flux. Il s'agit de mesures agrégées : Il faut définir des règles d'agrégation des paquets en flots. Ces règles conditionnent les résultats obtenus. Il est alors possible de déterminer le nombre de flots par unité de temps, le débit par flot, etc. Les mesures macroscopiques ont constitué le sujet d'étude du groupe de travail "Realtime Traffic Flow Measurement" (RTFM) de l'IETF[5]. Ce groupe a défini une architecture générale de mesures passives et a décrit la façon de définir les règles d'agrégation. Aujourd'hui, ce groupe a cessé ses activités.

Un problème important posé par les mesures passives est celui du volume des données capturées. En effet, la quantité de données capturées peut rapidement devenir colossale sur des liens à haut débit. Un autre inconvénient est lié à leur aspect local. Les mesures sont faites en plusieurs points de mesure du réseau et il est difficile de recouper les résultats issus de points de mesure différents.

Les techniques passives sont particulièrement adaptées à l'ingénierie du trafic puisqu'elles montrent la distribution et le comportement des flux qui circulent sur le réseau.

L'outil de capture du trafic le plus connu est Tcpdump [Tcp]. Cet outil ainsi que la plupart des outils de capture, utilise la librairie Libpcap [Tcp].

3.1.1.2 Mesures actives

Les mesures actives ont pour objectif de déterminer la QdS de bout en bout telle qu'elle est ressentie par l'application. Elles sont réalisées en générant un flot de mesure ("paquets-sonde"). Ce flot circule sur le réseau entre une source et une destination. A son arrivée, il est possible de calculer des métriques en l'analysant.

[5] Internet Engineering Task Force

Le groupe IP Performance Metric (IPPM) a été créé au sein de l'IETF. Ce dernier s'est fixé pour objectif la définition de standards de métriques et de statistiques afin de permettre entre autres la comparaison de mesures actives effectuées par des outils différents, celle-ci étant le plus souvent impossible. L'IPPM a produit des RFC notamment sur la mesure de métriques et le calcul de statistiques relatives à plusieurs paramètres (taux de pertes de paquets, délai, etc.). Il préconise un processus d'envoi des paquets-sonde poissonien ou périodique [PAMM98] [RGM02], le deuxième étant à priori plus adapté à la mesure de la QdS pour des applications multimédias à flux continus.

A noter aussi que l'IPPM travaille d'une part à la définition d'une MIB[6] qui inclut les métriques standards, et d'autre part à la spécification d'un protocole qui permettrait à des équipements de mesure de différents fabricants d'inter-opérer pour faire des mesures ou se les communiquer. L'IPPM demeure un groupe actif et poursuit sa démarche de standardisation.

L'inconvénient majeur des mesures actives est leur "intrusivité" sur le réseau (manque de finesse de la mesure). Il est donc nécessaire de limiter le volume du trafic de mesure afin de ne pas perturber le réseau. Or, aucune règle n'a été définie pour limiter le trafic de mesure jusqu'à présent. Ainsi ce problème reste ouvert.

Les relations entre les mesures actives et passives restent inconnues et il est impossible d'étendre la QdS observée par le flot de mesure à d'autres flots. Les mesures actives ne sont donc pas adaptées à priori à l'ingénierie du trafic.

3.1.2 Objectifs du chapitre

Ce chapitre présente les paramètres essentiels de "QdS réseau" : bandes passantes totale, minimale et disponible, délai unidirectionnel, variation de délai, délai aller-retour, pertes de paquets et route. Pour chaque paramètre, un état de l'art des techniques et des outils de mesure associés est exposé[7]. Notre motivation étant de réaliser des mesures dans l'objectif d'adapter les applications, cette étude concerne essentiellement des techniques de mesure actives.

[6] Management Information Base

[7] La plupart de ces outils sont repertoriés et téléchargeables via le site du laboratoire CAIDA [CAI].

3.2 Les paramètres

3.2.1 Bande Passante

L'utilisation du terme bande passante est un abus de langage issu du théorème de Shannon car il s'agit ici d'une capacité de transfert. Le terme "bande passante" désigne plus précisément les quatre paramètres "bande passante totale d'un lien", "bande passante minimale d'un chemin", "bande passante disponible d'un lien" et "bande passante disponible d'un chemin" dont nous donnons les définitions dans le paragraphe suivant.

3.2.1.1 Définitions (basées sur les définitions de [JD02b])

Soit deux systèmes terminaux, un système source SRC et un système destination DST. Soit P le chemin entre ces deux systèmes. P est la séquence de liens entre SRC et DST. Supposons P fixe et unique (pas de changement de routes ou de chemins multi-route entre SRC et DST).

- La bande passante totale du lien i, notée C_i, définit la capacité totale de transmission du lien i.
- Soit C la bande passante minimale du chemin P. Si H est le nombre de nœuds de P, alors la bande passante minimale du chemin est :

$$C = \min_{i=1...H} C_i$$

- Supposons que le lien i transmette $C_i u_i$ bits durant l'intervalle de temps T. u_i est le taux d'utilisation du lien i durant T, avec $0 \leq u_i \leq 1$. La bande passante disponible A_i du lien i est

$$A_i = C_i(1 - u_i)$$

- La bande passante disponible A du chemin P durant l'intervalle de temps T est le minimum des bandes passantes disponibles de tous les liens du chemin P :

$$A = \min_{i=1...H} \{C_i(1 - u_i)\}$$

3.2.1.2 Mesure de la bande passante totale sur un chemin

Cette partie présente plusieurs méthode de mesure de la bande passante totale. Pour des raisons de place, nous ne détaillerons que la première technique qui est utilisée par la majorité des outils existants.

Pathchar [Jac97], Bing [Bey95], Clink [Dow99] et Pchar [Mah99] mesurent la bande passante totale de chaque lien d'un chemin en utilisant la technique appelée *one-packet technique.* Celle-ci se base sur l'hypothèse que le délai d'un paquet varie linéairement en fonction de sa taille. Elle consiste alors à mesurer le délai aller-retour pour atteindre chaque routeur présent entre une source et une destination. Ce délai aller-retour est mesuré par la source en envoyant des paquets-sonde à chaque routeur. Le routeur à atteindre est déterminé en fixant la valeur du TTL[8] des paquets-sonde. Le champ TTL est prévu pour éviter qu'un paquet ne tourne indéfiniment dans le réseau. Il correspond à la durée de vie du paquet exprimée en nombre de routeurs traversés[9]. Il est décrémenté d'une unité lors du passage du paquet dans un routeur. Lorsque la valeur obtenue est égale à zéro, le routeur supprime le paquet et en informe son émetteur en lui envoyant un paquet ICMP[10] *time exceeded* (cf. figure 3.1).

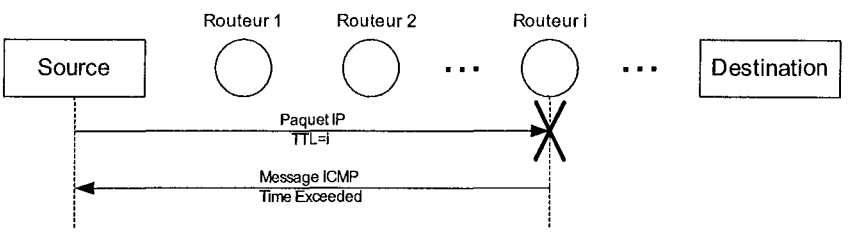


FIG. 3.1 – Champs TTL

La figure 3.2 illustre la relation entre le délai aller-retour et la bande passante totale, entre le système terminal source et le premier routeur du chemin. Le système source envoie un paquet-sonde UDP avec un TTL égal à 1. Arrivé au premier routeur, ce paquet est supprimé et le routeur retourne au système terminal un message ICMP *time exceeded.* Le délai aller-retour T_1 est égal à :

$$T_1 = \frac{s}{b_1} + d_1 + t + w$$

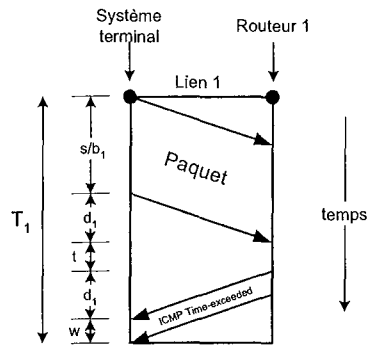
La technique du *one-packet* fait les hypothèses suivantes :

- Le temps t qui s'écoule entre la réception du paquet-sonde par le routeur et le renvoi du message ICMP est négligeable.

[8] Time To Live

[9] Initialement, le TTL était une durée exprimée en secondes.

[10] Internet Control Message Protocol



s : taille du paquet
b_1 : bande passante totale du lien 1
d_1 : délai de propagation du lien 1
t : temps de traitement
w : temps de transmission du paquet ICMP
Hypothèses : t et w négligeables
On a $T_1 = \frac{s}{b_1} + d_1 = \frac{s}{b_1} + C_1$
avec C_1 une constante

FIG. 3.2 – Délai aller-retour et bande passante totale entre le système source et le premier routeur

- Le temps de transmission du message ICMP est négligeable en raison de sa petite taille.

On a alors :

$$T_1 = \frac{s}{b_1} + d_1 = \frac{s}{b_1} + C_1$$

Remarque : Le délai de propagation est égal à la longueur physique du lien divisé par la vitesse de propagation du médium. $d_1 = C_1$ est donc une constante.

La figure 3.3 page 42 illustre la mesure de délai aller-retour entre le système terminal et le deuxième routeur. En faisant les mêmes hypothèses que précédemment, il vient :

$$T_2 = \frac{s}{b_1} + \frac{s}{b_2} + d_1 + d_2 = \frac{s}{b_1} + \frac{s}{b_2} + C_2 \text{ avec } d_1,\ d_2 \text{ et } C_2 \text{ constantes}$$

Plus généralement, l'expression du délai aller-retour entre le système terminal source et le routeur i du chemin est :

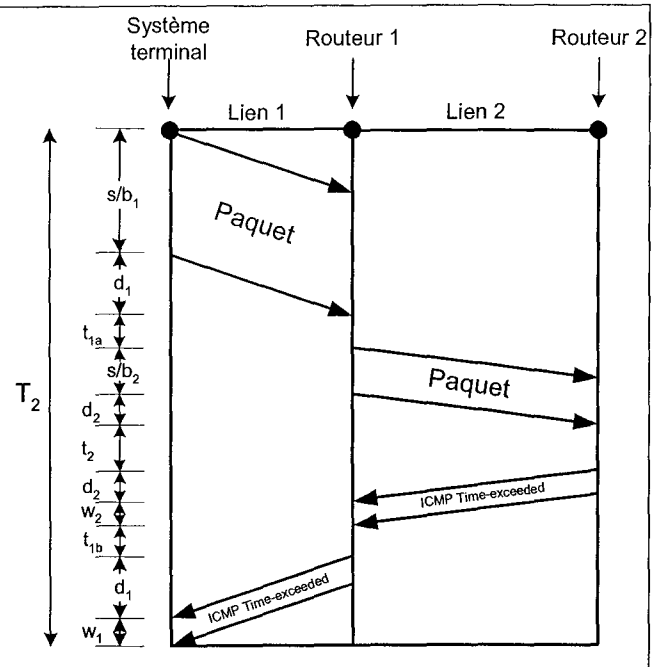
$$T_i = s \sum_{j=1}^{i} \left(\frac{1}{b_j}\right) + C_i \text{ avec } C_i \text{ constante}$$

L'objectif est de déterminer les b_i. On pose $k_i = \sum_{j=1}^{i} \left(\frac{1}{b_j}\right)$.

On a alors $k_i - k_{i-1} = \frac{1}{b_i}$, soit

$$b_i = \frac{1}{k_i - k_{i-1}}$$

k_i est le coefficient directeur de la droite $T_i = f(s)$, droite qui est obtenue en envoyant des paquets-sonde UDP de tailles différentes au routeur i et en mesurant les délais aller-retour (T_i) correspondants.



s : taille du paquet

b_i : bande passante totale du lien i

d_i : délai de propagation du lien i

avec t_{1a}, t_{1b} et t_2 : temps de traitement

w_1 et w_2 : temps de transmission des paquets ICMP

Hypothèses : t_{1a}, t_{1b}, t_2, w_1 et w_2 négligeables

On a $T_2 = \frac{s}{b_1} + \frac{s}{b_2} + d_1 + d_2 = \frac{s}{b_1} + \frac{s}{b_2} + C_2$

avec C_2 une constante

Plus généralement, $T_i = s \sum_{j=1}^{i} \left(\frac{1}{b_j}\right) + C_i$

avec T_i le délai aller-retour jusqu'au routeur i

et C_i une constante

FIG. 3.3 – Principe de la mesure de la bande passante totale

En répétant cette procédure pour tous les routeurs et jusqu'au système destination, on obtient les k_i pour i de 1 à H (H est le nombre de liens) et il est alors possible de calculer la bande passante totale de chaque lien (b_i).

Les outils précédents (dits *pathchare-like*) se distinguent essentiellement par leurs algorithmes de filtrage des résultats. A noter que, contrairement aux autres, Bing considère que le chemin et les liens empruntés entre la source et la destination sont parfaitement symétriques. Des études et des explications sur le fonctionnement de Pathchar et des propositions d'amélioration des algorithmes de mesure et de filtrage des résultats sont disponibles dans [Dow99], [Jia99] et [PV02].

- [Jia99] étend la méthode à la mesure de la bande passante totale dans les deux sens des liens, ce qui est particulièrement utile lorsque les liens sont asymétriques, c'est-à-dire lorsque la bande passante totale est différente dans les deux sens d'une liaison. L'algorithme proposé fait l'hypothèse que la route est symétrique (mêmes liens employés à l'aller et au retour).
- [PV02] propose d'utiliser la variation du délai plutôt que le délai pour effectuer les calculs. Cela a pour effet de diminuer le nombre de sondes à envoyer sur le réseau.

La figure 3.4 montre le type de résultats obtenus avec Pathchar. Les lignes numérotées correspondent aux différents noeuds du chemin, la bande passante totale est indiquée dans les lignes intermédiaires (lignes qui débutent par "|"). Par exemple, la bande passante totale mesurée du lien qui relie le système source (noeud 0) au premier routeur (noeud 1) est égale à 9,3 Mb/s.

Une technique de mesure alternative est introduite dans Nettimer [LB00]. Il s'agit de la technique dite de "talonnage" (*tailgating*). Elle étend la technique du *one-packet* présentée précédemment et celle de la paire de paquets (cf. 3.2.1.4). Dans ce cas, on envoie plusieurs paquets successivement avec un temps inter-paquets le plus court possible. Il existe alors des relations entre les temps passés par chaque paquet en file d'attente. Si on fait l'hypothèse qu'il est possible de forcer un paquet à être mis en file d'attente derrière son prédécesseur dans un routeur particulier et pas dans les routeurs suivants, il vient alors une expression relativement simple de la bande passante totale de chaque lien du chemin. En pratique, la source envoie deux paquets-sonde avec un temps inter-paquets le plus court possible. Le premier paquet, dit le "talonné" (*tailgated*), a une taille maximale de façon à ce que le deuxième, le "talonneur" (*tailgater*), soit placé en file d'attente sur un routeur ciblé. Afin que le talonné ne provoque pas la mise en file d'attente du talonneur sur

```
riesling ~ 79% 14:24: pathchar ka9q.ampr.org
pathchar to ka9q.ampr.org (129.46.90.35)
 mtu limitted to 1500 bytes at local host
 doing 32 probes at each of 64 to 1500 by 44
 0 192.172.226.24 (192.172.226.24)
 |    9.3 Mb/s,   269 us (1.83 ms)
 1 pinot (192.172.226.1)
 |     85 Mb/s,   245 us (2.46 ms),  1% dropped
 2 sdscdmz-fddi.cerf.net (198.17.46.153)
 |     45 Mb/s,   -13 us (2.70 ms)
 3 qualcomm-sdsc-ds3.cerf.net (134.24.47.200)
 |    8.8 Mb/s,   1 us (4.07 ms)
 4 krypton-e2.qualcomm.com (192.35.156.2)
 |    5.2 Mb/s,   1.02 ms (8.42 ms)
 5 ascend-max.qualcomm.com (129.46.54.31)
 |   53.2 Kb/s,   4.20 ms (243 ms)
 6 karnp50.qualcomm.com (129.46.90.33)
 |     12 Mb/s,   -172 us (243 ms),  +q 8.96 ms (13.0 KB) *3,  6% dropped
 7 unix.ka9q.ampr.org (129.46.90.35)

7 hops, rtt 11.1 ms (243 ms), bottleneck 53.2 Kb/s, pipe 4627 bytes
```

FIG. 3.4 – Résultats obtenus avec Pathchar [Jac97]

les routeurs suivants, son TTL aura préalablement été fixé de façon à ce qu'il soit détruit sur le routeur concerné (cf. figure 3.5). Le talonneur est un message TCP FIN. De cette façon, la destination est contrainte à retourner un message TCP RESET. Cela permet de mesurer le délai aller-retour.

Plusieurs variantes et améliorations de cette technique sont proposées dans [PV02] (*packet quartets*). Elles évitent le recours à certains filtrages des mesures, suppriment des phases de mesure et optimisent ainsi le nombre de paquets-sonde à envoyer sur le réseau.

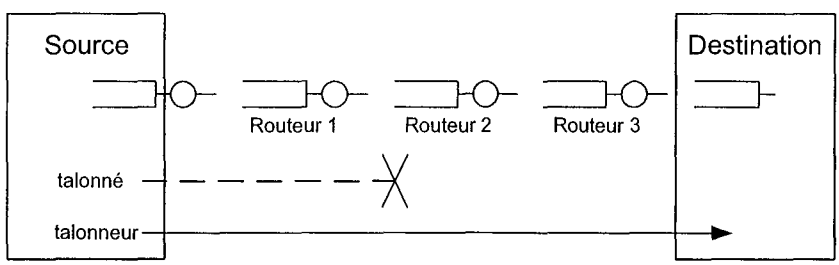


FIG. 3.5 – Technique du "talonnage"

```
jawks [507]$ ./pathload_rcv -s sirius
Receiver jawks starts measurements on sender sirius at Sun Apr 6 15:29:44 2003
Receiving Fleet 0, Rate 97.40Mbps
Receiving Fleet 1, Rate 193.55Mbps
Receiving Fleet 2, Rate 145.47Mbps
Receiving Fleet 3, Rate 121.44Mbps
Receiving Fleet 4, Rate 109.42Mbps
Receiving Fleet 5, Rate 103.41Mbps
***** RESULT *****
Available bandwidth range : 97.40 - 103.41 (Mb/s)
Measurements finished at Sun Apr 6 15:29:44 2003
Measurement latency is 5.69 sec
```

FIG. 3.6 – Résultats obtenus avec Pathload [Dov]

3.2.1.3 Bande passante disponible

La technique de mesure utilisée par Cprobe [CC96] est la suivante : une courte rafale de paquets est envoyée au destinataire. Le destinataire mesure Δ le temps écoulé entre la réception du premier et du dernier paquet. La bande passante disponible est le quotient de la quantité binaire de données envoyées q par le temps mesuré Δ.

$$A = \frac{q}{\Delta}$$

Pipechar [JYCA01] se base sur une approche similaire. Cette technique est intrusive. De plus, il a été montré dans [DRM01] qu'elle ne fournit pas des résultats corrects.

Pathload [JD02b] utilise la technique dite des *Self Loading Periodic Stream* (SLOPS) : Une source envoie des rafales de paquets à une destination. Le destinataire mesure le délai de chaque paquet des rafales et analyse sa variation. Si le délai est jugé constant, on en déduit que le débit de la rafale est inférieur à la bande passante disponible. Si le délai est croissant, alors le débit de la rafale est supérieur à la bande passante disponible. On envoie alors une seconde rafale à un débit supérieur dans le premier cas ou inférieur dans le second cas. Ce mécanisme est répété, et on approche par dichotomie la valeur de la bande passante disponible. Cette technique est à priori intrusive, puisqu'elle amène la source à saturer le réseau. La figure 3.6 illustre le type de résultats obtenus avec pathload. Les lignes qui débutent par "Receiving Fleet" indiquent le débit des rafales utilisées. La bande passante disponible est estimée entre 97,40 et 103,41 Mb/s. La mesure a été réalisée en 5,69 secondes.

Iperf [Ipe] mesure la bande passante maximale qu'obtiendrait une connexion TCP. La mesure est réalisée en mettant en place la connexion TCP et en essayant de transmettre des données à un débit le plus élevé possible. Cette technique est donc extrêmement intrusive.

Nous citons encore MRTG (Multi Router Traffic Grapher) [OR] car il est couramment utilisé par les administrateurs réseaux. MRTG permet d'interroger les routeurs et de connaître précisément leur charge (taux d'utilisation de leurs interfaces) en s'appuyant sur SNMP. Connaissant la bande passante totale de chaque lien, il est alors facile d'en déduire la bande passante disponible. MRTG génère des pages HTML qui affichent les résultats sous forme de graphiques. La granularité de mesure est limitée à celle qu'offre SNMP, habituellement 5 minutes. Pour connaître la bande passante disponible entre deux systèmes, il est donc indispensable de disposer de la liste des routeurs traversés et de bénéficier de droits d'accès en lecture aux MIB de ces routeurs.

La mesure de la bande passante disponible est en cours de discussion à l'IETF. Un draft récent [ASU+] propose une technique de mesure de ce critère en se basant sur la mesure de la fenêtre de congestion de TCP. Cette technique n'est donc pas intrusive. De plus, ce draft propose l'emploi d'un algorithme de prédiction de la bande passante disponible à venir. Il s'agit de l'algorithme Abest détaillé dans [ASC+02].

3.2.1.4 Bande passante minimale

La bande passante minimale est mesurée en utilisant la technique dite de la paire de paquets. Cette technique repose sur le phénomène "d'écartement" ou "dispersion" subi par deux paquets consécutifs suite à leur passage dans le goulet d'étranglement : l'espace temporel qui les sépare est accru (phénomène mis en évidence par Jacobson [Jac88], cf. figure 3.8). Cet espace temporel Δ_b correspond au temps de traitement du second paquet par le routeur d'accès au goulet d'étranglement (cf. figure 3.9). En supposant que cet espace temporel reste inchangé jusqu'à l'arrivée des paquets-sondes au destinataire, la bande passante minimale est alors calculée en envoyant deux paquets consécutifs de taille

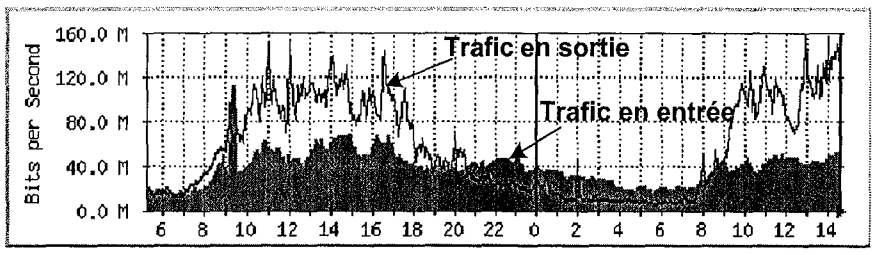


FIG. 3.7 – Résultats obtenus avec MRTG [OR]

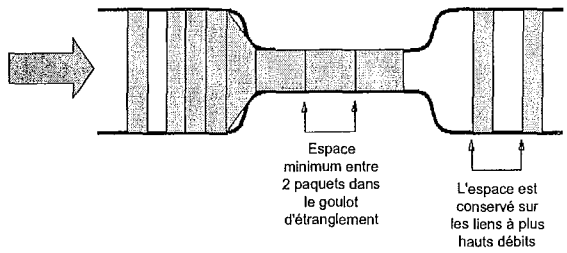


FIG. 3.8 – Phénomène de dispersion (d'après l'analogie au fluide introduite dans [Jac88])

connue et en mesurant le temps Δ_b qui sépare leurs arrivées au destinataire. La bande passante minimale correspond au quotient de la taille s du second paquet par le temps mesuré Δ_b[11] :

$$C = \frac{s}{\Delta_b}$$

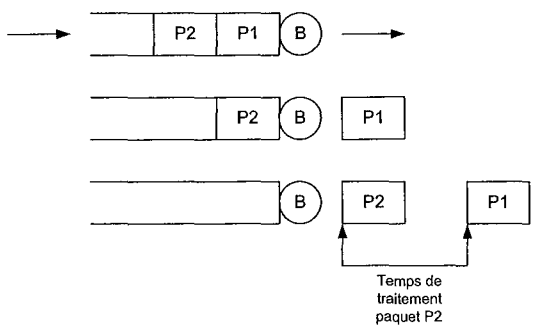


FIG. 3.9 – Arrivée des paquets-sonde au goulet d'étranglement

Les outils Bprobe [CC96], Pathrate [Dov] [DRM01], Sprobe [Sar01] et Nettimer [LB01] utilisent cette technique.

Dans le cas de Nettimer, la mesure est réalisée passivement en analysant des traces de trafic utile (analyse hors-ligne).

[11] Pour que cette technique fournisse des résultats fiables, la résolution de l'horloge du destinataire doit être suffisamment fine : dans [Pax97], il est montré que l'emploi de cette technique avec un système d'exploitation dont la résolution de l'horloge est égale à 10 ms, ne permet pas de faire la différence entre des liens de 200Ko/s et de 80 Mo/s.

Bprobe fait l'hypothèse que la dispersion reste la même si les paquets reviennent à la source. Dans ce cas, la mesure est possible en faisant faire un aller-retour aux paquets-sonde[12] (cf. figure 3.10). La bande passante minimale est alors mesurée par la source (cela évite de mettre en place une application côté source et une côté destination). Cette hypothèse est très critiquable car elle suppose que les paquets-sonde empruntent le même chemin à l'aller et au retour et que les caractéristiques du réseau soient identiques dans les deux sens (hypothèse inexacte si des liaisons sont asymétriques, DSL par exemple). De même, les hypothèses et les choix faits dans Sprobe sont très discutables et ne tiennent pas compte des observations formulées dans [DRM01] relatives à la taille des paquets-sonde par exemple.

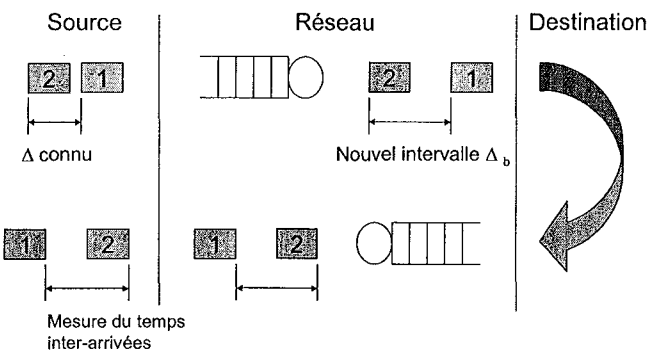


FIG. 3.10 – Mesure de la bande passante minimale par Bprobe

Seul Pathrate tient compte du trafic étranger susceptible de s'intercaler entre les paquets-sonde et qui bruitent les mesures.

De plus, la gamme de mesure est limitée par la dispersion minimale que le système terminal est capable de mesurer. Les expériences menées dans [DRM01] avec des stations SUN et des PC sous FreeBSD et Solaris montrent que la dispersion minimum est de l'ordre de 30 à 40 μs, soit une gamme de mesure limitée à 160 Mb/s[13].

[12] Les paquets-sonde sont alors des paquets ICMP *echo*.

[13] Valeur obtenue pour une taille de paquet-sonde de 800 octets, taille jugée empiriquement optimale dans [DRM01].

3.2.2 Délai unidirectionnel

Le délai unidirectionnel est le temps de transit d'un paquet de la source à la destination. Il inclut les délais de propagation, les délais de transmission ainsi que les délais induits par les mises en file d'attente dans les systèmes intermédiaires.

Soit deux systèmes terminaux, un système source SRC et un système destination DST. *Le délai de SRC à DST pour un paquet P est égale à D si SRC a envoyé le premier bit du paquet à t(heure réelle) et que DST a reçu le dernier bit du paquet à $t + D$ (heure réelle).*

Le délai unidirectionnel est souvent calculé en divisant le délai aller-retour (cf. 3.2.3) par deux. Or, les travaux de Paxson [Pax97] montrent que les routes sont de plus en plus asymétriques. Une tendance similaire concernant les liens (DSL, modem, satellite, etc.) est observée dans [Jia99]. Ce calcul est donc incorrect.

Pour mesurer le délai unidirectionnel, la source envoie à la destination un paquet estampillé de l'heure d'émission. A l'arrivée, la destination calcule le délai en soustrayant l'estampille du paquet à l'heure de réception lue sur sa propre horloge. Cette technique implique que les horloges des deux systèmes précédents soient synchronisées. En pratique, la synchronisation peut être réalisée par l'emploi de cartes GPS (PCI ou ISA), de modules de réception d'horloge radiofréquence ou encore en se synchronisant sur des serveurs NTP (Network Time Protocol). Les cartes GPS offrent la meilleure précision mais sont très coûteuses et posent des problèmes de réception en intérieur. Il est nécessaire d'installer des antennes extérieures. Les modules de réception radio sont beaucoup moins chers mais n'offrent pas une exactitude comparable[14]. Finalement, la synchronisation sur un réseau de serveurs NTP est la technique la plus largement utilisée. Elle est capable d'offrir une synchronisation de l'ordre de 10 ms sur l'Internet [JS99].

Dans tous les cas, le protocole NTP est utilisé : il régule la fréquence des horloges et maîtrise leur dérive [Mil91] en fonction des informations fournies par les différentes sources temporelles (serveurs NTP, GPS, etc.). Afin de ne pas briser la continuité temporelle de l'horloge, NTP ne la met pas directement à l'heure, mais la fait dériver volontairement. Cette dérive est réalisée en modifiant la valeur du pas d'incrémentation de l'horloge (*tick*). Des réserves sont à faire concernant l'utilisation de NTP sous Windows : à notre connaissance, les implémentations de NTP sous Windows sont en réalité des implémentations de SNTP (Simple NTP). Ce protocole se contente de remettre l'horloge à l'heure et risque

[14] L'exactitude de ces récepteurs dépend des conditions de propagations du signal radio fréquence.

ainsi de créer des discontinuités temporelles.

Les problèmes habituellement rencontrés lors de la mesure du délai sont détaillés dans [Jia99] et [Pax97]. La synchronisation des horloges est une problématique complexe et constitue à elle seule un vaste domaine d'étude.

