Architecture Autonome et Extensible pour une Couche de Transport Évolutive. Application aux Communications Aéronautique par Satellites

Mohamed Oulmahdi

▶ **To cite this version:**

Mohamed Oulmahdi. Architecture Autonome et Extensible pour une Couche de Transport Évolutive. Application aux Communications Aéronautique par Satellites. Réseaux et télécommunications [cs.NI]. INSA de Toulouse, 2017. Français. NNT : 2017ISAT0015 . tel-01660846v2

HAL Id: tel-01660846

https://theses.hal.science/tel-01660846v2

Submitted on 23 Feb 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :
l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)
Cotutelle internationale *Université Abderrahmane Mira de Béjaia*

Présentée et soutenue le *24/11/2017* **par :**
MOHAMED OULMAHDI

Architecture Extensible et Autonome pour une Couche de Transport Evolutive
Application aux Communications Aéronautiques par Satellite

JURY

CHÉRINE GHEDIRA	Professeur	Présidente
ABDELLAH BOUKERRAM	Professeur	Examinateur
LAYTH SLIMAN	Maitre de conférences	Examinateur
KHALIL DRIRA	Directeur de Recherche	Invité

École doctorale et spécialité :
MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture
Unité de Recherche :
LAAS-CNRS
Directeur(s) de Thèse :
Christophe Chassot, *Nicolas Van Wambeke* et *Abdelkamel Tari*
Rapporteurs :
Abdelhamid Mellouk et *Mourad Ousalah*

To my wife, and our futur child.

Remerciements

Les travaux présentés dans ce manuscrit ont été effectués au sein du Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique (LAAS-CNRS). Je tiens à remercier l'ensemble du personnel, en particulier son directeur.

Mes travaux ont été encadrés par Messieurs C. Chassot, A. TARI et N. Van Wambeke, à qui je souhaite témoigner ma plus sincère gratitude pour leur écoute et leurs conseils tout au long de ces trois années. Je les remercie tout particulièrement pour les nombreuses réflexions que nous avons pu mener ensembles et au travers desquelles ils ont partagé une partie de leur expérience avec moi.

La thèse a été réalisée en collaboration avec Thales Alenia Space, que je remercie pour l'intérêt qu'ils ont porté à mon travail, et les moyens qu'ils ont mis en oeuvre pour sa réalisation.

La thèse a aussi été réalisée en cotutelle avec l'université de Béjaia. Je tiens à les remercier pour leur accueil et les efforts de collaboration qu'ils ont fournis.

Enfin, je remercie mes collègues et amis, ainsi que tous ceux qui ont su m'accompagner et m'épauler pendant ces trois années de thèse.

Résumé

Ces dernières décennies ont été caractérisées par une évolution massive de l'Internet sur tous les plans, couvrant les applications et les technologies réseau. En conséquence, de nouveaux besoins pour les applications et de nouvelles contraintes réseaux apparaissent ; rendant ainsi les protocole (TCP et UDP notamment) de moins en moins efficaces, et plusieurs nouveaux protocoles ont été proposés. Cependant, à cause de plusieurs limites architecturales de la couche Transport, ces nouveaux protocoles n'ont pas été déployés.

Partant de ce constat, le travail effectué dans cette thèse porte sur la proposition et la réalisation d'une architecture pour la couche Transport, orientée services et basée composants, dotée de capacités d'extensibilité et d'auto-adaptation vis-à-vis des évolutions du contexte applicatif et réseau. La solution proposée repose, d'une part, sur un faible couplage entre les éléments extérieurs (applications et systèmes) et la couche Transport, ainsi qu'entre les composants internes de l'architecture. D'autre part, elle se base sur des modèles et des algorithmes lui permettant de détecter et de prendre en compte les évolutions du réseau ou des applications, et d'adapter son comportement en conséquence. Une implémentation complète de la solution est proposée et testée dans un cadre de communications aéronautiques par satellite. L'objectif étant la gestion de la transition des protocoles spécifique au monde aéronautique vers les protocoles de l'Internet, ainsi que la gestion, au niveau Transport, des liens physiques hétérogènes. Les tests démontrent la faisabilité d'une telle architecture extensible et autonome, les gains en performance qu'il est possible d'obtenir, et les coûts qui en résultent.

Table des matières

Résumé **i**

Table des matières **ii**

Table des figures **v**

Introduction **1**

1 Etat de l'art et Positionnement **5**
1.1 Evolution des protocoles de Transport 6
1.1.1 Approche générales et approches spécifiques 6
1.1.2 Solutions pour les applications et solutions pour les réseaux . . . 9
1.1.3 Nouveaux protocoles et nouvelles versions 10
1.1.4 Analyse comparative 10
1.2 Limites de la couche Transport 12
1.2.1 Configurabilité 12
1.2.2 Adaptabilité 13
1.2.3 Flexibilité 14
1.2.4 Complexité 14
1.3 Évolutions architecturales au niveau Transport 15
1.3.1 Nouvelles architectures 15
1.3.2 Outils conceptuels pour les architectures configurables 20
1.3.3 Autonomie 23
1.4 Le modèle ATN/OSI 25
1.4.1 Présentation de l'ATN 26
1.4.2 La pile de protocoles ATN/OSI 26
1.4.3 La transition vers IP 27
1.5 Présentation des contributions 28
1.5.1 Positionnement des contributions 28
1.5.2 Contributions de la thèse 29
1.6 Conclusion 29

2 Architecture pour la couche Transport **31**
2.1 Limites et besoins 32
2.1.1 Limites des architectures existantes 32
2.1.2 Besoins pour une couche Transport évolutive 34
2.2 Une nouvelle architecture pour la couche Transport 36
2.2.1 Principes généraux et positionnement 36
2.2.2 Description globale de l'architecture 37

2.2.3 Structuration par rapport aux besoins . . . 39
2.3 Description des composants . . . 39
2.3.1 Composants de gestion de l'architecture . . . 40
2.3.2 Bases de données . . . 41
2.3.3 Composants de gestion des connexions . . . 42
2.4 Cas d'utilisation et description comportementale . . . 43
2.4.1 Cas d'utilisation . . . 43
2.4.2 Description comportementale . . . 47
2.5 Conclusion . . . 50

3 Extensibilité de la couche Transport 53
3.1 Approches de conception . . . 53
3.1.1 Description des paradigmes . . . 54
3.1.2 Application à la couche Transport . . . 57
3.2 Extensibilité vis-à-vis des systèmes . . . 58
3.2.1 Indépendance au noyau . . . 59
3.2.2 Contraintes de mise à l'échelle . . . 60
3.3 Extensibilité des composants de communication . . . 60
3.3.1 Décomposition par services . . . 60
3.3.2 Composant d'intégration et base locale des composants . . . 61
3.3.3 Interactions entre composants . . . 61
3.3.4 Invocation des composants . . . 62
3.3.5 Autres éléments extensibles . . . 63
3.4 Extensibilité des composants de gestion . . . 65
3.4.1 Interface applications . . . 65
3.4.2 Composant de décision . . . 66
3.4.3 Cas du gestionnaire des applications legacy . . . 67
3.5 Extensibilité des applications . . . 68
3.5.1 Base globale des services . . . 68
3.5.2 Interface abstraite . . . 68
3.6 Conclusion . . . 69

4 Autonomie de la couche Transport 71
4.1 Besoins et problématique . . . 72
4.1.1 Besoin en autonomie . . . 72
4.1.2 Niveaux d'autonomie . . . 73
4.1.3 Autonomie de la sélection des composants . . . 74
4.2 Monitoring et changement de contexte . . . 77
4.2.1 Recueil des données de monitoring . . . 77
4.2.2 Profils de réseau et changements de contextes . . . 78
4.2.3 Algorithme de détection de changement de contexte . . . 80
4.3 Processus de décision . . . 86
4.3.1 Eléments théoriques . . . 86
4.3.2 Traitement de la requête de l'application . . . 87
4.3.3 Sélection des composants . . . 90

5 Implémentation et Évaluation 97
5.1 Implémentation et déploiement . . . 98
5.1.1 Intégration et interaction des composants . . . 98

5.1.2 Principes d'intégration au noyau 99
5.1.3 Décision prédéfinie 102
5.2 Cas d'étude et évaluation 103
5.2.1 Cas d'étude 103
5.2.2 Evaluation des performances 106
5.3 Conclusion 116

Conclusion et Perspectives **117**

Bibliographie **121**

Table des figures

1.1 Classification des solutions de Transport . . . 10
1.2 Architecture globale de ADAPTIVE . . . 16
1.3 Architecture globale de CTP . . . 17
1.4 Architecture globale de APPIA . . . 17
1.5 Architecture globale de CTP . . . 18
1.6 Architecture globale de CTP . . . 19
1.7 Exemple de décomposition de TCP en micro-protocoles . . . 21
1.8 Modèle événementiel . . . 21
1.9 Modèle événementiel . . . 22
1.10 Modèle de l'Autonomic Computing . . . 24
1.11 Schéma d'un réseau ATN . . . 26
1.12 Pile de protocoles ATN/OSI . . . 27
1.13 Problème de transition vers IP . . . 28

2.1 Architecture globale . . . 38
2.2 Interactions entre la couche Transport et les applications . . . 44
2.3 Exemple d'intéraction entre deux applications . . . 46
2.4 Interactions entre deux systèmes distants . . . 47
2.5 Exemple d'interactions entre deux systèmes distants . . . 48
2.6 Diagrame d'état de l'établissement de connexion . . . 49

3.1 Schéma d'une architecture orientée-service . . . 55
3.2 Schéma d'une architecture basée-composants . . . 56
3.3 Principe de la sépartion du noyau . . . 59
3.4 Modèle d'association service/composant . . . 61

4.1 Classification des solutions de Transport . . . 78

5.1 Principe d'interaction entre les composants de l'architecture . . . 100
5.2 Principe de la séparation du noyau . . . 102
5.3 Modèles de prise en compte de la transition vers les standards IPS . . . 104
5.4 Plateforme d'évaluation . . . 107
5.5 Test d'équivalence de performances . . . 108
5.6 Test de fairness . . . 108
5.7 Test de friendliness . . . 108
5.8 Occupation de la bande passante sans adaptabilité . . . 110
5.9 Occupation de la bande passante avec adaptabilité . . . 111
5.10 Pic momentané - Représentation des moyennes . . . 112
5.11 Pic momentané - Représentation des écarts . . . 112

5.12 Pic permanent - Représentation des moyennes 113
5.13 Pic permanent - Représentation des écarts 113
5.14 Evolution progressive momentannée - Représentation des valeurs 114
5.15 Evolution progressive momentannée - Représentation des écarts 114
5.16 Evolution progressive momentannée - Représentation de la moyenne des écarts 114
5.17 Evolution progressive permanente- Représentation des valeurs 115
5.18 Evolution progressive permanente- Représentation des écarts 115
5.19 Evolution progressive permanente- Représentation de la moyenne des écarts 115

Introduction

Ces dernières années ont vu une évolution massive dans le domaine des technologies de l'information et des télécommunications. Ceci concerne à la fois les parties matérielle et logicielle. En particulier, l'Internet a connu la même évolution au niveau des équipements de communication et des applications destinées aux utilisateurs. En effet, des technologies de plus en plus puissantes (fibre optique, routeurs à très haut débit, ...) viennent à présent relier les utilisateurs de l'Internet, contribuant à accroître les performances de façon phénoménale, tant pour les particuliers que pour les professionnels. De même, les applications de l'Internet offrent de plus en plus de services, impliquant notamment de la vidéo de très haute définition, aux immenses espaces de stockage en ligne.

Si l'on considère tout particulièrement les protocoles, relevant notamment des technologies réseaux sous-jacentes à l'Internet, nous pouvons constater les versions successives des protocoles relevant des couches physique et liaison de données, de l'Ethernet de base au Gigabits Ethernet, du WiFi b au g, au n et au AC. Il en est de même pour les protocoles applicatifs, pour le transfert de fichier, le courrier électronique, et autre. Des protocoles de plus haut niveau ont également émergé notamment pour le web, tel que le protocole SOAP pour les web services.

Dans ce contexte a priori très favorable, il existe cependant une partie de ces protocoles, moins visible, et dont l'évolution reste à examiner. Il s'agit des protocoles des couches (inter-)Réseau et Transport de l'Internet, qui font partie des systèmes d'exploitation. Les fameux protocoles IP, TCP et UDP, conçus dans les années 1970, encore présents dans l'Internet, ont-ils évolué, et d'autres protocoles ont-ils émergés à l'instar de ceux des autres couches ?

Dans les faits, ces protocoles, notamment ceux de la couche Transport, n'ont que très faiblement évolué depuis leur version de base. Quelques mécanismes supplémentaires ont été ajoutés à TCP, notamment dans les années 1990 pour (par exemple) protéger le réseau, mais le fonctionnement global du protocole est resté pratiquement le même. Ainsi, TCP constitue jusqu'à présent le seul protocole de Transport universellement déployé, en dépit des efforts de standardisation de protocoles tels que DCCP, SCTP ou plus récemment MPTCP.

Pour autant, nous trouvons dans la littérature une quantité importante de propositions de nouveaux protocoles de Transport, ou encore de nouvelles versions du protocole TCP. Toutes ces propositions présentent en principe de meilleures performances que le protocole TCP actuel ou sa version de base. Elles offrent aussi des services plus adaptés aux nouvelles applications de l'Internet. Dans le même temps, les performances de

TCP restent toujours dégradées, et le service qu'il offre aux applications n'a pas évolué.

Les réseaux satellites est l'un des exemples où un protocole comme TCP offre des performances dégradées. En effet, il se peut que dans certaines conditions, la portion de la bande passante satellite utilisée par ce protocole n'atteigne pas les 20

C'est cette problématique de l'évolution de la couche Transport que nous adressons dans ce mémoire de thèse, au travers duquel nous posons quatre questions principales :
- Pour quelles raisons les nouvelles propositions au niveau Transport n'ont pas été déployées, bien qu'elles soient plus évoluées
- Quelles sont les solutions qui ont été proposées, et quelles en sont les limites
- Comment proposer une solution qui permette l'évolution de la couche Transport via l'intégration des nouvelles propositions.
- Comment cette solution permettra de résoudre la problématique associée à notre cas d'étude.

Le travail présenté dans cette thèse consiste ainsi et dans un premier temps à étudier les obstacles face à l'évolution de la couche Transport, et à examiner les travaux qui ont été entrepris dans ce cadre. En effet, un certain nombre de travaux ont été menés dans l'objectif de proposer de meilleures architectures pour les protocoles de Transport, mais comme les autres propositions, elles n'ont dans les fait pas été déployées. De ces travaux et de l'étude de leurs limites, nous reprenons ou nous inspirons de certaines notions dans le cadre de la solution que nous proposons, notamment pour ce qui concerne la décomposition des services de Transport.

Nous proposons ensuite la conception d'une solution architecturale pour la couche Transport permettant de répondre à l'ensemble de ces obstacles. Cette solution est basée sur des paradigmes de conception plus évolués que ceux utilisés classiquement dans le monde des réseaux, tel que l'orienté service ou le basé composant ; elle repose également sur une séparation entre les éléments de communication proprement dits (au sens mécanismes de contrôle de reprise des pertes, de contrôle de congestion, etc.), et les éléments de gestion des éléments précédents. Notre solution se base également sur des mécanismes de Transport spécifiques, dont chacun est adapté à un type d'application et de réseau particulier. L'ensemble des services de communication sont séparés en des composants indépendants.

L'architecture de notre solution couvre deux dimensions importantes, qui sont l'extensibilité et l'auto-adaptabilité. L'extensibilité permet l'intégration de nouvelles solutions de Transport, et l'auto-adaptabilité permet de les prendre en compte et de les associer au domaine correspondant de façon dynamique et transparente pour l'application. Ces deux dimensions sont assurées par un ensemble de paradigmes de conception et de modèles théoriques, qui sont respectivement présentés et proposés dans ce mémoire.

Enfin, la solution est implémentée et testée dans le cadre d'un cas d'étude relatif aux communications aéronautique par satellite. Ceci rejoint les objectifs de la société Thales Alenia Space qui est spécialisée dans la construction de systèmes de communication satellitaires. La problématique traitée s'accorde avec la problématique générale de la thèse, en rapport avec l'évolution de la couche Transport. Il s'agit principalement de gérer la phase de transition de la pile de communication du réseau aéronautique (l'ATN) qui suit actuellement les standards ATN/OSI vers les protocoles de l'Internet adaptés

à l'aéronautique dans la norme ATN/IPS. Durant la phase de transition, des protocoles de piles différentes doivent coexister. Les tests réalisés portent sur la faisabilité de la solution, les gains obtenus ainsi que le coût associé.

Les contributions de ce travail peuvent être résumées en 4 parties principales :
- L'étude et l'analyse des limites architecturale de la couche Transport actuelle face à l'évolution de ses protocoles, et l'identification des besoins résultant ;
- La conception d'une solution architecturale qui permet de prendre en compte l'ensemble de ces limites ;
- L'intégration de nouveaux concepts et modèles pour assurer l'extensibilité et l'auto-adaptabilité de l'architecture proposée ;
- L'implémentation intégrale de la solution proposée et son évaluation dans le cadre d'un cas d'étude relevant des communications aéronautiques.

Ce mémoire est structuré en cinq chapitres.

Le Chapitre 1 consiste en un état de l'art des évolutions relevant de la couche Transport, des architectures proposées et des outils communs qui sont utilisés. Le chapitre présente également un état de l'art des communications aéronautiques dans le cadre desquels l'évaluation de notre solution est effectuée. Enfin le chapitre positionne notre travail par rapport aux solutions existantes, et présente les contributions.

Le Chapitre 2 présente la solution proposée, après étude des limites des solutions actuelles et des besoins qui en découlent. La solution est exposée par ses différents composants, et ses comportements de base, à l'aide de diagrammes UML 2.0.

Les Chapitre 3 et 4 sont respectivement consacrés aux dimensions d'extensibilité et d'auto-adaptabilité de l'architecture proposée. L'extensibilité couvre, en plus de l'architecture elle-même, les applications et les systèmes dans laquelle elle aura vocation à être déployée, et ce via différents paradigmes de conception. L'auto-adaptabilité est présentée principalement par les modèles et les algorithmes qui y sont utilisés, notamment pour le monitoring du réseau avec un établissement de profil et détection de changement de contexte, et pour le modèle de décision associé à la sélection des composants.

Le Chapitre 5 traite des aspects d'implémentation, de déploiement et de validation de l'architecture. Au-delà de l'implémentation que nous avons réalisée de notre solution (qui se situe dans l'espace utilisateur), nous expliquons également les principes techniques de déploiement de notre solution dans le noyau du système d'exploitation. Le chapitre présente aussi le déploiement de l'architecture dans un environnement de communication aéronautique, et son application aux problématiques spécifiques au cas d'étude considéré dans un environnement satellitaire. Enfin, le chapitre décrit les tests de validation effectués ainsi que les résultats obtenus.

Finalement, une conclusion de ce travail de thèse est présentée, incluant les travaux effectués, les résultats obtenus et les messages à retenir. Plusieurs perspectives de suites à donner à ce travail sont également exposées.

Chapitre 1

Etat de l'art et Positionnement

Le domaine des réseaux informatiques est l'un des plus évolutifs de notre époque. Au fil des années, les besoins des applications comme les capacités des réseaux n'ont cessé d'évoluer. Les applications ont vu leurs services se diversifier et s'améliorer, passant de simples échanges de textes et de transfert de fichiers à des services multimédias et interactifs. De ces nouveaux services résultent logiquement de nouveaux besoins en communication, que ce soit en termes de fonctionnalités ou de performances. Par ailleurs, les technologies réseaux ainsi que leurs protocoles (relevant notamment des couches basses) ont beaucoup évolué ces dernières décennies. Cette large hétérogénéité de "profils" réseau rend ainsi difficile la pose d'hypothèses génériques sur lesquelles appuyer la spécification de protocoles de plus haut niveau, notamment ceux relevant de la couche Transport.

La couche Transport, niveau d'abstraction se situant entre les couches applicative et réseau, se trouve directement concernée, et doit à son tour évoluer pour mieux servir les applications et les faire profiter au maximum de ces nouvelles technologies. Dans les faits, de nouvelles solutions ont été proposées au fur et à mesure que les applications et les réseaux ont évolué. La plupart de ces solutions sont dédiées à des contextes spécifiques, soit sous forme de nouveaux protocoles, soit sous forme d'améliorations de protocoles existants, le plus souvent TCP, tandis que d'autres se veulent plus générales. Cependant, le constat actuel est que ces solutions ne voient pas le jour pour la majorité d'entre elles et que le monde des réseaux continue de fonctionner avec les anciennes solutions (TCP en premier lieu) qui, depuis leur apparition dans le début des années 1980, n'ont pas fondamentalement évolué, à l'exception du traitement de besoins extrêmes.

Le non déploiement de ces nouvelles solutions vient d'un ensemble de limites de la couche Transport, qui n'a pas été conçue pour évoluer dans le temps. En réponse à ces limites et en l'absence d'une véritable architecture pour la couche Transport, de nouvelles architectures ont été proposées pour certains protocoles, principalement dans l'objectif de faciliter l'intégration de nouveaux mécanismes protocolaires et d'améliorer les performances offertes aux applications. Cependant, ces architectures ayant été proposées comme des protocoles, à l'instar des protocoles classique, elles sont toujours confrontées, lors de leur déploiement, à aux limites (extensibilité, dépendance, ...) dont souffre la couche Transport actuelle. Nous développerons ces limites un peu plus loin dans ce chapitre.

Dans ce chapitre, nous décrivons tout d'abord les évolutions qui ont été faites au niveau des protocoles de Transport en tant que solutions à des problématiques de communication dans des contextes spécifique ou généraux (section 1). Nous exposons ensuite les limites de la couche Transport qui justifient le non (ou le peu de) déploiement des nouveaux protocoles dans l'Internet. Nous introduisons alors les architectures qui ont été proposées pour répondre à ces limites ainsi que les outils conceptuels et les techniques adoptées pour les mettre en oeuvre, tout en discutant également de leurs limites (section 2).

Nous introduisons enfin le contexte d'application de la solution que nous proposons, qui est relatif aux communications aéronautiques (section 3).

Finalement, nous positionnons nos travaux et présenterons les contributions associées.

1.1 Evolution des protocoles de Transport

L'orientation des évolutions au niveau Transport a été guidée par plusieurs facteurs et plusieurs classifications peuvent ainsi être établies. Selon le champ d'application de la solution, on distingue des solutions : 1) spécifiques à un domaine particulier, 2) générales au domaine Internet notamment, ou 3) hybrides, c'est à dire combinant les deux approches. Selon la nature de l'évolution elle-même, on distingue des propositions de nouveaux protocoles, de nouveaux mécanismes pour des protocoles existants ou des améliorations à des mécanismes déjà existants. Enfin, selon la couche ciblée, les évolutions peuvent concerner les applications, les réseaux ou les deux à la fois. Les travaux sur le Transport se comptent par plusieurs centaines et il n'est donc pas possible de tous les référencer. A titre illustratif, nous en citerons les principaux en essayant de couvrir l'ensemble de la période d'évolution. Nous décrivons dans la suite de cette section les travaux sur le Transport selon les trois angles introduits ci-avant.

1.1.1 Approche générales et approches spécifiques

Les champs d'utilisation des solutions de Transport varient en fonction du besoin en termes de communication ou des avantages qu'il est possible d'obtenir en élargissant ou en restreignant ces champs d'utilisation. La majorité des travaux concernent des champs d'utilisation spécifiques. Il s'agit notamment de contextes applicatifs ou réseaux présentant certaines particularités qui rendent les protocoles classiques inefficaces. D'autres solutions tentent de couvrir un champ d'utilisation plus large (à l'échelle de l'Internet notamment) à la façon des protocoles classiques tels que TCP. Enfin, un dernier type propose une combinaison des deux aspects par le biais de protocoles "adaptatifs" adoptant des comportements différents selon le contexte. Nous étudions ci-après les trois approches.

Approche spécifique

Les évolutions des besoins des applications et des technologies réseaux rendent leurs particularités de plus en plus courantes. Les réseaux sans fil présentent des taux de pertes plus élevés que les réseaux filaires, les réseaux satellites présentent des délais plus élevés que les réseaux locaux, les réseaux ad-hoc de type "pair à pair" ou "machine

à machine" présentent des variations importantes dans les délais, etc. De la même façon, les applications multimédias et de transfert ont besoin de débits de plus en plus importants, pendant que des applications interactives et critiques ont besoin d'un minimum de délai, ou encore n'ont plus besoin de certains services, tels que ceux concourant à une garantie de fiabilité totale.

La plupart des travaux proposent donc de nouvelles solutions dont chacune est adaptée à un champ d'utilisation spécifique. Les solutions spécifiques ciblent un type particulier d'applications afin de lui offrir un meilleur service ou un type particulier de réseau afin d'en optimiser l'utilisation. Par conséquent, les solutions spécifiques présentent naturellement de hautes performances (relativement ou dans l'absolu) étant donné leur connaissance préalable des aspects de leurs contextes d'utilisation.

Les performances de ce type de solutions ne sont cependant pas équivalentes et varient selon le degré de spécificité du contexte ciblé. Cette variation de spécificité vient du fait que pour chaque groupe d'application ou de technologie réseaux ayant des besoins et des caractéristiques relativement similaires, il est possible de trouver un sous-groupe plus distinctif. On peut à ce titre trouver des solutions spécifiques à l'ensemble des réseaux sans fil, caractérisés par d'important délais et taux de pertes, comme les deux versions de contrôle de congestion proposées pour TCP dans [1][2] afin d'adapter le protocole à des liaisons sans fil. On peut néanmoins trouver des solutions, toujours spécifiques pour des liaisons sans fil, avec davantage de précision. A titre d'exemple, des solutions ont été proposées pour des réseaux satellitaires, comme dans [3][4], ou pour des réseaux locaux sans fil, comme dans [5][6], dont les différences entre les délais et les taux de pertes sont importantes. D'autres protocoles ont été proposés pour tenir compte de besoins particuliers, comme pour l'économie d'énergie dans [7][8].

Il en est de même pour les applications. Plusieurs solutions ont été proposées pour des applications gérant des flux vidéo, et présentant des besoins en délai et en débit. Toutefois, ces applications multimédias sont de différents types. On trouve donc des solutions générales, comme dans [9][10] qui proposent deux versions de TCP pour les applications multimédias. On trouve également d'autres solutions plus spécifiques traitant des applications multimédia interactives qui présentent des besoins en délai [11], à l'exemple des vidéo-conférences, ou encore des solutions pour des applications de streaming vidéo ayant plus de besoin en débit [12].

La spécificité d'une solution joue considérablement sur ses performances. En effet, c'est pour obtenir de meilleures performances que des solutions plus spécifiques ont été créées. Cependant, plus la solution est spécifique et plus son champ d'application se restreint, ce qui réduit ainsi ses possibilités de déploiement.

Approche générale

A l'inverse des solutions spécifiques, ce type de solutions a été conçu pour fonctionner de façon universelle. Comme il n'est pas possible de regrouper tous les besoins applicatifs et toutes les caractéristiques des technologies des réseaux en une seule solution, le principe est d'offrir un service couvrant les besoins indispensables des applications les plus utilisées, et de prendre en compte les conditions les plus contraignantes des réseaux, en considérant les hypothèses les plus courantes en terme de délais ou pertes notamment.

Les solutions de cette catégorie regroupent principalement les protocoles de l'Internet ou des variantes propriétaires (comme le protocole TP4 du réseau de télécommunication aéronautique ATN ou encore l'ancien protocole ATP d'Apple). Là encore, la plupart sont basées sur les protocoles de base TCP et UDP avec un service fiable/non fiable, ordonné/non ordonné, en flux d'octets ou de message, avec ou sans contrôle de congestion, etc. On peut citer SCTP (Stream Control Transport Protocol) [13], qui est une variante de TCP avec flux de messages et plusieurs flux par connexion. Le protocole DCCP (Datagram Congestion Control Protocol) [14] est en quelques sorte une variante du protocole UDP avec contrôle de congestion et établissement de connexion. On peut citer enfin le protocole MPTCP [15] qui est une version de TCP capable de transmettre ses unités de données sur plusieurs chemins.

Ces solutions sont capables de fonctionner pratiquement sur n'importe quelle technologie réseau et peuvent être utilisées par tout type d'application. Leurs performances sont toutefois très loin de celles des solutions spécifiques même dans les conditions les plus favorables. Cela se traduit par un "mauvais" service pour les applications, une sous-optimalité dans l'exploitation des réseaux sous-jacents, ou encore les deux à la fois. En effet, plus le contexte d'utilisation de la solution s'éloigne du modèle sur lequel elle se base, et plus les performances se dégradent.

En pratique, en dépit de leurs faibles performances, ces solutions sont les plus privilégiées et les plus promues pour leurs avantages de déploiement. On les retrouve ainsi dans la plupart des systèmes d'exploitation, des outils de développement, et la garantie de leur accès dans les réseaux est plus probable.

Approche hybride

Il existe une troisième catégorie de protocoles de Transport qui tente de combiner les deux approches précédentes et de tirer parti de leurs avantages respectifs. Il s'agit des solutions "adaptatives", capables d'adopter différents comportements selon le contexte de communication (applicatif et réseau), et ce au sein d'une même solution. Schématiquement, elles ont donc " l'aspect " des solutions générales et le comportement des solutions spécifiques.

Elles se basent sur un principe de composition dynamique suivant lequel le protocole dispose de plusieurs composants, chacun adapté à un service particulier avec plusieurs versions par service. L'idée, ensuite, est de sélectionner et de composer, selon les besoins de l'application et les caractéristiques du réseau sous-jacent, les composants les plus adaptés. Ceci requiert bien entendu des interfaces plus évoluées que celles des protocoles qualifiables de "monolithiques", ainsi que des connaissances du réseau (généralement par monitorage de celui-ci) et des applications.

Ce type de solutions offre des performances qui peuvent être comparable à celles des solutions spécifiques lorsque les composants sont correctement choisis en fonction du contexte. En même temps, elles sont conçues sous forme d'un protocole qui possède les avantages des solutions générales (déploiement, interface unique, ...). Cependant, si théoriquement ces solutions paraissent idéales, en pratique elle ne sont jamais déployées à l'instar des solutions spécifiques. Ceci est dû à un ensemble de limites dont elles souffrent, liées notamment à la difficulté de les faire évoluer et au coût résultant des traitements nécessaires pour gérer leur adaptabilité. Le travail que nous présentons

dans cette thèse entre, en partie, dans le cadre de ces architectures adaptatives, mais à un plus haut niveau (celui de la couche Transport) et avec des capacités d'évolution sans surcoût pour la gestion des fonctionnalités supplémentaires. Nous présenterons des exemples de ces solutions ainsi que leurs outils et leurs limites dans la suite de ce chapitre.

1.1.2 Solutions pour les applications et solutions pour les réseaux

Depuis l'apparition des réseaux locaux à haut débit à la fin des années 80, les solutions de base développées au niveau Transport de l'Internet (TCP notamment) sont rapidement devenues insuffisantes et inadaptées aux nouvelles technologies [16]. Cette difficulté d'adaptation résulte de l'ensemble des hypothèses sur lesquelles une solution donnée se base pour détecter les événements et effectuer les traitements associés. Ces hypothèses dépendent de la nature des flux applicatifs et des caractéristiques des réseaux sous-jacents, et il est très difficile d'aboutir à une caractérisation commune tenant compte de l'ensemble des aspects [17][18]. Plusieurs études ont été menées afin de mesurer l'efficacité des protocoles de Transport dans ces nouveaux environnements et en quelques années, des dizaines de solutions ont déjà été proposées comme alternatives à TCP dans les environnements où il n'était plus adapté. [19][20][21] sont parmi les tous premiers travaux en rapport avec les réseaux locaux, optiques et satellites. D'autres travaux ont suivi depuis l'époque jusqu'à nos jours. Parmi les derniers travaux, on peut citer [22][23][24], en rapport avec les nouvelles tendances concernant l'Internet des objets, le Cloud et l'Internet mobile. De nombreux autres exemples peuvent être trouvés dans la littérature.

De la même façon, les besoins classiques des applications utilisant ces protocoles, qui se résumaient initialement à un transfert de donnée fiable (i.e. sans perte ni erreur) et ordonné (i.e. sans déséquencement), ont évolué avec l'apparition d'applications multimédias, distribuées en temps réel, et plus généralement avec l'émergence de systèmes distribués critiques, notamment vis-à-vis du respect de contraintes temporelles. De nouvelles solutions de Transport sont aussitôt apparues pour offrir des services plus adaptés à ces nouveaux besoins. [25] fut l'un des premiers travaux à proposer un protocole de Transport pour la diffusion de télévision en haute définition. Plus récemment, de nouveaux travaux ont proposé des versions adaptées aux trafic http dans les centre de données (data centers) [26] et sur la qualité de service pour les flux multimédias temps-réel [27].

Enfin, il est possible que ces nouvelles solutions prennent en compte à la fois des types d'applications spécifiques et des contextes réseaux particuliers. Ceci vient en général en réponse à des dysfonctionnements qui peuvent se produire dans des solutions traitant uniquement d'un seul aspect (applicatif ou réseau). On trouve par exemple des versions de TCP pour les applications multimédia dans les réseaux sans fil, comme dans [28]. Un autre exemple est une version de TCP qui a été proposée dans [29] pour les applications interactives dans les réseaux satellites. Ici encore, plus la solution prend en compte davantage d'aspects et plus ses performances sont meilleures.

1.1.3 Nouveaux protocoles et nouvelles versions

Les propositions n'aboutissent pas toutes à de nouveaux protocoles. En fait, une grande partie des travaux sur le Transport ont conduit soit à des versions des protocoles existants incluant de nouveaux mécanismes, soit à la proposition de nouvelles versions des mécanismes existants. Les nouveaux protocoles ont notamment marqué les premiers travaux durant la fin des années 1980 et la première partie des années 1990. Plusieurs protocoles, spécifiques notamment, ont été proposés comme LNTP [30] pour les réseaux locaux à haut débit, et MHTP [31] pour les applications multimédias. Depuis, les propositions de nouveaux protocoles sont de moins en moins répandues, probablement en raison du large déploiement du protocole TCP et de la non évolution de la couche Transport. Il en existe tout de même de très récents, tels que MPMTP pour les applications multimédia multi-paths [32], ou encore OverUDP pour les réseaux pair-à-pair [33].

En revanche, pour les solutions générales et hybrides, la plupart des solutions ont été proposées sous la formes de nouveaux protocoles, bien qu'il ne s'agisse, pour quelques-unes, que de simples ajouts ou de combinaisons de protocoles existants, tels que MPTCP qui n'est qu'une combinaison de flux TCP. La Figure 1.1 ci-dessous illustre une classification des solutions de Transport.

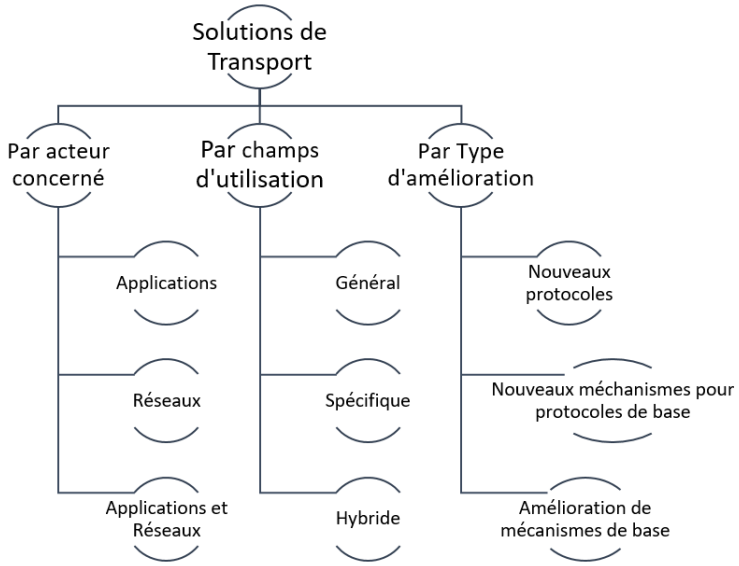

FIGURE 1.1 – Classification des solutions de Transport

1.1.4 Analyse comparative

Dans un contexte aussi hétérogène que celui des réseaux de communication, il est difficile d'établir des critères de comparaison pour définir de façon universelle ce que serait la meilleure solution de Transport. Même dans des contextes bien précis, le choix d'une solution reste difficile. Bien que les performances et le déploiement soient les critères les plus couramment considérés, beaucoup d'autres viennent s'ajouter à l'équation en fonction du domaine d'application.

Globalement, les solutions générales présentent une grande facilité de déploiement

grâce à leur adéquation à la plupart des environnements, mais au prix de performance faibles. Les solutions spécifiques présentent de meilleures performances mais avec un champ d'adaptation très restreint.

Ceci étant, le champ d'application restreint des solutions spécifiques les rend plus facile à maintenir. Cela vient du fait que l'on connaît exactement le contexte où elles seront déployées. D'un autre côté, ce type de solutions est plus difficile à universaliser, non seulement du fait de la spécificité de leur contexte de déploiement, mais aussi à cause de la difficulté de leur standardisation vu leur nombre et leur durée de vie souvent courte.

Les solutions générales sont plus faciles à standardiser compte tenu de leur nombre limité. Pour leur maintenance, la question est plus compliquée. En effet, leur large déploiement rend difficile et longue l'intégration d'une nouvelle solution ou le passage d'une version à une autre ; plusieurs années sont souvent nécessaires. Ceci est dû au large champ de leur déploiement et des dépendances architecturales qui caractérisent la couche Transport, que ce soit avec les systèmes ou les applications.

Les solutions adaptatives semblent donc idéales à première vue. Elles sont pourtant loin d'être privilégiées. Tel qu'explique ci-après, ceci est dû, en partie, au degré de maturité de ces solutions, notamment les concepts associés, et l'aspect restreint des communautés les proposant et les soutenant. En effet, les approches de conception de ces solutions, telles que la décomposition des services et leur mode d'invocation, sont spécifiques à ces architectures et ne sont pas universellement utilisées. Aussi, les versions sur lesquelles les tests de validation ont été effectués, sont généralement des versions de test développées par des particuliers et ne sont pas soutenues par la communauté à la façon des protocoles standardisés. Par ailleurs, ces solutions ont été proposées comme des protocoles à part entière, prévus pour être déployés au même niveau que les protocoles existants selon le mode architectural de la couche Transport actuelle. Ceci les confronte par conséquent à l'ensemble des limites de déploiement de la couche Transport qui n'a pas été initialement prévue pour évoluer.

Les solutions déployées commencent à devenir problématiques lorsque le service qu'elles offrent n'est plus adapté aux applications l'utilisant ou que les mécanismes associés ne sont pas adaptés aux réseaux physiques sous-jacents. Dans la plupart des cas, ce manque d'adéquation survient lorsque l'application est plus exigeante par rapport à la solution ou que le réseau sous-jacent est plus dégradé par rapport à ce qui a été prévu dans la solution. Sur ce point, les solutions générales sont plus avantageuses du fait qu'elles nécessitent moins d'hypothèses sur la nature des réseaux et qu'elle couvre les besoins les plus communs des applications.

Cependant, il se peut qu'une application moins exigeante se voit au final offrir un service plus dégradé que celui offert à une application plus exigeante. Cela est dû au surplus de service, en termes de données ou de procédures, qui nuit à des besoins plus nécessaires. Un exemple typique concerne les applications multimédia, notamment interactives telles que la vidéoconférence, pour lesquelles un service de Transport fiable peut nuire considérablement au délai de transmission qui se trouve être plus primordial. De la même façon, les performances de certaines solutions peuvent être moins optimales sur des réseaux présentant moins de contraintes. Ceci provient du surcoût de calcul ou de transmission relatif à une gestion, inutile, de certaines hypothèses pré-

Critère	Spécifique	Générale	Hybride
Performances	Elevées	Moyennes/Faibles	Elevées
Déploiement	Difficile	Facile	Moyen
Maintenance	Moyen	Difficile	Moyen

TABLE 1.1 – Analyse comparative selon la spécificité

Critère	Applications	Réseaux	Applis et Réseaux
Performances	Moyennes	Moyennes	Elevées
Déploiement	Difficile	Difficile	Moyen
Maintenance	Moyen	Moyen	Difficile

TABLE 1.2 – Analyse comparative selon le domaine d'application

vues dans des contextes moins favorables.

Pour conclure, quel que soit l'évolution des réseaux et des applications, que ceux-ci présentent plus ou moins de contraintes, que celles-ci requièrent plus ou moins de services, les solutions de Transport seront toujours amenées à évoluer pour s'y adapter. Il n'est donc pas possible de compter sur la nature des réseaux et des applications futures pour garantir des solutions de Transport stables et durables. Une couche de Transport évolutive devient dès lors incontournable ; elle constitue à ce titre l'objectif de notre travail. Les deux tableaux suivants résument une première analyse comparative des différentes approches.

1.2 Limites de la couche Transport

Tel que mentionné précédemment, le problème au niveau de la couche Transport n'est pas dans la disponibilité de solutions efficaces mais plutôt dans leur (faible) déploiement. Le problème du déploiement est dû à plusieurs limites architecturales de la couche Transport telle qu'elle a été conçue initialement. Nous présentons dans cette section un état de l'art sur les limites qui ont été soulevées dans les différents travaux sur les architectures des protocoles de Transport face au déploiement des nouvelles solutions. Nous présenterons une étude plus approfondie sur ces limites au chapitre II. Quatre obstacles ont été identifiés : la configurabilité, l'adaptabilité, la flexibilité et la complexité [34].

1.2.1 Configurabilité

La configurabilité est la première limite soulevée dans les travaux sur le Transport. Les premiers protocoles ont été conçus de sorte à offrir un certain nombre de services de base liés aux besoins initiaux, sous forme d'un ensemble non séparable (protocoles qualifiables de "monolithiques"). L'utilisateur se voit donc offrir l'ensemble de ces services sans avoir la possibilité d'en choisir un sous ensemble. Cette politique n'a pas posé de problème jusqu'à la venue de nouvelles applications ayant d'autres besoins qu'un transfert (fiable ou non fiable) de leurs données. Pour ces nouveaux besoins

comme le délai ou la gigue, une telle politique ne peut être adaptée, dans la mesure où des services non nécessaires pourraient nuire à des services plus importants. A titre d'exemple, pour des flux vidéo où une fiabilité partielle est suffisante et un délai très court est nécessaire, un service offrant une fiabilité totale nuira considérablement au délai à cause du coût temporel des retransmissions.

