

# Table des matières

Table des matières	1
Table des figures	5
Liste des tableaux	7
Introduction générale	9
Contexte . . . . .	9
Problématique . . . . .	10
Structure du document . . . . .	11
<b>1 Contexte, Problématique, État de l'art et Positionnement</b>	<b>13</b>
1.1 Contexte et Problématique . . . . .	14
1.1.1 Contexte . . . . .	14
1.1.1.1 Évolution des réseaux . . . . .	15
1.1.1.2 Évolution des terminaux . . . . .	17
1.1.1.3 Évolution des applications . . . . .	17
1.1.1.4 Évolution des protocoles de Transport . . . . .	18
1.1.2 Problématique . . . . .	19
1.1.2.1 Acteurs concernés . . . . .	20
1.1.2.2 Problèmes soulevés . . . . .	21
1.2 État de l'art . . . . .	24
1.2.1 Protocoles de Transport . . . . .	24
1.2.1.1 TCP . . . . .	24
1.2.1.2 UDP . . . . .	27
1.2.1.3 DCCP . . . . .	28
1.2.1.4 SCTP . . . . .	29
1.2.1.5 MPTCP . . . . .	30
1.2.1.6 CTP . . . . .	33
1.2.1.7 ETP . . . . .	34
1.2.2 En résumé . . . . .	37

1.3	Positionnement de la proposition . . . . .	38
1.3.1	Approche . . . . .	38
1.3.2	Exigences générales de conception de l'ATL . . . . .	39
1.3.3	Paradigmes de conception relatifs à l'architecture de l'ATL . . . . .	39
1.3.3.1	Conception logicielle basée composants . . . . .	40
1.3.3.2	Conception logicielle orientée service . . . . .	40
1.3.3.3	<i>Autonomic Computing</i> . . . . .	40
1.4	Conclusion . . . . .	41
<b>2</b>	<b>Architecture de l'ATL</b>	<b>43</b>
2.1	Exigences de conception et principes l'ATL . . . . .	44
2.1.1	Exigences de conception de l'ATL . . . . .	44
2.1.2	Principes des solutions proposées en réponse aux exigences de conception de l'ATL . . . . .	46
2.1.3	Approches d'implantation de ces principes . . . . .	48
2.1.3.1	Approche orientée services . . . . .	48
2.1.3.2	Approche basée composants . . . . .	50
2.1.3.3	Approche autonome . . . . .	51
2.1.3.4	Synthèse . . . . .	51
2.1.4	Fonctionnalités de l'ATL . . . . .	53
2.1.4.1	Fonction de gestion des services . . . . .	53
2.1.4.2	Fonction de gestion autonome de la communi- cation . . . . .	53
2.1.4.3	Fonction d'accueil, d'assemblage et de mise en œuvre des compositions . . . . .	54
2.2	Architecture de l'ATL . . . . .	54
2.2.1	Cas d'utilisation . . . . .	54
2.2.2	Modèle d'architecture de l'ATL . . . . .	57
2.2.2.1	Modèle général . . . . .	58
2.2.3	Composants de base de l'ATL . . . . .	60
2.2.3.1	Le <i>Flow</i> . . . . .	60
2.2.3.2	L' <i>Autonomic Manager</i> . . . . .	62
2.2.3.3	Le <i>Data Plan</i> . . . . .	66
2.2.3.4	Intégrateur des services et base de connaissances . . . . .	68
2.2.4	Composants secondaires . . . . .	69
2.2.4.1	<i>Flow Creator</i> . . . . .	69
2.2.4.2	<i>Signalization Message Controler</i> . . . . .	69
2.2.4.3	Multiplexage-démultiplexage entre couche réseau et <i>Flow</i> . . . . .	70
2.3	Conclusion . . . . .	72

<b>3</b>	<b>Description comportementale de l'ATL</b>	<b>73</b>
3.1	Création d'un point d'accès au service de Transport . . . . .	74
3.1.1	Instanciation d'un <i>Flow</i> . . . . .	74
3.1.2	Application cliente . . . . .	76
3.1.2.1	Application <i>legacy</i> . . . . .	77
3.1.2.2	Application <i>ATL-aware</i> . . . . .	79
3.1.2.3	Application <i>smart</i> . . . . .	80
3.1.3	Application serveur . . . . .	80
3.1.3.1	Application <i>legacy</i> . . . . .	80
3.1.3.2	Application <i>ATL-aware</i> . . . . .	81
3.1.3.3	Application <i>smart</i> . . . . .	81
3.1.4	Synthèse . . . . .	82
3.2	Reconfiguration d'un <i>Flow</i> . . . . .	83
3.2.1	Déroulement d'une reconfiguration . . . . .	83
3.2.1.1	Classification des cas de reconfiguration possibles . . . . .	83
3.2.1.2	Suivi du plan de reconfiguration . . . . .	84
3.2.1.3	Déroulement selon les cas . . . . .	85
3.3	Traitement des retransmissions lors des transitions . . . . .	88
3.4	Conclusion . . . . .	90
<b>4</b>	<b>Expérimentations et mesures</b>	<b>91</b>
4.1	De la composabilité des protocoles . . . . .	92
4.1.1	But de l'expérience . . . . .	92
4.1.2	Description de l'expérience . . . . .	93
4.1.2.1	Choix de l'application . . . . .	93
4.1.2.2	Micro-protocoles utilisés . . . . .	94
4.1.2.3	Simulation . . . . .	95
4.1.3	Analyse des résultats . . . . .	97
4.1.3.1	Comparaison des performances de MPTCP avec TCP et UDP . . . . .	97
4.1.3.2	Comparaison de MPTCP avec et sans mécanismes couplés . . . . .	98
4.1.4	Conclusion . . . . .	101
4.2	De l'intérêt de l'adaptation dynamique . . . . .	102
4.2.1	But de l'expérience . . . . .	103
4.2.2	Description de l'expérience . . . . .	103
4.2.2.1	Choix de l'application . . . . .	103
4.2.2.2	Micro-protocoles . . . . .	103
4.2.2.3	Simulation . . . . .	104
4.2.3	Analyse des résultats . . . . .	104
4.2.4	Conclusion . . . . .	105

## TABLE DES MATIÈRES

---

4.3	De l' <i>overhead</i> de la signalisation . . . . .	106
4.3.1	But de l'expérience . . . . .	106
4.3.2	Description de l'expérience . . . . .	107
4.3.2.1	Plateforme de test . . . . .	107
4.3.2.2	Protocoles de signalisation . . . . .	108
4.3.2.3	Protocole expérimental . . . . .	112
4.3.3	Résultats attendus . . . . .	114
4.3.4	Conclusion . . . . .	116
4.4	Conclusion . . . . .	117
<b>Conclusion générale</b>		<b>121</b>
	Rappel des contributions . . . . .	121
	Perspectives . . . . .	122
	Flux collaboratifs . . . . .	123
	Implémentation . . . . .	124
	Décision . . . . .	124
<b>A Publications de l'auteur</b>		<b>125</b>
A.1	Revue internationale avec actes et comité de lecture . . . . .	125
A.2	Conférences et workshop internationaux avec actes et comité de lecture . . . . .	125
<b>Acronymes</b>		<b>127</b>
<b>Bibliographie</b>		<b>129</b>

# Table des figures

1.1	L'environnement présente aujourd'hui des solutions multiples à des situations multiples . . . . .	20
1.2	Décomposition des fonctions de transport . . . . .	31
1.3	Décomposition de MPTCP en sous-couches . . . . .	32
1.4	Organisation schématique d'ETP . . . . .	35
2.1	Cas d'utilisation impliquant les applications . . . . .	55
2.2	Cas d'utilisation impliquant les utilisateurs humains . . . . .	56
2.3	Diagramme de structure composite de premier niveau . . . . .	59
2.4	Composition d'un <i>Flow</i> . . . . .	61
2.5	Organisation des fonctions de l' <i>Autonomic Manager</i> . . . . .	63
2.6	Caractéristiques du <i>Touchpoint</i> et lien avec l' <i>Autonomic Manager</i> . . .	64
2.7	La composition du <i>Data Plan</i> . . . . .	67
3.1	Une application de type <i>aware</i> ou <i>smart</i> ouvre un point d'accès au service de Transport via l'interface de contrôle de l'ATL . . . . .	75
3.2	Diagramme de séquence de création d'un <i>Flow</i> . . . . .	76
3.3	Instanciation d'un nouveau <i>Flow</i> . . . . .	77
3.4	Mécanisme de découverte de l'ATL . . . . .	78
3.5	Suivi du plan de reconfiguration . . . . .	86
4.1	La topologie du réseau simulé . . . . .	95
4.2	PNSR par image en utilisant MPTCP, scénario 4 . . . . .	100
4.3	PNSR par image en utilisant MPTCP-PR, scénario 4 . . . . .	101
4.4	PNSR par image en utilisant MPTCP-SD, scénario 4 . . . . .	102
4.5	La topologie utilisée pour les tests . . . . .	107
4.6	Cas d'utilisation du protocole de synchronisation d'ETP . . . . .	108
4.7	Diagramme de séquence du cas d'utilisation Distributed Sender Based .	110
4.8	Préparation à la reconfiguration . . . . .	111
4.9	Caractéristiques des cinq environnements réseau et compositions associées	113
4.10	Taux de pertes de l'environnement . . . . .	115
4.11	Taux de pertes vu par le service de Transport . . . . .	116

## TABLE DES FIGURES

---

4.12 Taux de pertes vu par le service sans système de reconfiguration . . . .	117
4.13 Délai induit par l'environnement . . . . .	118
4.14 Délai vu par le service . . . . .	119
4.15 Délai vu par le service sans système de reconfiguration . . . . .	120

# Liste des tableaux

1.1	Résumé de la problématique . . . . .	24
1.2	Résumé du problème d'extensibilité . . . . .	37
1.3	Résumé du problème de configurabilité . . . . .	37
1.4	Résumé du problème d'assujettissement . . . . .	37
1.5	Résumé du problème de déploiement . . . . .	38
2.1	Synthèse des problèmes, exigences, principes et approches de conception de l'ATL . . . . .	52
4.1	Résumé des scénarios 1 et 2 . . . . .	96
4.2	Résumé des scénarios 3 à 5 . . . . .	96
4.3	Résumé des scénarios 6 et 7 . . . . .	96
4.4	Les correspondances entre PNSR et MOS . . . . .	97
4.5	PNSR moyen en dB des simulations 1 à 4 . . . . .	97
4.6	PNSR moyen en dB des simulations 5 à 7 . . . . .	98
4.7	Délai moyen en millisecondes des simulations 1 à 4 . . . . .	98
4.8	Délai moyen en millisecondes des simulations 5 à 7 . . . . .	98
4.9	Taux de pertes des simulations 1 à 4 . . . . .	98
4.10	Taux de pertes des simulations 5 à 7 . . . . .	99
4.11	Résumé des scénarios . . . . .	104
4.12	PNSR en dB . . . . .	104
4.13	PNSR obtenu sans et avec adaptation dynamique . . . . .	105





# Introduction générale

## Contexte

L'évolution du paysage numérique au cours de la dernière décennie a été radicale à différents niveaux. L'ouverture au grand public de l'Internet et l'explosion de son utilisation qui a suivi, ont ainsi considérablement modifié l'utilisation qui est faite du réseau. Les applications multimédia et interactives sont aujourd'hui plébiscitées et ajoutent aux transferts de fichiers classiques ceux de flux audio et vidéo et plus généralement des échanges de données possédant des contraintes temporelles fortes comme les jeux en ligne. Ces nouvelles applications créent de nouveaux besoins sur le transfert des flux d'informations échangés qui ne sont plus correctement pris en compte par les solutions protocolaires traditionnelles.

Parallèlement les terminaux ont également suivi une évolution drastique, multipliant en leur sein les interfaces d'accès au réseau et donnant accès à de grandes capacités de stockage ou de puissance de calcul, y compris sur des machines nomades ou mobiles. Les parcs de serveurs (ou *datacenters*) sont devenus communs et permettent aux utilisateurs humains un accès continu et en tout lieu à leurs données personnelles. Ces machines et les communications qu'elles établissent offrent de nouvelles perspectives et créent ainsi de nouveaux contextes de besoins (en de débit ou de temps de transfert par exemple) pour lesquelles les protocoles actuels ne sont pas prévus.

Enfin, les réseaux eux-mêmes ont profondément évolué, et permettent dorénavant le nomadisme et la mobilité des terminaux de façon généralisée. Dans tous les domaines, que les réseaux soient locaux ou longue distance, avec ou sans fil, les capacités ont été décuplées. De nouveaux types de systèmes intermédiaires, appelés *middleboxes*, ont également été introduits, créant des situations inédites (le blocage de protocoles non reconnus par exemple) auxquelles ont à faire face les développeurs de solutions protocolaires.

Dans ce contexte et historiquement, les protocoles de Transport traditionnels de l'Internet, tels que TCP et UDP, ont connu plusieurs évolutions et adaptations et de nouvelles solutions ont vu le jour comme DCCP, SCTP et plus récemment

MPTCP. Le paysage des solutions protocolaires s'est donc densifié et complexifié, contribuant à créer un nouveau contexte auquel il s'agit aujourd'hui de faire face.

## Problématique

Ce nouveau contexte crée plusieurs points problématiques, tant du point de vue du développement d'applications que de la création de nouvelles solutions protocolaires.

Le développement d'applications se heurte à un choix large et complexe de solutions protocolaires au niveau Transport, requérant une expertise complète des protocoles existants, tant vis-à-vis du choix du protocole à utiliser que de sa configuration afin d'en faire une utilisation optimale. Le développeur d'applications doit donc, en plus d'effectuer les choix de la solution et de sa configuration, implémenter la gestion du protocole même au sein de son application.

Du point de vue du développeur de protocoles, la création protocolaire se heurte à deux problèmes majeurs. Le premier concerne l'extensibilité des protocoles existants. Toutes les solutions existantes ne permettent pas la création d'extensions. Certaines ne sont pas assez utilisées pour motiver la création de nouvelles options. Créer entièrement une nouvelle solution n'est pas plus aisé : en effet, celle-ci doit déjà être implémentée dans les différents systèmes d'exploitation, afin de permettre son utilisation au sein des différentes applications, qui doivent donc être mises à jour. Les *middleboxes* créent également des freins à l'adoption de nouvelles solutions protocolaires. Ces systèmes intervenant au niveau Transport, les nouvelles solutions doivent donc y être implémentées sous peine de se voir bloquées.

De ce constat global, nous dégageons cinq points de problématique limitant l'évolution du paysage protocolaire actuel au niveau Transport, relatifs : à la complexité de choix et d'utilisation des protocoles, à la configurabilité des protocoles, à leur extensibilité, à l'assujettissement des applications aux solutions protocolaires et au déploiement des nouveaux protocoles

Nous proposons dans ce manuscrit d'adresser ces cinq points via la redéfinition de l'architecture et du fonctionnement de la couche Transport actuelle, dans le but d'offrir un point d'accès générique au service de Transport qui laisse à l'application le soin de préciser le service qu'elle désire et non la solution à utiliser. Dans l'approche proposée, la solution protocolaire retenue est choisie de manière dynamique et autonome afin de fournir le service désiré par composition potentielle de plusieurs solutions protocolaires. Le but est d'offrir une abstraction plus poussée du réseau au sens large en déchargeant le développeur d'application de responsabilités qui ne devraient pas lui incomber. Le choix dynamique et autonome des solutions de Transport permet également de simplifier le déploiement de nouvelles solutions,

---

rendant leur intégration transparente et leur adoption automatique à mesure que le déploiement progresse.

Les contributions de ce manuscrit posent les bases architecturales et comportementales d'une telle couche de Transport que nous appelons *Autonomic Transport Layer* (ATL).

## Structure du document

Le présent document est divisé en quatre chapitres.

Le premier chapitre expose en détails le contexte dans lequel s'inscrivent les travaux présentés avant d'expliquer plus avant les cinq points de problématiques identifiés. Un état de l'art des solutions de Transport est ensuite exposé, décrivant le fonctionnement de différents protocoles classiques comme TCP, UDP, DCCP ou SCTP, ainsi que des solutions plus récentes ou en cours d'étude comme MPTCP, CTP ou ETP. Chacun de ces protocoles est confronté aux différents points de la problématique. Nous en déduisons les limites des solutions actuelles qui nous mènent à introduire l'approche d'une architecture nouvelle pour la couche Transport, à la fois orientée services, basée composants et dotée de capacités d'autonomie vis-à-vis des choix de composition à entreprendre pour répondre aux besoins des applications actuelles, dites *legacy*, et de celles à venir, conscientes de l'ATL.

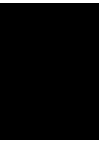
Le deuxième chapitre décrit le modèle architectural de l'ATL, c'est-à-dire la structure composite et le rôle de chaque composant nécessaire au fonctionnement de l'ATL. Dans un premier temps, nous décrivons les objectifs de l'ATL ainsi que les grands principes régissant sa conception : ceux architectures orientées services et des architectures basée composants. Dans un second temps, nous décrivons les trois grands types de fonctionnalités que l'ATL doit prendre en charge dans ses actions de gestion, de contrôle et de données. Ces différents principes mis en place, nous décrivons, suivant une approche *top/down*, la structure des différents composants de l'ATL et le rôle de chacun.

Le troisième chapitre reprend les éléments introduits dans le chapitre 2 et décrit leur comportement dans deux cas majeurs d'utilisation de l'ATL. Le premier cas est celui de l'ouverture d'un point d'accès au service de Transport. Celui-ci diffère en effet notablement de l'ouverture d'un point d'accès classique (tels que via les sockets TCP/UDP) et doit prendre en compte différents cas nouveaux introduits par la présence de l'ATL. Le second cas de figure concerne la reconfiguration dynamique du service de Transport. L'ATL offrant cette possibilité inédite, il est intéressant de décrire son déroulement afin d'en étudier par la suite les impacts sur la communication.

Le quatrième chapitre présente trois études réalisées autour de l'ATL. La première, menée en simulation, vise à combiner un protocole existant avec des mé-

canismes génériques externes afin d'observer le bénéfice retiré lors d'un transfert multimédia interactif. La deuxième, également menée en simulation, tend à vérifier le bénéfice induit par le passage d'une telle combinaison à une autre en cours de communication, en fonction de l'évolution des conditions du réseau. Ces deux études tendent à prouver qu'un véritable apport peut être retiré d'un tel fonctionnement de la couche Transport, mais appuient l'intérêt d'une mise en œuvre autonome comme celle apportée par l'ATL. La troisième expérience tente d'observer le coût induit par la signalisation nécessaire à une reconfiguration en cours de communication.

Finalement, nous terminons ce document par une conclusion générale qui résume les points abordés et les contributions apportées. Elle introduit également des perspectives de travail futur afin de préciser le modèle architectural actuel et d'étudier certaines problématiques auxquelles il est nécessaire de répondre pour offrir, via les principes de l'ATL une solution performante et utilisée.



# Contexte, Problématique, État de l'art et Positionnement

Ce chapitre présente le contexte dans lequel s'inscrit le travail décrit plus loin dans ce manuscrit. Dans la première partie, nous présentons tout d'abord l'évolution des réseaux en termes de technologies de transmissions — passage des technologies filaires aux technologies sans-fil nomades et mobiles, mais également des évolutions des terminaux qui ont vu leurs capacités augmenter, leurs types se diversifier avec l'apparition des appareils mobiles ou des capteurs, ainsi que l'apparition d'interfaces multiples, mêlant des technologies d'accès diverses sur un même appareil. Par la suite nous évoquons les différents types de services que peuvent requérir les applications en fonction de leurs besoins de communication, ainsi que l'apparition de nouveaux besoins dus à l'avènement du multimédia et des applications interactives. Finalement, nous voyons comment ces évolutions ont été adressées au niveau Transport de la pile protocolaire. Nous tirons ensuite de ce contexte cinq points de problématiques que nous nommons configurabilité, extensibilité, assujettissement, déploiement et complexité de choix.

Ces cinq points servent par la suite à confronter les différentes solutions existant au niveau Transport dans la deuxième partie du chapitre. Au cours de cet état de l'art, les protocoles classiques que sont *Transmission Control Protocol* (TCP) et *User Datagram Protocol* (UDP) et des protocoles plus récents comme *Datagram Congestion Control Protocol* (DCCP) ou *Stream Control Transmission Protocol* (SCTP) sont présentés puis mis à l'épreuve face à ces différents points. Des lacunes et mérites de chacun nous décrivons dans la troisième partie la vision scientifique dans laquelle s'inscrivent les contributions scientifiques qui seront présentées dans les chapitres suivants : celle d'une couche de Transport orientée service et basée composant, offrant un meilleur degré d'abstraction du réseau du point de vue

de l'application et permettant la gestion et l'optimisation du fonctionnement des protocoles de Transport sans intervention de cette dernière.

### Sommaire

---

1.1	Contexte et Problématique . . . . .	<b>14</b>
1.1.1	Contexte . . . . .	14
1.1.2	Problématique . . . . .	19
1.2	État de l'art . . . . .	<b>24</b>
1.2.1	Protocoles de Transport . . . . .	24
1.2.2	En résumé . . . . .	37
1.3	Positionnement de la proposition . . . . .	<b>38</b>
1.3.1	Approche . . . . .	38
1.3.2	Exigences générales de conception de l' <i>Autonomic Transport Layer</i> (ATL) . . . . .	39
1.3.3	Paradigmes de conception relatifs à l'architecture de l'ATL . . . . .	39
1.4	Conclusion . . . . .	<b>41</b>

---

## 1.1 Contexte et Problématique

Cette première partie s'articule en deux sous-parties. La première présente le contexte dans lequel s'inscrivent nos travaux, en faisant l'état de l'évolution des réseaux et de leurs utilisations en quatre temps : tout d'abord du point de vue des technologies de transmission et de leur disparité, puis des terminaux utilisant ces technologies pour communiquer entre eux et de leur diversité ; ensuite, nous nous intéressons aux applications et à l'évolution de leurs besoins avant de terminer en évoquant la manière dont les protocoles de Transport ont tenté d'apporter une réponse à ces environnements variables et nouveaux.

La seconde déduit de ce contexte les cinq points de problématique auxquels nous confrontons les différentes solutions de l'état de l'art dans la partie suivante.

### 1.1.1 Contexte

Les réseaux informatiques ont pris une place prépondérante dans notre société, notamment Internet qui a profondément modifié, et modifie encore, le tissu économique et social à un niveau mondial. Tout ceci repose aujourd'hui sur des échanges de données à très grande vitesse, sur des distances de plusieurs centaines de kilomètres, très souvent traversant plusieurs continents, connectant des terminaux très divers.

L'évolution observée depuis l'ouverture d'Internet au grand public a considérablement changé les usages qui sont faits du réseau. Le web notamment, a évolué de manière à permettre un accès simple à de nombreux usages encore trop méconnus ou complexes et leur démocratisation a permis leur utilisation aujourd'hui massive. Ainsi, à l'utilisation des mails et de l'échange de fichiers, est venue s'ajouter celle de l'audio, la vidéo ou les applications interactives, comme les jeux multi-joueurs en ligne ou les vidéoconférences.

Les besoins d'hébergement liés au grand nombre d'utilisateurs ont entraîné la création de datacenters. Au sein de ces structures, des centaines de serveurs topologiquement proches doivent communiquer à très grande vitesse, organisés en grille. Le monde de la finance a automatisé certains de ses processus, créant le high frequency trading, des transactions financières à raison de plusieurs milliers par secondes. Ces situations sont des exemples d'une évolution qui a emmené le réseau bien loin de ce à quoi il ressemblait quelques dizaines d'années auparavant. La diversité des technologies réseaux, des terminaux et des types d'applications n'a jamais été aussi grande.

Dans la suite de cette section, nous voyons l'évolution de ces trois domaines et nous évoquons comment les protocoles de Transport ont tenté de s'adapter à cette évolution.

### 1.1.1.1 Évolution des réseaux

Le contexte traditionnel et historique des réseaux est bien évidemment celui des réseaux filaires. Les différentes machines sont reliées ensemble par des câbles sur lesquels transitent les signaux électriques représentant les données à transmettre. De nombreux protocoles se sont développés suivant ce principe.

Dans le domaine des réseaux locaux, ou LAN (Local Area Network) le protocole Ethernet [IEE12a], qui s'est imposé comme incontournable, permettait initialement des capacités de 10 Mbps, permet aujourd'hui des capacités de 10 Gbps et cette capacité continue de croître. De plus, des déclinaisons de ce protocole ont été avancées afin de l'utiliser sur des réseaux métropolitains ou MAN (Metropolitan Area Network) et sur des réseaux longue distance ou WAN (Wide Area Network).