On définit les paramètres suivants [AKZ99a] :

- *Erreur de synchronisation (ou offset) :* Décalage horaire entre deux horloges, noté T_{SYNCH}. Par exemple, si l'horloge A est en retard de $5,4ms$ sur l'horloge B, alors $T_{SYNCH} = 5,4ms$.
- *Exactitude (accuracy) :* Mesure le décalage d'une horloge par rapport à l'heure UTC absolue.
- *Résolution :* Mesure la précision d'une horloge. C'est le pas avec lequel l'horloge s'incrémente (tick). Par exemple, les horloges des vieux systèmes d'exploitation (Linux version inférieure à 2.2.0, etc.) ont une résolution de l'ordre de $10ms$. Les versions plus récentes de Linux ont une résolution de l'ordre de $20\mu s$ [LB01].
- *Dérive (skew) :* Variation d'exactitude ou de synchronisation. Par exemple, une horloge peut "gagner" $1.3ms$ par heure. Dans ce cas, si cette horloge est en retard de $27.1ms$ par rapport à l'heure UTC à l'instant t, son retard sera de $25.8ms$ une heure plus tard. On dit que l'horloge dérive de $1.3ms$par heure par rapport à l'heure UTC. On peut aussi définir la dérive d'une horloge par rapport à une autre.
- *Variation de la dérive (drift)* [Mil91] : Variation du paramètre précédent. Bien que l'on considère le plus souvent la dérive des horloges constante, celle-ci est en réalité variable. Elle change notamment en fonction de la température.

Attention, les termes *drift* et *skew* sont quelquefois utilisés pour faire référence à la dérive. Dans certains papiers, le terme *skew* fait référence à l'erreur de synchronisation. L'IETF définit les erreurs et incertitudes de mesures suivantes, et propose une méthode de calibration des mesures :

3.2.2.1 Erreurs et incertitudes relatives aux horloges

On définit T_{SRC} l'heure à laquelle le paquet part de la source. T_{SRC} est mesuré par rapport à l'horloge de la source. De même, T_{DST} est l'heure à laquelle le paquet arrive à la destination. T_{DST} est mesuré par rapport à l'horloge de la destination.

L'erreur de synchronisation T_{SYNCH} entre les horloges de la source et de la destination entache d'erreur la mesure du délai. Si on connaît T_{SYNCH}, il est possible de corriger le

délai mesuré en ajoutant T_{SYNCH} à T_{DST} - T_{SRC}.

L'exactitude est importante uniquement dans les cas où l'on souhaite dater la mesure (identifier l'heure à laquelle la mesure a été faite). L'exactitude n'a pas d'impact sur la mesure du délai.

La résolution des horloges ajoute de l'incertitude à la mesure. On note la résolution de l'horloge de la source R_{SRC} et celle de la destination R_{DST}.

De cette façon, on définit une incertitude relative aux horloges égale à $T_{SYNCH}(t) + R_{SRC} + R_{DST}$.

3.2.2.2 Erreurs et incertitudes relatives au datage des événements

Le délai est défini comme le temps qui s'écoule entre le départ de la source et l'arrivée à la destination d'un paquet. En pratique, le datage de ces événements est erroné car il n'est pas simultané : le paquet est estampillé juste avant son départ de la source. De même, la date d'arrivée est déterminée après l'arrivée du paquet. Ainsi, la mesure inclut des délais supplémentaires appelés respectivement H_{SRC} et H_{DST} pour la source et la destination (cf. figure 3.11) :

- L'instant de départ du paquet est mesuré à T_{SRC}. Le paquet quitte réellement la source à $T_{SRC} + H_{SRC}$.
- Le paquet arrive réellement à la destination à $T_{DST} - H_{DST}$. Son instant d'arrivée est mesuré à T_{DST}.

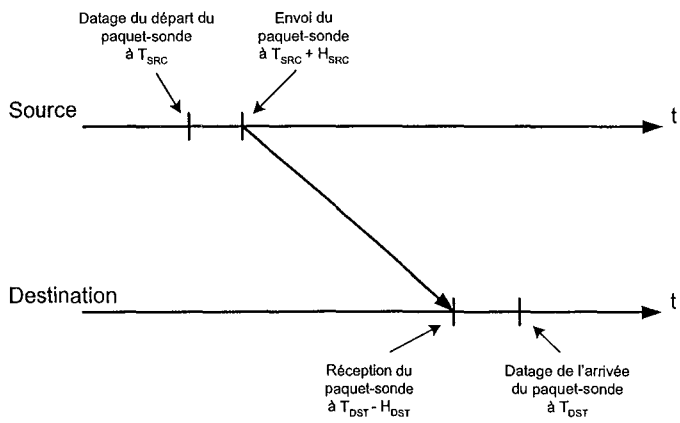


FIG. 3.11 – Erreurs relatives au datage des événements

```
$ ping 64.124.140.199

PING 64.124.140.199 (64.124.140.199) from 193.50.39.64 : 56(84) bytes of data.
64 bytes from 64.124.140.199: icmp_seq=0 ttl=241 time=163.773 msec
64 bytes from 64.124.140.199: icmp_seq=1 ttl=241 time=159.967 msec
64 bytes from 64.124.140.199: icmp_seq=2 ttl=241 time=159.973 msec
64 bytes from 64.124.140.199: icmp_seq=3 ttl=241 time=159.975 msec
64 bytes from 64.124.140.199: icmp_seq=4 ttl=241 time=159.975 msec
64 bytes from 64.124.140.199: icmp_seq=5 ttl=241 time=159.976 msec

--- 64.124.140.199 ping statistics ---
6 packets transmitted, 6 packets received, 0% packet loss

round-trip min/avg/max/mdev = 159.967/160.606/163.773/1.471 ms
```

FIG. 3.12 – Résultats obtenus avec Ping

3.2.2.3 Calibration

Le rôle de la calibration est de déterminer l'erreur de mesure.

D'après les paragraphes précédents, on a $T_{SYNCH}(t) + R_{SRC} + R_{DST} + H_{SRC} + H_{DST}$.

Dans certaines configurations et en utilisant un réseau isolé, il est possible d'évaluer l'erreur :

Les erreurs relatives aux horloges sont minimisées en utilisant des GPS. La somme $T_{SYNCH}(t) + R_{SRC} + R_{DST}$ est faible.

Les erreurs relatives au datage des événements $H_{SRC}+ H_{DST}$ peuvent être bornées en connectant les deux machines de mesure directement sur un réseau isolé.

Dans ces conditions, si les paquets-sonde sont petits, on peut faire l'hypothèse que le délai mesuré est minimal et peut être approximé par 0. Les valeurs mesurées correspondent à l'erreur de mesure. La valeur moyenne des mesures correspond à la part systématique de l'erreur, et les variations à la part aléatoire de l'erreur.

3.2.3 Délai aller-retour

Le délai aller-retour (round trip time ou RTT) inclut le délai de la source à la destination et le délai de la destination à la source. L'outil Ping mesure le délai aller-retour. La figure 3.12 montre le type de résultats obtenus avec ce dernier.

Soit deux systèmes terminaux, un système source SRC et un système destination DST. *Le délai aller-retour de SRC à DST pour un paquet P est égal à RTT si SRC a envoyé le premier bit du paquet à t (heure réelle), que DST a reçu le paquet et a aussitôt renvoyé un paquet semblable à SRC, et que SRC a reçu le dernier bit du paquet à* $t + RTT$ *(heure*

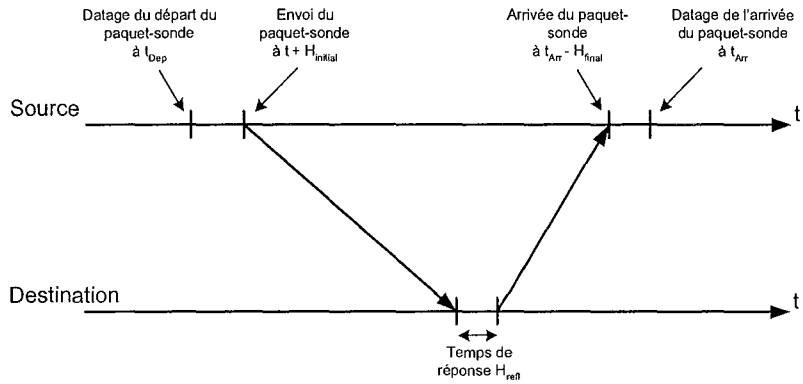


FIG. 3.13 – Erreurs relatives au datage des événements

réelle).

Pour effectuer la mesure, la source envoie un paquet à la destination. Dès la réception du paquet, la destination retourne le paquet à la source. Le délai aller-retour est le temps écoulé entre l'envoi du paquet et sa réception par la source. La mesure est plus simple que celle du délai unidirectionnel car elle ne nécessite pas de synchroniser les horloges des systèmes terminaux.

On considère négligeables les erreurs dues à la dérive de l'horloge, la durée de la mesure étant faible. Par contre, il faut s'assurer que rien ne puisse nuire à la continuité de l'horloge (protocole SNTP par exemple).

Les erreurs et incertitudes à prendre en compte sont :

- L'erreur relative à la résolution de l'horloge (cf. 3.2.2). Cette fois, les datages se font sur le même système, on a donc $2 * R_{SRC}$.
- Les erreurs relatives au datage des événements (non simultanéité du datage et des événements départ et arrivée d'un paquet, cf. 3.2.2), $H_{initial} + H_{final}$. Ces erreurs sont représentées sur la figure 3.13 : L'instant de départ du paquet est mesuré à $t = t_{Dep}$. Le paquet quitte réellement la source à $t_{Dep} + H_{initial}$. Le paquet revient à la source à $t_{Arr} - H_{final}$. Son instant d'arrivée est mesuré à $t_{Arr} = t_{Dep} + RTT$.
- L'erreur relative au temps de réponse de la destination : temps écoulé entre la réception du paquet par la destination et l'envoi de la réponse correspondante, H_{refl}. Cette erreur est illustrée sur la figure 3.13.

Il est possible d'évaluer l'erreur systématique et l'erreur aléatoire de mesure en utilisant

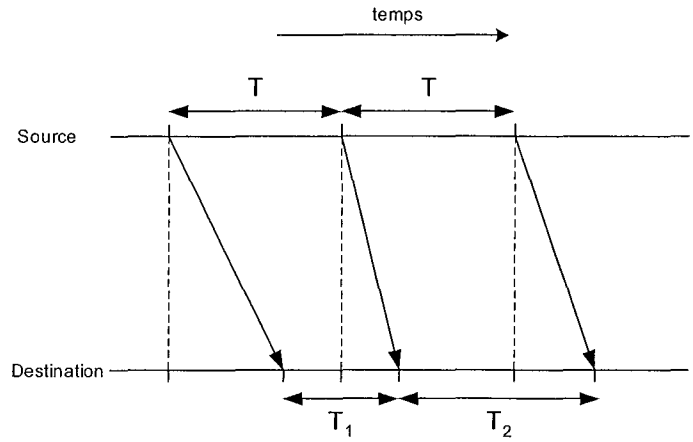


FIG. 3.14 – Variation du délai – Impact sur la périodicité des données

une démarche similaire à celle employée dans le cas du délai unidirectionnel (réseau isolé, paquets de petite taille, ici, on a $2 * R_{SRC} + H_{initial} + H_{final} + H_{refl}$).

3.2.4 Variation du délai (gigue)

Une variation importante du délai perturbe le transfert de médias continus comme la voix par exemple (cf. figure 3.14). En effet, un "rythme" constant de livraison des données est indispensable à la restitution correcte du signal audio.

Plusieurs métriques relatives à la variation du délai sont proposées par l'IETF [DC02] (cf. figure 3.15) :

- La métrique "variation de délai" est égale à la différence des délais de deux paquets identiques consécutifs. Soit d_i le délai du paquet i et d_{i+1} le délai du paquet $i+1$, la variation de délai est

$$dv_{i,i+1} = d_{i+1} - d_i$$

- Bien que l'IETF déconseille l'emploi du terme gigue, une métrique "gigue" (*jitter*) est définie : elle est déterminée en calculant la valeur absolue de la variation du délai et en lui appliquant (ou non) le filtre exponentiel proposé dans [SCFJ96].

$$g_{i,i+1} = |dv_{i,i+1}| \text{ ou } g_{i,i+1} = \frac{15}{16} g_{i-1,i} + \frac{1}{16} |dv_{i,i+1}|$$

- On peut aussi calculer la "variation de délai crête à crête" qui correspond à la

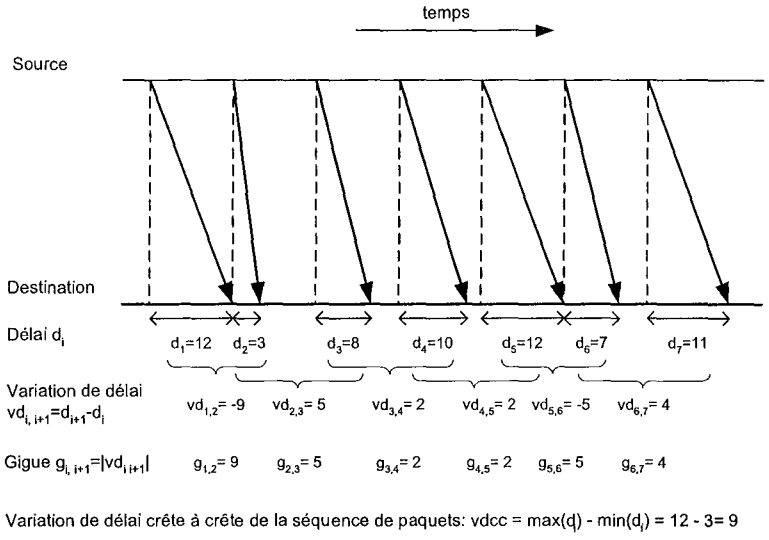


FIG. 3.15 – Calcul des métriques relatives à la variation du délai

différence des délais maximum et minimum mesurés pour une séquence de paquets.

$$vdcc = \max_{i=1...n}(d_i) - \min_{i=1...n}(d_i) \text{ avec } n \text{ le nombre de paquets de la séquence}$$

Certains auteurs définissent la gigue comme l'écart type des valeurs de délai[15].

La mesure de la gigue ne nécessite pas la synchronisation des horloges des deux systèmes terminaux puisque les erreurs de synchronisation sont annulées lorsqu'on calcule les différences de délai. On considère négligeables les erreurs dues à la dérive des horloges, la durée de la mesure étant faible.

Il faut prendre en compte :

- L'erreur relative à la résolution de l'horloge (cf. 3.2.2). Cette erreur deux fois supérieure à celle qui intervient dans le cas du délai. En effet, une valeur de variation de délai est calculée à partir de deux mesures de délai.
- Les erreurs relatives au datage des événements (non simultanéité du datage et des événements départ et arrivée d'un paquet, cf. 3.2.2).

[15]Dans ce cas, les valeurs de délai sont supposées être distribuées selon une loi normale [HvB98].

3.2.5 Pertes de paquets

La fiabilité d'une liaison est exprimée par le taux de pertes de paquets. Cette métrique est le quotient du nombre de paquets non reçus par le nombre total de paquets envoyés. La détection à l'émetteur de paquets non reçus par le destinataire implique l'utilisation de temps d'expiration (*time-out*) au delà duquel un paquet est considéré comme perdu s'il n'a pas été acquitté. La détection au récepteur de paquets non reçus implique l'utilisation de temps d'expiration au delà duquel un paquet est considéré comme perdu s'il n'a pas été reçu. Lorsque la détection des pertes est réalisée par le destinataire, il est difficile d'utiliser un temps d'expiration si les horloges de la source et de la destination ne sont pas synchronisées. En pratique, on pourra détecter les pertes en se basant uniquement sur les numéros de séquence des paquets : si plusieurs paquets avec un numéro de séquence supérieur sont arrivés mais pas le paquet intéressé.

La valeur du temps d'expiration est libre et dépend des besoins de l'utilisateur et/ou de l'application [AKZ99b]. La détermination de cette valeur n'est pas simple et implique de connaître le "degré de fraîcheur" de l'information requis par l'utilisateur et/ou l'application.

L'utilitaire le plus connu pour mesurer le taux de pertes de paquets est Ping. Celui-ci détermine les pertes qui se produisent lors de l'aller-retour de paquets-sonde ICMP entre une source et une destination. Il s'agit d'un taux global de pertes car il n'est pas possible de différencier les pertes qui se produisent à l'aller de celles qui se produisent au retour.

3.2.5.1 Mesure des pertes en utilisant le mécanisme d'acquittement de TCP

L'outil Sting [Sav99] permet de mesurer les pertes de paquets selon les deux directions d'une connexion. La technique utilisée se base sur l'analyse des acquittements du protocole TCP. Dans un premier temps, la source envoie plusieurs paquets à la destination. La source passe ensuite dans une phase d'analyse des acquittements. Si le dernier paquet envoyé est acquitté, tous les paquets ont été transmis car TCP utilise une politique d'acquittement "collectif". Le taux de perte est alors nul. Dans le cas contraire, l'acquittement reçu permet de déterminer le premier paquet perdu (cf. figure 3.16)[16]. La source retransmet alors ce paquet et recommence l'analyse des acquittements jusqu'à ce que tous les paquets soient acquittés. De cette façon, la source détermine le nombre de paquets perdus lors de la phase

[16] Si aucun paquet n'est reçu par le destinataire, l'émetteur le détecte par l'absence de retour d'acquittement (utilisation d'un *time-out*)

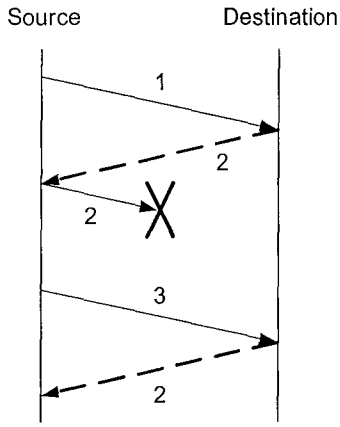


FIG. 3.16 – Sting, détermination des pertes aller

d'envoi.

Pour déterminer les pertes dans le sens retour, il faut détecter les acquittements perdus. Cela implique de connaître le nombre d'acquittements envoyés. Or celui-ci n'est pas nécessairement égal au nombre de paquets envoyés. En effet, la plupart des implémentations de TCP n'acquittent pas chaque paquet reçu : la destination ne répond pas instantanément lors de la réception d'un paquet, mais attend un certain temps (de 100 à $500ms$ selon les implémentations) dans l'espoir que le paquet suivant arrive. Cela permet d'acquitter plusieurs paquets à la fois et ainsi de réduire le nombre d'acquittements.

Pour forcer la destination à envoyer un acquittement pour chaque paquet qu'il reçoit, l'outil Sting met à profit l'algorithme *fast-retransmit* de TCP. Cet algorithme a été initialement conçu pour permettre à la destination de demander explicitement à la source de retransmettre un paquet perdu : en oubliant volontairement d'envoyer le premier paquet, on force la destination à demander sa retransmission pour chaque paquet reçu par la suite (cf. figure 3.17). Connaissant le nombre de paquets reçus par la destination, on connaît le nombre d'acquittements retournés. Sachant le nombre d'acquittements reçus par la source, on connaît le nombre d'acquittements perdus sur le chemin du retour.

3.2.5.2 Modèles de pertes

Il est couramment admis qu'une perte de paquets est une indication de congestion. Donc le paquet qui suit un paquet perdu a une probabilité à priori plus importante d'être

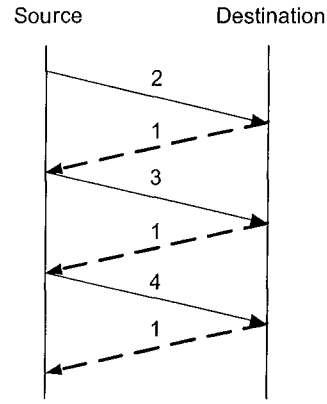


FIG. 3.17 – Mise à profit de l'algorithme *fast-retransmit* de TCP

perdu. Cette remarque conduit à prendre en compte la dépendance temporelle entre les pertes. Dans [BS98], il a été montré que cette dépendance est non nulle. La façon dont les pertes de paquets se produisent (en rafales, etc.) est un paramètre important pour certaines applications (telles que les applications audio et vidéo). En effet, à taux de perte équivalents, deux distributions des pertes différentes peuvent impliquer des dégradations différentes des performances de l'application.

La dépendance temporelle entre les pertes est habituellement modélisée par un modèle de Markov à 2 états [JS99], appelé aussi modèle de Gilbert (cf. figure 3.18). p est la probabilité que le paquet suivant soit perdu sachant que le précédent est arrivé. q est la probabilité que le paquet suivant arrive sachant que le précédent a été perdu (normallement $p + q < 1$).

Ce modèle peut être étendu (cf. figure 3.19) à n états. Cette modélisation prend en considération que les n événements consécutifs de perte ont des conséquences sur les événe-

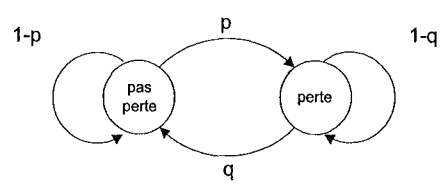


FIG. 3.18 – Modèle de Markov à 2 états (ou modèle de Gilbert)

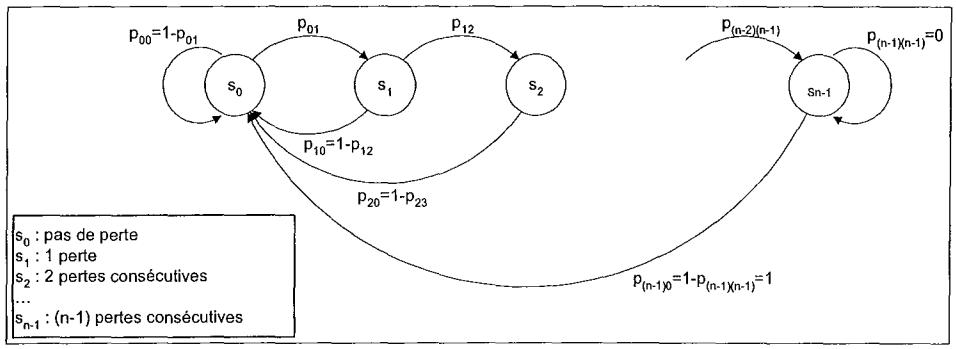


FIG. 3.19 – Modélisation des pertes de paquets par un modèle de Gilbert étendu [JS99]

ments à venir. Elle exprime les probabilités p_{ij} qu'un paquet soit reçu ou perdu sachant que les i paquets précédents ont été perdus. De plus, certains auteurs ont proposé des critères pour caractériser la dépendance des pertes de paquets [BS98]. Ces derniers travaux proposent aussi un critère pour caractériser l'asymétrie des pertes.

3.2.5.3 Métriques relatives à la distribution des pertes proposées par l'IPPM

Dans ce contexte, l'IPPM propose un certain nombre de métriques et de statistiques qui tiennent compte de la distribution des pertes [KR02] en introduisant les notions de période de perte et de distance entre pertes : une période de pertes est une séquence de paquets perdus successivement. La distance entre pertes indique le nombre de paquets entre un paquet perdu et le paquet perdu précédent (séparés ou non par des paquets reçus). A partir de ces deux notions, il est simple de calculer des paramètres tels que la longueur des périodes de pertes et la longueur des séquences séparant des périodes de pertes successives.

L'IPPM propose aussi le calcul d'un taux de pertes significatives : une perte est jugée significative si elle se produit à une distance inférieure à Δ de la dernière perte, Δ étant la contrainte sur les pertes. Ce critère peut être particulièrement intéressant dans le cas par exemple des codecs multimédias qui sont capables de s'affranchir des pertes en se basant sur l'historique des dernières informations reçues. Dans ce cas, Δ représente la taille de l'historique minimum. La figure 3.21 illustre le calcul du taux de pertes significatives : dans les deux cas, le taux de pertes est égal à 1% (5 paquets perdus sur 500). Dans le premier

cas, aucune perte n'est significative car l'écart entre les pertes successives est supérieur à $\Delta = 99$. Le taux de pertes significatives est nul. Par contre, dans le deuxième cas, les pertes des paquets 175 et 290 sont significatives. Le taux de pertes significatives est égal à 40% (2 pertes significatives sur 5).

3.2.6 Route

La route suivie par les paquets entre une source et une destination peut être déterminée par l'utilitaire Traceroute développé par Van Jacobson [Jac]. Celui-ci utilise le champ TTL des paquets IP (cf. 3.2.1.2).

Pour déterminer la route d'une source à une destination, Traceroute envoie à la destination des paquets-sonde UDP avec un numéro de port quelconque. Il commence par envoyer un paquet dont le TTL est fixé à 1. Lorsque le premier routeur reçoit le paquet, il décrémente le TTL. Le TTL est alors égal à 0. Le routeur supprime le paquet et envoie un message ICMP à la source pour l'informer de la suppression. Le paquet ICMP contient l'adresse du routeur dans son en-tête. Cette information permet à traceroute de connaître l'adresse du premier routeur. Traceroute envoie alors un second paquet avec un TTL égal à 2. Ce paquet est supprimé par le deuxième routeur. Le deuxième routeur envoie un message ICMP indiquant la perte à la source. La source en déduit l'adresse du deuxième routeur. Cette technique est répétée jusqu'à ce que Traceroute reçoive un message ICMP "port unreachable" de la destination (cf. figure 3.22).

Bien qu'étant un outil de référence, Traceroute rencontre un certain nombre de limites [Pax97] :

- La route peut changer entre plusieurs sondes. Il n'est pas garanti que les sondes qui

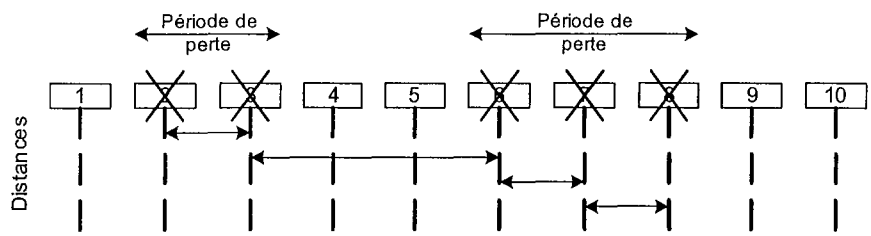


FIG. 3.20 – Périodes et distances de perte

Séquence	1	2	...	100	...	200	...	300	...	400	...	500			
Distance de perte				100		100		100		100		100			
Séquence	1	2	...	100	...	175	...	275	...	290	...	400			
Distance de perte				100		75		100		15		110			

FIG. 3.21 – Pertes significatives avec $\Delta = 99$

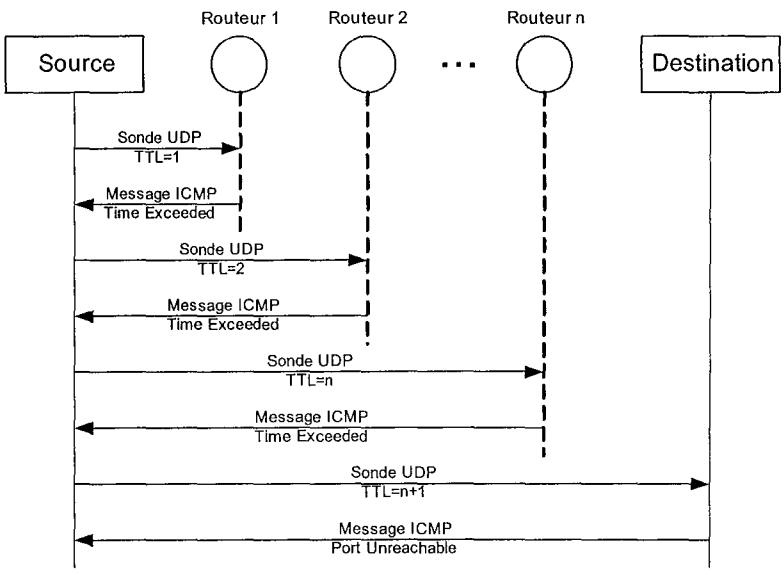


FIG. 3.22 – Détermination de la route avec Traceroute

déterminent le routeur n utilisent la même route que les sondes qui ont déterminé le routeur $n-1$.

- Traceroute détermine les nœuds de la couche IP, mais ne peut pas détecter les équipements d'interconnexion de couches inférieures (commutateurs ATM par exemple). Entre deux noeuds indiqués dans la route obtenue, il peut y avoir plusieurs liens physiques différents.
- Les paquets envoyés par traceroute sont adressés à des numéros de ports quelconques. Or, certains coupe-feu (firewalls) sont configurés pour supprimer les paquets UDP dont les numéros de port destinataires ne correspondent pas à un service prévu. Il peut ainsi arriver que Traceroute ne reçoive pas les messages ICMP (time exceeded et port unreachable) nécessaires à la détermination de la route.

3.3 Conclusion

Nous avons présenté dans ce chapitre, les paramètres essentiels de "QdS réseau". Pour chaque paramètre, nous avons dressé l'état de l'art des techniques et des outils de mesure qui lui sont associés. Le tableau 3.1 de la page 63 résume les outils de mesure et les métriques de l'IETF pour chaque paramètre de QdS. De plus, il indique des références bibliographiques dans lesquels le lecteur pourra trouver de précieux compléments d'information. Le tableau 3.2 de la page 64 présente les caractéristiques de chaque outil de mesure : il indique les paramètres mesurés, la classe de mesure, et précise si l'outil est coopératif ou non (c'est-à-dire s'il est nécessaire de le déployer sur les deux systèmes concernés par la mesure), le protocole sur lequel les mesures sont faites et le système d'exploitation supporté. Nous constatons que dans l'ensemble, les outils de mesure sont dédiés au calcul d'un seul paramètre, pour un protocole de transport. Il est en conséquence techniquement difficile de réaliser simultanément la mesure de plusieurs paramètres entre deux systèmes terminaux. Par ailleurs, les mesures réalisées par ces outils sont le plus souvent unidirectionnelles, ce qui signifie qu'il n'est pas possible de les réaliser dans les deux sens de la communication.

Dans le chapitre suivant, nous proposons une nouvelle architecture à QdS intégrée. Celle-ci a pour but l'adaptation des applications à la QdS, elle doit donc être capable de mesurer simultanément tous les paramètres de QdS pertinents pendant l'exécution de l'application. C'est pourquoi l'architecture inclut un service de métrologie de la QdS.