Dans ADAPTIVE [35][36], le manque de configurabilité est décrit comme un obstacle du fait de l'important surplus de données ou de calculs qu'il engendre, notamment dans la gestion de l'établissement de connexion dans le modèle requête/réponse. Dans CTP [37], le problème de configurabilité est décrit comme le fait d'appliquer le même traitement à l'ensemble des paquets et des flux d'un protocole. Dans Coyot [38] et Appia [39][40], la configurabilité consiste à instancier des services personnalisés spécifiques à chaque type d'applications.

Quel que soit l'angle sous lequel est traité le problème de configurabilité, l'idée de base est d'avoir un ensemble de services instanciables à la demande dans un même protocole, et ce, au lieu d'avoir un protocole spécifique à chaque type d'application ou de réseau par la réutilisation des services, et au lieu d'obliger l'application à utiliser des services non nécessaires au prix d'une diminution de performances.

1.2.2 Adaptabilité

Telle que conçue actuellement, la couche Transport fonctionne de manière statique en ceci que chaque protocole adopte le même comportement quel que soit l'application qui l'utilise et la nature du réseau sous-jacent. Ceci est probablement dû au fait que le nombre de solutions est tellement limité qu'un quelconque besoin d'adaptabilité ne puisse paraître nécessaire. Cependant, cette limite apparaît quand on prévoit un nombre important de solutions, ce qui est en partie l'objectif de cette étude. En effet, l'évolution des besoins des applications et des technologies réseaux remettent en cause l'efficacité des solutions de base existantes, cela suscite le besoin de déployer de nouvelles solutions, chacune adaptée à un contexte particulier. L'aspect statique de la couche Transport fait que le choix d'une solution (protocole) en tenant compte des aspects applicatifs et réseaux (performance, évolutions du contexte ...) est entièrement à la charge du développeur d'application ; la couche Transport elle-même n'est alors qu'une simple interface pour accéder aux services des différents protocoles.

L'application est donc tenue de choisir une solution particulière pour l'ensemble de ses communications et il ne lui est pas possible, notamment pour les protocoles applicatifs, de basculer d'une solution à une autre selon le contexte réseau.

ADAPTIVE est l'une des premières solutions à avoir posé ce problème, mais il était simplement question d'une flexibilité dans l'intégration (manuelle) de composants logiciels offrant des services plus adaptés que ceux des services de base. Plus récemment, l'architecture ATP [41] décrit ce problème de façon plus avancée en considérant la capacité d'une architecture à effectuer cette adaptabilité de façon autonome (sans intervention humaine).

1.2.3 Flexibilité

Aussi décrite comme une forme d'extensibilité interne, la flexibilité d'une architecture fait référence à son degré de maintenabilité sans avoir à en modifier le code source, au moins dans son intégralité. Etant donné la durée de vie relativement courte des solutions de Transport, compte tenu de l'évolution rapide des applications et des réseaux, les solutions protocolaires sont amenées à être régulièrement mises à jour. Cela peut devenir très problématique lorsque les solutions en question sont conçues de façon non modulaires, contenues dans un même fichier source, et surtout intégrées au noyau. Il devient ainsi très coûteux de déployer une nouvelle solution avec toutes les manipulations du noyau associées, et ce pour une durée relativement courte.

ADAPTIVE décrit ce problème par la difficulté à remplacer ou à mettre à jour un des composants d'une architecture à cause de l'effort et de l'expertise nécessaire pour toucher à ces architectures sans y introduire des bugs. Ceci force les applications à utiliser des solutions obsolètes bien que d'autres plus performantes sont proposées. Dans Coyote, l'extensibilité est décrite comme la capacité à ajouter autant que nécessaire de nouveaux composants à l'architecture. Pour CTP, l'extensibilité consiste en l'indépendance entre les différents composants, qui fait que l'intégration d'un nouveau composant n'induit pas à une modification des composants existants.

De manière générale, le besoin réside dans la capacité d'un système à intégrer de nouveaux composants avec un minimum de modifications sur les éléments existants.

1.2.4 Complexité

Par complexité, nous entendons ici la difficulté pour un développeur d'application utilisant directement la couche Transport de choisir la solution la plus adaptée lorsque celles-ci deviennent nombreuses. L'évolution de la couche Transport se base principalement, en termes de travaux de recherche, sur des solutions spécifiques en proposant une solution adaptée à chaque contexte particulier. Avec l'évolution rapide des applications et des réseaux, le nombre de ces solutions se compte en plusieurs centaines et il n'est pas possible à un développeur d'avoir la maîtrise de l'ensemble des solutions existantes.

Cette problématique a été posée par les premiers travaux comme Cactus et Appia, qui décrivent la difficulté pour un développeur face à la maîtrise de l'ensemble des aspects techniques de chacune des solutions existantes afin de pouvoir en choisir la plus adaptée. La difficulté de prendre connaissance de l'existence de ces solutions a également été posée. Par la suite, cette idée a été reprise par l'IETF et un groupe de travail dédié a été créé, le groupe TAPS (TrAnsPort Services) [42]. Il traite de la difficulté résultant de l'utilisation de protocoles indépendants pour offrir les différents services de Transport. Cette difficulté ne se résume pas à l'utilisation effective de l'interface de programmation, mais aussi aux obstacles de déploiement que peuvent rencontrer ces protocoles indépendants, que ce soit dans les systèmes ou dans les réseaux.

1.3 Évolutions architecturales au niveau Transport

En réponse aux limites présentées dans la section précédente, certains travaux ont proposé de nouvelles approches de conception de l'architecture des protocoles de Transport, visant à faciliter l'intégration de nouveaux services et mécanismes. Elles ont suivi l'approche hybride présentée en amont, combinant l'efficacité des solutions spécifiques adoptant un comportement adapté à chaque environnement, et la facilité de déploiement en ne présentant qu'une seule interface aux applications, et en évitant d'encombrer les systèmes d'exploitation par une multitude de solutions correspondant chacune à un contexte particulier.

Nous décrivons ci-après les principales solutions proposées. Compte tenu des différents aspects qu'elles ont abordés, nous mettons l'accent sur l'aspect architecture en réponse aux limites de la couche Transport citées précédemment. D'autres aspects techniques concernant la mise en oeuuvre de ces nouvelles architectures, telles que la gestion des compositions, sont communs et font l'objet d'un état de l'art spécifique (section 1.3.2). Nous présentons donc dans cette section les nouvelles architectures de Transport ainsi que les éléments conceptuels s'y rapportant.

1.3.1 Nouvelles architectures

Cette section présente les différentes architectures proposées au niveau de la couche Transport pour faciliter son évolution et en améliorer les performances. Une de leurs caractéristiques communes est la composition dynamique d'un service de Transport à base de composants élémentaires offrant chacun un service ou une fonctionnalité d'un service (micro-service). Ce principe de décomposition / composition est apparu dans plusieurs architectures depuis le début des années 90 [41]. Il ne se résume cependant pas aux architectures des protocoles de Transport et concerne également d'autres architectures plus générales comme x-Kernel [43], V-Stream [44] ou autres. Dans cette partie, nous traitons uniquement des architectures de protocoles de Transport, et nous ferons référence, si besoin, aux autres architectures quand nous discuterons notamment des aspects techniques et conceptuels.

ADAPTIVE

ADAPTIVE, A Dynamically Assembled Protocol Transformation, Integration, and eValuation Environment [35][36], est l'une des premières solutions qui s'est intéressée aux problèmes architecturaux de la couche Transport, en proposant un framework pour la composition dynamique de protocoles de Transport. Il part du fait que l'ensemble des protocoles qui existent actuellement (à cette époque) est très limité en termes de services et de performances. Les solutions existantes ne peuvent pas prendre en compte les évolutions des applications et des technologies réseaux. Ainsi, un ensemble aussi limité de services ne pourra jamais couvrir les besoins variés des applications, notamment les application multimédias et interactives. Enfin, le modèle de conception de ces protocoles, dépourvu de toute flexibilité, empêche de les faire évoluer. L'architecture ADAPTIVE est présentée dans la figure 1.2.

Les concepteurs d'ADAPTIVE pointent donc comme limite principale le manque de flexibilité des solutions existantes qui fait que leur maintenance est fortement coûteuse

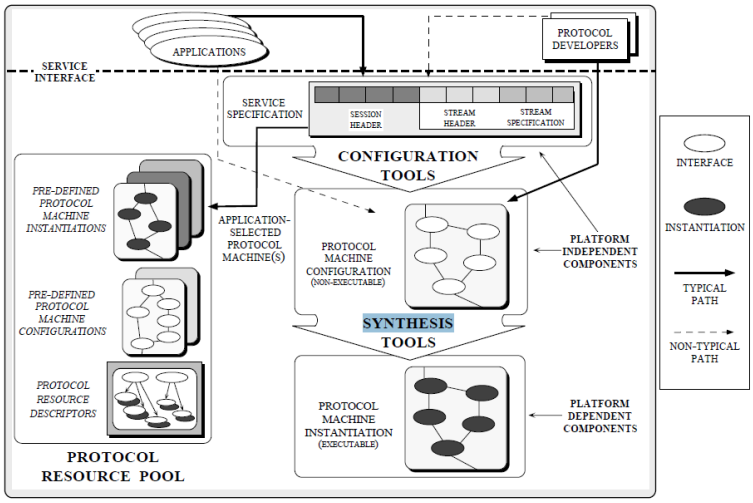

FIGURE 1.2 – Architecture globale de ADAPTIVE

notamment compte tenu de leur durée de vie relativement courte. Ils proposent comme solution une architecture plus souple qui facilite l'intégration de nouvelles fonctionnalités sans avoir à affecter les composants existants. Par ailleurs, ils traitent l'adaptabilité des services aux besoins spécifiques des différentes applications, et partent du principe qu'un protocole indépendant dédié à chaque situation est une approche irréalisable. La solution qu'ils proposent repose sur le fait que la spécificité des besoins ou des contextes réseaux se limite la plupart du temps à un seul élément (service ou fonction protocolaire) et le reste est partagé. Ils proposent par conséquent une solution basée sur la notion de micro-protocoles, reprise de xKernel et l'assemblage dynamique selon les besoins de l'application. Les problèmes liés aux performances sont traités par une instanciation des éléments uniquement nécessaires aux besoins de l'application, réduisant de cette façon le surplus de données et de calcul dû aux services non nécessaires. Enfin l'adaptabilité est traitée par cette instanciation dynamique de services, mais reste statique et à la charge de l'application. Par ailleurs, ADAPTIVE offre des outils avancés pour l'évaluation des performances afin d'aider l'utilisateur à tester facilement sa configuration.

CTP

CTP, pour Configurable and extensible Transport Protocol [37], est une réalisation concrète d'un protocole de niveau Transport traitant principalement du problème de performances des protocoles existants dû à une mal adaptation des protocoles existants aux nouveaux besoins des applications et un surcoût des services non nécessaires. Le schéma global de l'architecture est présenté sur la figure 1.3.

CTP adresse donc principalement le problème de configurabilité et part du principe qu'une combinaison de solutions basiques pour construire un pseudo-protocole pour chaque situation est bien plus efficace que la construction d'un protocole complet dédié à chaque situation. Il reprend la notion de micro-protocole à l'instar des deux architectures précédentes (ADAPTIVE et xKernel). Cependant, CTP se propose comme un protocole final et non pas comme un environnement de déploiement. Il propose un certain nombre de fonctionnalités de base (fiabilité, contrôle de flux, ...) avec différentes

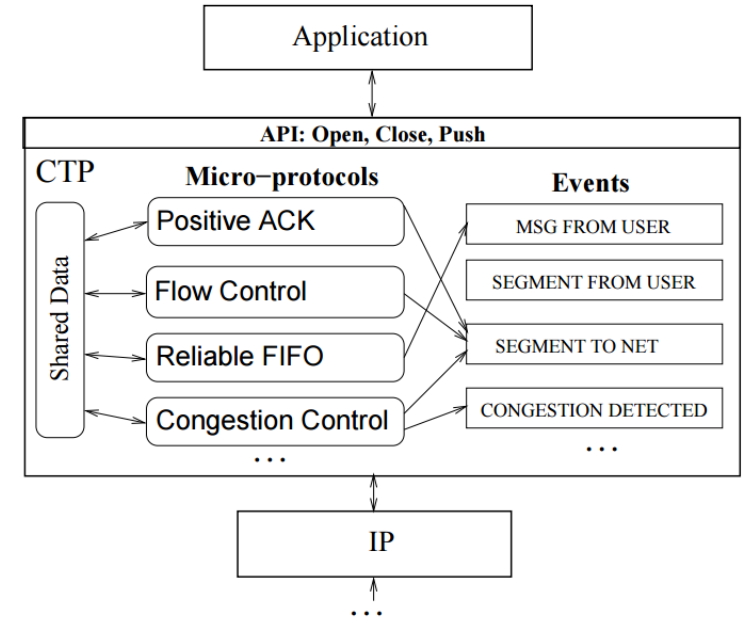

FIGURE 1.3 – Architecture globale de CTP

variantes pour chacune d'elles. Le choix d'une variante plutôt qu'une autre est laissé à la charge de l'application. Le choix se fait par des tests sur différentes combinaisons des différentes versions disponibles pour chaque service souhaité.

L'extensibilité de CTP se concentre sur l'intégration de nouvelles versions des fonctionnalités existantes, qui est rendu relativement facile. L'intégration d'une toute nouvelle fonctionnalité, quant à elle, est plus difficile.

Appia

Appia [39][40] est une autre architecture contenant le noyau d'un protocole configurable qui permet l'instanciation de différents services selon les besoins des applications. En termes d'architecture, Appia reprend les mêmes principes que ses prédécesseurs, mais ajoute la notion de multi-channel, avec laquelle une seule application peut avoir différentes instances de service en même temps, mais avec une coordination entre elles. L'architecture d'APPIA est présentée sur la figure 1.4.

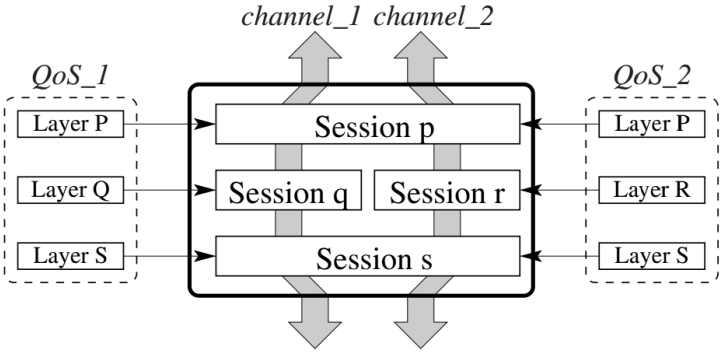

FIGURE 1.4 – Architecture globale de APPIA

Plutôt que d'avoir différentes connexions indépendantes pour une application, Appia permet de les coordonner en une seule connexion avec la notion de canal de communications.

ETP et ATP

ETP, Enhanced Transport Protocol [45], est un protocole configurable orienté QoS. Son apport principal se situe au niveau technique en apportant de nouvelles approches dans la gestion des architectures (re)configurables, notamment sur l'invocation des micro-protocoles en combinant les avantages des approches précédentes. Par ailleurs, ETP offre une interface permettant aux applications de définir de façon précise leurs besoins en termes de QoS ; il revient au protocole de choisir les composants adaptés au service en question. L'architecture globale de ETP est présentée sur la figure 1.5.

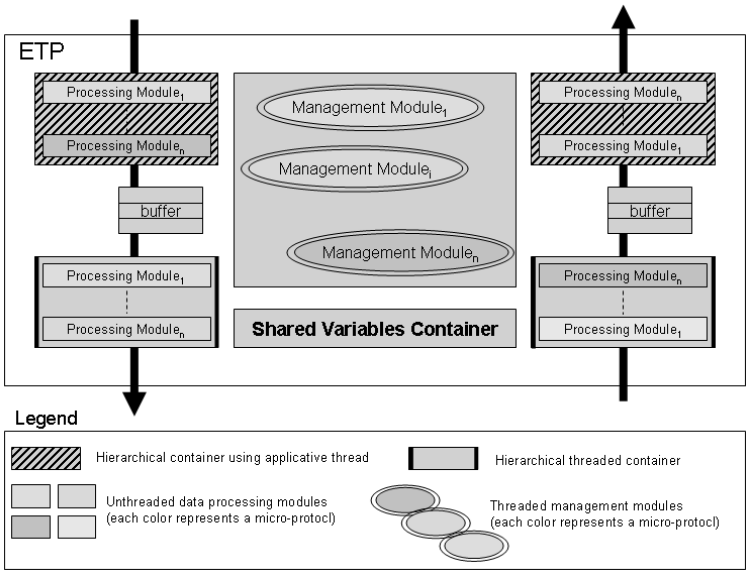

FIGURE 1.5 – Architecture globale de CTP

ATP, Autonomic Transport Protocol [41], étend l'architecture d'ETP en lui intégrant une couche d'adaptabilité aux caractéristiques du/des réseau/x sous-jacents. L'aspect autonomique de l'architecture d'ATP permet de doter le protocole de nouvelles capacités pour guider son adaptation structurelle ou comportementale avec un minimum d'intervention externe. Nous présenterons dans la suite les principes du paradigme de l'autonomic computing, de ses éléments de bases, ainsi que de son intérêt pour le Transport. La figure 1.6 présente l'architecture ATP.

ATL

ATL, pour Autonomic Transport Layer, est un modèle d'architecture pour une nouvelle couche introduit dans le cadre de la thèse de Doctorat de Guillaume Dugue [46]. L'idée générale du modèle est de conserver les protocoles de base et de les combiner avec des mécanismes protocolaires, notamment orienté services, suivant le principe de "micro-protocoles". Les compositions contiennent donc des protocoles entiers et des micro-protocoles.

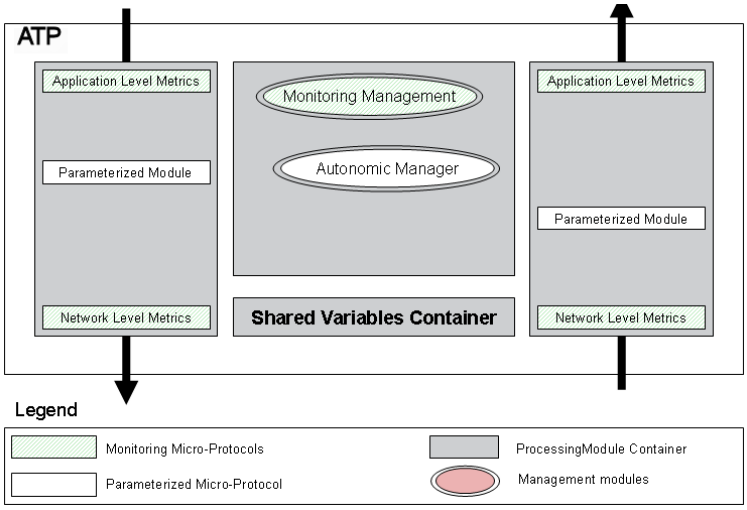

FIGURE 1.6 – Architecture globale de CTP

Un aspect de l'architecture est son interface orientée service, qui permet d'abstraire les détails des compositions aux applications, qui n'ont donc à invoquer que des services. En même temps, pour des applications souhaitant intervenir dans les détails des compositions, il est possible de choisir explicitement les différents composants. L'architecture tient aussi compte des aspects réseaux pour adapter la sélection aux caractéristiques réseau par un processus de monitoring.

Le travail publié sur l'ATL se limite en grande partie aux aspects conceptuels. Dans nos travaux, nous reprenons une partie des principes de l'ATL, notamment l'approche d'une couche de Transport plutôt qu'un protocole, et l'utilisation d'une interface orientée service pour les applications. Nous étendons les travaux initiaux sur le plan conceptuel, notamment en intégrant des capacités d'extensibilité via plusieurs paradigmes (présentés au chapitre 3) et en considérant l'ensemble des limites de la couche actuelle. Sur le plan technique, nous présentons également une implémentation complète de l'architecture avec des tests de validation.

TAPS

Depuis 2014, un groupe de travail de l'IETF, le groupe TAPS [42] (pour TrAnsPort Services), a été créé dans le but de définir clairement l'ensemble des services offerts par les protocoles de Transport, afin de proposer une couche abstraite pour les applications lors de leur utilisation de la couche Transport. TAPS part de la problématique de déploiement et d'accessibilité des nouveaux protocoles de Transport au niveau des systèmes et des fournisseurs d'accès. Ceci a pour conséquence qu'une application n'utilisant pas le protocole TCP risque de ne pas fonctionner universellement. TAPS tente donc d'établir une liste de services de Transport que l'application devra invoquer à la place des protocoles eux-mêmes. Ainsi, le système s'occupe du choix optimal du protocole approprié pour l'application selon les besoins requis, mais aussi selon les capacités du réseau.

Le projet est encore en cours. Au stade actuel, une RFC (8095) informationnelle a été publiée regroupant l'ensemble des services de la couche Transport, protocole par protocole. Une autre RFC, en cours de préparation, traite des primitives de services offertes

par les différents protocoles de Transport.

1.3.2 Outils conceptuels pour les architectures configurables

Malgré le caractère indépendant des travaux menés dans la conception des différentes architectures pour la couche Transport, certains éléments communs se dégagent comme base de référence pour le développement d'architectures configurables. Ces éléments concernent la gestion des composants internes, notamment les micro-protocoles, le principe de leur décomposition, ainsi que l'approche de leur sélection.

Nous présentons dans cette section ces différents éléments, et la manière avec laquelle les différentes architectures les ont traités.

Décomposition en micro-protocoles

La plupart des architectures composables pour la couche Transport se basent sur la notion de micro-protocoles. Cette notion a d'abord été utilisée par xKernel, et reprise ensuite par les autres solutions. Il s'agit de décomposer un protocole en un ensemble de composants élémentaires assurant chacun une de ses fonctions.

La granularité de cette décomposition diffère d'une architecture à une autre, et plus elle est fine, plus la configurabilité du protocole est avancée. Ceci étant, un niveau très fin augmente la complexité de l'architecture et par conséquent son utilisation par les applications. De façon générale, cette décomposition se situe au-dessous des services connus de la couche Transport. Suivant cela, un service de fiabilité sera (notamment) composé d'une fonction de contrôle d'intégrité et de reprise des pertes. Quand la granularité est plus fine, il est possible de séparer différentes fonctions, le calcul du RTT, la détection des pertes ... La figure 1.7 illustre un exemple d'une simple décomposition en micro-protocoles de TCP.

Invocation des composants

Dans toute architecture, en réponse à une demande de l'application, une composition de micro-protocoles est créée de façon à répondre au mieux les besoins de l'application. Une fois instanciés, les différents composants (micro-protocoles) sont invoqués selon une certaine politique afin d'effectuer les traitements associés.

Le modèle d'invocation des micro-protocoles diffère d'une architecture à une autre, mais on distingue deux grandes catégories : les modèles hiérarchiques dans lesquels les composants sont appelés systématiquement de façon séquentielle à chaque émission ou réception de données, et les modèles événementiels dans lesquels les composants sont invoqués en réponse à l'occurrence de certains évènements particuliers.

Modèles événementiels
Dans les modèles événementiels, plusieurs événements sont définis et repérés par le noyau de l'architecture. A ces événements sont associés un ou plusieurs micro-protocoles qui sont appelés à chaque fois que l'événement en question se produit. Les micro-protocoles peuvent eux-aussi générer des évènements pour qu'ils soient traités par d'autres micro-protocoles.

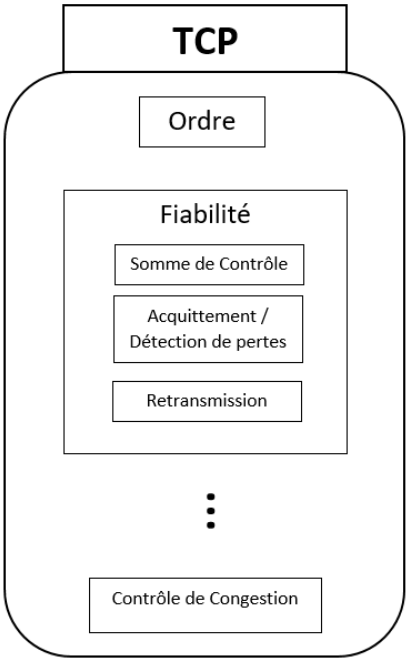

FIGURE 1.7 – Exemple de décomposition de TCP en micro-protocoles

Ce modèle a été proposé par x-Kernel et a été étendu par Coyot en ajoutant plus de granularité aux micro-protocoles. Il définit deux types d'événements : les événements systèmes qui sont prédéfinis et repérés par le noyau de l'architecture, et les évènements utilisateurs qui sont repérés uniquement par les micro-protocoles. Le service de Transport est donc offert par l'ensemble des fonctions réalisées par les différents micro-protocoles abonnés aux différents évènements.

La figure 1.8 illustre le schéma d'invocation des composants en mode événementiel.

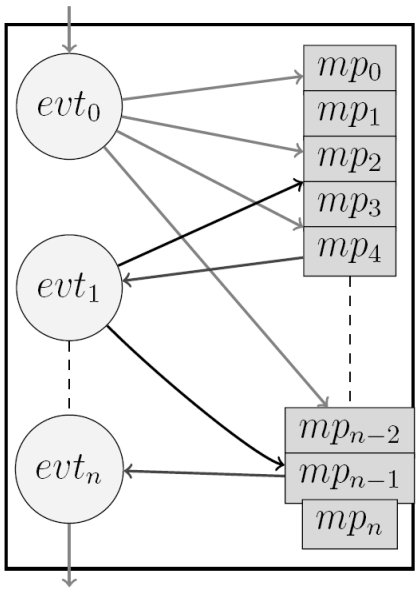

FIGURE 1.8 – Modèle événementiel

Cactus a étendu le modèle de Coyot en y ajoutant la possibilité de redéfinir les liens entre les micro-protocoles et les évènements. Ceci a été conçu dans le but de convenir aux applications présentant des besoins spécifiques en QoS.

Modèles hiérarchiques
Les modèles hiérarchiques ont été proposés en parallèle avec les précédents. Leur principe est de passer le message (les données applicatives) de façon séquentielle par l'ensemble des micro-protocoles instanciés dans une composition donnée.

V-Stream est l'un des premiers systèmes à avoir utilisé ce modèle pour gérer les entrées-sorties. Un stream est composé d'un ensemble de processing modules, et un message est traité de façon séquentielle par chacun de ces composants l'un après l'autre. Le modèle permet par ailleurs d'ajouter des composants au cours de l'exécution.

Par la suite, x-Kernel a été créé comme un système d'exploitation orienté réseau, permettant à l'utilisateur de créer son propre protocole de communication. L'utilisateur peut y déployer une pile de différents protocoles qui, en les regroupant ensemble, offrent le service requis par l'application. Bien qu'il n'ait pas été conçu comme une plateforme de composition, x-Kernel a été à la base de plusieurs expérimentations de Coyot et Cactus.

La figure 1.9 illustre le schéma d'invocation des composants en mode hiérarchique.

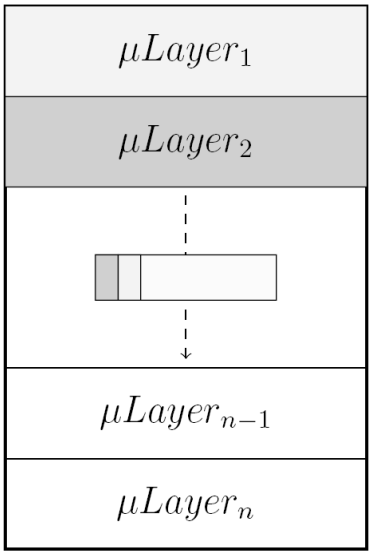

FIGURE 1.9 – Modèle événementiel

Modèle hybride
Initié par ETP et repris par ATP, ce modèle combine les deux approches précédentes. Les auteurs démontrent qu'en associant chaque micro-protocole au type d'invocation qui lui est adapté, il est possible d'obtenir de meilleures performances. Par exemple, les mécanismes de contrôle de congestion sont invoqués de façon événementielle lorsqu'une congestion se produit, alors que les mécanismes de calcul des sommes de contrôle sont appelés de façon hiérarchique.

1.3.3 Autonomie

Les nouvelles architectures qui ont été proposées au niveau Transport ont ajouté de nouveaux paradigmes et de nouvelles techniques pour faire évoluer les services de la couche Transport. L'objectif principal de ces architectures était d'avoir des comportements adaptés aux différents contextes applicatif et réseau plutôt que de fonctionner de manière statique, ceci pour tenir compte de l'aspect variable et hétérogène qui caractérise les réseaux et les applications.

Cependant, la gestion de cette adaptabilité se fait de façon statique durant le développement des architectures, et elle n'est pas projetée sur les évolutions futures des applications et des réseaux. Ceci impliquerait, comme pour les protocoles classiques, une mise à jour continue à mesure que les contextes évoluent.

L'architecture ATP, décrite en section 1.3.4 est la seule architecture, à notre connaissance, à avoir introduit le paradigme de l'Autonomic Computing pour résoudre le problème d'adaptabilité aux nouveaux contextes. Elle permet une adaptation dynamique et autonome des compositions en fonction des performances du réseau en se basant sur une surveillance des caractéristiques de la communication.

L'autonomic computing est un modèle de conception pour les architectures qui sont amenées à fonctionner dans des environnements hétérogènes, mais surtout évolutifs [47]. Ces architectures doivent faire face à des situations qui n'ont pas été connues au moment de leur conception et s'y adapter de façon autonome, sans intervention externe.

Les architectures autonomes sont basées généralement sur un surveillance (monitoring en anglais) du contexte dans lequel elles sont déployées afin d'en établir des profils suivant lesquels un comportement spécifique est adopté. Les profils en question et les comportements associés peuvent être définis de façon statique au moment de la conception de l'architecture. Toutefois, cette approche ne peut présenter d'intérêt que dans un cadre très restreint où le nombre des situations est, d'une part, connu et est facilement dénombrable, et d'autre part statique. En revanche, une architecture qui sera déployée dans un contexte complexe et fortement évolutif, comme dans le cas de l'Internet, doit pouvoir identifier des profils de façon dynamique et adapter son comportement a des situations non définies à priori [48].

Structuration du modèle

Un processus autonomic comporte basiquement quatre phases (illustrée sur la figure 1.10) : le monitoring, l'analyse, la décision et l'exécution [47]. La phase de monitoring permet de surveiller le réseau afin d'établir un profil d'une part, et d'autre part détecter des symptômes qui pourraient donner lieu à un éventuel changement de comportement. Les profils servent de base de comparaison aux autres fonctions pour analyser le contexte sous-jacent ou décider du comportement qui leur est approprié. Une fois un symptôme détecté, le profil établi par la fonction de monitoring est analysé afin d'en étudier les causes dans le but d'établir un diagnostic qui permettra de trouver la solution appropriée.

Lorsque c'est nécessaire, la phase de décision est alors invoquée afin de trouver un

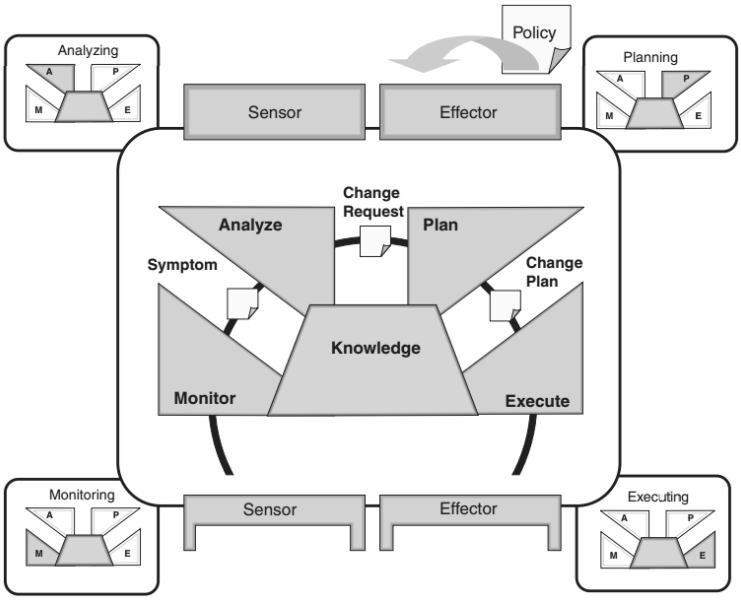

FIGURE 1.10 – Modèle de l'Autonomic Computing

comportement plus adapté que celui en cours, en tenant compte du profil du réseau établi lors de la phase de monitoring et de la base de données contenant l'ensemble des solutions disponibles dans l'architecture (ou les moyens de la déterminer ses solutions). Ces solutions sont implantées sous la forme d'un ensemble de composants, appelés dans la terminologie du paradigme composants autonomes, qui peuvent être assemblés entre eux pour constituer une seule solution. L'approche suivant laquelle la décision est opérée diffère d'une architecture à une autre. Plusieurs modèles existent et relèvent en particulier de la théorie de la décision. Nous en parlerons plus en détails au chapitre 4 de ce mémoire.

Dans le cas où une meilleure solution a été trouvée, il appartient à la phase d'exécution de mettre en place la nouvelle solution adoptée. Le déploiement de cette solution dépend de l'architecture elle-même. Il peut s'agir d'un simple changement de comportement, d'une reconfiguration ou d'un nouveau déploiement dans le cas d'une architecture basée composants.

Fonctions d'un système autonome

Un système autonome doit pouvoir opérer avec un minimum d'intervention externe (humaine) [49]. Il s'agit pour lui de s'adapter aux changements de contextes, de résoudre les défaillances internes ou externes en rapport avec l'environnement d'exécution, d'optimiser le fonctionnement du système, et enfin de le protéger. Une telle description reste bien entendu très généraliste, et il se peut que les besoins diffèrent d'un système à un autre.

Globalement, l'auto-adaptabilité permet au système de changer son fonctionnement, soit par variation de ses paramètres, par une reconfiguration de ses composants, ou par la création d'une nouvelle composition. Quel qu'en soit le modèle, l'objectif final est que le fonctionnement du système soit le plus adapté au contexte courant. L'auto-

adaptabilité permet donc d'opérer de façon différente selon le contexte d'application.

Une facette concerne la résolution autonome des problèmes. Le système doit détecter lui-même des éventuels dysfonctionnements, d'en trouver la cause, et de leur chercher la solution appropriée. Les éventuels problèmes qui peuvent surgir dans le système doivent donc être identifiés et traités, dans la mesure du possible, sans intervention humaine.

En plus de la gestion des problèmes et des changements de contexte, un système autonome doit également fonctionner de façon optimale. L'optimisation est par conséquent un plus par rapport à l'auto-adaptabilité qui se résume à la possibilité de fonctionner dans des contextes différents. L'optimisation permet d'opérer de façon performante que ce soit entre des contextes différents, ou même pour un contexte particulier. L'optimisation peut être appliquée soit de façon proactive par anticipation, ou de façon réactive.

Enfin, un système autonome doit assurer sa protection, et faire face à d'éventuels menaces internes ou externes. Il peut s'agir d'une variante de la gestion des défaillances en considérant les menaces comme des problèmes, et ce, lorsque la question de sécurité n'est pas très importante. Dans le cas contraire, l'auto-défense peut constituer une fonction à part entière.

Certains auteurs, comme dans [50][51], évoquent d'autres fonctions comme la gestion, l'apprentissage ou la description. Il est tout aussi possible d'ajouter d'autres fonctions en réponse à des besoins spécifiques à chaque système. Il n'y a dans ce sens aucune description claire des fonctions élémentaires ou secondaires, qui dépendent en effet du contexte d'application. Néanmoins, les fonctions d'adaptabilité, de configuration et de d'optimisation restent les plus courantes.

1.4 Le modèle ATN/OSI

Le réseau ATN (Aeronautical Telecommunication Network) regroupe l'ensemble des équipements, protocoles communication, et applications qui permettent la communication dans le cadre de l'aviation civile. Ces communications servent à la gestion des vols notamment. Il est géré et standardisé par l'organisme ICAO (International Civil Aviation Organization).

L'ATN dispose de son propre standard de communication, basé sur le modèle à sept couches de l'OSI. Les entités communicantes du l'ATN implémentent donc l'ensemble des sept couches du modèle suivant le standard 9880 [52]. Cette spécificité est à l'origine d'une grande complexité de la gestion du réseau, notamment lors de l'utilisation d'une partie du réseau de l'Internet. Par conséquent, et depuis quelques années, l'ICAO propose une migration de l'ATN vers la pile protocolaire de l'Internet (IPS pour Internet Protocols Suite).

Nous présentons dans cette section, de façon très globale, le schéma général du réseau ATN, ses protocoles, et des problèmes associés à cette transition vers les standards de l'Internet.

1.4.1 Présentation de l'ATN

Le schéma général d'un réseau ATN est illustré sur la figure 1.11. Il existe principalement deux types de communication : air-ground entre l'avion et la tour de contrôle, et ground-ground entre les tours de contrôle. Un modèle de communications air-air (entre deux avions) est proposé, mais il n'est pas encore standardisée. Les liens ground-ground sont toujours filaires. Entre les tours et les avions en revanche plusieurs liens existent. De base, la communication se fait par radio suivant le protocole VDL L-DACS [53]. Cependant, lorsque l'avion ne se trouve plus dans la zone de couverture, la communication se fait par satellite, suivant le protocole ESA IRIS par exemple. Aussi, lorsque l'avion est au sol, et assez proche de la tour, un autre type de lien, AeroMACS, est utilisé. Les liens terrestres, entre les tours de contrôle peuvent transiter, indirectement, par le réseau Internet.

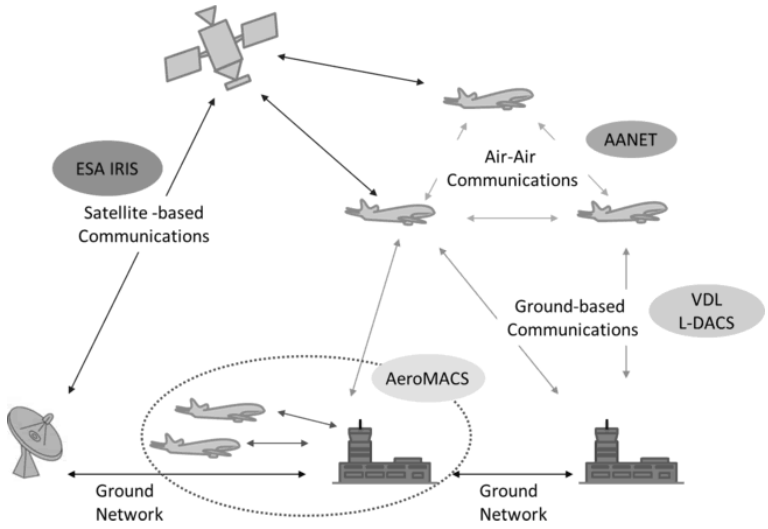

FIGURE 1.11 – Schéma d'un réseau ATN

L'ATN distingue principalement deux catégories de trafic : ATS (pour Air Traffic Service) et AOC (pour Airline Operation Control). Le trafic ATS concerne la gestion du trafic aérien, la météorologie, le partage des positions ... Pour l'AOC, il concerne le contrôle de départ, l'atterrissage, ... et autre contrôle sur les avions. L'ATN achemine aussi d'autres types de trafic comme AAC (pour Aeronautical Administrative Communications) ou autres [54].

1.4.2 La pile de protocoles ATN/OSI

Comme introduit un peu plus haut, le réseau ATN a été définit suivant le standard OSI à sept couches. Distingue les couches hautes (Upper Layer communication service) et les couches basses (ATN Internet communications service), comme illustré à la figure 1.12 [53].

Les applications sont de deux types : Air-Ground applications, définies dans le standard [55], et Ground-Ground applications, définies dans les standards [56][57]. Le standard définit plusieurs applications de la catégorie Air-Ground. Il s'agit principalement de l'application CM (Context Management) pour l'établissement de lien entre deux en-

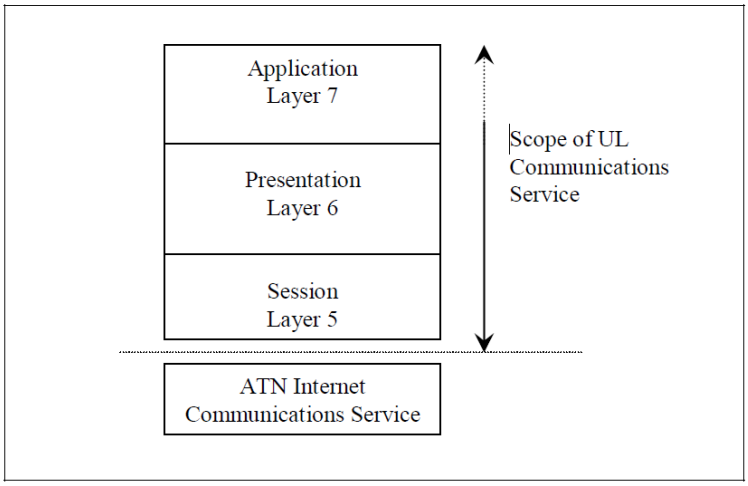

FIGURE 1.12 – Pile de protocoles ATN/OSI

tités, CPDLC (Controler Pilot Data Link Communication), ADS (Automatic Dependent Surveillance), FIS (Flight Information Service), etc. Pour chaque application, le standard spécifie les éléments de services (ASE pour Application Service Elements), l'ensemble de ces services sont regroupés dans une Application Entity (AE), qui constitue la partie communicante de chaque application. Les couches hautes sont aussi constituées de l'équivalent des couches session et présentation du modèle OSI. Le standard définit deux applications dans la catégorie Ground-Ground, il s'agit de AIDC (ATS Interfacility Data Communication) et AMHS (ATS Message Handling Service).

La partie ATN Internet communication services est constituée du protocoles TP4 au niveau Transport qui offre un service proche de celui de TCP, et du protocole CLNP (ConnectionLess Network Protocol), qui offre un service similaire au protocole IP. Les protocoles des couches liaisons et physique sont définis selon les liens physiques concernés, ils sont décrits dans le document [53]. L'accès aux services de communication pour les applications s'effectue via l'interface DS (Dialogue Service) qui offre des primitives de services correspondant aux protocoles des couches hautes.