En MAN ou en WAN, les technologies xDSL (Digital Subscriber Line), connues en France pour l'ADSL (Asynchronous DSL) [ITU99a] et l'ADSL2 [ITU99b] et bientôt l'arrivée du VDSL (Very high bit-rate DSL) [ITU11b] ont permis l'accès à haut débit à la majorité des foyers, cette augmentation étant supportée dans les cœurs de réseaux par l'utilisation de protocoles comme SDH [ITU07].

L'évolution des réseaux filaires a donc été caractérisée par une augmentation des débits et un nombre croissants d'utilisateurs.

Tous les types de réseaux ont également connus des évolutions via des accès sans fil. Le WiFi [IEE12b], en constante évolution et permettant déjà des débits

de plusieurs dizaines voire centaines de Mbps, en accès WLAN comme WMAN, est utilisé par de plus en plus d'appareils, fixes, nomades ou mobiles. Les réseaux WMAN et WWAN ont également eu leur lot de nouveautés, grâce au WiMaX ou aux évolutions toujours constantes des réseaux mobiles, comme la 3G (UMTS et HSPA), et déjà 4G (LTE), et bien sûr les réseaux satellites [ETS13].

Ces réseaux sans-fil introduisent de nouveaux défis, le médium étant partagé entre tous les utilisateurs et sensible au bruit électromagnétique, les causes de pertes de données seront plus diverses et certaines hypothèses valables sur les réseaux filaires deviennent caduques. Ce type de support permet également à l'utilisateur de devenir nomade ou mobile. Il peut alors se connecter au réseau successivement depuis des localisations géographiques différentes, voire se déplacer en cours de communication. Chacun se voit connecté de manière permanente et non plus ponctuelle, via un terminal ou un autre.

Au fil du temps, différents éléments ont été introduits au sein du réseau afin d'en améliorer les performances ou de pallier des problèmes de sécurité. Certains de ces éléments brisent une notion capitale dans le modèle classique de la pile OSI : le caractère de bout en bout de la couche Transport. En effet, celle-ci est supposée n'être implémentée que sur les hôtes terminaux de la communication et non sur les éléments intermédiaires, ces derniers n'implémentant que les niveaux suivants : physique, liaison de données et réseau. Certains éléments intermédiaires, nommés middleboxes [Car02], rompent cette règle. En effet, celles-ci vont intervenir sur l'en-tête de niveau Transport, que ce soit de manière passive, en lecture uniquement, ou active, en la modifiant.

Par exemple, les Network Address Translator (NAT) ont besoin d'utiliser le numéro de port pour fonctionner, et doivent donc avoir une compréhension des différents protocoles de Transport utilisés afin d'aller lire les valeurs dont ils ont besoin dans l'en-tête de Transport. Bien qu'intervenant de manière uniquement passive sur le niveau Transport, si le NAT n'implémente pas le protocole utilisé, la communication risque d'être rompue.

De la même manière, les Performance Enhancement Proxies (PEP) interrompent, de manière transparente pour les hôtes d'extrémité, la communication. Dans le cas de TCP par exemple, ceux-ci répondent à la demande de connexion à la place de l'hôte terminal visé, se faisant passer pour celui-ci, coupant ainsi la connexion de niveau Transport en deux ou plusieurs tronçons distincts quand celle-ci n'est censée être composée que d'un seul. Comme pour les NAT, si un PEP n'a pas implémenté un certain protocole de Transport, tout trafic véhiculé par celui-ci sera interrompu par le PEP.



### 1.1.1.2 Évolution des terminaux

Parallèlement aux réseaux les terminaux ont également subi une forte évolution ces dernières décennies. L'arrivée des réseaux nomades et mobiles a permis la création de terminaux portables et transportables, dont les smartphones et tablettes sont les plus récents exemples. Les stations de travail ont disparu au profit d'une convergence avec les ordinateurs personnels, permettant à tous d'avoir accès à une puissance de traitement honorable. Les capacités de stockage ont également fortement évolué. L'accessibilité accrue à ces ressources a créé une explosion de la demande en terme de contenus. Ainsi, afin d'y subvenir, de nombreux datacenters ont vu le jour à travers le monde, créant des organisations topologiques de machines inédites, véritables concentrations de serveurs aux capacités de stockage se comptant parfois en exaoctets, et aux capacités de calcul de plusieurs teraflops.

Tout ceci crée un environnement aux caractéristiques très hétérogènes, certains terminaux possédant des débits plus importants que d'autres, des ressources en terme d'énergie, de stockage ou de puissance de calcul différentes. Un même terminal peut également voir ses caractéristiques changer en cours de communication, à cause de la charge de la batterie des terminaux mobiles, ou lorsque l'interface réseau utilisée change, la plupart des systèmes possédants aujourd'hui plusieurs interfaces, de nature différentes ou non. En effet, la plupart des smartphones possèdent au moins aujourd'hui des interfaces WiFi, Bluetooth et USB en plus de leur antenne téléphonique, celle-ci implémentant la plupart du temps des technologies 2G et 3G, voire 4G, quand ceux-ci ne disposent pas en plus du WiMaX. Les ordinateurs portables disposent presque tous d'une interface Ethernet et WiFi. La plupart des serveurs disposent aujourd'hui de plusieurs interfaces Ethernet.

### 1.1.1.3 Évolution des applications

L'augmentation des performances des machines et des réseaux a permis l'avènement de nouvelles applications et l'amélioration des applications existantes. Celles-ci font parfois preuve de besoins radicalement différents de ce que l'on connaissait. Aux applications traditionnelles sont en effet venues s'ajouter les applications multimédia et interactives.

Les applications traditionnelles, principalement basées sur l'échange de fichiers, présentaient toutes les mêmes besoins d'ordre et fiabilité totaux dans le transfert de leur données. En effet, chaque octet doit arriver intact et dans l'ordre d'émission à l'application réceptrice afin que celle-ci puisse reconstituer à l'identique le fichier d'origine, qu'il s'agisse d'un envoi de mail, d'une page web, ou d'un transfert de fichier quelconque.

Les applications multimédia quant à elles présentent des besoins différents. Par exemple, un streaming vidéo aura besoin d'ordre, mais la fiabilité n'a plus

à être totale. Il n'y a en effet aucun besoin de retransmettre une image dont la date d'affichage est dépassée. Certaines données deviennent donc obsolètes avec le temps. De plus, les méthodes de compression actuelles différencient les images leur attribuant des importances différentes. Il est donc possible d'appliquer un traitement discriminant à l'intérieur d'un flux de données, contrairement à un transfert de fichier, où chaque octet a la même importance que les autres.

Les applications interactives, mettant en relation plusieurs personnes, réagissant les unes par rapport aux actions des autres, créent encore plus de besoins inédits. Afin que les interactions entre les participants soient fluides, qu'il s'agisse de jeu vidéo ou de vidéoconférence, le délai de transit d'un participant à l'autre doit être suffisamment court pour qu'il ne soit pas perceptible. En effet, pour un jeu vidéo de tir par exemple, type FPS, l'expérience jeu serait ruinée si lorsque vous tirez sur votre adversaire, celui-ci est en réalité à un endroit différent de celui où vous le voyez sur votre écran. Dans une communication, les délais trop longs risquent, en plus de saccader la conversation, de créer des échos très désagréables.

Dans [ITU11a], l'Union Internationale des Télécommunications (ITU-T) établit une classification générale et non exhaustive des applications de types audio, vidéo et données existantes en fonction de leur tolérance aux pertes de données lors des transmissions, et des délais de transmission admissibles. Celles-ci sont réparties selon :

- leur tolérance ou intolérance aux pertes ;
- les délais qu'elles peuvent supporter, et sont alors subdivisées en quatre catégories :
  - interactivité,
  - réactivité,
  - ponctualité,
  - non criticité du délai.

Notons que certaines catégories sont apparues grâce à l'émergence des applications audio et vidéo, et que d'autres, telle la télécopie, plus tolérante par exemple qu'un transfert de fichiers, peut être satisfaite avec l'utilisation d'un service de Transport plus stricte que ce qu'elle requiert. On observe ainsi une évolution des catégories due non seulement à l'apparition de nouveaux types d'application, mais également à la précision de certaines distinctions qui pouvaient auparavant sembler superflues.

### 1.1.1.4 Évolution des protocoles de Transport

Dans ces environnements très hétérogènes, la couche Transport du modèle OSI remplit un rôle particulier. En effet, elle a longtemps été la couche la plus haute implémentée dans la pile protocolaire (si on excepte la couche applicative), et donc en charge de faire le lien entre le réseau et les applications. Ainsi, il a semblé naturel

d'introduire à ce niveau les modifications nécessaires à l'adaptation à ces nouveaux environnements. Si le paysage protocolaire de cette couche a longtemps été dominé par TCP [Pos81] et UDP [Pos80], des protocoles nouveaux ou des adaptations de protocoles existants ont été étudiés et standardisés, créant ainsi une multiplication des solutions, et une multiplication des API permettant de les utiliser.

Ces nouveaux protocoles se concentrent principalement sur la réponse à apporter à une situation donnée, leur conception souvent monolithique n'accordant pas la possibilité de choisir les fonctionnalités à utiliser suivant les cas. Le choix traditionnel étant alors limité à un service total en utilisant TCP, ou nul en utilisant UDP. La responsabilité de ce choix est d'ailleurs toujours à la charge du développeur d'applications. En effet, la vision de la couche Transport aujourd'hui communément admise est celle d'une collection de protocoles disjoints les uns des autres, qui sont appelés par l'application désirant les utiliser. Il revient donc à celle-ci d'invoquer la méthode d'appel du protocole qu'elle souhaite utiliser.

Cet appel est décidé en *design time*, c'est -à-dire que le développeur choisit le protocole qu'il juge le plus pertinent et implémente l'appel à ce protocole particulier dans son application. Cette manière de faire ne laisse aucune place à une utilisation du protocole basée sur une estimation du contexte au cas par cas, sauf à implémenter ce type de prise de décision au sein de l'application, et donc à lui faire endosser une responsabilité supplémentaire.

L'invocation d'une solution de Transport se faisant par l'appel à un protocole particulier, c'est également au développeur que revient la charge d'estimer quel protocole satisfait le mieux le service qu'il souhaite. Il existe ici un besoin d'abstraction supplémentaire, le développeur devant faire lui même la correspondance entre service et solution protocolaire.

La figure 1.1 résume les différents défis qui se présentent dans le réseau à l'heure actuelle.

### 1.1.2 Problématique

Un environnement aussi évolutif crée différents problèmes, notamment celui de la complexité pour un développeur d'application de faire le bon choix de protocole mais aussi la complexité pour le développeur de protocoles d'intégrer une nouvelle solution au paysage existant. Dans cette section, nous commençons par présenter les différents acteurs concernés par la problématique, avant d'effectuer un tour d'horizon détaillé des différents points de celle-ci. Comme nous le verrons plus tard, certains points peuvent concerner plusieurs acteurs à la fois.

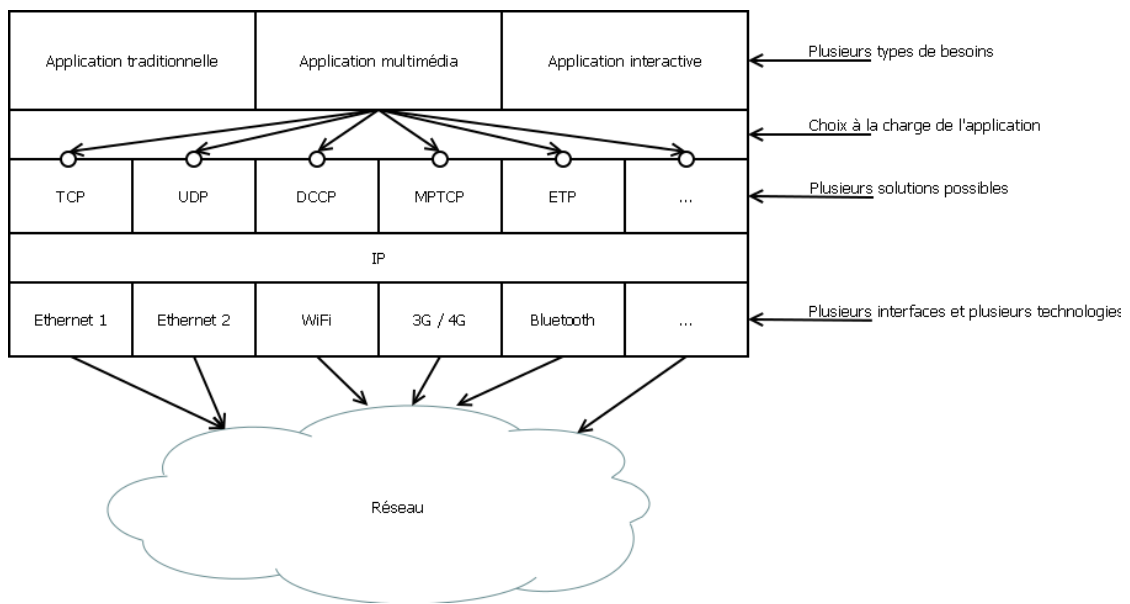


FIGURE 1.1: L'environnement présente aujourd'hui des solutions multiples à des situations multiples

### 1.1.2.1 Acteurs concernés

Trois types d'acteurs portant chacun un regard différent sur l'environnement sont identifiables :

- le développeur d'applications ;
- le développeur de protocoles ;
- le développeur de système d'exploitation.

Le souci du développeur d'applications est d'une part la simplicité d'utilisation du service, et d'autre part la conformité du service fourni par le protocole aux besoins de l'application qu'il développe. Il lui serait en effet contre-productif ou sous-optimal de devoir apprendre tous les rouages d'une solution pour l'utiliser pleinement, et d'apprendre ceci pour un nombre conséquent de solutions afin de faire des choix éclairés selon les situations. De même, il est difficilement concevable d'amener le développeur d'applications à réécrire toute ou partie de son application à l'avènement de chaque nouvelle solution au niveau Transport.

Le développeur de protocoles se soucie de la capacité à faire adopter facilement son protocole ou son amélioration d'un protocole existant. En effet, une fois la solution développée, il faut qu'elle soit promue :

1. auprès des développeurs d'applications pour les convaincre de l'intérêt d'apprendre à utiliser la nouvelle solution ;
2. auprès des développeurs de systèmes d'exploitation pour les convaincre de

l'intérêt d'intégrer la solution dans leur système ;

3. auprès des équipementiers pour les convaincre de l'utilité d'implémenter cette solution dans les différentes middleboxes qu'ils produisent.

Le développeur système se soucie quant à lui de la facilité d'intégrer une nouvelle solution ou de mettre à jour une solution existante. Il souhaite en effet minimiser la réécriture de son code pour chaque opération d'ajout ou de mise à jour.

### 1.1.2.2 Problèmes soulevés

Partant de l'évolution du contexte applicatif, machine et réseau, des réponses apportées au niveau Transport et des préoccupations des différents acteurs concernés, nous identifions cinq points de problématique. Tel qu'évoqué précédemment, certains points concernent plusieurs acteurs.

**Problème de complexité** (vis à vis de la connaissance, du choix et de l'utilisation des protocoles)

La complexité des solutions protocolaires actuelle s'exprime à différents niveaux et permet de soulever trois problèmes, décrits ci-après.

- Les protocoles de Transport sont de nature assez complexe de par leurs principes de fonctionnement, les techniques sur lesquelles sont basés leurs mécanismes, et enfin leurs algorithmes. Cette complexité a en particulier un impact direct sur la QoS offerte, dont il est important d'avoir conscience lorsque l'on souhaite répondre à des besoins en QoS des applications. Par exemple, un mécanisme de contrôle de congestion basé fenêtré, tel que celui appliqué dans la version de base de TCP, induit une variation du débit et du délai, incompatible avec les besoins en débit constant et en délai borné des applications multimédia interactives. Le premier problème alors posé résulte du constat que cette complexité n'est, dans les faits, pas transparente aux développeurs d'applications : ceux-ci doivent avoir une connaissance fine du fonctionnement du protocole sous-jacent avant de pouvoir développer leurs applications. Ceci nécessite des connaissances fines pour des développeurs dont les protocoles de Transport n'est pas le domaine d'expertise.
- Les protocoles de Transport sont également complexes dans leur utilisation. En effet, l'usage du service offert se fait via une interface de programmation (ou API pour *Application Programming Interface*), dont la maîtrise nécessite généralement des connaissances dans plusieurs domaines, notamment en programmation système.
- Enfin, la multiplication des propositions de protocoles de Transport et d'API confronte le développeur d'applications à une troisième difficulté relevant du choix de la solution la mieux adaptée aux besoins de son application.

### **Problème de dépendance (ou d'assujettissement)** (de l'application au protocole invoqué)

La complexité du choix du protocole n'est pas le seul problème qui résulte du caractère évolutif des solutions. En effet, ce choix s'effectue en *design time*, c'est à dire au moment de la conception de l'application. Ceci implique que l'application doit être écrite pour un protocole donné et s'y tenir pendant tout son cycle de vie.

Le problème vient alors du fait que la solution protocolaire choisie peut ne plus être plus adaptée aux besoins de l'application, en raison de changement des contraintes réseau par exemple, ou suite à l'évolution des besoins de l'application.

Notons que ce problème ne concerne pas uniquement les nouvelles solutions protocolaires, mais également toute migration d'une application vers une autre solution protocolaire. Ceci est dû au fait que son code est spécifique et ne peut fonctionner que sur cette solution, et tout changement de protocole doit impérativement passer par des modifications (à différentes échelles d'importance) du code de l'application. Cette dépendance aux protocoles, outre ses conséquences directes, est aussi en partie à l'origine des problèmes d'extensibilité et de déploiement décrits ci-après.

### **Problème d'extensibilité** (des solutions protocolaires existantes)

Le problème d'extensibilité comporte plusieurs facettes.

- L'intégration d'une nouvelle solution, ou la mise à jour d'une solution existante, pose un problème pour les développeurs de systèmes d'exploitation. En effet, toute modification protocolaire, impose des modifications dans les systèmes les hébergeant. Ceci concerne leur intégration dans les nouvelles versions du système, la mise à jour des versions antérieures ou la gestion de la compatibilité entre les deux. Ces implications expliquent la réticence des développeurs de systèmes d'exploitation vis à vis de l'adoption de nouvelles solutions, et conduit au constat que les environnements systèmes sont actuellement hétérogènes en termes de solutions de Transport supportées.
- Du point de vue des développeurs de protocoles, le constat précédent constitue un véritable obstacle quant à leur volonté de développer de nouvelles solutions étant donné la probabilité importante de non acceptation par les systèmes, ou au mieux, au regard des délais importants dans l'intégration (généralement plusieurs années).
- Du point de vue des développeurs d'applications, ces environnements hétérogènes les conduisent eux-aussi à prendre des précautions quant à l'usage de nouvelles solutions, personne ne pouvant accepter que son application ait des problèmes de fonctionnement pour la simple raison qu'elle tourne au sein d'un système qui ne supporte pas le protocole de Transport désiré.

En résumé, les développeurs d'applications attendent que les solutions de Trans-

port soient largement déployées pour les utiliser, pendant que dans le même temps, les développeurs de systèmes d'exploitation attendent que ces solutions soient utilisées par les applications avant de les déployer. Ce cercle vicieux fait que le problème d'extensibilité constitue la cause principale du problème de déploiement décrit ci-après.

### **Problème de déploiement** (des nouvelles solutions protocolaires)

Le déploiement de nouvelles solutions est le problème le plus restreignant dans l'évolution de la couche Transport. Il regroupe les problèmes d'extensibilité et de dépendance au protocole, en plus du problème de l'acceptabilité par le réseau.

- Tel que décrit précédemment, le problème d'extensibilité est à la source des difficultés d'adoption de nouvelles solutions par les systèmes, de leur utilisation par les applications, et en conséquence de leur développement lui-même. Vient ensuite le problème de dépendance des applications aux protocoles de Transport sous-jacents qui fait que toute modification de ces derniers impose des modifications pour les applications.
- A ceci vient s'ajouter le problème d'acceptabilité par le réseau. En effet, les middleboxes, quelles que soient leurs fonctions, ont souvent des politiques de fonctionnement basées sur les types de protocoles et leurs contenu. Toute modification d'un protocole existant ou intégration d'un nouveau protocole impose la mise à jour de ces middleboxes afin qu'elles puissent reconnaître la nouveauté. Comme certaines d'entre elles ne laissent passer que certains protocoles, un nouveau protocole risquerait de ne pas pouvoir les franchir bien qu'ils soient supportés par les systèmes d'extrémité.

### **Problème de configurabilité** (des services offerts)

Avec l'évolution des besoins en QoS des applications, les deux catégories de services offertes par les protocoles TCP et UDP sont devenues insuffisantes. Non seulement certaines applications se trouvent dans l'obligation d'utiliser des services non nécessaires induisant des quantités de données et des délais supplémentaires inutilement, mais aussi, certaines d'entre-elles peuvent voir la qualité de leurs service se dégrader à cause de ces services supplémentaires inutiles. La non configurabilité de ces protocoles qui fait que les applications sont obligées d'utiliser l'ensemble du service proposé dans son intégralité, a eu comme conséquence la complexification de la couche Transport par la création de nouveaux protocoles ne proposant aucun nouveau service, mais seulement des sous-ensembles des services des protocoles existants.

Point de problématique	Acteurs concernés
Complexité	Développeur d'applications
Assujettissement	Développeur d'applications
Extensibilité	Développeur d'OS, de protocoles et d'applications
Déploiement	Développeur de protocoles
Configurabilité	Développeur d'applications

TABLE 1.1: Résumé de la problématique

## 1.2 État de l'art

Dans cette section, nous présentons les solutions de niveau Transport existantes et les confrontons aux points de problématique identifiés précédemment. Certains de ces points ont cependant une portée globale et ne peuvent pas être étudiés ainsi au cas par cas. Le problème de complexité de choix, par exemple provient de la multiplicité des solutions qui seront évoquées dans la suite de cette section, et confronter chacune d'elle à ce problème aurait ainsi peu de sens.

La multiplicité des solutions de Transport repose aussi bien sur le nombre de protocoles existants que le nombre de versions de ces protocoles, en particulier TCP, qui a évolué et a été adapté à de très nombreuses reprises. On observe que les protocoles les plus récents s'orientent vers une résolution partielle des problèmes observés, sans jamais totalement les adresser cependant.

### 1.2.1 Protocoles de Transport

Comme la majorité du trafic utilise le protocole IP au niveau réseau, la plupart des protocoles de Transport utilisés sont ceux normalisés par l'IETF. Ces protocoles sont TCP, UDP, DCCP et SCTP, les deux premiers étant les plus utilisés car historiquement les premiers à avoir été créés. Ceux-ci offrent les deux types de services qui étaient utiles au moment de leur création. TCP propose un service garantissant un ordre et une fiabilité totaux sans garantie de délai, afin de permettre les transferts de fichiers, de mails et toutes les utilisations de ce type du réseau. UDP de son côté n'offre absolument aucune garantie et suffisait alors pour toutes les autres utilisations.

#### 1.2.1.1 TCP

TCP offre un service totalement fiable et ordonné, en se basant sur un contrôle des pertes et d'erreurs par l'envoi d'acquittements cumulatifs. Tout paquet perdu ou erroné sera ainsi non acquitté, l'émetteur du dit paquet sachant alors qu'il devra retransmettre le paquet erroné et tout paquet subséquent à celui-ci. Ce système



induit nécessairement une augmentation du délai de transit d'un paquet : la perte étant non déterministe, le nombre de retransmissions possibles d'un paquet (et donc des suivants) n'est pas borné. TCP tente également de protéger le réseau grâce à son système de contrôle de congestion, visant à limiter la quantité de paquets à traiter par les routeurs en état de saturation, et donc à limiter les pertes. Ce système implique une diminution du taux d'envoi de paquets par TCP en cas de congestion, ce qui implique de fortes variations dans les délais et les débits.

TCP a connu de nombreuses évolutions et adaptations au cours de son histoire. Les principales visent l'amélioration de son contrôle de congestion, que ce soit dans l'absolu ou dans certaines conditions particulières. D'autres concernent les options, comme les *Selective Acknowledgement* (SACK) [Mat96], permettant d'acquitter des messages correctement reçus malgré certains messages précédents manquants (ce que ne permet pas l'acquittement cumulatif) ou le *multipath* [For11] (cf. 1.2.1.5).

Les différents contrôles de congestion de TCP sont nombreux. Certains sont voués à se remplacer les uns les autres, les suivants étant considérés comme des améliorations des précédents. TCP New Reno [Hen12], par exemple, est considéré aujourd'hui comme l'implémentation de référence du contrôle de congestion de TCP et vise à remplacer les versions précédentes qu'étaient TCP Tahoe et Reno. Si cette version est celle retenue par certains OS, notamment FreeBSD, de nouvelles versions « non canoniques » ont été implémentées dans Windows et Linux.