La suite de nos travaux sur la métrologie se situe dans les deux chapitres suivants :

	Outils de mesure	Métriques de l'IETF	Compléments d'information
Bande passante totale	Pathchar [Jac97], Bing [Bey95], Clink [Dow99], Pchar [Mah99], Nettimer [LB00]		[Dow99] [Jia99]
Bande passante disponible	Cprobe [CC96], Iperf [Ipe], Pathload [JD02a] [JD02b], Pipechar [JYCA01]	draft [ASU+]	
Bande passante minimale	Bprobe [CC96], Pathrate [Dov] [DRM01], Nettimer [LB01], Sprobe [Sar01]		[Pax97]
Délai unidirectionnel		RFC 2679 [AKZ99a]	[Jia99] [Pax97] [Pax98] [JS99]
Délai aller-retour	ping	RFC 2681 [AKZ99c]	[JS99]
Variation de délai	Iperf [Ipe]	RFC 3393 [DC02]	[HvB98]
Pertes de paquets	Ping, Sting [Sav99] Iperf [Ipe]	RFC 2680 [AKZ99b], RFC 3357 [KR02]	[JS99]
Route	Traceroute [Jac]		

TAB. 3.1 – Récapitulatif des paramètres de QoS, des techniques/outils de mesure, et des métriques définies par l'IETF

dans le chapitre 4 sera exposé le service de métrologie que nous avons conçu, et dans le chapitre 5 sera présentée son implémentation. La conception modulaire de ce service permet d'intégrer n'importe quelle technique de mesure exposée précédemment. Utilisé par deux systèmes terminaux, il permet de mesurer simultanément et dans les deux sens de la communication, plusieurs paramètres différents. Une session de mesure bidirectionnelle est totalement contrôlée par un des deux systèmes terminaux qui y participent (paramétrage, mise en place des procédures de mesure, transfert et sauvegarde des résultats de mesure, etc.).

Outil	Délai aller-retour	Délai unidirectionnel	Variation de délai	Pertes de paquets	Bande passante disponible	Bande passante totale	Bande passante minimale	Classe de mesure	Type d'environnement	Protocole	Système d'exploitation
Ping	u			ar				a	nc	ICMP	*
Sting				b				a	nc	TCP	FreeBSD Linux
Bprobe							u	a	c	ICMP	SGI Irix
Cprobe					u			a	c	ICMP	SGI Irix
Sprobe							b	a	nc	TCP	FreeBSD Linux
Pathchar						u		a	nc	UDP	U
Bing						u		a	nc	ICMP	U
Clink						u		a	nc	UDP/ICMP	Linux
Pchar						u		a	nc	UDP/ICMP	U
Pathload					u			a	c	UDP	U
Pathrate							u	a	c	UDP	U
Pipechar NCS					u	u		a	c/nc	UDP/ICMP	U
Nettimer						u	u/b	a/p	c/nc	TCP	Linux
Iperf			b (UDP)	b (UDP)	b (TCP)			a	c	TCP/UDP	*

Colonnes des paramètres mesurés : "u" : unidirectionnel, "b" : bidirectionnel, "ar" : aller-retour.

Colonne classe de mesure : "a" : active, "p" : passive.

Colonne type d'environnement : "c" : coopératif, "nc" : non-coopératif.

Colonne système d'exploitation : "*" : la plupart, "U" : la plupart des Unix.

TAB. 3.2 – Caractéristiques des outils de mesures

Chapitre 4

"QdS-Adapt" : Une Architecture de QdS pour l'adaptation des applications

Dans ce chapitre, nous proposons une nouvelle architecture à QdS intégrée. Celle-ci est conçue pour permettre aux applications de s'adapter aux performances du réseau. Elle s'inspire de l'architecture Prayer [BG97] : elle en reprend la structure et la technique d'adaptation, et la complète en lui ajoutant les mécanismes de QdS présentés dans le chapitre 2, et notamment un service de métrologie de la QdS.

Dans un premier temps, nous présentons l'architecture Prayer. Nous discutons alors de ses avantages et inconvénients en nous appuyant sur l'étude menée au chapitre 2. La partie 4.2 est consacrée à la nouvelle architecture proposée. Nous commençons par décrire sa structure statique, puis nous expliquons son fonctionnement dynamique et le processus d'adaptation en ligne de l'application. Enfin, le service de métrologie des ressources est détaillé dans la partie 4.3.

4.1 Architecture Prayer

4.1.1 Présentation

L'architecture Prayer a été proposée par Bhargavan et Gupta dans [BG97]. Elle a été conçue pour des environnements dans lesquels des clients mobiles communiquent avec des serveurs fixes. Les serveurs sont connectés à un réseau filaire. Les clients mobiles

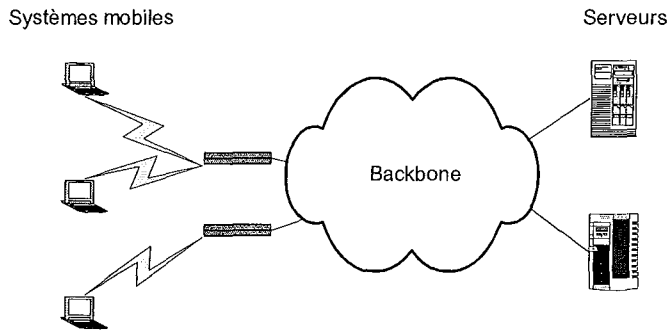


FIG. 4.1 – Environnement mobile et sans fil

utilisent des réseaux sans fil pour se connecter au réseau filaire (cf. figure 4.1). Dans ce type d'environnement, la QdS est très variable. Ceci est dû à la nature non fiable des liaisons sans fil, à la mobilité de l'utilisateur entre différentes cellules, et à sa migration entre différents réseaux sans fil. L'objectif de Prayer est de fournir à l'utilisateur la sensation de travailler dans un environnement stable malgré la nature variable de la QdS offerte par le réseau.

L'architecture Prayer est structurée en trois couches (cf. figure 4.2) : la couche "ressources", la couche "gestion des ressources", et la couche "application".

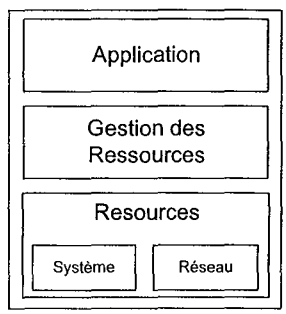


FIG. 4.2 – Représentation informelle de l'architecture Prayer

La première couche citée englobe les différentes ressources, c'est à dire les ressources réseau et les ressources propres au système terminal (processeur, mémoire, etc.).

La couche "gestion des ressources" inclut tous les mécanismes pour gérer et surveiller les ressources, et pour réagir aux changements de ressources disponibles.

La couche "application" fournit les règles qui définissent la politique d'adaptation (comment faire face aux changements de ressources disponibles).

4.1.2 Mécanisme d'adaptation

Dans cette partie, nous décrivons le mécanisme d'adaptation proposé dans l'architecture Prayer. Pour cela, nous avons choisi d'utiliser les réseaux de Petri colorés dont nous rappelons les concepts de base et la terminologie dans les trois paragraphes suivants.

4.1.2.1 Réseaux de Petri (d'après [Abd02])

Un réseau de Petri est défini par un ensemble de places, un ensemble de transitions, et un ensemble d'arcs orientés qui connectent les transitions aux places ou vice-versa. Des poids (des entiers naturels) sont associés aux arcs. Les places contiennent des jetons. Une place est qualifiée d'entrée à une transition, s'il existe un arc qui la relie à une transition (cet arc est appelé arc d'entrée). En revanche, s'il existe un arc qui relie une transition à une place, cette dernière est une place de sortie de la transition et l'arc correspondant est appelé arc de sortie. Une transition est dite sensibilisée lorsque chaque place d'entrée possède un nombre de jetons supérieur ou égal au poids de l'arc d'entrée. Les transitions sensibilisées peuvent être tirées. Le tir d'une transition enlève de chaque place d'entrée un nombre de jetons égal au poids de l'arc qui la relie à la transition.

Remarque : A la différence des définitions données dans [Abd02], nous ne tenons pas compte ici des arcs inhibiteurs.

4.1.2.2 Réseaux de Petri colorés

Les réseaux de Petri colorés sont des réseaux de Petri dans lesquels les jetons portent des couleurs. Une couleur est une information attachée à un jeton et permet de distinguer les jetons entre eux. Ainsi, chaque arc n'est pas seulement étiqueté par le nombre de jetons mais aussi par leur couleur. Le franchissement d'une transition est alors conditionné par la présence dans les places en entrée du nombre de jetons nécessaires, qui en plus satisfont les couleurs qui étiquettent les arcs. Après le franchissement d'une transition, les jetons qui étiquettent les arcs d'entrée sont retirés des places d'entrée tandis que ceux qui étiquettent les arcs de sortie sont ajoutés aux places de sortie de cette transition.

4.1.2.3 Représentation graphique

Les réseaux de Petri présentés dans la suite ont été conçus et simulés à l'aide du logiciel *DesignCPN* [Des]. Les symboles utilisés pour représenter les places, les transitions et les arcs sont illustrés sur la figure 4.3 : Une place est symbolisée par une ellipse dans laquelle est inscrit le nom de la place. Une transition est représentée par un rectangle. Le nom de la transition est écrit à côté du rectangle ainsi que la condition éventuelle de franchissement (ou garde). Un arc est désigné par une flèche sur laquelle est indiqué le nom d'une variable (de type couleur de jeton). Lorsqu'un arc pointe sur une transition, la variable de l'arc sert à évaluer la condition de franchissement de la transition, c'est-à-dire à déterminer si la transition doit être franchie ou non selon la (les) couleur(s) du (des) jeton(s) situé(s) dans la place d'entrée. Lorsqu'un arc pointe sur une place de sortie, la variable indique la couleur du jeton à mettre dans cette place suite au franchissement de la transition.

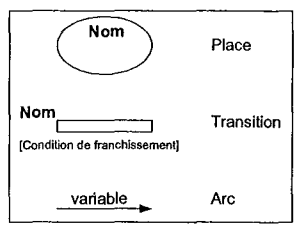


FIG. 4.3 – Symboles utilisés par le logiciel DesignCPN

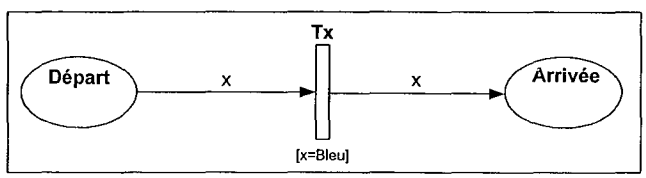


FIG. 4.4 – Exemple de modélisation avec *DesignCPN*

Dans l'exemple de la figure 4.4, deux places, une transition et deux arcs sont représentés. La place "Départ" est la place d'entrée, la place "Arrivée" est la place de sortie. Si la place d'entrée contient un jeton bleu (x=Bleu), alors la transition est franchie. Le jeton bleu est enlevé de la place de départ, un jeton bleu est ajouté dans la place de sortie.

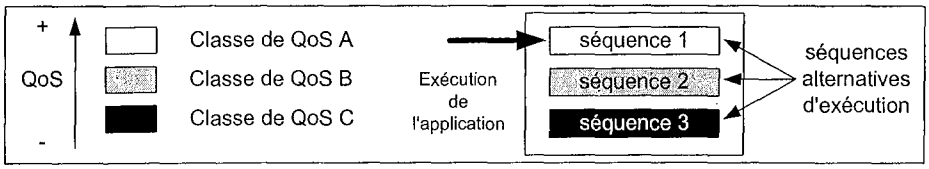


FIG. 4.5 – Classes de QoS et séquences d'exécution associées

4.1.2.4 Mécanisme d'adaptation

L'interaction entre l'application et la couche gestion des ressources est fondée sur les concepts de classe de QdS et de séquences d'exécution :

D'une part, l'application définit des classes de QdS qui sont des niveaux de disponibilité des ressources (ou niveaux de QdS disponible). Les classes sont spécifiées en fonction des quatre paramètres suivants : débit, délai, pertes et coût de transmission. Elles correspondent à des plages de variations de ces paramètres, c'est-à-dire à un ensemble de couples (valeur minimum, valeur maximum) pour chacun d'eux.

D'autre part, l'application renferme plusieurs séquences alternatives d'exécution (cf. figure 4.5). Une séquence est une portion de code exécutable. Chaque séquence est associée à une classe de QdS. Au démarrage de l'application, cette dernière négocie avec la couche de gestion des ressources une classe de QdS. L'application exécute alors la séquence d'exécution correspondante.

Les figures 4.5 et 4.6 illustre l'exemple d'une application pour laquelle trois séquences d'adaptation "1", "2" et "3" sont respectivement associées à trois classes de QdS "A", "B" et "C".

Les classes de QdS sont modélisées par trois couleurs de jeton : les couleurs A, B et C. A l'exécution de chaque séquence correspond une place (respectivement les places "Seq1", "Seq2" et "Seq3").

Au démarrage de l'application, un jeton est présent dans la place "Etat initial" et la classe de QdS négociée est indiquée par la présence d'un jeton de couleur correspondante (A, B ou C) dans la place "Classe". Selon la couleur de ce dernier, une seule des transitions "Exécuter seq1", "Exécuter seq2" et "Exécuter seq3" est franchissable. Suite au tir de celle-ci, un jeton est mis dans la place correspondante ("Seq1", "Seq2" ou "Seq3"). Lorsque la séquence d'exécution se termine, un jeton est introduit dans la place "Fin seq$N$" (avec N le numéro de la séquence associée), et la transition "Terminer seq$N$" est tirée. Le système

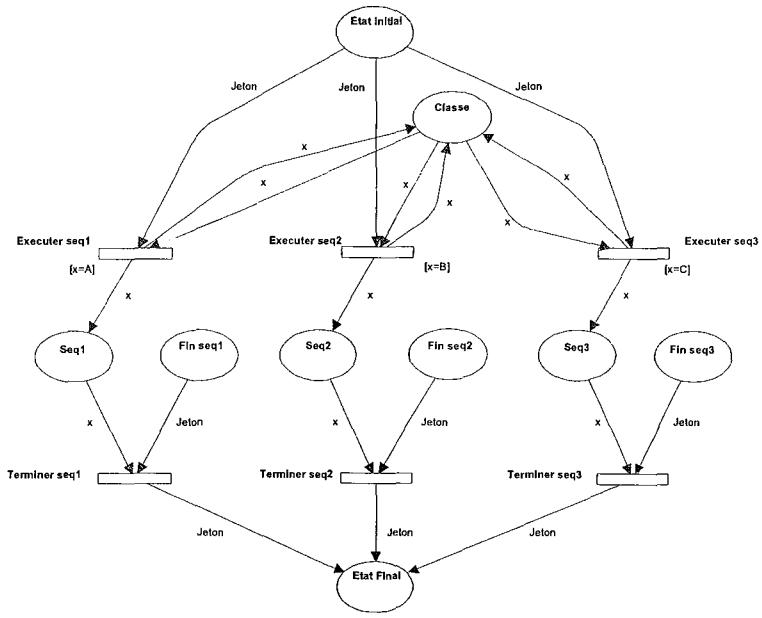


FIG. 4.6 – Choix et fin des séquences d'exécution

se trouve alors dans l'état final, l'application a terminé son exécution.

Adaptation en cours d'exécution

Durant l'exécution de la séquence, la couche de gestion des ressources se charge de garantir la classe de QdS négociée en utilisant des mécanismes de réservation de ressources. Si la classe n'est pas respectée ou si la couche de gestion des ressources peut garantir une meilleure classe, la couche de gestion en notifie l'application laquelle réagit alors au changement en fonction d'actions d'adaptation (*"fallback actions"*) prédéfinies et associées à la séquence d'exécution. Ces actions sont spécifiées par l'application avant le démarrage du bloc d'adaptation dans une séquence CAP (Classe, Action, Procédure) qui associe une classe de QdS, un couple d'actions d'adaptation et une séquence d'exécution (désignée par le champ procédure).

Le couple d'actions d'adaptation comporte une action à entreprendre lorsque la classe de QdS n'est pas respectée, appelée *"low_action"* pour laquelle il existe plusieurs primitives :

- BLOCK : l'application suspend son exécution jusqu'à ce que la classe de QdS soit à nouveau disponible,

- BEST_EFFORT : l'application ignore les changements de QdS et poursuit l'exécution de la séquence en cours,
- ROLLBACK : l'application recommence l'exécution du bloc d'adaptation courant avec une nouvelle classe de QdS,
- ABORT : l'application arrête son exécution et quitte le bloc d'adaptation en cours.

La deuxième action, appelée *"high_action"*, définit l'action à mener lorsqu'une classe de QdS supérieure peut être satisfaite. Il existe plusieurs primitives pour ce type d'actions (les définitions de ces actions sont identiques aux précédentes) :

- BEST_EFFORT,
- ROLLBACK,
- ABORT.

Le fonctionnement des actions d'adaptation est illustré dans la suite en reprenant l'exemple des figures 4.5 et 4.6 et en faisant l'hypothèse que la classe de QdS A est un niveau de QdS supérieur à la classe B, lui-même supérieur à la classe C.

La figure 4.7 représente l'action d'adaptation *"down_action"* associée à la séquence d'exécution 1. La primitive choisie est "BLOCK". Aucune action d'adaptation *"high_action"* n'est définie car la séquence 1 est associée à la classe de QdS A et il n'existe pas de classe de QdS supérieure à cette dernière.

Suite à la phase d'initialisation, nous supposons que l'exécution de la séquence 1 a été lancée. Dans ce cas, un jeton A est présent dans la place "Classe" ainsi que dans la place "Seq1".

La place "Nouvelle Classe" symbolise la notification d'un changement de classe de QdS par la couche de gestion des ressources : dans ce cas, un jeton de couleur correspondante à la nouvelle classe est ajouté dans cette place.

Remarque : Seul un changement de classe est notifié. Par définition, lorsqu'un jeton de couleur x est présent dans la place "Classe", aucun jeton de couleur x ne peut être introduit dans la place "Nouvelle classe".

Si la nouvelle classe est B ou C, l'application enclenche la primitive BLOCK correspondant à l'action d'adaptation *"down_action"*. Dans ce cas, la transition "Se bloquer" est franchie, un jeton est ajouté dans la place "Attente", le jeton présent dans la place "Seq1" est supprimé, et le jeton de la place "Classe" est remplacé par un jeton dont la couleur correspond à la nouvelle classe (mise à jour de la classe). Dans cette situation, seule l'arrivée d'un jeton A dans la place "Nouvelle Classe" permet de quitter l'attente et remettre un jeton dans la place "Seq1" par franchissement de la transition "Se débloquer".

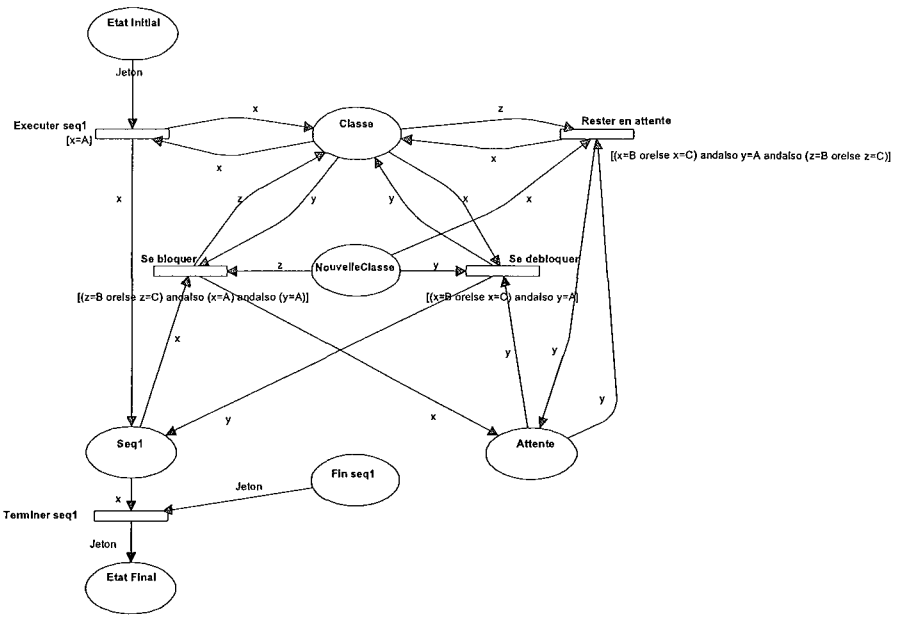


FIG. 4.7 – Séquence d'exécution 1 et action d'adaptation BLOCK

Si la classe courante est B et que la nouvelle classe est C (ou inversement), l'application reste en attente, et la classe est mise à jour.

Remarque : les arcs qui relient la place "Attente" à la transition "Rester en attente" peuvent sembler superflus à première vue. Ceux-ci sont indispensables lorsqu'on construit le réseau de Petri complet du mécanisme d'adaptation (cf. figure 4.10 page 75).

La figure 4.8 illustre les actions d'adaptation associées à la séquence d'exécution 2 : il s'agit de la primitive BEST_EFFORT pour l'action *"high_action"* et de la primitive ABORT pour l'action *"down_action"*.

Suite à la phase d'initialisation, nous supposons que l'exécution de la séquence 2 a été lancée. Dans ce cas, un jeton B est présent dans la place "Classe" ainsi que dans la place "Seq2".

Si la nouvelle classe de QdS est A (jeton A dans la place "Nouvelle classe"), la classe est mise à jour (couleur du jeton dans la place "Classe") et l'application poursuit l'exécution de la séquence 2. Si la nouvelle classe est C, l'application est en échec (place "Echec").

Remarque : les arcs qui relient la place "Seq2" à la transition "Best-effort" peuvent

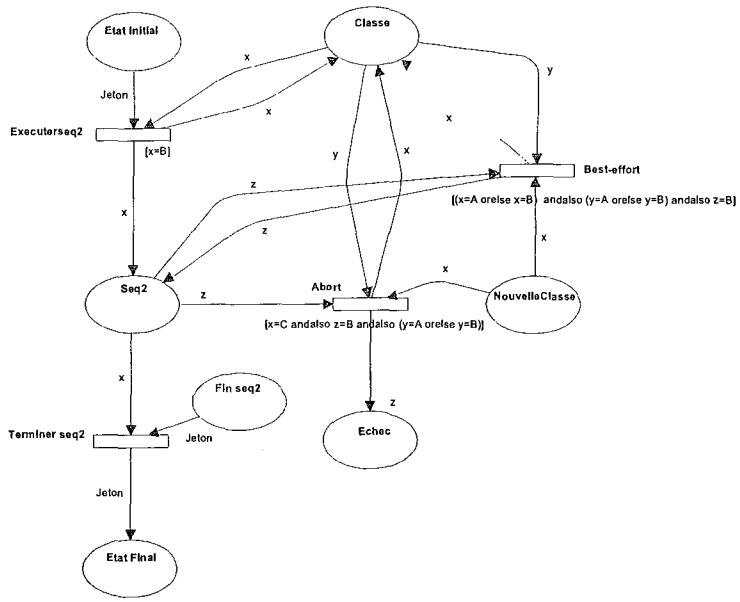


FIG. 4.8 – Séquence d'exécution 2 et actions d'adaptation BEST_EFFORT et ABORT

sembler superflus à première vue. Ceux-ci sont indispensables lorsqu'on construit le réseau de Petri complet du mécanisme d'adaptation (cf. figure 4.10 page 75).

La figure 4.9 présente l'action d'adaptation *"high_action"* associée à la séquence d'exécution 3. La primitive choisie est ROLLBACK. Aucune action d'adaptation *"down_action"* n'est définie car la séquence 3 est associée à la classe de QdS C et il n'existe pas de classe de QdS inférieure à cette dernière.

Suite à la phase d'initialisation, nous supposons que l'exécution de la séquence 3 a été lancée. Dans ce cas, un jeton C est présent dans la place "Classe" ainsi que dans la place "Seq3".

Si la nouvelle classe est A ou B, l'application est renvoyée à l'état initial par franchissement de la transition "Rollback" et la classe est mise à jour (couleur du jeton dans la place "Classe").

Le fonctionnement global du mécanisme d'adaptation est modélisé par superposition des réseaux de Petri précédents (cf. figure 4.10 page 75). Pour des raisons de visibilité, les conditions de franchissement ne sont pas indiquées sur cette figure (elles sont identiques à

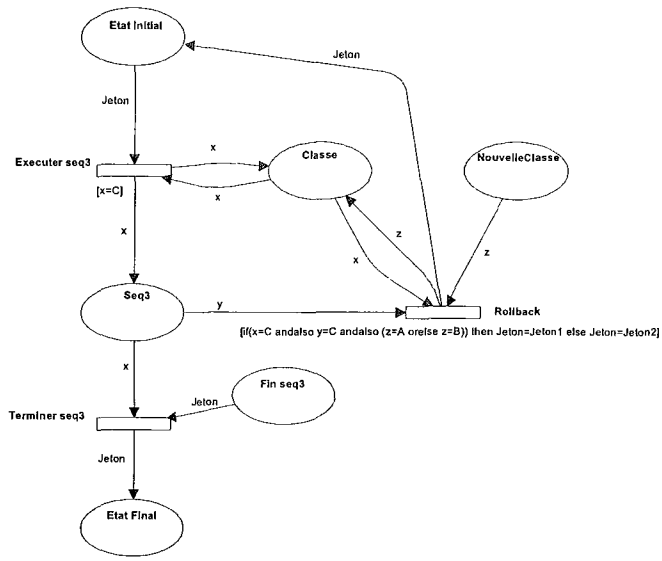


FIG. 4.9 – Séquence d'exécution 3 et action d'adaptation ROLLBACK

celles des figures précédentes).

Bloc d'adaptation

Une application peut effectuer séquentiellement plusieurs tâches différentes. Par exemple, une application de pilotage d'une installation industrielle peut inclure les trois tâches suivantes : (1) positionnement d'une pièce à l'aide d'un bras robotisé sur un centre d'usinage, (2) usinage de la pièce par le centre d'usinage et (3) dégagement de la pièce usinée à l'aide d'un deuxième bras manipulateur. Pour chaque traitement et pour une même classe de QdS, il peut être intéressant de choisir des actions d'adaptation différentes : Imaginons que cette application intègre trois séquences d'exécution associées à trois classes de QdS (A, B et C par ordre de niveau de QdS décroissant). Supposons qu'au démarrage, la classe de QdS soit A (c'est-à-dire la meilleure), l'exécution de la séquence associée est alors lancée. Cette séquence intègre le code exécutable correspondant aux trois tâches (positionnement, usinage et dégagement). Ceci implique que l'application réagira à une baisse de QdS en utilisant la même action d'adaptation quelque soit la tâche en cours. Cette stratégie peut ne pas être la plus appropriée : par exemple, nous préférerions ici qu'une baisse du niveau de QdS déclenche des actions différentes selon la tâche : exécution suspendue lors du positionnement (action BLOCK), échec lors de l'usinage (action ABORT) et poursuite de

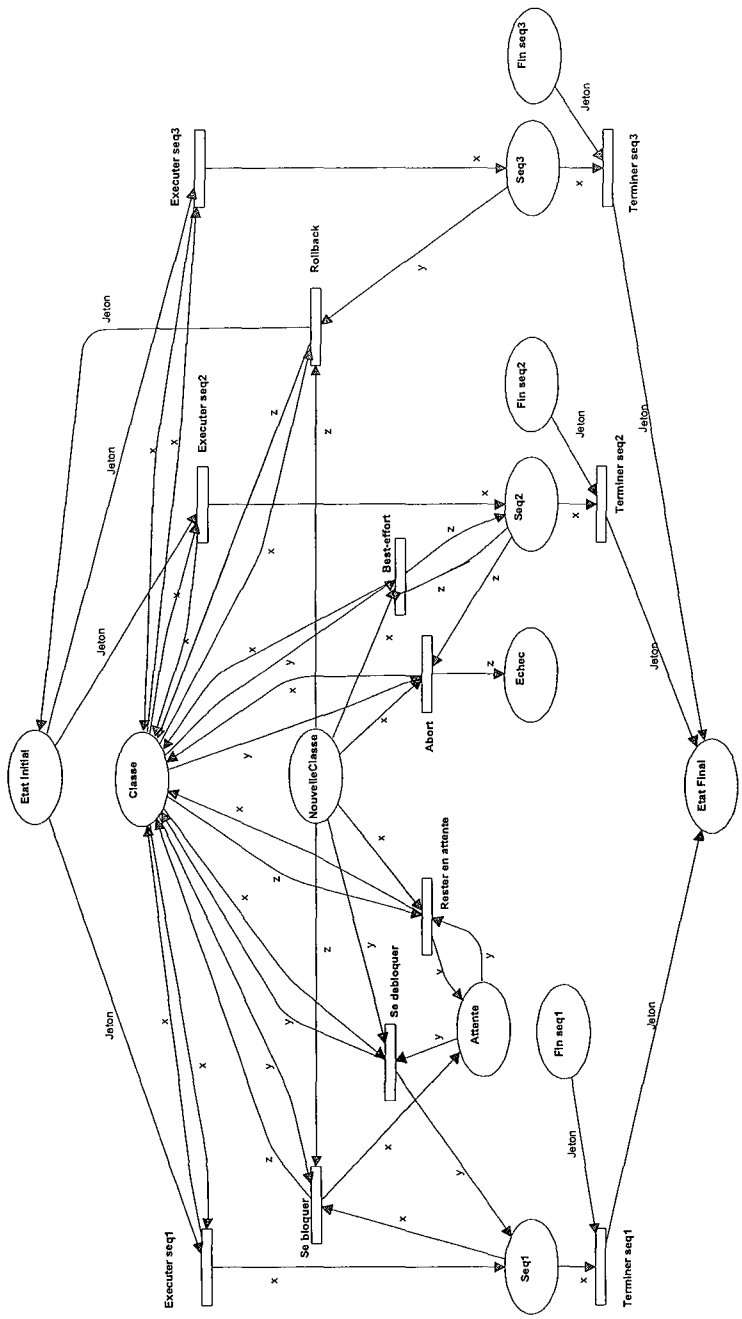


FIG. 4.10 – Réseau de Petri complet

Tâche	Classe	low_action
Positionnement	A	BLOCK
Usinage	A	ABORT
Dégagement	A	BEST_EFFORT

TAB. 4.1 – Tâches et primitives d'adaptation appropriées

l'exécution lors du dégagement (action BEST_EFFORT) (cf. tableau 4.1 et figure 4.11).

Ces cas de figure sont pris en considération par l'architecture Prayer au travers du concept de bloc d'adaptation. Un bloc d'adaptation est un ensemble de séquences d'exécution (cf. figure 4.11). Ainsi, pour chaque tâche de l'application, un bloc d'adaptation est défini et à chacune de ses séquences correspond des actions d'adaptation différentes.