1.4.3 La transition vers IP

La transition vers les standards de l'Internet a été motivée par la complexité qui résulte de la gestion de deux piles protocolaires lors de l'utilisation des réseaux de coeur de l'Internet. La procédure est encore dans son début, un standard a été définit [52] mais il ne présente que les spécifications d'un système utilisant les protocoles de l'Internet.

Une des exigences de cette transition est le fait de disposer, pour chaque équipement, des deux piles protocolaires durant la phase de transition, qui au passage peut durer plusieurs années, voire plusieurs décennies. La problématique qui en résulte, et qui constitue notre cas d'étude, concerne la gestion de coexistence des deux piles protocolaires pendant la phase de transition. La problématique est illustrée sur la figure 1.13 Nous proposons pour cela, au chapitre 5, un exemple d'application de notre architecture pour gérer des applications et des protocoles hétérogènes correspondant aux deux piles protocolaires, et ce de façon transparente et autonome.

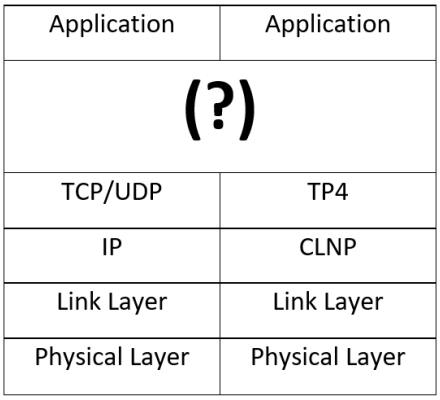

FIGURE 1.13 – Problème de transition vers IP

1.5 Présentation des contributions

Notre travail vise à répondre au problème de l'évolution de la couche Transport. Ce problème consiste en le fait que nous disposons actuellement d'une couche Transport fonctionnant avec des protocoles largement dépassés, offrant un service dégradé aux applications et utilisant de façon sous-optimale les capacités des réseaux, et ce, bien que plusieurs nouvelles solutions plus performantes existent déjà. Il s'agit donc, en premier lieu, d'identifier les obstacles qui empêchent le déploiement de ces nouvelles solutions, et de proposer, en un deuxième lieu, une solution architecturale pour y remédier.

1.5.1 Positionnement des contributions

Ce travail consiste à proposer une solution qui répond à l'ensemble des obstacles qui empêchent l'évolution de la couche Transport. Certains de ces obstacles, ou limites, ont été posés et/ou traités par quelques travaux précédents. Dans cette thèse, nous étudions l'ensemble de ces limites, en partant du principe que du moment où elles ne sont pas traitées dans leur intégralité et en une seule solution, le problème de l'évolution de la couche Transport ne peut être résolu [48]. De ce fait, notre travail ne consiste en aucun cas en la proposition de nouveaux protocoles ou mécanismes protocolaires (fiabilité, contrôle de congestion, ...), mais en la proposition d'une plateforme permettant à de tels protocoles ou mécanismes d'être déployés. Il s'agit donc bien de l'architecture d'une couche Transport et non pas de celle d'un protocole en particulier. Le déploiement des nouvelles solutions doit, bien entendu, prendre en considération leur utilisation par les applications, ainsi que leur gestion par la couche Transport en les associant à leur contexte d'adaptabilité respectif. Enfin, l'architecture doit tenir compte de la coexistence à assumer d'anciennes versions des applications et des systèmes d'exploitation durant la phase de transition.

Techniquement, une telle solution dispose d'une partie dédiée à la gestion, comme ses précédentes propositions. Cette partie comporte à son tour plusieurs outils et modèles pour assurer son fonctionnement : certains sont repris des travaux précédents et ne constituent qu'une réponse à des besoins particuliers ; d'autres constituent de nouvelles contributions, soit en tant que de nouvelles capacités ou fonctionnalités de la couche Transport, telles que l'extensibilité et l'interfaçage avec les applications, soit en tant que nouvelle approche pour la mise en oeuvre de fonctionnalités déjà existantes,

telles que l'invocation des composants et la gestion des évènements. Les contributions techniques peuvent se limiter à des propositions de développement ou de conception, telles que l'interaction entre composants, ou constituer des contributions scientifiques plus conséquentes sous formes de modèles ou d'algorithmes, tels que le modèle de décision présenté au chapitre 4.

1.5.2 Contributions de la thèse

La contribution générale de notre travail consiste en l'étude, la spécification, l'implémentation et la validation d'une architecture évolutive pour la couche Transport. De façon plus spécifique :

Nous étudions tout d'abord les limites de l'architecture actuelle ainsi que celles des architectures proposées pour la couche Transport. Nous tenons compte des différentes limites identifiées dans les propositions précédentes, de leurs différentes interprétations et prises en charge, et nous les étendons avec d'autres limites que nous avons identifiées. Nous décrivons enfin les besoins résultants pour faire face à toutes ces limites.

Nous proposons ensuite une architecture pour la couche Transport, suivant des paradigmes de conception avancés, tels que l'orienté service, l'orienté composant, ou encore le principe d'inconscience sémantique, en tenant compte de l'ensemble des limites identifiées. Parallèlement, nous étudions les éléments internes de la partie gestion de cette architecture en vue d'en adresser les limites et la façon dont ils étaient proposés dans les architectures précédentes. Nous reprenons et ajoutons par conséquent, selon le cas, des composants. Nous proposons aussi de nouvelles approches pour la gestion de notre architecture, via des solutions techniques comme pour la gestion des évènements, ou des modèles et algorithmes comme pour l'autonomie. Ces nouvelles approches sont proposées soit comme en réponse à des besoins spécifique de notre architecture, soit pour assurer certaines de ses fonctionnalités.

La solution conceptuelle est finalement implémentée dans son intégralité, avec ses différents composants, et testée d'abord dans un contexte basique, puis ensuite appliquée à un cas d'étude dans un contexte précis, qui est celui des communications aéronautiques par satellite. Les tests de validation concernent, en premier lieu, l'étude de la faisabilité d'une telle solution, en tenant compte de tous les aspects. En deuxième lieu, il s'agit d'étudier le coût résultant de la partie gestion. Ce coût concerne d'abord l'impact de l'approche de la solution sur les performances habituelles, et ensuite l'éventuel surplus en termes de calculs et/ou de données émises ainsi que le rapport entre ce coût et les gains engendrés.

1.6 Conclusion

Au cours de chapitre nous avons présenté le cadre général dans lequel s'inscrit notre travail. Nous y avons introduit les concepts généraux utilisés dans ce manuscrit ainsi que les états de l'art des travaux en rapport avec le nôtre. Nous avons dressé l'évolution qu'ont connue les solutions de Transport et les limites de la couche actuelle qui empêchent leur déploiement. Nous avons également discuté des nouvelles solutions

au niveau Transport, à la fois au niveau des communications proprement dites et au niveau architectural.

Dans un deuxième temps, nous avons traité plus en détails les solutions architecturales et les éléments communs à leur conception, tels que la gestion des compositions et de l'autonomie. Ensuite, nous avons présenté un bref état de l'art sur le contexte d'application de notre cas d'étude, en rapport aux communications aéronautiques. Enfin, nous avons décrit les contributions de notre travail.

Le chapitre suivant présente l'architecture conceptuelle que nous avons proposée pour la couche Transport, ainsi qu'un état de l'art spécifique sur les approches et les outils retenus pour sous-tendre la partie comportementale de l'architecture proposée.

Chapitre 2

Architecture pour la couche Transport

Nous avons présenté dans le chapitre précédent le positionnement de ce travail et les éléments clés mis en avant en tant que contributions de cette thèse. La problématique adressée concerne le besoin d'évolution de la couche Transport qui impose à ce jour l'utilisation de solutions offrant un service inefficace ou incomplet aux applications, et un usage sous-optimal des ressources réseaux, alors même que de nouvelles solutions plus efficaces voient le jour au fur et à mesure de l'évolution des besoins des applications et des technologies réseau.

Dans ce chapitre, nous présentons tout d'abord une étude des obstacles expliquant le non déploiement des nouvelles solutions de Transport. Cette étude vient compléter la liste des limites décrite au chapitre précédent. Dans un second temps, nous présentons une solution architecturale pour répondre à l'ensemble de ces limites. Cette solution est décrite par son principe de fonctionnement général, ses composants internes, les interactions internes et avec l'environnement externe, et enfin les modèles conceptuels associés. Une description plus détaillée, couvrant des contributions plus techniques, sera présentée dans les prochains chapitres.

2.1 Limites et besoins

Nous étudions dans cette section les limites de la couche Transport telle qu'elle est déployée actuellement ainsi que celles des tentatives des solutions proposées en réponse à ces limites. Nous établissons ensuite une liste des besoins résultants.

2.1.1 Limites des architectures existantes

Le fait de fonctionner, encore à ce jour, avec des solutions datant à quelques exceptions près des années 80, est dû à une série de limites qui rendent le déploiement de nouvelles solutions, par ajout ou par mise à jour, extrêmement coûteux à tous les points de vue (temps, difficulté, risques de bugs, ...) et touchant tous les acteurs concernés (applications, systèmes, ...) [48].

Les nouvelles architectures de Transport, présentées au chapitre précédent, ont apporté quelques solutions à ces limites, mais rencontrent elles-mêmes des difficultés pour être déployées, soit à cause d'un traitement partiel de ces limites, soit par des limites internes à l'architecture elle-même. Dans ce qui suit, nous présentons l'ensemble des limites pour le déploiement de nouvelles solutions au niveau Transport [34].

Problème de dépendance la difficulté dans l'intégration des nouvelles solutions vient avant tout du coût important qui résulte de leur déploiement, concernant notamment le système d'exploitation, mais aussi les applications et la version déployée du protocole lui-même en cas de mise à jour. Ce coût résulte de la forte dépendance qui existe entre la couche Transport et les acteurs externes, applications et systèmes notamment.

D'un côté, les interfaces par le biais desquelles les protocoles sont invoqués par les applications sont très spécifiques, leur syntaxe est donc incluse dans le code de l'application lors de sa conception. Une fois une application écrite pour un protocole donné, il n'est plus possible qu'elle utilise un autre protocole sans devoir modifier son code. Cette dépendance entre les applications et les protocoles de Transport sous-jacents implique également que toute modification apportée à l'interface d'un protocole aura un impact sur l'application. Plus important, il n'est pas possible pour une application de tirer parti des nouvelles fonctionnalités d'un protocole sans modifier son code source. A titre d'exemple, il est courant que des applications soient obligées de gérer elle-même certains services s'ils ne sont pas offerts par le protocole sous-jacent (cas par exemple des applications multimédias fonctionnant avec UDP qui doivent implémenter leur propre contrôle de congestion). Si le protocole supporte le contrôle de congestion après conception de l'application, il n'est plus possible d'en tirer parti sans modifier le code de l'application, ni de migrer vers un autre protocole, tel que DCCP par exemple qui, lui, intègre un contrôle de congestion.

D'un autre côté, ces protocoles sont implémentés comme une partie du système d'exploitation. Il en résulte que toute modification du code de ces protocoles doit se faire dans le noyau, ce qui nécessite une modification du système d'exploitation. Ce fait a plusieurs répercussions sur le déploiement des protocoles. D'abord le coût d'une modification de noyau est plus élevé tant en termes de risques de bug qu'en terme de travail d'implémentation. Ensuite, ces modifications du système vont créer une certaine hétérogénéité dans leurs différentes versions avec des fonctionnalités qui ne seront dispo-

nibles que dans certains systèmes, créant ainsi une incohérence dans le fonctionnement global de la couche Transport.

Problème d'extensibilité le terme d'extensibilité peut être vu de différentes façons selon le contexte. Dans notre cas, nous l'entendons par la capacité à pouvoir apporter des modifications à un programme par ajout de nouvelles fonctionnalités ou par mise à jour des fonctionnalités existantes, sans avoir à recompiler le programme, au moins dans son intégralité. En effet, les protocoles de Transport, à l'instar de tant d'autres, sont codés en un seul bloc non séparé. Par conséquent, toute modification aussi petite soit-elle dans le protocole, nécessite une recompilation entière du code, ou à défaut, l'implication d'autres composants dans la compilation de la partie modifiée. Le manque d'extensibilité relève donc d'une sorte de dépendance interne, qui nous ramène aux implications du problème précédent.

Problème de mise à l'échelle les systèmes d'exploitation sont de plus en plus complexes. Développés en leur sein, les protocoles de Transport contribuent à cette complexité. Il en résulte que plus on dispose de protocoles, plus on accroît la complexité des systèmes. Ces derniers ne peuvent donc pas intégrer autant de protocoles que nécessaires, et le choix s'oriente vers le déploiement de solutions de Transport générales moins coûteuses du fait qu'une seule solution couvre une large gamme de besoins applicatifs et réseaux. Ce problème, relevant d'une difficulté de mise à l'échelle, est un réel obstacle au développement des protocoles de Transport ; même pour des protocoles standardisés, tels que DCCP ou MPTCP, plusieurs années sont nécessaires avant que ceux-ci puissent être déployés. Ce constat ne laisse pratiquement aucune chance au déploiement effectif de solutions spécifiques dont le champ d'application est fortement réduit et dont la durée de vie est relativement courte.

Problème de complexité le problème de complexité concerne la difficulté de l'utilisation de la couche Transport quand celle-ci comporte un nombre important de solutions. L'évolution de la couche Transport ne peut se baser sur des solutions générales, et une solution spécifique par contexte impliquerait un grand nombre de solutions face auxquelles un développeur aura du mal à faire son choix. Il n'est donc pas commode de présenter l'ensemble de ces solutions et de laisser le choix au développeur. Problème d'accès réseau : pour des raisons technico-économiques, les fournisseurs d'accès n'autorisent pas tous les protocoles à passer par leurs routeurs [48]. En effet, ces fournisseurs appliquent des filtres sur les entêtes des paquets, et ne laissent passer que les protocoles autorisés afin de préserver la cohérence de leur réseau et éviter d'éventuelles congestions. Actuellement, il n'y a pratiquement que le protocole TCP qui soit universellement autorisé, ainsi que le protocole UDP avec plus de restrictions.

Boucle de réticence il s'agit de la réticence de chacun des acteurs impliqués dans le Transport envers le développement, le déploiement ou l'usage de nouvelles solutions à cause des réticences des autres. Ainsi, les développeurs d'applications ne peuvent pas utiliser un protocole, bien qu'il leur soit le plus adapté, s'ils n'ont pas la garantie que leurs applications puissent fonctionner n'importe où, notamment pour des applications distribuées à grande échelle dont les enjeux techniques et économiques peuvent s'avérer très important. En effet, une application qui ne fonctionne pas tout le temps correctement risque de perdre en crédit et d'être ainsi peu utilisée.

De la même façon, les systèmes d'exploitation ne peuvent pas se permettre de déployer

de nouveaux protocoles avant que ceux-ci ne prennent de l'envergure et que leur usage ne soit universel. Le coût de déploiement et les incohérences qui en résultent étant très élevés, il en est de même dans la complexité du noyau tant au développement qu'à l'exécution : un système d'exploitation ne peut pas investir dans un nouveau protocole si celui-ci ne présente pas un enjeu important, qui se caractérise principalement par une large demande de la part des applications.

Enfin, et en conséquence, les développeurs des protocoles ont du mal à investir de gros efforts dans la conception, le développement et l'étude continue de nouvelles solutions de Transport si celles-ci ont très peu de chances d'être déployées dans les systèmes d'exploitation et utilisées par les applications.

En résumé, les développeurs d'applications attendent que la solution soit largement déployée avant de l'utiliser, pour ne pas risquer d'avoir des dysfonctionnements liés à la non disponibilité de la solution. Les systèmes attendent qu'une solution soit largement déployée avant de l'intégrer, pour ne pas investir du temps et de l'effort pour une solution marginale. Enfin, les développeurs de protocole ne peuvent plus en développer dans de telles conditions. Nous nous retrouvons donc dans une boucle de réticence, dans laquelle chaque acteur prend ses gardes quant à sa contribution à l'évolution de la couche Transport à cause des réticences des autres acteurs.

2.1.2 Besoins pour une couche Transport évolutive

De la liste des limites que nous avons dressées dans la section précédente résulte un certain nombre de besoins pour que la couche Transport permette le déploiement, la gestion, l'adaptation et l'utilisation des nouvelles solutions protocolaires. Ces besoins sont les suivants.

Indépendance la couche Transport doit limiter au minimum ses interactions avec son environnement. Les applications dépendantes des services de la couche Transport doivent se limiter au service rendu et en aucun cas à la façon dont ce service est techniquement implanté. Ceci permettra, d'une part, à la couche Transport de faire évoluer les services offerts aux applications sans avoir à affecter les applications les utilisant, et d'autre part, aux applications de bénéficier de meilleurs services sans qu'elles aient à être modifiées.

Transparence tous les éléments de la couche Transport (composants internes) qui sont susceptibles d'évoluer doivent fonctionner de façon transparente les uns des autres. Les composants échangent des services entre eux sans savoir ni comment ces services sont réalités, ni qui les réalisent. Il s'agit donc d'une sorte d'indépendance interne entre les composants de l'architecture. Cette indépendance est également à mettre en oeuvre à un niveau de granularité plus fin à l'intérieur des éléments eux-mêmes. Globalement, l'implantation des fonctions qui auront à être réalisées par certains éléments au profit d'autres, comme nous le verrons dans le monitorage du réseau par exemple, doit être complètement transparente aux autres éléments. Ainsi, lorsqu'un composant de la couche Transport devra être mis à jour ou remplacé par un autre, les modifications ne se rapporteront qu'à l'élément en question, dès lors que la fonction réalisée demeurera la même. Si un composant (qui assure par exemple un contrôle de congestion), a besoin de la valeur du RTT, il ne doit pas savoir qui calcule cette valeur, ni comment elle est

calculée techniquement. De cette façon, même si le composant qui calcule le RTT varie selon les compositions, cela n'affectera pas le contrôle de congestion. Aussi la valeur en question doit être partagée de façon brute, sans aucun traitement supplémentaire, de sorte à ce que le contrôle de congestion puisse modifier sa formule d'estimation du RTT sans avoir à modifier les composants qui effectuent le monitoring de base.

Inconscience il s'agit du degré de connaissance sémantique que possède un élément de la couche vis-à-vis des données qu'il traite. Plus ce degré est faible, plus l'élément est extensible et sa durée de vie importante. Ceci concerne les éléments génériques traitant de concepts de haut niveau. A titre d'exemple, un élément chargé de sélectionner les composants adaptés à un certain contexte réseau ne doit pas se baser sur la sémantique du contexte en question, de son changement ou du service rendu pour l'application. Le processus de sélection doit se baser uniquement sur des valeur objectives (délais, débit, pertes ...) dont la description est externe au composant (via une base de données externe). De cette façon, il est possible d'ajouter autant de profils de réseau, de services, de composants, ou de métriques, sans avoir à modifier le composant de sélection.

Extensibilité telle que nous l'avons définie, le besoin en extensibilité est assuré via la réponse aux trois besoins précédents. Ainsi, l'indépendance règle les problèmes liés aux applications et aux systèmes d'exploitation, en limitant (annulant) sur ces derniers l'impact d'éventuelles évolutions. La transparence réduit le coût résultant de l'évolution interne de la couche Transport (évolution de ses composants). Enfin, l'inconscience des composants de gestion de l'architecture vis-à-vis de la sémantique des données augmente leur durée de vie en offrant la possibilité de les faire évoluer sans avoir à modifier l'élément lui-même.

Mise à l'échelle la couche Transport doit offrir un moyen permettant d'intégrer les différentes solutions sans "encombrer" le système d'exploitation. En termes d'implémentation, la couche Transport doit ainsi intégrer le moins de code possible dans le noyau afin d'en dépendre au minimum, et ce, en n'y incluant que la partie qui ne risque pas d'évoluer, ni surtout d'être étendue. Lors de l'exécution, la couche Transport doit assurer un fonctionnement optimal en ne chargeant que les éléments nécessaires. Répondre au besoin de mise à l'échelle lève ainsi les limites des systèmes d'exploitation à ne pouvoir intégrer qu'un nombre très limité de protocole.

Services et autonomie la couche Transport doit limiter l'interaction avec les applications aux services qui leur sont fournis. Ceci vise à réduire la dépendance classique des applications aux protocoles et à faciliter par la même occasion son utilisation. Par ailleurs, la couche Transport doit prendre la responsabilité du choix des composants nécessaires en tenant compte des besoins des applications et des réseaux de communication sous-jacents.

Accès universel en plus d'optimiser la communication en fonction des caractéristiques réseau, la couche Transport doit d'abord rendre cela possible de façon universelle, c'est à dire qu'elle doit pouvoir éviter d'éventuel blocages de la part des réseaux intermédiaires. Pour ce faire, elle doit tenir compte de l'ensemble des contraintes posées par les fournisseurs d'accès afin de lever toute cause d'un éventuel refus d'accès aux données émises par la couche Transport, et ce quel que soit le mode de transmission et les composants associés.

2.2 Une nouvelle architecture pour la couche Transport

Dans l'optique de répondre aux besoins à satisfaire face aux limites d'évolution de la couche Transport, l'un des objectifs de nos travaux est de présenter une nouvelle architecture adaptée à cette fin. Nous présentons dans cette section les principes et la philosophie de cette nouvelle architecture ; une description plus détaillée sera présentée dans les sections suivantes.

2.2.1 Principes généraux et positionnement

La solution que nous proposons est une nouvelle architecture pour la couche Transport :
- Permettant son évolution (c'est à dire l'intégration de nouvelle solutions protocolaires sans impact sur les applications, les systèmes et la couche elle-même), ainsi la prise en compte de nouvelles solutions ;
- Intégrant une forme d'autonomie dans le choix des solutions à mettre en oeuvre en réponse aux besoins des applications et aux caractéristiques réseau pour chaque communication.

Notre approche n'est donc pas de proposer un nouveau mécanisme assurant la communication au niveau Transport (tel qu'un mécanisme de controle de congestion ou de reprise de pertes), ou un nouveau protocole offrant un certain nombre de services.

Dans ce travail, nous partons du fait qu'il existe de meilleures solutions (au sens mécanismes) que celles déployées actuellement, et ce sur plusieurs plans (adéquation aux applications, meilleur service, meilleures performances, ...). Notre optique est alors de proposer et de développer un cadre permettant à ces solutions d'être déployées, invoquée, (re-)paramétrées voir remplacées, ceci de façon dynamique, c'est à dire à l'initialisation et/ou durant la communication, et autonome, c'est à dire sans que l'application aient à en faire le choix.

Sur le plan conceptuel, nous partons du principe que du moment que les limites architecturales de la couche actuelle n'ont pas été traitées dans leur ensemble, le problème de la couche Transport persistera. Ceci peut facilement être déduit des nombrables architectures de protocoles ou de plateformes qui ont été proposées au niveau Transport, mais qui n'ont dans les fait pas été déployées / utilisées dans l'Internet. Une importante et récente étude sur la couche Transport, a d'ailleurs abouti à la même conclusion [48]. Les auteurs estiment que la raison principale, qui a fait que toutes propositions de nouvelles architectures au niveau Transport n'ont pas été concrètement déployées, réside dans le fait qu'elles considèrent, toutes, un aspect particulier des limites de la couche Transport actuelle, et qu'elles ne traitent pas la problématique dans son intégralité. L'architecture que nous proposons prend en compte l'ensemble des besoins décrits précédemment, chacun de ses composants ayant été conçus pour répondre à un ou plusieurs de ces besoins.

Notre proposition se positionne donc de la façon suivante :
- Comparativement aux travaux traitant des performances des protocoles de Transport qui proposent de nouveaux mécanismes ou qui en améliorent les mécanismes existants, notre architecture est différente en ceci qu'elle propose une solution pour

permettre à ces nouveaux mécanismes d'être déployés et choisis au bon moment ;
– Comparativement aux travaux traitant des aspects architecturaux, notre solution se pose d'abord comme une architecture de la couche Transport et non pas comme celle d'un nouveau protocole qui sera déployé selon le paradigme actuel, en plus des protocoles déjà existants. Ensuite, notre architecture répond à l'ensemble des problèmes et des limites des architectures existantes.

Notre solution comporte à la fois les deux aspects généraux et spécifiques des architectures de Transport. En effet, l'architecture que nous proposons peut être utilisée par n'importe quel type d'application et déployée avec n'importe quelle technologie réseau. De surcroît et du fait de son caractère évolutif, elle peut facilement s'adapter pour être utilisée par de nouvelles applications et supporter de nouveaux types de réseau. En le même temps, l'architecture permet d'aboutir à un comportement protocolaire spécifiquement adapté à chaque situation. Par cette combinaison, l'architecture offre ainsi les possibilités de déploiement des solutions générales et les performances des solutions spécifiques. Son principal apport comparativement aux architectures similaires est sa capacité d'évoluer et de prendre en charge de nouvelles situations, et de les déployer sans impact sur les applications et les systèmes. Avec de telles caractéristique, cette architecture a vocation à être réellement déployée.

Suivant le même principe, notre architecture combine la performance des architectures implantées dans le noyau et la flexibilité de celles implantée en espace utilisateur. Notre architecture sépare les composants de communication et la partie de gestion de la couche Transport, et n'implante que ces dernier dans le noyau, ce qui fait que les éléments susceptibles d'évoluer ne sont pas implémentés physiquement dans le noyau. Ils sont chargés de façon dynamique, uniquement en cas de besoin, par la partie gestion, Nous permettant ainsi des possibilités de maintenabilité et d'évolution pour la couche Transport sans avoir à apporter des modifications au noyau.

2.2.2 Description globale de l'architecture

Notre architecture se base sur une séparation entre les éléments de communication proprement dits (encapsulation, bufferisation, fiabilité, congestion ...) et la partie de gestion de l'architecture (interfaçage avec les applications et le réseau, gestion des compositions, ...). Dans les architectures proposées au niveau Transport, y compris celles qui sont basées sur la notion de composants, il n'y a pas de séparation claire entre les éléments constituant l'architecture. C'est cependant cette forte liaison entre les composants qui fait que la modification des uns implique la modification des autres, et augmente par conséquent le coût des extensions. Également, cette forte liaison empêche la plupart du temps des compilations séparées, ce qui est très coûteux pour des manipulations dans le noyau.

L'idée est donc de séparer la partie ayant vocation à évoluer de celle relativement stables (au moins sur une longue durée). La partie évolutive de l'architecture, constituée des composants de communication, sera déployée indépendamment du noyau de telle sorte à pouvoir la modifier sans impact sur le système. La partie statique, constituée des éléments de gestion de l'architecture, peut, quant à elle, être intégrée au noyau. Concernant l'exécution cependant, l'ensemble de l'architecture sera chargée dans le noyau. Un des avantages des architectures déployées en dehors du noyau réside en

la simplicité de modification par extension ou par mise à jour. La séparation des deux parties de notre architecture permet d'avoir cette simplicité de déploiement, tout en conservant la puissance d'une exécution à l'intérieur du noyau. L'architecture globale est décrite dans le diagramme de la figure 2.2.

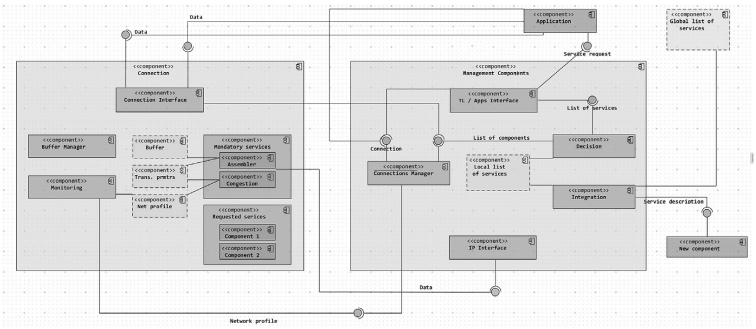

FIGURE 2.1 – Architecture globale

On y distingue deux grandes parties. La partie droite comporte les éléments de gestion de la couche et la partie gauche les éléments de communication proprement dits. Dans la suite de ce manuscrit :

– nous appellerons "composants de gestion" les composants de la partie gestion (interface réseau, décision, ...) ainsi que les composants de gestion des connexions (interface, buffer, ...) ;
– les composant dédiés aux services de communication (assemblage, contrôle de flux, fiabilité ...) seront appelés "composants de communication".

Partie Gestion

Pour la partie gestion, il ne peut y avoir qu'une seule instance des éléments de gestion dans un système donné, et toutes les communications de Transport sont gérées par cette seule instance.

Cette partie comporte une interface offerte à toutes les applications. Cette interface permet, d'une part, aux applications d'exprimer leurs besoins en termes de services de Transport, et d'autre part, de gérer une grande partie des interactions entre les composants de gestion.

Parallèlement, on trouve également : 1) le composant de décision qui est responsable de la sélection des composants de communication à déployer et du suivi des évolutions du réseau, 2) le composant de gestion des connexions, et enfin 3) le composant d'intégration de nouveaux composants de communication. Nous étudierons l'ensemble plus en détails dans la suite de ce chapitre.

En plus de ces composants, on trouve dans cette partie plusieurs bases de données qui contiennent différentes listes relatives soit à l'architecture globale soit à un composant de gestion particulier. La plus importante est la liste locale des composants dans laquelle sont décrites les associations service/composants avec les différents paramètres associés. Nous l'étudierons plus en détail, avec les autres bases de données, dans la suite de ce chapitre.

Partie Communication

La partie connexion est instanciée pour chaque communication. Elle comporte, en plus des composants dédiés à la communication proprement dite, un buffer (avec ses fonctions de gestion) et des mémoires partagées destinées à la l'interaction entre les composants de communication d'une part, et entre la connexion et les composants de gestion d'autre part. Parmi ces composants, certains sont instanciés par défaut. Il y a principalement le composant d'assemblage, qui est présent pour toutes les connexions. Il s'occupe de la création des PDU par invocation des autres composants et du regroupement des entêtes. D'autres composants par défaut, décrits dans une base de données dédiée, sont instanciés selon le contexte.

2.2.3 Structuration par rapport aux besoins

L'architecture que nous proposons a été conçue pour répondre à l'ensemble des besoins identifiés précédemment. Ainsi, chaque élément de la couche est défini pour répondre à un ou plusieurs besoins.

Ainsi, l'interface de service assure l'indépendance entre les applications et la couche Transport, par une transparence vis-à-vis des aspects techniques des services rendus. Avec cette interface, les applications ne dépendent plus d'un protocole particulier et n'ont plus besoin de connaître la façon avec laquelle le/les services requis sont réalisés. Ceci permet à la couche Transport d'évoluer et de s'adapter aux nouvelles technologies réseau sans impacter les applications, et, dans l'autre sens, aux applications de changer de services, de façon statique ou dynamique, sans avoir à modifier leur code.

L'extensibilité est assurée, d'abord, par la mise au point d'un mécanisme d'intégration des nouveaux composants de communication via le composant d'intégration ; ensuite, pour prendre en compte ces nouveaux composants, par une liste que ce dernier met à jour au profit d'autres composants de gestion qui auront, par la suite, à les instancier. De cette façon, la recherche des composants de communication associée à une liste de services requis par l'application se fera à partir d'une liste extensible.

Le composant de décision assure de son côté l'autonomie de la couche vis-àvis du choix des composants protocolaires pour répondre aux requêtes des applications en tenant compte des caractéristiques du réseau. Il assure ainsi la sélection des composants de communication associés aux services requis par l'application, tout en sélectionnant les composant les plus adaptés au contexte réseau. Enfin, la liste des services par défaut, qui fait partie des bases de données locales, assure un accès universel par l'imposition, pour chaque communication, des services nécessaires pour garantir que la transmission ne soit pas bloquée par les éléments du réseau intermédiaire.

2.3 Description des composants

Dans la mesure où notre architecture ne propose pas les composants de communication proprement dits (ils existent par ailleurs et ont vocation à être intégrés dans la couche Transport), nous ne présentons dans cette section que les composants de gestion et les bases de données associées.

2.3.1 Composants de gestion de l'architecture

Les composants de gestion sont implantés directement dans le noyau et assurent le lien entre les applications et le réseau, ainsi que la gestion des composants et des communications.

Application Interface Ce composant sert d'interface entre les applications et la couche Transport ainsi que de relai pour les autres composants de gestion. Pour les applications, cette interface contribue à l'abstraction des détails d'implémentation et de mise en oeuvre associés aux services de Transport. Les applications lui transmettent leurs requêtes sous forme d'une liste de services qui sera traitée et traduite en une liste de composants de communication, liste qui sera enfin instanciée et mise en relation avec l'application en question. Ce processus, qui sera décrit plus en détails dans la suite, est réalisé en collaboration avec les composants de décision et de gestion des connexions. Ainsi, l'interface gère la communication entre ces différents composants pour chaque communication.

Cette interface assure l'abstraction et la transparence de la réalisation concrète par des requêtes orientées service. Elle assure aussi l'adaptabilité de la couche aux variations des besoins des applications en cours de communication.

Connections Manager le gestionnaire des connexions s'occupe de l'instanciation des connexions, du chargement effectif des composants de communication et de leur déchargement une fois la connexion achevée. Aussi, il permet la modification des compositions, soit suite à un changement de requête par l'application, soit suite à un changement du contexte réseau. Il gère la liaison entre l'interface applicative, le composant de décision et les connexions.

Le gestionnaire des connexions est un composant purement technique, ne contribuant à aucun des besoins de haut niveau identifiés précédemment. Il s'agit d'un choix interne visant à déléguer l'instanciation des connexions à un composant dédié et d'alléger l'interface application et le module de décision.

Decision component le composant de décision s'occupe de la traduction des requêtes de l'application d'un format de haut niveau orienté service à un format plus technique, sous forme d'une liste de composants logiciels assurant les services en question. Le composant de décision peut être sollicité à l'établissement de la connexion ou en cours de communication suite à un changement significatif dans l'état du réseau.

Le module de décision assure l'adaptabilité de la couche Transport vis-à-vis des caractéristiques du réseau et des besoins de l'application. Il assure également l'abstraction des détails techniques aux applications par la mise en oeuvre de la possibilité d'une interaction orientée service qui sera traduite en composants. Il assure enfin l'extensibilité des composants de communication par une sélection dynamique, non préfixée, à partir d'une base de données extensible.

Integration component le composant d'intégration s'occupe de l'ajout des nouveaux composants et services à la couche Transport et de la gestion de certaines bases de données. Il est invoqué par les nouveaux composants de communication souhaitant être intégrés à la couche. Il assure l'identification de la description du composant en question, son enregistrement physique, et la mise à jour des bases de données concernées avec les nouvelles.

La mise à jour des bases de données se fait selon la nature du composant. Si le composant offre un nouveau service qu'il est le premier à proposer, la base globale des services doit être mise à jour ; la base locale est mise à jour de toute façon pour y intégrer le nouveau composant afin qu'il puisse être pris en compte lors de la prochaine décision.

IP Interface l'interface IP (ou interface réseau) assure la communication avec la couche réseau et par conséquent gère le multiplexage et le démultiplexage des paquets. Toutes les données en provenance des connexions en cours transitent forcément par ce composant ; aucune connexion n'est autorisée à communiquer directement avec le réseau. Ceci permet (si cela est opportun) d'assurer l'équité de l'occupation de la bande passante (locale) entre les différentes connexions. Un comportement remettant en cause cette équité pourrait être envisagé à des fins de QoS différenciée par exemple.

A l'instar du gestionnaire des connexions, l'interface IP correspond à un besoin interne, fonctionnant en gros selon le même principe que les protocoles classiques, et ne répond à aucun des besoins fondamentaux identifiés pour la nouvelle architecture.

Gestionnaire des application legacy par "applications legacy", nous entendons les applications écrites pour un des protocoles de Transport actuels, et donc inconsciente de la nouvelle architecture. Le gestionnaire de ces applications est invoqué dans le cas où la couche Transport a été sollicitée via l'interface classique. Il s'occupe de plusieurs fonctions, qui concourent globalement à l'abstraction de l'existence de la nouvelle architecture pour ce type d'applications. Il se base notamment sur la translation des primitives classiques en des primitives de la nouvelle architecture.

Ce composant doit en premier intercepter les appels classiques et les transférer, soit vers la partie gestion (interface application) dans le cas d'une initiation de la connexion, soit vers la connexion résultante dans le cas d'un transfert de données. Il a donc à gérer le lien entre les applications et les connexions qui ont été créées, et ce de façon complètement transparente pour ces applications.

2.3.2 Bases de données

A la partie gestion est associé un ensemble de bases de données contribuant principalement à l'extensibilité de l'architecture. On trouve la base globale des services, la base locale des composants, la liste des valeurs monitorées, et la liste des composants par défaut.

La liste globale des services est à destination des développeurs et contient la description de l'ensemble des services offerts par la couche Transport. La disponibilité de ces services peut cependant varier d'un système à un autre selon leurs capacités et la fréquence de leurs mises à jour. La base des services est structurée de façon hiérarchique et extensible. La hiérarchie est réalisée par un mécanisme de "sous-services". Ainsi, chaque service peut proposer plusieurs niveaux de détails (sous-services), et les applications peuvent demander un service de façon très générale ou plus spécifique. A chaque niveau de service est associé un sous-service par défaut qui est sélectionné pour le niveau demandé.

A chaque service est associé, en plus de sa description sémantique, un identifiant unique avec lequel il sera invoqué par l'application. Aussi, une liste de paramètres est décrite pour chaque service, certains obligatoires et d'autres optionnels.

Par ailleurs, en interne à la partie gestion, d'autres bases sont maintenues afin de gérer la création et le suivi des connexions. La plus importante est la base locale des composants qui contient les informations sur les composants de communication enregistrées lors de leur intégration dans l'architecture. Cette base contient, entre autres, les liens physiques des modules logiciels des composants, les services qu'ils peuvent offrir, leurs domaines d'adaptabilité, ... Ces informations vont servir par la suite au composant de décision pour sélectionner les composants les plus adaptés aux besoins des applications et au contexte réseau.

D'autres bases stockent les éléments de configuration que sont les paramètres de monitoring et les services par défaut. Les paramètres de monitoring sont les paramètres de réseau pour lesquels il existe au moins un composant qui puisse les assurer ; ces paramètres sont décrits par leurs identifiants ainsi que leurs métriques. Les services par défaut sont les services qui seront instanciés pour chaque connexion indépendamment de la requête de l'application. Cette partie est dédiée aux administrateurs qui vont fixer la liste de ces composants selon les politiques de gestion adoptées.

2.3.3 Composants de gestion des connexions

Une connexion est instanciée pour chaque requête d'application. Elle est constituée de composants assurant la gestion de la connexion en interne, de composants de communication, et de mémoires partagées.

Les composants de gestion comportent notamment un gestionnaire du buffer et un composant d'assemblage. Le gestionnaire du buffer gère l'ajout et le retrait de données vers et depuis le buffer. Il a pour rôle d'abstraire à la connexion la mécanique liée à la gestion de la mémoire et de prendre en compte des situations dans lesquelles plusieurs composants doivent donner leur accord avant de retirer des données du buffer.

Le composant d'assemblage s'occupe de la transmission de données depuis et vers le buffer. Une fois les données de l'application disponibles dans le buffer d'émission, l'assembleur les récupère. Ces données sont ensuite transmises à l'ensemble des composants de communication pour d'éventuels traitements et ajouts d'entêtes. L'ordre d'appel des composants de communication est fixé en amont par le composant de décision. Les composants de communication peuvent ajouter un entête, comme pour le contrôleur de flux, modifier les données, comme pour les composants de compression ou de chiffrement, ou simplement faire des calculs comme pour le contrôleur de congestion

Parmi les composants d'une connexion, certains sont choisis en fonction des besoins de l'application tandis que d'autres sont choisis par défaut (tel qu'un contrôle de congestion). Les composant relatifs aux besoins de l'application sont choisis par le composant de décision en tenant compte d'un ensemble de paramètres, comme l'état du réseau, les capacité du système distant, ... Les composants par défaut sont toujours choisis et ajoutés à la connexion sans demande préalable de l'application.

La liste des composants par défaut diffère en fonction la politique adoptée dans chaque

système. Un composant de contrôle de congestion peut par exemple être imposé pour lever la non acceptation par le réseau des protocoles non "TCP-friendly" imposés par les fournisseurs d'accès. Il peut s'agir aussi d'un module de chiffrement dans le cadre de communications critiques, ou encore d'un composant de compression en cas de ressources restreintes. Quel qu'en soit la raison, cette possibilité est offerte afin de permettre l'imposition d'un ensemble de services Transport pour toutes les connexions sans que ceux-ci aient à être invoqués par l'application. L'aspect par défaut de ces composants n'est considéré que dans le processus décisionnel.

Une autre catégorie de composants sont les composants de monitoring. Ils sont utilisés dans le cas où les composants de base (correspondant uniquement aux services requis par l'application) ne couvrent pas les besoins en monitoring de celle-ci. Ceci est en rapport avec l'approche adoptée dans notre architecture qui fait que, par soucis de performances, les composants de communication sont utilisés, dans la mesure du possible, pour effectuer le monitoring du réseau, et ce afin d'éviter le transfert d'un surplus de données. Dans le cas où cela n'est pas suffisant, les composants de monitoring sont utilisés. Ce processus est décrit plus en détail au chapitre 4.

La dernière partie du composant Connection consiste en les mémoires partagées entre composants. Ces mémoires sont créées en fonction des composants instanciés. En effet, chaque composant comprend dans sa description l'ensemble des valeurs qu'il peut partager et celles dont il a besoin. Au moment de la création d'une connexion, le gestionnaire récupère toutes les valeurs partagées par l'ensemble des composants de la connexion en question, leur réserve de la mémoire et envoie une référence de ces mémoires aux composants qui en ont besoins. Ensuite, pendant la communication, les composants peuvent lire ou écrire dans ces. Ces valeurs peuvent être par exemple, pour le composant de fiabilité, le taux d'erreur et la valeur du RTT à usage du contrôleur de congestion. Ces valeurs sont publiées par les composants de façon brute, c'est-à-dire sans aucun traitement. La valeur du RTT sera donc le temps réel d'aller-retour sans aucune formule de modification. Il est donc à la charge des composants utilisant ces valeurs d'y effectuer les traitements appropriés.

2.4 Cas d'utilisation et description comportementale

Après cette description architecturale, nous traitons dans cette section l'aspect comportemental de notre architecture. Nous décrivons les interactions avec les acteurs externes dans différentes situations, ainsi que le déroulement de certains processus élémentaires.