Compound TCP [STBT08] est la version aujourd'hui utilisée sous Windows, bien qu'une version Linux ait été développée à l'université de Caltech mais non maintenue (ce patch est incompatible avec le noyau Linux à partir de la version 2.6.17). Elle est utilisée par défaut depuis Windows Vista et Windows Server 2008 bien que des mises à jour facultatives permettent de l'inclure dans Windows XP et Windows Server 2003. Tout comme TCP Vegas [BP95] ou FAST TCP [WJLH06], Compound se base sur le temps d'attente des paquets comme indication de congestion, et non sur les pertes.

Dans sa version 2.6.8, Linux utilise par défaut Binary Increase Congestion control TCP (BIC TCP) [XHR04], puis, à partir de la version 2.6.19, son évolution, CUBIC TCP [HRX08]. CUBIC n'utilise plus les retours d'acquittement comme signal d'augmentation de sa fenêtre d'émission, mais au contraire le temps écoulé depuis le dernier événement de congestion ce qui permet une croissance de la fenêtre beaucoup plus continue.

Ces différentes versions de l'algorithme de contrôle de congestion de TCP visent, tout comme HighSpeed TCP [Flo03], à améliorer les performances du protocole sur les réseaux à forte bande passante et fort délai de transit, ou Long Fat Networks, tels que définis dans [Jac88].

D'autres versions visant à adresser des contextes spécifiques, tel que Data Cen-

ter TCP [AGM<sup>+</sup>10], ont également été développées. Beaucoup de ces versions visent notamment les réseaux sans fil, parfois d'un type particulier. TCP Hybla [CF04] notamment, qui cible originellement les liens comportant des sections satellites, optimise le comportement de la fenêtre de congestion sur les chemins comportant un RTT important, en décorrélatant l'impact du RTT sur la taille de la fenêtre de congestion. Malheureusement, ses améliorations étant basées sur le comportement observé de la fenêtre de congestion en milieu satellitaire, elles offrent des performances bien moindres sur des réseaux classiques, les hypothèses sur lesquelles elles sont construites n'étant pas vérifiées.

Le cas des réseaux mobiles ajoute en plus les pertes dues au temps de handover, lors d'un passage d'une station de base à une autre. Ceci a pu être adressé par Freeze TCP [GMPG00], qui charge l'appareil mobile de signaler la proximité d'un handover en surveillant la puissance du signal reçu.

**Problème d'extensibilité.** TCP prévoit un système d'options, grâce notamment à un champ d'en-tête dédié de longueur variable, permettant d'adjoindre des comportements supplémentaires. C'est ainsi qu'en plus de l'acquittement cumulatif, TCP peut gérer des acquittements sélectifs [Mat96] ou utiliser plusieurs interfaces de manière concurrentes [For11] (cf. 1.2.1.5). Cependant le caractère monolithique de TCP, c'est-à-dire la non dissociation des différentes fonctions de son service de base comme le contrôle de congestion, de flux, des pertes et d'erreurs, rend difficile l'insertion de services modifiés comme de la fiabilité partielle. Certaines options de ce type avaient cependant été introduites, comme les acquittements négatifs [Fox89] ou la gestion de l'ordre partiel [Con94], mais devant l'impopularité de leur déploiement, celles-ci ont été officiellement abandonnées [Egg11].

**Problème de configurabilité.** TCP offre un service totalement fiable et totalement ordonné. Sa structure monolithique ne donne pas la possibilité de choisir l'une ou l'autre de ces fonctionnalités, ou de ne les utiliser que de manière partielle. Dans une certaine mesure cependant, TCP est configurable. Certains comportements, comme la concaténation de paquets de petite taille [Nag84] peuvent être désactivés manuellement. Si certains peuvent être modifiés via l'API du protocole, d'autres nécessitent une intervention dans les paramètres du système et ne peuvent donc pas être pris en charge par l'application.

**Problème d'assujettissement.** Les capacités d'autonomie de TCP sont limitées à la variation de certains de ses paramètres, comme sa fenêtre de congestion, en fonction de l'évolution du réseau. La modification d'autres paramètres tels que l'algorithme de calcul du Retransmission Time Out (RTO) doit être prise en charge

par l'application ou l'administrateur du système, ce qui implique une surveillance du réseau et une gestion du protocole de leur part. TCP dépend donc d'une entité extérieure à la couche Transport pour effectuer certains ajustements ayant trait au contrôle de la communication, ou qui pourraient l'optimiser.

**Problème de déploiement.** Par son caractère historique, TCP s'est imposé comme le protocole universel. Son déploiement ne pose donc aucun souci, chaque OS qui implémente la pile protocolaire IP intégrant une version de TCP. Il est également le protocole nécessairement implémenté dans les middleboxes présentes dans le réseau. TCP est aujourd'hui le seul protocole à pouvoir être utilisé partout, même s'il peut parfois être bloqué à cause de l'utilisation de certaines options.

#### 1.2.1.2 UDP

UDP n'offre aucune garantie d'ordre ni de fiabilité. De plus, il n'implémente aucun système de contrôle de congestion. Ceci permet de limiter les fluctuations de délai évoquées dans le cas de TCP. UDP est donc mieux adapté aux applications multimédia, mais énormément de pertes peuvent être à déplorer, limitant la qualité du flux. De plus, ne disposant d'aucun contrôle de congestion, il met le réseau en péril en risquant de congestionner les routeurs intermédiaires.

**Problème d'extensibilité.** N'offrant aucune garantie, ajouter des fonctions à UDP est possible selon le plus large spectre qui soit, cependant UDP ne prévoit aucun mécanisme permettant d'étendre celles-ci. Ainsi, toute extension devra être prise en charge par l'application ou par une couche middleware, qui invoquera elle-même UDP. Les messages de niveau middleware seront alors encapsulés dans le datagramme UDP. On ne peut donc pas littéralement parler d'extensions d'UDP.

**Problème de configurabilité.** UDP offre le service le plus simple qui soit : aucune garantie, aucun contrôle de congestion, il se contente de multiplexer l'accès des applications à la couche Transport (via les numéros de port). Les seuls paramètres configurables concernant le fonctionnement global d'UDP sur le système et ne sont pas accessibles par l'application ni ne modifient aucun mécanisme protocolaire. La configurabilité d'UDP est donc nulle.

**Problème d'assujettissement.** UDP offrant un service sans aucune garantie, tout mécanisme supplémentaire, tel qu'un contrôle de congestion, un mécanisme de fiabilité ou d'ordre partiel ou total, doit être pris en charge et implémenté au niveau de l'application.

**Problème de déploiement.** UDP est, comme TCP, présent sur la majorité voire la totalité du réseau. Il fait partie de la suite protocolaire IP et est donc implémenté dans tous les systèmes d'exploitation. Du point de vue des middleboxes, UDP est accepté dans une grande majorité, notamment car il est le protocole de Transport utilisé pour le Domain Name System (DNS).

### 1.2.1.3 DCCP

DCCP [Kho06] vise à pallier les défauts d'UDP en offrant un service non fiable et non ordonné, mais avec contrôle de congestion. Le choix du contrôle de congestion est laissé à l'application. Celle-ci peut décider d'utiliser un contrôle de congestion « à la TCP » [Flo06a], l'algorithme TCP-Friendly Rate Control (TFRC) [Flo08, Flo06b] ou bien encore TFRC for Small Packets (TFRC-SP) [Flo07, Flo09]. DCCP est ainsi plus adapté que TCP pour les applications tolérantes aux pertes et au désordre, et plus adapté qu'UDP pour les transferts sur Internet.

**Problème d'extensibilité.** Bien que DCCP inclue un champ « options » dans son en-tête, celles-ci ne servent pas, contrairement à TCP, à étendre ses fonctionnalités, mais à spécifier un certain nombre d'informations optionnelles selon le type de paquet, et selon le contrôle de congestion choisi. DCCP n'inclue donc pas nativement de mécanisme permettant l'extension de ses fonctionnalités, et du service offert.

**Problème de configurabilité.** DCCP est un protocole qui entre parfaitement dans la catégorie des protocoles configurables. En effet, comme nous l'avons mentionné ci-dessus, DCCP offre la possibilité de pouvoir choisir l'algorithme de contrôle de congestion entre trois possibilités. Ce changement doit cependant être fait à un niveau système et n'est donc pas paramétrable de manière différenciée pour chaque application à ce jour. DCCP est donc un protocole configurable de manière statique, et avec certaines lourdeurs. En effet, un tel changement au niveau système ne peut s'effectuer qu'avec les droits administrateur sur la machine concernée, droits que ne possèdent pas forcément toutes les applications. De plus, dû au caractère global de cette modification, deux applications ayant un choix différent sur la question pourraient entrer en conflit et se gêner l'une l'autre dans la configuration du protocole.

**Problème d'assujettissement.** DCCP présente sur ce point le même défaut qu'UDP. En effet, hormis le contrôle de congestion qui est à présent pris en charge par le protocole, tout mécanisme supplémentaire doit être implémenté et pris en charge par l'application afin d'être disponible.

**Problème de déploiement.** DCCP est à l'heure actuelle face à un problème de déploiement considérable. Seules trois implémentations ont vu le jour. La première, dans le noyau Linux, ne dispose pas d'une API complète. La deuxième vise un haut degré de portabilité et la troisième concerne la plateforme d'implémentation Google Go, mais aucune de ces deux implémentations ne dispose d'une version stable ni n'a bénéficié d'une mise à jour récente. De ce fait, DCCP n'est utilisé par aucune application grand public, ou à grande échelle. Cette faible utilisation fait écho à son implémentation tout aussi rare dans les différentes middleboxes présentes dans le réseau. Il est à noter qu'une mise à jour de [Kho06] effectuée par [Phe12] offre à DCCP la capacité d'être encapsulé dans un datagramme UDP (nommé alors DCCP-UDP), et non plus directement dans un paquet IP, afin de permettre le passage à travers les différentes middleboxes, et principalement les NATs. Notons cependant que si cette méthode permet de faciliter la transition de DCCP-UDP à DCCP, elle ne modifie rien quant à la transition d'UDP vers DCCP (ou DCCP-UDP).

#### 1.2.1.4 SCTP

SCTP [Ste07] a été proposé afin de tirer parti de la multiplicité d'interfaces offertes par les terminaux modernes, et ainsi pallier certains problèmes notamment dus à la mobilité de ces terminaux. SCTP va en effet utiliser toutes les interfaces disponibles pour un même flux applicatif. On ne parle ainsi plus de connexion, mais d'association. Le protocole peut ainsi utiliser plusieurs interfaces au sein d'une même association et utiliser une interface en tant que back up, afin d'assurer la continuité de la communication si la connexion principale venait à être défaillante. Plusieurs extensions ont été ajoutées au protocole comme un système de fiabilité partielle [Ste04], ou sont aujourd'hui à l'étude, comme la possibilité d'utiliser les différents chemins d'une même association de manière concurrente [DBA13].

**Problème d'extensibilité.** SCTP inclue un système similaire à celui des options de TCP. En effet, un système de champ d'en-tête optionnel permettant la définition d'extensions est présent et a notamment permis l'introduction de la gestion de la fiabilité partielle au sein du protocole [Ste04].

**Problème de configurabilité.** SCTP possède différentes options configurables à deux niveaux : au niveau du système et au niveau du socket. Comme pour les précédentes solutions, les configurations effectuées au niveau du système ne peuvent pas être réalisées par l'application, et concernent l'ensemble des associations SCTP. Celles-ci concernent notamment le calcul du *Retransmission Time Out*, ou la reconfiguration dynamique d'adresses. Les options modifiables au niveau du socket concernent les adresses primaires de l'association (celles utilisées comme chemin

principal de communication), l'algorithme de Nagel, la fragmentation de paquets, ou la récupération d'informations de monitoring, ainsi que le contrôle de la fiabilité partielle. SCTP possède donc une composante fondamentale configurable, le système de fiabilité partiel. Celui-ci est activable au besoin mais cependant entièrement contrôlé par l'application qui doit elle-même décider du nombre maximum de retransmissions acceptables. De plus, bien qu'encore aujourd'hui à l'étude, on peut supposer qu'une extension d'utilisation concurrente des différents chemins d'une même association, sera également optionnelle et contrôlable par l'application.

**Problème d'assujettissement.** SCTP présente une légère différence par rapport aux précédentes solutions, dû au fait que la fiabilité partielle est optionnelle. De ce fait, le développeur d'application est moins contraint dans l'utilisation qu'il fait du protocole, puisqu'il peut choisir tout ou partie du service présent. Cependant ce choix et sa gestion restent à l'initiative et à la charge de l'application, rendant le processus lourd.

**Problème de déploiement.** S'il est moins important que pour DCCP, SCTP est également confronté à certains problèmes de déploiement. En effet, tous les systèmes d'exploitation ne l'intègrent pas en natif ce qui limite d'autant son utilisation. De plus, le service de base offert par SCTP étant redondant avec celui de TCP, peu de développeurs d'applications décident de se former à l'utilisation de ce nouveau protocole pour mettre à jour leur application, l'utilisation de TCP étant plus sûre. En effet, ce dernier est disponible partout et implémenté dans la totalité des middleboxes du marché, contrairement à SCTP, souvent bloqué.

### 1.2.1.5 *Multipath* TCP (MPTCP)

MPTCP [For11] est une extension au protocole TCP traditionnel qui consiste à lui adjoindre la capacité à utiliser plusieurs interfaces simultanément, et de manière concurrente. L'initiative de créer ce *working group* à l'IETF découle des soucis d'adoption que rencontre SCTP depuis sa création. Ainsi, le projet MPTCP consiste à tirer parti des avantages induits par l'utilisation de toutes les interfaces d'une machine, concept dont l'intérêt a été prouvé avec SCTP, tout en acceptant le constat que seul TCP est accepté partout dans le réseau. MPTCP est conçu de manière à être utilisé de manière transparente par les applications et par le réseau, sans modification nécessaire de ces derniers. Cependant, des applications conscientes de la présence de MPTCP peuvent en tirer parti plus conséquemment. Étant une extension de TCP, MPTCP vise à fournir le même service d'ordre et de fiabilité totaux sans garantie de délai, tout en améliorant la résilience, notamment en mobilité (par l'utilisation de plusieurs chemins simultanément), et en améliorant

également le débit, par addition des débits possibles sur chaque interface (l'objectif étant de toujours faire au moins aussi bien que TCP dans sa version classique).

**Problème d'extensibilité.** MPTCP se base sur une décomposition de la couche Transport en sous-couches, idée proposée dans les travaux de *Transport Next Generation* (Tng) [FI08], en attribuant à une sous-couche supérieure les fonctions relatives à la sémantique applicative, et à une sous-couche inférieure celles relatives à la gestion du réseau, comme décrit sur la figure 1.2.

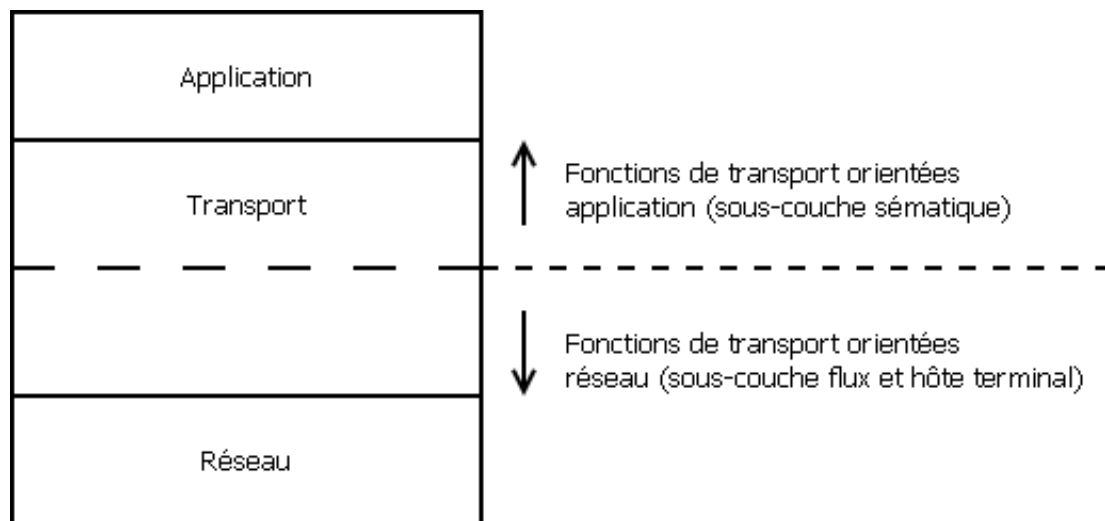


FIGURE 1.2: Décomposition des fonctions de transport

En se basant sur ce concept, MPTCP décompose la couche Transport comme sur la figure 1.3.

En offrant ce cadre, MPTCP permet l'étude et l'intégration de fonctions étendues en fonction de leur orientation réseau ou sémantique applicative. Ainsi, peut-on imaginer l'intégration de différents algorithmes de contrôle de congestion dédiés aux protocoles multi-chemins tels que proposés dans [DABR12], que l'on pourrait échanger selon les conditions du réseau et les besoins de l'application. De la même manière, la sous-couche sémantique pourrait accueillir des fonctions permettant de modifier le service offert MPTCP, y intégrant de la fiabilité ou de l'ordre partiels par exemple. Cependant, les fondations même de MPTCP reposent sur un ensemble de flux à ordre et fiabilité totaux, contraignant les modifications possibles du service offert. Ainsi, introduire une contrainte de délai impliquerait impérativement un impact sur le débit, ces deux grandeurs étant liées par l'usage de TCP.

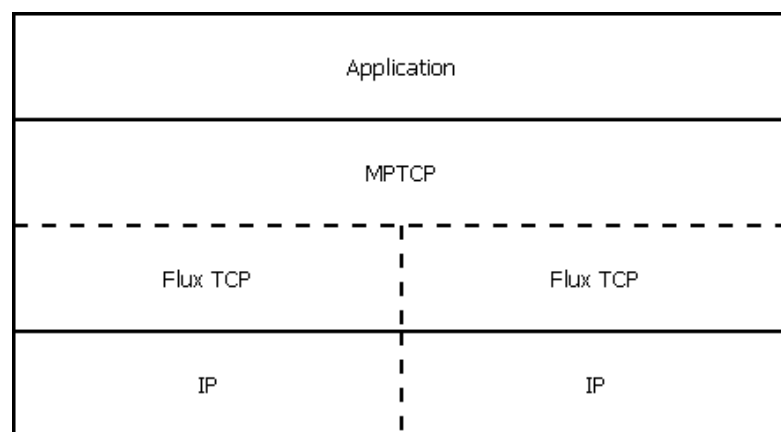


FIGURE 1.3: Décomposition de MPTCP en sous-couches

**Problème de configurabilité.** MPTCP étant encore en cours de standardisation et d'implémentation, sa configurabilité n'est pas encore définitivement établie. Cependant, certaines extensions à l'API socket classique ont été préconisées dans [Sch13]. Ces extensions, nommées *basic API*, servent à des applications qui ont conscience de la présence sous-jacente de MPTCP. Celles-ci permettent d'activer ou de désactiver l'utilisation de MPTCP (et ainsi n'utiliser que TCP en version classique), ou de modifier l'ensemble des adresses, donc des interfaces, à utiliser dans la connexion MPTCP qu'elles emploient. D'autres options permettent de récupérer des informations propres à une connexion MPTCP, comme la liste des adresses utilisées par la connexion. Il est à noter que [Sch13] suggère dix extensions supplémentaires à l'API décrite afin de constituer une API avancée, permettant un contrôle plus poussé de l'application sur le protocole, par exemple en communiquant un profil décrivant les attentes de l'application en termes de performance, ou de caractéristiques concernant la communication (débit d'émission, délai inter-paquets, *etc.*). Cette API avancée n'est à l'heure actuelle ni standardisée, ni à l'étude.

**Problème d'assujettissement.** Tel que les RFCs le décrivent aujourd'hui, l'application ne peut qu'utiliser le service totalement fiable et totalement ordonné fourni par MPTCP. Bien que permettant l'introduction de mécanismes modifiant ce comportement, aucun n'est aujourd'hui en cours de standardisation, le protocole étant encore jeune. La gestion du protocole respecte le paradigme communément admis dévouant cette charge à l'application, et nécessitant donc une connaissance et une expertise de la part du développeur d'applications.



**Problème de déploiement.** MPTCP présente de nombreuses forces concernant son déploiement et son adoption. Du point de vue du développeur d'applications, son avantage principal est d'être géré avant tout par le système, et ne nécessite donc aucune modification dans le code des applications pour être utilisé. MPTCP est donc d'ores et déjà compatible avec toute application utilisant TCP sans réclamer de mise à jour de celles-ci. Son utilisation la plus basique ne réclame pas de formation spécifique du développeur d'applications non plus, si tant est que celui-ci soit déjà familier avec l'utilisation de TCP. Du point de vue du réseau, MPTCP est vu par ce dernier comme plusieurs flux TCP, indépendants les uns des autres. Ainsi, la majorité des middleboxes laisseront-elles ces flux circuler sans encombre. Les quelques problèmes possibles proviendront essentiellement des middleboxes bloquant les options TCP inconnues (MPTCP se manifestant par une nouvelle valeur d'option dans l'en-tête des segments TCP), ainsi que celles modifiant les numéros de séquence, MPTCP mettant en place une correspondance entre les numéros de séquence de chaque flux, et un numéro de séquence applicatif.

#### 1.2.1.6 CTP

Introduit dans [BWH<sup>+</sup>07], *Configurable Transport Protocol* (CTP) est un protocole visant des capacités élevées en termes de configurabilité. CTP se base sur le principe de la composition de micro-protocoles introduits dans [Exp03]. Les micro-protocoles représentent chacun une fonction de Transport de base, atomique. Leur composition permet ainsi la création d'un service de Transport à la carte en fonction des besoins de l'application. CTP fonctionne sur un principe évènementiel : chaque message arrivant dans le protocole déclenche une série d'évènements en fonction des informations contenues dans son en-tête. Chaque évènement informe un micro-protocole qu'il doit prendre le message en charge. Ceci permet de paralléliser les traitements lorsqu'ils sont indépendants les uns des autres.

**Problème d'extensibilité.** Afin de permettre le déploiement de la composition adéquate de part et d'autre de la communication, CTP utilise un champ de 32bits, associant chacun à un micro-protocole particulier. Ainsi, bien que les auteurs appellent à la création de nouveaux micro-protocoles dans [BWH<sup>+</sup>07], CTP est limité à une collection ne dépassant pas les 32 micro-protocoles. Ce nombre, s'il est suffisant pour des services de Transport « classiques » risque de se révéler faible dès que l'on considère la possibilité d'utiliser plusieurs implémentations d'un même micro-protocole en fonction des situations, ou que l'on décide d'ajouter des micro-protocoles implémentant des fonctions plus exotiques, relatives à la Qualité de Service ou à la sécurité par exemple.

**Problème de configurabilité.** CTP a été créé dans le but d'offrir une configurabilité maximale. En effet, l'application peut choisir la composition de micro-protocole qu'elle désire, et ainsi la dessiner selon ses besoins, sans devoir utiliser des services qui lui seraient inutiles, ou se résigner à implémenter elle-même ceux qui manquent, comme dans le cas des solutions monolithiques classiques. De plus, même si elle est limitée, l'extensibilité de CTP offre la possibilité d'ajouter des fonctions supplémentaires nécessaires à la satisfaction des besoins de l'application. CTP n'est cependant pas reconfigurable en cours de communication, forçant l'application à conserver le même service composite tout au long de celle-ci.

**Problème d'assujettissement.** Le manque de configurabilité dynamique contraint donc l'application dans le service qu'elle décide à l'ouverture de connexion. Ce n'est cependant pas le seul point sur lequel l'application se retrouve assujettie à CTP. En effet, la gestion complète du protocole repose sur elle. C'est ainsi à elle de décider de la meilleure composition de micro-protocoles à utiliser. Il est d'ailleurs précisé que cette composition devra être établie par l'expérience, aucun système de décision n'étant présent pour épauler l'application dans son choix.

**Problème de déploiement.** Cette dernière contrainte est un véritable frein à l'adoption de CTP. Plus encore que pour les solutions précédentes, le développeur d'applications doit apprendre en profondeur le fonctionnement du protocole pour être capable de l'utiliser. De plus, son fonctionnement différant fortement de celui des protocoles classiques, toute application souhaitant l'utiliser devra être mise à jour. Finalement, toutes les middleboxes devront également être capable de le prendre en charge, CTP n'étant pas conçu pour être transparent du point de vue du réseau.