Les blocs d'adaptation peuvent être modélisés en utilisant la même méthode que précédemment. Cette fois, à un bloc correspond un réseau tel que celui qui est présenté sur la figure 4.10 de la page 75. La place "Etat final" d'un bloc d'adaptation correspond alors à la place "Etat initial" du bloc suivant et les places "Classe" et "Nouvelle classe" sont communes à tous les blocs.

Conclusion

Cette méthode d'adaptation permet aux applications de fournir elles-mêmes les règles avec lesquelles elles doivent s'adapter tout en déléguant à l'architecture le soin de piloter l'adaptation. La complexité du processus d'adaptation liée à la gestion des ressources est totalement extérieure aux applications. Le principe de séparation de [Cam96] est respecté (cf. chapitre 2).

Le cadre proposé par Prayer implique de garantir la cohérence entre les classes de QdS du client et du serveur. En effet, il est nécessaire que le client et le serveur utilisent des

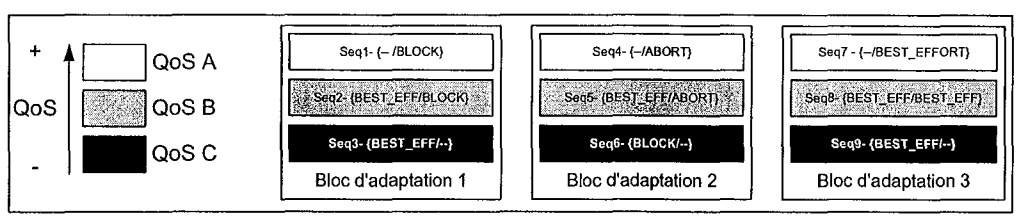


FIG. 4.11 – Classes de QoS, blocs d'adaptation, séquences d'exécution et actions d'adaptation

classes de QdS correspondantes et qu'ils réagissent de manière cohérente aux changements de classe :

D'une part, le délai entre la détection du changement de QdS sur les deux systèmes terminaux peut provoquer l'instabilité des applications.

D'autre part, il est possible qu'un système terminal seulement détecte un changement de QdS significatif. Par exemple, un hôte mobile peut détecter que le niveau d'énergie de ses batteries a franchi un seuil minimum et qu'une action d'adaptation doit être initiée. Dans un tel scénario, l'hôte mobile peut être amené à changer de classe de QdS même si le serveur n'a pas détecté qu'un changement était nécessaire.

Pour ces raisons, seul le client initie l'adaptation. Ainsi, une action d'adaptation supplémentaire "SERVER" est prévue pour l'application côté serveur. Cette action permet au serveur de recevoir une notification du client et de changer de séquence CAP comme il convient : quand le client détecte un changement de classe de QdS, sa couche gestion des ressources en notifie le serveur et lui spécifie sa nouvelle séquence CAP. Quand le serveur reçoit la nouvelle séquence CAP du client, il commute sur sa séquence CAP correspondante. Cette approche suppose que les séquences CAP du client et du serveur soient coordonnées.

4.1.3 Critique

L'architecture Prayer fournit un cadre très simple pour l'adaptation des applications. Au travers du concept de classe de QdS, les applications offrent plusieurs profils d'exécution qui sont choisis dynamiquement par l'architecture selon la disponibilité des ressources. Prayer permet aussi aux applications de préciser des règles d'adaptation par l'intermédiaire des fonctions d'adaptation. La maîtrise du processus d'adaptation est laissée à l'architecture. Ce cadre a l'avantage majeur de respecter le principe de transparence.

L'analyse de Prayer, en se fondant sur l'étude bibliographique du chapitre 2, montre que cette architecture est incomplète (cf. tableau 4.2) :

- Le nombre de paramètres de QdS utilisés est insuffisant. De manière générale, la complexité de la spécification de la QdS ne peut pas être réduite à l'expression des quatre paramètres débit, délai, pertes et coût de transmission.
- La spécification de la QdS choisie implique que le développeur de l'application exprime les besoins de l'application par des métriques (le délai par exemple). Ceci est contraire aux remarques relatives à la spécification de la QdS formulées dans le

	Architecture Prayer
Spécification de la QdS	incomplète / imprécise
Translation de la QdS	non
Préférences utilisateur	non
Surveillance de la QdS	non
Transparence/ Séparation	oui
Mécanismes de contrôle	non
Stabilité	non

TAB. 4.2 – Architecture Prayer et contraintes d'une architecture à QoS intégrée

chapitre 2. Il en résulte donc l'absence de mécanisme de traduction.

- Les préférences de l'utilisateur ne sont pas prises en compte. L'ordre de préférence des classes de QdS est arbitraire.
- La surveillance de la QdS n'est pas explicitée. Les paramètres de QdS et les techniques de mesure qui leur sont associées ne sont pas décrits.
- Aucun mécanisme de contrôle n'est prévu pour s'assurer que le trafic généré par l'application est conforme aux spécifications annoncées.
- La technique utilisée pour gérer la stabilité de l'application se contente de bloquer celle-ci dans la classe de QdS courante et ne procure pas ainsi la meilleure dynamique de contrôle de la renégociation de la classe de QdS (lorsqu'une classe de QdS de niveau supérieur est disponible, la renégociation de la classe en cours n'est enclenchée que si la disponibilité des ressources n'a pas changé durant un certain intervalle de temps).
- Les problèmes posés par l'exécution simultanée de plusieurs applications sur le même système terminal ne sont pas abordés (préférences inter-applications de l'utilisateur, etc.).

Toutefois, l'architecture Prayer est suffisamment "ouverte" pour pouvoir être complétée. C'est pourquoi l'architecture proposée dans la suite s'inspire de Prayer.

4.2 Proposition d'une nouvelle architecture, "QdS-Adapt"

4.2.1 Présentation

L'architecture "QdS-Adapt" proposée [ML02a] [ML02c] est basée sur l'architecture Prayer. Elle adopte une structure analogue en trois couches et intègre un mécanisme d'adaptation similaire basé sur les concepts de classe de QdS, de bloc d'adaptation et d'action d'adaptation.

L'architecture a été complétée afin de tenir compte des remarques formulées dans le paragraphe précédent :

- Un service de métrologie de la QdS et des agents de surveillance permettent de mettre en place une politique de surveillance des ressources adaptée pour chaque application.
- Le concept de classe de QdS est étendu et permet de spécifier la QdS aux couches application et gestion. La traduction entre les niveaux de spécification est réalisée par un élément dédié.
- Des interfaces utilisateurs permettent à l'utilisateur de saisir ses préférences pour chaque application et ses préférences inter-applications.
- Pour chaque application, un historique renseigne sur le déroulement passé de l'exécution. Le mécanisme d'adaptation en tient compte dans le but d'éviter l'instabilité de l'application.
- Des mécanismes de contrôle sont ajoutés par l'intermédiaire de gestionnaires de flux et de contrôleurs de trafic.

L'architecture est illustrée sur la figure 4.12. Les trois couches qui la composent sont : la couche "ressources", la couche "gestion", et la couche "application".

La **couche "ressources"** contient les différentes ressources, c'est à dire les ressources réseau et les ressources propres au système terminal (processeur, mémoire, etc.).

La **couche "gestion"** renferme tous les mécanismes nécessaires à la gestion de la QdS dans l'architecture. Une partie des éléments qui la composent sont uniques et sont utilisés pour la gestion de toutes les applications qui s'exécutent sur un système terminal. Il s'agit du superviseur, de l'ordonnanceur, du composant de traduction et du service de métrologie de QdS. Les autres éléments sont associés à une application. Il s'agit des classes QdS métriques, du contrôleur de trafic, de l'historique et de l'agent de surveillance.

Les différents éléments de la **couche "application"** sont l'interface utilisateur inter-applications, les "classes de QdS application", l'interface utilisateur et le gestionnaire de

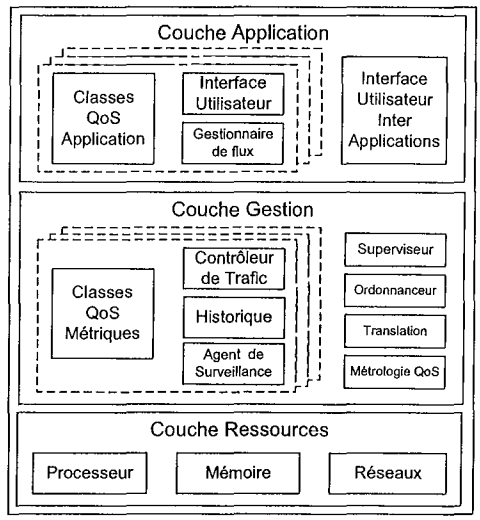


FIG. 4.12 – Représentation informelle de l'architecture de QoS proposée

flux. L'interface utilisateur inter-applications est unique et est utilisée pour toutes les applications. Les trois autres éléments sont associés à chaque application qui s'exécute sur le système terminal.

Le superviseur pilote l'adaptation de toutes les applications qui s'exécutent sur le système. Il est chargé de déterminer et d'indiquer à l'application la séquence à exécuter. Il est aussi responsable de l'activation des fonctions d'adaptation durant l'exécution d'une séquence. Le superviseur se base sur les informations délivrées par l'ordonnanceur et le composant de traduction d'une part, et par l'agent de surveillance de QdS et l'historique de chaque application d'autre part.

L'ordonnanceur définit les priorités entre les applications en fonction des préférences inter-applications fixées par l'utilisateur. Les préférences de l'utilisateur sont saisies par l'intermédiaire de l'interface utilisateur inter-applications de la couche application.

La QdS est spécifiée en termes applicatifs par le développeur sous la forme de "classes de QdS application". Elles représentent les différentes configurations qu'offre l'application. Ces configurations sont proposées à l'utilisateur par l'intermédiaire de l'interface utilisateur.

Le composant de traduction traduit les "classes de QdS application" en termes quantitatifs (métriques) sous la forme de "classes de QdS métriques". Ces dernières sont utilisées par le superviseur pour décider de la politique de surveillance nécessaire et pour déterminer

la classe de QdS dans laquelle l'application doit s'exécuter.

Un agent de surveillance est associé à chaque application. Il est chargé de mettre en œuvre la politique de surveillance élaborée par le superviseur. Il s'appuie sur le service de métrologie de QdS. Il lui indique les paramètres à mesurer et la fréquence avec laquelle ils doivent être mesurés.

Un contrôleur de trafic vérifie que le trafic généré par l'application est conforme aux spécifications annoncées. Si ce n'est pas le cas, le gestionnaire de flux (couche application) en est informé et modifie les paramètres de l'application en conséquence. Par exemple, cela peut consister à affiner les réglages d'un codec vidéo qui génère un flux vidéo plus volumineux que prévu.

Un historique renseigne le superviseur sur les classes de QdS dans lesquelles l'application s'est exécutée aux instants précédents.

Remarque : aucun composant destiné à la réservation de ressources n'est présent dans l'architecture, son objectif principal étant l'adaptation de l'exécution de l'application (contrairement à l'architecture Prayer dans laquelle des réservations de ressources sont effectuées).

Une vue du modèle de l'architecture est présentée sur la figure 4.13 qui illustre les différents éléments décrits précédemment et les relations entre ces éléments. De plus, ce diagramme de classes montre comment les concepts de bloc d'adaptation, de classe de QdS et d'action d'adaptation sont intégrés à l'architecture. Le service de métrologie de la QdS n'est pas représenté pour des raisons de simplicité. Il est explicité plus loin.

4.2.2 Fonctionnement

Cette partie décrit le fonctionnement de l'architecture de QdS. Tout d'abord, les différentes étapes de configuration qui précèdent le démarrage d'une application sont présentées. Elles sont illustrées par des diagrammes de collaboration UML. Ces diagrammes représentent une vue dynamique du système. Ils ont l'intérêt de montrer simplement les rôles joués par les objets et les interactions entre ces objets à travers l'envoi de messages. L'adaptation en ligne de l'application est ensuite décrite. Elle est illustrée par un diagramme d'activité UML. Celui-ci permet de modéliser le comportement du mécanisme d'adaptation.

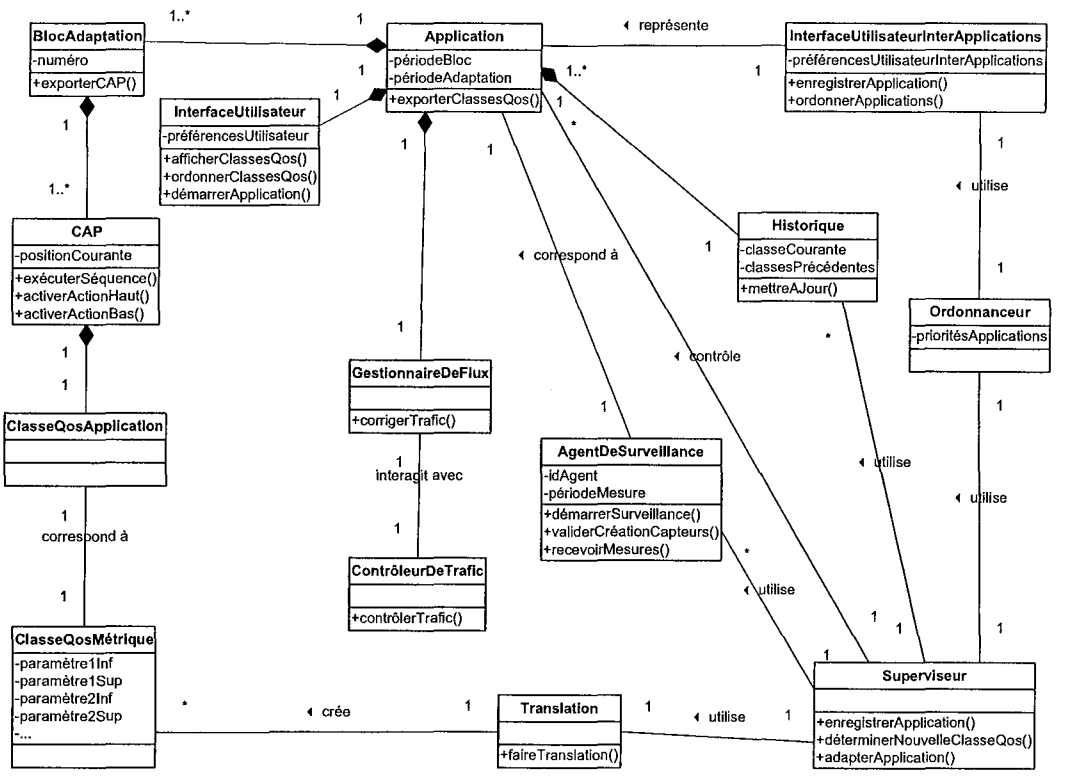


FIG. 4.13 – Modélisation UML de l'architecture

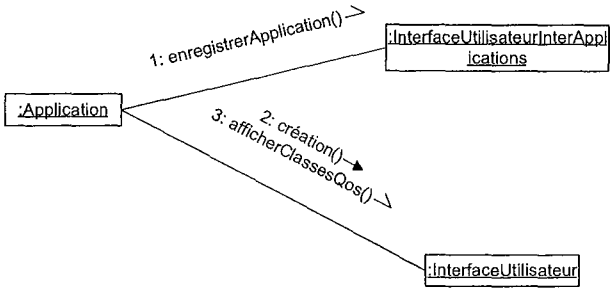


FIG. 4.14 – Enregistrement de l'application auprès de l'interface inter-applications et création de l'interface utilisateur propre

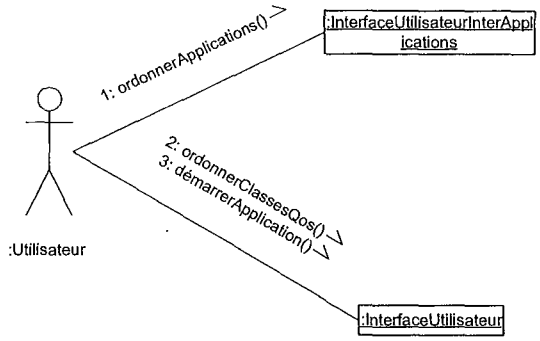


FIG. 4.15 – Saisie des préférences de l'utilisateur

4.2.2.1 Configuration de l'architecture et de l'application

Lorsqu'une application démarre, elle s'enregistre auprès de l'interface utilisateur inter-applications. L'application crée ensuite son interface utilisateur propre (cf. figure 4.14).

Une fois l'interface utilisateur de l'application créée, l'utilisateur est invité à saisir ses préférences. Il commence par indiquer ses préférences inter-applications via l'interface utilisateur inter-application. Les préférences inter-applications sont exprimées sous forme d'un ordre de préférence entre les différentes applications qui s'exécutent sur le système terminal. L'utilisateur saisit ensuite ses préférences pour l'application en utilisant l'interface utilisateur propre à l'application. L'interface lui permet d'exprimer ses préférences concernant les classes de QdS de l'application en les classant par un ordre de préférence. L'utilisateur lance ensuite l'exécution de l'application (cf. figure 4.15).

Par ailleurs, l'application s'enregistre auprès du superviseur et exporte ses classes de

QdS. A ce stade, les classes de QdS spécifient la QdS à un niveau applicatif. Ces spécifications sont ensuite traduites en spécifications quantitatives sous forme de métriques par le composant de traduction. Ce dernier crée alors une "classe de QdS métrique" pour chaque classe de QdS applicative. Une fois la traduction réalisée, le superviseur connaît les paramètres de QdS pertinents et instancie un agent de surveillance qui est responsable de l'observation des ressources pour l'application. L'agent démarre ensuite la surveillance à la demande du superviseur (cf. figure 4.16).

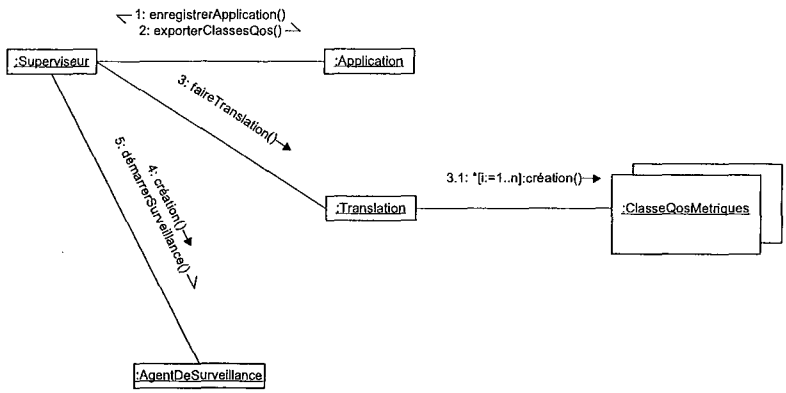


FIG. 4.16 – Translation et lancement de la surveillance de la QoS

4.2.2.2 Adaptation de l'application en ligne

Le mécanisme d'adaptation des applications est similaire à celui utilisé dans l'architecture Prayer. L'interaction entre l'application et la couche gestion des ressources s'appuie ainsi sur les concepts de classe de QdS et de bloc d'adaptation (cf. 4.1). Avant le démarrage de l'exécution d'un bloc d'adaptation, l'application spécifie un ensemble de séquences CAP. Une séquence CAP associe une classe de QdS, deux actions d'adaptation (une "ActionHaut" et une "ActionBas") et une séquence d'exécution. Le superviseur détermine alors la classe de QdS la plus adaptée et démarre la séquence d'exécution correspondante.

Cette classe de QdS est choisie en fonction des ressources disponibles, des préférences de l'utilisateur et de la priorité de l'application (une méthode de calcul est proposée ci-dessous). Les ressources disponibles sont déterminées par l'agent de surveillance. Les préférences de l'utilisateur sont fournies par l'interface utilisateur. La priorité de l'application est indiquée par l'ordonnanceur. Cette information est déduite des préférences

inter-applications. De plus, le superviseur tient compte de l'historique afin d'éviter l'instabilité de l'application (changement permanent de classe de QdS, déclenchement trop fréquent des actions d'adaptation). L'historique indique les classes de QdS dans lequel le système se trouvait précédemment et les résultats passés des calculs de la classe de QdS la plus adaptée.

Calcul de la classe de QdS la plus adaptée

L'objectif de ce calcul est de déterminer parmi les classes disponibles pour le bloc d'exécution courant, laquelle est à la fois compatible avec la QdS courante et les préférences de l'utilisateur. Le calcul proposé ici ne tient pas compte des préférences inter-applications.

Soit A l'ensemble des n couples $\{(a_i, p_i)\}$ de l'application, où a_i est une classe de QdS et p_i la préférence associée à la classe a_i. p_i est un entier dont la valeur est comprise entre 1 et n, 1 indiquant que la classe de QdS est la classe préférée, n que la classe de QdS est la classe la moins appréciée :

$$A = \{(a_i, p_i)\} \text{ avec } i = 1 \text{ à } n,\ p_i \in \{1; n\}$$

Soit B l'ensemble des couples de classes de QdS et des préférences associées du bloc d'exécution courant :

$$B \subset A$$

Soit D l'ensemble des classes de QdS de l'application disponibles en fonction de la QdS courante :

$$D \subset A$$

L'ensemble des classes de QdS disponibles pour le bloc est :

$$E = B \cap D = \{(e_j, q_j)\} \text{ avec } j = 1 \text{ à } m \text{ et } m \leqslant n$$

La classe de QdS la plus adaptée est alors :

$$e_j \text{ tel que } q_j = \min_{k=1..m} q_k$$

Durant l'exécution du bloc d'adaptation, le superviseur continue à déterminer périodiquement la classe de QdS la plus adaptée. Si celle-ci est différente de la classe en cours, l'action d'adaptation correspondante est initiée : selon que la nouvelle classe est supérieure ou inférieure à la classe en cours, le superviseur déclenche l'action "ActionHaut" ou l'action "ActionBas" (cf. figure 4.17).

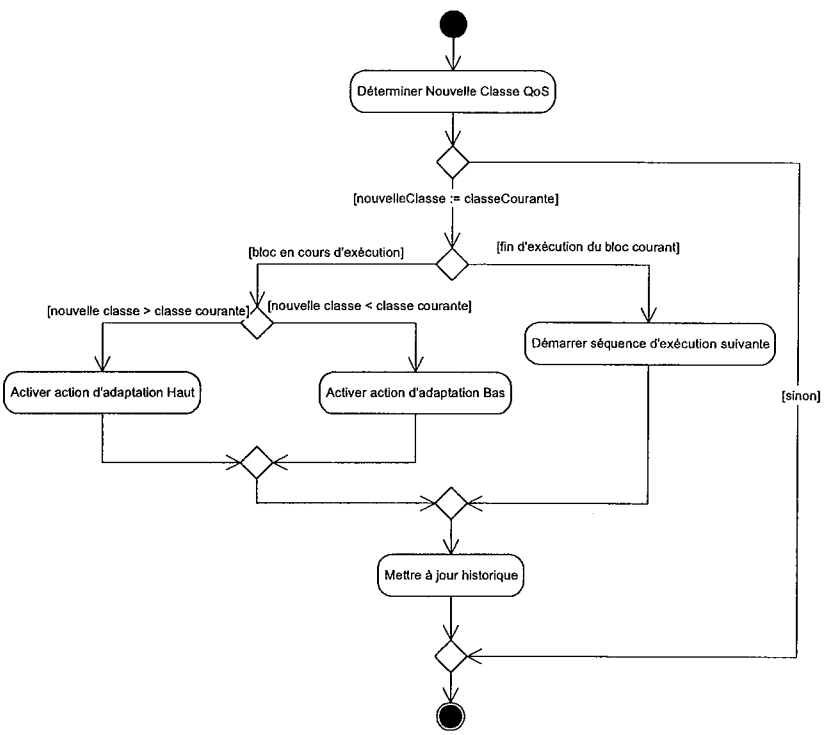


FIG. 4.17 – Adaptation en ligne

Les actions d'adaptation sont identiques à celles définies dans Prayer. L'action SWITCH a été ajoutée. Elle permet le passage d'une séquence à une autre en cours d'exécution. Les actions "ActionBAS" sont BLOCK (l'application suspend son exécution jusqu'à ce que la classe de QdS soit à nouveau disponible), BEST_EFFORT (ignore les changements de QdS), ROLLBACK (exécute à nouveau le bloc d'adaptation courant avec la nouvelle classe de QdS), ABORT (annule et quitte le bloc d'adaptation en cours), et SWITCH (passage d'une séquence à une autre en cours d'exécution). Les actions "ActionHAUT" sont BEST_EFFORT, ROLLBACK (avec la nouvelle classe de QdS), ABORT et SWITCH.

L'intérêt des actions d'adaptation réside dans leur capacité à définir la manière appropriée avec laquelle l'application doit réagir à un changement de QdS sans attendre le bloc d'adaptation suivant. Par exemple, quand l'architecture est utilisée pour une application de diffusion vidéo, et que la nouvelle classe de QdS est inférieure en raison d'un taux de perte supérieur au taux correspondant à cette classe, l'application peut continuer son exécution sans échouer. Dans ce cas, les actions de type BEST_EFFORT sont appropriées car un blocage du système est inutile et la renégociation de la classe de QdS peut attendre la fin du bloc d'adaptation courant. Les actions BLOCK, ROLLBACK ou SWITCH sont utiles par exemple aux applications pour lesquelles un dysfonctionnement de l'une d'entre elles peut menacer la sécurité du système et de son environnement. C'est le cas des applications d'asservissement à distance. Une violation de la classe de QdS peut mener à l'instabilité du système asservi. Il peut être alors judicieux de bloquer le système jusqu'à ce que la classe de QdS soit à nouveau disponible, ou bien de recommencer l'exécution du bloc courant avec une classe de QdS plus adaptée, ou encore de changer de classe de QdS à la volée.

La période avec laquelle le superviseur détermine la classe de QdS la plus adaptée (ou période d'adaption) dépend de l'application : selon l'application, l'exécution est découpée en blocs d'adaptation périodiques ou non-périodiques (pour chaque cas, nous donnons un exemple dans le chapitre 5). Lorsque les blocs sont périodiques, la période d'adaptation est un sous-multiple de la période des blocs : la nouvelle classe de QdS est déterminée n fois pendant l'exécution d'un bloc. n peut alors être imposé par l'architecture ou choisi par le développeur de l'application, et la période d'adaptation est ensuite calculée automatiquement. Lorsque ce n'est pas le cas, la période peut par exemple être fixée arbitrairement par le développeur.

Remarque : la période d'adaptation conditionne la période de mesure puisque de nouvelles mesures sont nécessaires à chaque fois que l'architecture détermine la classe de QdS

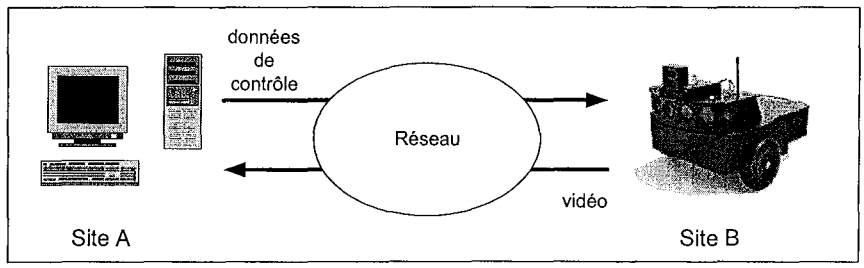


FIG. 4.18 – Application de télé-robotique

la plus adaptée.

4.3 Service de métrologie de la QdS

Le rôle du service de métrologie est de mesurer la QdS offerte par le réseau entre deux systèmes terminaux. Il correspond au bloc "Métrologie de QdS" de l'architecture (cf. figure 4.12).

Le service de métrologie permet de réaliser des mesures bidirectionnelles, c'est-à-dire pour chaque sens de la communication (aller et retour). En effet, l'adaptation de l'application peut nécessiter de connaître les caractéristiques du chemin dans les deux sens. La figure 4.18 illustre l'exemple d'une application de télé-opération d'un robot mobile. Le robot situé sur le site B est télé-piloté par un opérateur depuis le site A. Les données de contrôle sont envoyées au robot par l'opérateur. Une caméra offre à l'opérateur un retour visuel. Le chemin aller peut être différent du chemin retour. Pour chaque trafic, les paramètres pertinents sont différents : par exemple, le taux de perte de A à B permet d'évaluer la fiabilité du transfert des données de contrôle. La bande passante disponible de B à A conditionne le choix de la compression à employer pour le retour vidéo. Le degré global d'interactivité est exprimé par le délai aller-retour (mesuré depuis le site A).

Le service de métrologie peut employer des techniques de mesures passives ou actives. Les mesures passives sont effectuées en observant (en capturant) le trafic utile de l'application et en l'analysant. Cette analyse requiert une connaissance approfondie du trafic observé. Les mesures passives n'induisent pas de trafic supplémentaire sur le réseau. Au contraire, les mesures actives sont réalisées en envoyant des "paquets-sonde" au récepteur au travers du réseau. Des mesures actives sont indispensables puisque l'architecture a be-

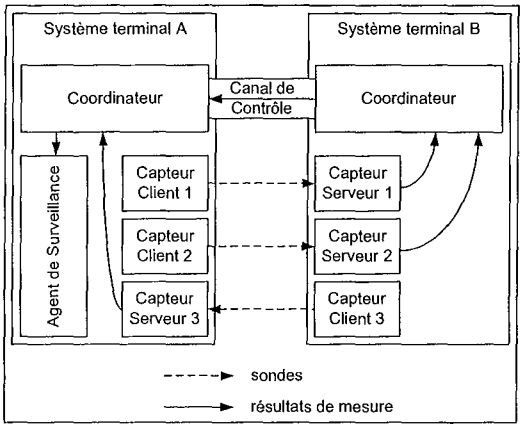


FIG. 4.19 – Agent de surveillance et service de métrologie de la QoS

soin de connaître la QdS disponible avant de débuter l'exécution d'un bloc d'adaptation. Les mesures passives peuvent être utilisées en complément.

Le service de métrologie est conçu de façon modulaire : les techniques de mesures sont implémentées sous la forme de boîtes appelées "capteurs". N'importe quelle technique de mesure présentée dans le chapitre 3 peut y être intégrée a priori. Utilisé entre deux systèmes terminaux, le service de métrologie de l'architecture permet ainsi de mesurer plusieurs paramètres différents simultanément et dans les deux sens de la communication.

La fréquence avec laquelle les mesures sont effectuées est paramétrable, ce qui permet de définir une politique de surveillance adaptée à la fréquence des blocs de l'application.

Le service de métrologie a fait l'objet de deux publications dans des conférences internationales [ML02b] [ML02d].