2.4.1 Cas d'utilisation

Cette section décrit les cas d'utilisation illustratifs des interactions de notre architecture avec les acteurs externes. Un acteur externe peut être une application utilisant la couche Transport ou une couche Transport d'un système distant ; les deux peuvent être conscients ou non de la nouvelle architecture.

Les applications conscientes (ou aware) sont conçues pour fonctionner avec l'interface de la nouvelle architecture, alors que les applications non consciente (ou legacy) conti-

nuent d'utiliser l'interface socket classique. Il en est de même pour les systèmes. La couche Transport est donc amenée à communiquer et à gérer les différents acteurs, qu'ils soient aware ou legacy, et ce afin de considérer la phase de transition durant laquelle des applications et systèmes incompatibles vont coexister.

Nous détaillerons ces différentes interactions dans ce qui suit.

Cas d'utilisation orientés Application

Dans un schéma général, les applications spécifient un service à la couche Transport avant de pouvoir émettre ou recevoir des données. Les applications legacy choisiront de façon classique le protocole souhaité et interagiront avec lui. Les applications aware spécifient leurs besoins en utilisant la nouvelle interface de service proposée par notre architecture.

Cas d'utilisation 1 : Applications legacy - Demande d'un protocole

Pour initier la communication, une application legacy choisit le protocole avec lequel elle souhaite communiquer et le demande à la couche Transport via l'interface socket classique en spécifiant les paramètres habituel (type du réseau, type de protocole, adresse, ...). L'objectif est soit d'établir un lien avec l'entité distante pour les protocoles orientés connexion, soit simplement d'ouvrir un accès réseau pour les protocoles non connectés.

Notre couche Transport doit en conséquence offrir, pour une application legacy, la même interface socket classique, pour qu'elle puisse solliciter un protocole donné et l'utiliser avec les primitives habituelles sans avoir à modifier son code. La gestion des applications legacy en tant que cas particulier doit être complètement transparente, et une application déjà écrite pour l'interface classique doit pouvoir fonctionner normalement, c'est à dire sans modification.

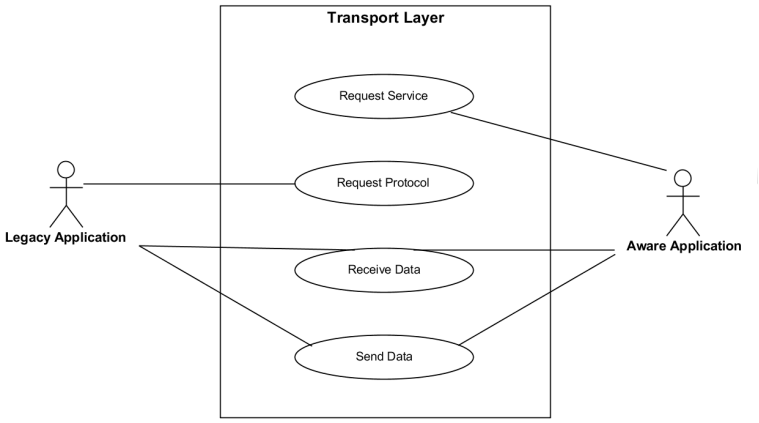

FIGURE 2.2 – Interactions entre la couche Transport et les applications

La couche Transport doit pouvoir détecter les appels classiques et les intercepter en donnant, pour les applications en question, l'illusion d'une l'interface socket habituelle. Par la suite, et selon les capacités du système distant, la communication peut être gérée de façon classique (protocolaire) ou suivant le mode de la nouvelle architecture. Dans les deux cas, la présence d'une couche Transport avec une architecture différente et

les choix associés selon les différents contextes doivent rester transparents pour les applications.

Si le système distant dispose de la nouvelle architecture, la couche Transport crée une composition correspondant au bloc de services du protocole sollicité, avec toutes les optimisations prévues dans la nouvelle architecture (adaptation au réseau, services par défaut, ...). Au cours de la communication, l'ensemble des primitives classiques seront traduites en appels aux méthodes correspondantes de la nouvelle architecture.

Dans le cas où le système distant ne dispose pas de la nouvelle architecture, des composants équivalents aux protocoles classiques sont instanciés, et le lien entre eux et les applications reste toujours géré par la couche Transport. Ces composants vont néanmoins fonctionner selon le paradigme de la nouvelle architecture (de façon transparente aux connexions), mais seulement choisis de sorte à ce que les paquets qui en sortent correspondent exactement aux paquets habituels des protocoles sollicités. Les différents cas sont illustrés à la figure 2.2

Cas d'utilisation 2 : Applications legacy - Envoi et réception de données

L'envoi de données, à l'instar de l'initiation de la connexion, se fait à travers des primitives sockets classiques. La couche Transport doit s'occuper de la traduction de ces primitives en des appels aux primitives correspondantes des connexions de la nouvelle architecture. Dans le cas d'un envoi de données, ces dernières seront transférées aux buffers de la connexion sous-jacente via la primitive d'envoi (typiquement un send). Une fois dedans, elles seront traitées comme toutes autres données.

Dans l'autre sens, la réception de données se traduit par un appel système classique (typiquement un recv) qui est redirigé vers un appel de la connexion correspondante ; une fois les données présentes dans le buffer de réception, elle sont retournées à l'application.

Cas d'utilisation 3 : Applications aware - Demande d'un service

Les applications aware, conçues pour fonctionner avec la nouvelle architecture, n'utiliseront plus la notion de protocole, mais uniquement celle de service de Transport. Ces services seront non seulement invocables séparément selon le besoin, mais également paramétrables. Tel que décrit précédemment, une base de données destinée aux développeurs regroupe l'ensemble des services qui sont supportés à un instant donné par la couche Transport. Les développeurs peuvent ainsi les intégrer dans leurs requêtes.

Une fois la requête reçue, la couche Transport contacte le système distant afin d'explorer ses capacités, d'abord en termes de présence de la nouvelle architecture, et ensuite, si c'est le cas, en termes de services présents. Si la nouvelle architecture est supportée par le système distant, ce dernier lui transmettra un sous-ensemble des services requis par l'application initiatrice de la communication correspondant aux services supportés par le système distant. Ces services doivent à la fois être supportés par le système distant, mais aussi avoir été demandés en amont par une application serveur. Si les deux systèmes s'entendent sur les services de la communication en cours, une connexion sera instanciée de chaque côté, et une référence vers chacune de ces deux connexions sera retournée aux entités applicatives respectives côté client et côté serveur. Dans le cas où les services demandés ne sont pas disponibles (en partie ou en totalité) au niveau

du système distant, l'application cliente en est informée et peut par conséquent continuer avec le sous-ensemble des services ou annuler la connexion. Une fois la connexion créée, l'application interagira directement avec elle via le composant de connexion.

Cas d'utilisation 4 : Applications aware - Envoi et réception de données

Le transfert de données des applications aware est traitée de la même façon que pour les applications legacy. La seule différence réside dans les primitives invoquées qui sont, dans ce cas, directement les primitives send et receive de la connexion. Il n'y a donc pas de translation de primitives ni de communication indirecte avec la connexion. Le diagramme de la figure illustre un exemple d'interaction entre deux applications. La partie relative à l'interaction entre les deux couches Transport est réduite au minimum, elle sera décrite plus en détail dans le cas d'étude associé. Un exemple d'interaction entre deux applications est illustré à la figure 2.3.

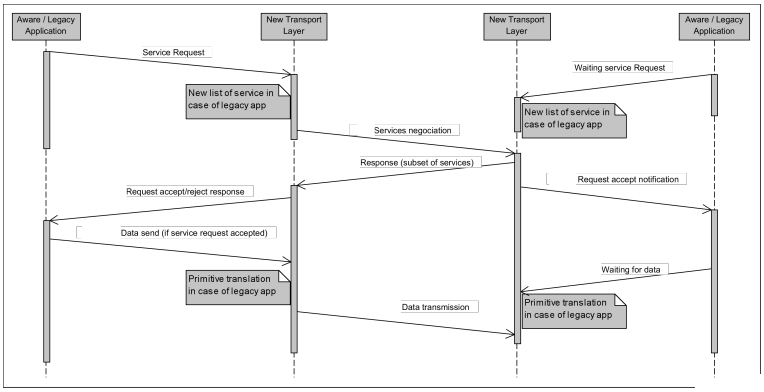

FIGURE 2.3 – Exemple d'intéraction entre deux applications

Cas d'utilisation orientés Système

Une fois les requêtes des applications reçues par la couche Transport, les interactions suivantes concerneront les deux entités de l'architecture Transport amenées à communiquer (côté client et côté serveur). Le type d'interaction, c'est-à-dire le comportement en conséquence, dépend de la capacité de chacun des systèmes en termes de support de la nouvelle architecture. Selon les cas, une entité de l'architecture Transport peut recevoir une demande de service dans le cas d'un système distant supportant la nouvelle architecture, ou d'un premier paquet d'un protocole dans le cas contraire. Dans les deux cas, cela sera suivi d'un transfert de donnée si l'établissement de la communication aboutit. Ces interactions sont illustrées sur la figure 2.4.

Cas d'utilisation 1 : Système legacy

Une entité de l'architecture de Transport peut donc être contactée par son entité paire (depuis système distant), qui peut supporter ou non la nouvelle architecture. Dans le cas d'un système legacy, la couche Transport reçoit le premier paquet du protocole utilisé par l'application distante. Il peut s'agir d'un paquet d'établissement de connexion pour un protocole orienté connexion, ou d'un premier paquet de données pour un protocole sans connexion.

Dans les deux cas, la couche Transport vérifie la présence d'une application serveur

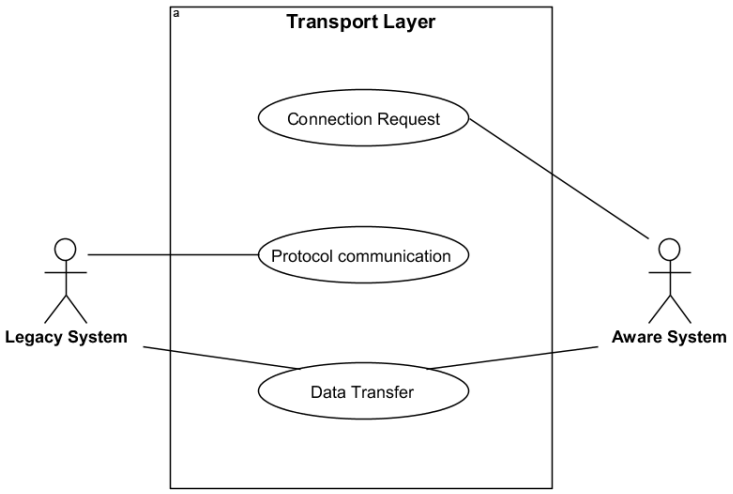

FIGURE 2.4 – Interactions entre deux systèmes distants

avec des services correspondant à ceux du protocole en question. Si c'est le cas, la communication se poursuit, sinon elle est refusée. Dans le premier cas, une nouvelle connexion est créée avec uniquement les services correspondants au protocole, sous réserve que l'application serveur soit en accord dans le cas où il ne s'agirait que d'un sous-ensemble de ces services. Cependant, la connexion sera créée avec une composition particulière dont les paquets résultants possèderont les mêmes entêtes que les paquets du protocole client.

A la réception, le transfert de données se fait de façon classique en passant les données par l'ensemble des composants de la connexion ciblée, et en les mettant dans le buffer de réception pour l'application serveur. Dans l'autre sens, les paquets, ayant les mêmes entêtes que le protocole client, seront transmis de façon transparente au système distant.

Cas d'utilisation 2 : Système Aware

Pour un système aware, le premier paquet correspond toujours une demande de connexion, il ne sera donc jamais un paquet de données. A la réception, le même processus se produit que pour les systèmes legacy. La seule différence est que les services considérés sont exactement les mêmes que ceux requis par l'application émettrice. Il n'y a donc pas de translation d'un protocole à un ensemble de services. Aussi, à l'émission des données vers le système distant, les paquets auront des entêtes personnalisés en fonction des services instanciés. Le diagramme de figure illustre un exemple d'interaction entre deux systèmes. La partie relative à l'interaction avec l'application est réduite au minimum.

2.4.2 Description comportementale

Dans cette section, nous décrivons en détail les phases élémentaires dans le comportement de la couche Transport que nous proposons. Il s'agit de la phase d'établissement de connexion et de la phase de transfert de données. La figure 2.5 illustre un exemple d'interaction entre deux systèmes.

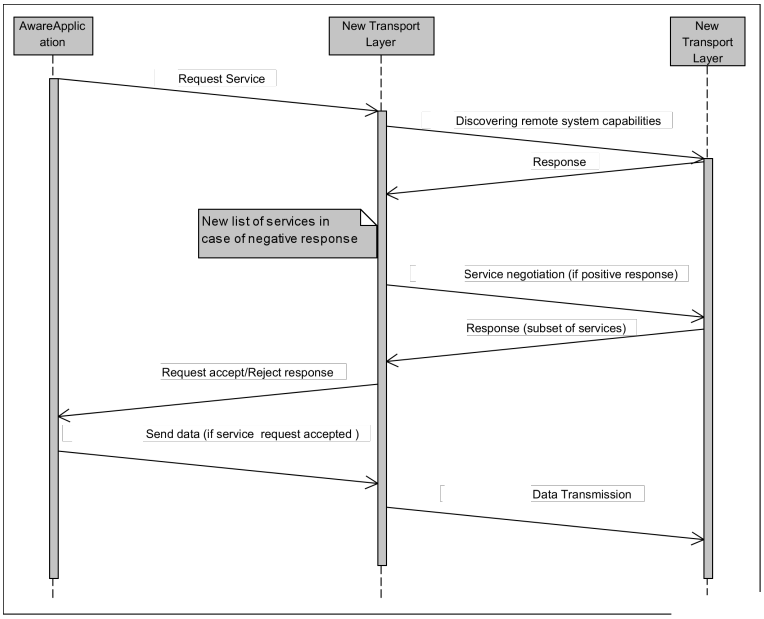

FIGURE 2.5 – Exemple d'interactions entre deux systèmes distants

Etablissement des connexions

La politique appliquée par défaut est d'établir une connexion entre les deux systèmes communicants avant tout transfert de données. Ce choix n'a pas uniquement pour objectif de garantir, pour l'application, que ses données seront envoyées à une entité active, mais aussi de permettre la gestion des compositions de service en tenant compte de l'incompatibilité pouvant exister entre les disponibilités de services dans les deux, ainsi que des compatibilités entre types de systèmes et d'applications. Nous considérons donc dans cette partie uniquement la politique d'établissement par défaut des connexions.

La gestion de l'établissement des connexions suit le modèle décrit dans le diagramme d'état de la figure 2.6. Certaines parties de ces processus relèvent des capacités d'extensibilité ou d'autonomie de l'architecture qui nécessitent une description plus approfondie, ce qui ne correspond pas aux objectifs de ce chapitre. Elles seront, par conséquent, décrites indépendamment dans les deux prochains chapitres.

Entre le client et le serveur, la seule différence réside dans la primitive à invoquer. Pour le reste, l'approche est exactement la même. L'application spécifie l'ensemble de ses services avec leurs éventuels paramètres. Ces services et leurs paramètres sont décrits dans la base globale des services, et sont invoqués en utilisant leurs identifiants. L'application différencie éventuellement les services obligatoires et les services optionnels. Cette différenciation vient simplifier la procédure de négociation des services en réduisant les échanges. En effet, de cette manière, l'échec d'une demande de connexion se produit uniquement lorsqu'un des services obligatoires n'est pas disponible. Pour les services optionnels, la couche Transport les déploie s'ils sont supportés par les deux parties de la communication ; dans le cas contraire, ils ne seront pas pris en compte. L'application sera tout de même informée si un service optionnel n'est pas disponible.

Une autre façon de faire serait de recevoir l'ensemble des services en un seul bloc, de

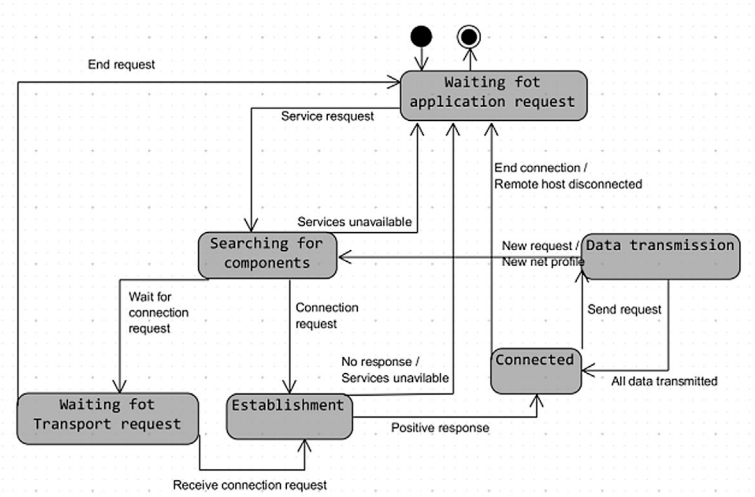

FIGURE 2.6 – Diagrame d'état de l'établissement de connexion

vérifier leur disponibilité et d'en informer l'application, qui aura par la suite le choix de continuer ou d'annuler sa requête. Toutefois, ce processus est plus lent car il implique notamment une attente, active ou passive, de l'application, ce qui ne constitue donc pas une bonne approche. Le traitement de la requête débute en local, par la gestion des dépendances et des incompatibilités. Ceci concerne les services qui doivent ou ne peuvent être déployés avec ceux présents dans la requête de l'application. Le traitement des dépendances et des incompatibilités sera décrit en détails au Chapitre 4.

La couche Transport, plus précisément l'interface application, complète d'abord la requête par les services dont dépend ceux requis par l'application, s'ils n'y sont pas déjà. Par la suite, elle recherche d'éventuelles incompatibilités entre l'ensemble de ces services (y compris les dépendances). Dans le cas où au moins deux services obligatoires sont incompatibles, la requête de l'application est rejetée ; dans le cas contraire, l'interface passe à la phase suivante.

Cette phase consiste à vérifier la disponibilité des services en local en parcourant la base locale des composants. Si tous les services sont disponibles, l'interface contacte le système distant avec la liste des services complète (incluant les dépendances) pour vérifier leurs disponibilités. Un processus équivalent est lancé au niveau du système distant, et une liste comprenant un sous-ensemble (qui peut être complet) des services requis est renvoyé au système client. Si tous les services obligatoires sont disponibles, un retour positif est renvoyé à l'application, avec éventuellement les services optionnels manquants, et une connexion est créée. Dans le cas contraire, c'est-à-dire si un des services obligatoires n'est pas disponible, la requête de l'application est rejetée.

Du côté du serveur, la procédure est identique. L'interface application reçoit une requête de l'application (attente d'une connexion), effectue toute les vérifications, mais uniquement en local. Dans le cas où il n'y a pas d'incompatibilité et que tous les services sont disponibles, l'interface se met en attente d'une requête correspondant à ces services, dans le cas contraire, la requête est rejetée.

Transfert de données

Le transfert de données se fait via les buffers d'émission et de réception, indépendamment de tout traitement par les composants de communication. C'est donc, vu de l'application, une simple mémoire partagée, avec des primitives de lecture et d'écriture.

Pour la connexion en interne, c'est un peu plus sophistiqué. Le buffer gère la taille des paquets selon les requêtes de l'assembleur, la gestion de sa mémoire lui-même, et la gestion des autorisations de suppression. Nous exposerons toutes ces fonctionnalités dans les chapitres suivants. Ici, nous disons simplement que le gestionnaire de buffer assure plusieurs fonctionnalités pour permettre l'interaction entre l'application et la connexion qu'elle utilise, et par ailleurs, que les applications, à l'instar des composants de communication, n'ont pas d'accès direct au buffer.

Pour l'émission de données, l'application dépose ses données dans le buffer d'émission ; elles sont ensuite récupérées en bloc par le composant d'assemblage qui crée les paquets en passant les données par les différents composants et récupérer les éventuels entêtes. Lorsque le paquet est prêt, il est transmis à l'interface réseau pour transmission effective.

En réception, le processus inverse est appliqué. Les données sont reçues par l'assembleur depuis l'interface réseau, traités par les composants de communication dans l'ordre inverse et stockés dans le buffer de réception de l'application. Lorsque le buffer d'émission est plein, l'application est bloquée en attente. Pour le buffer de réception en revanche, les données sont détruites dès qu'il soit plein, ce qui ne devrait pas se produire quand un service de contrôle de flux est déployé.

L'interface réseau gère de son côté le multiplexage et le démultiplexage des paquets depuis et vers les connexions. Elle gère aussi en parallèle les paquets destinés à la partie gestion de l'architecture. L'interface assure un partage équitable entre l'ensemble des connexions. Un partage différencié peut cependant être envisagé pour garantir une certaine qualité de service à certains types d'applications, mais cela ne sera pas traité dans ce travail.

2.5 Conclusion

Nous avons présenté dans ce chapitre une description à la fois générale et détaillée de l'architecture que nous proposons pour la couche Transport. Nous avons dressé, au départ, l'ensemble des limites de l'architecture actuelle et les besoins qui en résultent. Après une description générale de la solution que nous proposons, nous avons ensuite présenté en détail la composition de son architecture ainsi que ses comportements de base.

Le cadre architectural présenté jusque-là permet de répondre à l'ensemble des limites considérées dans ce travail. La séparation des parties de gestion et de communication, les composants d'intégration et de décisions basés sur des bases de connaissances permettent assurent l'évolution de la couche Transport en lui offrant des capacités d'extensibilité et d'adaptation. Dans les deux prochains chapitres, nous présenterons, plus concrètement, la façon avec laquelle ces capacités sont assurées. Il s'agit de paradigmes

et d'approches de conception, de modèles et d'algorithme qui permettent de gérer les différents composants de l'architecture ainsi que les interactions entre eux.

Enfin, le dernier chapitre illustre les détails d'implémentation et déploiement. Précisons ici que l'architecture a été développée dans son intégralité et testée dans le cadre d'un cas d'étude. Nous étudierons ainsi au Chapitre 5 les tests et les résultats obtenus.

Chapitre 3

Extensibilité de la couche Transport

Au chapitre précédent, nous avons décrit l'architecture proposée pour la couche Transport. Les différents composants de l'architecture ont été présentés par leurs principes de fonctionnement et leurs comportements de base. Ces composants assurent l'évolution de la couche Transport par des capacités d'extensibilité et d'adaptabilité. Dans ce chapitre, nous traitons plus en détails de l'extensibilité de l'architecture. Les aspects relevant de l'adaptabilité sont traités dans le chapitre 4.

Le cadre architectural présenté au chapitre 2 permet, en lui-même, d'assurer une certaine extensibilité pour l'architecture, notamment en rapport avec les nouvelles fonctionnalités de communication, que ce soit pour gérer de nouveaux services / composant de Transport ou de nouvelles technologies réseau. Cependant, la notion d'extensibilité est beaucoup plus large, et doit aussi couvrir les composants de gestion et les interactions au sein de l'architecture. En réponse à ces besoins, nous présentons ici les composants, paradigmes et techniques qui garantissent l'extensibilité de notre architecture.

Comme expliqué précédemment, la notion d'extensibilité peut avoir plusieurs significations. Nous l'entendons ici comme la capacité d'évoluer, par ajout ou modification de nouveaux composants, sans aucun impact sur les autres composants de l'architecture, les systèmes ou les applications. C'est cette capacité d'extensibilité qui lui permettra d'évoluer dans le temps. En effet, nous estimons que, parmi toutes les limites de la couche Transport qui l'empêche d'évoluer, l'extensibilité est la plus importante. Par ailleurs, nous estimons que le fait de ne pas traiter cette limite qui a fait que les architectures précédentes n'ont pas pu être déployées.

Le chapitre présente en premier les approches et paradigmes de conceptions utilisés dans l'ensemble de l'architecture. Ensuite il décrit les outils qui permettent l'extensibilité de l'architecture vis-à-vis des systèmes, l'extensibilité des composants de communication et de gestion, et enfin l'extensibilité vis-à-vis des applications.

3.1 Approches de conception

Le premier élément ayant un impact sur le degré d'extensibilité de l'architecture réside dans son approche de conception. Tel que présenté au chapitre 2, nous avons opté pour un modèle orienté services et basé composants, associé à une inconscience sémantique des composants de gestion des éléments qu'ils manipulent.

Ce modèle, qui conduit à attribuer un composant logiciel à chaque service de Transport, permet de rendre la gestion de l'architecture et la construction des compositions inconsciente de la sémantique des composants. Pour chaque situation, une composition est créée en tenant compte du contexte réseau en cours, du besoin de l'application et de la description de chaque composant. Ainsi aucun lien sémantique n'existe entre la partie de gestion de l'architecture et les composants de communication. Cette approche permet par la suite d'ajouter de nouveaux composants à l'architecture sans avoir à modifier quoi que ce soit dans sa partie de gestion, les nouveaux composants étant alors traités de la même façon que les composants déjà existant.

L'approche orientée services permet par ailleurs de séparer les services de Transport, tels qu'ils sont vus par les applications, de la façon dont ils sont réalisés concrètement par la couche Transport. Ceci lève la dépendance classique qui existe entre les applications et les protocoles de Transport. Rappelons en effet qu'actuellement les applications sont écrites pour un protocole particulier, ce qui rend leur utilisation impossible avec d'autres protocoles sans modification du code de ces applications. Dans notre architecture, les applications demandent uniquement des services, et la façon dont ceux-ci sont implantés leur est complètement transparente. L'approche par composant offre une indépendance logicielle entre les composants offrant les différents services. Les modifications portant sur un composant n'affectent ainsi ni les autres composants de communication, ni les composants de gestion. Enfin, l'inconscience sémantique permet aux composants de gestion de pouvoir tenir compte des nouveaux services et composants de communication sans avoir à être modifiés.

Dans la suite de cette section, nous approfondissons l'ensemble de ces paradigmes (section 3.1.1), leur application à notre proposition d'architecture de la couche Transport (section 3.1.2), et la façon avec laquelle ils permettent d'assurer son extensibilité (section 3.1.3).

3.1.1 Description des paradigmes

L'approche suivant laquelle l'architecture a été conçue contribue en grande partie à résoudre les problèmes d'extensibilité dont souffre la couche Transport actuelle. Néanmoins, d'autres outils de conception, plus techniques, ont été nécessaires pour assurer cette extensibilité. Nous les étudions dans la suite de ce chapitre.

Approche orientée service (SOA)

Le modèle orienté-service a été proposé dans le but de permettre la portabilité des applications dans des environnements hétérogènes. Il se base sur le principe de la séparation entre un service et sa réalisation [58]. Les acteurs externes ayant à interagir avec un système, simple ou complexe, ont uniquement connaissance de la nature du service réalisé ainsi que de l'interface nécessaire pour l'invoquer (paramètres d'appel notamment). L'approche orientée services permet d'avoir un couplage très faible entre les différents composants d'un système, ou entre deux systèmes différents, en limitant autant que possible leurs interactions. Un couplage fort résulte soit d'une interaction importante entre les composants, ou d'une spécificité vis-à-vis d'une approche particulière de la réalisation du service en question [59].

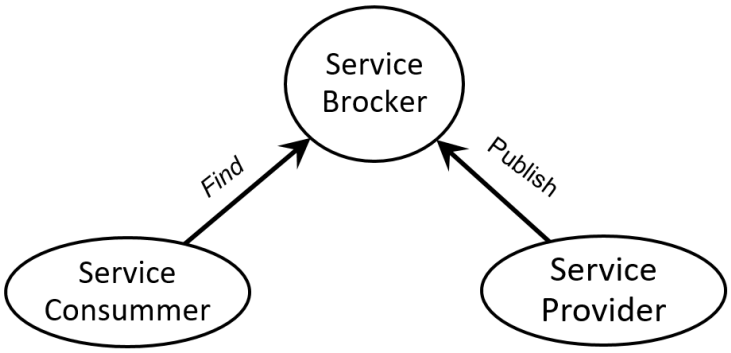

FIGURE 3.1 – Schéma d'une architecture orientée-service

L'approche orientée services s'apparente au paradigme de la programmation orientée objets qui sépare les interfaces des classes et leur implémentation concrète. Les services s'apparentent donc à des objets mais à plus grande échelle. Il peut s'agir de composants différents d'un même programme, de plusieurs programmes d'un même système, ou même de systèmes distants complètement différents. En outre, la notion d'objet s'applique au niveau du codage (code source) alors que la notion de service s'applique au niveau de programmes finis (compilés).

Un faible couplage entre les différents services permet de pouvoir maintenir plus facilement les composants qui les implémentent, dès lors que les modifications qui y sont apportées n'affectent pas les autres services. La facilité de maintenance dont nous parlons ici concerne la minimisation de l'impact de la maintenance d'un composant sur le reste du système.

La notion de "service", en elle-même, est difficile à déterminer. Aucun consensus n'a pu être établi [60]. Globalement, un service est la fonction de plus haut niveau dont l'utilisateur de service a besoin de connaitre la description. Un service doit donc être aussi abstrait que possible pour ne porter comme description que le minimum de détails dont l'utilisateur a besoin pour l'invoquer [58].

Dans leur application courante, les architectures orientées services utilisent un modèle par "inscription", suivant lequel un acteur intermédiaire (typiquement le Service Broker de la figure 3.1) gère le lien entre les fournisseurs et les consommateurs de services [61].

Comme illustré sur la figure, l'utilisateur ou le consommateur de services contacte cet acteur intermédiaire afin de récupérer la liste des services offerts ainsi que les fournisseurs associés. Dans d'autres modèles, l'acteur intermédiaire peut ne pas se limiter la publication des services, mais gère aussi les requêtes et les réponses entre les utilisateurs et les fournisseurs de services.

Approche basée composants

Le principe de l'approche par composants est plus simple. Il s'agit de séparer, ou " modulariser ", un programme complexe en un ensemble de sous programmes relativement simples. Il s'agit d'un paradigme générique appliqué dans tous les domaines d'ingénierie. Dans notre cas, il s'agit notamment des modules compilés (exécutables)

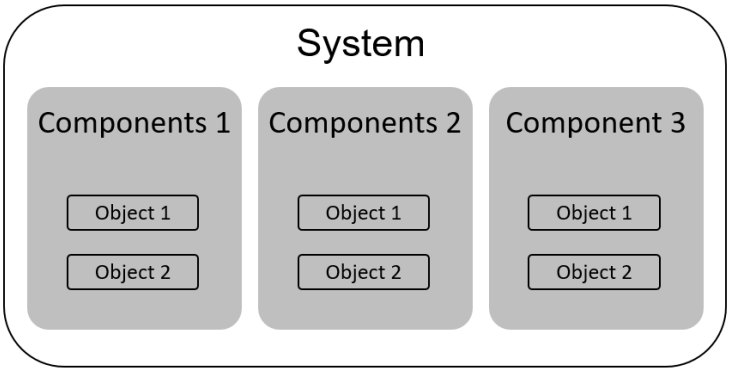

FIGURE 3.2 – Schéma d'une architecture basée-composants

d'un programme. Au lieu d'être établi en un unique bloc sous la forme d'un seul fichier binaire, le programme est écrit et compilé en plusieurs fichiers de sortie (figure 3.2). Cette approche permet une meilleure lisibilité du programme et facilite sa maintenance. En outre, dans le cadre de gros projets, cette approche facilite le partage de tâches entre les différents développeurs, et permet la réutilisation des composants [62].

Cela étant, la facilité de maintenance que l'approche par composant permet d'offrir n'a aucun rapport avec ce que permet l'approche par services. En effet, il se peut qu'un programme correctement modularisé puisse avoir des problèmes de maintenance si ses composants sont fortement liés. Par exemple, lorsque la communication entre les composants est synchrone (sous forme d'appel de procédure ou de méthodes), des modifications portant sur les fonctions ou leurs interfaces peuvent entraîner une chaîne d'autres modifications dues aux dépendances.

La notion de composant est très ancienne et date de la fin des années 60. Depuis la parution des architectures orientées services, les deux concepts ont été fortement associés [63]. Cependant, la différence est bien claire, un composant peut offrir un ou plusieurs services et un service peut être offert par un ou plusieurs composants. De plus, les services sont appliqués à plus grande échelle ; on parle généralement de composants au sein d'un même programme, et l'on parle de services entre deux programmes différents (et d'objets au sein d'un même composant).

Inconscience sémantique

La notion d'inconscience sémantique (semantic awareness) est relativement récente, et sa définition exacte n'est pas encore établie. Le concept a néanmoins été utilisé dans plusieurs travaux, relevant notamment des plateformes Web. Il a également été proposé pour des systèmes autonomes afin de les rendre plus flexibles et plus extensible [64]. Le principe est que la sémantique des éléments considérés (dans notre cas : par le gestionnaire autonomique) ne doit pas être incluse dans le composant gérant cette autonomie, de sorte à minimiser le couplage entre le composant qui gère l'autonomie, et les composants gérés [65].

Cette sémantique doit toutefois être gérée quelque part pour qu'elle puisse être considérée dans les choix du gestionnaire autonomique. Elle doit être décrite dans une base de connaissance externe selon des anthologies et des règles formelles, relevant le plus

souvent de la théorie des langages, comme dans [66][67]. Cette approche nécessite un protocole syntaxique rigoureux entre le gestionnaire autonomique et le gestionnaire de la base de connaissance. Cela peut se révéler difficile à réaliser lorsque le gestionnaire autonomique est censé communiquer avec des acteurs externes et hétérogène. Par ailleurs, bien que l'approche assure l'extensibilité des composants d'un système autonome, elle n'assure pas l'extensibilité de l'approche autonomique elle-même. Si le protocole est amené à être modifié ou à évoluer, c'est toute la chaîne qui sera affectée.

L'autre approche consiste à déléguer la gestion de cette sémantique aux composants autonomes [65][68]. Les composants autonomes traduisent la description sémantique vers une description basique de sorte à ramener le problème de l'autonomie à un problème d'optimisation de base (tri, optimum, classification ...). Le problème traité par le gestionnaire autonomique devient donc indépendant de la sémantique des données traitées, et devient ainsi plus flexible et plus extensible.

En dépit du fait que l'approche n'ait pas encore été largement utilisée, et que ses aspects théoriques n'aient pas été formellement définis, son impact sur les architectures peut être conséquent. En effet, en séparant la sémantique des éléments communs, on réduit les dépendances et on évite ainsi une importante quantité de travail liée aux composants autonomes qui dépendent du composant commun en question.

3.1.2 Application à la couche Transport

La notion de décomposition a été appliquée par les toutes premières architectures proposées au niveau Transport (voir chapitre 1). Les composants ont été associés à la notion de "micro-protocole" et permettaient une haute configurabilité de ces architectures par une composition dynamique de ces micro-protocoles. L'approche orientée services est quant à elle nouvelle dans notre architecture. Elle a cependant été initiée par les travaux de [46] décrits au chapitre 1.

Dans notre cas, nous avons adopté une approche orientée services entre les applications et la couche Transport, et une approche par composants à l'intérieur de la couche elle-même. Pour les services, nous avons adopté une démarche relativement différente de ce qui se passe habituellement dans des architectures orientée services. D'abord, la publication des services ne passe pas par un acteur externe, mais par une base de données globale partagée (base globale des services). Compte tenu de la quantité relativement limitée des services offerts par la couche Transport, et du rythme de son évolution qui est aussi relativement lent, le besoin d'une découverte dynamique des services ne se présente pas. Par ailleurs, la sémantique de ces services n'est pas censée être connue par les applications, les services sont donc à destination des développeurs et les applications n'ont pas à les découvrir dynamiquement.

Par ailleurs, nous avons posé l'hypothèse d'un service par composant. Ce choix facilite le déploiement des composants et limite les liens entre eux. Il devient ainsi possible de modifier ou d'intégrer un composant sans impact sur les autres. Ceci permet aussi, indirectement, d'avoir une indépendance logicielle entre les services de la couche, c'est-à-dire qu'un seul composant logiciel est à modifier pour maintenir un service donné.

Les services sont exposés aux applications via la base globale des services, mais les composants de communication sont connus uniquement en interne, stockés dans la

base locale des composants sous forme de correspondances composant/service. D'autres bases locales existent aussi au sein de l'architecture pour permettre l'extensibilité de certains composants ou certains paramètres particuliers ; nous en parlerons un peu plus loin.

L'approche par services permet ainsi l'extensibilité de la couche Transport vis-à-vis des applications, et l'approche par composants permet l'extensibilité des composants de communication. Il reste toutefois un dernier aspect de l'extensibilité concernant les composants de gestion eux-mêmes. En effet, les composants de gestion doivent eux-aussi être extensibles pour pouvoir suivre l'évolution des services et composants, et autres éléments (paramètres, valeurs monitorées, ...). Si ces composants de gestion doivent être modifiés à chaque fois qu'un nouvel élément doit être pris en compte, on se retrouve alors au même problème de départ, qui fait que le coeur de la couche Transport doit être impliqué dans toute opération d'extension ou de maintenance.

L'extensibilité des composants de gestion est assurée par le principe d'inconscience sémantique vis-à-vis de tout élément susceptible d'évoluer. Ainsi, le composant de décision est complètement inconscient de ce que peuvent signifier les paramètres de surveillance de réseau (délai, débit, ...). Lorsque l'on veut surveiller un autre paramètre, il suffit de l'ajouter à la liste correspondante et il sera pris en compte immédiatement. Cet aspect d'inconscience apparaît à plusieurs reprises dans notre architecture ; nous en ferons référence dans la suite à chaque endroit où il est appliqué.

3.2 Extensibilité vis-à-vis des systèmes

La couche Transport étant une partie intégrante du système d'exploitation, elle est fortement dépendante de ses contraintes. Tel qu'analysé au Chapitre 2, un des obstacles face à son évolution réside dans les limites des systèmes d'exploitation. En effet, ceux-ci sont tellement complexes et surchargés qu'ils limitent fortement la quantité des protocoles qu'ils déploient, du moment que ceux-ci arrivent à assurer l'essentiel du besoin.

Par ailleurs, cette dépendance aux systèmes a pour conséquence un autre obstacle face à l'extensibilité qui réside dans l'opération de maintenance elle-même. Les mises à jour du noyau sont extrêmement coûteuses, et les systèmes ne peuvent pas se permettre d'en faire autant que la couche Transport a besoin pour rester à jour face aux évolutions applicatives et réseaux.

Dans cette optique, le second besoin pour assurer l'extensibilité de la couche Transport est de lever cette dépendance avec le noyau des systèmes d'exploitation au sein duquel les protocoles de Transport sont classiquement logés.

Notre approche repose tout d'abord sur une séparation des parties évolutives et des parties statiques de notre architecture, pour ne garder que ces dernières dans le noyau, tout en assurant une exécution de l'ensemble dans l'espace noyau. Ensuite, l'évolution des parties implantées dans le noyau est assurée de façon autonome suivant le principe d'inconscience sémantique introduit précédemment. Nous détaillons ci-après ces deux aspects.

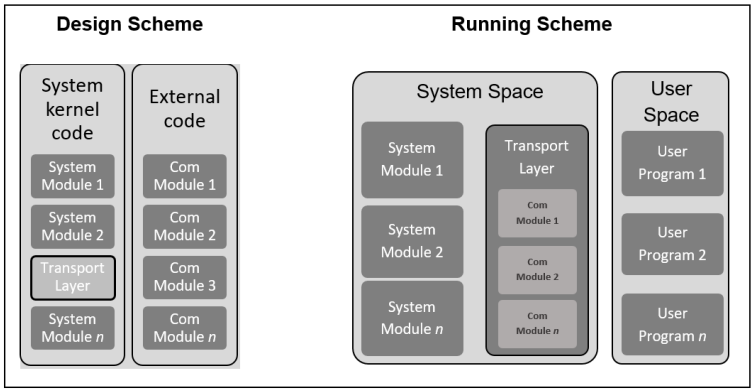

FIGURE 3.3 – Principe de la sépartion du noyau

3.2.1 Indépendance au noyau

Le constat que l'on peut tirer des différents travaux effectués au niveau Transport et des propositions résultantes (cf. Chapitre 1) est que les seules parties qui nécessitent une évolution sont celles qui gèrent la communication proprement dite. Les parties responsables de la gestion du protocole lui-même restent, quant à elle, inchangées, et sont pratiquement les mêmes entre tous les protocoles. Par exemple, les différentes évolutions du protocole TCP décrites au chapitre 1 concernent uniquement les mécanismes de communication (fiabilité, contrôle de congestion, ...). Les autres éléments du protocole, tels que la gestion des buffers, la numérotation des paquets, le format des en-têtes, etc. restent relativement les mêmes.

Suivant ce principe, nous avons séparé la partie gestion de la couche Transport de la partie communication, de sorte à ce que seule la première partie fasse partie du noyau. Les composants de gestion n'auront pas à évoluer, sur le plan algorithmique, dans le futur et n'auront donc pas de conséquence sur le noyau. Les composants de communication, quant à eux, peuvent évoluer du moment qu'ils sont implémentés indépendamment du noyau. La partie gestion sera lancée par défaut au chargement du système, et existera en une seule instance. Les composants de communication seront chargés de façon dynamique selon les besoins des applications. Ceci fait que les composants de communication seront tout de même exécutés dans l'espace noyau, comme des modules de la partie gestion, bien qu'ils soient conçus indépendamment. Ce modèle est illustré dans le schéma de la figure 3.3.

La séparation des composants de communication du noyau permet leur compilation de façon indépendante, et les composants logiciels résultants sont stockés et chargés au besoin. L'intégration de ces nouveaux composants passe par le module d'intégration (de la partie gestion) afin d'effectuer les vérifications nécessaires ; nous en parlerons plus en détail dans les sections suivantes. Techniquement, la gestion du chargement dynamique des composants vers le noyau est assez complexe. Plusieurs détails sont à considérer, en rapport avec le lancement, l'identification et le chargement des composants, suivant les besoins des applications, la gestion de l'usage multiple des composants, et bien d'autres paramètres. Nous exposerons ceci plus en détail au chapitre 5.

On note enfin que la décomposition permet la réutilisation des composants qui ne se-

ront chargé qu'une seule fois, même s'ils sont utilisés par plusieurs applications. Cette propriété n'a pas de rapport avec l'extensibilité, elle constitue toutefois une optimisation du fonctionnement global de la couche Transport.