### 1.2.1.7 ETP

Développé au LAAS-CNRS, *Enhanced Transport Protocol* (ETP) est un *framework* pour la composition de protocoles de Transport composites. Tout comme CTP, il repose sur la composition de micro-protocoles créant ainsi des protocoles composites répondant aux besoins de l'application. ETP adopte une structure différente de celle de CTP, préférant une approche hybride entre organisation hiérarchique, comme le modèle OSI, et le modèle événementiel, sur lequel repose CTP. Cette organisation schématisé sur la figure 1.4 est articulée autour de cinq conteneurs :

- Application Out (AppOut) ;
- Application In (AppIn) ;
- Protocol Out (ProtOut) ;
- Protocol In (ProtIn) ;

– Management.

Chaque conteneur se destine à recevoir les différentes fonctions nécessaires au fonctionnement des différents micro-protocoles qui vont constituer le protocole. Ainsi, le conteneur de management va-t-il accueillir les fonctions de contrôle dudit micro-protocole. Ce conteneur adopte une approche événementielle. Les fonctions qui y prennent place vont s'activer sur notifications d'évènements produites par les fonctions qui prendront place dans les autres conteneurs.

Ces quatre autres conteneurs peuvent se diviser selon deux axes. Un axe vertical sépare d'un côté les conteneurs AppOut et ProtOut, et les conteneurs AppIn et ProtIn de l'autre, formant un plan de données en sortie d'une part, qui traitera les messages émis par l'application, et un plan de données en entrée, qui traitera les messages reçus à destination de l'application.

Ils se divisent également selon un axe horizontal, regroupant les conteneurs AppOut et AppIn d'une part, et ProtOut et ProtIn d'autre part. Les premiers exécutent leurs fonctions au rythme des demandes d'émission et de réception de l'application. Les seconds exécutent les fonctions qu'ils hébergent au rythme auquel le réseau les accepte ou les transmet. Ces rythmes pouvant être très différents, des buffers font office de tampon entre les deux plans horizontaux (App et Prot) pour absorber cette différence.

Ces quatre conteneurs s'organisent de manière hiérarchique. Les différentes fonctions qui y sont hébergées sont empilées selon un certain ordre, et s'exécuteront sur les différents messages dans cet ordre.

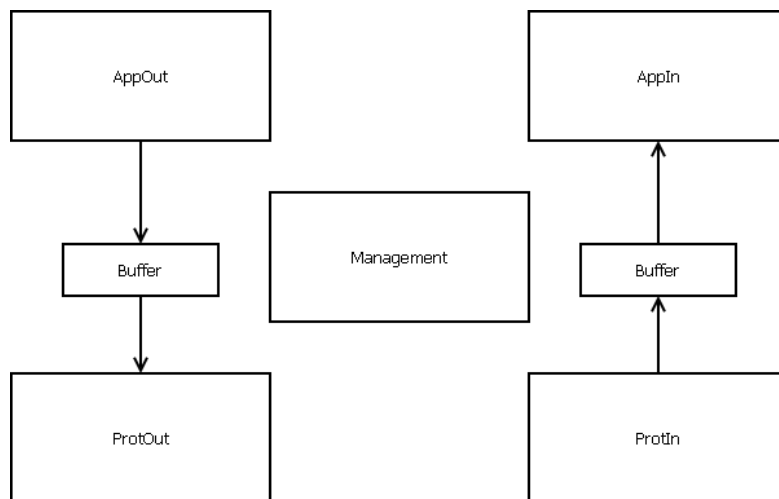


FIGURE 1.4: Organisation schématique d'ETP

**Problème d'extensibilité.** ETP prévoit la possibilité d'accueillir de nouveaux micro-protocoles. Ceux-ci sont désignés par un identifiant unique, rendant les possibilités d'extensions quasiment illimitées. Le système de synchronisation, contrairement à celui de CTP, n'est pas limité en place ce qui permet également cette extensibilité illimitée. Chaque micro-protocole pouvant adjoindre son ou ses champs d'en-tête personnalisés, cependant, le risque créé est celui d'un *overhead* important lors de l'utilisation de nombreux micro-protocoles dans le même protocole composite, risquant de provoquer de la fragmentation au niveau IP.

**Problème de configurabilité.** ETP est configurable à l'initialisation et reconfigurable dynamiquement en cours de communication. Le déploiement de la nouvelle composition du protocole s'effectue en parallèle de l'ancienne, permettant ainsi le traitement d'éventuelles retransmissions ou de paquets encore en vol. Cette méthode s'accompagne cependant d'un coût en termes de ressources utilisées la rendant peu viable sur les terminaux limités, tels que des capteurs. La reconfiguration dynamique requiert de plus une signalisation propre induisant un certain *overhead* sur le réseau.

**Problème d'assujettissement.** Les configurations d'ETP sont décidées par l'application, imposant au développeur d'applications de connaître les micro-protocoles qui s'offrent à lui mais également de décider de leur organisation hiérarchique au sein des conteneurs du plan de données. C'est également à l'application de commander les différentes reconfigurations en cours de communication, lui laissant la tâche de surveiller, en plus de ses propres besoins, si l'état du réseau réclame de modifier le service fourni par le protocole.

**Problème de déploiement.** ETP repose techniquement au dessus d'UDP et est ainsi vu par le réseau comme la charge utile du protocole de Transport. ETP peut donc être accepté par le réseau où UDP l'est, c'est-à-dire quasiment partout. ETP doit cependant être utilisé par les développeurs qui sont contraints, pour ce faire, d'apprendre son utilisation. Proche de celle d'UDP, elle doit cependant être complétée en fournissant au protocole la composition de micro-protocoles à utiliser. C'est également au développeur d'applications de décider des changements de composition. Tout ceci fait peser un besoin en connaissance et une responsabilité supplémentaire sur le développeur d'application rendant improbable le déploiement d'ETP en pratique.

### 1.2.2 En résumé

Les différents tableaux suivants résument les différents points exposés sur les différents protocoles.

Protocole	Extensibilité
TCP	Par options
UDP	Aucune
DCCP	Aucune
SCTP	Par options
MPTCP	Structure en couche pouvant accueillir des extensions
CTP	Limitée
ETP	Forte

TABLE 1.2: Résumé du problème d'extensibilité

Protocole	Configurabilité
TCP	Limitée
UDP	Nulle
DCCP	Contrôle de congestion
SCTP	Limitée
MPTCP	Basique standardisée, étendue prévue
CTP	Totale
ETP	Totale

TABLE 1.3: Résumé du problème de configurabilité

Protocole	Assujettissement
TCP	Service monolithique
UDP	Service monolithique
DCCP	Choix du CC à la charge de l'application
SCTP	Choix à la charge de l'application
MPTCP	Choix à la charge de l'application
CTP	Choix à la charge de l'application
ETP	Choix à la charge de l'application

TABLE 1.4: Résumé du problème d'assujettissement

L'abondance de ces différents paramètres nous place face au problème de la complexité des choix qu'ont à faire les développeurs d'applications. En effet, un

Protocole	Déploiement
TCP	Disponible partout
UDP	Disponible partout
DCCP	Faible
SCTP	Faible
MPTCP	A venir
CTP	Aucun
ETP	Aucun

TABLE 1.5: Résumé du problème de déploiement

protocole donné, même s'il est faiblement déployé, peut donner de meilleures performances dans un réseau fermé, non relié à Internet, qu'un protocole plus vastement déployé. Cependant, ceci peut mener à des complications si l'application devait finalement être portée sur un contexte de réseau plus ouvert, relié à Internet.

Le choix que doit faire le développeur d'application, pour être éclairé, doit être basé sur une connaissance poussée des différents protocoles disponibles. Or la complexité de certains, standardisés parfois par une grande quantité de documents, rend cette tâche ardue, consommatrice de temps et réclamant un effort sur une expertise qui ne devrait pas relever du développement logiciel. Pour cette raison, et grâce à son omniprésence, TCP est souvent le choix fait par défaut, notamment car de plus en plus d'applications utilisent HTTP pour leur transferts, et que celui-ci repose sur TCP. Cependant, le service de TCP est faiblement configurable, et c'est alors à l'application de prendre en charge les modifications qu'elle souhaite voir appliquées.

## 1.3 Positionnement de la proposition

Face aux limites des protocoles de Transport, la thèse soutenue dans ce mémoire est celle d'une architecture innovante pour la couche Transport, nommée ATL, qui permet de répondre aux différents points de la problématique discutés précédemment.

Cette section en présente l'approche, les exigences générales et les principes de conception retenus en conséquence. Les chapitres 2 et 3 sont dédiés à l'explicitation et aux détails de mise en œuvre de ces trois volets.

### 1.3.1 Approche

L'objectif n'est pas de définir un nouveau protocole, mais de tirer partie de l'ensemble des solutions protocolaires (protocoles et mécanismes) existantes ou

à venir, par le biais d'un environnement d'intégration et de composition au sein d'une véritable couche Transport, dans le but de répondre au mieux aux besoins des applications en tenant compte des capacités du réseau et des systèmes terminaux.

La proposition présentée dans ce document traite également des considérations de déploiement de l'ATL qui doit tenir compte, d'une part, du degré de conscience de l'ATL de la part des applications, et d'autre part, de la disponibilité de l'ATL dans les différents systèmes s'interconnectant. En d'autres termes, la mise en œuvre de l'ATL sur un hôte terminal ne doit pas conduire au besoin de réécrire les applications traditionnelles dites « *legacy* », développées en amont de l'ATL ; de même, le déploiement de l'ATL sur un hôte terminal doit permettre à ce dernier d'interagir avec un hôte sur lequel l'ATL n'est pas déployé, que ce soit en tant qu'initiateur (client) de la communication, ou en tant que serveur.

### 1.3.2 Exigences générales de conception de l'ATL

Cette nouvelle architecture pour la couche Transport doit permettre de faciliter l'accès aux protocoles de Transport en abstrayant le développeur d'applications de la complexité relative au choix de protocole et à son utilisation.

Cette couche vise également à permettre un accès à l'ensemble des services disponibles, ceci de façon hautement configurable et personnalisé en fonction du contexte applicatif, machine et réseau, brisant ainsi la dépendance classique entre applications et protocoles de Transport dans le but d'offrir davantage de souplesse au développement et à la maintenance des applications. L'un des objectifs de conception de l'ATL est qu'il dispose de capacités d'autonomie qui lui permettent de tenir compte en temps réel, dans ses choix de composition, des évolutions des besoins applicatifs et de l'état du réseau.

Par ailleurs, l'architecture de l'ATL doit faciliter le développement, la mise à jour et l'intégration des nouvelles solutions protocolaires en exonérant les développeurs d'applications, de protocoles et de systèmes des conséquences liées à toute sorte de modification dans les protocoles de Transport.

Enfin, l'ATL doit prendre en compte d'autres considérations relatives au déploiement de ces nouvelles solutions.

### 1.3.3 Paradigmes de conception relatifs à l'architecture de l'ATL

Pour répondre aux exigences de conception de l'ATL, quatre paradigmes de base ont été retenus, dont l'application sera décrite dans les chapitres 2 et 3 :

- le paradigme associé aux architectures dites basées-composant ;
- le paradigme associé aux architectures dites orientées-services ;

- le paradigme de l'*Autonomic Computing* (AC) ;
- la dimension sémantique.

### 1.3.3.1 Conception logicielle basée composants

La notion de « basé composants » est un paradigme de conception logicielle qui vise à faciliter le développement des systèmes complexes en séparant ces derniers en un ensemble de composants élémentaires. La facilité vient du fait que la division du système le rend moins complexe et permet en plus de déléguer chaque composant à un groupe de développeurs appropriés. Par ailleurs, les composants étant développés séparément, ils peuvent être utilisés par plusieurs autres composants par le biais d'interfaces clairement définies. Cette capacité de réutilisation permet d'économiser du temps et de l'espace. Généralement, le recours au basé composant est à des fins d'optimisation. Cependant, dans le cas d'architectures extensibles, comme le cas des drivers systèmes par exemple, l'approche de conception logicielle basée composant est la seule solution pour assurer cette extensibilité.

### 1.3.3.2 Conception logicielle orientée service

L'« orienté services » est un autre paradigme de conception logicielle qui permet une séparation nette entre le service offert et la façon avec laquelle il est offert (son implémentation). Cette séparation offre, d'une part, une abstraction complète en réduisant les interactions à l'interface du service. D'autre part, elle permet plus de souplesse à l'architecture quant à sa maintenance et son extensibilité. La gestion des services dans de telles architectures est, par exemple, effectuée en se basant sur des descriptions sémantiques des services et de leurs interfaces.

### 1.3.3.3 *Autonomic Computing*

Le concept d'AC a été proposé par IBM pour faire face à la complexité croissante dans la gestion jusque-là manuelle des différentes facettes relevant des technologies de l'information, et en particulier des systèmes logiciels. Le principe est d'intégrer des capacités d'auto-gestion dans ces systèmes induisant une intervention minimale des utilisateurs. IBM a également proposé une architecture fonctionnelle basée sur un ensemble de composants et d'interfaces bien définies ainsi que d'un cahier des charges précis du comportement individuel et collectif attendu de la part des différents composants du système. Cette architecture, dont les points d'entrée et de sortie sont équivalents à des capteurs (*sensors*) et à des actionneurs (*actuators*) distingue quatre fonctions majeures : le monitoring du système auto-géré, l'analyse des données capturées et l'identification de symptômes, la planification des actions à entreprendre pour répondre à ces symptômes, et finalement la mise en œuvre de ces actions. Dans les grandes lignes, les actions entreprises



peuvent relever de la configuration du système, de l'optimisation de ce système, de sa réparation ou encore de sa sécurisation.

## 1.4 Conclusion

L'objectif général de ce chapitre était de présenter la problématique adressée dans notre travail de thèse et de positionner la proposition soutenue dans l'état de l'art des protocoles de Transport sous-jacents au transfert des applications distribuées dans l'Internet.

Pour cela, nous avons tout d'abord présentés les évolutions de ces 20 dernières années du contexte de notre travail, constitué des réseaux supports de l'Internet, des applications distribuées et de leurs besoins, et enfin des terminaux sur lesquelles s'exécutent ces applications.

Nous avons ensuite analysé les limites des protocoles de Transport face à ces évolutions, en identifiant cinq points durs de problématique :

- problème de complexité de choix ;
- problème d'extensibilité ;
- problème de configurabilité ;
- problème d'assujettissement ;
- problème de déploiement ;

auxquels nous avons confronté les différentes solutions de l'état de l'art selon trois points de vue : le développeur d'application, le développeur de système d'exploitation et le développeur de protocoles. Si certains de ces protocoles répondent à différents degrés à une partie de ces points, aucune ne les satisfait tous. Notamment, la complexité de choix et l'assujettissement ne sont adressés par aucune.

La démonstration de ces limites nous a conduits à positionner notre proposition, l'ATL. L'ATL n'est pas une nouvelle proposition de protocole, mais une proposition d'architecture pour la couche Transport, orientée service et basée composants, créant une véritable abstraction du service sous-jacent. Les enjeux de l'ATL sont multiples et visent en particulier à soustraire le choix et la gestion du protocole de Transport utilisé au développeur d'applications, lui permettant ainsi de concentrer sa demande sur le service requis et non plus la solution protocolaire à mettre en place pour assurer ce service. L'ATL permet ainsi l'intégration transparente de nouveaux protocoles et mécanismes protocolaires sans intervention au niveau de l'application.

Dans la suite de ce document, le chapitre 2 se focalise sur la définition des différents composants de l'ATL, à différents niveaux de description, leurs rôles, leur organisation et les échanges qu'ils effectuent entre eux. Le chapitre 3 se concentre ensuite sur la description comportementale de ces composants dans deux cas d'uti-

## 1. CONTEXTE, PROBLÉMATIQUE, ÉTAT DE L'ART ET POSITIONNEMENT

---

lisation centraux : l'ouverture d'un point d'accès au service de Transport par une application, et la reconfiguration dynamique d'un service de Transport.

# Architecture de l'ATL

Ce chapitre présente le modèle architectural (ou structurel) de l'ATL suivant le plan ci-dessous.

Dans la première partie (section 2.1), nous détaillons les exigences de conception de l'ATL introduites au chapitre 1. Nous présentons ensuite les principes permettant d'y répondre et enfin les approches logicielles envisagées pour implanter ces principes. De l'exposé de ces approches, nous dégageons enfin les fonctionnalités attendues de l'ATL.

La seconde partie du chapitre (section 2.2) présente le modèle architectural de l'ATL suivant le formalisme UML 2.0. Les cas d'utilisation de l'ATL sont tout d'abord introduits. Le modèle général de l'ATL est ensuite présenté. Vient enfin la présentation des différents composants de l'ATL distingués en composants de premier niveau et composants secondaires.

La dernière partie (section 2.3) conclut le chapitre et introduit le chapitre 3, dédié à la description comportementale de l'ATL au travers de deux cas d'utilisation.

## Sommaire

2.1	Exigences de conception et principes l'ATL . . . . .	44
2.1.1	Exigences de conception de l'ATL . . . . .	44
2.1.2	Principes des solutions proposées en réponse aux exigences de conception de l'ATL . . . . .	46
2.1.3	Approches d'implantation de ces principes . . . . .	48
2.1.4	Fonctionnalités de l'ATL . . . . .	53
2.2	Architecture de l'ATL . . . . .	54
2.2.1	Cas d'utilisation . . . . .	54
2.2.2	Modèle d'architecture de l'ATL . . . . .	57

2.2.3	Composants de base de l'ATL . . . . .	60
2.2.4	Composants secondaires . . . . .	69
2.3	Conclusion . . . . .	72

---

## 2.1 Exigences de conception et principes l'ATL

Cette section présente les exigences de conception et les principes de l'ATL. Notons que cette section ne présente pas l'architecture de l'ATL, cette partie faisant l'objet de la section 2.2.

### 2.1.1 Exigences de conception de l'ATL

Pour aborder les cinq points de la problématique énoncée au chapitre 1, l'ATL doit répondre à un certain nombre d'exigences que nous introduisons ci-après.

**Face à la complexité liée à la connaissance, au choix et à l'utilisation des solutions protocolaires,** l'ATL doit permettre aux développeurs d'applications :

- de se voir masquer la connaissance et la complexité du choix des solutions protocolaires à invoquer ; autrement dit, partant d'une spécification par le développeur d'application des caractéristiques du service souhaité, c'est à l'ATL que revient le choix de la solution protocolaire sous-jacente la plus adaptée ; notons que cette abstraction ne doit pas fermer la possibilité pour un développeur averti, d'effectuer par lui-même le choix de la solution protocolaire à mettre à œuvre ;
- de se voir masquer la complexité d'utilisation des solutions protocolaires qui seront invoquées ; autrement dit, le développeur d'application doit disposer d'une *Application Programming Interface* (API) générique, lui permettant de caractériser le service souhaité, et non le protocole ; l'ATL doit cependant permettre la prise en compte des applications existantes — applications dites *legacy* — traditionnellement basées sur TCP et UDP et sur les API correspondantes — il ne s'agit donc pas de réécrire ces applications — une API supplémentaire, permettant d'exprimer la solution protocolaire souhaitée, doit enfin être mise à disposition des développeurs avertis.

**Face au problème de dépendance des applications aux solutions protocolaires,** l'ATL doit permettre une adaptation de la solution protocolaire retenue en réponse à une évolution du contexte machine/réseau, ou bien à une modification de l'expression du service requis par l'application. Cette adaptation consiste

soit en la modification de certains des paramètres de la solution, soit en la modification de la structure même de la solution, par exemple par remplacement d'un mécanisme protocolaire par un autre, plus adapté au nouveau contexte.

- En réponse à une évolution du contexte machine/réseau, l'ATL doit donc être doté d'une capacité d'autonomie dans la prise de connaissance du contexte et de ces évolutions, en plus de celle liée au choix et à la mise en œuvre des solutions protocolaires à invoquer.
- En réponse à une évolution des besoins applicatifs, la même capacité d'autonomie est requise de la part de l'ATL dans le choix et la mise en œuvre des adaptations de la solution protocolaire en cours. Vis à vis de la prise de connaissance de l'évolution des besoins applicatifs, l'ATL doit au moins offrir — via l'API générique — la possibilité aux développeurs d'applications de modifier les caractéristiques du service requis durant le temps d'exécution de l'application. Cette possibilité est donc offerte à des applications évoluées, conçues de sorte à pouvoir adapter leurs besoins aux variations du contexte considéré. Notons qu'idéalement, l'ATL pourrait être doté d'une capacité d'autonomie dans la prise de connaissance du contexte applicatif, et inférer par lui-même l'évolution des besoins applicatifs ; cette dernière possibilité n'a pas été considérée dans nos travaux et en constitue l'une des perspectives.

**Face au problème d'extensibilité des solutions protocolaires,** lorsqu'apparaît une nouvelle version d'un protocole existant ou bien qu'émerge une nouvelle solution protocolaire — qu'il s'agisse d'un protocole ou d'un mécanisme, l'ATL doit permettre son intégration au sein de l'ensemble des services et solutions protocolaires qu'il gère déjà, en vue de la mise à disposition du service correspondant et de l'utilisation de la solution protocolaire.

**Face au problème de manque de configurabilité du service requis,** c'est à dire la difficulté pour un développeur d'application de pouvoir requérir un service particulier plutôt que tous ceux imposés par les protocoles basés sur un principe de conception « monolithique » tels que TCP, l'ATL doit résoudre le problème grâce à la nature « composable » des différents protocoles et mécanismes protocolaires qu'il gère (chaque protocole pouvant se voir adjoindre des mécanismes externes), mais également par une connaissance complète de l'étendue de la configurabilité de chaque protocole et mécanisme protocolaire qu'elle peut instancier.

Ceci permet un plus grand contrôle et une plus grande latitude dans les décisions concernant les services que l'ATL peut mettre en œuvre pour satisfaire les besoins des applications. Grâce à ces principes, l'ATL peut virtuellement configurer n'importe quel service à volonté, via les paramètres variables des composants

qu'elle instancie, ou par l'adjonction de mécanismes externes venant palier un manque.

**Face au problème d'acceptabilité des nouvelles solutions protocolaires,** on observe les deux points suivants.

- Le principal souci lié aux solutions nouvelles tient à leur capacité à être utilisées par les applications et à être acceptées par le réseau. En libérant l'application du choix de la solution protocolaire, l'ATL adresse ainsi le premier problème. Il doit par ailleurs être doté d'une capacité à choisir les solutions à instancier en fonction de l'environnement réseau — qui peut par exemple bloquer les flux non TCP via des middlebox configurées à cet effet — pour répondre au problème d'acceptabilité par le réseau.
- Le déploiement de L'ATL pose également un problème en lui-même dans la mesure où l'on ne doit pas imposer son déploiement sur toutes les machines, ni son utilisation par une application initialement conçues sur les bases de la couche Transport classique. La conception de l'ATL doit donc répondre à cette double exigence.

### 2.1.2 Principes des solutions proposées en réponse aux exigences de conception de l'ATL

Afin de répondre aux différentes exigences précédentes, l'ATL a été conçu suivant certains principes que nous présentons ici. La mise en œuvre de ces principes est elle-même basée sur un ensemble de paradigmes de conception décrits dans la section 2.1.3.

**Premier principe : l'abstraction.** Le principe d'abstraction est le principe clé de la solution que propose l'ATL. En effet, tout réside dans l'idée de séparer les détails d'implémentation et de gestion des solutions protocolaires, des applications qui vont en utiliser les services.

Suivant ce principe, on limite les interactions entre les applications et la couche Transport à un ensemble d'interfaces génériques, ce qui a comme conséquence :

- de libérer les développeurs d'applications du choix et de la difficulté d'utilisation des solutions protocolaires sous-jacentes, et répond donc au problème de la complexité protocolaire ;
- de passer, du point de vue de l'application, d'un paradigme d'utilisation de protocoles à un paradigme d'utilisation de services, levant ainsi la dépendance des applications aux solutions protocolaires en leur permettant de pouvoir s'exécuter « au-dessus » de n'importe quel protocole offrant le service souhaité ;

- d'effectuer des modifications sur les protocoles et les mécanismes protocolaires sans conséquence sur les applications, répondant ainsi — en tout ou partie — à l'objectif d'acceptabilité par les applications.

**Second principe : l'extensibilité.** Le principe d'extensibilité réside dans la capacité de l'ATL à intégrer de nouvelles solutions protocolaires, ou à mettre à jour les solutions existantes sans aucune implication supplémentaire sur les applications ou les systèmes.

Le principe d'abstraction de l'ATL répond aux besoins des applications, en ceci qu'il n'y aura aucune implication sur ces dernières si une modification se porte sur une solution.

Concernant le besoin des développeurs de systèmes d'exploitation, l'architecture de l'ATL doit être telle que l'intégration de nouveaux composants impose uniquement des mises à jour de la base de connaissance de l'ATL, sans mise à jour des composants de gestion de l'ATL.