4.3.1 Principe

Le service de métrologie est illustré sur la figure 4.19. Il est composé d'un coordinateur et de capteurs. Ces entités coopèrent avec les agents de surveillance des différentes applications (cf. 4.2.1).

Sur le système terminal client (c'est-à-dire qui initie l'adaptation), un agent de surveillance est instancié par le superviseur. L'agent s'adresse au service de métrologie (en particulier au coordinateur) et demande la mise en œuvre de la politique de surveillance pour une application donnée. Le coordinateur établit un canal de contrôle entre lui et son homo-

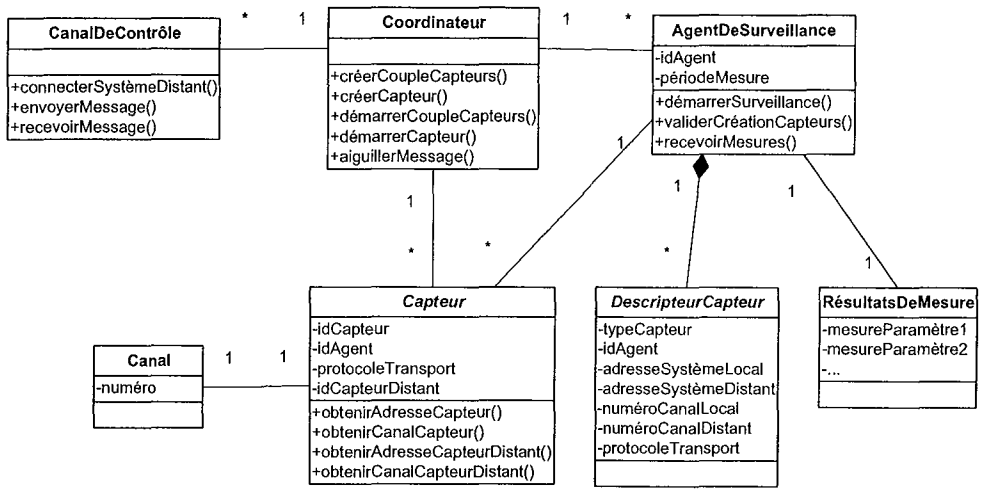


FIG. 4.20 – Modélisation du service de métrologie

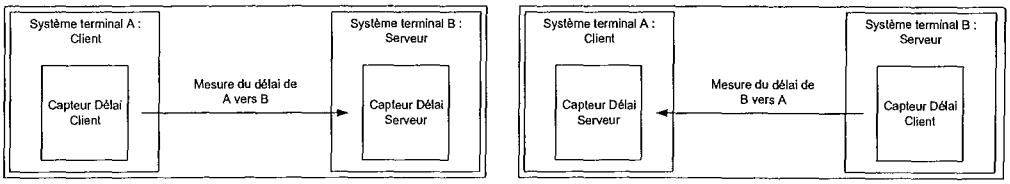
logue du système terminal serveur. Ce canal est employé par les deux coordinateurs pour coopérer et mettre en place des capteurs qui sont les entités qui effectuent les mesures. Durant la phase de mesure, les capteurs envoient les résultats à l'agent de surveillance.

Le canal de contrôle et les canaux employés par les capteurs pour réaliser les mesures sont distincts. Le canal de contrôle utilise le protocole de transport TCP en raison de sa fiabilité. Le(s) protocole(s) de transport employé(s) par les capteurs pour envoyer les paquets-sonde dépend(ent) de la mesure demandée et de la technique de mesure mise en oeuvre.

Une vue du modèle du service de métrologie est présentée sur la figure 4.20 qui illustre les différents éléments décrits précédemment ainsi que les éléments supplémentaires "RésultatsDeMesure" et "DescripteurCapteur" : l'agent de surveillance stocke les résultats de mesure dans la classe "RésultatsDeMesure". Les classes "DescripteurCapteur" lui permettent de spécifier les caractéristiques des capteurs dont il a besoin.

4.3.1.1 Agent de surveillance

Un agent de surveillance est instancié par le superviseur et définit la politique de surveillance pour une application (paramètres à mesurer, période de mesure). Il décrit les caractéristiques des capteurs à mettre en place. Il regroupe les résultats des mesures demandées pour une application et les met à disposition du superviseur.



a. b.

FIG. 4.21 – Système terminaux client et serveur, capteurs client et serveur

4.3.1.2 Capteurs

Les capteurs sont les entités qui réalisent les mesures. Ils peuvent, a priori, implémenter n'importe quelle technique de mesure exposée dans le chapitre 3. Dans les travaux présentés, nous nous sommes essentiellement intéressés aux techniques de mesures actives. Dans ce cas, la mesure d'un paramètre entre deux systèmes terminaux consiste à mettre en place un couple de capteurs, dans lequel un capteur joue le rôle d'émetteur de paquets-sonde et l'autre de récepteur. Par définition, l'émetteur de sonde est appelé le capteur client, le récepteur de sonde le capteur serveur.

La figure 4.21 illustre le cas dans lequel le système A demande au système B la mise en place d'un couple de capteurs (A est le client, B est le serveur). Dans le premier cas (a), on souhaite mesurer le délai de A vers B. Le capteur client (émetteur de paquets-sonde) est placé sur A et le capteur serveur (récepteur de paquets-sonde) sur B. Dans le deuxième cas (b), on souhaite mesurer le délai de B vers A. Le capteur client est alors sur B, le capteur serveur sur A. Les notions de client et serveur pour les capteurs et pour les systèmes terminaux sont totalement indépendantes. De plus, le même couple de capteurs est utilisé pour mesurer le délai unidirectionnel dans les deux sens. Il suffit d'inverser la position des capteurs pour changer la direction selon laquelle la mesure est effectuée. Cette remarque est valable pour tous les paramètres mesurés.

Un capteur est associé à un seul agent de surveillance. Quand il produit une mesure, il la fait parvenir à son agent de surveillance.

Des exemples d'implémentation de capteurs sont donnés dans le chapitre 5.

Remarque : des capteurs peuvent aussi être développés pour réaliser des mesures passives, ou pour mesurer les ressources des systèmes terminaux (niveau de batterie, etc.).

4.3.1.3 Coordinateur et canal de contrôle

Un coordinateur est présent sur chaque système terminal. Les coordinateurs ont les rôles suivants :

- Mettre en place, démarrer et arrêter les couples de capteurs sous le contrôle d'un agent de surveillance.
- Assurer la communication entre les entités (capteurs, agents de surveillance) situées sur des systèmes terminaux différents.

La communication entre deux coordinateurs est réalisée au travers d'un canal de contrôle qui n'est accessible qu'à eux seuls. Deux types de messages circulent sur le canal de contrôle. Le premier sert à la communication directe entre les coordinateurs qui se servent de ces messages pour collaborer par exemple à la mise en place d'un couple de capteurs. C'est le cas du message "DemandeNouveauCapteur" qui est employé par un coordinateur pour demander à un autre coordinateur de créer un capteur. Le second type de messages est utilisé par les capteurs et leur agent de surveillance pour communiquer entre eux. Dans ce cas, les coordinateurs ont un rôle "d'aiguilleurs". Par exemple, lorsqu'un capteur doit faire parvenir une mesure à son agent de surveillance, il l'envoie sous la forme d'un message "Mesures" au coordinateur qui est sur le même système terminal que lui. Ce dernier détermine où se situe l'agent de surveillance destinataire. Si l'agent est sur le même système terminal, le coordinateur lui fait suivre le message directement. Dans le cas contraire, le message est transmis au coordinateur du système terminal sur lequel est situé l'agent de surveillance. Ce coordinateur se charge ensuite de transmettre le message à l'agent de surveillance.

4.3.2 Fonctionnement

Une procédure de mesure typique est illustrée sur la figure 4.22 : soit deux systèmes terminaux A et B, un agent de surveillance situé sur A demande à son coordinateur la mise en place d'un couple de capteurs sur A et B. L'agent indique au coordinateur les caractéristiques des capteurs désirés. Le coordinateur commence par réserver un port de communication sur le système A qui sera utilisé par la suite par le nouveau capteur. Le coordinateur envoie ensuite un message de demande de création de capteur au coordinateur de B. Ce message renferme les spécifications du capteur demandé sur B (ainsi que le numéro de port utilisé par le capteur sur A). Dès que ce message est reçu par le coordinateur de B, celui-ci réserve un port sur B, crée le capteur demandé, l'associe au port réservé et retourne

un message de confirmation de création de capteur au coordinateur de A. Ce message contient l'identifiant du capteur nouvellement créé. Un identifiant de capteur est unique. C'est une chaîne de caractères qui contient notamment l'adresse IP du système sur lequel est situé le capteur, et le port de communication utilisé par le capteur. Le coordinateur de A crée alors le capteur demandé sur A et retourne les identifiants des deux capteurs à l'agent de surveillance. L'agent de surveillance demande à son coordinateur de débuter la phase de mesure. Le coordinateur de A doit alors démarrer l'exécution des capteurs en commençant par celle du capteur qui est chargé de recevoir les sondes. Dans l'exemple présenté, le capteur qui reçoit les sondes est situé sur B. Pour le démarrer, le coordinateur de A envoie un message de démarrage au coordinateur de B. Une fois le capteur mis en action, le coordinateur de B en informe le coordinateur de A en lui envoyant un message de confirmation. Le coordinateur de A démarre ensuite le capteur situé sur A (l'émetteur de sonde).

Durant la phase de mesure, le capteur qui réalise les mesures (ici, le capteur situé sur B) appelle périodiquement son opération dédiée au calcul de la mesure (ici, "mesurerDélai()"). Quand l'opération est terminée, le résultat est envoyé au coordinateur local (ici, le coordinateur de B). Celui-ci le fait suivre à l'agent de surveillance (ici, sur A). Le coordinateur de B transmet alors le résultat de la mesure au coordinateur de A qui le fait suivre à l'agent de surveillance.

4.4 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle architecture à QdS intégrée, l'architecture "QdS-Adapt", dédiée à l'adaptation des applications aux performances du réseau. Au travers des concepts de "classes de QdS" et de "blocs d'adaptation", les applications spécifient leur politique d'adaptation dynamique aux variations de QdS. L'adaptation des applications est pilotée par une couche de gestion qui inclut un service de métrologie de la QdS. Les applications fournissent elles-mêmes les règles d'adaptation à cette couche qui les applique en s'appuyant sur les mesures du service de métrologie. La conception de ce dernier autorise l'implémentation sous forme de capteurs de n'importe quelle technique de mesure exposée dans le chapitre 3.

L'implémentation du service de métrologie et de l'architecture "QdS-Adapt" intégrée sur une plateforme de télé-opération est présentée dans le chapitre suivant.

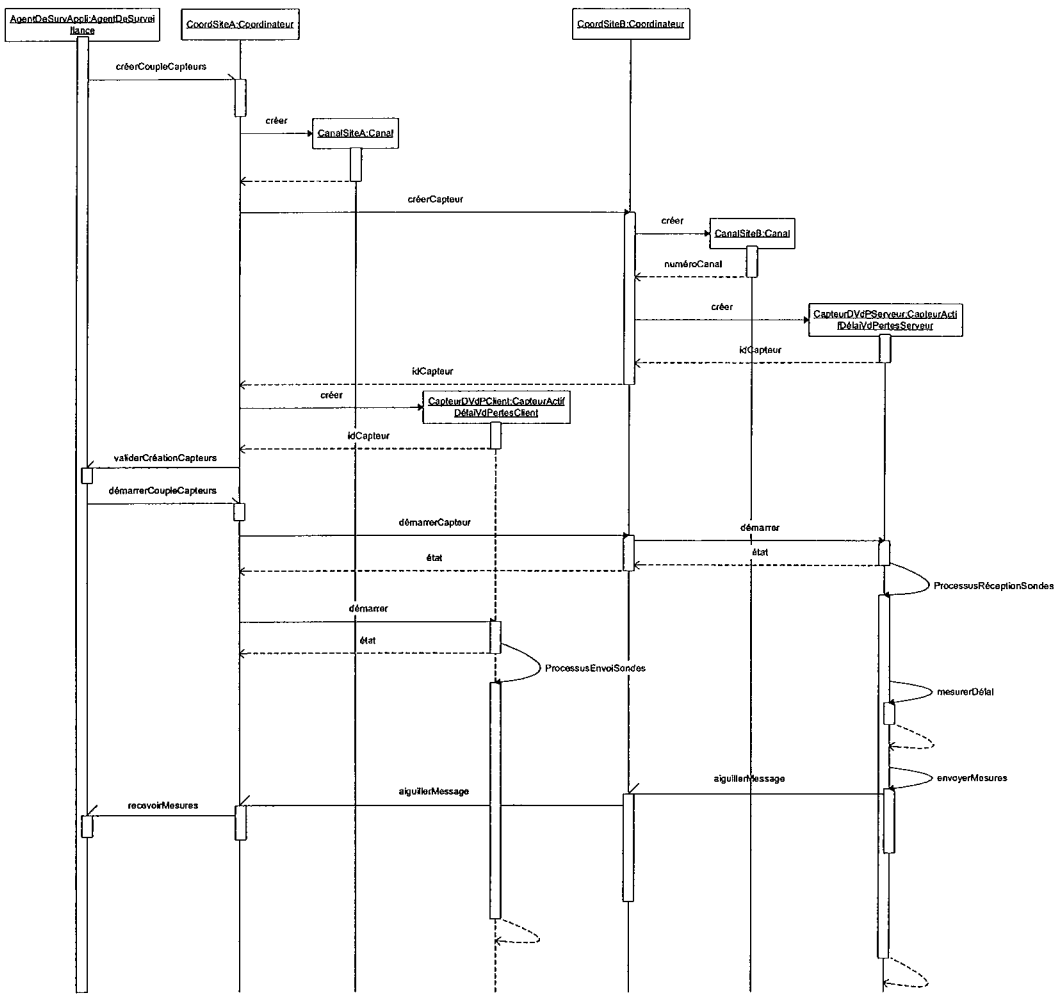


FIG. 4.22 – Mise en place de capteurs et démarrage du processus de mesure

Chapitre 5

Expérimentations

Ce chapitre est consacré à l'implémentation du service de métrologie et à celle de l'architecture "QdS-Adapt" sur une plateforme de télé-opération d'un robot mobile. Une application "récréative" de télé-pilotage du robot mobile et une application d'asservissement à distance de la position du robot ont été développées. Ces deux applications adaptent leur exécution au délai aller-retour mesuré entre le robot et son poste de commande. De par sa nature, la première application a un intérêt essentiellement démonstratif. Quant à la seconde, elle permet d'obtenir des résultats expérimentaux exploitables d'un point de vue scientifique. Tous les développements ont été réalisés en C/C++ sur Linux.

La première partie de ce chapitre présente l'implémentation du service de métrologie en détaillant les capteurs de mesure développés. Après avoir défini la télé-opération et la problématique qu'elle induit dans 5.2.1, la plateforme de télé-opération est décrite dans 5.2.2. Les deux applications de télé-opération sont alors présentées dans 5.2.3 et 5.2.4 respectivement.

5.1 Implémentation du service de métrologie

Le service de métrologie de la QdS a été implémenté pour réaliser des mesures sur les réseaux IP pour les protocoles de transport UDP et TCP. A ce jour, le service intègre des capteurs pour la mesure active, c'est à dire par l'émission de paquets-sonde, du délai unidirectionnel, de la variation de délai, des pertes de paquets et du délai aller-retour. De plus, une interface graphique a été développée pour permettre l'emploi du service dans un contexte différent de celui de l'architecture de QdS (cf. figure 5.1).

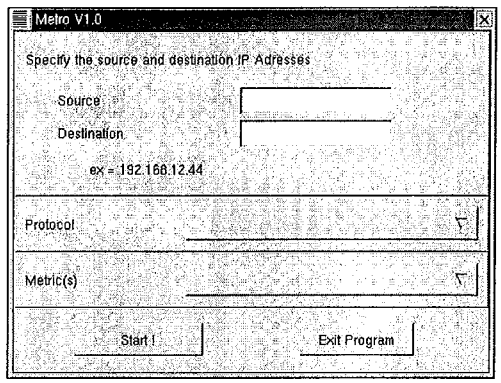


FIG. 5.1 – Interface graphique du service de surveillance

La figure 5.2 montre les classes relatives aux capteurs de mesure du délai, de la variation du délai et des pertes (classes CapteurActifDélaiVdPertesClient et CapteurActifDélaiVdPertesServeur) et du délai aller-retour (classes CapteurActifRttClient et CapteurActifRttServeur).

5.1.1 Capteurs de mesure du délai, de la variation de délai et des pertes

Ces trois paramètres sont mesurés en mettant en place un capteur client qui envoie périodiquement des sondes à un capteur serveur. La période d'envoi des paquets-sonde et leur taille sont spécifiées respectivement par les attributs "périodeSondes" et "tailleSondes". Les attributs booléens "délai", "variationDélai" et "pertes" indiquent au capteur serveur les paramètres qu'il doit mesurer.

Les paquets-sonde contiennent une estampille temporelle indiquant l'instant de leur expédition et un numéro de séquence. Pour chaque paquet, le délai unidirectionnel est calculé par la différence entre son instant d'arrivée au capteur serveur et l'estampille temporelle qu'il contient. Les horloges des systèmes sont supposées synchronisées (en utilisant un réseau de serveurs NTP par exemple). La variation de délai est égale à la différence entre les délais de deux paquets consécutifs. Un paquet est considéré perdu lorsque son délai dépasse la valeur indiquée par l'attribut "timeout".

Le capteur serveur appelle les opérations associées à la mesure des paramètres désirés ("mesurerDélai()", mesurerVariationDélai()" et "mesurerPertes()") périodiquement selon la période indiquée par l'attribut "périodeMesure". Selon les opérations utilisées, il calcule le taux de pertes moyen et/ou le délai moyen et/ou la variation de délai maximum

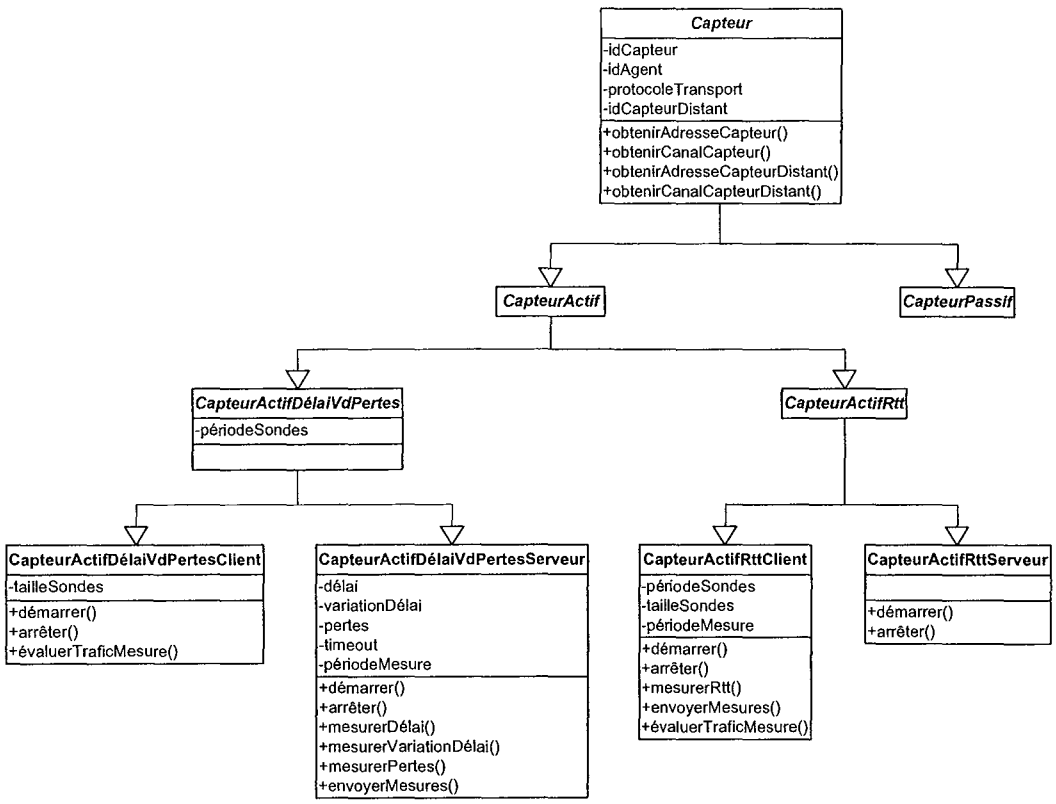


FIG. 5.2 – Diagramme de classes de capteurs

observés lors de la période de mesure qui s'achève (c'est-à-dire depuis le dernier appel de l'opération). L'opération "envoyerMesures()" permet ensuite d'envoyer les résultats à l'agent de surveillance.

L'opération "évaluerTraficMesure()" permet de calculer le débit du trafic de paquets-sonde.

5.1.2 Capteurs de mesure du délai aller-retour

Pour mesurer le délai aller-retour, un capteur client envoie des paquets-sonde à un capteur serveur. A leur arrivée au capteur serveur, ce dernier les retourne immédiatement au capteur client. La période d'envoi des paquets-sonde et leur taille sont spécifiées respectivement par les attributs "périodeSondes" et "tailleSondes". Les paquets-sonde sont estampillés lors de leur envoi. Pour chaque paquet, le capteur client calcule le délai aller-retour par la différence entre leur instant d'arrivée et leur instant d'envoi.

Le capteur client appelle l'opération "mesurerRtt()" périodiquement selon la période indiquée par l'attribut "périodeMesure". Il calcule alors le délai aller-retour moyen observé durant la période de mesure qui s'achève (c'est-à-dire depuis le dernier appel de l'opération). Le résultat est ensuite transmis à l'agent de surveillance en employant l'opération "envoyerMesures()".

L'opération "évaluerTraficMesure()" permet de calculer le débit du trafic de paquets-sonde.

5.2 Application à la télé-opération d'un robot mobile

5.2.1 Télé-opération et systèmes à retard

5.2.1.1 Définition

La télé-opération consiste à contrôler un système automatisé (robot, etc.) à distance. L'objectif initial de la télé-opération était d'assister l'homme dans l'accomplissement de tâches complexes lorsque celui-ci est soit inefficace, soit dans l'impossibilité d'agir (environnement hostile, faible accessibilité, etc.) et là où le robot autonome n'existe pas encore, faute d'intelligence suffisante [Lel00].

Le schéma classique de télé-opération a une structure "maître-esclave", aussi est-il constitué de deux parties interactives :

- l'opérateur commande le robot via une interface de commande, le maître,

- la partie opérative du robot, l'esclave, répond aux ordres du maître et lui transmet en retour les données qu'il a captées.

La transmission entre le maître et l'esclave a d'abord été mécanique (engrenages, câbles, etc.), l'opérateur possédait une vision directe sur la zone de travail (à travers une paroi transparente par exemple). Un inconvénient majeur de ce type de dispositif est la limitation de la distance maître-esclave. Par la suite, le passage à une transmission électrique (filaire ou radio, analogique ou, plus récemment, numérique) a ouvert de nouvelles perspectives. L'opérateur a désormais la possibilité d'enregistrer des séquences de manipulations que le robot répétera automatiquement, et surtout cela permet d'augmenter la distance maître-esclave. Ensuite, des expérimentations de télé-opération de robots via l'Internet sont apparues au milieu des années 90. L'emploi des technologies de l'Internet a ouvert le champ à de nouvelles applications en raison de l'augmentation des distances et du degré de sophistication des interfaces homme-machine qu'elles rendent possibles : le télé-enseignement qui permet par exemple à des étudiants d'accéder à des systèmes automatisés de production partagés entre plusieurs établissements universitaires, la télé-médecine qui met à disposition les compétences d'un spécialiste pour le diagnostic ou l'intervention chirurgicale à distance. L'Internet a permis aussi le développement de nouvelles applications commerciales en mettant la robotique au service du grand public (visite virtuelle, commerce électronique, etc.). Plus généralement, différents bénéfices peuvent être retirés de l'intégration des technologies de l'Internet dans les architectures de contrôle industriel : technologie peu onéreuse, interfaces homme-machine sophistiquées, personnalisables et indépendantes de la plate-forme matérielle, messageries pour la communication entre opérateurs, environnements intégrés permettant une remontée d'informations et une gestion de la production, de la qualité, etc [Ogo01].

5.2.1.2 Problèmatique de la télé-opération

L'utilisation d'un réseau de communication entre le maître et l'esclave affecte les performances de la boucle de télé-opération. En effet, selon la QdS disponible sur le réseau (délai, pertes de paquets, etc.) les informations transférées sont sujettes à des retards et/ou des altérations.

La présence de retard dans la boucle rend la télé-opération plus difficile pour l'opérateur et peut conduire le système à l'instabilité. L'impact du retard sur les applications de télé-opération a fait l'objet de nombreuses études. Par exemple, dans [JB99], l'influence

combinée des pertes de paquets et du délai sur la performance de stabilité (marge de phase) d'un asservissement est étudiée (les pertes sont considérées comme une altération de la période des échantillons). Ce travail montre l'intérêt de relier la QdS du réseau aux indices de performances spécifiques des applications.

La majorité des travaux dont l'objectif est la stabilisation des systèmes télé-opérés en présence de retard, concernent les systèmes pour lesquels le délai est constant [OF97]. Pour ces derniers, il est possible de déterminer des conditions de stabilité en fonction du retard et des caractéristiques du système. Malheureusement, les retards introduits par l'utilisation d'un réseau de type Internet sont variables par nature, et les résultats obtenus dans les études précédentes sont le plus souvent inapplicables. Face à ce problème, [Lel00] propose de réguler les retards en vue de les rendre constants. Cette approche permet d'utiliser les résultats obtenus pour les systèmes à délai constant.

Remarque : dans [Sri02], la théorie de l'automatique stochastique est employée pour prendre en compte la variabilité du retard. La condition de stabilité obtenue est alors elle aussi stochastique et il n'est pas possible de garantir une stabilité stricte du système.

5.2.2 Plate-forme d'expérimentation

L'architecture présentée précédemment (incluant le service de métrologie) est mise en œuvre sur une plateforme de télé-opération d'un robot mobile. Cette plateforme (cf. figure 5.3) est composée d'un robot mobile, d'un ordinateur et d'une caméra embarqués et d'un poste de commande. Le robot et son ordinateur embarqué sont interconnectés par un port série. L'ordinateur embarqué est connecté au réseau par une technologie sans fil (carte et point d'accès ethernet sans-fil 802.11b). La caméra embarquée est équipée d'un émetteur radio-fréquence. Le signal vidéo analogique est capté par un récepteur radio-fréquence connecté à un serveur vidéo http qui échantillonne ce signal et le met en ligne sous forme d'images JPEG.

5.2.3 Application de télé-pilotage

L'objectif de l'application est de permettre à un opérateur situé sur le poste de commande de piloter le robot mobile afin de lui faire suivre le trajet illustré sur la figure 5.4 (A-B-C-D-E). Cela implique la circulation de différents flux entre l'opérateur et le système mobile : commande de l'opérateur vers le robot, retour vidéo et mesures issues des sonars du robot vers l'opérateur. Le fonctionnement correct du système est conditionné

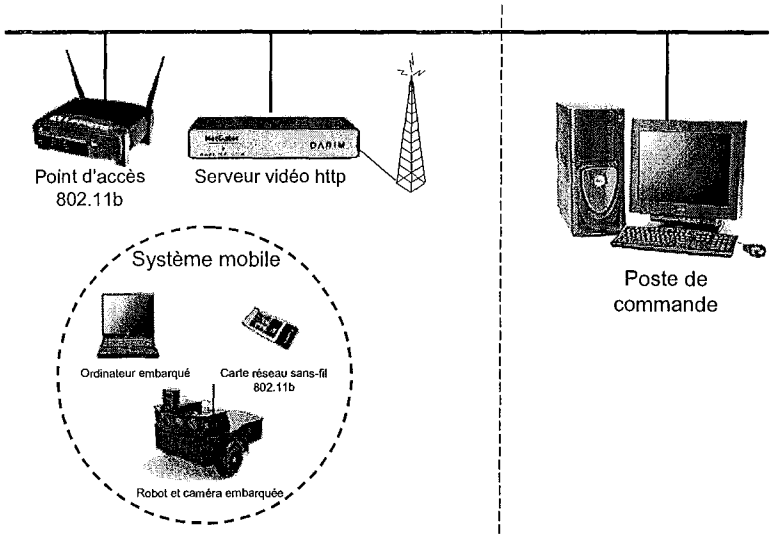


FIG. 5.3 – Plateforme d'expérimentation

par l'interactivité disponible entre l'opérateur et le système mobile, c'est-à-dire par le délai aller-retour.

5.2.3.1 Stratégie d'adaptation

Pour faire face aux problèmes introduits par le réseau, une stratégie intéressante consiste à changer le degré d'autonomie du robot selon l'importance des perturbations [LWD96] : dans le cas présent, le contrôle du robot sera partagé entre le robot et l'opérateur. Plus le délai aller-retour sera important, moins les fonctions de contrôle seront délo-

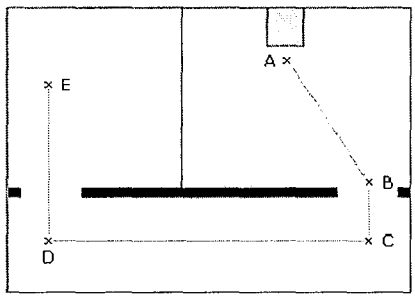


FIG. 5.4 – Trajet du robot

calisées, c'est à dire confiées à l'opérateur, et plus le robot maîtrisera son déplacement lui-même. Ainsi, on définit les trois modes de pilotage suivants :

- Dans le mode « libre » ou « joystick », le client maîtrise complètement les mouvements du robot. Ce mode de pilotage interactif ne permet un contrôle fiable et sans danger du robot que lorsque le délai aller-retour est faible.
- Dans le mode « point à point », le robot maîtrise complètement sa trajectoire, le client se contente d'indiquer le point de référence du trajet à atteindre. Ce mode est utile quand l'interactivité entre l'opérateur et le robot est impossible, c'est à dire lorsque le délai aller-retour est très élevé.
- Dans les situations intermédiaires, on peut utiliser le mode « assisté » ou « rail ». Dans ce cas, la maîtrise des degrés de liberté du robot est partagée entre le robot et l'opérateur : le robot contrôle sa propre orientation, l'opérateur maîtrise la vitesse linéaire du robot.