3.2.2 Contraintes de mise à l'échelle

En plus des possibilités de maintenance et d'extension qu'offre la séparation entre la gestion et la communication, elle permet aussi aux développeurs de systèmes d'exploitation de s'affranchir de la contrainte du nombre de solutions de Transport intégrables dans le noyau. En mettant les composants de communication en dehors du noyau, les limites face au nombre de protocoles à déployer ne se pose plus, du fait qu'il ne s'agit plus d'étendre le noyau. Il est par conséquent possible de mettre autant de solutions que souhaité, et l'on se débarrasse ainsi de la contrainte des solutions générale. Il est possible cette façon d'avoir autant de solutions spécifiques qu'il y a de contextes, notamment en termes de réseau, et de continuer à fonctionner à la manière d'une solution générale. En outre, la réutilisation des composants, comme évoqué précédemment, permet de ne charger qu'une seule instance de chaque composant, et seulement au besoin qui puis est. Outre le fait de lever la contrainte sur le nombre de solutions intégrables dans le noyau (lors du développement du système), on lève ainsi également la contrainte durant le temps d'exécution du système.

3.3 Extensibilité des composants de communication

Les composants de communication constituent la partie évolutive de la couche Transport, ils sont amenés à être fréquemment modifiés ou étendus. Par conséquent, leur extensibilité est la plus importante dans l'architecture. Elle est assurée, d'abord, à travers les principes de décomposition et d'association service/composant, ensuite, par le composant d'intégration et la base locale des services, et enfin, par des approches techniques d'implémentation. Cette section décrit l'ensemble de ces outils.

3.3.1 Décomposition par services

Le premier outil pour l'extensibilité des composants de communication réside dans le principe de décomposition service / composant. Le fait d'avoir des composants indépendants d'un point de vue logiciel permet une maintenance sans impact sur le reste de l'architecture.

L'exigence derrière cette décomposition, qui exige que tout composant logiciel doit offrir un service dans son intégralité, contribue aussi à l'extensibilité. Le choix de ces composants et leur association aux services de l'application devient alors une problématique facile à résoudre par le biais d'une base de données.

Aussi, les services sont associés aux applications, et les composants sont associés aux contextes réseau. Ainsi, le choix des compositions se limite à sélectionner le meilleur composant pour le réseau sous-jacent parmi tous ceux qui offrent le même service requis par l'application. Le modèle d'association services/composants est illustré sur la figure 3.4.

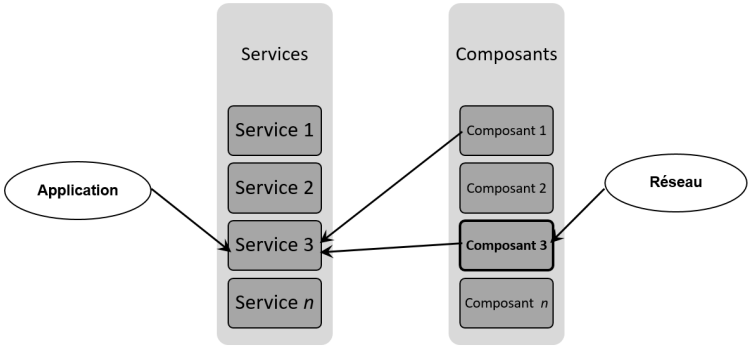

FIGURE 3.4 – Modèle d'association service/composant

3.3.2 Composant d'intégration et base locale des composants

Le composant d'intégration de la partie gestion de l'architecture permet l'intégration des nouveaux composants et la mise à jour des bases de données. Le passage, pour les nouveaux composants de communication, par un composant dédié à leur intégration permet d'effectuer les vérifications d'intégrité et de conformité nécessaires au maintien d'un état cohérent de la couche Transport.

Il s'assure que les nouveaux composants fournissent toutes les informations nécessaires à leur gestion à l'intérieur de l'architecture. Il s'agit principalement du service que le composant en question fourni pour l'application, de ses plages d'adaptabilité pour différents paramètres réseaux, des valeurs de monitoring (au sens délai, taux de perte, etc.) qu'il fournit et de celles dont il a besoin. Par ailleurs, il assure le respect de certaines exigences en rapport avec l'adaptabilité de ces composants, qui seront décrites au chapitre 4.

Le processus de décision (décrit au Chapitre 4) se base directement sur la base locale des composants, sans avoir de connaissances préalables sur les services, les composants ou encore leurs paramètres. Cette approche, qui consiste à utiliser une base de données externe pour décrire les liens entre les services et les composants, utilisé souvent dans les approches orientée services, offre une forte extensibilité en réduisant le processus de décision à un problème d'optimisation classique, qui n'a besoin d'aucune information d'ordre sémantique sur les éléments du problème.

3.3.3 Interactions entre composants

Les composants de communication sont amenés à échanger des informations de monitoring pour pouvoir fonctionner. Ces échanges ne sont pas très visibles dans les protocoles classiques parce qu'ils se font de façon triviale, mais le principe est le même. Dans notre proposition, un contrôleur de congestion, par exemple, a besoin de connaître le taux de pertes ou le RTT, ces valeurs sont fournies par le gestionnaire de fiabilité et des retransmissions.

Pour assurer un fonctionnement correct et cohérent de la couche Transport après la décomposition des différents services, il faudra conserver tous ces échanges pour réutiliser les informations déjà recueillies. Une autre approche consisterait à ajouter des mécanismes au différents composants afin qu'ils obtiennent eux-mêmes ces informations ;

nous verrons cependant dans le prochain chapitre pourquoi ce n'est pas une bonne approche. Nous avons par conséquent adopté une approche par échanges d'information par le biais de mémoires partagées.

Le principe est de créer un espace mémoire partagé pour chaque valeur de monitoring est de l'associer aux composants qui auront à s'en servir, et ce en lecture et/ou en écriture. Comme expliqué au chapitre précédent, ces mémoires sont créées en fonction des composants déployés dans la connexion considérée, en se servant des informations de description de chaque composant.

Cette approche permet une extensibilité à la fois des composants, des connexions, et des valeurs de monitoring. La création des mémoires partagées dépend des composants déployés. Par conséquent, quel que soit le nombre et le type des composants d'une connexion, il est possible de lui créer les mémoires partagées nécessaires pour faire communiquer l'ensemble des composants au sein d'une connexion. Ceci vient du fait que le nombre de mémoires considérées, des composants qui les partagent, et de ceux qui les utilisent, n'est pas figé dans les connexions, mais créé de façon dynamique. Notons que les connexions n'ont pas de connaissance sur la sémantique des composants ni des valeurs partagées : il ne s'agit pour elles que d'identifiants et de références vers ces mémoires.

Cette approche offre aussi la possibilité de faire évoluer autant que nécessaire les valeurs de monitoring. Ces valeurs sont contenues dans la description des composants et ne sont pas censées être connues à l'avance. Pour chaque connexion, les composants sont parcourus afin d'identifier de façon dynamique les valeurs nécessaires.

Enfin, la communication entre ces composants est complètement transparente. Les composants n'ont aucune connaissance des composants qui partagent ces valeurs ni de ceux qui les utilisent. Ils indiquent uniquement qu'ils ont besoin d'une ou de plusieurs valeurs et la couche Transport s'occupe de les leur procurer. En réduisant ainsi l'interaction entre les composants et par conséquent leurs dépendances, il n'y aura aucune conséquence, suite à la modification d'un composant, sur les autres composants qui utilisent ses valeurs partagées.

3.3.4 Invocation des composants

Comme nous l'avons décrit au chapitre 1, plusieurs modèles ont été adoptés pour invoquer les composants d'une architecture de Transport. La différence entre ces modèles réside principalement dans leur influence sur les performances globales de la communication. Nous discuterons dans cette section d'autres considérations que les performances. En effet, la manière d'invoquer les composants définit les capacités d'une architecture à évoluer, notamment pour ce qui concerne la gestion des évènements.

L'approche hiérarchique convient aux fonctions qui agissent sur les paquets de données et doivent être appelées systématiquement, dans un ordre précis, pour chaque paquet émis ou reçu [41]. Pour des fonctions agissant sur des événements comme la perte de paquets ou l'expiration d'un timer, l'approche événementielle est plus appropriée. Suivant cette distinction, l'architecture ETP [45] a proposé un couplage entre les deux approches pour obtenir des performances optimales.

Cependant, aucune des approches précédentes ne peut assurer une extensibilité de son architecture. Les événements doivent être figés, ou à défaut connus (sémantiquement) par le gestionnaire des événements afin de les relier aux composants associés. Pour le modèle hiérarchique, c'est la sémantique des composants qui doit être connue par la partie gestion de l'architecture.

Dans notre solution, nous avons adopté un nouveau modèle qui associe les deux approches hiérarchique et événementielle, mais pour chaque composant. En d'autres termes, chaque composant est invoqué de façon hiérarchique mais ne fonctionne que suite à un événement dont il est seul à connaître la sémantique. Ce modèle est basé sur l'idée de déléguer à chaque composant non seulement le traitement des événements, mais aussi leur détection. De cette façon, il suffit, pour ces composants, de disposer d'un mécanisme qui leur permet de gérer le temps afin de détecter les événements temporels. En résumé, le modèle fait appel à une invocation hiérarchique et à un comportement événementiel.

Une fois l'événement détecté, il sera traité de façon classique par le même composant. Dans ce modèle, seuls les composants de communication possèdent une connaissance sémantique des événements les concernant. Les connexions, l'assembleur, ou tout autre composant de gestion sont complètement inconscients de cette sémantique. Les seuls événements connus sont l'arrivée des données depuis le réseau ou depuis l'application.

Les composants disposent tous de fonctions pour gérer séparément le cas de traitement de données et le cas de traitement des événements. Mais tout composant doit être capable de gérer ces événements au cas où ils se produisent pendant le traitement des données. Les fonctions de gestion des événements sont donc un sous-programme des fonctions de gestion de données, ou en d'autres termes, les fonctions de gestion de données possèdent une copie de celle de gestion des événements. Pour cela, tous les composants implémentent une interface commune afin d'assurer l'existence de l'ensemble de ces fonctions.

Ce modèle permet de disposer d'autant d'événements que nécessaires, et l'évolution de ces événements vient automatiquement à mesure que de nouveaux composants sont intégrés dans la couche Transport. Notons que ce principe de limitation de toutes les fonctions conscientes de la sémantique des données et des traitements aux niveaux des composants séparés du noyau, assure une extensibilité complète des composants quant à la gestion des événements.

De même, la construction des en-têtes suit le même principe que la création des mémoire partagées. L'assembleur récupère (indirectement) les informations sur la taille des en-têtes dont chaque composant a besoin, et pour chaque paquet, il réserve assez de taille pour contenir l'ensemble de en-têtes de tous les composants, en incluant la partie utilisée par l'assembleur lui-même. L'information relative aux en-têtes, elle aussi, n'est pas figée mais récupérée de façon dynamique depuis la description des composants.

3.3.5 Autres éléments extensibles

Outre les composants de communication offrant les services à solliciter par les applications, d'autres éléments de l'architecture sont susceptibles d'évoluer à mesure que

de nouveaux composants sont intégrés. Ces éléments concernent soit des composants particuliers, soit les paramètres de ces composants.

Composants par défaut : les composants par défaut sont des composants qui offrent des services internes à la couche Transport (c'est à dire à usage de l'architecture elle-même), ou bien au profit du réseau ou des applications, qui sont instanciés pour toutes connexion ou selon le contexte, sans avoir à être requis au préalable par les applications.

Le choix de ces composants peut être fixé à l'avance, ou dépendre de la politique d'administration de la couche Transport. Le seul élément fixé est le composant d'assemblage. En toute logique, ce composant devrait normalement faire partie intégrante des connexions, et être instancié automatiquement avec elles. Cependant, le fonctionnement de ce composant dépend du type de l'application ainsi que celui du réseau. Des services peuvent ainsi y être intégrés (comme l'algorithme de Naggle), ou des optimisations selon la nature de réseau peuvent y être appliquées (comme la taille et la fréquence d'émission des paquets [1]). L'assembleur est par conséquent considéré comme tout autre composant de communication offrant un service interne de création de paquets ainsi que d'autres éventuels services. Les connexions attendent donc un composant d'assemblage au moment de leur instanciation, mais le choix de ce composant est effectué par le composant de décision à l'instar des autres composants de communication.

Les autres composants par défaut dépendent de la politique d'administration de la couche Transport selon son contexte d'application. Il peut s'agir d'un service orienté connexion pour éviter une occupation inutile de la bande passante, d'un contrôle de congestion pour protéger le réseau, ou encore d'un service de chiffrement dans des contextes d'application sensibles..

Composants de monitoring : nous verrons dans le prochain chapitre que le monitoring de réseau est effectué par les composants de communication eux-mêmes. Dans certains cas, les composants ont cependant besoin d'utiliser des valeurs de monitoring qu'ils ne peuvent pas récupérer eux-mêmes. Ils doivent donc faire appel à d'autres composants pour leur procurer ces valeurs. Il n'est toutefois pas obligatoire que ces composants nécessaires au monitoring soient disponibles dans la composition, si les services qu'ils offrent (en plus de la fonction de monitoring) ne sont pas requis par l'application.

Dans de telles situations, des composants particuliers dédiés au monitoring sont utilisés. Ces composants sont gérés comme le reste des composants de communication avec comme service, dans leur description, le monitoring. Ils se peut donc qu'il y ait plusieurs composants pour la même valeur de monitoring, chacun adapté à un type de réseau particulier, sélectionnés par le module de décision.

A l'instar des composants habituels de communication, les composants de monitoring peuvent être étendus ou maintenus, sans impact sur le reste de l'architecture. De plus, étant traités comme des composants habituels, les composants de gestion n'ont pas à

1. Rappelons (cf Chapitre 2) que c'est le composant d'assemblage qui crée les paquets, récupère les données des buffers et invoque les composants appropriés C'est donc lui qui décide de la taille des paquets, de la fréquence d'envoi, ... Une optimisation concernant ces éléments doit donc être incluse dans le composant d'assemblage.

être modifiés pour tenir compte des nouveaux composants. Il s'agit simplement d'ajouter le service de monitoring dans la requête de l'application pour que le composant soit sélectionné et ajouté à la connexion. Ce processus est décrit en détail au chapitre 4.

Les valeurs monitorées sont elles aussi extensibles, à mesure que les composants de communication offrent plus de valeurs. Elles sont stockées dans une base dédiée, avec des informations sur les métriques notamment. Cependant, pour assurer la cohérence de l'architecture, ces valeurs ne sont ajoutées à la base que s'il existe au moins un composant de monitoring qui puisse les procurer. Par la suite, l'usage des composants de communication à la place des composants spécifiques au monitoring n'est qu'une optimisation supplémentaire par le composant de décision.

3.4 Extensibilité des composants de gestion

Dans la section précédente, nous avons présenté les éléments de notre architecture qui permettent l'évolution des composants de communication. Ces derniers peuvent ainsi être modifiés ou intégrés à la couche Transport et pris en compte sans impact sur les autres composants. Cette prise compte fait que ces derniers peuvent gérer des composants (de communication) dont ils n'avaient pas connaissance au moment de leur conception.

Par conséquent, les composants de gestion doivent évoluer aussi en parallèle des composants de communications. Cependant, et contrairement à ces derniers, les composants de gestion se trouvent au niveau du noyau, et ne doivent pas être modifiés à chaque fois qu'un nouveau composant de communication est intégré. L'extension de ces composants est assurée par le principe d'inconscience sémantique, à travers des bases de connaissances.

Techniquement, les composants de gestion sont créés au départ sans aucune information sur les éléments évolutifs de l'architecture. Ces informations ne sont pas non plus acquises avec le temps, mais récupérées de façon dynamique depuis les bases de connaissances pour chaque opération faisant intervenir ces éléments évolutifs.

Nous décrivons dans cette section les composants de gestion qui auront à faire à des éléments évolutifs. Il s'agit principalement du composant d'interface avec les applications et du composant de décision de notre architecture.

3.4.1 Interface applications

L'interface avec les applications ("Interface application") gère plusieurs éléments évolutifs : les services, les composants, les services par défaut, les composants de monitoring, ... Pour suivre l'évolution de l'ensemble de ces éléments, cette interface se base sur des algorithmes indépendants de la sémantique des éléments qu'ils traitent. Ces algorithmes seront décrits au chapitre 4.

Lors de la phase de traitement de la requête de l'application, l'interface application gère en premier lieu les dépendances entre services. L'application peut en effet demander certains services qui ne peuvent fonctionner qu'à la présence d'autres services. Dans ce cas, l'interface complète la requête par les services manquants. Ces dépen-

dances font partie de la description de chaque service, et sont traitées en fonction de cela. C'est à dire que l'interface application, en interne, n'est au courant d'aucune de ces dépendances, ni de leur signification. Pour chaque requête émise par l'application, elle récupère dans la description des services en question, l'ensemble des dépendances pour chacun d'eux, et les ajoute dans la requête. L'algorithme ne se basant pas sur des connaissances a priori, il est possible de disposer, de modifier et d'ajouter autant de dépendances que nécessaire ; celles-ci seront prises en compte automatiquement par l'interface sans aucun besoin de modification.

Le même principe est appliqué pour la gestion des incompatibilités entre services, lorsque la requête de l'application est incohérente. Les incompatibilités sont récupérées de façon dynamique depuis la description des services en question et traités en conséquence.

Les services par défaut sont eux aussi décrits dans une base de connaissance. Bien qu'ils soient considérés comme des services particuliers, leur traitement se fait de la même façon que pour les autres services. La requête de l'application est complétée éventuellement par ces services par défaut, et le processus se poursuit avec l'ensemble des services sans savoir lequel a été invoqué par l'application et lequel a été complété par l'interface. De même, les composants de monitoring, bien qu'ils soient des composants particuliers, sont eux aussi traités comme les autres composants. Le processus de traitement de la requête de l'application est décrit en détail au prochain chapitre.

3.4.2 Composant de décision

A l'instar de l'interface application, le composant de décision traite plusieurs éléments évolutifs. Il peut suivre l'évolution des éléments sans avoir à être modifié. Pour les services de communication, les composants et les services par défaut, le principe est le même que pour l'interface application.

Nous focalisons ainsi notre propos sur la phase de monitoring. Le composant de décision doit faire face à l'évolution des réseaux, des valeurs de monitoring et des composants effectuant ce monitoring, tout cela sans avoir à être modifié.

Pour cela, le premier principe sur lequel il s'appuie réside dans le fait que le monitoring est effectué par les composants de communication et non pas par le composant de décision lui-même. De cette façon, il n'a pas besoin d'être conscient de la signification, de l'origine ou du traitement des événements réseau. Pour chaque connexion, le composant de décision récupère les informations de monitoring partagées par les composants de communication associés à la connexion en question, et se base sur celles-ci pour surveiller d'éventuels changements dans le contexte réseau [2].

Par ailleurs, il n'a aucune connaissance préalable de ces valeurs de monitoring ni de leur signification. Elles sont stockées dans une base de connaissance, et pour chaque connexion, le module de décision parcourt l'ensemble des composants et récupère les références des valeurs partagées. L'algorithme associé traite chaque valeur comme une donnée brute n'ayant aucune signification. Il l'associe simplement au composant de

2. La notion de contexte réseau (ou profil réseau) sera décrite plus en détail dans le chapitre 4 dédié à l'autonomie du système.

communication qui lui correspond le mieux.

Les différents réseaux sont décrits par ces valeurs de monitoring et non par leur nature (réseau local sans fil, réseau filaire longue distance, réseau satellite, ...). Ils sont ainsi décrits par leur débit, leur délai ou encore leur taux de perte. De cette façon, le composant de décision n'identifie pas les réseaux, il établit des profils et les compare aux plages d'adaptabilité des composants disponibles dans la couche Transport. Quelle que soit la technologie sur laquelle il fonctionne, le composant de décision peut la gérer, sans avoir de connaissances préalables, en identifiant son profil et lui associant les composants adaptés.

Cette flexibilité permet au composant de décision de prendre en compte n'importe quelle technologie et n'importe quel jeu de composants dans l'architecture. Il s'agit pour lui de résoudre un problème d'optimisation, où tout est décrit par des valeurs, sans aucune sémantique. L'extensibilité du composant de décision est l'une des plus importantes pour l'évolution de la couche Transport. Ceci vient du fait que l'évolution des services et des composants, bien qu'elle soit sans impact, ne servira à rien si elle ne peut pas être prise en compte de façon dynamique par le composant de décision.

Le dernier point concernant la décision est en rapport avec la sémantique des dysfonctionnements à traiter. Dans l'approche générale des systèmes autonomes, lorsque l'on détecte un problème, on procède en premier lieu à un diagnostic qui permet d'identifier la nature du problème et les causes sous-jacentes. Selon le diagnostic établi, le système tente de trouver la solution la plus appropriée pour se remettre dans un état de fonctionnement correct. Dans notre architecture, la partie autonomique est inconsciente de la sémantique de ces problèmes. Son rôle se limite à l'association des meilleurs composants à une situation donnée. La gestion du problème lui-même est de l'ordre des composants de communication. C'est au niveau de ces composants là que la nature du problème est identifiée, et que la solution adéquate est appliquée. Cette approche permet à la couche Transport de faire évoluer ses diagnostics et ses solutions afin de tenir compte des nouveaux problèmes et d'adopter de nouvelles solutions, sans pour autant avoir besoin d'apporter des modifications à ses composants de gestion.

3.4.3 Cas du gestionnaire des applications legacy

Le gestionnaire des applications legacy est le seul composant de gestion de l'architecture qui n'est pas complètement extensible. Ceci est dû à plusieurs raisons. La première est que l'interception des requêtes et l'identification des protocoles ciblés nécessite des connaissances spécifiques pour différencier les appels et effectuer les traductions. La deuxième raison réside dans le fait que l'évolution des protocoles de Transport classiques est très lente ou presque inexistante. Avoir des composants conscients est toujours plus optimal et moins coûteux, et il est plus rentable de les utiliser quand le besoin d'une extensibilité ne se présente pas.

Cela dit, les services correspondants à chaque protocole sont, quant à eux, extensibles, et ne sont pas figés dans le gestionnaire. Une fois le protocole identifié, les services correspondants sont récupérés de façon dynamique depuis une base de données qui peut être mise à jour en fonction de l'évolution des services de Transport.

3.5 Extensibilité des applications

Nous avons parlé de l'extensibilité des systèmes, des réseaux, de la couche Transport avec ses parties de gestion et de communication. Le dernier acteur qui est l'application elle-même. Il s'agit pour la couche Transport, d'une part, de pouvoir prendre en compte les nouveaux besoins des applications, et d'autre part, d'évoluer en interne sans les affecter. Le premier objectif est réalisé via la gestion des nouveaux services via la base globale des services, et le deuxième grâce à l'interface abstraite.

3.5.1 Base globale des services

La couche Transport gère les services en premier via le composant d'intégration. C'est lui qui identifie les services qui ne sont pas encore disponibles au niveau de la couche lorsqu'un nouveau composant est intégré. Ensuite, ce nouveau service est ajouté à la base globale des services pour être visible aux développeurs d'application. Les développeurs se basent sur cette liste lorsqu'ils formulent leurs requêtes.

La sémantique de ces services n'est connue que par les applications (développeurs) et les composants de communication. La couche Transport (partie gestion), servant d'intermédiaire entre les deux, ne sait rien de leur signification. Il est donc possible pour les applications de formuler n'importe quelle requête et elle pourra être prise en compte par la couche Transport, du moment que des composants adaptés sont disponibles.

Un dernier élément en relation avec l'extensibilité des applications se trouve dans la possibilité d'anticiper la disponibilité des services. Les services sont sollicités auprès de la couche Transport avec leurs identifiants. Il est possible pour les applications d'invoquer des services qui ne sont pas encore disponibles, si leurs identifiants sont connus à l'avance ; ils seront alors automatiquement pris en compte lorsque des composants offrant ces services seront disponibles. Il suffit pour cela de mettre ces services dans la partie optionnelle de la requête (cf. Chapitre 2), qu'ils soient indispensables ou non.

Si ces services sont optionnels, alors qu'ils soient disponibles ou non, cela n'affectera pas l'application ; celle-ci fonctionnera seulement de façon plus optimale lorsque ces services seront disponibles. Dans le cas où les services requis sont obligatoires (bien qu'ils soient dans la partie optionnelle de la requête [3]), les applications doivent, elles-mêmes, gérer les services en question. Ceci vise à éviter aux développeurs de réécrire leurs applications lorsque les nouveaux services deviennent disponibles.

3.5.2 Interface abstraite

Pour que la couche Transport puisse évoluer, c'est à dire ici, en intégrant de nouveaux composants de communication, la dépendance classique entre les applications et la couche Transport a été levée. En effet, les applications n'ont aucune connaissance des composants qui s'occupent de la transmission de leurs données, ni de la façon avec

3. Certains services peuvent être obligatoires pour l'application, mais par risque qu'ils ne soient pas offerts par la couche Transport, l'application peut les mettre dans la partie optionnelle de la requête pour ne pas voir sa requête refusée par l'absence de ces services au niveau Transport. Si c'est le cas, l'application en sera notifiée mais la requête ne sera pas refusée et l'application devra gérer elle-même le/les services en question.

laquelle la sélection de ces composant est effectuée.

Grâce à l'interface abstraite, avec laquelle les applications expriment leurs besoins en services uniquement, la couche Transport peut s'étendre en interne autant qu'il est nécessaire afin de prendre en compte les évolutions des technologies réseaux. Pour chaque service, quel que soit le composant sélectionné et la façon avec laquelle il l'a été, l'application n'en sera pas affectée.

3.6 Conclusion

Nous avons décrit dans ce chapitre les éléments qui assurent l'extensibilité de notre architecture. Nous sommes partis du constat que le manque d'extensibilité dans la couche Transport actuelle, qui fait que les modifications ou les extensions des protocoles existants, engendrerait une série d'implications sur son environnement, constitue l'un des principaux obstacles à son évolution.

Pour y remédier, nous avons fait en sorte d'assurer que tout élément de notre architecture susceptible d'évoluer puisse être étendu ou modifié sans aucun impact sur son environnement. Nous avons pour cela conçu l'architecture suivant des paradigmes assurant une indépendance entre la couche Transport et les applications. Nous avons séparé la partie évolutive de la couche et l'avons implanté en dehors du noyau. Concernant les composants de gestion faisant partie du noyau, nous les avons conçus de manière à ce qu'ils puissent évoluer de façon autonome, et donc sans aucune modification. Enfin, pour les applications, nous avons proposé une interface abstraite afin de lever leurs dépendances classiques avec la couche Transport.

De cette façon, la couche Transport peut évoluer autant que nécessaire afin de tenir comptes des nouveaux besoins des applications et des évolutions des technologies réseaux. Cependant, pour que toutes ces extensions puissent être prises en compte, les composants de gestion doivent évoluer de façon autonome. Cette autonomie constitue une partie importante de notre architecture, du fait que c'est elle qui permet à l'extensibilité de la couche Transport de prendre sens et donc d'avoir de l'intérêt. Le prochain chapitre sera dédié aux modèles et techniques assurant l'autonomie de la couche Transport.

Chapitre 4

Autonomie de la couche Transport

Au chapitre précédent, nous avons présenté l'ensemble des outils permettant l'extensibilité de notre architecture en offrant la possibilité d'ajouter et de maintenir des éléments de l'architecture, que ce soit au niveau gestion ou communication, et ce sans impact sur les applications, les systèmes hôtes, ou l'architecture elle-même. La couche Transport sera donc amenée à utiliser des composants qui n'étaient pas disponibles au moment de sa conception. Une telle architecture extensible ne peut donc pas être conçue de façon statique, avec des composants et des services connus au préalable. Par ailleurs, la nature des liens de bout-en-bout diffère d'une connexion à une autre, et ne peut être connu qu'après le lancement de la communication. Par conséquent, l'architecture ne peut se baser sur des compositions prédéfinies, mais doit s'adapter à chaque type d'application et à chaque type de réseau, et ce pour chaque communication.

Partant de ce constat, nous avons doté notre architecture de capacité d'auto-adaptabilité afin qu'elle puisse adapter son comportement à chaque situation. Il s'agit principalement d'adapter la communication associée à chaque connexion en fonction des besoins de l'application qui l'utilise et des caractéristiques du lien réseaux de bout-en-bout avec l'entité distante. Ceci est assuré par une création dynamique de différentes compositions selon le besoin. Ce chapitre est consacré en majeur partie à ce niveau d'adaptabilité, qui présente plusieurs problématiques techniques et scientifiques, notamment durant les phase de monitoring et de planification de la boucle autonomique.

Un autre niveau d'autonomie concerne la gestion du comportement global de la couche Transport, indépendamment des besoins des application en termes de services de communication et indépendamment des caractéristiques du lien réseau. Il s'agit de gérer les applications conçues pour l'interface socket classiques, des systèmes distant d'incluant pas la nouvelle architecture, et d'éventuelles contraintes sur les réseaux intermédiaires. Ce niveau inclue bien les quatre phase de la boucle autonomique, mais ne constitue une réelle problématique. Nous le présenterons brièvement en première section.

La suite du chapitre est structurée en trois sections. La première section présente le besoin lié à la problématique de l'autonomie. Elle présente brièvement le niveau d'autonomie liés à la gestion de l'environnement de déploiement ; l'autre niveau est ensuite présenté un peu plus en détail, en décrivant les différentes phases du processus autonomique. Dans un deuxième temps, l'accent est mis particulièrement sur les phases de monitoring et de décision. La phase de monitoring est présentée à la section 2, et traite

principalement des problèmes de l'établissement des profils réseau et de la détection des changements de contexte. La section 3 présente la phase de décision, elle décrit d'abord les principes théoriques liés à la décision multicritère. Ensuite elle présente le modèle et les algorithmes associé aux traitements de la requête d'application d'une part, et d'autre part à la sélection des composants selon les besoins de l'application et l'état du réseau.

4.1 Besoins et problématique

Tel que présenté au chapitre 1, l'autonomie consiste en la capacité d'un système à s'adapter aux évolutions de son contexte sans intervention extérieure (humaine). D'une façon générale, un système autonome se base sur la surveillance de son environnement dans le but de détecter d'éventuels changements nécessitant un comportement différent. Selon le standard définit par IBM [47], le paradigme de l'autonomic computing comporte quatre phases : la surveillance de l'environnement et la levée des symptômes, l'analyse de ces symptômes pour décider d'un besoin de modification comportementale, la décision du nouveau comportement en question, et enfin sa mise en place (voir le chapitre 1 pour plus de détails).

Concernant la couche Transport, l'objectif est d'ajouter une capacité d'autonomie dans la gestion des fonctions d'adaptation de l'architecture. Celles-ci répondent à deux types de besoins, fonctionnel et non fonctionnel. Le besoin fonctionnel concerne l'association des différents composants de communication à leur contexte respectif, notamment en rapport avec le réseau. Le besoin non fonctionnel couvre l'évolution de la partie de gestion par la prise en compte de nouveaux composants.

4.1.1 Besoin en autonomie

Nous avons décrit au premier chapitre les principes du paradigme de l'autonomic computing. Cependant, ce modèle est très général, et conçu à la base pour la gestion des systèmes physiques complexes, dans lesquels le coût des interventions humaines est particulièrement élevé. Dans notre architecture, le contexte est plus simple, et certaines fonctions, prévues dans le standard (voir chapitre 1), peuvent être implicites ou non nécessaires. Nous considérons ici uniquement les fonctions de configuration, d'optimisation et de gestion d'erreurs. La fonction de sécurité (pour les composants de gestion) n'a pas d'intérêt dans le cadre des besoins de l'architecture. La sécurité des transmissions est, quant à elle, assurée par le biais de composants de communication dédiés si l'application invoque le service correspondant.

Globalement, l'auto-configuration concerne les applications. L'optimisation ainsi que la gestion d'erreurs concernent le réseau. La gestion d'erreurs est relativement simple et se résume à la gestion des changements de contexte réseau en cours de la communication. L'auto-configuration porte sur la création des compositions associées aux services requis par les applications. Enfin, l'optimisation concerne le choix du composant le plus adapté au contexte réseau, soit au début ou en cours de la communication. Nous décrirons l'ensemble de ces fonctions en détail dans la suite de ce chapitre.

Le besoin fonctionnel de l'autonomie concerne l'adaptation du comportement de la

couche Transport au contexte applicatif et réseau, plus notamment pour le réseau. Rappelons que la couche Transport dispose d'une expression explicite des besoins applicatifs (sous forme de service requis) ; elle doit en revanche relever elle-même les informations relatives au réseau. L'objectif est de masquer aux applications les détails d'implémentation des composants offrant les services requis, en sélectionnant les composant pouvant assurer ses services, et ce sans intervention externe. Aussi, plusieurs composants peuvent être disponibles en réponse à un même service ; la couche Transport doit sélectionner, parmi eux, le plus adapté au réseau sous-jacent. Ce besoin, bien qu'il soit trivial, est nécessaire pour assurer le fonctionnement de base d'une couche Transport basée services/composants. C'est donc un besoin nécessaire même pour une architecture non extensible.

Le besoin non fonctionnel concerne la gestion autonome de l'extensibilité de l'architecture. L'architecture doit pouvoir intégrer et prendre en compte de nouveaux services et composants de communication sans impact sur ses composants de gestion. Il s'agit donc de sa capacité à répondre aux besoins des applications, sans avoir de connaissance à priori sur leur fonctionnement ni sur disponibilités en termes de services et de composants. Il en est de même pour le réseau, aucune modification ne doit être apportée aux composants de gestion pour pouvoir supporter de nouvelles technologies réseaux.

4.1.2 Niveaux d'autonomie

L'architecture doit adapter son comportement à deux endroits. Le premier se situe entre la couche Transport, les applications et les liens réseaux de bout-en-bout. Le deuxième se situe entre la couche Transport et son environnement d'exécution. Cet environnement concerne la nature des applications qui l'utilisent, la nature de l'architecture de la couche Transport du système distant, et les politiques du réseau intermédiaire.

Gestion de l'environnement de déploiement

Durant la phase de déploiement, la nouvelle architecture peut être utilisée par des applications classiques (legacy) ou à communiquer avec des systèmes sur lesquels la nouvelle architecture n'est pas déployée. De plus, les réseaux d'accès imposent la plupart du temps, pour de raisons techniques ou économique, des contraintes sur la nature des protocoles de Transport utilisés. La couche Transport doit donc être capable d'adapter son fonctionnement aux contraintes de l'environnement applicatif, système et réseau.

La problématique sous-jacente concerne donc ces trois acteurs, application, système et réseau. Très basiquement et pour rappel (les différents cas de figures sont illustrés au Chapitre II, section 2.4.

- pour gérer les applications classiques, il faut détecter les appels classiques et les traduire en des appels services.
- le choix de ces services va ensuite dépendre de la nature du système distant (client ou serveur), selon qu'il supporte ou non la nouvelle architecture.
- en fin, l'architecture doit s'assurer qu'aucun obstacle intermédiaire (un middleboxe stoppant par exemple le trafic non TCP) ne bloque la transmission de ses données, et prendre les mesures nécessaires le cas échéant.

La détermination de cette facette du contexte requiert un monitoring. Bien que plusieurs éléments et plusieurs situations soient à gérer, la mécanique associée est beaucoup plus simple que celle du monitoring associé à la gestion du choix des composants associés aux services requis par l'application. Cette partie ne sera donc pas traitée.

Gestion des services et des composants

Ce niveau d'autonomie constitue, sur le plan conceptuel, la partie la plus complexe de l'architecture. Il s'agit de la gestion, au profit de chaque application, de la liste des composants associés aux services requis. Elle comporte les fonctions de configuration, d'optimisation et de gestion d'erreurs du modèle autonomique (voir Chapitre 1, section 1.3.3).

Ce niveau inclut le monitoring du réseau, la détection des changements de contexte, la détection de non-conformité des compositions en cours, la création et le déploiement des nouvelles compositions. Ceci comprend également la considération d'autres fonctions, comme la gestion des nouveaux composants et services ou la gestion des compositions parallèles. Nous étudierons ces différents aspects dans la suite de ce chapitre. Sur les quatre phases de la boucle autonomique, les phases de monitoring et de planification sont les plus complexe. Nous leur consacrons par conséquent deux sections distinctes à la fin du chapitre. Elles sont cependant abordées brièvement ci-après avec les deux autres phases (Analyse et Exécution).

4.1.3 Autonomie de la sélection des composants

Le processus de sélection des composants se base sur plusieurs éléments, à savoir les différentes bases de connaissance créées lors de l'intégration des composants, la requête soumise par l'application, et les informations sur l'état du réseau (voir chapitre 2).

Le modèle autonomique qui se rapporte à la sélection de composants comporte, sur ses quatre phases : le recueil des informations relevant du réseau et la détection des changements de contexte réseau pour la phase de monitoring, la vérification de la conformité pour la phase d'analyse, le choix des composants pour la phase de planification, et enfin leur déploiement pour la phase d'exécution. Nous introduisons ci-après l'ensemble de ces phases, nous décrirons plus en détails le monitoring et la planification dans des sections séparées.

Monitoring

Le monitoring est la première phase dans le processus autonomique. Elle consiste en un recueil de données et un traitement dans le but de lever d'éventuels symptôme. Le recueil de données, tel que présenté au Chapitre 3, se base sur les composants de communication déjà déployés. Le rôle du monitoring porte donc principalement sur le traitement des données recueillies, et ce à deux niveaux : l'établissement du profil réseau et la détection d'un éventuel changement du contexte.

Dans la suite, nous utiliserons indifféremment les notions de contexte et de profil réseau. En toute rigueur, un contexte réseau est caractérisé par son profil. Le profil est la

description "technique" d'un contexte réseau, constituée d'un ensemble de valeurs correspondant aux métriques considérées par le monitoring (délai, débit, ...). Il s'agit bien entendu du profil de bout en bout. Les profils sont établis, pour chaque connexion, en effectuant des traitements (principalement des moyennes) sur les données recueillies par les composants de communication.

La détection de changement de contexte consiste à observer l'évolution des valeurs monitorées dans l'objectif de détecter des changements significatifs amenant le réseau dans un état pour lequel les composants déployés peuvent ne plus être adaptés. Ces changements peuvent survenir de façon brusque (sous forme de pics) ou de façon progressive. Les changements brusques peuvent ne constituer que des changements temporaires. L'objectif est donc de distinguer les pics momentanés, les pics constants, et les évolutions progressives. Il faut aussi définir plus formellement ce qu'un "changement significatif" veut dire. Nous verrons tout cela en détail en section 3.

Analyse

La phase d'analyse ne constitue pas une réelle problématique. Elle consiste simplement à vérifier la conformité de la composition en cours par rapport au nouveau contexte réseau. Cette vérification s'effectue pour chaque valeur monitorée. Dans le cas où une ou plusieurs de ces valeurs se situent hors de l'intervalle d'adaptabilité de l'un des composants en cours, le module de décision est invoqué afin de vérifier s'il existe un composant plus adapté. Il est à noter que cette analyse se fait composant par composant, indépendamment les uns des autres. Cela signifie que la nouvelle composition concernera uniquement le sous-ensemble de composants jugés non appropriés au nouveau contexte. Il se peut qu'il s'agisse cependant de la composition entière, mais pas nécessairement.

Dans certaines architectures autonomes, la phase d'analyse est beaucoup plus complexe, plus complexe que le monitoring notamment. Elle permet d'identifier l'origine des symptômes levés par la fonction de monitoring, et se base sur des modèles sémantiques relativement complexes [68]. Dans notre architecture, nous avons opté pour une autre approche. En effet, l'analyse des causes des problèmes et des solutions potentielles nécessitent une connaissance de leur sémantique, comme ce que signifie une congestion ou une dégradation du signal. Cette dépendance sémantique constitue une limite face à l'extensibilité d'une architecture (tel que décrit au chapitre 3). Il s'agit par conséquent de trouver uniquement le meilleur élément pour gérer le contexte actuel du réseau. Toute la sémantique au niveau de la couche Transport est implantée au niveau des composants de communication.

Planification

La phase de planification, également dite de décision, permet d'associer les composants les plus adaptés au réseau sous-jacent qui permettent d'assurer les services requis par l'application. Elle est invoquée dans deux situations : suite à une requête de l'application, ou suite à un changement du contexte réseau. Dans le premier cas, elle permet de créer une première composition, sans tenir compte de l'état du réseau, du fait que ce dernier n'est pas encore connu. Elle se sert des composants par défaut des services en question. Pour rappel, pour chaque service de la couche Transport, il existe un et

un seul composant déclaré comme composant par défaut, qui sera sélectionné lorsque l'état du réseau n'est pas encore connu. Il s'agit des composants ayant comportement général, pouvant fonctionner de façon acceptable dans la plupart des contextes.

Dans la deuxième situation, la phase de décision est invoquée suite à un changement de l'état du réseau. Dans ce cas, cette phase ne donnera pas forcément lieu à une nouvelle composition, mais ce sera le cas s'il existe des composants plus adaptés que ceux de la composition en cours. Ce deuxième mode de décision est bien plus complexe du fait qu'il tient compte du paramètre réseau. La vraie problématique de décision se situe par conséquent à ce niveau-là. Nous l'étudierons en détails en section 4.

Exécution

Le déploiement d'une composition diffère selon qu'il s'agisse de la première composition (à l'initialisation de la connexion) ou du remplacement d'une composition déjà déployée.

Dans le premier cas il s'agit simplement d'instancier l'ensemble des composants de communication et de les associer à une connexion, pour laquelle une référence est renvoyée à l'application.

Dans le deuxième cas, lorsqu'une autre composition est déjà déployée, le déploiement de la nouvelle composition est plus difficile. En effet, si l'on attend que toutes les données présentes dans le buffer d'émission soient transmises par la première composition avant de déployer la nouvelle, la rupture dans la transmission de données peut altérer le fonctionnement de l'application, ou à défaut nuire à ses performances, notamment si les paramètres réseau ne sont pas très favorables. En revanche, si l'on déploie la nouvelle composition en remplacement directe de la première, des incohérences de données peuvent se produire même si l'on récupère le contenu de l'ancien buffer.

La solution est donc de déployer la deuxième composition en parallèle avec la première, et laisser celle-ci s'occuper des données déjà présentes dans ses buffers jusqu'à ce qu'elles soient toutes transmises ; les nouvelles données seront en revanche traitées par la nouvelle composition.