Enfin, ces modifications, notamment celles conduisant à offrir de nouveaux services de Transport, doivent être communiquées de façon publique aux développeurs d'applications. Les mises à jour doivent ainsi être des opérations simples qui seront assurées par une entité de l'ATL que le développeur du protocole aura à invoquer pour que sa solution protocolaire soit prise en compte, en lui transmettant un certain nombre d'informations décrivant sa nouvelle solution.

**Troisième principe : la configurabilité.** La configurabilité est un principe d'optimisation qui permet d'offrir à l'application l'ensemble de services dont elle a besoin et uniquement ceux-ci — c'est à dire sans service supplémentaire. Ce principe va à l'encontre du caractère monolithique de la plupart des protocoles de Transport existants qui impose aux applications tous les services pré-implantés.

Cette configurabilité peut être assurée par l'ATL ou par le composant protocolaire lui-même. Dans le premier cas, l'ATL doit sélectionner et composer un ensemble optimal de composants qui offriront le service demandé par l'application. Dans le deuxième cas, l'ATL doit offrir, pour les composants qui sont à la base configurables — tels que ETP — un environnement dans lequel ils pourront conserver leur caractère configurable.

**Quatrième principe : l'autonomie.** A l'instar de la configurabilité, l'autonomie est un principe d'optimisation qui permet à l'ATL de prendre des décisions en fonction des changements du contexte applicatif et réseau. La prise de décision est relative au choix de la composition « optimale » de protocoles/composants protocolaires en réponse à une requête de l'application, que ce soit au moment de

l'établissement de la connexion, ou durant la communication en cas de changement de contexte.

Un changement de contexte se produit lorsque les besoins de l'application changent, par exemple lorsque celle-ci passe d'un type de fonctionnalité à un autre ou d'un type de donnée à un autre. Les changements de contexte peuvent aussi être liés au réseau : il peut s'agir d'une diminution ou d'une augmentation des paramètres réseau, tel que le débit, le délai, le taux de perte, ou encore l'apparition ou la disparition d'un problème de congestion ou le changement du type de réseau (Ethernet vers 3G par exemple). Dans toutes ces situations, aussi bien avant ou durant la communication, le principe d'autonomie est de permettre une adaptation de la solution protocolaire transparente pour l'application.

### 2.1.3 Approches d'implantation de ces principes

La mise en œuvre des principes précédents repose sur un certain nombre de paradigmes de conception qui sont présentés dans cette section.

#### 2.1.3.1 Approche orientée services

L'approche orientée services vise principalement à séparer les services offerts de leur implémentation interne en réduisant les interactions entre l'application et ces derniers à une simple interface. Cette façon de faire offre une abstraction complète des solutions techniques utilisées, et constitue ainsi l'approche la plus importante de notre solution.

L'objectif est de définir chaque protocole ou composant protocolaire par l'ensemble des services qu'il offre. Du point de vue de l'application, il n'y aura plus de notion de protocole de Transport, mais seulement de service de Transport, au sens où elle n'invoque plus un protocole particulier, tel que TCP ou UDP, mais un service particulier, par exemple un service orienté-connexion en mode flux d'octets avec garantie d'ordre et de fiabilité totale. Le choix du protocole et du ou des composants protocolaires qui assureront le service requis ne sera plus à la charge de l'application.

Le principe de mise en œuvre repose sur la capacité de l'ATL à maintenir une liste des correspondances entre protocole ou composant protocolaire et service(s) correspondant(s) et, quand il reçoit une requête d'une application contenant une liste de services souhaités, à chercher dans la liste des correspondances la composition protocolaire la plus adaptée pour assurer l'ensemble des services.

De cette façon, l'approche orientée services assure parfaitement le principe d'abstraction de l'ATL, et permet de répondre à plusieurs parties de la problématique :



- en réduisant l'interface avec les applications à une liste de services accessibles, on élimine le problème de complexité relatif :
  - au choix du protocole de Transport, car celui-ci n'est plus de l'ordre des applications mais de l'ATL,
  - à l'usage de celui-ci car il n'y aura plus besoin de connaître les détails relatifs au fonctionnement de chaque protocole ou mécanisme protocolaire comme cela se fait actuellement, l'application ayant simplement à connaître l'ensemble des services offerts et à solliciter ceux qui lui conviennent ;
- la séparation des détails techniques permet, par le biais des services, de lever la dépendance classique entre applications et protocoles et ce, sur deux plans :
  - tout d'abord, les applications ne seront plus écrites pour un protocole particulier mais pour un ensemble de services et peuvent donc être exécutées « au-dessus » de n'importe quelle solution protocolaire offrant les mêmes services,
  - ensuite, les modifications apportées aux composants protocolaires n'affecteront pas le fonctionnement des applications, puisque ces modifications ne leur seront pas visibles. La réponse au problème de dépendance permet ainsi de répondre à une partie du problème de l'extensibilité concernant les applications. Ceci signifie qu'il sera possible, avec une approche orientée services, d'ajouter ou de mettre à jour n'importe quelle solution protocolaire au sein de l'ATL sans aucune conséquence sur les applications. Pour les autres parties du problème d'extensibilité (*i.e.* concernant les développeurs de systèmes ou de protocoles), la solution nécessitera l'adoption d'approches complémentaires, tel que préconisé par la suite ;
- Enfin, pour les protocoles qui sont à la base configurables (tels que ETP), l'approche orientée service permet de conserver leur configurabilité avec la possibilité de paramétrage des services. Chaque service peut disposer d'une liste de paramètres que l'on peut ou doit lui fournir à son invocation. Ceci concerne la configurabilité des services eux-mêmes, et pour ce qui est de la configurabilité de l'ensemble des services, c'est à dire le fait de choisir un sous ensemble des services offerts, ceci va de soit, puisque dans ce paradigme, il n'y a aucune différence entre les services offerts séparément par des composants indépendants et les services offerts séparément par un même composant.

**Conséquences en termes de besoins fonctionnels d'une approche orientée services.** L'adoption d'une approche orientée services impose la mise en œuvre, au sein de la couche Transport, de fonctions supplémentaires relatives à la gestion des services. Ces fonctions concernent notamment :

- l'intégration des nouveaux services incluant leur prise en compte par les

- composants de gestion de l'ATL ;
- l'association entre services et composants afin de facilement trouver quels sont les services fournis par chaque composant ;
- la vérification de leurs disponibilité sur les hôtes locaux et distants.

Ces différentes fonctions sont regroupées dans la fonctionnalité de gestion des services et seront déléguées à différents composants de l'architecture de l'ATL. Elles sont décrites plus en détails en section 2.1.4.1 de ce chapitre.

### 2.1.3.2 Approche basée composants

L'approche basée composants concerne l'organisation interne des solutions protocolaires qui seront gérées par l'ATL. Le principe est de mettre chaque protocole ou composant protocolaire dans un composant logiciel indépendant offrant un ensemble de services via une interface propre.

En séparant chaque protocole ou composant protocolaire en un composant logiciel indépendant, on facilite son intégration dans l'ATL. L'ajout à la bibliothèque de composants et la mise à jour de la base de connaissances de cette dernière constituent ainsi les seules étapes de l'installation d'un nouveau composant :

- aucune partie du système d'exploitation ni de l'architecture de l'ATL ne sera à modifier ; cette souplesse assure une extensibilité complète de la couche Transport, et répond ainsi aux limitations d'extensibilité de la couche actuelle du point de vue des développeurs de système et de protocole ;
- en conséquence, le développeur de protocole n'aura plus à se préoccuper de la prise en charge de son protocole par le système ;
- en parallèle, en associant chaque composant à un ensemble de services, les mises à jour seront transparentes pour les applications ; de cette façon, nous répondons à l'ensemble des problèmes relatifs à l'extensibilité.

Par ailleurs, la séparation des composants protocolaires permet d'offrir la possibilité de choisir et de composer au besoin les composants qui assureront un service donné, au sens où les applications ne seront plus obligées de fonctionner avec des services insuffisamment adéquats, comme c'est le cas avec l'architecture actuelle de la plupart des protocoles de Transport à caractère monolithique. Le principe de configurabilité est ainsi assuré.

Enfin, cette approche permet la création de solutions protocolaires à granularité variable. La possibilité d'exprimer la dépendance d'un composant envers un autre ou envers un service assuré par un autre est capitale. Ainsi, un développeur souhaitant créer un mécanisme de contrôle des pertes et nécessitant un estampillage des paquets par un numéro de séquence pourra soit développer son propre estampillage, soit déclarer son composant protocolaire dépendant de l'utilisation d'un composant d'estampillage adéquat.

**Conséquences en termes de besoins fonctionnels d'une approche basée composants** Les conséquences de l'approche basée composant concernent principalement leurs intégrations au sein de l'ATL. Les interfaces de ces composants doivent être définies de façon suffisamment générique pour qu'ils puissent être intégrés de façon transparente dans l'ATL, c'est-à-dire sans avoir à récrire le code de l'ATL. Enfin, celle-ci doit pouvoir les utiliser de façon séparée selon les besoins des applications. De plus, l'ATL doit les invoquer de façon indépendante et gérer les liens entre eux tout au long de la communication.

### 2.1.3.3 Approche autonome

L'autonomie de l'ATL vise à rendre transparent pour l'utilisateur humain les prises de décision relatives au choix des solutions protocolaires, à l'initiation de la communication ou en réponse à un changement du contexte du réseau. Ceci impose donc la capacité de l'ATL à trouver des compositions optimales pour satisfaire les applications en tenant compte de leurs besoins et de l'état du réseau. Dans une approche autonome, cette prise en compte ne doit pas se limiter au moment de l'ouverture de connexion, mais doit être poursuivie tout au long de la communication pour s'adapter dynamiquement aux changements du contexte applicatif ou réseau, c'est-à-dire au changement des besoins applicatifs ou de l'état du réseau.

**Conséquences en termes de besoins fonctionnels d'une approche autonome** En conséquence, l'ATL doit, d'une part, effectuer une surveillance permanente du réseau afin d'en connaître l'état en temps réel, et éventuellement détecter tout changement conséquent. Par la suite l'ATL doit pouvoir trouver la composition protocolaire optimale pour le contexte courant (état du réseau et besoins des applications) à l'ouverture de la communication aussi bien qu'au cours de celle-ci. D'autre part, elle doit offrir aux applications la possibilité de pouvoir modifier leurs requêtes (services requis) durant la communication et ce sans interruption.

Ces différentes fonctions sont regroupées au sein du composant gestionnaire de l'autonomie, l'*Autonomic Manager*, introduit en section 2.2.3.2 dans ce chapitre.

### 2.1.3.4 Synthèse

Le tableau 2.1 fournit une vue synthétique des sections précédentes.

## 2. ARCHITECTURE DE L'ATL

Problèmes	Exigences de conception de l'ATL	Principes de conception de l'ATL	Approche logicielle d'implantation
Complexité	Masquer aux développeurs d'applications la connaissance et la complexité du choix des solutions protocolaires à invoquer. Masquer aux développeurs d'applications la complexité d'utilisation des solutions protocolaires qui seront invoquées.	Abstraction Sémantique	Approche basée sémantique
Dépendance	Permettre une adaptation de la solution protocolaire retenue en réponse à une évolution du contexte machine/réseau, ou bien à une modification de l'expression du service requis par l'application, c'est-à-dire : <ul style="list-style-type: none"> <li>– être doté d'une capacité d'autonomie dans la prise de connaissance du contexte et de ces évolutions, en plus de celle liée au choix et à la mise en œuvre des solutions protocolaires à invoquer ;</li> <li>– être doté d'une capacité d'autonomie dans le choix et la mise en œuvre des adaptations de la solution protocolaire en cours ;</li> <li>– offrir aux développeurs d'applications la possibilité de modifier les caractéristiques du service requis durant le temps d'exécution de l'application.</li> </ul>	Abstraction Autonomie	Approche orientée services  Approche autonome
Manque d'extensibilité	Permettre son intégration au sein de son pool de services et de solutions protocolaires, en vue de la mise à disposition du service correspondant et de l'utilisation de la solution protocolaire.	Extensibilité	Approche basée composants
Manque de configurabilité	Gérer la nature composable des différents protocoles et mécanismes protocolaires, chaque protocole pouvant se voir adjoindre des mécanismes externes.	Configurabilité	

TABLE 2.1: Synthèse des problèmes, exigences, principes et approches de conception de l'ATL

### 2.1.4 Fonctionnalités de l'ATL

Cette section décrit l'ensemble des nouvelles fonctionnalités nécessaires pour mettre en œuvre les principes décrits précédemment. Ces fonctionnalités résultent directement des approches de conception de l'ATL, et sont introduites précédemment pour chacune des approches. Il s'agit en d'autres termes de toutes les fonctionnalités qui ne figurent pas dans l'approche traditionnelle de la couche et des protocoles de Transport, et qui permettent d'assurer la valeur ajoutée apportée par la nouvelle architecture de l'ATL. Ceci concerne principalement la gestion des services et l'autonomie.

#### 2.1.4.1 Fonction de gestion des services

La gestion des services est une fonctionnalité qui résulte de l'approche orientée-services permettant de réduire la complexité de l'interface de l'ATL et d'assurer son extensibilité. Elle regroupe les tâches relatives à l'intégration de nouveaux composants protocolaires dans l'ATL et la correspondance entre ceux-ci et les services qu'ils offrent.

La première tâche sera donc la constitution et le maintien de la liste des services au fur et à mesure que de nouveaux composants sont ajoutés à l'ATL. Ce rôle incombe à l'intégrateur de services. L'information sur la correspondance entre composants protocolaires et services offerts est ainsi intégrée dans la base de correspondance des composants et des services afin que les applications puissent les invoquer et que l'ATL puisse utiliser le nouveau composant lors des prochaines requêtes applicatives si nécessaire.

Une fois les services stockés, les applications peuvent désormais les invoquer auprès de l'ATL. Pour ce faire, la base de correspondance des composants et des services permet sa consultation afin de retrouver les différents composants pouvant, seuls ou au sein d'un ensemble composite, assurer le service requis.

#### 2.1.4.2 Fonction de gestion autonome de la communication

Cette fonctionnalité assure le caractère autonome de l'ATL qui vise à décider du service composite initial à mettre en place et à adapter le comportement de celui-ci de façon autonome selon les variations du contexte réseau (changement du type du réseau ou de son état). Elle comporte quatre tâches correspondant aux quatre parties de la boucle autonome décrite un peu plus loin.

La première tâche consiste à recueillir les informations utiles sur l'état du réseau. L'ATL doit donc disposer d'un service de monitoring du réseau, en souscrivant par exemple à un service externe ou en implantant le sien propre. Une fois ces informations recueillies (débit, délai, taux de perte ...), l'ATL doit les

analyser afin de détecter les variations et subséquemment le besoin éventuel d'un changement de la composition protocolaire courante.

En cas de besoin d'un changement de composition, l'ATL doit revoir les composants sélectionnés pour chaque service et l'affiner, éventuellement une autre fois, afin de trouver pour chaque service le composant le plus adapté à l'état courant du réseau, en tenant compte de l'ensemble des paramètres de ces services pour lesquelles des information de surveillance ont été recueillies.

### 2.1.4.3 Fonction d'accueil, d'assemblage et de mise en œuvre des compositions

L'ATL doit également permettre l'accueil des composants protocolaires utilisés dans la mise en place des différents services composite. Ainsi, si plusieurs implémentations peuvent être possibles, l'architecture doit prendre en compte le besoin d'une structure où prendra place cette composition et permettant son fonctionnement — en assurant notamment l'ordonnancement des messages à travers les différents composants, sa configuration, sa modification voire sa suppression. Les différents composants assurant ces fonctionnalités seront décrits en 2.7.

## 2.2 Architecture de l'ATL

Cette section présente le modèle architectural de l'ATL ainsi que ses différents composants. Suivant une approche basée sur le formalisme UML 2.0, nous décrivons d'abord les cas d'utilisation de l'ATL (section 2.2.1). Nous présentons ensuite le modèle d'architecture de l'ATL et de ses principaux composants (section 2.2.3). Nous décrivons enfin les composants secondaires de l'ATL (section 2.2.4).

### 2.2.1 Cas d'utilisation

L'architecture d'une solution telle que l'ATL doit être guidée par des cas d'utilisation faisant intervenir les différents acteurs du système. Ces acteurs peuvent être les utilisateurs du système ou d'autres éléments interagissant avec le système lors de son utilisation. Nous avons déjà évoqué les différents types d'applications utilisant l'ATL et de systèmes l'hébergeant. Ceux-ci donnent lieu aux cas d'utilisation de la figure 2.1.

On peut voir sur ce diagramme que les applications représentent des acteurs primaires, placés à gauche du système, indiquant que ces acteurs sont les initiateurs des cas d'utilisations auxquels ils sont reliés. Le réseau, en revanche est un acteur secondaire. Il intervient dans la réalisation des cas d'utilisation auxquels il est relié, mais n'initie pas ces derniers.

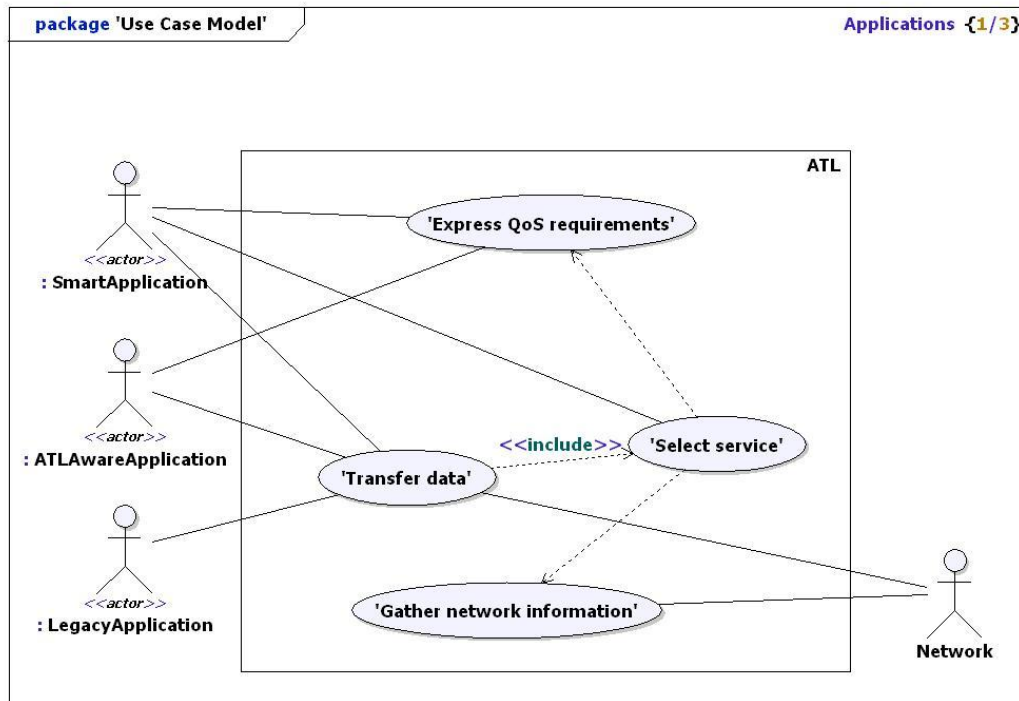


FIGURE 2.1: Cas d'utilisation impliquant les applications

Ce diagramme illustre également les prérogatives respectives des différents types d'applications. On y voit que les applications *legacy* ne peuvent qu'initier le cas d'utilisation « Transfer data » quand les applications *aware* et *smart* peuvent également exprimer les services à utiliser. Les applications *smart* peuvent également récupérer la responsabilité des choix normalement effectués par l'ATL dans la composition de composants protocolaires qui sera instanciée, via le cas d'utilisation « Select service ».

Au delà des applications, l'ATL va également devoir offrir certains services aux utilisateurs humains. Ceux-ci peuvent être de plusieurs types et présentent des besoins différents, il s'agit : de l'utilisateur de la machine (ou utilisateur du système), de l'administrateur de la machine (ou administrateur du système) et du développeur de mécanismes protocolaires. Nous pouvons les voir sur le diagramme de cas d'utilisation de la figure 2.2.

L'utilisateur du système doit pouvoir exprimer certaines préférences quant à l'utilisation des ressources disponibles sur sa machine. Ainsi la possibilité de modifier les politiques de prise de décision en fonction des différentes applications doit lui être offerte. Par exemple, l'utilisateur doit pouvoir exprimer qu'il veut que le

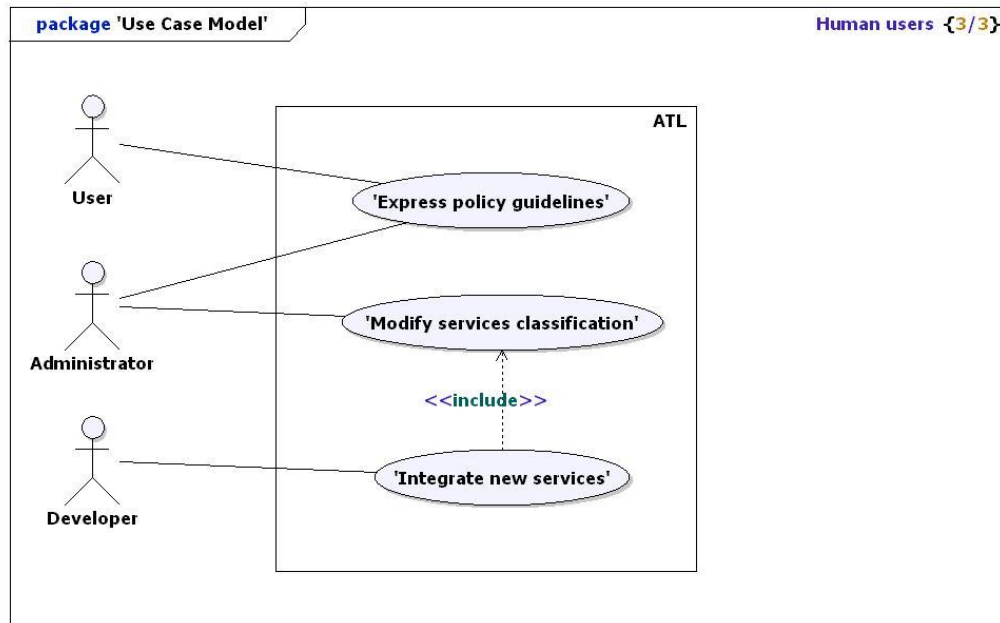


FIGURE 2.2: Cas d'utilisation impliquant les utilisateurs humains

débit d'émission de son gestionnaire de téléchargement BitTorrent soit limité, afin de laisser une capacité plus importante pour le serveur web qui héberge son blog.

L'administrateur du système doit également avoir la possibilité d'exercer ce genre de contrôle. Les préférences qu'il exprime doivent de plus avoir préséance sur celles exprimées par les utilisateurs du système. De plus, l'administrateur doit pouvoir aller modifier le comportement de l'ATL en influant sur les solutions que celle-ci peut instancier pour satisfaire les services requis par les applications. Un administrateur doit pouvoir interdire un protocole, ou en désactiver certaines options, modifier la connaissance qu'a l'ATL de ceux-ci afin que ses décisions futures soient conformes aux politiques de son entreprise. Par exemple, il doit pouvoir imposer d'utiliser un chiffrement via SSL au lieu de TLS, ou d'utiliser un contrôle de congestion basé sur TFRC et non sur TCP.

Finalement, le dernier utilisateur humain est le développeur de composants protocolaires à intégrer à l'ATL. L'ATL doit en effet offrir les moyens pour pouvoir développer de nouveaux protocoles, de nouvelles implémentations pour tel ou tel mécanisme, et permettre leur intégration de manière transparente pour les autres acteurs.

Les acteurs secondaires ont également un rôle important à jouer dans ces cas d'utilisation. Ceux-ci sont de deux types. Tout d'abord, les systèmes avec lesquels



on souhaite communiquer. Comme nous l'avons indiqué, ces systèmes peuvent être de deux types :

**les systèmes *legacy*** qui ne disposent pas de l'ATL ;

**les systèmes *ATL-enabled* ou *enabled*** qui en disposent.

Il est capital qu'un système *enabled* soit capable d'initier une communication ou d'accepter une communication entrante avec un autre système *enabled*, bien évidemment, mais également avec un système *legacy*. Sans cette condition, l'adoption de l'ATL serait compromise car cela scinderait le parc informatique mondial en deux parties non interopérables. Ainsi, afin d'adapter son comportement au type de système avec lequel elle souhaite communiquer, l'ATL doit posséder un système de découverte de la présence de l'ATL sur le système distant.