5.2.3.2 Mise en œuvre de l'architecture

L'application de pilotage [GML03] est divisée en deux sous-applications : l'une d'elles est située sur le poste de commande, l'autre sur le PC embarqué. Le poste de commande est le système terminal client, c'est sur celui-ci que l'adaptation de l'application est pilotée.

L'application située sur le poste client offre une interface graphique (cf. figure 5.5) qui permet à l'opérateur d'interagir avec le robot, de visualiser sa position sur un plan, sa vitesse, son orientation et l'état de ses sonars. L'interface bénéficie également du retour vidéo de la caméra embarquée placée sur le robot. Les images sont téléchargées du serveur http périodiquement. L'opérateur pilote le robot en utilisant les flèches directionnelles du clavier ou en cliquant sur les boutons de contrôle de l'interface graphique. L'application envoie les ordres de contrôle de l'opérateur à l'application embarquée.

L'application située sur le PC embarqué dialogue avec la carte-contrôleur du robot via le port série pour connaître la position, la vitesse, la direction et l'état des sonars du robot. Elle fait parvenir périodiquement ces informations à l'application cliente. Elle reçoit les consignes de déplacements envoyées par l'application du poste client, les interprète et communique les ordres correspondants à la carte-contrôleur.

Trois classes de QdS correspondant aux trois modes de pilotage sont définies. Elles sont implémentées dans l'application du poste client et dans celle du PC embarqué :

Du côté client, la saisie des commandes de l'opérateur est modifiée selon le mode

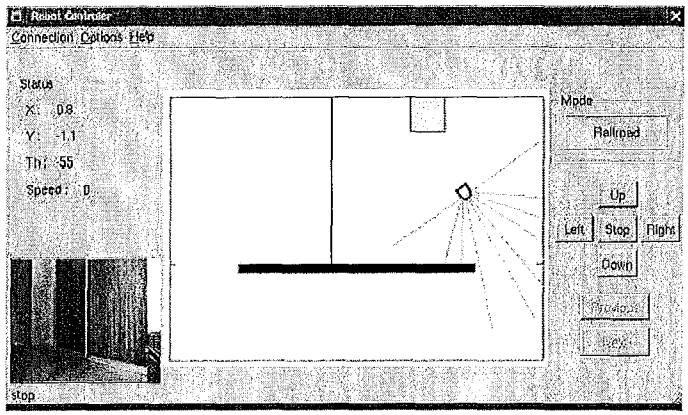


FIG. 5.5 – Interface graphique de l'application

de pilotage. Dans le mode "libre", l'opérateur indique au robot la direction à suivre en utilisant les quatre flèches directionnelles du clavier de l'ordinateur ou en cliquant sur les boutons "Up", "Down", "Left" ou "Right". Dans le mode "assisté", seules les flèches vers le haut et vers le bas du clavier et les boutons "Up" et "Down" sont pris en compte (les boutons "Left" et "Right" sont grisés) pour indiquer au robot s'il doit avancer ou reculer. Dans le mode "point à point", les flèches haut et bas et les boutons "Previous" et "Next" indiquent au robot d'aller respectivement au point suivant et précédent.

Du côté serveur, l'application intègre trois algorithmes de contrôle associés aux modes de pilotage. Dans le mode "libre", l'application reçoit les commandes et les fait suivre au robot. Dans le mode "assisté", l'application calcule la direction que le robot doit suivre pour se rendre au point suivant ou au point précédent. Elle élabore les ordres de pilotage selon la direction à prendre et les commandes de l'opérateur relatives au déplacement vers l'avant ou vers l'arrière. Dans le mode "point à point", l'application calcule la route que le robot doit emprunter pour se rendre au point suivant ou précédent selon d'une part les commandes envoyées par l'opérateur et d'autre part la position courante, et elle pilote le robot en conséquence.

Un panneau destiné à la saisie des préférences de l'utilisateur est prévu sur l'application du poste client. L'utilisateur exprime ses préférences en ordonnant les classes de QdS (cf. figure 5.6).

Les valeurs admissibles du délai aller-retour (rtt_{max}) pour chaque mode sont déterminées expérimentalement. Les valeurs obtenues (cf. tableau 5.1) sont approximatives car

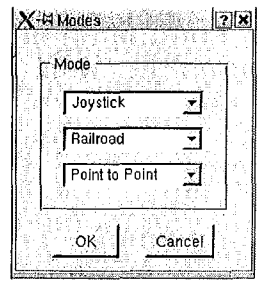


FIG. 5.6 – Préférences de l'utilisateur

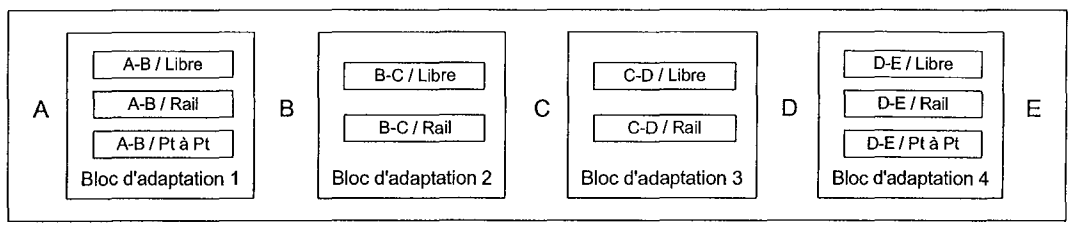


FIG. 5.7 – Blocs d'adaptation de l'application de télé-pilotage

elles dépendent de la perception subjective de l'opérateur pour lequel elles ont été établies. Une étude plus approfondie et de multiples tests seraient nécessaires pour déterminer des seuils significatifs pour la majorité des utilisateurs.

Les blocs d'adaptation de l'application sont présentés sur la figure 5.7. Ils correspondent aux étapes qui composent le trajet de A à E. Selon le bloc, toutes les classes de QdS ne sont pas systématiquement disponibles : par exemple, pour traverser un couloir où des personnes sont susceptibles de circuler il est préférable de pouvoir prendre la main sur le robot pour agir en cas d'imprévu (mode "libre" ou "rail" seulement).

A chaque séquence d'exécution correspondent des actions d'adaptation. Les actions

Classe de QdS	Mode de pilotage	$rtt_{max}(s)$
1	Libre	0,10
2	Assisté	0,25
3	Point à point	1,00

TAB. 5.1 – Correspondance (translation) classe de QoS - retard

"Haut" définissent les actions à mener lorsqu'une classe supérieure de QdS peut être satisfaite, les actions "Bas" définissent les actions à entreprendre lorsque la classe de QdS courante n'est pas respectée. Les actions d'adaptation implémentées sont :

- BEST_EFFORT : l'application ignore les changements de QdS, le mode de contrôle du robot reste identique jusqu'au point suivant.
- SWITCH : l'application bascule d'un mode de pilotage à un autre au cours d'un bloc.
- BLOCK : l'application suspend son exécution jusqu'à ce que la classe de QdS soit à nouveau disponible, le robot est immobilisé.

Les séquences CAP qui associent les classes de QdS, les séquences d'exécution et les actions d'adaptation sont présentées dans le tableau 5.2.

Séquence d'exécution	Classe de QdS	Action Haut	Action Bas
Aller de A à B	1	-	SWITCH
Aller de A à B	2	BEST_EFFORT	SWITCH
Aller de A à B	3	BEST_EFFORT	BLOCK
Aller de B à C	1	-	SWITCH
Aller de B à C	2	BEST_EFFORT	BLOCK
Aller de C à D	1	-	SWITCH
Aller de C à D	2	BEST_EFFORT	BLOCK
Aller de D à E	1	-	SWITCH
Aller de D à E	2	BEST_EFFORT	SWITCH
Aller de D à E	3	BEST_EFFORT	BLOCK

TAB. 5.2 – Séquences CAP

Remarques :

- Aucune action "Haut" n'est définie pour les séquences correspondant à la classe de QdS 1 car aucune classe de QdS n'est supérieure à celle-ci.
- L'utilisation de l'action BEST_EFFORT plutôt que SWITCH pour les actions "Haut" permet d'éviter les oscillations entre deux séquences différentes durant l'exécution d'un bloc lorsque le *rtt* varie autour d'une limite d'une classe de QdS (oscillations entre les classes 1 et 2 lorsque le rtt varie autour de $0,1s$ par exemple). De la sorte, le système « reste bloqué » dans la classe de QdS inférieure durant l'exécution du bloc, dès la première commutation vers la classe inférieure.

- Quand le délai aller-retour est supérieur à $1s$, aucune des classes de QdS spécifiées n'est disponible. Dans ce cas, l'application bascule dans la séquence associée à la classe la plus basse (la classe 2 ou 3 selon le bloc) si elle n'y était pas et l'action d'adaptation Bas est activée : Ici, le système se bloque. Si le délai aller-retour est à nouveau inférieur à $1s$ avant la fin du bloc, l'application reprend l'exécution de la séquence associée à la classe de QdS la plus basse.

Les blocs, séquences et actions d'adaptation sont modélisés par des réseaux de Petri dans l'annexe A page 131.

Le délai aller-retour est mesuré par le service de métrologie de QdS intégré.

5.2.4 Application de télé-asservissement

L'objectif est d'asservir la position x du robot mobile depuis le poste de commande (cf. figure 5.8). Le robot est commandé en vitesse. Il mesure sa position x grâce à ses capteurs (odomètres) et la retourne au poste de commande. Une commande en vitesse v est calculée sur le poste de commande puis elle est transmise au robot. Le correcteur est de type proportionnel k.

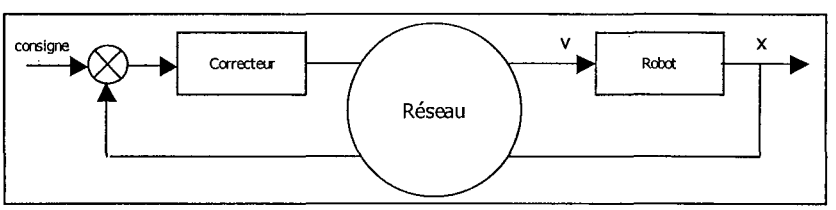


FIG. 5.8 – Boucle de « télé-asservissement »

L'utilisation du réseau pour les transmissions de la commande et de la mesure (position du robot) introduit un retard dans la boucle d'asservissement.

Remarque : ce type de commande se classe dans la famille de la "commande à boucle globale" [PLSS92], la boucle d'asservissement intègre le maître, l'esclave et les délais de transmission. Les méthodes habituelles de stabilisation adaptées à ce type de problème (prédicteur de Smith, etc.) prennent pour hypothèse la constance du retard [Lel00], ce qui n'est pas le cas ici.

L'impact du délai sur l'asservissement est étudié dans le paragraphe suivant.

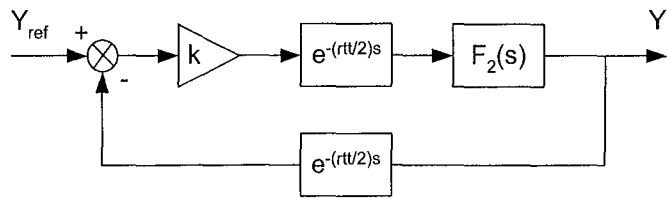


FIG. 5.9 – Boucle d'asservissement

5.2.4.1 Impact du retard sur l'asservissement

Dans la suite, le délai du réseau est considéré symétrique (délai identique à l'aller et au retour). Le délai aller-retour, c'est à dire le retard global de la boucle, est noté rtt (round-trip time).

Cette étude est réalisée en simulation avec Matlab. La fonction de transfert du robot a été préalablement identifiée à l'aide de la boîte d'identification CONTSID présentée dans [HGRY02]. On obtient la fonction suivante pour la réponse en vitesse :

$$F_1(s) = \frac{16,6117.e^{-70.10^{-3}s}}{s^2 + 4,0450s + 16,5611}$$

On ajoute un intégrateur pour obtenir la réponse en position :

$$F_2(s) = \frac{16,6117.e^{-70.10^{-3}s}}{s^3 + 4,0450s^2 + 16,5611s}$$

La boucle d'asservissement est représentée sur la figure 5.9.

La fonction de transfert du système en boucle fermée est :

$$FTBF(s) = \frac{16,6117.k.e^{-(70.10^{-3}+\frac{rtt}{2})s}}{s^3 + 4,0450s^2 + 16,5611s + 16,6117.k.e^{-(70.10^{-3}+rtt)s}}$$

La réponse indicielle du système est simulée pour différents retards qui provoquent des dépassements (cf. figure 5.10).

Le dépassement est très gênant dans les cas où le robot se déplace à proximité d'obstacles (mur par exemple), et peut provoquer des collisions.

Il convient alors de mettre en œuvre une stratégie pour supprimer les risques de dépassement du système.

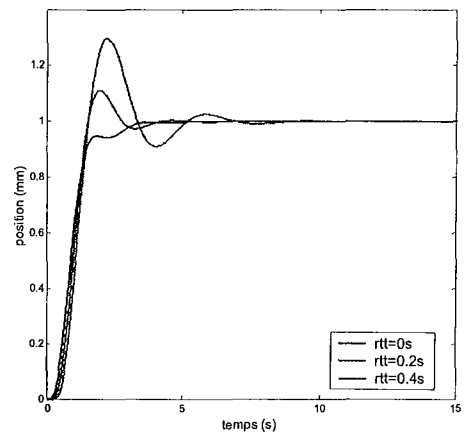


FIG. 5.10 – Réponse indicielle pour plusieurs valeurs de retard

5.2.4.2 Stratégie pour supprimer le dépassement

La figure 5.11 montre qu'à retards identiques, le dépassement du système dépend du gain k (résultat bien connu des automaticiens). La solution la plus simple pour supprimer le dépassement consiste alors à calculer la valeur du gain de la boucle d'asservissement pour le délai maximum.

Cette approche présente l'inconvénient majeur de ralentir le système : Par exemple, si l'on désire supprimer les dépassements en présence de retards inférieurs ou égaux à $1s$ (soit en choisissant un gain $k = 0,3$ par exemple), le temps de montée passe de $1,38s$ (avec un gain de 0.9) à $6,53s$ (cf. tableau 5.3).

k	$0,3$	$0,5$	$0,7$	$0,9$
t_m	$6,53$	$3,59$	$2,37$	$1,38$

TAB. 5.3 – Temps de montée en fonction du gain (valeurs obtenues pour rtt=0)

Se contenter de fixer une valeur de gain pour un retard maximum n'est donc pas satisfaisant. En effet, pour les situations dans lesquelles le retard est très inférieur à la valeur maximum, le système est inutilement lent.

Il serait alors intéressant de choisir la valeur du gain selon des plages définies de retards.

Le tableau 5.4 présente les retards maxima calculés pour un dépassement nul, pour différentes valeurs de k.

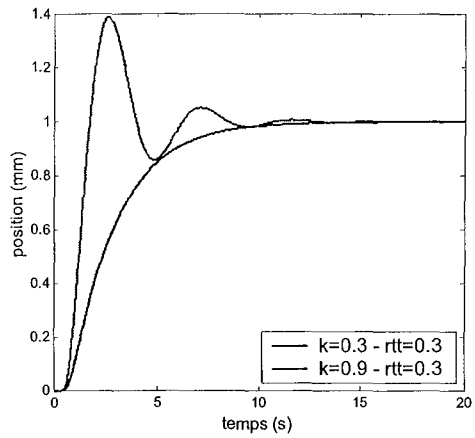


FIG. 5.11 – Réponses indicielles à retard identique et gains différents

Remarques :

- Ces valeurs sont très élevées et pour certaines très supérieures aux délais couramment observés aujourd'hui sur l'Internet. Ceci est dû à l'extrême lenteur du système étudié. Une étude similaire d'un système plus rapide (robot se déplaçant à plusieurs m/s par exemple) conduirait à obtenir des retards maxima beaucoup plus faibles et à l'échelle des délais observés sur l'Internet.
- Il est possible de définir des lois de commande beaucoup plus évoluées et performantes grâce à la théorie des systèmes à retard [Ric00]. Ici, notre objectif principal est d'étudier l'intérêt d'adapter l'application, la stratégie de commande mise en oeuvre a été volontairement choisie pour sa simplicité.

5.2.4.3 Mise en œuvre de l'architecture

L'application [ML03] est divisée en deux parties : une partie est située sur le poste de commande, elle est chargée de recevoir la mesure de la position du robot, de calculer la commande et de l'envoyer au PC embarqué. L'autre partie, localisée sur le PC embarqué, a pour rôle de récupérer la position du robot (interrogation de la carte contrôleur du robot via le port série), d'envoyer cette information au poste de commande, de recevoir la commande du poste de commande et de la faire suivre à la carte-contrôleur du robot (toujours via le port série). L'application ne dialogue pas directement avec les capteurs et

les actionneurs du robot mais avec sa carte-contrôleur qui est un système de commande fermé dont nous n'avons pas la maîtrise (cf. figure 5.12).

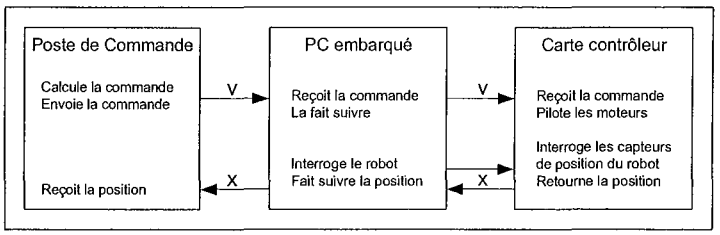


FIG. 5.12 – Structure de l'application

L'application est développée selon le modèle de l'architecture présentée précédemment. Pour cela, quatre classes de QdS sont définies pour la partie de l'application située sur le poste de commande. Chaque classe correspond à une configuration applicative, c'est à dire à une valeur de gain k, elle est destinée à être utilisée selon la valeur mesurée du retard (cf. tableau 5.5).

On définit un bloc d'adaptation. Ce bloc est répété périodiquement (cf. figure 5.13). Il est constitué de quatre séquences d'exécution. Chaque séquence consiste à calculer la commande durant la période du bloc, pour chaque classe de QdS. Ici, à une classe de QdS ne correspond qu'une séquence d'exécution. La période d'un bloc d'adaptation est de $2s$.

Trois actions d'adaptation différentes sont implémentées :

- BEST_EFFORT : l'application ignore les changements de QdS, la commande continue à être calculée avec le même gain.
- SWITCH : l'application commute d'une séquence à une autre en cours d'exécution. La valeur de k est modifiée en cours d'exécution du bloc.
- BLOCK : l'application suspend son exécution jusqu'à ce que la classe de QdS soit à nouveau disponible. Le robot est immobilisé.

Durant l'exécution d'un bloc, la classe de QdS la plus adaptée est recalculée à une période de $0,5s$.

Les séquences CAP sont présentées dans le tableau 5.6.

Remarques :

- De même que pour l'application de télécontrôle, aucune action "Haut" n'est définie pour la séquence correspondant à la classe de QdS 1 car aucune classe de QdS n'est

k	0,3	0,5	0,7	0,9
$rtt_{\max}$	1,10	0,50	0,30	0,10

TAB. 5.4 – Retards maxima (dépassement nul) en fonction de k

Classe de QdS	k	$rtt_{max}(s)$
1	0,9	0,1
2	0,7	0,3
3	0,5	0,5
4	0,3	1,1

TAB. 5.5 – Correspondance (translation) classe de QoS - retard

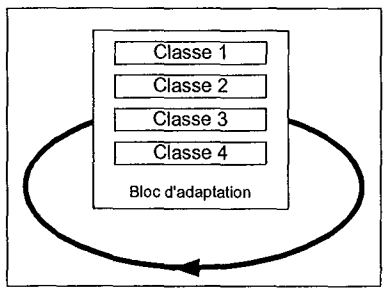


FIG. 5.13 – Bloc d'adaptation

Séquence d'exécution	Classe de QdS	Action Haut	Action Bas
Réguler	1	-	SWITCH
Réguler	2	BEST_EFFORT	SWITCH
Réguler	3	BEST_EFFORT	SWITCH
Réguler	4	BEST_EFFORT	BLOCK

TAB. 5.6 – Séquences CAP

supérieure à celle-ci. De plus, l'action BEST_EFFORT est aussi utilisée pour les actions "Haut" afin d'éviter les oscillations entre deux séquences différentes durant l'exécution d'un bloc lorsque le *rtt* varie autour d'une limite d'une classe de QdS (cf. paragraphe 5.2.3.2).

- Quand le délai aller-retour est supérieur à $1,1s$, aucune des classes de QdS spécifiées n'est disponible. Dans ce cas, l'application bascule dans la séquence associée à la classe la plus basse (la classe 4) si elle n'y était pas et l'action d'adaptation Bas est activée : Ici, le système se bloque. Si le délai aller-retour est à nouveau inférieur à $1,1s$ avant la fin du bloc, l'application reprend l'exécution de la séquence associée à la classe de QdS 4.

Le bloc, les séquences et les actions d'adaptation sont modélisés par des réseaux de Petri dans l'annexe B page 141.

5.2.4.4 Mesure du délai aller-retour (rtt)

Le service de métrologie de QdS de l'architecture n'est pas utilisé dans cette application : à ce jour, celui-ci n'implémente que des techniques de métrologie active. Ces techniques génèrent un trafic supplémentaire sur le réseau contrairement aux techniques passives basées sur l'analyse du trafic utile. Nous avons choisi de ne pas utiliser le service de métrologie car dans le cas présent une métrologie passive semble plus appropriée : le retard est mesuré en se fondant sur les dates d'envoi et de réception des paquets d'informations utiles (paquets véhiculant les commandes et les mesures). Les paquets émis par le PC de commande indiquant la commande sont estampillés lors de leur envoi. Dès la réception d'un de ces paquets sur le PC embarqué, ce dernier fait suivre la commande au robot, l'interroge sur sa position, retourne cette dernière ainsi que l'estampille temporelle précédente. Le PC de commande reçoit ces informations. Il calcule alors le *rtt* en soustrayant l'estampille temporelle contenue dans le paquet à l'heure à laquelle il a reçu ce paquet.

5.2.4.5 Résultats expérimentaux

Les résultats présentés sont ceux obtenus avec le système réel (cf. figure 5.3). Les retards sont générés par un émulateur de réseau [LML01].

La figure 5.14 montre les réponses indicielles obtenues à *rtt* identiques pour deux valeurs de gain différentes. On vérifie bien qu'à *rtt* identiques, le dépassement augmente

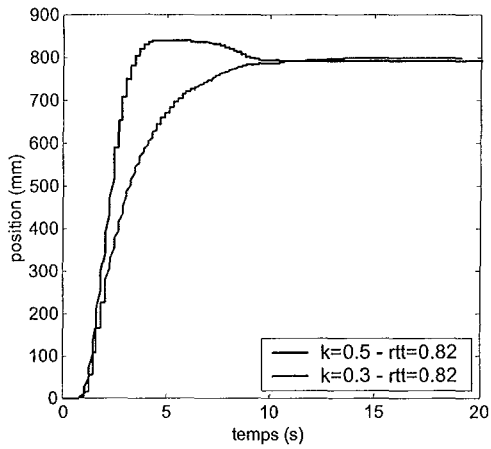


FIG. 5.14 – Réponses indicielles à retards identiques pour deux valeurs de gain différentes

avec le gain.

La figure 5.15 de la page 115 présente les résultats obtenus pour une expérience au cours de laquelle le *rtt* varie.

La consigne et la réponse du système en fonction du temps sont représentées sur la figure du haut. La figure du milieu indique la valeur du *rtt* mesurée durant l'expérience. La valeur du gain en fonction du temps est représentée sur la figure du bas.

On vérifie bien que la valeur du gain k change en fonction du *rtt* mesuré,

Le système suit la consigne et semble correctement identifié,

A $t = 54,5s$, le retard dépasse $1,1s$. Le système ne s'immobilise pas instantanément. C'est le seul moment pour lequel le système dépasse la consigne. Ceci s'explique par le temps de réaction du mécanisme d'adaptation, c'est à dire la période avec laquelle la classe de QdS la plus adaptée est calculée (ici $0,5s$).

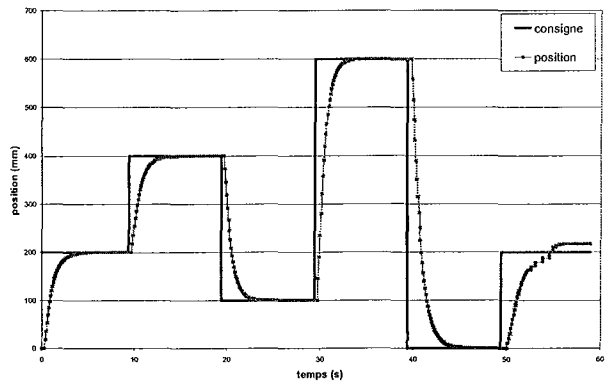
5.3 Conclusion

Ce chapitre a présenté l'implémentation du service de métrologie et de l'architecture à QdS intégrée sur une plateforme de télé-opération d'un robot mobile. Le service de métrologie intègre des capteurs de mesure du délai unidirectionnel, de la variation de délai, des pertes de paquets et du délai aller-retour. Les mesures sont possibles sur les réseaux IP et pour les protocoles de transport UDP et TCP. Une application de télé-pilotage du

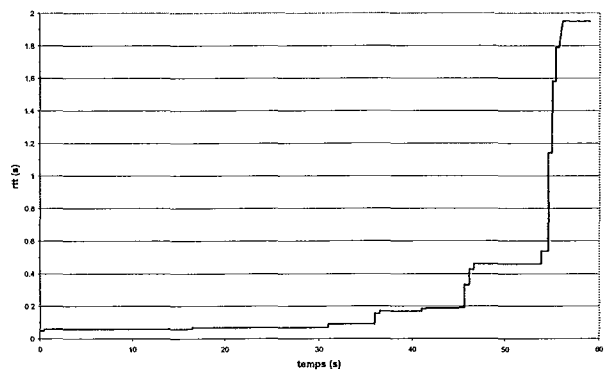
robot mobile et une application d'asservissement à distance de la position du robot ont été développées. Ces deux applications adaptent leur exécution au délai aller-retour mesuré entre le robot et son poste de commande.

La première application a été développée pour son intérêt démonstratif. Elle illustre simplement le principe d'adaptation des applications et montre qu'il est possible d'adapter une application à la QdS mesurée.

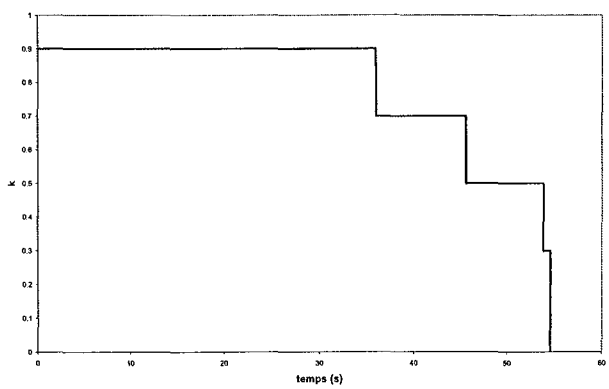
Les résultats expérimentaux obtenus avec la deuxième application montrent l'intérêt de notre approche : l'adaptation de la loi de commande à la QdS du réseau permet d'éviter les dépassements du système. Les risques de collision du robot avec son environnement sont alors limités. La stratégie utilisée d'adaptation de la loi de commande n'a pas la prétention d'être la meilleure. Elle a été choisie pour sa simplicité. Les possibilités offertes par l'architecture (notamment une métrologie plus complète que celle utilisée ici) pourraient permettre la mise en œuvre de stratégies plus élaborées et ainsi l'amélioration des performances de l'asservissement.



a. Consigne et réponse du système



b. Retard (rtt)



c. Gain

FIG. 5.15 – Résultats expérimentaux

Chapitre 6

Conclusion et perspectives

Durant ces dix dernières années, les applications mettant à contribution des réseaux de communication à commutation de paquets se sont complexifiées : les réseaux "grande distance" initialement conçus pour acheminer des trafics sans contraintes particulières (messagerie, transfert de fichiers, etc.) doivent aujourd'hui satisfaire les besoins d'applications critiques dont les trafics sont fortement contraints en terme de délai, de bande passante, etc. Le fonctionnement de ces applications est donc directement conditionné par la QdS fournie par le réseau de communication, qu'il est indispensable de prendre en considération. Pour cela, deux approches concourantes sont envisageables :

- La première repose sur la gestion des ressources, par réservation ou par gestion de priorités. La réservation dynamique ne résiste pas au facteur d'échelle et la gestion des priorités n'offre pas de garantie. L'essentiel des travaux proposés dans la littérature s'inscrivent dans cette approche.
- Partant de l'hypothèse qu'une maîtrise du système de communication n'est pas totale ou est impossible, la seconde permet de considérer que l'application peut, dans une certaine mesure, s'adapter à la variation de la QdS si elle connaît cette variation.

Les travaux menés dans le cadre de cette thèse se positionnent dans la seconde approche et apportent les contributions suivantes : une architecture de QdS pour l'adaptation, un service de métrologie de la QdS et deux applications, une de télé-pilotage d'un robot mobile et l'autre de télé-asservissement en position de ce robot.

- L'architecture proposée s'inspire d'un modèle de la littérature (modèle Prayer). Elle présente des améliorations majeures notamment en prenant en compte les préférences de l'utilisateur, en considérant un nombre moins limité de paramètres de QdS, en

simplifiant la spécification de la QdS par la présence d'un module de traduction de paramètres de QdS, et en intégrant un service de métrologie de la QdS disponible.

- Le service de métrologie de la QdS est architecturé de manière originale pour accueillir des "capteurs" ou opérateurs de mesure de toutes sortes de paramètres. Sa conception a été effectuée en utilisant le langage de modélisation UML. Il a été implémenté pour réaliser des mesures sur les réseaux IP pour les protocoles de transport UDP et TCP. A ce jour, il inclut des capteurs pour la mesure du délai unidirectionnel, de la variation de délai, des pertes de paquets et du délai aller-retour. De plus, une interface graphique optionnelle permet l'emploi du service dans un contexte différent de celui de l'architecture de QdS.
- L'application à la télé-opération d'un robot mobile permet de vérifier le bon fonctionnement de l'architecture sur deux exemples. Le premier est une application de télé-pilotage du robot qui a été développée pour son intérêt démonstratif. Elle illustre simplement le principe d'adaptation des applications et montre qu'il est possible d'accorder une application à la QdS mesurée. Le deuxième est une application d'asservissement à distance de la position du robot. Les expérimentations ont montré que l'architecture rend possible la mise en œuvre d'une stratégie pour s'adapter aux retards introduits par le réseau dans la boucle de commande et ainsi éviter par exemple les dépassements du système.