Ce processus est complètement transparent pour l'application et aucun changement ne lui sera visible. C'est la référence de la première connexion qui sera modifiée afin de pointer vers la nouvelle, et l'application continuera à utiliser la même référence. Pour le système distant en revanche, le passage à la nouvelle composition doit lui être notifié. En effet, y compris pour un même service, les composants de communication peuvent ne pas être compatibles. Par ailleurs, les deux systèmes doivent pouvoir différencier leurs connexions respectives lors de la transition.

Bien que le protocole associé à la négociation de ces compositions soit simple et basique dans sa logique, son implémentation réelle est plutôt complexe. Par conséquent, et faute de temps, nous ne traiterons pas de cet aspect dans le cadre de ce travail. Pour les tests et les simulations, nous partirons d'hypothèses simplifiées permettant d'éviter les situations d'incompatibilité.

4.2 Monitoring et changement de contexte

Comme introduit plus haut, le monitoring a pour rôle de relever, en temps réel, les différentes valeurs qui caractérisent l'état du réseau à un instant donné. Ces valeurs diffèrent d'une communication à une autre, en fonction des composants déployés. Ces valeurs sont relevées par défaut par les composants de communication, c'est-à-dire les composants correspondants aux services requis par l'application. Il se peut cependant que d'autres composants de monitoring dédiés soient utilisés (voir Chapitre 3).

Les valeurs de monitoring servent à deux objectifs :
- le premier objectif consiste en la détection d'éventuels changements du contexte réseau. Elles sont dans ce cas utilisées par le composant de monitoring lui-même pour comparer le profil enregistré au profil actuel afin de conclure à un éventuel changement. Ces changements doivent être significatifs et relativement durables afin de les distinguer des pics de changement passagers ;
- le deuxième objectif est en rapport avec le choix des composants. En effet, le composant de décision se base sur les profils établis lors du monitoring pour sélectionner de nouveaux composants plus adaptés.

4.2.1 Recueil des données de monitoring

L'établissement des profils commence par le recueil des données de l'état du réseau auprès des composants de communication. Dans l'optique de limiter le surplus de données émises sur le réseau, le monitoring est autant que possible effectué par les composants de communication déjà déployés dans les connexions. En effet, les composants de communication se servent des données déjà envoyées pour récolter des données sur l'état du réseau (comme le RTT calculé par le composant de fiabilité). De plus, ceci permet, tel que décrit au Chapitre 3 de limiter la conscience sémantique de ces valeurs par les composants de gestion afin de garantir l'extensibilité du composant de décision.

Les valeurs de monitoring publiées par les composants de communication font partie de la description de ces derniers dans les bases de connaissance. A la création de chaque connexion, une mémoire partagée est réservée pour l'ensemble des valeurs monitorées par tous les composants de la connexion, et des références à ces valeurs sont transmises aux autres composants selon leurs besoins, y compris le composant de monitoring.

Dans le cas où une des valeurs dont un des composants a besoin n'est pas pris en charge par les composants déjà déployés, les composants spécifiques au monitoring sont ajoutés à la composition. Ce processus est réalisé pendant le traitement de la requête de l'application, il sera décrit un peu plus loin. Par contre, pour l'établissement des profils, l'ajout de composants de monitoring n'est pas automatique. En effet, lorsque le besoin en monitoring se limite à la création des profils, autrement dit lorsque certaines valeurs monitorées ne sont utilisées par aucun des composants de communication, la couche Transport n'ajoutera pas automatiquement de composants dédiés au monitoring, comme dans le premier cas.

En effet, les composants dédiés doivent envoyer du trafic supplémentaire sur le réseau

afin de mesurer ces valeurs. Ce surplus de données peut nuire aux performances perçues par l'application. Il est donc à l'application d'exprimer son choix pour spécifier à la couche Transport si elle souhaite ou non utiliser des composants de monitoring qui ne seront utilisés que pour la décision, c'est à dire, dont aucun composant de communication ne se sert. Dans le premier cas, les profils réseaux se limiteront aux valeurs disponibles selon la composition déployée. Dans le deuxième cas, la couche Transport ajoute autant de composants que nécessaire pour assurer un monitoring complet du réseau, c'est-à-dire monitorer toutes les valeurs pour lesquels des composants associés sont disponibles. Le principe du monitoring est présenté à la figure 4.1.

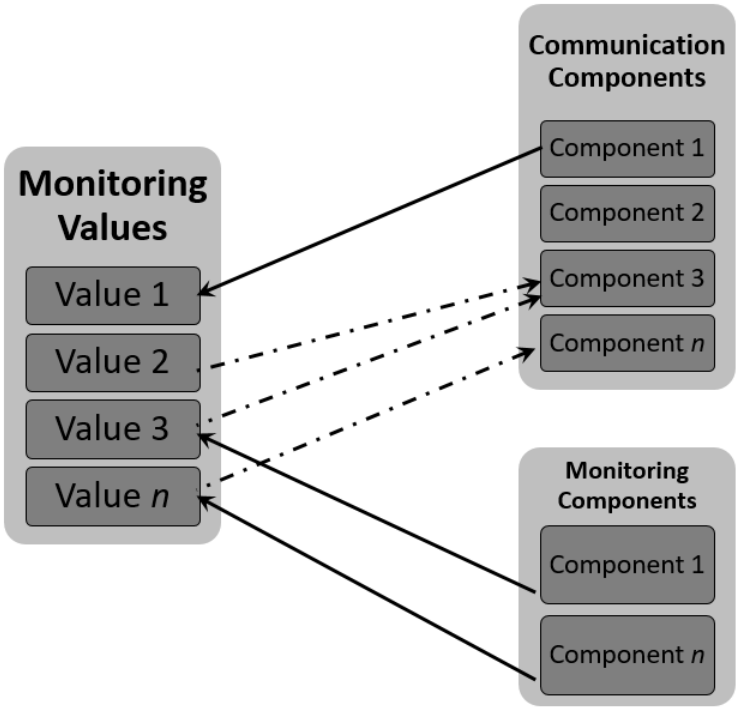

FIGURE 4.1 – Classification des solutions de Transport

On note que les composants de monitoring sont traités comme des composants de communication ordinaires. La seule partie consciente de leur existence est l'interface application (i.e. le composant de gestion entre les applications et la couche Transport). C'est elle qui ajoute ces composants à la requête de l'application. Ils sont décrits comme des composants de communication offrant un service de monitoring spécifique. L'interface application ajoute à la requête de l'application, en plus des services requis, les services relatifs à toutes les valeurs qui ne sont pas fournies par les composants associés aux services de base de l'application. Ensuite, ceux-ci seront traités indifféremment par le composant de décision en sélectionnant les composants associés à ces services selon de l'état du réseau.

4.2.2 Profils de réseau et changements de contextes

Les profils réseau sont établis à partir des valeurs de monitoring en utilisant des moyennes glissantes successives en partant de l'instant où la composition en cours est déployée. Ce principe est plus adapté qu'une fenêtre glissante, car cette dernière peut conduire à

ne pas repérer une évolution progressive, et ne permet pas de garder trace des moyennes réelles aux bords de deux profils. Aussi, elle ne permet de tenir compte que d'un nombre limité de valeurs (historique de monitoring limité). Le profil réseau est constitué de l'ensemble des moyennes correspondant aux différentes valeurs monitorées. Ces moyennes sont calculées au fur et à mesure que les composants de communication publient leurs valeurs monitorées.

L'établissement d'un profil, pour chaque valeur de monitoring, est relativement complexe. La difficulté réside notamment dans la distinction entre les "vrais" changements de profil et les pics momentanés. Mais aussi, quelle différence de valeur constitue un changement de contexte et laquelle constitue une simple variation dans le même contexte. Par ailleurs, cette complexité est renforcée par le fait qu'il n'y a pas de description préalable pour les différents réseaux, qui sont identifiés uniquement par leurs profils, ceci pour des raisons d'extensibilité tel que décrit au Chapitre 3.

La détection des changements de contextes se base, en grande partie, sur la détection des pics. Cela vient du fait que la plupart de ces changements, qui soient significatifs, surviennent de façon brusque. Le problème revient donc, comme annoncé précédemment, à identifier les moments des pics, et de distinguer ceux qui sont momentanés ou passagers. Dans la littérature, il existe plusieurs techniques pour identifier les pics et détecter des changements de contexte, en distinguant les deux. Ces techniques se basent le plus souvent sur des calculs statistiques ou probabilistes, comme dans [69][70]. L'établissement des profils dépend fortement du domaine d'application et des besoins associés.

Les algorithmes permettant d'identifier des pics dans des séries de données ont été initialement proposés pour des applications ne concernant pas le monitoring. Il s'agissait par exemple d'application relevant de l'imagerie médicale [71] ou du traitement du signal [72]. L'application de ce principe au monitoring est relativement récente (principalement années 2000) [73]. L'approche commune consiste à utiliser des fonctions de lissage, comme dans [74], peu sensibles aux pics instantanés, en parallèle avec des fonctions de comparaison. Ces fonctions de comparaison se basent sur différents principes, simples comme le calcul du maximum local, ou plus complexes avec des méthodes statistiques et probabilistes [75]. Le problème reste cependant dans la différenciation des variations locales et des changements réels de valeurs. La question revient alors à déterminer à partir de quelle différence de valeur on peut conclure à un changement (qu'il soit momentané ou durable). Généralement, les algorithmes ont toujours un paramètre lambda configurable qui représente la sensibilité de l'algorithme.

Cette sensibilité dépend par ailleurs de l'historique considéré par l'algorithme pour un moment donné dans la série. Plus l'historique est important, et plus l'algorithme devient précis, mais aussi plus lent en conséquence [76]. Certains algorithmes considèrent uniquement le passé d'un point donné [77], d'autres ajoutent le futur [78]. La plupart des solutions utilisent la méthode des fenêtres glissantes pour se déplacer dans la série de données.

Une autre catégorie de solutions se base sur des fonctions de poids [79], qui sont aussi probabilistes. Elles attribuent à chaque élément de la série un poids correspondant à la probabilité qu'il constitue un changement, et ce, en tenant compte soit des valeurs précédentes uniquement, ou aussi des suivantes. Une dernière approche est celle de

la classification, utilisée notamment pour la détection de changement de contexte [80]. Elle consiste, dans une série de données, à effectuer au préalable une classification selon les différentes valeurs, et ainsi établir une liste de classes de profils. Par la suite, les nouvelles valeurs sont comparées afin d'être affectées à la classe correspondante. Ces méthodes fonctionnent à base d'heuristiques non déterministes, et se réfèrent à une classification à priori. Une description synthétique de l'ensemble de ces méthodes peut être trouvée dans [80].

Quel que soit la méthode adoptée, toutes ces solutions ont des points communs. D'abord, elles se basent, en partie ou en totalité, sur des données existantes et non relevées en temps réel. Ensuite, le calcul des estimations utilise des formules relativement complexes et lentes, qui ne sont pas adaptées à une application dans un contexte temps réel. Enfin, elles se basent toutes sur une certaine connaissance préalable du contexte d'application (profil global des données, hypothèses, ...) ce qui ne correspond pas au principe d'indépendance sémantique de l'architecture (voir Chapitre 3).

Dans notre cas, le besoin est d'identifier des changements de contextes, aussi vite que possible, sans avoir aucune connaissance préalable du contexte d'application, ni des données recueillies. De plus, les calculs doivent être effectués en temps réel à mesure que des données sur le réseau sont relevées. Par ailleurs, les profils établis sont des profils de bout en bout, et donc propres à chaque connexion. La fonction de monitoring est par conséquent associé à chaque connexion, et s'effectue avec la même fréquence que celle de l'arrivée des paquets. De ce fait, il devient nécessaire de restreindre cette fonction à des opérations simples, faute de quoi les performances du protocole risqueraient d'être significativement affectées, notamment pour des transfert haut débit ou temps réel.

Nous présentons dans cette partie la solution que nous avons proposée pour la détection des changements de contexte, basée sur un algorithme léger, exécutable en temps réel, et sans connaissance préalable des contextes.

4.2.3 Algorithme de détection de changement de contexte

L'algorithme que nous proposons distingue quatre modes de variation de données : des pics momentanés, des pics permanents, des évolutions progressives momentanées et des évolutions progressives permanentes. Chaque mode présente ses propres particularités, difficultés, et approches de détection. Nous considérons un changement de contexte comme une variation importante dans le cours des données pendant une durée suffisamment longue. Les valeurs de cette variation et de cette durée diffèrent selon le(s) paramètre(s) de monitoring considéré(s) ; elles seront donc représentées dans la suite par des paramètres variables dans les fonctions.

Les pics momentanés consistent en d'importants écarts de valeurs qui surviennent brusquement, et pour un laps de temps relativement court. Ces pic ne doivent pas être considérés comme des changements de contextes, parce que le coût de la reconfiguration serait plus pénalisant que de fonctionner avec la configuration en cours. En effet, si pour chaque pic de donnée, on refait une configuration, il se peut que le contexte change une deuxième fois avant même que la nouvelle configuration correspondant au premier changement ne soit déployée. L'architecture passera alors tout son temps

à changer de configuration, créant ainsi un état instable de la communication, et un retard important dû à la reconfiguration et à la gestion des compositions parallèles (voir Chapitre 5). Quand ces pics durent assez longtemps, ils deviennent permanents, et un changement de contexte devient dès lors nécessaire. La détection d'un pic est relativement simple, vu l'écart important qu'il constitue en comparaison aux valeurs précédentes. Le vrai problème réside dans la différentiation entre les pics momentanés et les pics permanents.

Un changement de contexte peut cependant se produire, non pas de façon brusque, mais progressive. En effet, les valeurs peuvent augmenter par exemple jusqu'à atteindre le niveau d'un pic, mais avec de petits écarts progressifs qui n'atteignent pas le seuil d'un pic. Contrairement aux pics, les évolutions progressives sont plus difficiles à détecter. En revanche, une fois le seuil d'un nouveau contexte est atteint, le schéma a moins de chance de régresser pour revenir au précédent contexte.

Le principe de l'algorithme est d'abord de détecter un éventuel changement de contexte, par un pic ou par une évolution progressive. Une fois détecté, l'algorithme garde trace du profil du réseau au moment où la variation a été détectée, et surveille si les nouvelles valeurs mènent vers un nouveau contexte. Dans le cas d'un pic, il doit attendre assez longtemps (mais juste assez) avant de conclure à un changement de contexte. Dans le cas d'une évolution progressive, il doit attendre à ce que l'évolution atteigne le seuil d'un nouveau contexte. Si un nouveau contexte est identifié, le profil précédent est remplacé par celui en cours, et recommence depuis le début de la variation. Dans le cas contraire, la variation est momentanée et ne doit pas être prise en compte, l'algorithme recommence donc par le dernier point stable correspondant au profil précédent.

Formellement, on pose le problème ainsi : soit X l'ensemble des mesures prises pour une des valeurs de monitoring, et X_t la mesure à l'instant t. A tout instant tn, nous avons $X = X_{t_0}, X_{t_1}, ..., X_{t_n}$, $t_0$étant l'instant de début de monitoring (il s'agit de l'instant où la composition en cours est déployée et non pas l'instant de création de la connexion).

La détection des pics se fait par comparaison de l'écart entre la mesure d'une valeur à un instant donné et la moyenne des écarts précédents. Lorsque cette valeur est "significativement" élevée (i.e. supérieure à un paramètre ? décrit ci-après), on conclut qu'il s'agit d'un pic.

$$\text{On dit que } X_{t_n} \text{ est un pic si : } \left[(X_{t_n} - M\delta)/M\delta\right] > \varphi$$

$$M\delta(t_n) = \left[|X_{t_0} - X_{t_1}| + |X_{t_1} - X_{t_2}| + ... + |X_{t_{n-2}} - X_{t_{n-1}}|\right]/n - 1$$

Le paramètre φ est défini différemment selon la valeur monitorée. Il représente le décalage habituel pour une valeur donnée. Les évolutions progressives sont détectées par mesure du rapport entre les valeurs descendantes et les valeurs ascendantes. En effet, pour maintenir un schéma de donnée relativement constant, il faut qu'il y ait un certain équilibre dans le rapport entre les valeurs ascendantes et les valeurs descendantes. Schématiquement, cela veut dire que la courbe qui représente les données doit, globalement, monter autant qu'elle descende. Pour une évolution progressive, les valeurs

montantes doivent dominer les valeurs descendantes, ou l'inverse. Formellement : on considère qu'un ensemble X est en évolution progressive si :

$$\left[M^+/M^-\right] > \varphi \text{ ou } \left[M^+/M^-\right] < \varphi$$

$$M^+ = \sum_{i=0}^{n} X_{t_i} \text{ tel que } X_{t_i} > X_{t_{i-1}}$$

$$M^- = \sum_{i=0}^{n} X_{t_i} \text{ tel que } X_{t_i} < X_{t_{t-1}}$$

φ est défini de la même façon que pour les pics.

Pour conclure qu'un pic correspond à un changement de contexte, il faut qu'il reste dans la même plage de valeur pendant assez de temps. Si l'on attend trop longtemps, on retarde le passage au nouveau contexte, et par conséquent on perd en performances, mais si l'on n'attend pas assez, on risque d'être piégé par un pic momentané. Pour gérer les attentes, qui au passage ne doivent pas dépendre d'un paramètre temps, mais de l'évolutions des mesures, on utilise une fonction récursive de lissage classique pour calculer la moyenne (voir [81] pour plus de détails sur les méthodes de lissages) :

$$M(t_i) = [M(t_{i-1}) * \varphi] + [X_{t_i} * (1 - \varphi)]$$

Nous utiliserons la même fonction pour détecter les changements de contexte dans le cas d'une évolution progressive. Le paramètre ? dépend, de la même façon que précédemment, de la valeur monitorée. Il est à noter que, techniquement, les différentes formules sont calculées en temps réel, c'est à dire pour chaque nouvelle mesure. L'algorithme 1 complet est décrit ci-après en pseudo code.
L'algorithme 1 permet de détecter un éventuel changement de contexte. Il se peut qu'il s'agisse réellement d'un changement ou d'une variation passagère. Lorsqu'une telle situation est détectée, l'algorithme sauvegarde le profil actuel et se met dans un état de test de continuité, en utilisant soit l'algorithme 2 soit l'algorithme 3 selon le cas. Si le nouveau profil reste relativement constant pour une durée assez suffisance, un changement de contexte est conclu, et l'ancien profil est remplacé par le nouveau profil. Une phase d'analyse et ensuite exécutée afin de vérifier si la composition en cours est toujours adaptée au nouveau contexte, ou si une nouvelle composition est nécessaire.

Il y a deux principales particularités de notre approche par rapport à celles existantes en littérature. La première est le fait de traiter les données en temps réel, c'est à dire à mesure que les valeurs de monitoring sont publiées. La deuxième est son inconscience complète de la sémantique des données traitées. En effet, l'approche peut être appliquée à n'importe quelle série de données en ajustant les paramètres variables. Un exemple d'inconscience sémantique est l'absence de profils prédéfinis à la base desquels on détecte les variations. Les profils sont établis au fur et à mesure du traitement des données.

Algorithm 1 Monitoring

Require:
The last measures LastXi, NewXi
The moving averages LastAvrg, NewAvrg
The gap averages LastDelta, NewDelta, Delta
The evolution averages Pavrg, Navrg
The classical Xi averages SLastAvrg, SNewAvrg
The thresholds $\varphi_1, \varphi_2, \varphi_3$
The index i

```
1: i ← 0
2: loop
3:   NewXi ← waitForNewValue()
4:   LastAverge ← (LastAvrg * φ1) + (NewAvrg * (1 − φ1))
5:   Delat ← |NewXi − LastXi|
6:   SLastAvrg ← ((SLastAvrg * (i − 1)) + NewXi)/i
7:   if (|Delta − LastDelta|/LastDelta) > φ2 then
8:     if (tesBurst()) then
9:       LastAvrg ← NewAvrg
10:      LastDelta ← NewDelta
11:    else
12:      LastDelta ← ((LastDelta * (i − 1)) + delta)/i
13:    end if
14:  else
15:    if ((Pavrg/Navrg) > φ3)or((Navrg/Pavrg) > φ3) then
16:      if (testEvolution()) then
17:        LastAvrg ← NewAvrg
18:        LastDelta ← NewDelta
19:      end if
20:    else
21:      LastDelta ← ((LastDelta * (i − 1)) + delta)/i
22:      LastXi ← NewXi
23:      i ← i + 1
24:      if NewXi > LastXi then
25:        Pavrg ← (Pavrg * 0.5) + (Delta * 0.5)
26:      else
27:        Navrg ← (Pavrg * 0.5) + (Delta * 0.5)
28:      end if
29:    end if
30:  end if
31: end loop
```

Algorithm 2 Pic Permanent

Require:
The last measures LastXi, NewXi
The moving averages LastAvrg, NewAvrg
The gap averages LastDelta, NewDelta, Delta
The evolution averages Pavrg, Navrg
The classical Xi averages SLastAvrg, SNewAvrg
The thresholds φ_1, φ_3
The index i
The boolean FackBurst

1: $NewAvrg \leftarrow LastAvrg$
2: $FackBurst \leftarrow$ **false**
3: $i \leftarrow 0$
4: **while** (NewAvrg < SLastAvrg) **and** (**not** FackBurst) **do**
5: $NewXi = waitForNewValue()$
6: $NewAvrg \leftarrow (NewAvrg * \varphi_1) + (NewXi * (1 - \varphi_1))$
7: $Delta \leftarrow |NewXi - LastXi|$
8: $NewDelta \leftarrow ((NewDelta * (i - 1)) + Delta)/i$
9: $SNewAvrg \leftarrow ((NewDelta * (i - 1)) + NewXi)/i$
10: $LastXi \leftarrow NewXi$
11: $i \leftarrow i + 1$
12: **if** $NewXi > LastXi$ **then**
13: $Pavrg \leftarrow ((Pavrg * ((i/2) - 1)) + Delta)/(i/2)$
14: **else**
15: $Navrg \leftarrow ((Navrg * ((i/2) - 1)) + |Delta|)/(i/2)$
16: **end if**
17: **if** $((Pavrg/Navrg) > \varphi_3)$**or**$(Navrg/Pavrg) > \varphi_3)$ **then**
18: $FackBurst \leftarrow$ **true**
19: **end if**
20: **end while**
21: **if not**FackBurst **then**
22: $LastAvrg \leftarrow NewAvrg$
23: $LastDelta \leftarrow NewDelta$
24: $SLastAvrg \leftarrow SNewAvrg$
25: **end if**

Algorithm 3 Evolution Progressive Permanente

Require:
The last measures LastXi, NewXi
The moving averages LastAvrg, NewAvrg
The gap averages LastDelta, NewDelta, Delta
The evolution averages Pavrg, Navrg
The classical Xi averages SLastAvrg, SNewAvrg
The thresholds φ_1, φ_4
The index i
The boolean FackBurst

1: $NewAvrg \leftarrow LastAvrg$
2: $FackBurst \leftarrow$ **false**
3: $SLastAvrg \leftarrow NewXi$
4: $i \leftarrow 0$
5: **while** (NewAvrg < SLastAvrg) **and** (**not** FackBurst) **do**
6: $NewXi = waitForNewValue()$
7: $NewAvrg \leftarrow (NewAvrg * \varphi_1) + (NewXi * (1 - \varphi_1))$
8: $Delta \leftarrow |NewXi - LastXi|$
9: $NewDelta \leftarrow ((NewDelta * (i - 1)) + Delta)/i$
10: $SNewAvrg \leftarrow ((NewDelta * (i - 1)) + NewXi)/i$
11: $i \leftarrow i + 1$
12: **if** $(NewXi/LastDelta) < \varphi_4$ **then**
13: $FackBurst \leftarrow$ **true**
14: Break
15: **end if**
16: $LastXi \leftarrow NewXi$
17: **if** $NewXi > LastXi$ **then**
18: $Pavrg \leftarrow ((Navrg * ((i/2) - 1)) + Delta)/(i/2)$
19: **else**
20: $Navrg \leftarrow ((Navrg * ((i/2) - 1)) + |Delta|)/(i/2)$
21: **end if**
22: **if not**FackBurst **then**
23: $LastAvrg \leftarrow NewAvrg$
24: $LastDelta \leftarrow NewDelta$
25: $SLastAvrg \leftarrow SNewAvrg$
26: **end if**
27: **end while**

4.3 Processus de décision

Dans cette section nous décrivons les modèles et algorithmes permettant la sélection des composants de communication avant l'instanciation des connexions. Nous présentons d'abord quelques bases théoriques sur la décision, nous décrirons ensuite les solutions proposées.

4.3.1 Eléments théoriques

La problématique de la sélection des composants se positionne globalement au niveau de la théorie de la décision, et particulièrement dans la catégorie de la classification appliquée à l'aide à la décision. Le domaine de la décision est très vaste sur le plan théorique, avec un nombre important de modèles, d'approches, de techniques et d'algorithmes qu'il n'est pas possible de dresser dans ce contexte. Par ailleurs, aucune topologie universelle ne permet de regrouper l'ensemble des méthodes existantes [82].

La théorie de la décision relève de l'optimisation du domaine de la recherche opérationnelle, dont la majeure partie concerne la décision multicritère. En effet, lorsque l'optimisation se limite à un seul critère, le problème est relativement simple. Les méthodes de décisions peuvent être déterministes ou non déterministes, à solution unique ou à multi-solutions, à agrégation complète ou partielle, etc. La différence principale se situe dans la problématique à la base de laquelle la recherche s'effectue. On distingue en littérature la problématique de choix, de tri, de rangement, et autres. Une synthèse de ces méthodes pour être trouvée dans [83][84].

Comme noté ci-avant, les typologies de classification des méthodes de décision sont trop nombreuses pour être toutes citées, à fortiori pour citer les méthodes elles-mêmes. Nous faisons le point ici uniquement sur l'approche que nous avons adoptée, à savoir par fonction d'utilité. Elle fait partie de la catégorie des méthodes par agrégation complète, c'est-à-dire faisant intervenir l'ensemble des critères en même temps (dans une même formule), pour ramener enfin le problème multicritère à un simple problème unicritère basé sur le résultat de cette fonction qui prend en compte l'ensemble des éléments de l'espace de décision.

Il existe plusieurs variantes de cette méthode, chacune adaptée à une problématique plus ou moins particulière. Elles diffèrent notamment par la formule de calcul de l'utilité elle-même. Par ailleurs, cette méthode, à l'instar des autres méthodes d'agrégation totale, suppose une comparabilité équivalente entre tous les critères du problème. Il s'agit la plupart du temps de critères de même type. Une des difficultés de cette méthode apparaît lorsque les critères de décision ne sont pas tous d'une même importance. Des fonctions pondérées sont alors utilisées en affectant des coefficients différents pour chaque critère, mais cela ne résout pas parfaitement le problème [41].

Dans notre modèle de décision, nous allons ramener le problème à une sélection par fonction d'utilité, mais nous poserons certaines hypothèses assez fortes pour éviter de se retrouver dans les limites de l'approche par utilité, et de permettre l'obtention d'un résultat optimal et déterministe. Nous exposons dans le suite ces hypothèses, le modèle de décision et les algorithmes associés. En amont du processus de sélection des composants proprement dit, un processus de traitement de la requête de l'application est

effectué avant de la transmettre au module de décision. Nous décrivons d'abord l'ensemble de ces traitements, et nous présenterons ensuite la sélection des composants.

4.3.2 Traitement de la requête de l'application

Le traitement d'une requête de services exprimée par l'application regroupe l'ensemble des fonctions qui sont appliquées (éventuellement) à cette requête avant de la transmettre à la phase de sélection des composants. Ces traitements consistent globalement à ajouter ou à enlever des services de la requête. L'ajout des services concerne la gestion des dépendances, des services par défaut et des services de monitoring. La suppression des services concerne la gestion des incompatibilités [41].

Bien que ces fonctions ne permettent pas la sélection proprement dite des composants, elles l'influencent grandement. Les compositions peuvent varier de façon significative selon le résultat de ces traitements. Pour cette raison, nous les incluons dans le processus de décision. On note ici que tous ces traitements sont effectués avant la phase de négociation des services avec le système distant (voir Chapitre 2).

Dépendances et incompatibilités

Les dépendances concernent des services dont dépend d'autres services. C'est à dire des services sans lesquels d'autres services ne peuvent pas fonctionner. La liste des dépendances fait partie de la description de chaque service dans la base globale. A la réception de la requête de l'application, et pour chaque service requis, l'interface application recherche dans la description correspondante et récupère la liste de ses dépendances. Si l'un de ces services ne figure pas dans la requête de l'application, il y est ajouté dans la partie obligatoire. L'intervention de l'application pour donner l'accord quant à l'ajout des dépendances peut varier selon la politique d'administration de la couche Transport (nous en parlerons au Chapitre 5). L'algorithme 4 de gestion des dépendances est décrit ci-après.
Bien que l'algorithme comporte plusieurs boucles imbriquées, le ramenant à une complexité polynomiale de l'ordre de $O(SR2D)$, les valeurs de R et D sont relativement petites, rendant l'exécution effective de l'algorithme légère.

Les incompatibilités sont traitées par un processus similaire. La liste des compatibilités est présente dans la description de chaque service. Pour chaque service de la requête, la couche Transport récupère la liste de ses dépendances ; si un de ces services est présent dans la partie obligatoire de la requête de l'application, la requête est rejetée. S'il s'agit d'une incompatibilité présente dans la partie optionnelle, le service en question est retiré de la requête. L'algorithme 6correspondant se traduit ainsi :

Services par défaut

La gestion des services par défaut permet à la couche Transport de garantir certaines fonctionnalités, jugées incontournables selon le contexte, sans faire intervenir les applications. La liste des services par défaut fait partie des bases de données de la couche Transport, et gérés par l'administrateur du système, indépendamment de l'applications et des autres systèmes supportant l'architectures. Il se peut de ce fait que cette liste diffère d'un système à un autre. Ces services peuvent aussi être divisés en services

Algorithm 4 Gestion des dépendances

Require:
The list of services S
The list of a service dépendances D
The application request R
The boolean inRequest

```
1: for all requestedservices : S_i do
2:   for all listofservices : S_j do
3:     if S_i = S_j then
4:       D ← Dependances(S_i)
5:       for all dependances : S_k do
6:         inRequest ← false
7:         for all requestedservices : S_n do
8:           if S_n = S_k then
9:             inRequest ← true
10:          end if
11:        end for
12:        if notinRequest then
13:          add(S_k, R)
14:        end if
15:      end for
16:    end if
17:  end for
18: end for
```

Algorithm 5 Gestion des incompatibilité

Require:
The list of services S
The list of a service incompatibilities P
The application request R
The boolean inRequest

```
1: for all requestedservices : S_i do
2:   for all listofservices : S_j do
3:     if S_i = S_j then
4:       D ← incompatibilities(S_i)
5:       for all incompatibilities : P_k do
6:         inRequest ← false
7:         for all requestedservices : S_n do
8:           if S_n = S_k then
9:             inRequest ← true
10:          end if
11:        end for
12:        if inRequest then
13:          if S_k.Optional then
14:            remove(S_k, R)
15:          else
16:            exit()
17:          end if
18:        end if
19:      end for
20:    end if
21:  end for
22: end for
```

obligatoires et services optionnels, pour permettre la négociation avec les systèmes distants.

Le principe devient très simple : il suffit de parcourir la liste des services par défaut, et de les ajouter dans la requête de l'application, soit dans la partie obligatoire, soit dans la partie optionnelle. Il n'est pas nécessaire de vérifier leur présence au préalable dans la requête de l'application, ces services ne sont visibles qu'en interne. L'algorithme **??** de gestion des incompatibilités est décrit ci-après.

Algorithm 6 Gestion des incompatibilité

Require:
The list of default services S
The application request R
1: **for all** $requestedservices : S_i$ **do**
2: **if** $S_i.optional$ **then**
3: $addOptional(S_i, R)$
4: **else**
5: $addMandatory(S_i, R)$
6: **end if**
7: **end for**

Composants de monitoring

Les composants de monitoring sont des composants particuliers servant à effectuer le monitoring d'une valeur spécifique. Leur particularité concerne uniquement l'interface application. Le reste des composants de gestion (décision, gestionnaire des connexions, assembleurs ...) ils sont traités comme des composants ordinaires. Ces services sont ajouté seulement en dernier recours ; c'est-à-dire lorsqu'aucun des composant de communication correspondants aux service requis par l'application, ou aux services par défaut, ne permet le monitoring d'une valeur dont un des composants précédents ou la composant de décision a besoin.

L'algorithme correspondant (7) doit en premier récupérer la liste des valeurs de monitoring que propose les services présents dans la requête (après gestion des dépendances et des incompatibilités, et insertion des services par défaut), ainsi que les valeurs monitorées par ceux-ci. Ensuite l'algorithme compare les deux ensembles, et complète, éventuellement, la requête de l'application par les services correspondants aux valeurs de l'ensemble des besoins qui ne sont pas présentes dans l'ensembles des valeurs monitorées.

4.3.3 Sélection des composants

La sélection des composants est la dernière étape du processus de décision. Elle permet de traduire une liste de services en une liste de composants de communication. La sélection se base sur la requête émise par l'application, après traitement par l'interface application, et sur l'état du réseau. Cependant, et comme introduit plus haut, la prise en compte de l'état du réseau n'est pas systématique. Elle est exclue dans deux situa-

Algorithm 7 Gestion des composants de monitoring

Require:
The list of monitoring services S
The application request R
The list of monitored values M
The list of needed values N
The list of temporary values T
The boolean inList

```
1: for all requestedservices : S_i do
2:    T ← S_i.MonitoredValues
3:    for all temporaryvalues : T_i do
4:       add(T_i, M)
5:    end for
6:    T ← S_i.NeededValues
7:    for all temporaryvalues : T_i do
8:       add(T_i, N)
9:    end for
10: end for
11: for all neededvalues : N_i do
12:    inList ← false
13:    for all monitoredvaluesM_j do
14:       if N_i = Mj then
15:          inList ← true
16:       end if
17:    end for
18:    if notinList then
19:       for all monitoringservices : S_k do
20:          if S_k.Value = N_i then
21:             add(S_j, R)
22:          end if
23:       end for
24:    end if
25: end for
```

tions : lors de la première sélection où l'état du réseau n'est pas encore connu, et lorsque l'information sur l'état du réseau est absente faute de composant de monitoring.

Cette sélection se ramène à un problème d'aide à la décision multicritères, qui consiste à optimiser une fonction de choix parmi l'ensemble des éléments disponibles. Parmi les approches de décisions décrites précédemment, nous avons opté pour l'approche par agrégation complète en utilisant une fonction d'utilité. Le choix de cette approche se justifie par plusieurs facteurs, dont le plus important est le fait que les approches par agrégation complète sont les seules à pouvoir garantir un choix unique dans toutes les situations. En effet, étant donné que le résultat de la décision est à destination d'un composant logiciel, l'existence de plusieurs choix possibles sans moyen de distinction va amener inévitablement l'architecture à faire des choix arbitraires.

Un autre facteur est l'existence d'un élément supplémentaire de comparaison, commun à tous les critères, ce qui n'est pas fréquent dans les problèmes de décision. En effet, il ne s'agit pas de trouver le meilleur composant de communication de façon absolue, mais de trouver le composant le plus adapté à un profil du réseau particulier. Pour chaque profil, le résultat de décision peut être différent. Ceci vient du fait que les composants sont décrits par un intervalle d'adaptabilité. La fonction d'utilité permet d'intégrer facilement le profil réseau.

L'approche par agrégation complète nécessite cependant certaines hypothèses pour avoir du sens, notamment en relation avec la comparabilité entre critères hétérogène. Dans notre problématique, la fonction d'utilité permet d'identifier l'appartenance ou non à un intervalle, ce qui permet d'avoir le même résultat quel que soit la nature du critère en question, et sans avoir de connaissances préalables sur sa sémantique. La fonction d'utilité U peut donc être formulée ainsi

Soit un ensemble de critères $Cr = Cr_1, Cr_2, ..., Cr_n$, soit un ensemble de composants $Cm = Cm_1, Cm_2, ..., Cm_n$, et soit un profil de réseau Pr de Cr dans $\mathbb{R}^n$, tel que, pour chaque Cr_i de Cr associe une valeur dans $\mathbb{R}$. La fonction d'utilité U se définit alors :

$$U\big(Pr(Cr), Cm_i\big) = \prod_{i=1}^{n} U'(Pr_i, Cm_i)$$

$$U'(Pr_i, Cm_i) = 1 \text{ si } Pri \in Cmi \text{ , } 0 \text{ si non.}$$

Littérairement, cette fonction vaut 1 pour un composant donné si et seulement si toutes les valeurs du profil en cours se situent dans les intervalles respectifs d'adaptabilité du composant en question. Elle permet donc d'associer l'ensemble des critères ainsi que le profil réseau en une seule valeur de retour. Et le problème de décision revient donc à simplement optimiser la fonction d'utilité U. Formellement :

$$\text{Min} \left\{ U\big(Pr(Cr), Cm_1\big), U\big(Pr(Cr), Cm_2\big), ..., U\big(Pr(Cr), Cm_n\big) \right\}$$

Une autre hypothèse, qui garantit que l'optimum sur l'ensemble Cm est un singleton, est le fait qu'il n'existe aucun couple de composants dont les intervalles d'adaptabilité sont équivalents pour l'ensemble des critères. Cette hypothèse est garantie par le

composant d'intégration qui s'assure de n'ajouter un composant que s'il présente des différences, au moins pour un critère, dans ses intervalles d'adaptation par rapport aux composants déjà existants. Donc formellement :

$$\forall Cm_1 \in Cm, \forall Cm_2 \in Cm, \exists Cr_i \in Cr / I(Cm_1, Cr_i) \neq I(Cm_2, Cr_i)$$

Avec $I(Cm_i, Cr_j)$ est l'intervalle d'adaptabilité d'adaptabilité du composant i pour le critère j. Une dernière hypothèse est l'absence de chevauchement entre les intervalles d'adaptabilité entre les composants pour chaque critère.

$$\forall Cm_1 \in Cm, \forall Cm_2 \in Cm, \forall Cr_i \in Cr / I(Cm_1, Cr_i) \cap I(Cm_2, Cr_i) = \phi$$

De cette façon, on garantit que le résultat du problème d'optimisation est soit un singleton, soit un ensemble vide. Dans le cas d'un ensemble vide, l'algorithme de décision choisit un composant par défaut, qui est unique et présent pour chaque service, et ce bien entendu, sans tenir compte du profil du réseau.

Sélection par défaut

Le mode de sélection par défaut est invoqué à la première instanciation de la connexion suite à une requête de l'application, ou alors lorsque le profil du réseau n'est pas assez complet pour établir une sélection fine. Dans ce dernier cas, il est possible de considérer uniquement la partie du profil réseau disponible, mais ceci peut nuire aux performances lorsque le composant choisi est spécifique à un contexte différent, et que cette différence réside au niveau du critère non monitoré. Par conséquent, nous partons du principe qu'une solution générale reste toujours un meilleur choix qu'une solution spécifique utilisée hors son contexte d'adaptabilité.

Dans ce mode, le problème est relativement simple, l'algorithme recherche le composant par défaut correspondant au service en question. Ce composant est unique (c'est garanti par le composant d'intégration), il n'existe donc qu'un seul composant par défaut pour chaque service. L'algorithme 8 est décrit en pseudo-code ci-après :

Algorithm 8 Sélection par défaut

Require:
The application request R
The list of components C
The list of selecte components S

```
1: for all requestedservices : S_i do
2:   for all components : C_j do
3:     if (C_j.Default)and(C_i.Service = S_i) then
4:       add(C_i, S)
5:       Break
6:     end if
7:   end for
8: end for
```

Sélection adaptative

Dans le cas où le profil réseau est connu, soit au début de la communication, soit suite à un changement de contexte, la sélection sera adaptative pour tenir compte de l'état du réseau. L'algorithme calcule une fonction d'utilité pour chaque composant, et en choisit un. Les hypothèses garantissant que le résultat est soit unique soit inexistant, le choix se portera sur le maximum des valeurs d'utilité. Comme les valeurs d'utilité sont soit 0 soit 1, l'algorithme recherche le composant ayant la valeur 1. Une version optimisée de l'algorithme évite de faire les calculs d'utilité pour tous les composants. L'algorithme 9 de base est décrit ci-après.

Algorithm 9 Sélection adaptative - Version de base

Require:
The application request R
The list of components C
The list of selecte components S
The list of network profile P
The utility value U

1: **for all** $requestedservices : S_i$ **do**
2: **for all** $components : C_j$ **do**
3: **if** $C_j.Service = S_i$ **then**
4: $U \leftarrow 1$
5: **for all** $profilecriteria : P_k$ **do**
6: **if** $(P_k.Value < max(C_j, P_k))$**and**$P_k.Value > min(C_j, P_k))$ **then**
7: $U = 0$
8: $Break$
9: **end if**
10: **end for**
11: **end if**
12: **end for**
13: **if** $U = 1$ **then**
14: $add(C_i, S)$
15: $Break$
16: **end if**
17: **end for**

L'algorithme s'arrête au moment où le composant approprié est trouvé (du fait qu'il n'en existe qu'un seul). Mais dans le cas où le composant en question se trouve en fin de liste, ou que le profil du réseau ou la liste des composants ne permettent pas de trouver un composant adapté, l'algorithme est obligé de parcourir toute la liste. Sa complexité dans ce cas est polynomiale, de l'ordre de $O(RCP)$. Nous proposons pour réduire le temps d'exécution de l'algorithme une version plus optimale qui procède par élimination, sans avoir à calculer la fonction d'utilité pour l'ensemble des critères. En fait, l'algorithme choisit les composants adaptés au profil réseau selon un seul critère, ensuite, une même sélection est faite sur le sous-ensemble résultant de la première sélection selon le deuxième critère, et ainsi jusqu'à parcourir l'ensemble des critères. De cette façon le nombre de comparaisons n'est égale au nombre des composants que pour le premier critère. A partir de là, l'ensemble des choix diminue après chaque critère.

De plus, cette version de l'algorithme permet de détecter plus rapidement l'absence d'une solution adaptée sans passer par tous les composants et pour chaque critère. Sa complexité dans ce cas est très difficile à estimer, du fait que le nombre de composants à éliminer pour chaque critère est arbitraire. L'algorithme 10 est décrit ci-après.