Le réseau constitue l'autre acteur secondaire, indispensable puisque c'est par lui que vont transiter les messages échangés entre les différentes instances de l'ATL. Comme indiqué dans le premier chapitre, celui-ci fait preuve aujourd'hui d'une grande hétérogénéité et d'une grande dynamique au niveau de ses caractéristiques et de leurs variations au cours du temps. Les multiples technologies d'accès et la présence de différents types de middleboxes créent une variété de scénarios possibles. Le rôle de l'ATL sera de s'adapter à ces différents scénarios de manière à toujours satisfaire au mieux l'application, en fonction des préférences des utilisateurs et administrateurs, et des solutions mises à disposition par les développeurs, en tenant compte de la présence ou de l'absence d'une instance de l'ATL sur le système distant avec lequel s'établit la communication.

Pour tenir compte des variations dans le réseau, l'ATL va devoir en permanence surveiller ses performances afin de prédire ou détecter les changements qui seront susceptibles d'induire une modification de la solution choisie afin de maintenir un service acceptable.

### 2.2.2 Modèle d'architecture de l'ATL

En observant les protocoles classiques et l'architecture actuelle de l'Internet, les différents types de fonctionnalités intervenant dans une communication pour acheminer les données, contrôler cet acheminement, et gérer la communication, peuvent être répartis selon les trois plans suivants.

- Le plan de données gère l'acheminement à proprement parler des données de l'utilisateur. C'est lui qui s'occupe de l'encapsulation des messages, ou du multiplexage vers les différentes applications par exemple. Les protocoles permettant l'encapsulation de données, tels que IP au niveau réseau de la pile protocolaire, sont des protocoles du plan de données. Ils offrent un service d'acheminement à un niveau supérieur le requérant. Le plan de données de l'ATL remplit ce même rôle.

- Les plans de contrôle et de gestion, qui assurent la prise en charge des deux autres types de fonctions. Au sein de l'ATL, le contrôle et la gestion servent tous deux à gérer la communication, mais à des échelles de temps différentes : les actions de contrôle sont des actions à portée temporelle réduite ou courte, à l'échelle du paquet ou de quelques RTT, comme la vérification de l'intégrité d'un paquet, ou la détection de pertes ; les actions de gestion portent sur des échelles de temps plus grandes, comme les modifications de tables de routage, ou des règles de sécurité.

Nous pouvons remarquer que dans les protocoles existants, les actions de contrôle sont souvent prises en charge par le protocole du plan de données. TCP ou Ethernet vérifient par exemple eux-mêmes si les unités de données qu'ils reçoivent sont erronées ou non. A l'inverse, les actions de gestion sont généralement effectuées par des protocoles dédiés, comme les protocoles de routage, dont RIP, OSPF, ou BGP sont les représentants les plus connus ; d'autres protocoles connus sont également des protocoles prenant en charge des actions de gestion : ICMP, ARP, SNMP, etc.

Notons que ces protocoles sont souvent indifféremment qualifiés de protocoles du plan de contrôle ou de protocole du plan de gestion. Le but ici n'est pas de savoir lequel des deux est juste afin de rétablir une classification claire des protocoles selon ces trois plans. Nous utilisons ce vocabulaire afin de distinguer non pas différents types de protocoles, mais différents types d'actions, et remarquons alors que les actions de transfert de données et de contrôle sont souvent gérées par un même protocole, quand les actions de gestion sont souvent gérées par un protocole dédié.

Nous effectuons la même dichotomie au sein de l'ATL où une partie de l'architecture sera dédiée à l'exécution de la communication, donc au transfert de données et au contrôle de celui-ci, et où une seconde partie sera dédiée à la gestion de la communication, grâce à une relation constante avec la première et qui permettra d'influer sur celle-ci.

Articulée autour de ces trois types de fonctions, l'architecture de l'ATL repose sur l'identification des besoins des utilisateurs exprimés via les cas d'utilisation précédents. De ces cas d'utilisations et des considérations exposées dans la section précédente, nous détaillons un premier niveau d'architecture composite de l'ATL (section 2.2.2.1). Les détails des composants de base de premier niveau sont présentés en section 2.2.3.

### 2.2.2.1 Modèle général

Pour offrir les différents services présentés ci-dessus, l'architecture de l'ATL s'organise en plusieurs niveaux de composants, représentés sur la figure 2.3.

On observe sur ce diagramme différents composants nécessaires au bon déroulement de la prise en charge des communications. Pour la gestion de la commu-

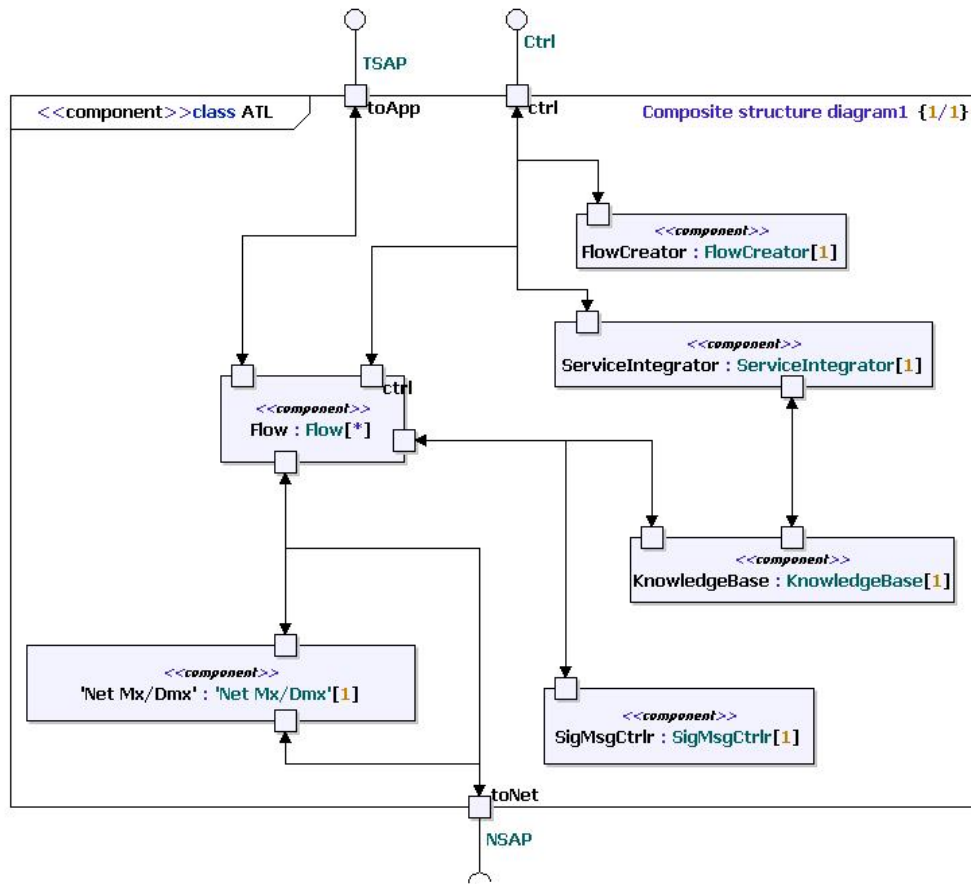


FIGURE 2.3: Diagramme de structure composite de premier niveau

nication proprement dite, le composant plus important est le *Flow*, qui contient deux composants principaux :

- l'*Autonomic Manager* qui s'occupe de la gestion autonome de la communication avant et après l'établissement de la connexion ;
- le *Data Plan* qui s'occupe de la communication proprement dite, en accueillant l'ensemble des composants protocolaires pour la communication, dénommée également « service composite », destinée à une communication particulière.

On remarque que le *Flow* dispose de deux interfaces vers l'application : la première sert à l'échange de données entre l'application et le *Flow* ; la seconde est l'interface de contrôle, permettant à l'application d'exprimer ses besoins ainsi que ses contraintes concernant les décisions de l'ATL dans le cas d'une application *smart*.

Chaque *Flow* est associé à un et un seul point d'accès au service de Transport.

Le nombre de *Flow* présents au sein du système est donc variable avec le temps, et peut également être nul si aucune application n'a ouvert de communication. Chaque *Flow* n'appartient qu'à une et une seule application, mais une application peut posséder plusieurs *Flow* si elle ouvre plusieurs points d'accès. Un serveur web, par exemple, aura un *Flow* ouvert par client visitant le site qu'il héberge. Le *Flow* est décrit plus avant en section 2.2.3.1.

Avant que ne débute effectivement une communication, l'*Autonomic Manager* d'un *Flow* utilise la base des composants et des services pour décider de la composition adéquate selon les besoins de l'application et l'état du réseau. Cette base, commune à tous les *Flow* du système, recense tous les composants protocolaires disponibles et les services qui leur sont associés et permet aux *Autonomic Manager* de choisir parmi cet ensemble le service composite le mieux adapté.

L'intégrateur de services permet l'installation de nouveaux composants au sein de l'ATL, notamment via la mise à jour de la base des composants et services. Notons que celle-ci peut également être modifiée par un utilisateur humain tel qu'un administrateur système ou par un logiciel de gestion hiérarchiquement supérieur.

Le rôle du *Flow Creator* est l'instanciation des *Flow* lors de la création d'un TSAP. Le *Signalization Message Controler* est le composant responsable de la signalisation relative aux informations globales. Finalement, le multiplexeur-démultiplexeur réseau (Net Mx/Dmx) constitue l'interface entre les *Flow* et le réseau. Ces composants sont décrits dans la section 2.2.4 relative aux composants secondaires.

On remarque que l'ATL dispose, en plus des interfaces avec l'application et de celle avec la couche réseau sous-jacente, d'une interface d'administration destinée aux utilisateurs humains, afin de leur permettre d'exprimer des préférences générales comme l'indique le diagramme de cas d'utilisation de la figure 2.2. C'est à travers celle-ci que les commandes des utilisateurs et des administrateurs sont intégrées dans l'ATL. C'est également grâce à elle que des programmes extérieurs peuvent venir administrer l'ATL le cas échéant.

### 2.2.3 Composants de base de l'ATL

Cette section est consacrée à la description du composant principal de l'ATL : le *Flow* et ses sous-composants. Ce sont eux qui vont accueillir et gérer la composition de composants protocolaires instanciant la communication.

#### 2.2.3.1 Le *Flow*

Le composant principal de l'ATL est le *Flow*. Un *Flow* est mis à disposition d'une application pour chaque point d'accès qu'elle ouvre.

La décomposition du *Flow*, illustrée sur la figure 2.4, fait apparaître deux composants principaux : le *Data Plan* et l'*Autonomic Manager*, dont les fonctions

permettent de distinguer la prise en charge des actions de type transfert de données et contrôle pris en charge par les composants protocolaires (*Data Plan*), et les actions de gestion du service composite (*Autonomic Manager*).

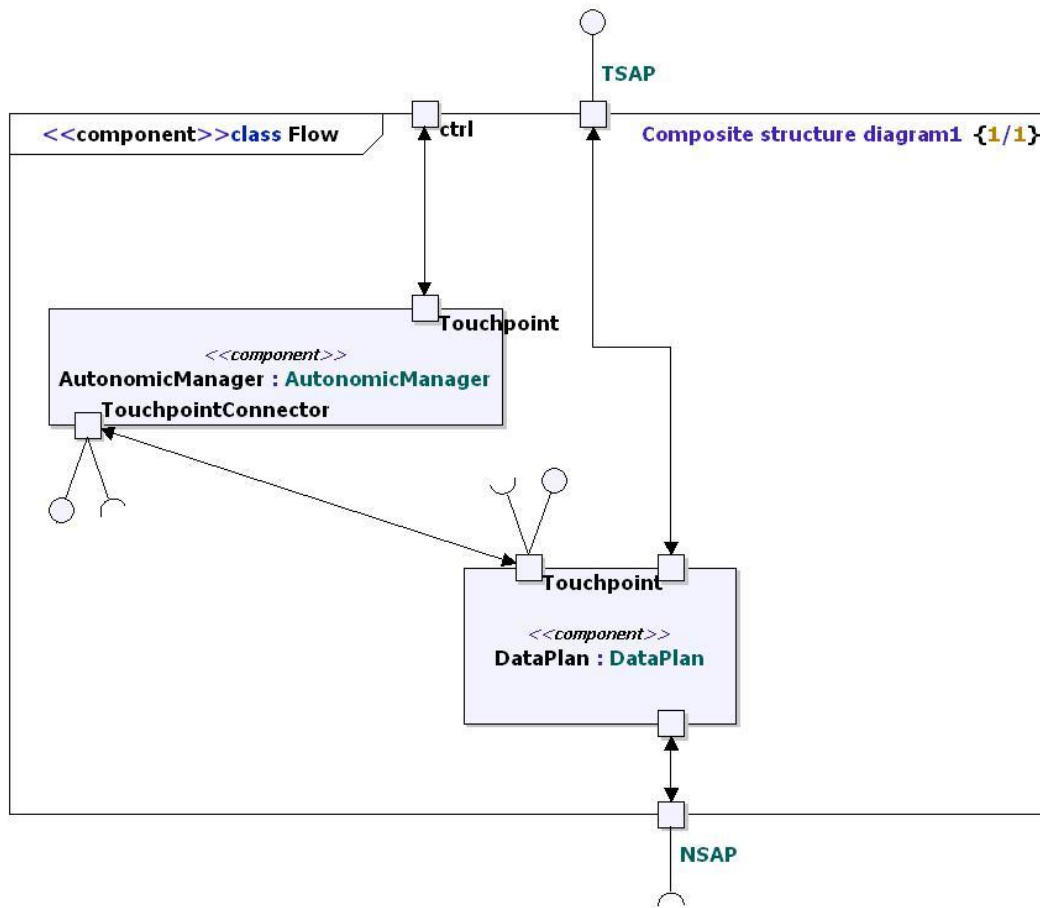


FIGURE 2.4: Composition d'un *Flow*

Le *Flow* dispose de plusieurs interfaces :

- la première le relie à l'application et est composée d'une partie données, permettant l'échange de ces dernières, et d'une partie contrôle, comme introduit en section 2.2.2.1 :
  - la partie données est redirigée vers le *Data Plan*,
  - la partie contrôle est redirigée vers l'*Autonomic Manager*, afin de lui communiquer les différentes informations sur les services désirés et les différentes contraintes que l'application souhaite exprimer ;
- la deuxième interface relie le *Flow* à la couche réseau par le biais du *Data Plan* afin de permettre l'échange de données utilisateurs encapsulées ou à

destination de l'application. Celle-ci est reliée directement à la couche réseau sous-jacente en sens descendant et à un démultiplexeur dans le sens montant (cf. 2.2.4.3).

La liaison entre l'*Autonomic Manager* et le *Data Plan* est décrite dans la section 2.2.3.2.

Notons enfin que l'*Autonomic Manager* est également relié au *Flow Creator*, décrit en 2.2.4.1, et au *Signalization Message Controller*, décrit en 2.2.4.2.

### 2.2.3.2 L'*Autonomic Manager*

L'*Autonomic Manager* est un des composants internes majeur d'un *Flow*. Avant d'en décrire l'architecture fonctionnelle dans le cadre de l'ATL, nous introduisons tout d'abord ce concept tel que défini par IBM.

**État de l'art relatif au concept d'*Autonomic Manager* tel que défini par IBM** IBM définit l'*Autonomic Manager* comme le composant permettant la gestion autonome d'une ressource. Dans le cas de l'ATL, la ressource s'avère être un service composite, une composition de protocoles et mécanismes protocolaires destinés à satisfaire les besoins de l'application.

L'*Autonomic Manager* est composé d'une boucle de contrôle constituée de quatre fonctions, et d'une base de connaissance à laquelle ces quatre fonctions peuvent accéder. Ces quatre fonctions sont les suivantes.

**Le monitoring :** Le but de cette fonction est la surveillance des informations d'état du système et leur pré-analyse afin de détecter des symptômes d'un comportement potentiellement anormal ou qui dériverait de l'état souhaité.

**L'analyse :** La fonction d'analyse est invoquée par la fonction de monitoring lorsque cette dernière détecte un symptôme. Le symptôme est utilisé afin de modéliser l'état actuel du système qui est alors comparé à l'état souhaité. Si les deux modèles divergent, une alarme est alors levée pour avertir la fonction suivante.

**La planification :** Lorsque la fonction d'analyse lève une alarme, la fonction de planification prend en charge la décision des actions à entreprendre pour modifier le système afin de refaire converger son état vers l'état souhaité. En résulte un plan de modification, discuté en 2.2.3.2, qui est communiqué à la dernière fonction de la boucle.

**L'exécution :** Cette dernière fonction prend en charge les modifications décidées par la fonction de planification. La réalisation effective, l'orchestration et la synchronisation des différentes étapes du plan de modification sont à la charge de cette fonction.

Ces différentes fonctions s'organisent autour de la base de connaissances comme le montre la figure 2.5. Cette base de connaissance sera décrite plus loin en 2.2.3.4.

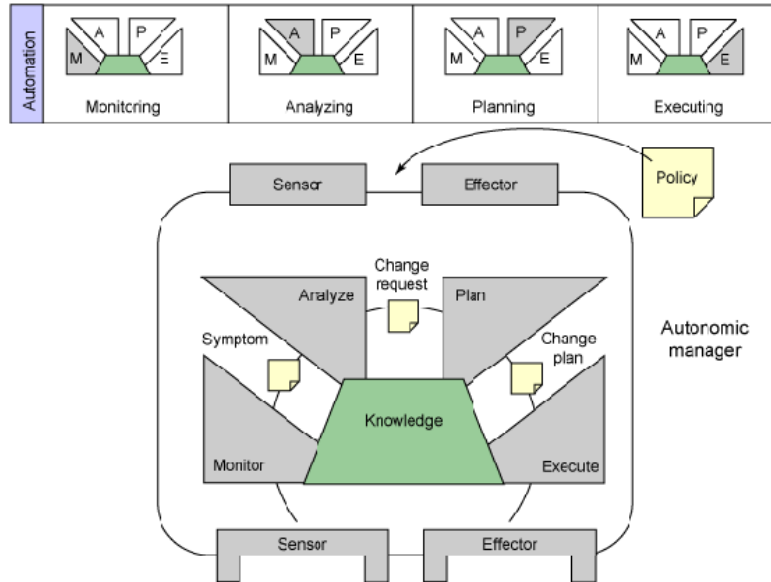


FIGURE 2.5: Organisation des fonctions de l'*Autonomic Manager*

La communication entre la ressource gérée et l'*Autonomic Manager* est assurée par une interface nommée *Touchpoint* et capable d'un comportement de type capteur (*sensor*) afin de récupérer des informations sur la ressource, et de type effecteur (*effector*) afin d'appliquer des modifications sur la ressource. La figure 2.6 représente les différentes interactions possibles entre l'*Autonomic Manager* et le *Touchpoint* dans les deux types de comportement.

**Application au sein de l'ATL** Au sein d'un *Flow*, la liaison entre l'*Autonomic Manager* et le *Data Plan* — qui représente la ressource gérée — est illustrée figure 2.4. Cette liaison permet à la fonction de monitoring de l'*Autonomic Manager* de récupérer les informations fournies par les différents composants protocolaires, et à la fonction d'exécution d'aller modifier les paramètres de ces composants, ou de modifier la structure composite elle-même.

Les décisions prises par la fonction de planification de l'*Autonomic Manager* peuvent avoir une portée locale ou répartie.

**Portée locale :** les décisions de l'*Autonomic Manager* n'impactent que les composants de l'hôte sur lequel il se trouve. Celles-ci n'ont aucune influence sur le système distant ou la liaison entre les deux systèmes, et ne nécessitent

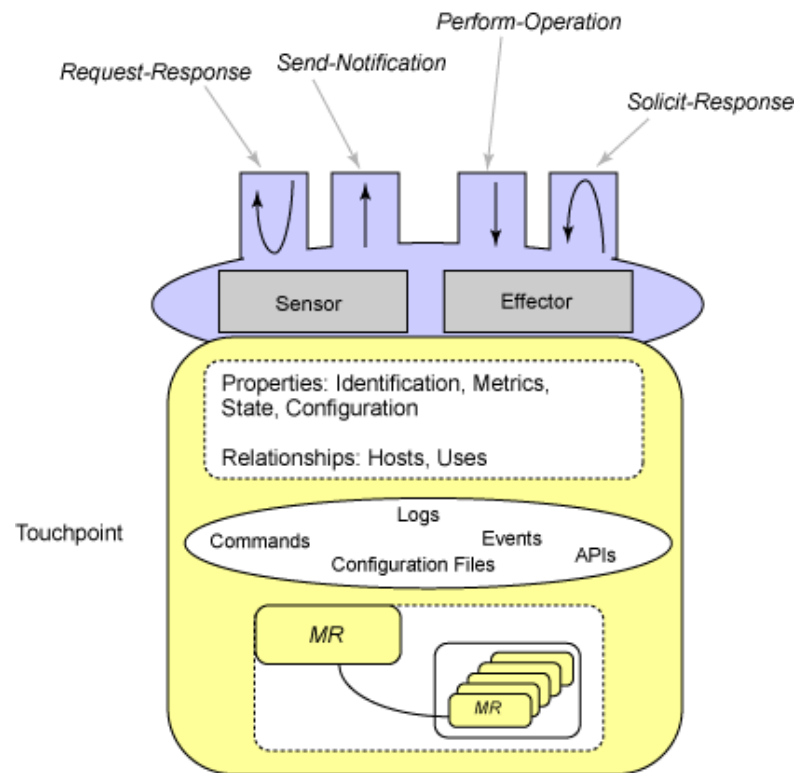


FIGURE 2.6: Caractéristiques du *Touchpoint* et lien avec l'*Autonomic Manager*

donc pas d'intervention de ce dernier. Informer l'hôte distant de ces décisions n'est a priori pas nécessaire mais peut cependant présenter un intérêt dans certains cas.

**Portée répartie :** les décisions de l'*Autonomic Manager* impactent les composants situés sur l'hôte distant ou la liaison entre les deux hôtes, et nécessitent une intervention de ce dernier. Une synchronisation des deux systèmes pouvant être nécessaire, la présence d'un protocole de signalisation la permettant est requise ; dans l'architecture de l'ATL, cette signalisation relève de la fonction d'exécution de la boucle de contrôle.

Sur ce second point, notons qu'une signalisation sera également nécessaire au niveau de la fonction de monitoring afin de requérir des informations de l'hôte distant ou de lui en fournir, ainsi qu'au niveau de la fonction de planification afin d'établir des prises de décisions en collaboration avec l'hôte distant.

Selon les cas, l'*Autonomic Manager* ne sera pas seul gestionnaire de la communication. En effet, si l'hôte distant possède également l'ATL, les deux *Autonomic Manager* (un par *Flow*) devront être en contact afin d'échanger certaines informations, comme précisé dans la partie Monitoring.



Pour la même raison, les prises de décisions doivent également pouvoir être partagées. Les deux *Autonomic Manager* vont avoir une double responsabilité.

- Tout d'abord, ils vont gérer individuellement les modifications locales de leur service composite respectif. En effet, la décision de modifier tel ou tel paramètre, d'enlever ou ajouter tel ou tel composant protocolaire, leur incombe tant que cette modification est purement locale. Par exemple, modifier à la volée l'algorithme de calcul du RTO dans le bloc de service TCP d'un certain *Channel* ne requiert aucune action de la part de l'*Autonomic Manager* pair.
- La seconde responsabilité est la prise de décision répartie. Il s'agit cette fois de gérer les modifications qui vont requérir des décisions sur chaque hôte. Les deux *Autonomic Manager* doivent donc être en communication, afin de se transmettre les informations de monitoring nécessaires ainsi que les consignes d'exécution.

Dans une prise de décision répartie les *Autonomic Manager* peuvent s'organiser de différentes manières. Nous identifions ici trois hiérarchies possibles : collaborative, participative et subordonnante.

**En mode collaboratif :** les deux *Autonomic Manager* ont un poids égal dans la prise de décision. Chaque décision ou compromis doit donc faire consensus afin d'être appliqué. Ce mode peut-être envisagé lorsque les deux flux applicatifs sont de même importance, comme dans le cas par exemple, d'une vidéoconférence en pair à pair, ou chaque participant envoie un flux vidéo et un flux audio, souvent du même ordre de qualité. Notons que le cas d'une vidéoconférence centralisée sur serveur est différent : le serveur jouant le rôle de point d'échange central des données celui-ci aura un rôle plus important.

**En mode participatif :** chaque *Autonomic Manager* peut proposer des modifications ou des actions sur la gestion de la communication, mais la décision finale revient à l'un des deux seulement. L'*Autonomic Manager* « participant » ne peut que faire des propositions qui devront être finalement approuvées par l'*Autonomic Manager* « décideur ». Dans ce mode, l'*Autonomic Manager* participant devient géré par l'*Autonomic Manager* décideur, sans pour autant perdre son intelligence.