Dans la perspective de ces travaux, il serait intéressant :

- d'établir une classification des applications, ce qui permettrait d'associer à chaque classe un modèle de spécification de la QdS et des fonctions de traduction. Ceci contribuerait aussi à définir de façon plus précise une politique adaptée de surveillance de la QdS : paramètres pertinents à mesurer, période de mesure.
- d'étudier la finesse des techniques de métrologie employées, c'est-à-dire de déterminer les perturbations engendrées par le trafic de mesure, dans le cas des mesures actives sur le système mesuré (réseau et systèmes terminaux) et sur la qualité des résultats obtenus.
- de quantifier les incertitudes de mesures pour les différents paramètres et d'étudier notamment l'impact des performances des systèmes terminaux qui effectuent les mesures sur les valeurs mesurées.

Concernant l'application de télé-asservissement, il serait souhaitable dans un premier temps de compléter nos travaux par une étude de stabilité qui prenne en compte à la fois

les commutations de la loi de commande et la variation du retard. Dans un second temps, une loi de commande plus évoluée et basée sur une métrologie davantage élaborée pourrait être conçue en collaboration avec des spécialistes de la commande des systèmes à retard et elle pourrait être mise en œuvre sur l'architecture proposée.

Bibliographie

[Abd02] S. Abdellatif. *Contribution à la modélisation et à l'analyse de la qualité de service dans les réseaux à commutation de paquets.* Thèse, Université Paul Sabatier de Toulouse, Janvier 2002.

[AKZ99a] G. Almes, S. Kalidindi, et M. Zekauskas. A One-Way Delay Metric for IPPM, RFC 2679, Septembre 1999.

[AKZ99b] G. Almes, S. Kalidindi, et M. Zekauskas. A One-Way Packet Loss Metric for IPPM, RFC 2680, Septembre 1999.

[AKZ99c] G. Almes, S. Kalidindi, et M. Zekauskas. A Round-trip Delay Metric for IPPM, RFC 2681, Septembre 1999.

[ASC+02] T. Anjali, C. Scoglio, L. C. Chen, I. F. Akyildiz, et G. Uhl. Abest : An available bandwidth estimator within an autonomous system. Dans *Proc. IEEE Globecom*, 2002.

[ASU+] T. Anjali, C. Scoglio, G. Uhl, A. Sciuto, et J. A. Smith. Available Bandwidth Measurement in IP Networks, Internet Draft, Expiration date April 2003.

[Bao98] Y. Bao. *Quality of Service control for real-time multimedia applications in packet-switched networks.* Thèse, University of Delaware, Mai 1998.

[BCA+00] G. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. Costa, H. Duran, N. Parlavantzas, et K. Saikoski. A principled approach to supporting adaptation in distributed mobile environments. Dans *Proc. 5th International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE-2000)*, Limerick, Irlande, Juin 2000.

[BDS95] I. Busse, B. Deffner, et H. Schulzrinne. Dynamic QoS Control of Multimedia Applications based on RTP. Dans *Proc. First International Workshop on*

High Speed Networks and Open Distributed Platforms, St. Petersburg, Russie, Juin 1995.

[Bey95] P. Beyssac. BING, a Bandwidth measurement tool based on pING, 1995.

[BG97] V. Bharghavan et V. Gupta. A framework for application adaptation in mobile computing environments. Dans *Proc. IEEE Compsac'97*, Novembre 1997.

[BG98] J. Bollinger et T. Gross. A framework-based approach to the development of network-aware applications. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 24(5) :376–390, Mai 1998.

[BK99] S. N. Bhatti et G. Knight. Enabling QoS adaptation decisions for Internet applications. *Computer Networks*, 31 :669–692, 1999.

[BLLS02] M. Benaissa, V. Lecuire, F. Lepage, et A. Schaff. Un algorithme d'ajustement du délai de playout pour les flux audio continus dans les réseaux mobiles ad hoc. Dans *Proc. Colloque Francophone sur l'Ingéniérie des Protocoles, CFIP 2002*, Montréal, Canada, Mai 2002.

[BNBH98] S. Brandt, G. Nutt, T. Berk, et M. Humphrey. Soft real-time applications execution with dynamic quality of service assurance. Dans *Proc. International Workshop on Quality of Service*, Napa, USA, 1998.

[BS98] M. S. Borella et D. Swider. Internet Packet Loss : Measurement and Implications for End to End QoS. Dans *Proc. ICPP Workshops on Architectural and OS Support for Multimedia Applications/Flexible Communication Systems/Wireless Networks and Mobile Computing*, 1998.

[CAC94] T. Connolly, P. Amer, et P. Conrad. An Extension to TCP : Partial Order Service, RFC 1693, Novembre 1994.

[CAI] CAIDA. Cooperative Association for Internet Data Analysis, http ://www.caida.org.

[Cam96] A. T. Campbell. *A Quality of Service Architecture.* Thèse, Computing Department, Lancaster University, 1996.

[CC96] R. Carter et M. Crovella. Measuring bottleneck link speed in packet-switched networks. Technical Report 1996-006, Boston University, Mars 1996.

[CK99] F. Chang et V. Karamcheti. Automatic configuration and run-time adaptation of distributed applications. Technical Report TR1999-793, Octobre 1999.

[CL99] F. Cheong et R. Lai. QoS Specification and mapping for distributed multimedia systems : a survey of issues. *The Journal of Systems and Software*, 45 :127–139, 1999.

[CT90] D. D. Clark et D. L. Tennenhouse. Architectural considerations for a new generation of protocols. Dans *Proc. ACM SIGCOMM*, 1990.

[DC02] C. Demichelis et P. Chimento. IP Packet Delay Variation Metric for IPPM, RFC 3393, Novembre 2002.

[Dem02] I. Demeure. Une contribution à la conception et à la mise en ouvre d'applications sous contraintes de QoS temporelles, réparties, adaptables. Université des Sciences et Technologies de Lille, Décembre 2002.

[Des] DesignCPN. http ://www.daimi.au.dk/designcpn/.

[DHT95] C. Diot, C. Huitema, et T. Turletti. Multimedia applications should be adaptive. Dans *Proc. HPCS'95, Mystic (CN)*, 1995.

[Dif] DiffServ. Groupe de travail Differentiated Services de l'IETF, http ://www.ietf.org/html.charters/diffserv-charter.html.

[Dio95] C. Diot. Adaptative Applications and QoS Guaranties. Dans *Proc. IEEE Multimedia Networking*, Aizu, Japon, Septembre 1995.

[Dov] C. Dovrolis. Pathrate, http ://www.pathrate.org.

[Dow99] A. B. Downey. Using Pathchar to estimate Internet link characteristics. Dans *Proc. ACM SIGCOMM'99*, pages 222–223, 1999.

[DRM01] C. Dovrolis, P. Ramanathan, et D. Moore. What do packet dispersion techniques measure ? Dans *Proc. INFOCOM*, pages 905–914, Avril 2001.

[DS97] C. Diot et A. Seneviratne. Quality of service in heterogeneous distributed systems. Dans *Proc. 30th Hawaï International Conference on System Sciences*, Hawaï, USA, Janvier 1997.

[FDBC99] A. Friday, N. Davies, GS. Blair, et KWJ. Cheverst. Developing adaptative applications : the MOST experience. *Integrated Computer-Aided Engineering*, 6(2) :143–157, 1999.

[FG99] M. Fry et A. Ghosh. Application level active networking. *Computer networks*, 31 :655–667, 1999.

[FRS00] I. Foster, A. Roy, et V. Sander. A quality of service architecture that combines resource reservation and application adaptation. Dans *Proc. 8th International Workshop on Quality of Service*, Mai 2000.

[GML03] E. Gnaedinger, F. Michaut, et F. Lepage. Implémentation d'une architecture de QdS spécifique permettant l'adaptation d'une application de contrôle à distance d'un robot mobile. Dans *Proc. Quatrième Conférence Internationale sur l'Automatisation Industrielle*, Montréal, Canada, Juin 2003.

[GP94] R. Gopalakrishnan et G. Parulkar. Efficient quality of service support in multimedia computer operating systems. Technical Report WUCS-94-26, Washington University, Novembre 1994.

[HGRY02] E. Huselstein, H. Garnier, A. Richard, et P. Young. La boîte à outils CONTSID d'identification de modèles à temps continu - Extensions récentes. Dans *Proc. Conférence Internationale Francophone d'Automatique*, Nantes, France, Juillet 2002.

[HvB98] A. Hafid et G. von Bochmann. Quality-of-Service adaptation in distributed multimedia applications. *Multimedia Systems*, 6(5) :299–315, 1998.

[Int] IntServ. Groupe de travail Integrated Services de l'IETF, http ://www.ietf.org/html.charters/intserv-charter.html.

[Ipe] Iperf. http ://dast.nlanr.net/projects/iperf.

[Jac] V. Jacobson. traceroute, ftp ://ftp.ee.lbl.gov/traceroute.tar.gz.

[Jac88] V. Jacobson. Congestion avoidance and control. Dans *Proc. ACM SIGCOMM'88*, pages 314–329, Août 1988.

[Jac93] V. Jacobson. vat x11 based audio conferencing tool, unix manual page, Février 1993.

[Jac97] V. Jacobson. Pathchar, a tool to infer characteristics of Internet paths, Avril 1997.

[JB99] G. Juanole et I. Blum. Influence de fonctions de base (communication-ordonnancement) des systèmes distribués temps réel sur les performances d'applications de contrôle-commande. Dans *Proc. CFIP*, 1999.

[JD02a] M. Jain et C. Dovrolis. End-to-End Available BW Measurement Methodology, Dynamics, and Relation with TCP Throughput. Dans *Proc. ACM SIGCOMM*, 2002.

[JD02b] M. Jain et C. Dovrolis. Pathload : a measurement tool for available bandwidth estimation. Dans *Proc. PAM'02*, 2002.

[Jia99] W. Jiang. Detecting and measuring asymmetric links in an IP network. Technical Report CUCS009-99, Janvier 1999.

[JRR97] M. Jones, D. Rosu, et M. Rosu. CPU Reservations and Time Constraints : Efficient, predictable Scheduling of Independent Activities. Dans *Proc. 16th ACM Syposium on operating Systems Principles*, Saint-Malo, France, Octobre 1997.

[JS99] W. Jiang et H. Schulzrinne. QoS Measurement of Internet Real-Time Multimedia Services. Technical Report CUCS015-99, Columbia University, New York, Décembre 1999.

[JYCA01] G. Jin, G. Yang, B. Crowley, et D. Agarwal. Network Characterisation Service (NCS). Dans *Proc. 10th IEEE Symposium on High Performance Distributed Computing*, Août 2001.

[KHP99] P. Keleher, J. K. Hollingsworth, et Dejan Perkovic. Exposing application alternatives. Dans *Proc. International Conference on Distributed Computing Systems*, pages 384–392, 1999.

[KLM97] S. Khan, K.F. Li, et E.G. Manning. Padma : An architecture for adaptive multimedia systems. Dans *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 1997.

[KR02] R. Koodli et R. Ravikanth. One-way Loss Pattern Sample Metrics, RFC 3357, Août 2002.

[LB00] K. Lai et M. Baker. Measuring link bandwidths using a deterministic model of packet delay. Dans *Proc. ACM SIGCOMM*, Stockholme, Suède, 2000.

[LB01] K. Lai et M. Baker. Nettimer : A tool for measuring bottleneck link bandwidth. Dans *Proc. USENIX Symposium on Internet Technologies and Systems*, San Francisco, USA, Mars 2001.

[LD86] I.D. Landau et L. Dugard. *Commande adaptative, aspects pratiques et théoriques.* Editions Masson, 1986.

[Lel00] A. Lelevé. *Contribution à la téléopération de robots en présence de délais de transmission variables.* Thèse, Université de Montpellier II, Décembre 2000.

[LL00] V. Lecuire et F. Lepage. Contrôle déterministe de la fiabilité par un marquage direct sur les paquets. *RERIR / EJNDP*, 9, Mai 2000.

[LML01] V. Lecuire, T. Maurey, et F. Lepage. Un émulateur de l'Internet réalisé sous RTLinux. Dans *Proc. 9th RTS'2001*, Paris, France, Mars 2001.

[LWD96] I-Shen Lin, F. Wallner, et R. Dillmann. Interactive control and environment modeling for a mobile robot based on multisensor perceptions. *Robotics and Autonomous Systems*, 18, 1996.

[LWNL98] B. Li, D. Wu, K. Nahrstedt, et J.W.S. Liu. End-to-End QoS Support for Adaptative Applications Over the Internet. Dans *Proc. Internet Routing and QoS*, Boston, USA, 1998.

[LY97] K. Lakshman et R. Yavatkar. Integrated CPU and Network I/O QoS Management in an End-System. *Computer Communications Journal, Special Issue on Quality of Service in Distributed Systems*, 21, Avril 1997.

[Mah99] B. Mah. Pchar, http ://www.employees.org/ bmah/Talks/pchar-NGI-99-Slides.pdf, 1999.

[Mil91] D. L. Mills. Internet Time Synchronisation : the Network Time Protocol. *IEEE Trans. Communications*, 39(10) :1482–1493, Octobre 1991.

[MK98] P. Moghé et A. Kalavade. Terminal QoS of Adaptive Applications. *Bell Labs Technical Journal*, 1998.

[ML02a] F. Michaut et F. Lepage. A QoS Architecture for Application Execution Adaptation. Dans *Proc. SNPD'02 ACIS 3rd International Conference on Software Engineering Artificial Intelligence, Networking and Parallel/Distributed Computing*, Madrid, Espagne, Juin 2002.

[ML02b] F. Michaut et F. Lepage. A Tool to Monitor the Network Quality of Service. Dans *Proc. NET-CON'2002, IFIP-IEEE Conference on Network Control and Engineering*, Paris, France, Octobre 2002.

[ML02c] F. Michaut et F. Lepage. An Architecture for Application Adaptation based on QoS Monitoring. Dans *Proc. SMC'02, IEEE International Conference on Systems, Man and Cybernetics*, Hammamet, Tunisie, Octobre 2002.

[ML02d] F. Michaut et F. Lepage. Networks Quality of Service Monitoring. Dans *Proc. IAR Annual Meeting*, Grenoble, France, Novembre 2002.

[ML03] F. Michaut et F. Lepage. Un cadre pour la prise en compte du retard dans la téléoperation d'un robot mobile. Dans *Proc. Journées Doctorales d'Automatique 2003*, Valenciennes, France, Juin 2003.

[Mul97] P.A. Muller. *Modélisation objet avec UML.* Editions Eyrolles, 1997.

[Nah95] K. Nahrstedt. *An architecture for End-to-End Quality of Service Provision and its experimental validation.* Thèse, University of Pennsylvania, 1995.

[Net] Netsizer. http ://www.netsizer.com.

[NIM] NIMI. National Internet Measurement Infrastructure, http ://www.ncne.nlanr.net/nimi.

[NL97] J. Nieh et M. Lam. The design, implementation and evaluation of SMART : A scheduler for multimedia applications. Dans *Proc. 16th Symp. Operating Systems Principles*, pages 184–197, Octobre 1997.

[NS95a] K. Nahrstedt et J. M. Smith. The QoS broker. *IEEE Multimedia*, 2(1) :53–67, 1995.

[NS95b] K. Nahrstedt et R. Steinmetz. Resource management in networked multimedia systems. *IEEE Computer*, 29(5) :52–63, Mai 1995.

[OF97] R. Oboe et P. Fiorini. Issues on Internet-based teleoperation. Dans *Proc. SYROCCO'97*, 1997.

[Ogo01] P. Ogor. *Une architecture générique pour la supervision sûre à distance de machines de production avec Internet.* Thèse, Université de Bretagne Occidentale, Décembre 2001.

[OMRW98] M. Ott, G. Michelitsch, D. Reininger, et G. Welling. An architecture for adaptive QoS and its application to multimedia systems design. *Computer Communications*, 21(4) :334–349, Avril 1998.

[OR] T. Oetiker et D. Rand. Multi Router Traffic Grapher, http ://www.people.ee.ethz.ch/ oetiker/webtools/mrtg.

[Owe01] P. Owezarski. Que nous dit la métrologie sur le futur d'Internet ? Dans *Proc. JRES 2001*, Lyon, France, Décembre 2001.

[PAM00] V. Paxson, A. Adams, et M. Mathis. Experiences with nimi. Dans *Proc. Passive and Active Measurements (PAM)*, 2000.

[PAMM98] V. Paxson, G. Almes, J. Mahdavi, et M. Mathis. Framework for IP Performance Metrics, RFC 2330, Mai 1998.

[Pax97] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics.* Thèse, Computer Science Division, University of California, Berkeley, and Information and Computing Sciences Division Lawrence Berkeley National Laboratory, University of California, Avril 1997.

[Pax98] V. Paxson. On calibrating measurements of packet transit times. Dans *Proc. SIGMETRIC'98*, 1998.

[PLSS92] R.P. Paul, T. Lindsay, C. Sayers, et M. Stein. Time-delay insensitive virtuel-force reflecting teleoperation. Dans *Proc. Artificial Intelligence, Robotics and Automation in Space*, Toulouse, France, Septembre 1992.

[PV02] A. Pasztor et D. Veitch. Active probing using packet quartets. Dans *Proc. Internet Measurement Workshop*, Marseille, France, Novembre 2002.

[RC00] L. M. Rojas-Cardenas. *Architecture de transport à ordre et fiabilité partiels pour les applications multimédias adaptatives à temps contraint.* Thèse, LAAS CNRS, 2000.

[RGM02] V. Raisanen, G. Grotefeld, et A. Morton. Network performance measurement for periodic streams, RFC 3432, Novembre 2002.

[Ric00] J-P. Richard. Time Delay Systems : An overview of some recent advances and open problems. Dans *Proc. 2nd IFAC Workshop on Linear Time Delay Systems*, Ancona, Italie, Septembre 2000.

[RIP] RIPE. Réseaux IP Européens, http ://www.ripe.net.

[Rés01] Réseau National de Recherche en Télécommunications. METROPOLIS, METROlogie Pour l'Internet et ses services, http ://www.telecom.gouv.fr/rnrt/, http ://www-rp.lip6.fr/metrologie, 2001.

[Sar01] S. Saroiu. Sprobe : A fast tool for measuring bottleneck bandwidth in uncooperative environments, http ://sprobe.cs.washington.edu, 2001.

[Sav99] S. Savage. Sting : A TCP-based Network Measurement Tool. Dans *Proc. USENIX Symposium on Internet Technologies and Systems*, pages 71–79, Boulder, USA, Octobre 1999.

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederick, et V. Jacobson. RTP : A Transport Protocol for Real-Time Applications, Request For Comments RFC 1889, Janvier 1996.

[Sis97] D. Sisalem. End-to-end quality of service control using adaptative applications. Dans *Proc. IFIP 5th International Workshop on QoS (IWQoS'97)*, New York, USA, Mai 1997.

[Sis98] D. Sisalem. Fairness of adaptative multimedia applications. Dans *Proc. International Conference on Communications (ICC'98)*, Atlanta, USA, Juin 1998.

[SKSZ99] A. Schill, S. Kümmel, T. Springer, et T. Ziegert. Two approaches for an adaptive multimedia transfer service for mobile environments. *Computer & Graphics*, 23 :849–856, 1999.

[SLA+99] R. De Silva, B. Landfeldt, S. Ardon, A. Seneviratne, et C. Diot. Managing application level quality of service throught tomten. *Computer Networks*, 31 :727–739, 1999.

[Sou03] A. Soudani. *Integration des fonctionnalités multimédias dans les systèmes répartis temps réels : Application au système de communication industrielle.* Thèse, Université du Centre, Monastir et Université Henri Poincaré, Nancy, Février 2003.

[Sri02] D. Srinivasagupta. *Studies in Process Modeling, Design, Monitoring, and Control, with Applications to Polymer Composites Manufacturing.* Thèse, Washington University, Saint Louis, MO, USA, Décembre 2002.

[Sur] Surveyor. http ://www.advanced.org/surveyor.

[Tcp] Tcpdump and Libpcap. http ://www.tcpdump.org.

[VG96] A. Vega-Garcia. *Mécanisme de contrôle pour la transmission de l'audio sur l'Internet.* Thèse, Université de Nice-Sophia Antipolis, INRIA, Octobre 1996.

[WS99] X. Wang et H. Schulzrinne. Comparison of adaptive internet multimedia applications. *Institute of Electronics, Information and Communication Engineers (IEICE) Transactions on Communications*, E82-B :806–818, Juin 1999.

[Wu97] Q. Wu. *Contribution à l'intégration de communications multimédias dans un environnement MMS.* Thèse, Université Henri Poincaré, Nancy, Juin 1997.

Annexe A

Modélisation de l'application de télé-pilotage

L'objectif de cette annexe est de définir le fonctionnement de l'application de télé-pilotage présentée dans le chapitre 5 en utilisant le formalisme des Réseaux de Petri présenté dans le chapitre 4 (page 67).

A.1 Rappels

L'application intègre trois classes de QdS (cf. tableau A.1) et quatre blocs d'adaptation (cf. figure A.1).

Nous pouvons remarquer que les blocs 1 et 4 (AB et DE) ont une structure identique, ils sont composés de trois séquences d'exécution associées aux classes de QdS 1, 2 et 3. Ces blocs seront modélisés par les mêmes réseaux de Petri. Il en est de même pour les blocs 2 et 3 (BC et CD) qui sont composés de deux séquences d'exécution associées aux classes de QdS 1 et 2.

Les séquences CAP qui associent les classes de QdS, les séquences d'exécution et les

Classe de QdS	Mode de pilotage	$rtt_{max}(s)$
1	Libre	0,10
2	Assisté	0,25
3	Point à point	1,00

TAB. A.1 – Classes de QoS de l'application de télé-pilotage

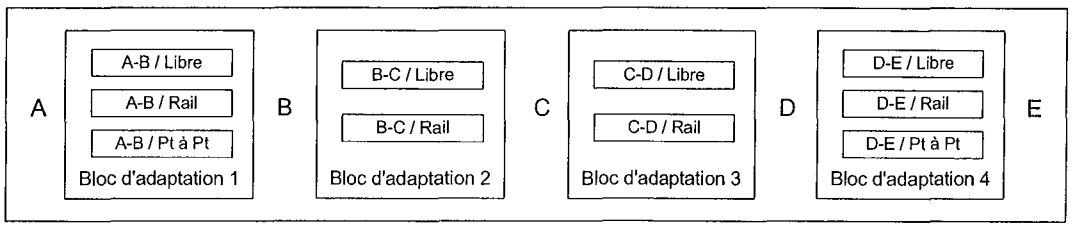


FIG. A.1 – Blocs d'adaptation de l'application de télé-pilotage

actions d'adaptation sont présentées dans le tableau A.2.

Séquence d'exécution	Classe de QdS	Action Haut	Action Bas
Aller de A à B	1	-	SWITCH
Aller de A à B	2	BEST_EFFORT	SWITCH
Aller de A à B	3	BEST_EFFORT	BLOCK
Aller de B à C	1	-	SWITCH
Aller de B à C	2	BEST_EFFORT	BLOCK
Aller de C à D	1	-	SWITCH
Aller de C à D	2	BEST_EFFORT	BLOCK
Aller de D à E	1	-	SWITCH
Aller de D à E	2	BEST_EFFORT	SWITCH
Aller de D à E	3	BEST_EFFORT	BLOCK

TAB. A.2 – Séquences CAP

Remarque : Quand le délai aller-retour est supérieur à $1s$, aucune des classes de QdS spécifiées n'est disponible. Dans ce cas, l'application bascule dans la séquence associée à la classe la plus basse (la classe 2 ou 3 selon le bloc) si elle n'y était pas et l'action d'adaptation Bas est activée : Ici, le système se bloque. Si le délai aller-retour est à nouveau inférieur à $1s$ avant la fin du bloc, l'application reprend l'exécution de la séquence associée à la classe de QdS la plus basse.

A.2 Modélisation

Pour des raisons de place, nous ne détaillerons dans la suite que les réseaux de Petri associés aux blocs 2 et 3. Le modèle complet correspondant aux blocs 1 et 4 est obtenu en

utilisant la même méthodologie.

A.2.1 Blocs 2 et 3 (BC et CD)

Les réseaux de Petri présentés dans cette partie sont valables pour les blocs 2 et 3.

Les classes de QdS sont modélisées par trois couleurs de jeton : les couleurs A, B et C associées respectivement aux classes 1, 2 et 3. Pour prendre en compte les cas dans lesquels aucune des classes de QdS n'est disponible, nous définissons la couleur supplémentaire D. L'exécution de chaque séquence correspond à une place (respectivement les places "Seq_BC_1" et "Seq_BC_2").

La couleur du jeton présent dans la place "Classe" indique la classe de QdS courante.

La place "Nouvelle Classe" symbolise la notification d'un changement de classe de QdS par la couche de gestion : dans ce cas, un jeton de couleur correspondante à la nouvelle classe est ajouté dans cette place. Par définition, lorsqu'un jeton de couleur x est présent dans la place "Classe", aucun jeton de couleur x ne peut être introduit dans la place "Nouvelle classe".

A.2.1.1 Démarrage et fin de séquence

Au démarrage de l'application, un jeton est présent dans la place "Etat initial" et la classe de QdS est indiquée par la présence d'un jeton de couleur correspondante (A, B, C ou D) dans la place "Classe". Selon la couleur de ce dernier, une seule des transitions "Exécuter seq_BC_1", "Exécuter seq_BC_2" et "Se bloquer" est franchissable (cf. figure A.2). Suite au tir de celle-ci, un jeton est mis dans la place correspondante ("Seq_BC_1", "Seq_BC_2" ou "Attente") :

- Si la classe de QdS est A, l'application doit exécuter la séquence BC_1.
- Si la classe de QdS est B, l'application doit exécuter la séquence BC_2.
- Si la classe de QdS est C ou D, l'application doit se bloquer jusqu'à ce que la classe de QdS soit A ou B (ce qui entraînera l'exécution de la séquence BC_2).

Lorsque l'application exécute une séquence et que cette dernière se termine, un jeton est introduit dans la place "Fin seq_BC_N" (avec N le numéro de la séquence associée), et la transition "Terminer seq_BC_N" est tirée. Le système se trouve alors dans l'état final, l'application a terminé l'exécution du bloc.

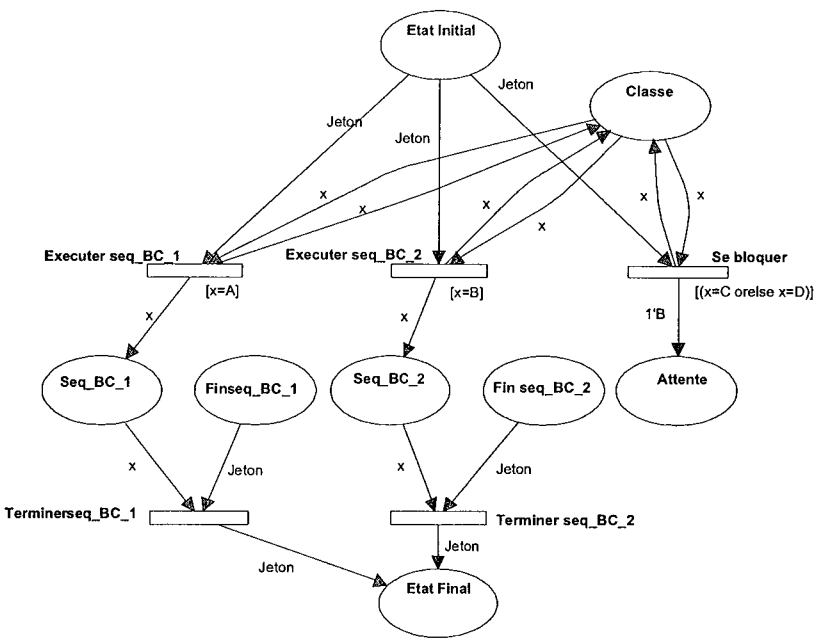


FIG. A.2 – Choix et fin des séquences d'exécution

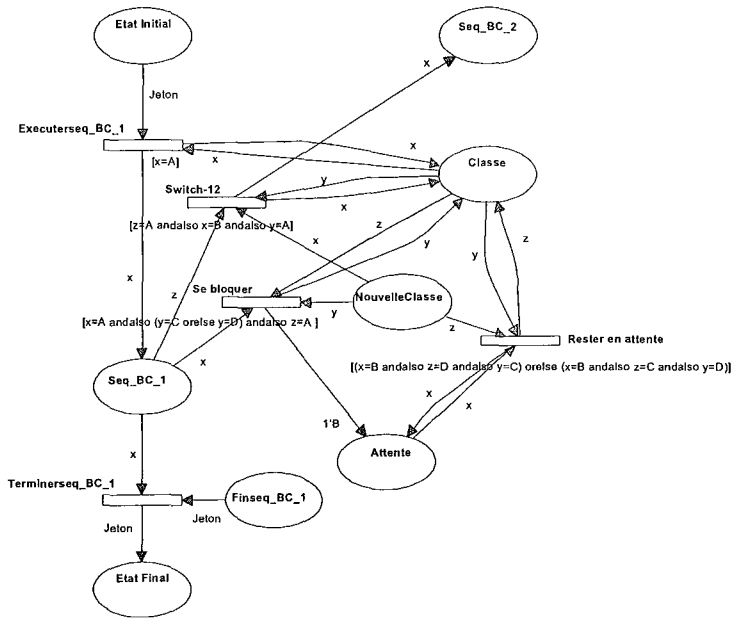


FIG. A.3 – Séquence d'exécution BC_1 et action d'adaptation SWITCH

A.2.1.2 Séquence BC_1

La figure A.3 représente l'action d'adaptation Bas associée à la séquence d'exécution BC_1. La primitive choisie est "SWITCH". Aucune action d'adaptation Haut n'est définie car la séquence BC_1 est associée à la classe de QdS A et il n'existe pas de classe de QdS supérieure à cette dernière.

Suite à la phase d'initialisation, nous supposons que l'exécution de la séquence BC_1 a été lancée. Dans ce cas, un jeton A est présent dans la place "Classe" ainsi que dans la place "Seq_BC_1".

Si la nouvelle classe est B, l'application enclenche la primitive SWITCH correspondant à l'action d'adaptation Bas. Dans ce cas, la transition "Switch-12" est franchie, un jeton est ajouté dans la place "Seq_BC_2", le jeton présent dans la place "Seq_BC_1" est supprimé, et le jeton de la place "Classe" est remplacé par un jeton dont la couleur est B (mise à jour de la classe).