Algorithm 10 Sélection adaptative - Version optimisée

Require:
The application request R
The list of components C
The list of selecte components S
The list of network profile P
The utility value U

```
1: for all requestedservices : S_i do
2:   for all profilecriteria : P_k do
3:     for all components : C_j do
4:       if (C_j.Service = S_i)and(P_k.Value < max(C_j, P_k))andP_k.Value >
         min(C_j, P_k)) then
5:         remove(C_i, C)
6:       end if
7:       if C.Empty then
8:         Break
9:       end if
10:    end for
11:    if C.Empty then
12:      Break
13:    end if
14:  end for
15:  if C.Empty then
16:    add(default(S_i), S)
17:  else
18:    add(C_i, S)
19:  end if
20: end for
```

Conclusion

Nous avons présenté dans ce chapitre la partie autonomique de notre architecture avec l'ensemble de ses fonctions ainsi que les algorithmes associés. Nous avons aussi présenté des bases théoriques pour les problèmes conceptuels ainsi que les problèmes techniques où cela s'avérait nécessaire. A ce stade, la plupart des fonctionnalités de notre architecture ont été exposées. Il reste à présent la gestion de l'environnement de déploiement et les cas particuliers. Nous consacrerons le prochain chapitre pour discuter de toutes ces particularités. Ce dernier chapitre est dédié à l'évaluation de ce qui a été présenté jusque-là.

Chapitre 5

Implémentation et Évaluation

Nous avons décrit, dans les chapitres précédents, le fonctionnement de l'architecture que nous proposons. Nous sommes partis des limites de l'architecture actuelle face à l'évolution de la couche Transport. Nous avons identifié les besoins associés et proposé une nouvelle architecture extensible et auto-adaptative vis-à-vis des applications et des réseaux. Les aspects extensibles et auto-adaptatifs de l'architecture ont été présentés de façon exhaustive.

Dans ce dernier chapitre, nous passons en revue les détails pertinents de l'implémentation effective de l'architecture, notamment en rapport avec son déploiement dans le noyau. Ensuite, nous procédons à son évaluation dans le cadre d'un cas d'étude. Le cas d'étude se positionne au niveau des communications aéronautiques par satellite. Nous y considérons une double problématique : l'adaptation des connexions aux différents liens physiques disponibles, ainsi que la gestion de la transition des protocoles sous-jacents vers un nouveau standard. Les tests concernent la faisabilité de la solution, ses gains ainsi que son coût.

Le chapitre est structuré en deux parties. La première partie décrit les détails d'implémentation et déploiement de l'architecture. Elle notamment l'interaction entre composants et les principes d'intégration dans le noyau. La deuxième partie décrit le cas d'études associé aux communications aéronautiques ainsi que le principe de déploiement de l'architecture. Enfin, la deuxième partie décrit les test de validation incluant l'équivalence de performance de l'architecture par rapport aux protocoles existant, le gain en adaptabilité ainsi que le surcoût associé.

5.1 Implémentation et déploiement

A ce stade, notre architecture a été implémentée et testée dans l'espace utilisateur. L'implémentation a été réalisée en langage C++ à la fois pour passer facilement de la conception à l'implémentation par le biais du paradigme de l'orienté objet, mais aussi pour faciliter la transition vers le langage C en vue d'une intégration au noyau. En outre, le langage C++ permet une gestion efficace de la mémoire, ce qui permet de gagner en performances.

Dans cette section nous présentons, en premier lieu, les détails de mise en oeuvre de notre architecture en expliquant certains choix d'implémentation des composants et le mode de leurs interactions. Cette description a un double objectif : d'une part, elle explicite certains des défis d'implémentation associés aux nouvelles fonctionnalités de notre architecture, telles que la gestion autonome des évènements, ou certaines caractéristiques telles que l'extensibilité ; d'autre part, elle vise à conforter la deuxième partie de ce chapitre sur l'évaluation des performances en précisant les choix susceptibles d'avoir un impact sur les résultats.

En deuxième lieu, nous présentons les principes de déploiement effectifs de l'architecture dans le noyau du système d'exploitation. Ce déploiement n'ayant pas été traité dans notre travail, nous présentons cependant le changement à effectuer, les problèmes résultants et la façon avec laquelle il était prévu de les taiter.

5.1.1 Intégration et interaction des composants

Les composants de gestion et de communication sont tous implémentés dans des classes. Les composants de communication héritent d'une classe de base abstraite qui leur impose d'implémenter les fonctions nécessaires à leur intégration, leur sélection et au final leur utilisation.

Pour leur intégration, les composants de communication doivent fournir les méthodes nécessaires qui permettent leur description, notamment en termes de service offert et de plage d'adaptabilité. Ainsi, le composant d'intégration peut effectuer les vérifications nécessaires et mettre à jour les bases de données internes.

Notons qu'en vue d'une application réelle, les composants auront à être certifiés par une autorité commune pour tous les systèmes supportant l'architecture. Cette partie dépasse le cadre de ce travail et elle n'est pas traitée.

Les composants doivent implémenter d'autres méthodes pour permettre leur sélection. Les méthodes en question ne concernent pas leur description, puisque celle-ci est déjà définie dans la liste des composants gérée par le composant d'intégration. Ils doivent fournir des méthodes pour récupérer les valeurs qu'ils peuvent monitorer ainsi que les valeurs de monitoring dont ils ont besoin. En outre, ils doivent fournir une méthode leur permettant d'accéder aux références de ces valeurs.

Ce modèle de gestion des composants rejoint l'aspect extensible de l'architecture. En effet, pour pouvoir fonctionner avec des composants qui n'étaient pas connus au moment de sa conception, mais surtout avec des composants dont elle ignore la sémantique, l'architecture doit pouvoir récupérer les informations de chaque composant de

façon dynamique. Il en est de même pour la création des références aux valeurs monitorée et pour l'établissement du lien entre les composants qui partagent ces valeurs et ceux qui les utilisent.

De la même façon, l'interaction entre les différents composants de l'architecture rejoint aussi cet aspect extensible, notamment quant au choix de la nature synchrone ou asynchrone de ces interactions. En effet, un lien synchrone assure une hiérarchie entre les composants dans le sens appelant vers appelé, mais suppose une connaissance préalable de la structure du composant pour invoquer la fonction concernée. A l'inverse, les liens asynchrones offrent davantage de flexibilité et sont donc plus extensibles, notamment du fait que la communication est basée sur l'information partagée et non pas sur le composant qui la partage. Le choix du type de liens entre les composants dépend donc des besoins en extensibilité et en hiérarchie d'invocation.

Entre composants du même type, l'interaction n'est jamais synchrone, à l'inverse de l'interaction entre les composants de gestion et ceux de communication. Techniquement, le rôle des composants de communication est d'offrir un certain nombre de fonctions selon un modèle commun. Ces fonctions sont destinées à être invoquées par la partie gestion de l'architecture, que ce soit pendant leur intégration, leur sélection ou pendant leur utilisation par les connexions. De ce fait, l'interaction entre les composants de gestion et ceux de communication est toujours synchrone, par invocation de ces fonctions, mais uniquement dans cet ordre-là. C'est-à-dire que les composants de communication ne peuvent invoquer des fonctions d'autres composants, qu'ils soient de gestion ou de communication.

Exceptionnellement, à l'intérieur des connexions, le module de monitoring utilise les informations réseau relevées par les composants de communication afin d'établir des profils du réseau. C'est la seule situation où l'interaction se fait en mode asynchrone.

Entre les composants de communication par contre, l'interaction est toujours asynchrone, via des mémoires partagées. Ceci étant, cette interaction n'a pas pour objet de provoquer une réaction précise de la part du composant en interaction avec celui initiateur, mais uniquement de partager l'information en question. En fait, les composants de communication ne savent même pas avec qui il partagent leurs valeurs, ni qui partage les leurs, ni d'ailleurs si leurs valeurs sont utilisées réellement ou pas. Ces différentes interactions sont illustrées au diagramme de la Figure 5.1.

5.1.2 Principes d'intégration au noyau

Tel que décrit précédemment, l'architecture a été implémentée en langage objet (ici C++) dans l'espace utilisateur. En effet, une intégration complète dans le noyau n'entrant pas dans le cadre de notre objectif global, nous avons opté pour une solution qui facilite les tests, et qui offre en même temps de bonnes performances tout en facilitant le passage à un langage classique (typiquement le C). Ceci étant, nous présentons ci-après l'approche envisagée pour une intégration et un fonctionnement de notre architecture dans le noyau.

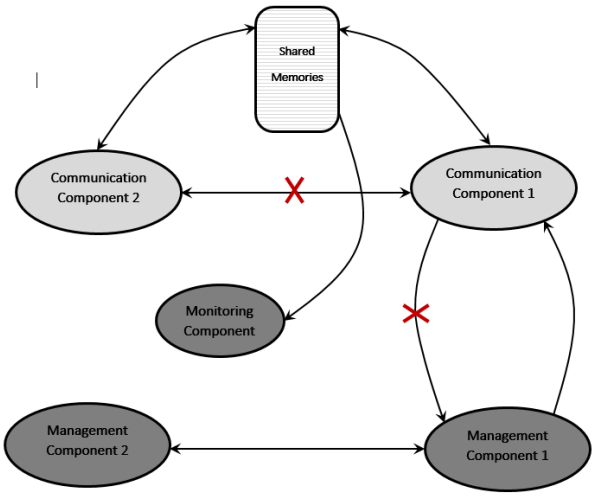

FIGURE 5.1 – Principe d'interaction entre les composants de l'architecture

Modules du noyau, chargement et déchargement

Les modules du noyau sont des "bouts" de code compilés indépendamment. Leur intérêt est de permettre l'utilisation de composants qui n'étaient pas disponibles au moment de la compilation du noyau. Il peut s'agir d'extensions du système ou de composants développés par des tiers. L'autre intérêt est de ne pas avoir à recompiler l'ensemble du noyau pour en modifier ou mettre à jour une partie. Enfin, les modules permettent de ne charger que ce qui est nécessaire, évitant ainsi de surcharger le noyau par des parties qui ne sont pas du tout ou qui sont rarement utilisées [85].

Ces modules peuvent être chargés ou déchargés du noyau à tout moment. Une fois chargés, les modules fonctionnent comme des éléments de base du noyau, avec les mêmes privilèges que ceux-ci. Le chargement des modules peut se faire de deux façons : soit statiquement, en incluant l'opération dans le code du noyau, soit dynamiquement en utilisant les fonctions système prévues à cet effet [86].

Un des exemples les plus courants de modules de noyau sont les drivers de périphériques. L'utilisation modulaire des drivers regroupe les trois intérêts des modules de noyau. En effet, les systèmes n'incluent que les drivers connus au moment de leur développement ; par conséquent, les nouveaux périphériques doivent ajouter leurs drivers après que le système ait été installé, ce qui amène le système à travailler avec des éléments qui n'étaient pas présents lors de sa compilation. Ensuite, même les drivers installés ne sont pas toujours utilisés, par détachement ou désactivation de certains périphériques. Ainsi, le système charge uniquement les drivers qui peuvent être utilisés à un instant donné. Enfin, les drivers sont sujets à de fréquentes mises à jour, notamment pour des corrections de bugs ; de cette façon, le système n'a pas à être recompilé dans son intégralité pour modifier un driver [87].

Notons que même si une partie du système est développée sous forme d'un module, son utilisation n'est pas forcément dynamique. Il existe en effet des modules du système qui sont chargés systématiquement, généralement au chargement du système, et dont l'appel est fait statiquement au moment de la compilation du noyau.

Pour les modules d'extension, le système impose une structure bien définie afin qu'il

puisse les invoquer. Cette structure peut être générique ou spécifique à un type particulier de modules. Dans un cas générique, le module doit implémenter au moins deux fonctions qui seront appelées respectivement au chargement et au déchargement du module.

Pour des cas spécifiques en revanche, c'est un peu plus complexe. En effet, les modules spécifiques définissent une variable de type d'une structure prédéfinie par le système. Cette structure contient un certain nombre de champs d'informations d'une part, et de pointeurs de fonctions d'autre part. Les champs d'information servant à identifier le module en question afin de le relier à un événement ou un périphérique spécifique.

Un exemple consiste en les drivers de cartes réseaux. Ces drivers sont traités de façon spécifique par le système d'exploitation. Sous Linux, une structure de donnée spécifique, net-device, doit être implémentée par tout driver d'une carte réseau. Cette structure comprend, entre autres, en plus des pointeurs vers les fonctions communes à tous les modules pour leur chargement et déchargement, des champs pour l'identification de la carte, son nom, etc. En outre, elle contient des pointeurs vers des fonctions permettant (par exemple) d'envoyer et de recevoir des données. Les modules contenant ces drivers doivent renvoyer une variable du type de cette structure, pour que le système puisse à la fois les sélectionner et les invoquer.

Principes pour l'intégration au noyau

Pour une intégration dans le noyau, notre architecture doit d'abord être traduite dans un langage basique, en l'occurrence le langage C. Étant déjà écrite en C++, la transition sera relativement simple. La partie la plus importante concerne les composants de communication. En effet, pour les composants de gestion, le fait de les avoir conçus sous forme de composants constitue un choix conceptuel et non pas un besoin fonctionnel. Il est par conséquent possible de retirer l'aspect objet des composants de gestion, et les développer comme de simples fonctions. Il faudra cependant définir les structures de données nécessaires au passage des informations entre les différentes fonctions.

Les composants de communication, ainsi que les connexions les contenant, doivent préserver leur indépendance, même dans une conception procédurale. Tous les composants de communication doivent être implémentés sous forme de modules indépendants de l'architecture, et doivent pouvoir être utilisés par la partie de gestion sans avoir à les connaître par avance.

Les modules seront gérés tels des modules de drivers, en implémentant une structure de données commune à tous les composants de communication. Cette structure viendra remplacer la classe abstraite de laquelle dérive actuellement ces composants. Les champs des classes actuelles resteront inchangés, et les méthodes utilisées par la partie de gestion seront remplacées par des pointeurs de fonctions équivalentes. Chaque module doit par conséquent définir une variable de ce type de structure, remplir les différents champs par les informations nécessaires, et affecter aux différents pointeurs les fonctions correspondantes.

Les connexions seront elles-aussi des instanciations d'une structure commune contenant les mêmes éléments que la classe correspondante actuellement. Le gestionnaire des connexions tient à jour un tableau de références à ces connexions et les gère de la

même façon que dans le cadre de notre implémentation actuelle (en mode objet dans l'espace utilisateur).

Dans le noyau, la partie de gestion sera chargée par défaut au lancement du système, et fera partie du code du noyau (figure). Les composants de communication, en revanche, seront chargés au besoin selon les requêtes des applications. Pour assurer le chargement d'une seule instance de chaque composant, la partie de gestion enregistre des références vers tous les modules chargés. Ces modules seront déchargés dès lors qu'aucune connexion ne les utilise. Nous rappelons ici la figure 5.2 du chapitre 3, qui illustre le principe de la séparatinon du noyau.

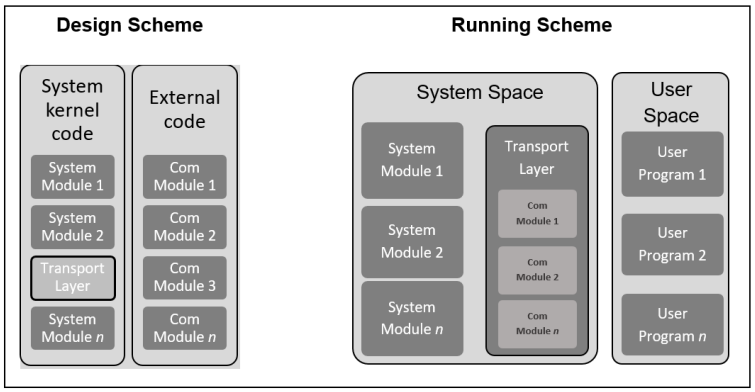

FIGURE 5.2 – Principe de la séparation du noyau

5.1.3 Décision prédéfinie

Notre architecture a été conçue pour être extensible et auto-adaptative, selon un principe générique d'inconscience de la sémantique des éléments manipulés, et ne nécessitant aucune intervention externe.

Cependant, il se peut que dans certains environnements critiques, la décision ne soit pas laissée à la charge de la machine. C'est le cas notamment lorsque la certification des algorithmes de décision est soit très difficile, soit irréalisable.

Dans de telles situations, l'architecture peut fonctionner selon un mode manuel pour ce qui concerne la définition des contextes réseau et l'association des composants. En effet, la partie décisionnelle ne sera plus basée sur une détection de changement de contexte, ni sur des fonctions de sélection par valeur d'utilité, mais uniquement par des tests d'appartenance. Dans ce mode, une liste de contextes réseau sont définis avec des intervalles correspondants à chaque valeur de monitoring considérée. Ensuite, une liste de correspondance contexte/composant est définie pour chaque service.

Un monitoring du réseau sera effectué comme dans le cas nominal ciblé, mais sans détection de changement de contexte. A la place, un test d'appartenance est effectué en temps réel pour vérifier si les composants déployés correspondent au profil réseau actuel (ce test étant effectué précédemment uniquement en cas de détection d'un changement de contexte). Si une non-correspondance est détectée pendant un certain laps de temps, qui lui aussi est prédéfini, l'architecture change de composition.

L'approche orientée service peut en revanche toujours continuer à être utilisée sans risque sur de tels système, à partir du moment où les applications sont certifiées.

5.2 Cas d'étude et évaluation

Dans cette section nous examinons les performances générales de l'architecture que nous proposons, ses gains ainsi que ainsi que le surcoût engendré. Nous présentons d'abord le cas d'étude utilisé pour l'évaluation, le principe de cette évaluation, et enfin les scénarios de test et les résultats obtenus.

5.2.1 Cas d'étude

L'évaluation de notre architecture est effectuée dans un contexte de communications aéronautiques. Nous décrivons dans ce qui suit la problématique dans le cadre de laquelle s'inscrit notre cas d'étude, ainsi que les besoins associés.

Problématique et besoins

Le cas d'étude comprend deux problématiques couvrant à la fois l'aspect protocolaire et l'aspect performance.

Le premier aspect concerne la gestion de la phase de transition de la pile protocolaire ATN/SI à celle de l'ATN/IPS. Ces deux architectures ont été présentées au chapitre 1. Durant cette phase, les deux piles devront coexister et seront amenées à communiquer. Dans ce cas, la couche Transport doit faire abstraction aux applications du monde aéronautique des services réseaux sous-jacent, et doit aussi inclure les deux couches de session et de présentation (présentes dans la pile ATN/SI) à la couche Transport, ceci de façon transparente à l'application.

Le deuxième aspect concerne la gestion des différents liens physiques utilisés par les équipements aéronautiques. Face à l'importante hétérogénéité qui existe entre les liens physiques du réseau ATN, la couche Transport doit adapter son comportement en fonction des caractéristiques du lien sous-jacent. Par ailleurs, et pour des questions de sécurité, notamment de certification, la couche Transport doit se baser, lors de la sélection des composants de communication, sur des compositions certifiées prédéfinies.

Enfin, la couche Transport doit gérer de façon transparente les deux types d'applications qui vont coexister durant la phase de transition. Elle doit pouvoir, à la fois, supporter les interfaces des deux types d'applications, mais aussi gérer les communications éventuelles entre des applications de types différents.

Gestion des applications

Lors de la transition au modèle IPS du réseau ATN, les standards [52] prévoient qu'il n'y ait pas de modification, ou au moins que ces modifications soient réduites au minimum, au niveau de la couche application. L'accès aux services de communication pour les applications se fait via l'interface DS (Dialogue Service). Les standards proposent alors une modification de cette interface afin de convenir aux nouvelles spécifications

du modèle IPS. Ainsi, lors de la phase de transition, une seule version des applications et du DS existeront, et l'accès aux communications se fera via la pile IPS.

La gestion des applications legacy, c'est-à-dire écrites pour la pile ATN/OSI, est déjà réalisée au niveau des DS[1], telle que prévu dans les standards. Cependant, les DS conçus pour la pile IPS ne prévoient pas une interconnexion via l'ancienne pile ATN/OSI. Les nouveaux DS ne pourront pas communiquer avec des systèmes n'implémentant pas la couche IPS. La seule hétérogénéité qui est prise en charge se situe donc au niveau applicatif.

Les DS gèrent les applications legacy, qui par conséquent ne vont pas différer des nouvelles applications écrites pour la pile IPS, en conservant les mêmes primitives d'accès au service de communications, et en traduisant les requêtes classiques en des requêtes TCP ou UDP. Il ne s'agit cependant pas d'un simple changement de protocole (TP4 vers TCP), mais de la " fusion " de trois couches de communication (session, présentation et transport) en une seule couche de Transport.

Par conséquent, pour pouvoir faire communiquer des systèmes ayant des piles protocolaires différentes, deux approches de solution sont possibles, en modifiant le DS IPS ou en le conservant (voir Figure 5.3) :
– Suivant la première approche, la couche Transport offrira deux points d'accès différents pour les deux piles protocolaires, et selon la nature du système distant, le DS choisira la pile correspondante au niveau local.
– Suivant la deuxième approche, l'hétérogénéité des systèmes sera transparente pour le DS, et sa gestion se fera au niveau de la couche Transport.

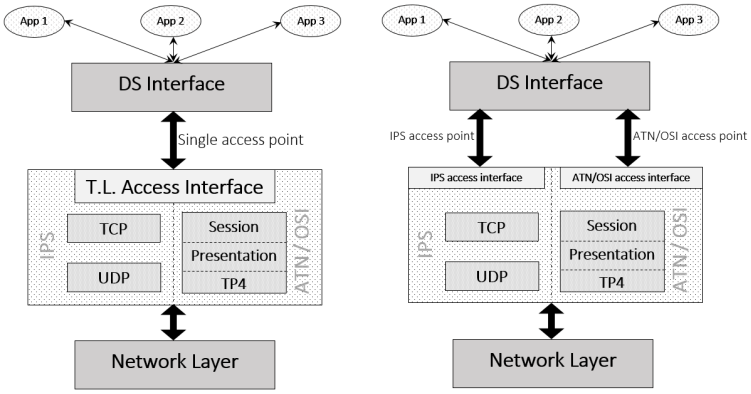

FIGURE 5.3 – Modèles de prise en compte de la transition vers les standards IPS

La première approche est plus simple mais nécessite une évolution des standards afin de pouvoir détecter la nature du système distant. Cela étant, cette découverte se fait d'un seul côté (initiateur de la communication), et ne nécessite pas une évolution d'un protocole particulier. Le besoin en standardisation n'est donc pas trivial, et dépend de la politique du domaine aéronautique. Ceci concerne aussi la pile protocolaire à choisir en priorité, bien qu'intuitivement l'usage de la pile IPS soit plus évident.

Dans le deuxième cas, c'est la couche Transport qui s'occupe de la détection de la nature du système distant. Ainsi, les évolutions vont se situer uniquement au niveau

1. DS, pour Dialogue Service, est l'interface d'accès aux services réseau de l'ATN (voir chapitre 1)

Transport ; les applications et l'interface DS n'auront pas à être modifiés. C'est ce cas de figure qui nous intéresse, comme application à notre architecture. En effet, l'architecture est, par conception, prévue pour assurer la détection des capacités du système distant, notamment quant à la présence de l'architecture sur ce dernier. C'est cette partie qui sera modifiée et adaptée au modèle aéronautique afin de gérer la coexistence des deux piles protocolaires durant la phase de transition. Nous présentons cette approche dans la section ci-après.

Gestion des communications hétérogènes

La gestion des deux piles protocolaires se traduit par trois niveaux de décision : les besoins de l'application, les liens physiques et les protocoles déployés. Dans son modèle générique, l'architecture prend en compte uniquement les deux premiers paramètres. L'intégration du paramètre relatif aux protocoles déployés doit se faire de façon transparente au composant de décision. Il sera intégré à la phase de traitement de la requête de l'application par l'interface application (avant son passage au module de décision).

De leur côté, les services associés aux composants sont définis en associant la pile protocolaire à laquelle ils correspondent. Ainsi, l'interface application définit la liste des services dans la requête qui sera passée au module de décision. Bien entendu, une découverte de la pile déployée dans le système distant doit s'effectuer en amont, pour chaque connexion, suivant le même processus prévu dans l'architecture générique (voir Chapitre 2.). A ce niveau aussi, un ordre de découverte doit être spécifié pour définir la pile protocolaire prioritaire à privilégier lorsqu'elle est disponible.

Le processus de décision reste inchangé quant à la présence des deux piles protocolaires, la gestion de l'hétérogénéité est donc traitée au niveau service par l'interface application. Pour les communications ATN/OSI, des composants assurant les services des couches sessions et présentation doivent être déployés. Enfin, l'interface réseau différencie les couches basses à invoquer selon le type de réseau de bout en bout. Si la couche réseau est générique, cette différenciation n'est plus nécessaire.

Un autre changement doit être fait au niveau du monitoring. En effet, dans l'architecture générique, le monitoring est basé sur la détection de changement de contexte, qui lui est basé en partie sur l'identification des pics. Par conséquent, la vérification de l'adaptabilité de la composition en cours est faite uniquement après détection d'un changement de contexte. Dans le contexte de l'ATN, les différents profils réseau à considérer sont identifiés auparavant afin de permettre des résultats déterministes au niveau autonomique. Le monitoring se basera donc sur ces profils, qui sont constitués d'intervalles de valeurs pour chaque mesure de monitoring. Les fonctions de détections de pics ne seront plus nécessaires, et les le monitoring sera plus simple. Il s'agit en fait d'utiliser une moyenne glissante et tester l'appartenance de sa valeur à l'intervalle du profil réseau en cours, et ce pour chaque valeur monitorée. Le module de décision est invoqué dès lors qu'un dépassement des bornes du profil en cours est détecté.

Compositions prédéfinies

Etant donné le caractère critique des communications dans les environnements aéronautiques, les modules logiciels intégrés dans un système subissent des procédures de certification minutieuses avant de pouvoir être utilisés. Par conséquent, la délégation

de l'établissement des compositions à un algorithme basé sur un modèle de décision peut ne pas convenir au mode d'intégration du monde aéronautique.

Dans ce type de situations, le module de décision peut être configuré de façon manuelle de telle sorte à imposer des compositions précises pour chaque contexte réseau. Le principe est alors de disposer d'une liste de compositions (groupes de composants) associés à des groupes de services, ceci en parallèle de la liste de composants associés aux services de base. Le module de décision ne fait donc pas l'association entre services et composants via des profils réseaux détectés, mais réalise cette association via des profils réseau préétablis. En effet, même si les profils réseau sont définis manuellement, l'algorithme vise à établir un profil réseau et à rechercher le profil auquel il correspond parmi les profils déjà établis. Ensuite, et selon le groupe de services correspondants à l'application en question, l'algorithme sélectionne la composition appropriée.

De cette façon, il est possible de garantir que, pour chaque contexte réseau, la composition déployée sera exactement la composition voulue. Par conséquent, les certifications vont se faire par composition de la même façon qu'elles se faisaient pour les protocoles classiques du réseau ATN.

5.2.2 Evaluation des performances

Dans cette section nous décrivons les différents tests que nous avons effectués pour évaluer les performances de notre architecture. La première partie décrit le principe de notre évaluation et les objectifs des tests, la suite décrit les tests et les résultats obtenus.

Principe d'évaluation

Les objectifs de la campagne de mesures que nous avons effectuée sont au nombre de trois.

Le premier objectif est de démontrer que les performances obtenues sur la base d'une telle architecture (modulaire et incluant une gestion séparée des aspects communication et gestion, ainsi qu'un monitoring du réseau) sont comparables à celles des protocoles " classiques ", notamment TCP, dans des conditions d'instanciation de fonctionnalités équivalentes (contrôle de congestion, reprise des pertes, etc.). En cela, nous montrons que notre architecture, qui permet par conception d'offrir de nouvelles fonctionnalités, comparativement aux protocoles classiques, ne perd pas en performances face à ces protocoles.

Le deuxième objectif est d'évaluer les gains qui peuvent être obtenus en tirant parti des nouvelles fonctionnalités offertes par l'architecture que nous proposons. Nous prenons dans ce cas un exemple appliqué au cas d'étude ciblé, afin de démontrer les faiblesses d'un protocole classique et la façon avec laquelle nous pouvons les dépasser avec la nouvelle architecture. Il s'agit bien entendu des gains en termes de performances uniquement, les gains architecturaux en termes de configurabilité, d'extensibilité ou d'autre, ayant déjà été discutés dans les chapitres 2 et 3.

Le troisième objectif est d'évaluer le coût des composants de gestion permettant d'implanter les fonctionnalités supplémentaires de l'architecture, lorsqu'ils sont confrontés à de grandes quantités de données à traiter. Il s'agit principalement des algorithmes uti-

lisés pour assurer l'autonomie de l'architecture (monitoring et décision). Ces données concernent à la fois les données de monitoring, et les données des applications.

Equivalence de performances

La première campagne d'évaluation vise à démontrer que les traitements effectués par la partie de gestion pour assurer les nouvelles fonctionnalités de la couche Transport, n'influencent pas les performances globales comparées à un protocole classique. Pour cela, nous comparons les performances de notre architecture au protocole TCP en tenant compte des paramètres de comparaison standards définis par l'IETF [88] ; Il s'agit de fairness et friendliness. Le premier concerne l'équité dans le partage de la bande passante entre les connexions d'un même protocole, et le deuxième avec les connexions d'un autre protocole.

Nous évaluons d'abord les performances individuelles de notre architecture par rapport au protocole TCP vis-à-vis de l'occupation de bande passante. Ensuite, nous étudions le comportement de plusieurs connexions établies par notre architecture en termes de partage équitable de la bande passante. Et enfin, nous mesurons le degré d'équité de notre solution en présence d'un protocole classique.

Dans les trois scénarios, l'application utilisant notre architecture est supposée demander un service équivalent à celui offert par TCP. Nous considérons donc un service connecté, en mode flux d'octet, avec garantie d'ordre et de fiabilité, et un mécanisme de contrôle de congestion de type. L'application est un transfert de fichier classique de taille suffisante pour permettre une analyse du comportement de notre solution durant la phase de transfert de données.

Pour les tests, nous utilisons une plateforme de machines virtuelles illustrée ci-après sur la figure 5.4. La plateforme comporte quatre machines faisant office de hôtes reliés deux à deux dans des réseaux locaux virtuels (VLANs) différents. Les deux VLANs sont reliés par une cinquième machine servant de routeur. Ce routeur émule le comportement d'un réseau satellite exprimé en termes de délai aller-retour de 500 ms (correspondant à la moyenne d'un délai d'une communication par satellite géostationnaire). Les machines sont gérées par un système d'exploitation Linux Ubuntu server 14.04. Toutes les connexions utilisées dans les tests se font entre deux machines de VLAN différents, et occupent toute la bande passante offerte par le routeur.

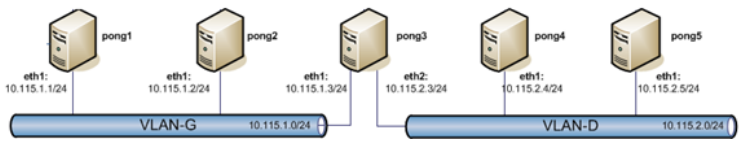

FIGURE 5.4 – Plateforme d'évaluation

Les mesures prises portent sur le débit du transfert de données brutes " sortant " de la couche Transport (incluant les entêtes, les retransmissions, les paquets de contrôle, ...). Le débit est mesuré avec une moyenne arithmétique classique par bloc d'une durée d'une seconde. Les résultats obtenus sont illustrés sur les figure 5.5, 5.6 et 5.7. Dans tous les tracés, les temps (en abscisses) est mesuré en secondes, et le débit (en coordonnées) est mesuré en kilo octets.

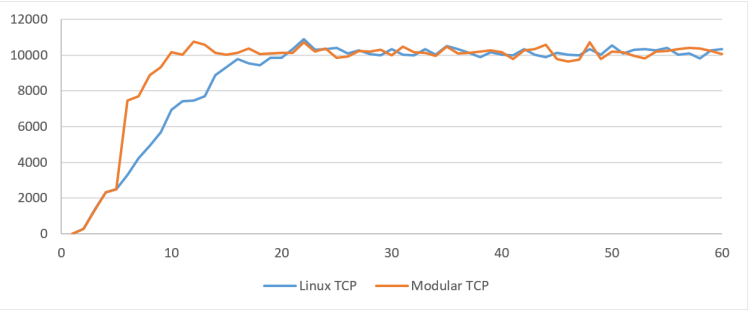

FIGURE 5.5 – Test d'équivalence de performances

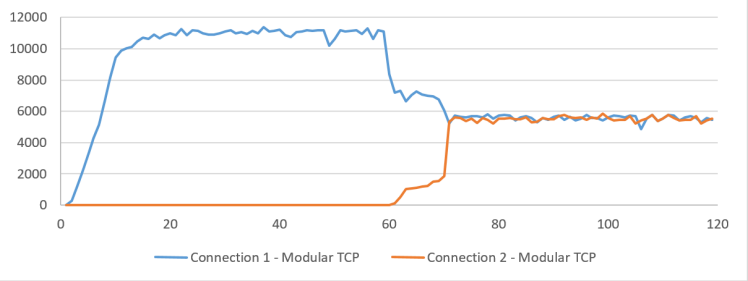

FIGURE 5.6 – Test de fairness

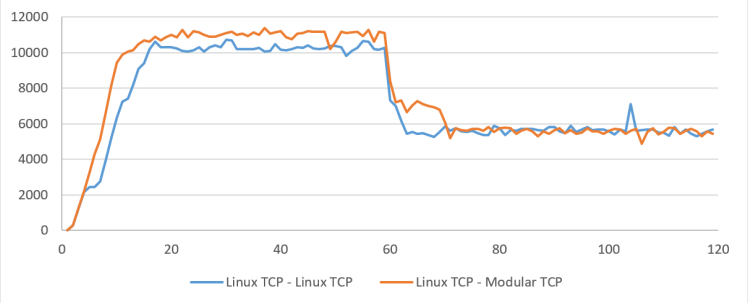

FIGURE 5.7 – Test de friendliness

Dans le premier scénario (figure 5.5) : nous mesurons les performances de la même application émettant au maximum (" greedy source " [89]), une première fois en utilisant le protocole TCP et une deuxième fois en utilisant notre architecture. Comme décrit précédemment, les connexions de Transport sous-jacentes sont établies entre des machines appartenant à deux VLAN différents. Le délai important de 500 ms permet par ailleurs de mieux observer les transitions et les convergences. Les deux connexions sont établies indépendamment, les valeurs correspondantes aux deux courbes sont par conséquent prises sur deux intervalles de temps différents. Les points de départ correspondent toutefois aux débuts des transmissions respectives des deux connexions.

Les résultats obtenus illustrent une équivalence apparente entre le taux d'occupation de la bande passante lors des deux transmissions. Durant 60 secondes, la transmission établie par notre architecture occupe le maximum de bande passante qu'un protocole classique puisse occuper. Les traitements supplémentaires effectués par la partie gestion pour assurer les fonctionnalités supplémentaires de l'architecture n'ont donc aucune influence sur ses performances.

Dans le deuxième scénario (figure 5.6) : nous lançons deux transmissions dans le même intervalle de temps avec un écart de temps entre les transmissions. Les communications sont toutes les deux réalisées par notre architecture. Il s'agit donc ici de mesurer l'équité (" fairness ") dans le partage de la bande passante entre les connexions établies suivant notre architecture.

Au début, une seule transmission est lancée, nous pouvons voir dans le graphique qu'elle occupe la totalité de la bande passante. Après 60 secondes, nous lançons la deuxième transmission. Nous remarquons alors que la première connexion réduit son débit jusqu'à ce qu'elle coïncide avec celui de la deuxième transmission. De son côté, la deuxième transmission augmente graduellement son débit jusqu'à ce qu'il coïncide avec celui de la première transmission. L'équité des transmissions de notre architecture est donc ici montrée. Bien entendu, les mesures ont été prises au niveau de l'émetteur.

Dans le troisième scénario (figure 5.7) : le même test est effectué que dans le second scénario, mais avec cette fois-ci TCP comme protocole pour la deuxième transmission. On peut constater que les valeurs de débit sont pratiquement les mêmes, notre architecture peut donc assurer, avec des transmissions d'autres protocoles, la même équité qu'elle assure entre ses propres transmissions.

Gains de l'adaptabilité

Le deuxième objectif dans l'évaluation de l'architecture est de démontrer l'effet que peut avoir un comportement adaptatif sur les performances par rapport à un comportement classique (statique). Le gain en adaptation est plutôt évident, et ce, sous différents points de vue. Le premier vient du fait que les performances des protocoles ou mécanismes spécifiques dans des contextes précis a déjà été démontré pour toutes les solutions proposées. Ce sont principalement ces gains qui ont motivé notre travail. Il est par conséquent naturel que si l'on met chacune de ces solutions spécifiques dans son contexte approprié, il en résultera des performances optimales. Par ailleurs, l'effet de l'adaptabilité a déjà était démontré dans les travaux précédents sur les architectures de Transport (voir Chapitre I).

L'objectif de cette partie de l'évaluation est donc de montrer, avec un exemple, la capacité de notre architecture à basculer d'une solution spécifique à une autre, en fonction d'un changement de contexte réseau sous-jacent et des variations de performances qui en résultent. Ce changement de comportement est, bien entendu, transparent pour l'application. Ainsi, cette dernière n'a rien à invoquer ou à retransmettre lors de la transition.

Le principe de l'expérience est de lancer une transmission à débit maximal sur une liaison filaire (Ethernet) et de calculer le taux d'occupation de l'interface. Après une période de temps (ici 30 secondes), la transmission est basculée sur une liaison satellite (présentant un délai de transit aller-retour - RTT - beaucoup plus long de l'ordre de 500 ms). Nous observons alors la différence de taux d'occupation de la bande passante entre ces deux phases en utilisant le protocole TCP.

Dans un deuxième temps, nous effectuons la même expérience mais en utilisant notre architecture dans le but de pouvoir adapter le comportement du protocole lors du passage sur la liaison satellite. Nous utilisons ainsi une version différente du contrôle de congestion : un contrôle classique (new reno [90]) pour la liaison Ethernet, et un autre spécifique (hybla [91]) pour l'interface satellite, conçue pour améliorer l'occupation de la bande passante dans les communications satellite. Rappelons ici que cette adaptation (ici un changement de contrôle de congestion) est réalisée de façon autonome par notre architecture, de façon transparente pour l'application.

Les expérimentations ont été réalisées sur la plateforme de la Figure 5.4 précédente, entre deux machines de deux VLAN différents. Le réseau d'accès de la machine émettrice est de type Ethernet. L'émulation de l'interface satellite est réalisée au niveau de la machine 3 servant de routeur. Les résultats sont illustrés sur les figure 5.8 et 5.9.

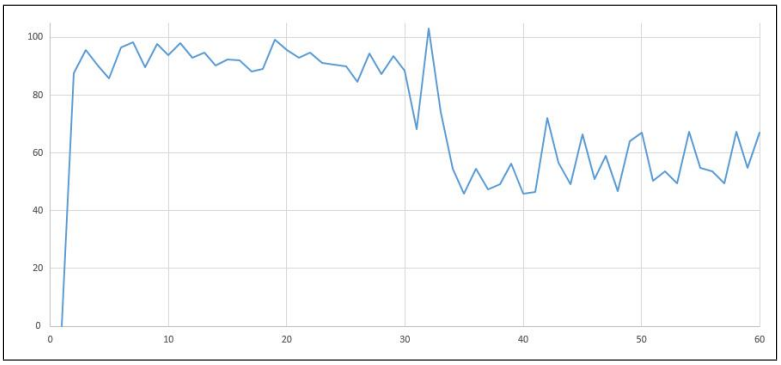

FIGURE 5.8 – Occupation de la bande passante sans adaptabilité

Le premier diagramme illustre la baisse du taux d'occupation de la bande passante quand nous passons sur la liaison satellite satellite. C'est un des exemples les plus connus du " mauvais " comportement du protocole TCP dans des environnements spécifiques [19]. Le taux d'occupation baisse pratiquement de moitié, et ceci dans des conditions pourtant relativement favorables. Dans la réalité, lors de dégradations des conditions météorologiques, les taux de pertes peuvent augmenter considérablement et le taux d'occupation peut diminuer encore en dessous des 20

Dans le deuxième diagramme nous effectuons un changement de comportement lorsque

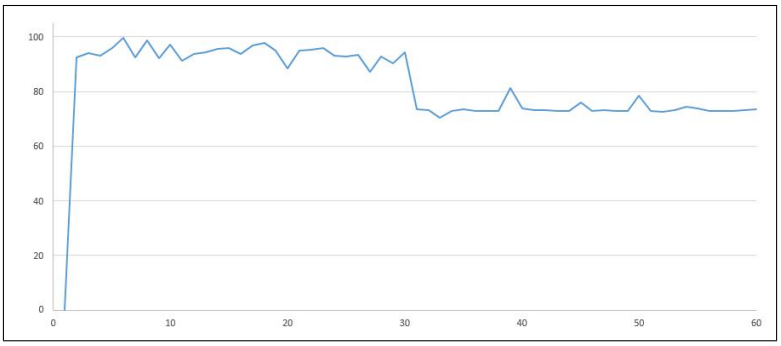

FIGURE 5.9 – Occupation de la bande passante avec adaptabilité

nous basculons vers l'interface satellite. Deux composants correspondants aux deux versions du contrôle de congestion sont présents dans l'architecture. La communication est lancée avec la première version, et la deuxième est instanciée au moment du changement vers l'interface satellite. Le changement à ce niveau est effectué de façon manuelle, et reflète uniquement le gain en adaptabilité, et non pas la capacité d'auto-adaptabilité qui sera étudiée dans la suite.

Nous remarquons clairement la différence entre les deux graphiques et le gain que permet un déploiement adaptatif des versions spécifiques de service de Transport selon les spécificités de l'interface physique. Il est important de noter que la spécificité considérée ne concerne ici que le composant de contrôle de congestion. Des gains plus importants peuvent être obtenus en y associant d'autres mécanismes, liés notamment à la gestion de la fiabilité.

Mesures des composants dédiés à la gestion de l'autonomie

Nous étudions dans cette partie les éléments d'autonomie de notre architecture, liés à la détection de changement de contexte réseau d'une part et la décision (c'est à dire planification d'une nouvelle - et meilleure - solution) d'autre part. Pour le changement de contexte, nous étudions le comportement des différents algorithmes selon les changements qui peuvent se produire dans le réseau. Pour la décision, nous évaluons les performances des deux versions de l'algorithme et son impact à la fois sur le surcoût en termes de calcul, et sur le temps de réponse en cas de changement de contexte.