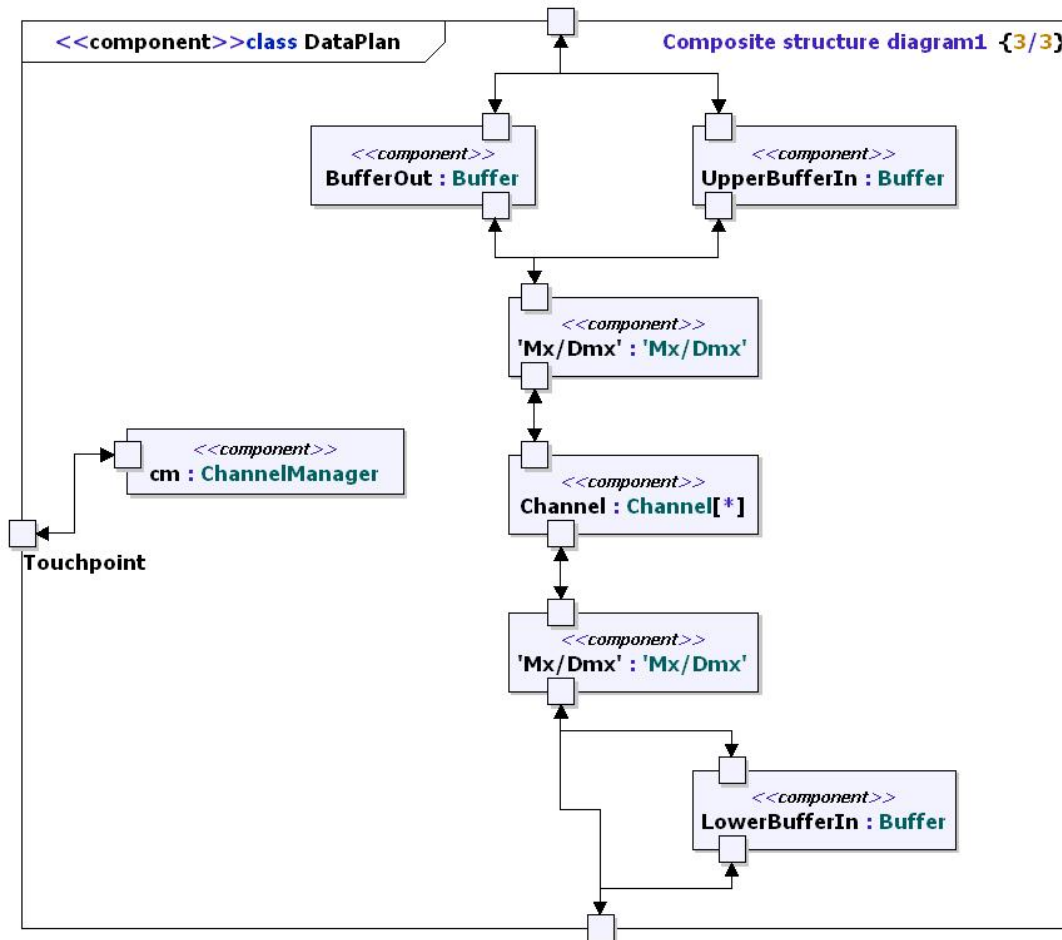
**En mode subordonnant :** un des *Autonomic Manager* se subordonne totalement à l'autre, c'est-à-dire que comme dans le cas précédent, il devient géré par l'*Autonomic Manager* pair à la différence cependant que l'*Autonomic Manager* subordonné abandonne toute intelligence concernant la gestion distribuée au profit de l'*Autonomic Manager* distant, ne devenant qu'un agent de monitoring et d'exécution de ce dernier. Ce mode peut être utile dans le cas où un des deux hôtes possède une très faible capacité de calcul (un capteur par exemple), ne pouvant ainsi pas prendre en charge plus que des décisions purement locales.

**Plan de modification** Une fois la décision prise sur les actions à entreprendre, le plan de modification contient la suite des actions à engager. Il décrit, point par point, quelles sont les modifications à faire, dans quel ordre, et si des conditions de transitions sont à respecter. La variété d'actions de reconfiguration possibles à entreprendre est a priori colossale, d'autant que celles-ci peuvent être spécifiques à certains composants protocolaires. Par exemple, la décision de désactiver l'algorithme de Nagel peut-être prise au sein de TCP ou de SCTP, mais n'a aucun sens dans le cas d'UDP. Si cet exemple représente une décision locale sans impact sur le pair distant de la communication, certaines reconfigurations nécessiteront par contre de synchroniser les deux *Flow*. Le déploiement ou le remplacement d'un ou plusieurs composants protocolaires par exemple, comme la transition de TCP vers DCCP, donne lieu à une évidente synchronisation. Le plan doit donc être organisé en étapes que chaque *Autonomic Manager* suivra à un rythme dépendant de leur organisation relative (cf. 2.2.3.2). Le plan mis à disposition de chacun peut différer. En effet, les actions à entreprendre ne sont pas forcément les mêmes sur chacun des deux *Flow*. Ainsi, chaque *Autonomic Manager* n'a besoin que de la connaissance de son propre plan. Ceux-ci acquièrent le dit plan par différents moyens selon la manière dont les décisions sont prises (cf. 2.2.3.2). En effet, si le mode retenu est la collaboration, chaque *Autonomic Manager* mettra en place son propre plan, via une négociation conjointe avec son pair. Dans le mode participatif ou subordonatif, l'*Autonomic Manager* maître est en charge de mettre le plan de chacun en place, puis le transmettre à son subordonné une fois ceci fait.

### 2.2.3.3 Le *Data Plan*

Afin d'accueillir les instances des différents composants protocolaires, le composant géré par l'*Autonomic Manager*, le *Data Plan*, est composé de plusieurs éléments permettant l'assemblage et la manipulation de ceux-ci. Ces éléments sont représentés sur la figure 2.7.

**Le *Channel*** Le composant principal du *Data Plan* est le *Channel*. C'est au sein de celui-ci que sera instancié le service composite qui aura la charge de gérer la communication. Un *Channel* n'est pas nécessairement unique. En effet, plusieurs *Channel* peuvent être instanciés au même moment, contenant des services composites différents, notamment dans certains scénarios de reconfiguration. Ainsi, chaque *Channel* possède un identifiant, permettant de savoir quel message doit être traité par quel *Channel*. Les composants prenant place dans le *Channel* sont les composants protocolaires. Tout service composite se compose d'un composant protocolaire de type protocole et d'autant de composants protocolaires de type micro-protocole que nécessaire.

FIGURE 2.7: La composition du *Data Plan*

Ceci implique que les protocoles existants dans les systèmes actuels soient convertis en composants protocolaires pour être intégrés à l'ATL. Le cœur du protocole n'a pas à être modifié. Seule sa manière de communiquer avec le reste de la pile réseau du système doit être adaptée. Ainsi, au lieu d'être directement relié à l'application et au protocole de la couche réseau, le protocole de Transport devra récupérer les messages depuis les micro-protocoles situés au-dessus et au-dessous de lui dans le *Channel* dans lequel il prend place. Ceci peut être fait soit en reliant directement les interfaces de chaque composant protocolaire directement, soit en utilisant le *Channel* comme ordonnanceur. Celui-ci aurait alors la charge de faire transiter le message de composant en composant, récupérant ce dernier après chaque traitement pour le passer au suivant.

**Multiplexeur-démultiplexeur** Afin de permettre l'aiguillage entre les différents *Channel*, quand plusieurs sont présents, un multiplexeur-démultiplexeur prend position entre les *Channel* et l'interface vers l'application, et entre les *Channel* et l'interface vers le réseau. Ceux-ci effectuent leurs tâches grâce aux identifiants des différents *Channel*, qui ont été préalablement estampillés dans les messages.

**Buffers** Lorsqu'un message arrive de l'application ou du réseau, il est placé dans un buffer en attendant d'être pris en charge par le multiplexeur-démultiplexeur qui l'aguillera vers le *Channel* adéquat. Ces buffers jouent le rôle de tampon entre l'application et les *Channel* ainsi qu'entre les *Channel* et le réseau, afin d'absorber les différences de rythme entre chaque couche. Ce sont également eux qui sont en charge d'estampiller chaque message entrant avec l'identifiant de *Channel* qui doit les traiter, puis de retirer l'estampille à la sortie du *Data Plan*, afin que celle-ci n'interfère pas avec les applications (principalement les applications *legacy*), avec le réseau (car on créerait alors une sorte de nouveau protocole, qui ne serait pas reconnu partout), et avec le système distant (principalement les systèmes *legacy*).

**Channel Manager** Ce composant est le relais de l'*Autonomic Manager* au sein du *Data Plan*. C'est lui qui remonte les différentes informations de monitoring à l'*Autonomic Manager*, mais également qui gère les créations et les modifications de *Channel* selon les ordres de l'*Autonomic Manager*. Il possède également la capacité de mettre les multiplexeurs-démultiplexeurs en pause afin de geler la communication, et de notifier ceux-ci ainsi que les buffers des identifiants des nouveaux *Channel*, afin d'estampiller et d'aiguiller correctement les nouveaux messages, en fonction des *Channel* présents.

### 2.2.3.4 Intégrateur des services et base de connaissances

La fonctionnalité de gestion des services (cf. 2.1.4.1) est notamment prise en charge par la base des composants et services et par l'intégrateur de services. Ceux-ci permettent d'effectuer la correspondance entre services et composants protocolaires, ce qui peut impliquer le cas échéant une négociation avec l'hôte distant — si celui-ci dispose également de l'ATL.

**Correspondances entre services et composants** La correspondance entre les composants protocolaires disponibles et les services qu'ils offrent est nécessaire pour la traduction des requêtes applicatives constituées de services, en solution protocolaire potentiellement composite permettant de satisfaire ces requêtes.

Pour ce faire l'ATL dispose d'une base répertoriant ces services et comportant pour chaque composant protocolaire donné, la description du service qu'il offre.

L'*Autonomic Manager* de chaque *Flow* peut y effectuer une requête permettant de récupérer le ou les composants susceptibles de satisfaire tout ou partie des services requis — le cas d'un service partiellement couvert par un composant pouvant être pallié par sa composition avec un ou plusieurs autres fournissant les services manquants.

L'intégration de nouveaux composants doit permettre leur utilisation immédiate par les différentes applications, et donc par les différents *Flow* de l'ATL. Ainsi, le rôle de l'intégrateur est de maintenir la base des composants et services afin que chaque nouveau composant installé sur le système voit la description du service qu'il offre être mis à disposition des *Autonomic Manager* pour leurs prises de décision futures.

## 2.2.4 Composants secondaires

### 2.2.4.1 *Flow Creator*

La création d'un *Flow* est effectuée par le *Flow Creator*. C'est lui qui reçoit les demandes de création, correspondant à l'ouverture des différents points d'accès au service de Transport, que ceux-ci soient ouverts par des applications *legacy*, *aware* ou *smart*. Il lui incombe donc d'intercepter les appels de méthodes classiques afin de rediriger leur prise en charge vers l'ATL, ce qui implique une modification de l'implémentation de l'API socket classique, de manière à ce que les applications *legacy* ne soient pas modifiées, mais appellent tout de même l'ATL (à leur insu) lors de l'ouverture de sockets. Cette API devra également être étendue afin de prendre en compte les spécificités des applications *aware* et *smart*.

C'est lui également qui récupère les différentes informations passées par l'application à la création du point d'accès, comme le protocole de Transport désiré par les applications *legacy* ou les spécifications de besoins en qualité de service pour les applications *aware* ou *smart*, ainsi que d'éventuelles contraintes supplémentaires pour les applications *smart*.

On voit directement sur la figure 2.3 que le *Flow Creator* est une nouvelle manière d'implémenter la partie Transport de l'API socket classique pour le cas des applications *legacy* et dans le contexte de l'ATL. Dans le cas des applications *aware* et *smart*, le *Flow Creator* est l'implémentation des nouvelles fonctionnalités des extensions d'API, ou de la nouvelle API, nécessaire au fonctionnement de ces nouveaux types d'applications.

### 2.2.4.2 *Signalization Message Controler*

Le composant suivant présent sur la figure 2.3 est le *Signalization Message Controler* (SigMsgCtrlr). Ce composant est chargé de l'échange des messages de

gestion entre deux instances de l'ATL, concernant des informations sur l'état global de l'ATL sur l'hôte distant et non pas sur l'état d'un *Flow* en particulier. Si les deux systèmes impliqués dans une communication disposent de l'ATL, un *Autonomic Manager* désirant avoir une information sur l'ATL du système distant, comme par exemple la liste des services instanciables sur ce système, adressera cette requête à son *Signalization Message Controler* qui la reliera à celui du système distant. Si au contraire l'*Autonomic Manager* souhaite avoir une information concernant précisément le *Flow* distant auquel il est relié, telle que des informations de monitoring de la communication, il devra directement s'adresser à l'*Autonomic Manager* de ce *Flow*.

C'est notamment le *Signalization Message Controler* qui aura la charge de découvrir si l'ATL est présente sur une machine distante à laquelle on souhaite se connecter. Afin d'économiser certaines ressources système et réseau ainsi que temporelles, ce composant peut embarquer des fonctions de cache, gardant en mémoire certaines des informations qu'il a déjà découvertes, afin de les distribuer à un *Autonomic Manager* les requérant sans avoir besoin qu'un message de requête ne parcoure le réseau.

### 2.2.4.3 Multiplexage-démultiplexage entre couche réseau et *Flow*

Le dernier composant est un multiplexeur-démultiplexeur se situant entre les *Flow* et la couche réseau. Son rôle est de récupérer les différents messages provenant du réseau afin de les aiguiller vers le *Flow* adéquat. Pour une telle opération, les *Flow* doivent avoir un adressage leur permettant d'être différenciés au sein de la machine. Dans un premier temps nous voyons comment fonctionne ce type d'adressage dans les systèmes actuels avant de proposer une manière d'adresser les *Flow* de l'ATL.

**Adressage dans les systèmes actuels** Sur un système *legacy*, un socket est identifié, en mode connecté, selon le flux auquel il appartient. Or un flux est défini par le quintuplet : ( $@IP_{src}$ ,  $@IP_{dest}$ , protocole de Transport,  $port_{src}$ ,  $port_{dest}$ ). En mode non connecté, un socket est défini par le triplet ( $@IP_{dest}$ , protocole de Transport,  $port_{dest}$ ). Le passage au protocole de Transport est assuré par la couche réseau. Le passage au socket est assuré alors par le protocole de Transport selon le numéro de port. Une fois le paquet arrivé sur le système hôte, l'identification locale réside donc dans le couple (protocole de Transport,  $port_{dest}$ ). Dans le cas de l'ATL cependant, tout message échangé entre deux *Flow* ne comporte pas toujours ces informations, comme les messages de gestion échangés entre *Autonomic Manager* par exemple. De plus, le service composite étant variable, le protocole utilisé dans celui-ci peut également varier, rendant obsolète le couple (protocole de Transport,  $port_{dest}$ ) pour l'identification d'un *Flow* dans l'ATL.

**Adressage d'un *Flow*** Un *Flow* est associé à un et un seul point d'accès au service de Transport, et réciproquement. Adresser un *Flow* revient donc à adresser le point d'accès associé. Un *Flow* comportant toujours un composant protocolaire de type protocole, et celui-ci utilisant alors un numéro de port, le couple (protocole de Transport,  $\text{port}_{dest}$ ) peut être utilisé par l'ATL pour aiguiller les paquets arrivant de la couche réseau vers le *Flow* correspondant.

Cependant, lorsque l'*Autonomic Manager* n'a pas encore statué sur le protocole de Transport à utiliser, ni l'une ni l'autre des deux valeurs du couple précédemment cité n'est encore disponible. Pourtant, certains messages à destination du *Flow* peuvent déjà être émis et reçus, comme ceux permettant la négociation du service composite qui sera à instancier, négociation qui permettra à l'*Autonomic Manager* de décider du protocole de Transport à utiliser par la suite. Ceux-ci doivent disposer d'une information permettant leur identification et leur démultiplexage.

Si l'application ouvrant le point d'accès est une application *legacy*, elle fournit comme à l'heure actuelle le protocole de Transport et le numéro de port avec lesquels elle souhaite travailler. Sinon, l'application doit fournir un identifiant traduisant le service qu'elle offre à travers ce point d'accès, suivant ainsi la construction orientée service de l'ATL. Un navigateur web pourra ainsi fournir comme identifiant CLTWEB, signifiant que le point d'accès venant d'être ouvert est destiné à être client d'une communication web (utilisant http). Le système y adjoint un numéro afin de différencier différents points d'accès ouverts et offrant le même service. Ainsi, un navigateur web souhaitant charger deux pages web simultanément ouvrira deux points d'accès (un par page), identifiés CLTWEB par l'application, et complétés par le système en CLTWEB1 et CLTWEB2.

Cet identifiant est associé automatiquement grâce à une table de traduction à un couple (protocole de Transport,  $\text{port}_{dest}$ ). Par exemple, un serveur web ouvrant un point d'accès le nommera avec l'identifiant SRVWEB, qui sera automatiquement associé au couple (tcp, 80).

Si l'ATL est présente sur le système distant, alors les messages de négociation de la composition en provenance de celui-ci désigneront le *Flow* auquel ils sont destinés via les identifiants textuels. Sinon, on déduira que le *Flow* vers lequel diriger les messages entrant est celui dont le couple (protocole de Transport, n° port) correspond à ce que contient le message ou on utilisera ce couple comme protocole de Transport à utiliser.

Ceci sous-entend le besoin de standardiser certains identifiants textuels ainsi que leur correspondance au couple (protocole de Transport,  $\text{port}_{dest}$ ), comme sont à l'heure actuelle standardisés certains de ces couples pour des utilisations particulières. Le couple (tcp, 80) est par exemple standardisé pour les communications utilisant http. L'identifiant correspondant à ce couple ainsi que cette correspondance devront également être standardisés pour permettre la transition vers l'ATL

de manière transparente.

### 2.3 Conclusion

Ce chapitre a présenté les différents composants qui forment la structure de l'ATL. Tout d'abord, nous avons détaillé les objectifs que doit remplir l'ATL et les paradigmes de conceptions employés pour les atteindre : la conception orientée service et basée composant. Les intérêts et impacts de ces deux approches ont été notamment abordés.

Dans une seconde partie, nous avons décrit les trois types de fonctionnalités qui doivent être présentes au sein de l'ATL pour assurer le bon déroulement des communications. Ces fonctionnalités peuvent être du type données, contrôle ou gestion. Puis nous avons détaillé les différents composants permettant de mettre ceci en place, d'abord à un premier niveau de décomposition, puis plus en détails dans le cas des composants dits primaires, ceux qui accueillent les flux de données, mais également les composants secondaires, permettant aux premiers de fonctionner correctement.

Le chapitre suivant s'attarde sur les différents comportements dont font preuve ces composants dans deux cas particuliers : celui de l'ouverture d'un point d'accès au service de Transport par l'application, quel que soit son type — *legacy*, *aware* ou *smart* — et celui de la reconfiguration du service composite proposé par un *Flow*.



## Description comportementale de l'ATL : cas de la création et de la reconfiguration d'un service composite

Le précédent chapitre a introduit et décrit la structure composite de l'ATL, en détaillant le rôle de chacun des composants dans le but de satisfaire les objectifs fixés à la fin du chapitre 1.

Dans le présent chapitre, nous décrivons le comportement des différents éléments précédemment introduits dans deux cas particuliers :

- la création d'un point d'accès au service de Transport ;
- la reconfiguration d'un *Flow*.

Ces deux situations revêtent une importance particulière.

Le premier cas mérite d'être étudié car l'ouverture d'un point d'accès avec l'ATL diffère des solutions classiques. En effet ici, la liaison avec le protocole de Transport n'est pas directe. Pourtant, les applications *legacy* ne doivent pas sentir de différence afin de ne pas avoir à subir de mise à jour. De plus les applications *aware* et *smart* communiquent différemment avec la couche Transport que les applications *legacy*.

Le second cas présente un intérêt car s'il n'est pas totalement nouveau, les problématiques de reconfiguration comportementale et structurelle, abordées en 3.2, sont ici déclinées dans le contexte de flux de données de Transport et présentent des particularités. Il s'agit de plus d'une fonctionnalité centrale de l'ATL sans laquelle ses objectifs ne seraient pas atteints.

La première partie du chapitre décrit le cas de l'ouverture d'un point d'accès au service de Transport, et la deuxième le cas de la reconfiguration. La troisième partie aborde les problèmes de traduction d'état dans le cadre des reconfigurations, à travers la problématique du traitement des messages nécessitant des retransmissions.

#### Sommaire

3.1	Création d'un point d'accès au service de Transport . . . . .	<b>74</b>
3.1.1	Instanciation d'un <i>Flow</i> . . . . .	74
3.1.2	Application cliente . . . . .	76
3.1.3	Application serveur . . . . .	80
3.1.4	Synthèse . . . . .	82
3.2	Reconfiguration d'un <i>Flow</i> . . . . .	<b>83</b>
3.2.1	Déroulement d'une reconfiguration . . . . .	83
3.3	Traitement des retransmissions lors des transitions . . . . .	<b>88</b>
3.4	Conclusion . . . . .	<b>90</b>

---

## 3.1 Création d'un point d'accès au service de Transport

Cette section décrit le comportement de l'ATL dans les différents cas de figure possibles, selon le type de l'application — *legacy*, *aware* ou *smart* —, du système distant avec lequel on souhaite communiquer — *legacy* ou *enabled* — et selon que l'on se place du côté client (au sens initiateur) ou serveur de la communication.

Une synthèse des différents cas de figure est ensuite proposée afin de simplifier le nombre de cas à prendre en compte.

### 3.1.1 Instanciation d'un *Flow*

Parmi toutes les situations citées ci-dessus, certaines actions sont nécessaires dans tous les cas. Cette suite d'actions concerne l'instanciation effective du *Flow* afin qu'il puisse par la suite accueillir le service composite. En effet, que le système distant soit ATL-*enabled* ou *legacy*, que l'application demandant la création du TSAP soit *legacy*, ATL-*aware* ou *smart*, que l'on soit serveur ou client, les actions décrites ici seront toujours menées, et correspondent à la préparation du *Flow*. Nous appelons cette suite d'actions : comportement commun.

Lorsque l'application effectue une demande de création d'un point d'accès au service de Transport (TSAP) comme dans le diagramme de la figure 3.1, elle utilise une méthode telle la méthode socket de l'API éponyme en C, ou le constructeur de

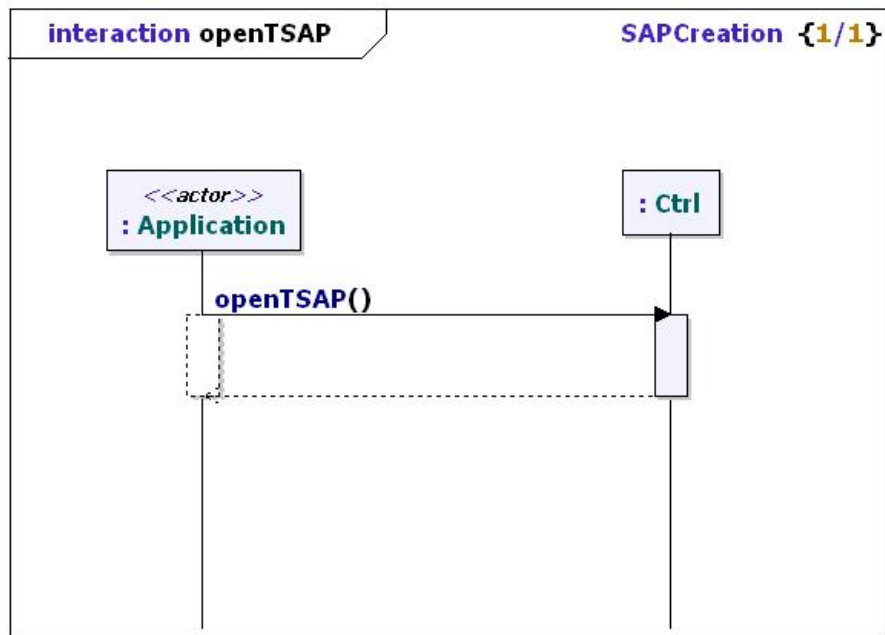


FIGURE 3.1: Une application de type *aware* ou *smart* ouvre un point d'accès au service de Transport via l'interface de contrôle de l'ATL

la classe *Socket* en Java. Dans un système comportant l'ATL, ces appels à méthode de création, qu'elles proviennent des API classiques ou d'une API dédiée à l'ATL, doivent être récupérés par le *Flow Creator*. Celui-ci crée alors un nouveau *Flow* au sein de l'ATL, et le lie au TSAP. Dans le *Flow* sont instanciés un *Autonomic Manager* et un *Data Plan*, dédiés à ce *Flow*, comme décrit sur le diagramme de la figure 3.3.