Si la nouvelle classe est C ou D, l'application doit se bloquer jusqu'à ce que la classe de QdS soit A ou B ce qui entraînera l'exécution de la séquence BC_2. Dans ce cas,

la transition "Se bloquer" est franchie, un jeton de couleur B est ajouté dans la place "Attente", le jeton présent dans la place "Seq_BC_1" est supprimé, et le jeton de la place "Classe" est remplacé par un jeton dont la couleur est C ou D selon le cas (mise à jour de la classe).

A.2.1.3 Séquence BC_2

La figure A.4 illustre les actions d'adaptation associées à la séquence d'exécution BC_2 : il s'agit de la primitive BEST_EFFORT pour l'action Haut et de la primitive BLOCK pour l'action Bas.

Suite à la phase d'initialisation, nous supposons que l'exécution de la séquence BC_2 a été lancée. Dans ce cas, un jeton B est présent dans la place "Classe" ainsi que dans la place "Seq_BC_2".

Si la nouvelle classe de QdS est A (jeton A dans la place "Nouvelle classe"), la classe est mise à jour (couleur du jeton dans la place "Classe") et l'application poursuit l'exécution de la séquence BC_2. Si la nouvelle classe est C ou D, l'application se bloque (place "Attente").

L'application ne peut quitter l'attente que lorsque la classe de QdS est A ou B. Lorsque c'est le cas, l'application reprend l'exécution de la séquence BC_2.

Le fonctionnement global du mécanisme d'adaptation est modélisé par superposition des réseaux de Petri précédents (cf. figure A.5 page 138). Pour des raisons de visibilité, les conditions de franchissement ne sont pas indiquées sur cette figure (elles sont identiques à celles des figures précédentes).

A.2.2 Blocs 1 et 4 (AB et DE)

Le réseau de Petri correspondant aux blocs 1 et 4 est présenté sur la figure A.6 page 139. Les places "Seq_AB_1", "Seq_AB_2" et "Seq_AB_3" correspondent aux séquences d'exécution associées respectivement aux classes de QdS 1, 2 et 3.

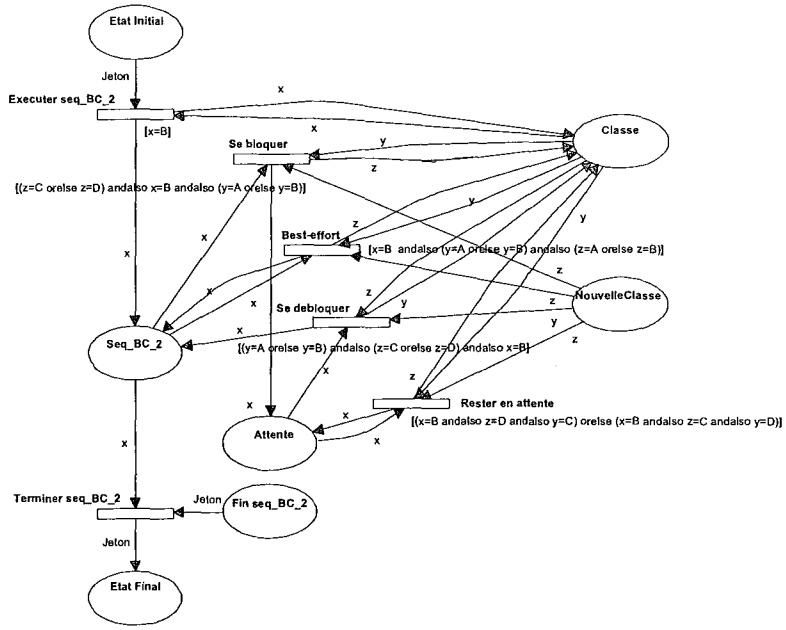


FIG. A.4 – Séquence d'exécution BC_2 et actions d'adaptation BEST_EFFORT et BLOCK

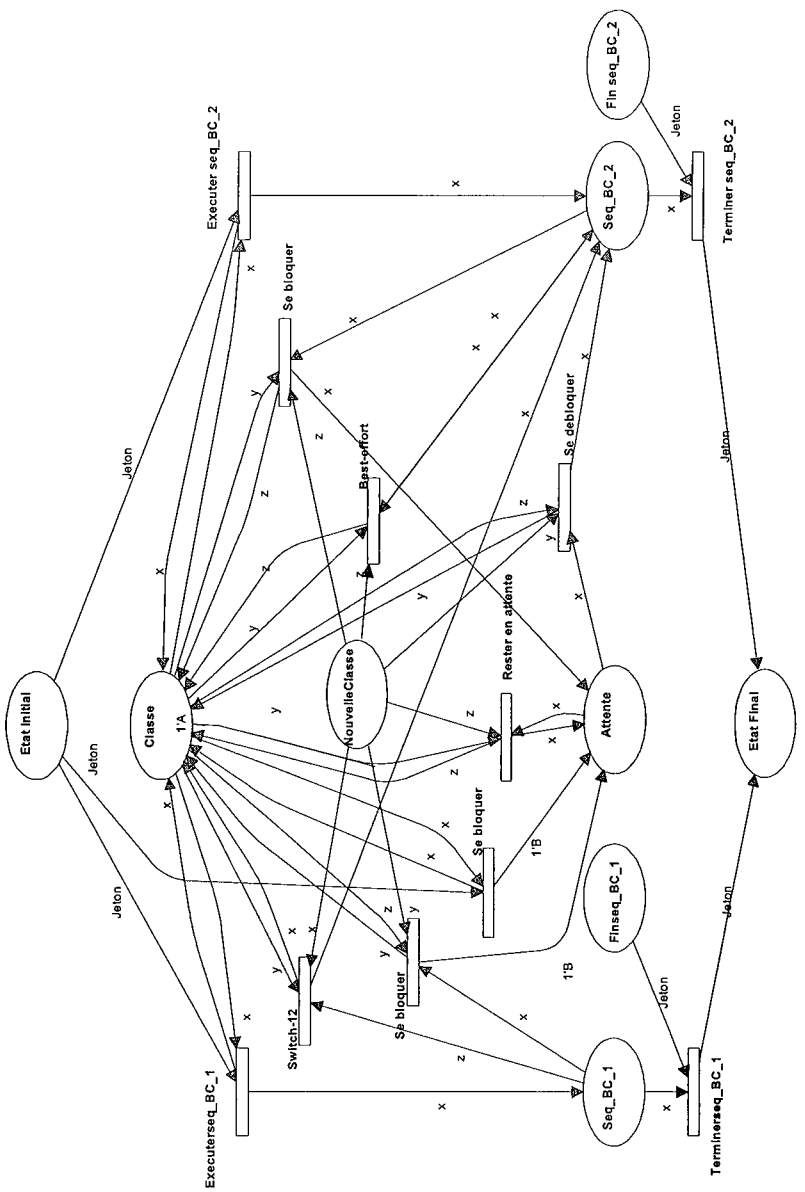


FIG. A.5 – Réseau de Petri complet des blocs 2 et 3

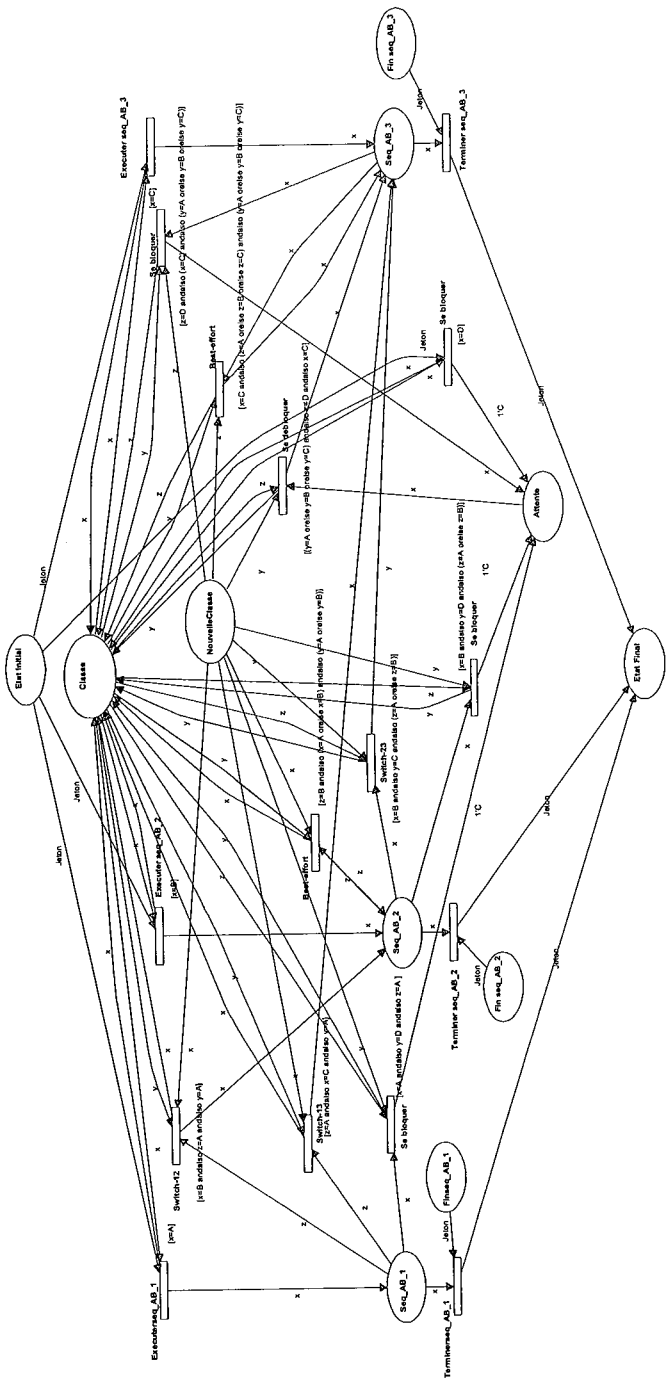


FIG. A.6 – Réseau de Petri complet des blocs 1 et 4

Annexe B

Modélisation de l'application de télé-asservissement

L'objectif de cette annexe est de définir le fonctionnement de l'application de télé-asservissement présentée dans le chapitre 5 en utilisant le formalisme des Réseaux de Petri présenté dans le chapitre 4 (page 67).

B.1 Rappels

L'application intègre quatre classes de QdS (cf. tableau B.1) et un bloc d'adaptation (cf. figure B.1). Ce bloc est répété périodiquement. Il est constitué de quatre séquences d'exécution qui correspondent aux quatre classes de QdS.

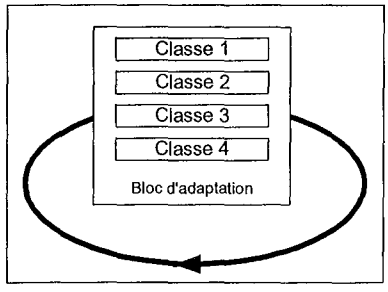


FIG. B.1 – Bloc d'adaptation de l'application de télé-asservissement

Les séquences CAP qui associent les classes de QdS, les séquences d'exécution et les

Classe de QdS	k	$rtt_{max}(s)$
1	0,9	0,1
2	0,7	0,3
3	0,5	0,5
4	0,3	1,1

TAB. B.1 – Classes de QoS de l'application de télé-asservissement

actions d'adaptation sont présentées dans le tableau B.2.

Séquence d'exécution	Classe de QdS	Action Haut	Action Bas
Réguler	1	-	SWITCH
Réguler	2	BEST_EFFORT	SWITCH
Réguler	3	BEST_EFFORT	SWITCH
Réguler	4	BEST_EFFORT	BLOCK

TAB. B.2 – Séquences CAP

Remarques :

- Quand le délai aller-retour est supérieur à $1,1s$, aucune des classes de QdS spécifiées n'est disponible. Dans ce cas, l'application bascule dans la séquence associée à la classe la plus basse (la classe 4) si elle n'y était pas et l'action d'adaptation Bas est activée : Ici, le système se bloque. Si le délai aller-retour est à nouveau inférieur à $1,1s$ avant la fin du bloc, l'application reprend l'exécution de la séquence associée à la classe de QdS 4.

B.2 Modélisation

Pour modéliser le caractère périodique de cette application, nous avons choisi d'utiliser les Réseaux de Petri temporisés. Ceux-ci sont une extension des Réseaux de Petri employés précédemment. Ils permettent de spécifier une durée de sensibilisation pour chaque transition, c'est-à-dire une durée entre l'instant auquel la transition est sensibilisée et l'instant auquel elle est tirée. Cette durée n, si elle est non-nulle, est indiquée à côté des transitions par un champ de la forme "@ $+n$".

Les classes de QdS sont modélisées par quatre couleurs de jeton : les couleurs A, B, C et D associées respectivement aux classes 1, 2, 3 et 4. Pour prendre en compte les

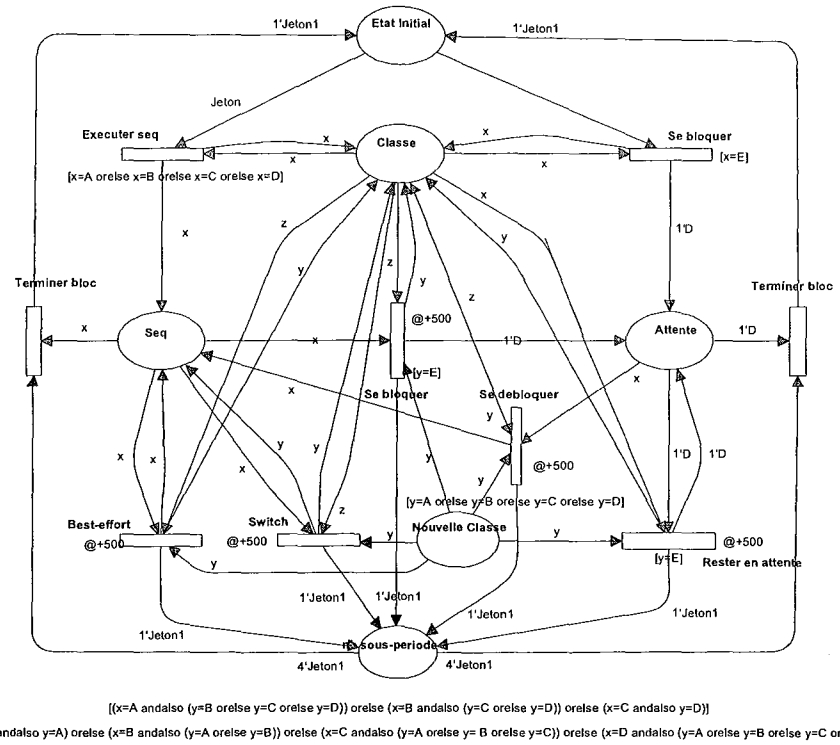


FIG. B.2 – Réseau de Petri du bloc d'adaptation

cas dans lesquels aucune des classes de QdS n'est disponible, nous définissons la couleur supplémentaire E. L'exécution d'une séquence est représentée par la présence d'un jeton de couleur correspondante dans la place "Seq" : Par exemple, l'exécution de la séquence associée à la classe de QdS 1 est modélisée par un jeton de couleur A dans la place "Seq".

La couleur du jeton présent dans la place "Classe" indique la classe de QdS courante. La couleur du jeton présent dans la place "Nouvelle Classe" donne la classe de QdS la plus adaptée et conditionne l'activation où non des actions d'adaptation. Contrairement aux modèles présentés précédemment, la présence d'un jeton dans cette place n'exprime pas nécessairement que la classe de QdS la plus adaptée est différente de la classe de QdS courante (il est possible d'avoir simultanément un jeton de couleur x dans la place "Classe" et dans la place "Nouvelle Classe").

Au démarrage de l'application, un jeton est présent dans la place "Etat initial" et la classe de QdS est indiquée par la présence d'un jeton de couleur correspondante (A, B, C,

D ou E) dans la place "Classe". Selon la couleur de ce dernier, une seule des transitions "Exécuter seq" et "Se bloquer" est franchissable (cf. figure B.2). Suite au tir de celle-ci, un jeton est mis dans la place correspondante ("Seq" ou "Attente").

La période d'un bloc d'adaptation est $2s$. Durant l'exécution d'un bloc, la classe de QdS la plus adaptée est recalculée par la couche de gestion qui l'indique en introduisant un jeton de couleur correspondante dans la place "Nouvelle Classe" à une période de $0,5s$. Autrement dit, la couleur du jeton présent dans la place "Seq" est remise à jour périodiquement (toutes les $0,5s$) lorsqu'une action d'adaptation SWITCH ou BEST_EFFORT doit être initiée. Lorsque l'action BLOCK doit être enclenchée, le jeton de la place "Seq" est supprimé, et un jeton D est positionné dans la place "Attente". Le jeton ne peut quitter cette place que lorsque la nouvelle classe de QdS est A, B, C ou D ou si l'exécution du bloc est terminée. La fin de l'exécution du bloc (fin de la période de $2s$) est indiquée par la présence de quatre jetons dans la place "nb sous-période", c'est-à-dire après quatre calculs de la classe de QdS la plus adaptée.

Remarques :

- Les durées de sensibilisation sont indiquées en ms.
- Les deux gardes indiquées en bas de la figure sont associées respectivement aux transitions Switch et Best-effort.

Annexe C

Langage UML

UML (Unified Modeling Language) est un langage d'analyse et de conception objet. Cet annexe présente les concepts de base et les différents types de diagrammes UML utilisés dans ce mémoire. Les définitions sont extraites de [Mul97].

C.1 Les objets

L'objet est une unité atomique formée de l'union d'un état et d'un comportement. L'objet révèle son vrai rôle et sa vraie responsabilité lorsque, par l'intermédiaire de l'envoi de messages, il s'insère dans un scénario de communication.

Les objets informatiques définissent une représentation abstraite des entités d'un monde réel ou virtuel, dans le but de les piloter ou de les simuler. Cette représentation abstraite peut être vue comme une sorte de miroir informatique, qui renvoie une image simplifiée d'un objet qui existe dans le monde perçu par l'utilisateur.

En UML, un objet se représente sous la forme d'un rectangle ; le nom de l'objet est souligné. Il est souvent difficile de trouver un nom pour désigner chaque objet ; c'est pourquoi la notation permet d'indiquer un nom générique plutôt que leur nom individuel. Le diagramme C.1 montre des élèves et des professeurs. Les deux points précisent qu'il s'agit d'objets anonymes, de genre Elève et Professeur.

C.1.1 Caractéristiques fondamentales d'un objet

Tout objet présente les trois caractéristiques suivantes : un état, un comportement et une identité.

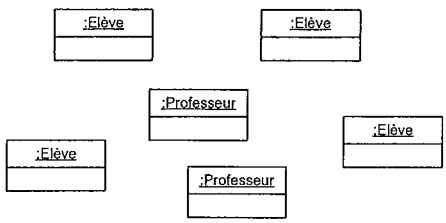


FIG. C.1 – Représentation d'objets anonymes

C.1.1.1 L'état

L'état regroupe les valeurs instantanées de tous les attributs d'un objet sachant qu'un attribut est une information qui qualifie l'objet qui le contient. L'état d'un objet, à un instant donné, correspond à une sélection de valeurs, parmi toutes les valeurs possibles des différents attributs.

C.1.1.2 Le comportement

Le comportement regroupe toutes les compétences d'un objet et décrit les actions et les réactions de cet objet. Chaque atome de comportement est appelé opération. Les opérations d'un objet sont déclenchées suite à une stimulation externe, représentée sous la forme d'un message envoyé par un autre objet.

Les interactions entre objets se représentent au moyen de diagrammes dans lesquels les objets qui interagissent sont reliés entre eux par des lignes continues appelées liens. La présence d'un lien signifie qu'un objet connaît ou voit un autre objet. Les messages naviguent le long des liens, a priori dans les deux directions.

C.1.1.3 L'identité

En plus de son état, un objet possède une identité qui caractérise son existence propre. L'identité permet de distinguer tout objet de façon non ambiguë, et cela indépendamment de son état. Cela permet, entre autres, de distinguer deux objets dont toutes les valeurs d'attributs sont identiques.

C.1.2 Diagramme d'objets

Les diagrammes d'objets montrent des objets et des liens. Ils représentent la structure statique et s'utilisent pour faciliter la compréhension des structures de données complexes.

C.2 Communication entre objets

Les systèmes informatiques à objets peuvent être vus comme des sociétés d'objets qui travaillent en synergie afin de réaliser les fonctions de l'application. Lors de l'exécution d'un programme, les objets contribuent solidairement au bon déroulement de l'application informatique. Le comportement global d'une application repose donc sur la communication par échange de messages entre les objets qui la composent.

C.2.1 Le concept de message

L'unité de communication entre objets s'appelle le message. Le message est le support d'une relation de communication qui relie, de façon dynamique, les objets. Les messages peuvent être de deux types dont les deux utilisés ici sont (cf. diagramme C.2) :

- Les **messages simples** : Cette catégorie convient pour les systèmes à un seul flot d'exécution, dans lesquels un seul objet à la fois est actif. Le passage du contrôle s'effectue lors de l'envoi d'un message de l'objet actif vers un objet passif. Un envoi de message simple se représente par une flèche simple.
- Les **message asynchrones** : Un message asynchrone n'interrompt pas l'exécution de l'expéditeur. L'expéditeur envoie le message sans savoir quand, ni même si, le message sera traité par le destinataire. Du point de vue du destinataire, un envoi asynchrone doit pouvoir être pris en compte à tout moment. Un envoi de message asynchrone se représente par une demi-flèche.

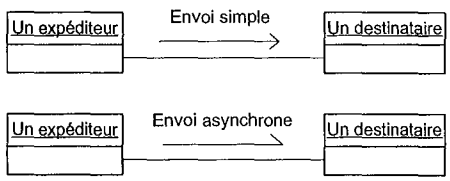


FIG. C.2 – Représentation d'envois de messages

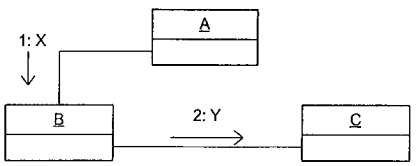


FIG. C.3 – Exemple de diagramme de collaboration

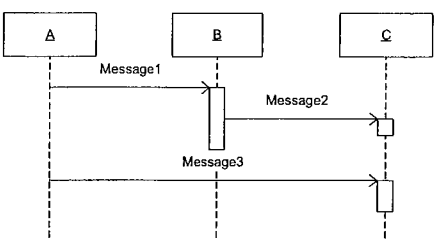


FIG. C.4 – Exemple de diagramme de séquence

C.2.2 Représentation des interactions entre les objets

Les objets interagissent pour réaliser collectivement les services offerts par les applications. Les diagrammes d'interaction représentent les objets les uns par rapport aux autres et montrent comment ils communiquent au sein d'une interaction. Il existe deux sortes de diagrammes d'interaction : les diagrammes de collaboration et les diagrammes de séquence. Le diagramme de séquence est particulièrement bien adapté pour la représentation d'interactions complexes, du fait de sa forme quasi tabulaire. Le diagramme de collaboration se prête mieux à la découverte des abstractions, car il permet de montrer les objets du domaine dans une disposition physique proche de la réalité.

- Les **diagrammes de collaboration** montrent quelques objets dans une situation donnée. Les objets sont représentés sous forme de rectangles, des liens relient les objets qui se connaissent (c'est-à-dire qui peuvent interagir) et les messages échangés par les objets sont représentés le long de ces liens. L'ordre d'envoi des différents messages est matérialisé par un numéro placé en tête du message (cf. diagramme C.3).
- Avec les **diagrammes de séquence**, l'accent est mis sur la communication. Chaque objet est représenté par une barre verticale. Le temps s'écoule de haut en bas, de sorte que la numérotation des messages est optionnelle (cf. diagramme C.4).

C.3 Les classes

Le monde réel est constitué de très nombreux objets en interaction. Ces objets constituent des amalgames souvent trop complexes pour être compris dans leur intégralité du premier coup. Pour réduire cette complexité, l'être humain a appris à regrouper les éléments qui se ressemblent et à distinguer des structures de plus haut niveau d'abstraction, débarrassées de détails inutiles.

La démarche d'abstraction procède de l'identification des caractéristiques communes à un ensemble d'éléments, puis de la description condensée de ces caractéristiques dans ce qu'il est convenu d'appeler une classe. La classe décrit le domaine de définition d'un ensemble d'objets. Chaque objet appartient à une classe. Les généralités sont contenues dans la classe et les particularités sont contenues dans les objets. Les objets informatiques sont construits à partir de la classe, par un processus appelé instanciation. De ce fait, tout objet est une instance de classe.

Chaque classe est représentée sous la forme d'un rectangle divisé en trois compartiments. Le premier compartiment contient le nom de la classe, le second les attributs et le dernier les opérations (cf. diagramme C.5).

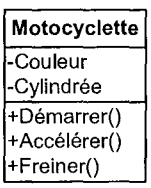


FIG. C.5 – Exemple d'une classe Motocyclette

C.3.1 Diagramme de classes

Les diagrammes de classes expriment de manière générale la structure statique d'un système, en termes de classes et de relations entre ces classes (associations, généralisation,etc.).

C.4 Diagramme d'activités

Un diagramme d'activités représente l'état de l'exécution d'un mécanisme sous la forme d'un déroulement d'étapes regroupées séquentiellement. Une activité est représentée par un rectangle arrondi (cf. diagramme C.6). Les activités sont reliées par des transitions automatiques représentées par des flèches. Lorsqu'une activité se termine, la transition est déclenchée et l'activité suivante démarre. Les transitions entre activités peuvent être gardées par des conditions booléennes, mutuellement exclusives. Les gardes se représentent à proximité des transitions dont elles valident le déclenchement. Une condition est matérialisée par un losange d'où sortent plusieurs transitions.

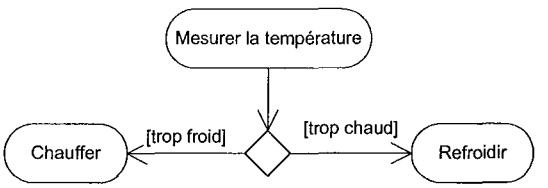


FIG. C.6 – Exemple de diagramme d'activités

Annexe D

Publications

Conférences Internationales avec Comité de Lecture

Gnaedinger E., Michaut F., Lepage F., "Implémentation d'une architecture de QdS spécifique permettant l'adaptation d'une application de contrôle à distance d'un robot mobile," *Quatrième Conférence Internationale sur l'Automatisation Industrielle*, Montréal, Canada, Juin 2003.

Michaut F., Lepage F., "Networks Quality of Service Monitoring," *IAR Annual Meeting*, Grenoble, France, Novembre 2002.

Michaut F., Lepage F., "A Tool to Monitor the Network Quality of Service," *NETCON'2002, IFIP-IEEE Conference on Network Control and Engineering*, Paris, France, Octobre 2002.

Michaut F., Lepage F., "An Architecture for Application Adaptation based on QdS Monitoring," *SMC'02, IEEE International Conference on Systems, Man and Cybernetics*, Hammamet, Tunisie, Octobre 2002.

Michaut F., Lepage F., "A QdS Architecture for Application Execution Adaptation," *SNPD'02 ACIS 3rd International Conference on Software Engineering Artificial Intelligence, Networking and Parallel/Distributed Computing*, Madrid, Spain, Juin 2002.

Conférence Nationale avec Comité de Lecture

Michaut F., Lepage F., "Un cadre pour la prise en compte du retard dans la téléoperation d'un robot mobile," *Journées Doctorales d'Automatique 2003*, Valenciennes, France, Juin 2003.

Monsieur **Fabien MICHAUT**

DOCTORAT DE L'UNIVERSITE HENRI POINCARE, NANCY-I

en Automatique, Traitement du Signal et Génie Informatique.

VU, AP PROUVÉ ET PERMIS D'IMPRIMER N° 897

Nancy, le 10 décembre 2003

Le Président de l'Université

UNIVERSITÉ HENRI POINCARÉ, NANCY 1 * LE PRÉSIDENT

Claude BURLET

UNIVERSITÉ HENRI POINCARÉ · NANCY 1

24-30, rue Lionnois B.P. 3069 54013 Nancy cedex - Tél. 03 83 85 48 00 - Fax 03 83 85 48 48

Adresse électronique : _____ @uhp.u-nancy.fr

Adaptation des applications distribuées à la Qualité de Service fournie par le réseau de communication

Mots-clés:
Réseaux de Communication, Qualité de Service (QdS), Application Adaptative, Métrologie Réseau, Télé-opération.

Résumé:
Aujourd'hui, les échanges d'information prennent des dimensions géographiques plus importantes. De ce fait, les réseaux utilisés sont le plus souvent hétérogènes et non déterministes. Parallèlement, les informations transférées sont de plus en plus complexes et contraintes en termes de Qualité de Service (QdS). Ainsi, les réseaux de communication à commutation de paquets initialement conçus pour acheminer des trafics sans contraintes particulières doivent à présent satisfaire les besoins d'applications distribuées "critiques".
Les travaux menés dans cette thèse visent à doter les applications distribuées de moyens leur permettant de pallier l'insuffisance du système de communication en adaptant leur exécution en ligne. Dans ce contexte, nous apportons les contributions suivantes: une architecture de QdS "QdS-Adapt" et un service de métrologie de la QdS. L'architecture proposée fournit un cadre générique pour l'adaptation des applications et intègre le service de métrologie de la QdS. Ce dernier est conçu de manière originale pour accueillir des opérateurs de mesure de toutes sortes de paramètres.
La faisabilité et le bon fonctionnement de l'architecture ont été vérifiés au travers des deux applications de télé-opération. Les expérimentations montrent que l'architecture rend possible la mise en œuvre d'une stratégie pour s'adapter à la variation de la QdS du réseau et qu'il est possible de limiter l'impact des dégradations introduites par le réseau sur le fonctionnement de l'application.

Keywords:
Communication Networks, Quality of Service (QoS), Adaptive Application, Network Metrology, Tele-operation

Abstract:
Various new networking applications have appeared in the past ten years. Packet switched networks initially designed for the transfer of text data are used nowadays by more complex traffics with strong constraints in terms of Quality of Service (QoS).
Our approach considers that no strict guarantee can be obtained from communication system. In this context, we propose a new QoS Architecture called "QdS-Adapt". It offers a general framework that permits applications to adapt dynamically their execution to resources fluctuations. The architecture includes a network metrology service. It has been designed to allow the implementation of various measurement techniques.
The feasibility of the architecture has been verified on two tele-operation applications of a mobile robot. Experimentations show that the architecture allows on-line adaptation strategies to be implemented and that the impact of QoS degradations on the application functioning can be limited.