Détection du changement de contexte Tel que présenté au Chapitre 4, les changements de contexte réseau peuvent survenir de deux façons différentes : soit par un changement brusque, soit par une évolution progressive. Nous étudions ici quatre scénarios possibles qui en résultent (Tableau 5.2.2) : le cas d'un pic momentané, le cas d'un pic permanent, le cas d'une évolution progressive momentanée, et enfin le cas d'une évolution progressive permanente.

Nous utilisons des séries de données représentant les quatre scénarios et nous suivons le comportement de l'algorithme pour chacun des scénarios. Selon le cas, les courbes représentant les résultats des tests peuvent être séparées dans deux ou trois diagrammes en raison de différence de ratios entre les métriques considérées.

Scénario 1 : Cas d'un pic momentané

	Scénario	Figures
Scénario 1	Pic momentanné	5.10, 5.11
Scénario 2	Pic permanent	5.12, 5.13
Scénario 3	Evolution momentanée	5.14, 5.15, 5.16
Scénario 4	Evolution permanente	5.17, 5.18, 5.19

TABLE 5.1 – Analyse comparative selon la spécificité

Les figures 5.10 et 5.10 présente les résultats obtenus pour premier scénario, avec un pic instantané. Les deux schémas correspondent à la même série de données. Le schéma de la Figure 5.10 représente la série de donnée de base ainsi que les deux moyennes arithmétique et glissante. La Figure 5.11 représente l'évolution des écarts et de leur moyenne ainsi que le seuil de détection de pics.

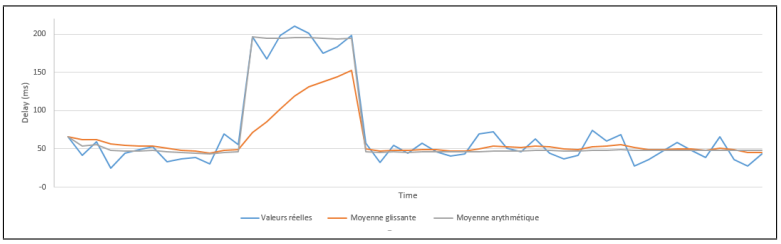

FIGURE 5.10 – Pic momentané - Représentation des moyennes

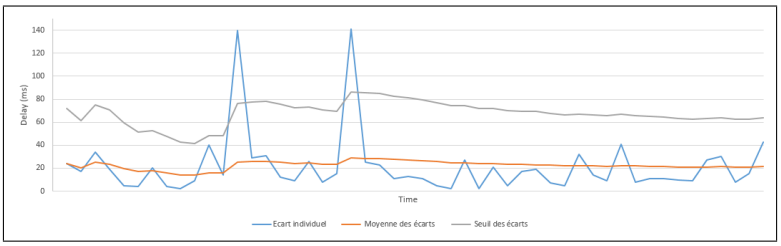

FIGURE 5.11 – Pic momentané - Représentation des écarts

Lorsque le pic de données se produit sur la figure 5.10, nous remarquons que l'écart correspondant dépasse le seuil de pics. Le pic est donc détecté et l'algorithme entre dans une phase d'attente avant de confirmer le changement de contexte. La moyenne arithmétique bascule au départ de la valeur de pics et l'algorithme attend qu'elle converge avec la moyenne glissante. Mais lorsque le deuxième pic, dans le sens inverse se produit, les deux moyennes ne se rejoignent pas encore, l'algorithme conclut donc que le pic est momentané et reprend depuis la valeur du premier pic avec les deux moyennes. Nous pouvons voir sur la figure 5.11 la détection du deuxième pic. Le taux de croissance de la moyenne glissante, correspondant au paramètre ? qui est propre à chaque valeur monitorée, définit, indirectement, le temps d'attente nécessaire avant de conclure par un changement de contexte. Plus le paramètre est petit, plus le temps d'attente est grand et vice versa.

Scénario 2 : Cas d'un pic permanent

Les courbes présentées sur les figures 5.12 et 5.13 ci-dessous illustrent le cas d'un pic permanent, c'est-à-dire correspondant à un vrai changement de contexte qui se produit suite à un pic de données (il s'agit là de données de monitoring et non de transmission, portant sur le délai ou le débit par exemple).

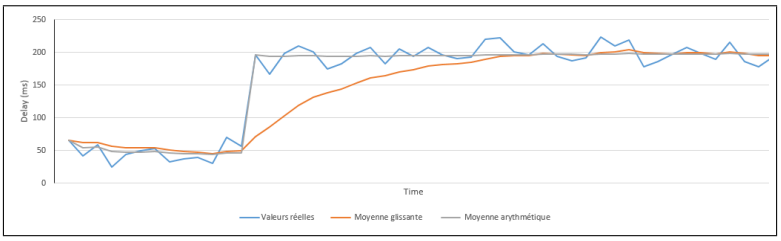

FIGURE 5.12 – Pic permanent - Représentation des moyennes

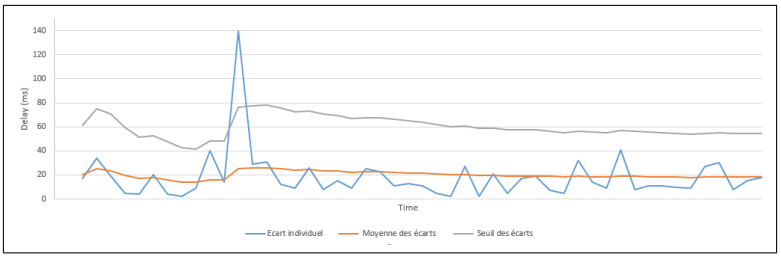

FIGURE 5.13 – Pic permanent - Représentation des écarts

Dans ce deuxième scénario, la première partie est équivalente à celle du premier scénario, la différence vient du fait qu'une fois que le pic s'est produit, la communication continue avec la même valeur ; il n'y a donc pas de deuxième pic dans le sens inverse. Nous le remarquons d'ailleurs sur la figure 5.13 où seul le premier pic apparait. Par conséquent, sur la figure 5.13, nous pouvons voir que la courbe de la moyenne glissante finit par rejoindre celle de la moyenne arithmétique qui, de la même façon que pour le premier scénario, redémarre depuis la valeur du pic lorsque celui-ci se produit. Une fois que les deux moyennes se sont rejointes, l'algorithme conclut par un changement de contexte et réinitialise ses variables par les valeurs relevées au moment du pic.

Scénarios 3 et 4 : Cas d'une évolution progressive permanente + Cas d'une évolution progressive momentanée

Les deux autres scénarios concernent le deuxième mode de changement de contexte par évolution progressive. Là encore, il se peut que l'évolution soit permanente ou momentanée. Le premier scénario illustre le cas d'une évolution progressive momentannée. Le premier schéma présente la série de données de base (Figure 5.14), le deuxième donne la succession des écarts croissants et décroissants (Figure 5.15), et le dernier fournit le rapport entre les deux (Figure 5.16). Le deuxième scénario représente, avec des schémas similaires, le cas d'une évolution progressive permanente (Figures 5.17, 5.18, et **??**) Pour rappel (cf. Chapitre 4, section 4.3), l'algorithme utilise le rapport entre les montées et les descentes des valeurs recueillies. Le principe est que pour qu'une courbe soit croissante en moyenne, il faut qu'elle monte plus qu'elle ne descend.

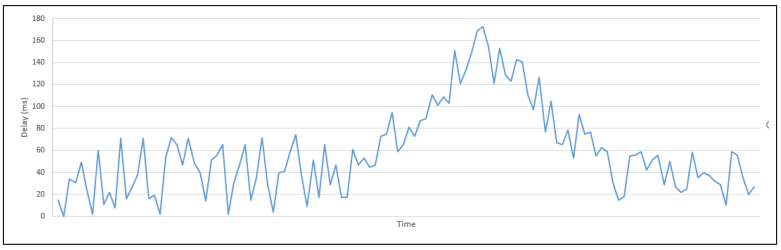

FIGURE 5.14 – Evolution progressive momentannée - Représentation des valeurs

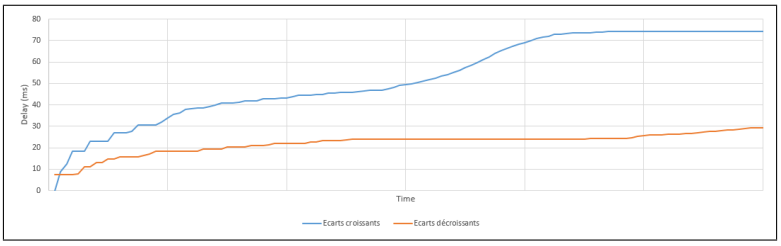

FIGURE 5.15 – Evolution progressive momentannée - Représentation des écarts

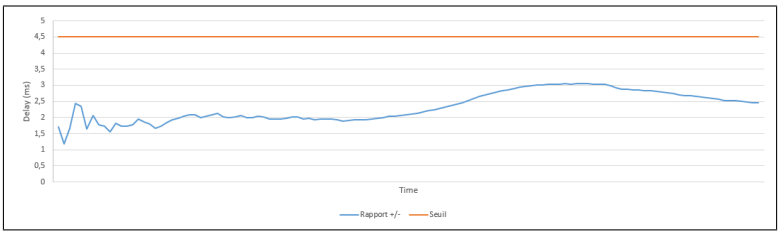

FIGURE 5.16 – Evolution progressive momentannée - Représentation de la moyenne des écarts

Le premier constat est le lien évident qui existe entre la croissance de la série de donnée (Figure 5.13) et le rapport entre les écarts croissants et décroissants (Figure 5.17). Nous remarquons que lorsque la courbe commence à monter, la moyenne des écarts positifs commence de son côté à dominer celle des écarts négatifs. Lorsque la moyenne atteint le seuil, l'algorithme conclut par un changement de contexte. Dans le cas contraire, comme illustré dans le scénario 2, la courbe des rapports commence à régresser avant d'atteindre le seuil, et l'algorithme conclut par une évolution momentanée.

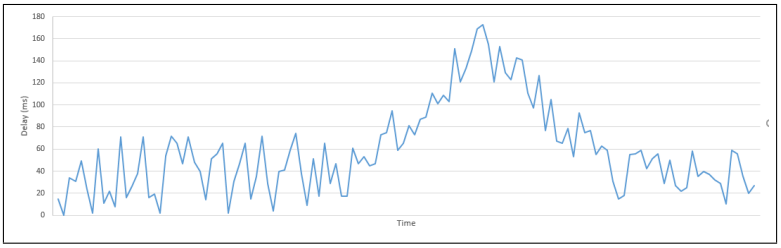

FIGURE 5.17 – Evolution progressive permanente- Représentation des valeurs

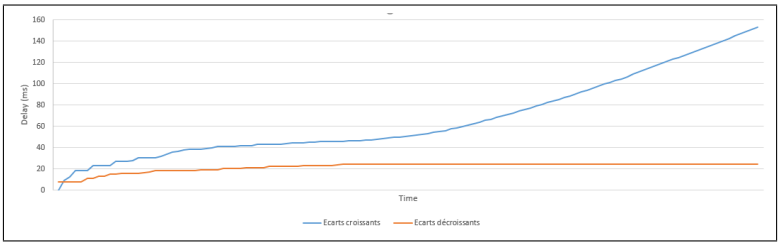

FIGURE 5.18 – Evolution progressive permanente- Représentation des écarts

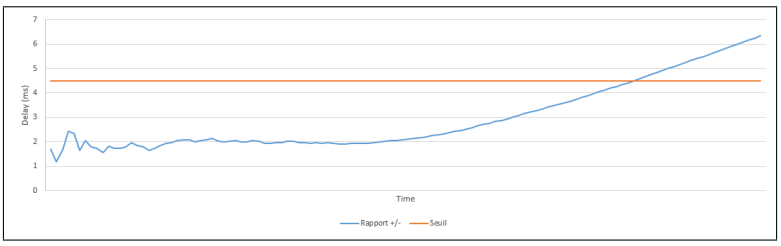

FIGURE 5.19 – Evolution progressive permanente- Représentation de la moyenne des écarts

Décision

Le dernier point concerne l'analyse de performances de l'algorithme de décision. La première version, qui effectue un calcul systématique de la fonction d'utilité pour tous les composants jusqu'à trouver le composant approprié, est facile à calculer. Dans le pire des cas, c'est à dire celui où le composant adapté se trouverait enfin de liste, l'algorithme aura à parcourir l'ensemble des composants. Par conséquent, l'algorithme doit effectuer, pour chaque service et pour chaque composant, un calcul de la fonction d'utilité qui dépend du nombre de critères de monitoring. La complexité devient donc : O(SrCmCr), avec Sr le nombre de services requis par l'application, Cm le nombre de composants disponibles dans l'architecture, Cr le nombre de critères de monitoring pour la communication en cours.

Dans la deuxième version, dans laquelle l'algorithme fait une réduction du nombre de composants au fur et à mesure qu'il avance dans l'examen des critères du profil, la complexité est plus difficile à calculer. En effet, le nombre de composants à éliminer après chaque critère est aléatoire. Le nombre de services restant inchangés, la complexité devient donc O(Sr O(CmCr)). Dans tous les cas, la complexité reste polynomiale. Si l'on y ajoute le nombre relativement limité de composants et services, le temps nécessaire pour établir une liste de composants à partir d'un profil réseau reste très petit.

5.3 Conclusion

Ce chapitre a été dédié au déploiement et à l'évaluation de l'architecture de la couche Transport que nous avons proposée. Nous avons tout d'abord décrit les détails de l'implémentation de la gestion des interactions entre les différents composants de l'architecture. Nous avons ensuite présenté le cas d'étude de notre architecture et les spécificités du domaine de déploiement considéré. Enfin nous avons exposé l'évaluation et la validation de l'architecture. L'évaluation a concerné l'équivalence de performances par rapport aux protocoles classiques, le gain de la nouvelle architecture, et les aspects autonomiques de l'architecture relevant de la détection du changement de contexte réseau d'une part, et de la planification (décision) des actions de reconfiguration d'autre part.

Bien que l'architecture n'ait pas été déployée dans le noyau, à ce stade, nous démontré sa faisabilité en présentant le principe de son déploiement. Par ailleurs, nous avons vu comment notre architecture permet de répondre aux deux problématiques du contexte aéronautique, à savoir la gestion de la transition vers les protocoles de l'Internet, et la gestion de l'hétérogénéités des liens de communication. Enfin, les tests démontrent les performances de notre proposition à la fois du fait que la partie de gestion n'affecte pas ses performances, mais aussi aux gains qu'il est possible d'obtenir en adoptant un comportement adaptatif. La dernière partie a démontré l'efficacité et la performance des algorithmes utilisés pour assurer cette adaptabilité.

Conclusion et Perspectives

Cette thèse a présenté l'architecture, les modèles et les outils d'une couche de Transport extensible et autonome, et son application dans le domaine des communications aéronautiques. Nous sommes partis du problème de l'évolution de la couche Transport actuelle qui se caractérise par l'usage de protocoles parfois inadaptés à la fois pour les nouvelles applications et les nouvelles technologie réseaux, et ce bien que de nombreuses autres solutions plus efficaces aient été proposées. Le besoin principal était donc de permettre le déploiement de nouveaux protocoles et mécanismes dans la couche Transport.

Le travail a été réalisé en trois étapes. La première a été consacrée à l'étude des obstacles face à l'intégration des nouvelles solutions de Transport, pour déduire les besoins architecturaux permettant de les contourner. Durant la deuxième étape, une architecture a été proposée comme solution aux limites de la couche Transport actuelle. Dans le cadre proposé, de nouveaux paradigmes de conception, de nouveaux modèles théoriques et de nouveaux algorithmes ont été mis en oeuvre pour répondre aux différents besoins identifiés. Enfin, la dernière étape a concerné l'implémentation, le test, et le déploiement de notre architecture dans le cadre d'un cas d'étude portant sur les communications aéronautiques.

Après avoir présenté un état de l'art sur les aspects relevant de la problématique ciblée, notamment en rapport avec l'évolution des protocoles et des architectures de Transport (Chapitre 1), nous avons présenté au Chapitre 2 l'architecture que nous avons proposée pour la couche Transport. Cette architecture est guidée par l'ensemble des besoins établis en amont. Nous y avons décrit l'ensemble des composants de l'architecture, ainsi que les principaux comportements.

Les Chapitres 3 et 4 ont été consacrés à la description des deux aspects principaux de notre architecture, à savoir : l'extensibilité et l'auto-adaptabilité. L'importance de ces deux propriétés vient du fait que ce sont elles qui permettent l'évolution de l'architecture par l'intégration et la prise en compte, de façon adaptative, des nouvelles solutions de Transport. L'extensibilité a été réalisée principalement via l'application de paradigmes de conception tels que l'orienté services et l'inconscience sémantique, ceci à différent niveaux de l'architecture. L'autonomie a été réalisée via différents modèles et algorithmes suivant le paradigme de l'autonomic computing, notamment sur les phases de monitoring et de planification.

Le dernier chapitre (Chapitre 5) a traité des aspects implémentation, test et déploiement de l'architecture dans le cadre d'un cas d'étude en rapport avec les communications aéronautiques par satellite. Nous avons tout d'abord présenté les principaux choix d'implémentation et expliqué la faisabilité de l'approche proposée pour l'archi-

tecture. Nous avons également montré en quoi cette architecture répond à la problématique spécifiée dans le cas d'étude ciblé. Enfin, nous avons présenté les tests effectués et les résultats obtenus.

Grâce à l'architecture proposée, nous avons pu répondre à l'ensemble des besoins associés aux limites de la couche Transport actuelle. En effet, notre proposition permet d'intégrer de nouveaux services et mécanismes de Transport, sans difficulté de résistance au facteur d'échelle vis à vis du nombre de solutions à intégrer, et sans impact sur le système d'exploitation hôte ni sur les applications utilisant l'architecture. Les nouveaux services et composants peuvent être pris en compte dès leur intégration, y compris durant les communications et sans interruption de celles-ci. L'indépendance vis-à-vis des systèmes d'exploitation a été réalisée par le principe de séparation des parties gestion et communication de l'architecture, pour n'inclure que la première partie (car non ou très faiblement évolutive) dans le code du noyau. Pour les applications, l'indépendance a été réalisé par le biais d'une nouvelle interface permettant aux développeurs d'applications ou aux applications elles-mêmes d'exprimer des requêtes de services.

En plus de ses possibilités d'extensibilité, l'architecture permet d'aboutir à un service optimal, c'est à dire adapté spécifiquement à chaque application et à chaque contexte réseau. Cette optimalité est assurée par des composants de communication spécifiques, et par un module de gestion de l'autonomie permettant la détection du contexte réseau et son association aux composants appropriés en tenant compte des besoins de l'application. L'adaptabilité de l'architecture est non seulement complètement autonome (sans besoin d'intervention externe), mais aussi transparente pour les systèmes, les applications, et les autres composants de l'architecture.

Sur le plan pratique, l'architecture a été implémentée et testée dans le cadre du cas d'étude considéré. L'implémentation a dû faire face à plusieurs défis en rapport avec les nouvelles fonctionnalités de l'architecture. Elle a toutefois permis de démontrer la faisabilité d'une telle approche avec une implémentation fonctionnelle. L'architecture étant développée uniquement dans l'espace utilisateur dans le cadre de ce travail, nous avons cependant illustré le principe et la faisabilité de son déploiement dans le noyau du système d'exploitation.

Concernant le cas d'étude considéré, l'architecture permet de répondre aux deux problématiques posées, à savoir : la gestion de la phase de transition de la pile ATN vers les standards de l'Internet, et la gestion de l'hétérogénéité des liens physiques. Pour cette dernière, le problème est équivalent au problème d'adaptabilité générique traité dans l'architecture. Pour la phase de transition, un déploiement spécifique est nécessaire. Cependant, l'architecture étant déjà adaptative, il suffit d'ajouter le paramètre concernant les différentes piles protocolaires dans le processus de décision. Par ailleurs, le passage par le composant d'intégration pour chaque ajout ou modification offre la possibilité d'imposer une certification sur les composants à intégrer, et ce sans intervention externe.

Enfin, les tests ont porté sur plusieurs aspects de l'architecture. D'abord, nous avons démontré que les éléments de gestion associés aux nouvelles fonctionnalités et capacités de l'architecture n'affectent pas les performances des communications. Nous avons également montré les gains en performances qu'il est possible d'obtenir par l'adapta-

tion des composants de communication au contexte réseau et applicatifs. Enfin, nous avons testé l'efficacité et le coût des algorithmes de la partie gestion de l'architecture.

Perspectives

L'objectif qui a été fixé dans le cadre de ce travail était de proposer et d'implémenter une architecture répondant à la problématique de l'évolution de la couche Transport avec application à un cas d'étude spécifique. Nous avons couvert un ensemble important de problématiques spécifiques à chaque partie de l'architecture. Cependant, chacune de ces problématiques, comme pour la décision ou l'intégration dans le noyau, nécessiterait une étude dédiée. Dans notre travail et compte tenu de la durée limitée de la thèse, nous avons traité des aspects qui nous ont semblé essentiels, en allant jusqu'à un niveau d'approfondissement suffisant pour établir une architecture fonctionnelle.

Par conséquent, plusieurs parties de l'architecture, bien qu'elles soient fonctionnelles, nécessitent une étude plus approfondie. Ces études peuvent concerner des positionnements plus précis, des modèles plus poussés, une implémentation plus complète, ou encore des tests plus rigoureux.

Principalement, il s'agit de positionner davantage les algorithmes de détection de changement de contexte par des tests permettant de les comparer aux techniques déjà existantes. Les bases théoriques du modèle décisionnel peuvent, elles-aussi, être davantage comparées aux autres approches de décision. Sur le plan de la réalisation, un objectif ambitieux serait d'intégrer notre architecture dans le noyau d'un système d'exploitation. Ceci permettrait d'une part de démontrer la possibilité d'une telle intégration, mais aussi d'avoir des résultats de tests plus réalistes et comparables avec les meilleures implémentations des protocoles actuels. Par ailleurs, d'autres paramètres de tests pourraient être pris en compte, via notamment le développement et l'intégration d'autres exemples de composants protocolaires.

Concernant le cas d'étude considéré, l'architecture pourrait être déployée concrètement afin d'obtenir davantage de spécification de déploiement, et d'en mesurer les gains de façon plus concrète. En effet, en l'absence d'une plateforme réelle, l'application au cas d'étude a été limitée à des tests de performances sur des plateformes d'émulation du contexte aéronautique. Une application plus concrète serait bénéfique, notamment via une étude plus spécifique.

Outre le développement de cette architecture pour une couche de Transport, les principes d'autonomie et d'extensibilité peuvent servir de modèle pour toute architecture évolutive. Aussi, avec le composant d'interface réseau, il est possible d'utiliser l'architecture comme une puissante plateforme de tests sur les protocoles de Transport. Il est ainsi possible d'adopter plusieurs configurations réseau possibles au niveau de l'interface, et d'utiliser les composants de monitoring pour relever des mesures de performances. Son aspect extensible permet d'intégrer facilement des composants, et de les tester avec un minimum de travail d'adaptation.

Enfin, et dans un contexte d'Internet élargi aux objets connectés et aux applications attenantes, dans lequel se développent actuellement de multiple solution dites de virtualisation, l'étude des opportunités liées au déploiement dynamique de composants

et/ou protocoles de Transport par le biais des nouvelles technologies liées aux concepts de virtualisation de fonctions réseau et des réseaux pilotables par le logiciel est une perspective majeure que nous donnons à nos propositions.

Bibliographie

[1] S. Seshan E. Amir, H. Balakrishnan and R. H. Katz. "Efficient TCP over networks with wireless links". *Proceedings 5th Workshop on Hot Topics in Operating Systems (HotOS-V)*, 1995.

[2] Z. Li L. Xiao and N. Zhao. "A TCP Performance Improved mechanism for Wireless Network". *Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*, 2009.

[3] Peng Zhang Shiduan Cheng Jing Wu, Yu Shi and Jian Ma. "ACK delay control for improving TCP throughput over satellite links". *IEEE International Conference on networks*, 1999.

[4] Seung-Yong Lee Jong-Mu Kim and Jae-Hyun Kim. "TCP congestion window tuning for satellite communication using cross-layer approach". *Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2016.

[5] S. Paul S. Gopal and D. Raychaudhuri. "Investigation of the TCP simultaneous-send problem in 802.11 wireless local area networks". *IEEE International Conference on Communications*, 2005.

[6] M. Mzyece D. A. Rambim and K. Djouani. "Enhancement of TCP in 802.11e Wireless Local Area Networks". *2011 IEEE Vehicular Technology Conference (VTC Fall)*, 2011.

[7] C. Chassot M. Oulmahdi and E. Exposito. "Reducing Energy Cost of Keepalive Messages in 3G Mobiles". *27th International Conference on Advanced Information Networking and Applications Workshops*, 2013.

[8] C. Chassot M. Oulmahdi and E. Exposito. "An energy-aware TCP for multimedia streaming". *International Conference on Smart Communications in Network Technologies*, 2013.

[9] D. Sisalem and A. Wolisz. "Towards TCP-friendly adaptive multimedia applications based on RTP". *Proceedings IEEE International Symposium on Computers and Communications*, 1999.

[10] A. G. Berumen and M. Marot. "BoD algorithm for TCP and multimedia applications in a DVB-S2/RCS system". *International Workshop on Satellite and Space Communications*, 2009.

[11] J. I. Khan and R. Y. Zaghal. "Jitter and delay reduction for time sensitive elastic traffic for TCP-interactive based world wide video streaming over ABone". *International Conference on Computer Communications and Networks*, 2003.

[12] H. Ujikawa K. Ogura J. Katto T. Fujikawa, Y. Takishima and H. Izumikawa. "A hybrid TCP-friendly rate control for multimedia streaming". *International Packet Video Workshop*, 2010.

[13] R. Stewart. "Stream Control Transmission Protocol". *IETF RFC 4960*, 2007.

[14] S. Floyd E. Kohler, M. Handley. "Datagram Congestion Control Protocol". *IETF RFC 4340*, 2007.

[15] M. Handley O. Bonaventure A. Ford, C. Raiciu. "TCP Extensions for Multipath Operation with Multiple Addresses". *IETF RFC 6824*, 2013.

[16] S. T. Chanson M. S. Atkins and J. B. Robinson. "LNTP-an efficient transport protocol for local area networks". *IEEE Global Telecommunications Conference and Exhibition. Communications for the Information Age, vol. 2*, 1988.

[17] I. Groenbaek. "Conversion Between the TCP and ISO Transport Protocols as a Method of Achieving Interoperability Between Data Communications Systems". *IEEE Journal on Selected Areas in Communications, vol. 4, no. 2*, 1986.

[18] M. Kaiserswerth B. W. Meister H. Rudin W. A. Doeringer, D. Dykeman and R. Williamson. "A survey of light-weight transport protocols for high-speed networks". *A survey of light-weight transport protocols for high-speed networks*, 1990.

[19] Aas G. Rydningen Henriksen, E. "A transport protocol supporting multicast file transfer over satellite links". *Computers and Communications*, 1992.

[20] Pin-Han Ho Shihada, B. "Transport control protocol in optical burst switched networks : issues, solutions, and challenges". *Communications Surveys and Tutorials, IEEE , vol.10, no.2*, 2008.

[21] A. Romanow. "TCP over ATM with Congestion : Some Performance Results". *Proceedings of the 6th IEEE Workshop on Local and Metropolitan Area Networks*, 1993.

[22] S. Bajeja and A. Gosai. "Performance evaluation of traditional TCP variants in wireless multihop networks". *International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016.

[23] Y. M. Yussoff R. A. Rahman M. Masirap, M. H. Amaran and H. Hashim. "Evaluation of reliable UDP-based transport protocols for Internet of Things". *IEEE Symposium on Computer Applications and Industrial Electronics (ISCAIE)*, 2016.

[24] Yifei Lu. "SED : An SDN-based explicit-deadline-aware TCP for cloud Data Center Networks". *Tsinghua Science and Technology, vol. 21, no. 5*, 2016.

[25] J. Zdepski Raychaudhuri, R. Siracusa and K. Joseph. "A flexible and robust packet transport protocol for digital HDTV". *Global Telecommunications Conference, GLOBECOM*, 1992.

[26] T. Zhang J. Chen Huang, J. Wang and Y. Pan. "Tuning the Aggressive TCP Behavior for Highly Concurrent HTTP Connections in Data Center". *IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, 2016.

[27] B. Cheng Y. Yang M. Wang J. Wu, C. Yuen and J. Chen. "Bandwidth-Efficient Multipath Transport Protocol for Quality-Guaranteed Real-Time Video Over Heterogeneous Wireless Networks". *IEEE Transactions on Communications, vol. 64, no. 6*, 2016.

[28] Dabran and D. Raz. "TCP cooperation for multimedia over wireless networks". *International Conference on Broadband Communications, Networks and Systems*, 2008.

[29] Yi Li Gang Liu, Hong Ji and Xi Li. "TCP performance enhancement for mobile broadband interactive satellite communication system : A cross-layer approach". *International Conference on Communications and Networking in China (CHINACOM)*, 2013.

[30] S. T. Chanson M. S. Atkins and J. B. Robinson. "LNTP-an efficient transport protocol for local area networks". *IEEE Global Telecommunications Conference and Exhibition. Communications for the Information Age*, 1988.

[31] J. H. Huang and S. H. Lee. "MHTP-a multimedia high-speed transport protocol". *Global Telecommunications Conference, 1992. Conference Record., GLOBECOM*, 1992.

[32] Y. Park O. C. Kwon, Y. Go and H. Song. "MPMTP : Multipath Multimedia Transport Protocol using Systematic Raptor Codes over Wireless Networks". *IEEE Transactions on Mobile Computing, vol. 14, no. 9*, 2015.

[33] M. Janbeglou and N. Brownlee. "OverUDP : Tunneling Transport Layer Protocols in UDP for P2P Application of IPv4". *International Conference on Advanced Information Networking and Applications Workshops*, 2015.

[34] Nicolas Van Wambeke Mohamed Oulmahdi, Christophe Chassot. "Transport protocols : limitations, evolution obstacles and solutions for an actual deployment in the Internet". *International Journal of Parallel, Emergent and Distributed Systems*, 2015.

[35] D. F. Box D. C. Schmidt and T. Suda. "ADAPTIVE, A Dynamically Assembled Protocol Transformation, Integration and eValuation Environment". *Journal of Concurrency : Practice and Experience*, 1993.

[36] Donald F. Box Douglas C. Schmidt and Tatsuya Suda. "ADAPTIVE : A flexible and adaptive transport system architecture to support lightweight protocols for multimedia applications on high-speed networks". *Proceedings of the 1st Symposium on High-Performance Distributed Computing*, 1992.

[37] M. Hiltunen R. D. Schlichting P. G. Bridges, G. T. Wong and M. J. Barrick. "A Configurable and Extensible Transport Protocol". *IEEE/ACM Transactions on Networking, vol. 15, no. 6*, 2007.

[38] Richard D. Schlichting Nina T. Bhatti, Matti A. Hiltunen and Wanda Chiu. "Coyote : a system for constructing fine-grain configurable communication services". *ACM Transactions on Computer Systems*, 1998.

[39] Christophe Gregoire Sergio Mena, Xavier Cuvellier and Andre Schiper. "Appia vs. cactus : Comparing protocol composition frameworks". *Proceeidngs of the 22nd Symposium on Reliable Distributed Systems*, 2003.

[40] A. Pinto H. Miranda and L. Rodrigues. "Appia : A flexible protocol kernel supporting multiple coordinated channels". *21st International conference on Distributed Computing Systems*, 2001.

[41] C. Chassot N. Van Wambeke, E. Expósito and M. Diaz. "ATP : A Microprotocol Approach to Autonomic Communication". *IEEE Transactions on Computers, vol. 62, no. 11*, 2013.

[42] M. Kuehlewind G. Fairhurst, B. Trammell. "Services Provided by IETF Transport Protocols and Congestion Control Mechanisms". *IETF RFC*, 2017.

[43] N. C. Hutchinson and L. L. Peterson. "The x-kernel : An architecture for implementing network protocols". *IEEE Transactions on Software Engineering*, 1991.

[44] Dennis M. Ritchie. "A stream input-output system". *ATandT Bell Laboratories Technical Journal*, 1984.

[45] Ernesto J. Exposito G. "Design and implementation of a QoS oriented transport protocol for multimedia applications". *PhD thesis, Institut National Polytechnique de Toulouse*, 2003.

[46] G. Dugué. "Modélisation d'une architecture orientée service et basée comosant pour une couche de Transport autonome, dynamique et hautement configurable". *Thèse doctorat*, 2014.

[47] IBM. "An architectural blueprint for autonomic computing". *White Paper*, 2006.

[48] G. Papastergiou et al. "De-Ossifying the Internet Transport Layer : A Survey and Future Perspectives". *IEEE Communications Surveys and Tutorials, vol. 19, no. 1*, 2017.

[49] H. Chen J. Yang Y. Zhang M. Parashar H. Liu S. Hariri, B. Khargharia. "The Autonomic Computing Paradigm". *Cluster Computing, no. 9*, 2009.

[50] S. Poslad. "Autonomous Systems and Artificial Life, in Ubiquitous Computing : Smart Devices, Environments and Interactions". *John Wiley and Sons*, 2009.

[51] M. R. Nami and K. Bertels. "A Survey of Autonomic Computing Systems". *International Conference on Autonomic and Autonomous Systems*, 2007.

[52] International Civil Aviation Organization (ICAO). "Manual for the ATN Using IPS Standards And Protocols". *Document 9896*, Ed1.

[53] International Civil Aviation Organization (ICAO). "Global Operational Data Link Document". Ed3.

[54] International Civil Aviation Organization (ICAO). " Manual of Technical Provisions for the Aeronautical Telecommunication Network (ATN)". *Document 9705*, Ed3.

[55] International Civil Aviation Organization (ICAO). "Manual On Detailed Technical Specifications For The Aeronautical Telecommunication Network (Atn) Using ISO/OSI Standards And Protocols - Part I - Air-Ground Applications". *Document 9896*, Ed1.

[56] International Civil Aviation Organization (ICAO). "Manual On Detailed Technical Specifications For The Aeronautical Telecommunication Network (Atn) Using ISO/OSI Standards And Protocols, Part IIA : Ground-Ground Application - ATS Message Handling Service (ATSMHS)". *Document 9896*, Ed1.

[57] International Civil Aviation Organization (ICAO). "Manual On Detailed Technical Specifications For The Aeronautical Telecommunication Network (Atn) Using ISO/OSI Standards And Protocols, Part IIB : Ground-Ground Application - ATS Interfacility Data Communications (AIDC)". *Document 9896*, Ed1.

[58] Thomas Erl. "SOA : Principles of Services Design". *SOA Systems*, 2008.

[59] Martin Fowler. "Patterns of Entreprise Application Architecture". *Pearson Education*, 2008.

[60] Thomas Erl. "Service Oriented Architecture : Concepts, Technology, and Design". *Pearson Education*, 2005.

[61] Robert Daigneau. "Service Design Patterns". *Pearson Education*, 2012.

[62] Murali K. Ganapathy Lynn B. Reid Katherine Riley Dan Sheeler Andrew Siegel Klaus Weide Anshu Dubey, Katie Antypas. "Extensible component-based architecture for FLASH, a massively parallel, multiphysics simulation code". *Journal of Parallel Computing, Volume 35, Issues 10-11*, 2009.

[63] Kevin Kelly Alan Brown, Simon Johnston. "Using Service-Oriented Architecture and Component Based Development to Build Web Service Applications". *A Rational White Paper*, 2002.

[64] Chen M.Y. Chen Y.M. Chu, H.C. and C.J. Lin. "Developing a Semantic-Awareness Architecture for Content Management in e-Learning". *Proceedings of E-Learn : World Conference on E-Learning in Corporate*, 2006.

[65] Y. Yamada L. Batterink, C. M. Karns and H. Neville. "The Role of Awareness in Semantic and Syntactic Processing : An ERP Attentional Blink Study". *Journal of Cognitive Neuroscience, vol. 22, no. 11*, 2010.

[66] Yuh-Ming Chen Ming-Yen Chen, Ming-Fen Yang and Hui-Chuan Chu. "Development of a Semantic Awareness Framework for Textual Content Management in e-Learning". *Sixth IEEE International Conference on Advanced Learning Technologies*, 2006.

[67] J. E. Corcoles J. Marquez and A. Quintanilla. "Scoring results in a geospatial services search engine according to geographic and semantic awareness". *International Conference on Next Generation Web Services Practices*, 2011.

[68] Nick Bassiliades Di1nosthenis Anagnostopoulos Ioannis Vlahavas Ourania Hatzi, Di1nitris Vrakas. "Semantic Awareness in Automated Web Service Composition through Planning". *Proceeding of 6th Hellenic Conference on Artificial Intelegence*, 2010.

[69] H. Sahbi. "Interactive Satellite Image Change Detection With Context-Aware Canonical Correlation Analysis". *IEEE Geoscience and Remote Sensing Letters, vol. 14, no. 5*, 2017.

[70] T. Tanprasert and T. Kripruksawan. "An approach to control aging rate of neural networks under adaptation to gradually changing context". *International conference onNeural Information Processing*, 2002.

[71] S. Derrode C. Carincotte and S. Bourennane. "Unsupervised change detection on SAR images using fuzzy hidden Markov chains". *IEEE Transactions on Geoscience and Remote Sensing, vol. 44, no. 2*, 2006.

[72] A. Heil and M. Gaedke. "Environment-Awareness : Quantitative Processing of Context Changes". *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2008.

[73] S. A. S. Kumar S. Selvaprabhu and A. S. Chandar. "Human health control monitor system using smart mobiles : Context changes dependent human behavior". *International Conference on Science Technology Engineering and Management*, 2016.

[74] H. Sahbi. "Interactive Satellite Image Change Detection With Context-Aware Canonical Correlation Analysis". *IEEE Geoscience and Remote Sensing Letters, vol. 14, no. 5*, 2017.

[75] C. Huo X. Sun K. Chen, Z. Zhou and K. Fu. "A Semisupervised Context-Sensitive Change Detection Technique via Gaussian Process". *IEEE Geoscience and Remote Sensing Letters, vol. 10, no. 2*, 2013.

[76] Ghada Gharbib Mohamed Jmaiela Nesrine Khaboua, Ismael Bouassida Rodriguezb. "A Threshold Based Context Change Detection in Pervasive Environments : Application to a Smart Campus". *5th International Conference on Ambient Systems, Networks and Technologies*, 2014.

[77] A. Banerjee and S. K. S. Gupta. "Your mobility can be injurious to your health : Analyzing pervasive health monitoring systems under dynamic context changes". *IEEE International Conference on Pervasive Computing and Communications*, 2012.

[78] A. Shah K. Nusratullah, S. A. Khan and W. H. Butt. "Detecting changes in context using time series analysis of social network". *SAI Intelligent Systems Conference*, 2015.

[79] L. Salgado N. García M. Nieto, C. Cuevas. "Line segment detection using weighted mean shift procedures on a 2D slice sampling strategy". *Journal of Patterns Analysis and Applications*, 2011.

[80] M. J. Carty. "Image Analysis, Classification, and Change Detection in Remote Sensing". *CRC Press*, 2014.

[81] J.S Simonoff. "Smoothing Methods in Statistics". *Springer*, 1998.

[82] J. Martel A. Guitouni, M. Bélanger. "Cadre Méthodologique pour Différencier les Méthode Multicritères". *DRDC Valcartier*, 2010.

[83] J.C. Pomerol et S. Barba-Romero. "Choix multicritère dans l'entreprise, principe et pratique". *Hemès*, 1993.

[84] L. Henrier. "Systèmes de d'Evaluation et de Classification Multicritères pour l'aide à la Décision". *Thèse Doctorat*, 2000.

[85] R. Rosen. "Linux Kernel Networking, Implementation and Theory". *Apress*, 2014.

[86] M. Venkatesulu S. Seth. "TCP/IP Architecture, Design and Implementation in Linux". *IEEE*, 2008.

[87] T. Hebbert. "The Linux TCP/IP Stack : Networking for Embedded Systems". *Charle Hiver Media*, 2006.

[88] J. Padhye S. Floyd, E. Kohler. "Profile for Datagram Congestion Control Protocol (DCCP), Congestion Control ID 3 : TCP-Friendly Rate Control (TFRC)". *RFC 4342*, 2006.

[89] A. Banerjea and S. Keshav. "82. A. Banerjea and S. Keshav". *Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 2*, 1993.

[90] A. Gurtov Y. Nishida T. Henderson, S. Floyd. "The NewReno Modification to TCP's Fast Recovery Algorithm". *RFC 6582*, 2012.

[91] R. Dayma and R. Vanzara. "Improved TCP Hybla : A TCP enhancement for link with high RTT and error rate". *University International Conference on Engineering (NUiCONE)*, 2012.

Publications de l'auteur

Journaux internationaux

1. Oulmahdi M, Chassot C, Van Wambek N. "Transport Protocols : Limitations, Evolution Obstacles, and Solution Approach". *International Journal in Parallel and Distributed Systems*, 2015.
2. Oulmahdi M, Chassot C, Exposito E. "Energy Saving Mechanisms on High Communication Layers". *International Journal of High Speed Networks*, Volume 4, 2013.

Conférences internationales

1. Oulmahdi M, Van Wambeke N, Chassot C. "On the feasibility of implementing TCP Using a Modular approach". *International Conference on Advanced Information and Networking Applications*, 2017.
2. Oulmahdi M, Dugue G, Chassot C. "Towards a Service-oriented and Component-based Transport Layer". *Smart Communications in Network Technologies*, 2014.
3. Oulmahdi M, Dugue G, Chassot C. "Design Principles of a Service-oriented and Component-based Transport Layer". *International Track on Adaptive and Reconfigurable Service-oriented and Component-based Architectures and Applications*, 2014.
4. Oulmahdi M, Chassot C, Exposito E. "An Energy-Aware TCP for Multimedia Streaming". *4th International Conference on SmArt Communications in NEtwork Technologies*, 2013.
5. Oulmahdi M, Chassot C, Exposito E. "Reducing Energy Cost of Keepalive Messages in 3G Mobiles". International Workshop on Energy-Aware Systems, Communications and Security, 2013.