Une fois le *Flow* instancié, ainsi que son *Data Plan* et son *Autonomic Manager*, le *Flow Creator* communique à ce dernier les caractéristiques du service demandé par l'application.

Celles-ci vont différer selon le type d'application :

- Une application *legacy* communiquera le protocole de Transport qu'elle souhaite utiliser ainsi que le numéro de port utilisé sur le système serveur de la communication.
- Une application *aware* communiquera les caractéristiques qu'elle souhaite voir respectées par le service ainsi que le nom du point d'accès dont elle

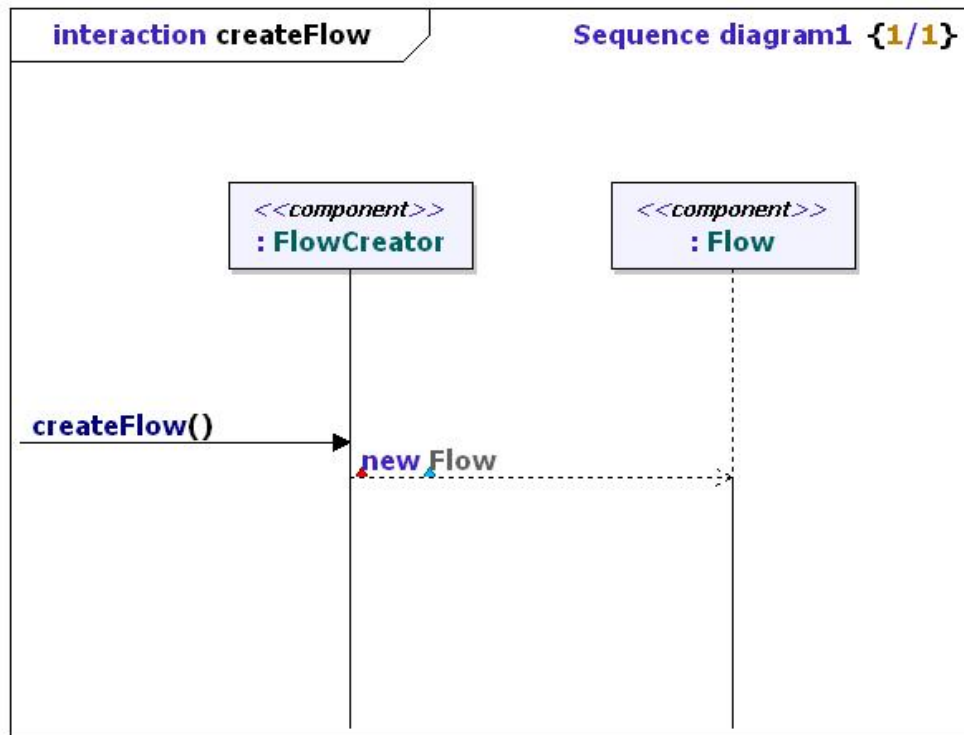


FIGURE 3.2: Diagramme de séquence de création d'un *Flow*

demande la création (cf. 2.2.4.3).

- Une application *smart* communiquera les mêmes informations qu'une application *aware*, ainsi qu'un certain nombre de contraintes supplémentaires, en imposant par exemple l'utilisation d'un certain protocole comme base de la composition protocolaire (cf. 2.2.1).

Le *Flow* est alors prêt à accueillir un nouveau *Channel* comportant le service composite qui assurera la communication.

### 3.1.2 Application cliente

Cette section décrit le comportement de l'ATL lorsque l'application est cliente de la connexion, c'est-à-dire qu'elle fournit à l'ATL l'adresse IP du système distant via une demande de connexion explicite — comme par exemple, pour une

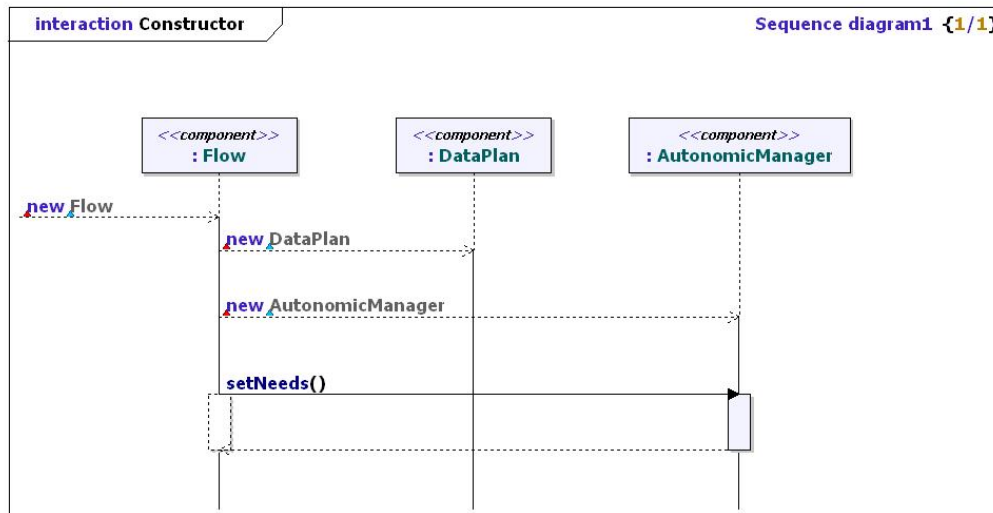


FIGURE 3.3: Instanciation d'un nouveau *Flow*

application *legacy* ouvrant un socket TCP — ou émet un message à destination du système distant — comme par exemple une application *legacy* utilisant UDP.

#### 3.1.2.1 Application *legacy*

Une application *legacy* doit pouvoir ouvrir un TSAP tel qu'elle l'a toujours fait, c'est-à-dire via une méthode faisant appel à l'API socket classique. Elle indique via cette méthode, le protocole de Transport ainsi que le numéro de port à utiliser. Cette méthode lui retourne ensuite ledit socket. Du point de vue de l'application, ce comportement ne doit pas être modifié.

Après avoir effectué le comportement commun, l'*Autonomic Manager* doit, afin de décider du service composite à déployer, récupérer certaines informations.

- Il connaît déjà le protocole qui lui a été demandé par l'application et peut en déduire une contrainte de décision (qui est de fait : « utiliser ce protocole ») et un type de Qualité de Service demandée (par exemple : « ordre et fiabilité totale » si TCP est demandé) ;
- Ce service peut être suffisant. Cependant, il peut-être plus adéquat de le coupler à d'autres en fonction de l'évolution des conditions réseaux.
- De plus, si le système hébergeant le serveur dispose de l'ATL, il peut avoir des desiderata ou des exigences quant aux composants protocolaires à utiliser pour implanter les services requis. En conséquence de quoi l'*Autonomic*

### 3. DESCRIPTION COMPORTEMENTALE DE L'ATL

*Manager* doit savoir si le serveur auquel il compte se connecter dispose ou non de l'ATL également.

L'application étant cliente, l'ATL dispose de l'adresse du serveur. Ainsi, l'*Autonomic Manager* va effectuer une demande de découverte de la présence de l'ATL sur le serveur grâce au *Signalization Message Controller*. Ce dernier va alors effectuer une requête auprès de son homologue sur l'ATL distante. La réponse, positive ou négative, sera alors transmise à l'*Autonomic Manager* requérant.

Bien que d'autres solutions soient envisageables, une solution simple de mise en œuvre de la requête du *Signalization Message Controller* consiste à envoyer un paquet TCP de type SYN sur un port prévu à cet effet. La réception d'un SYNACK constituera alors une réponse positive, celle d'un RST une réponse négative. L'inconvénient majeur d'une telle solution consiste bien sûr en la réservation d'un numéro de port à cette utilisation. Son avantage principal, cependant, réside dans l'utilisation d'un paquet TCP, paquet qui ne sera filtré que par une minorité de middleboxes, permettant ainsi d'être compatible avec la quasi totalité du réseau existant.

Quelle que soit la méthode d'implémentation retenue, la découverte doit être rapide. Un modèle simple de type requête/réponse serait alors à privilégier, limitant ainsi le temps de découverte à un RTT maximum, comme illustré sur la figure 3.4.

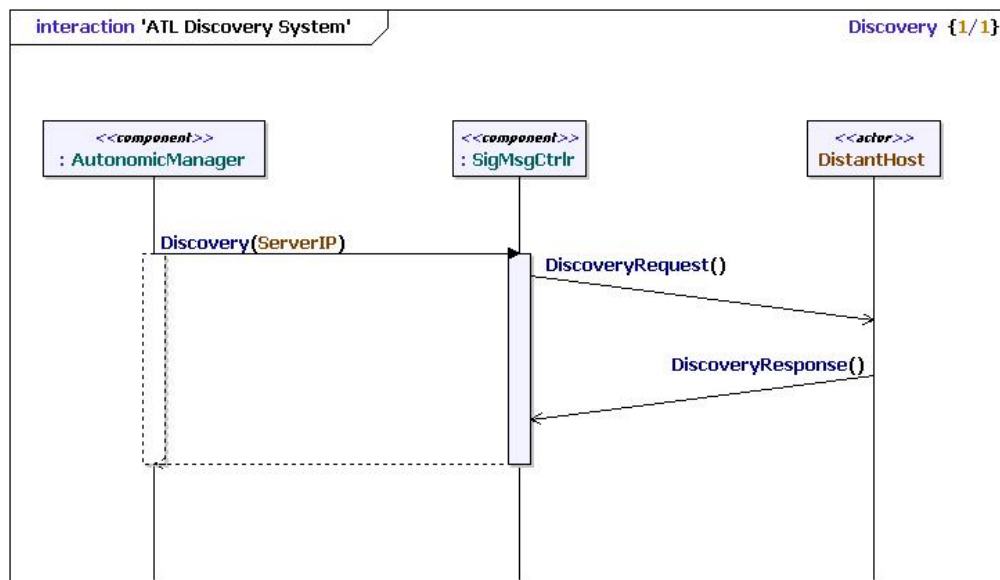


FIGURE 3.4: Mécanisme de découverte de l'ATL

En cas de réponse négative (*i.e.* l'hôte distant ne dispose pas de l'ATL) l'*Autonomic Manager* doit décider seul d'un service composite utilisant le protocole requis par l'application. En effet, l'ATL n'étant pas présente sur le système distant, on peut conclure que le protocole que celui-ci utilisera est nécessairement celui demandé par l'application. Il est donc impératif de l'utiliser. Cependant, s'il le juge utile, l'*Autonomic Manager* peut décider d'ajouter certains mécanismes protocolaires locaux au service composite.

En cas de réponse positive, l'*Autonomic Manager* va entrer en relation avec son pair distant sur le serveur, afin d'entamer une négociation avec celui-ci concernant la composition du service composite à mettre en place. Cette négociation concerne les composants protocolaires distribués, ceux-ci étant répartis sur les deux systèmes. Des mécanismes protocolaires locaux peuvent y être adjoints de chaque côté de la communication, si tant est que ceux-ci ne créent pas de conflit avec les composants distribués.

Une fois le service composite décidé, l'*Autonomic Manager* va l'instancier en demandant la création d'un nouveau *Channel* au *Channel Manager* du *Data Plan* (cf. 3.1.1).

#### 3.1.2.2 Application ATL-aware

Une application ATL-aware ouvre un TSAP en passant la description de ses besoins en Qualité de Service (*Quality of Service*) (QoS) en paramètre, ainsi que le nom qu'il attribue au TSAP (cf. 2.2.4.3).

Après instanciation d'un nouveau *Flow*, l'*Autonomic Manager* ne connaît que les besoins en terme de services requis par l'application cliente.

Il ignore cependant sur quel protocole se baser pour répondre aux services requis. Cette question peut facilement être résolue en négociant la composition avec l'hôte distant. Ainsi :

- l'*Autonomic Manager* va effectuer une demande de découverte au *Signalization Message Controller*, comme précédemment ;
- tout comme dans le cas d'une application *legacy*, une réponse positive va donner lieu à une négociation basée sur les critères de services requis par l'application, et les éventuelles contraintes imposées par celles-ci ;
- dans le cas d'une réponse négative, l'*Autonomic Manager* va décider d'un service composite basé sur le protocole de Transport obtenu via traduction du nom attribué au TSAP (cf. 2.2.4.3), et comportant éventuellement un ou plusieurs blocs de service locaux s'il le juge nécessaire.

Dans l'exemple où un client web ATL-aware ouvre un TSAP, si le *Signalization Message Controller* retourne une réponse positive, les deux *Autonomic Manager* (*i.e.* : du *Flow* client et du *Flow* serveur) entamerons une négociation pouvant

aboutir à l'utilisation de différents protocoles comme TCP ou SCTP, cette décision pouvant différer d'une négociation à l'autre.

Si le *Signalization Message Controler* retourne une réponse négative, l'*Autonomic Manager* traduit le nom du *Flow*, CLTWEB (cf. 2.2.4.3), en (tcp, 80), indiquant qu'il doit utiliser le protocole TCP sur le port 80.

#### 3.1.2.3 Application *smart*

Dans le cas d'une application *smart*, le comportement sera le même que précédemment, pour une application ATL-*aware*, à ceci près que l'application a la possibilité d'indiquer des contraintes supplémentaires à l'*Autonomic Manager* dans ses prises de décisions.

Toute situation ne pouvant aboutir à cause de ces contraintes devra alors être reportée à l'application. En effet, en imposant des contraintes supplémentaires, l'application reprend en charge certaines responsabilités qui seraient assumées par l'*Autonomic Manager* dans le cas d'une application ATL-*aware*.

Par exemple, une application imposant le cryptage des communications en utilisant TLS [Die08] recevra une erreur si aucun composant ne prenant en charge ce protocole n'est disponible sur le système.

### 3.1.3 Application serveur

Le comportement décrit ici recouvre les cas où l'application est le serveur de la communication, c'est-à-dire qu'elle ouvre un point d'accès disponible pour les connexions entrantes — comme pour un serveur utilisant TCP — ou des messages émis depuis d'autres systèmes distants — comme dans le cas d'une application UDP uniquement réceptrice.

#### 3.1.3.1 Application *legacy*

Après avoir instancié un nouveau *Flow*, et comme côté client, on doit tout d'abord savoir si le système distant dispose de l'ATL. Cependant, cette fois, on ne dispose pas de l'adresse du client.

Ainsi, les deux cas de figures suivants se présentent.

1. Le client (distant) dispose de l'ATL. Celui-ci a émis une requête de découverte afin de savoir si le serveur dispose de l'ATL. Celle-ci est récupérée par le *Signalization Message Controler* qui y répond positivement. Par la suite, le client contacte l'*Autonomic Manager* du *Flow* adéquat sur le serveur pour négocier le service composite. Le *Flow* dans lequel se situe cet *Autonomic Manager* est désigné par le nom de service qu'aura spécifié l'application cliente à l'ouverture de son TSAP (cf. 3.1.2.1).



2. Le client ne dispose pas de l'ATL. Le premier message en provenance de celui-ci est alors récupéré et analysé par le multiplexeur-démultiplexeur situé entre les *Flow* et la couche réseau afin de déterminer le protocole de Transport utilisé par le client. Celui-ci doit normalement être le même que celui que nous a imposé l'application serveur lorsqu'elle a ouvert le TSAP. Ce TSAP correspond alors à un *Flow* notamment identifié par ce protocole de Transport et par un certain numéro de port. Le message sera alors dirigé vers le *Flow* identifié par le protocole de Transport et le numéro de port utilisés dans son en-tête.

Dans le premier cas, la négociation a été engagée par le client afin de décider la partie distribuée du service composite à instancier, sur laquelle bâtir également la partie locale le cas échéant. Dans le second, on peut directement décider d'une composition locale basée sur le protocole demandé.

#### 3.1.3.2 Application *ATL-aware*

Le comportement est ici similaire au cas précédent, à la différence que, comme côté client pour une application *ATL-aware* (cf. 3.1.2.2), on ne dispose que des besoins en terme de services de l'application.

Si on reprend les deux cas précédents, qui s'appliquent également ici, on remarque que rien ne change dans le premier de ces cas, mais que dans le second, le *Flow* de l'application serveur ne dispose pas d'un couple (protocole de Transport, n° de port) pour l'identifier — l'application utilisant un nom de service comme décrit dans 2.2.4.3.

Dans le premier cas, le *Flow* vers lequel diriger le message de négociation de l'*Autonomic Manager* du client a été déterminé par ce nom de service. Dans le second, on ne dispose que du couple (protocole de Transport, n° de port destination). La correspondance avec le nom de service du *Flow* auquel diriger ce message se trouve alors dans la table de traduction introduite en 2.2.4.3.

#### 3.1.3.3 Application *smart*

Comme côté client, deux cas se présentent ici :

1. si l'application a imposé le protocole à utiliser, le comportement sera le même que pour une application *legacy* ;
2. sinon, le comportement sera le même que pour une application *ATL-aware*.

L'observation de cette dichotomie simple dans le cas d'une application serveur donne lieu à la classification alternative décrite à la section 3.1.4.

#### 3.1.4 Synthèse

On remarque dans les deux sections précédentes que le comportement de l'ATL à la création d'un point d'accès au service de Transport est conditionné à deux paramètres indépendants, et donne donc lieu à quatre cas. Ces deux paramètres sont :

1. la connaissance que le système a de l'adresse IP du système distant avec lequel communiquer :
  - une application cliente connaît l'adresse IP du système distant,
  - une application serveur ne la connaît pas ;
2. le fait que l'application impose ou non le protocole de Transport à utiliser :
  - une application *legacy* l'impose,
  - une application *aware* ne l'impose pas,
  - une application *smart* peut ou non l'imposer.

Quel que soit le cas, la première action à effectuer est bien évidemment d'instancier un *Flow* grâce au *Flow Creator*. Par la suite, les actions à mener vont donc être les suivantes.

**On dispose de l'adresse IP du système distant.** Dans ce cas, la première action à effectuer est de lancer la procédure de découverte de l'ATL sur le système distant via le *Signalization Message Controller*. Si la réponse est positive, on peut immédiatement entamer une procédure de négociation du service à mettre en place. Sinon l'*Autonomic Manager* doit décider d'une composition utilisant des mécanismes protocolaires locaux, basée sur le protocole de Transport déterminé :

- par l'application si celle-ci nous l'a imposé (comme le ferait une application *legacy*) ;
- grâce à la table de traduction si l'application a seulement nommé le point d'accès au service de Transport (comme le ferait une application *aware*).

**On ne dispose pas de l'adresse IP du système distant.** Dans ce cas on doit adopter un comportement « serveur » et attendre un message entrant afin de connaître l'identité du pair de la communication. Soit le premier message reçu est un message de négociation de service, et le système distant dispose donc de l'ATL. On dirige ce message vers le *Flow* identifié par le nom correspondant au nom de service véhiculé dans le message. On peut alors poursuivre avec lui la négociation.

Soit le message reçu est un message de données et on déduit alors que le système distant ne dispose pas de l'ATL. On dirige alors le message vers le *Flow* correspondant au couple (protocole de Transport, n° de port) du message entrant car le *Flow* est ainsi identifié par l'application ou via la table de traduction. On peut alors décider d'une composition locale.

Dans tous les cas, une fois ces étapes effectuées, on peut instancier la composition et démarrer la communication.

## 3.2 Reconfiguration d'un *Flow*

Une fois le service composite instancié et la communication établie, les conditions dans lesquelles ce service travaille peuvent varier. Ainsi non seulement les objectifs de qualité de service de l'application peuvent être modifiés par celle-ci, mais l'environnement réseau est par nature dynamique et peut donner lieu à des modifications importantes de ses caractéristiques, ne permettant plus à un service donné d'atteindre les objectifs pour lesquels il a été choisi. Dans une telle situation, la solution réclame la reconfiguration, plus ou moins en profondeur, du dit service.

On observe principalement deux types de reconfiguration.

1. La première, dite reconfiguration « comportementale », consiste à ne pas modifier le service composite dans sa structure, c'est-à-dire, dans le cas de l'ATL, conserver les mêmes composants protocolaires, mais interviendra sur les paramètres de ces derniers afin d'altérer leur fonctionnement. Ainsi, modifier la valeur maximum que peut prendre le *Retransmission Time Out* (RTO) dans TCP constitue un exemple de reconfiguration comportementale.
2. La seconde est une reconfiguration dite « structurelle ». Celle-ci implique que la structure du service composite va être modifiée par l'ajout, la suppression ou le remplacement d'un ou plusieurs composants protocolaires. On peut par exemple ainsi décider de ne plus utiliser SCTP au cours d'une communication pour le remplacer par MPTCP, car l'apparition de middleboxes bloquerait alors SCTP.

Cette section aborde la problématique de la reconfiguration en décrivant le comportement de l'ATL pour mener à bien les différents cas.

### 3.2.1 Déroulement d'une reconfiguration

#### 3.2.1.1 Classification des cas de reconfiguration possibles

Il existe deux cas de reconfiguration :

- comportementale, où l'on modifie un ou plusieurs paramètres uniquement ;
- structurelle, où l'on va modifier la composition en changeant un ou plusieurs composants protocolaires, ceci selon deux possibilités :
  - l'ajout ou le retrait des composants protocolaires concernés,
  - le déploiement de la nouvelle composition dans un nouveau *Channel* si la précédente méthode est impossible.

Le choix de l'une ou l'autre de ces deux possibilités peut par exemple être influencé par la manière dont on souhaite traiter les messages à retransmettre (cf. 3.3).

Dans le troisième cas ci-dessus deux méthodes peuvent être utilisées pour déployer la nouvelle composition : séquentielle ou parallèle.

- Avec la première méthode, on déploie le nouveau *Channel* séquentiellement dans le temps. C'est-à-dire que l'*Autonomic Manager* va d'abord devoir attendre que le *Channel* courant ait terminé son exécution avant de le détruire puis de déployer le nouveau *Channel*.
- La seconde méthode consiste à déployer et démarrer la nouvelle composition alors que la précédente finit son travail. Ainsi, les nouveaux envois de données passent par la nouvelle composition pendant que les retransmissions passent par l'ancienne.

Ces deux méthodes sont opposées sur deux points : l'*overhead* temporel et l'utilisation mémoire.

- Sur les systèmes très contraints en ressources, la méthode parallèle peut s'avérer coûteuse.
- Cependant, la première présente l'inconvénient de devoir attendre la fin du premier *Channel* avant de démarrer le suivant. Si le premier instancie un service composite dont le temps d'arrêt est non déterministe, le coût en temps peut rapidement devenir trop important.

#### 3.2.1.2 Suivi du plan de reconfiguration

Toute reconfiguration se déroule en une succession d'étapes définies par le plan d'exécution qui a été décidé par la fonction de planification des *Autonomic Manager* selon leur organisation.

Certaines étapes du plan nécessitent une synchronisation des deux *Flow* pour être effectuées. Afin de permettre l'adaptation à une grande variété d'actions possibles et à venir, notamment due à l'importante multiplicité des composants protocolaires et des versions de ceux-ci que permet l'ATL, le protocole de synchronisation des actions doit être le plus simple possible, laissant au plan lui-même le soin d'encapsuler la complexité.

Ce type de décomposition de la complexité est déjà utilisé depuis longtemps pour la gestion de réseaux avec le protocole SNMP, possédant un nombre réduit de commandes génériques, et permettant d'aller interroger des bases de données spécifiques et potentiellement complexes, les *Management Information Base* (MIB) [Pre02].

Selon un principe similaire, nous proposons ici les messages suivant pour le protocole :

- *setPlan*, permettant à l'*Autonomic Manager* décisionnaire d'imposer le plan à l'autre ;
- *nextStep*, permettant à l'*Autonomic Manager* décisionnaire d'ordonner le passage à la réalisation de l'étape suivante à l'autre *Autonomic Manager*, celui-ci étant bloqué en attendant cet ordre ;
- *stepAck*, permettant à un *Autonomic Manager* d'informer la réalisation d'une étape à l'autre *Autonomic Manager*.

Le but de ces messages est uniquement de permettre la synchronisation d'étapes interdépendantes. Dans le cas d'*Autonomic Manager* collaboratifs (cf. 2.2.3.2), le plan n'est pas imposé par un *Autonomic Manager* sur l'autre, il est décidé par consensus. De la même manière, aucun n'autorise l'autre à passer à l'étape suivante, chacun informe l'autre de son état d'avancement dans la réalisation, et s'auto-débloque lorsqu'il le juge approprié.

Ainsi le protocole reste générique en n'encodant absolument aucune des commandes spécifiques aux différents composants protocolaires utilisables par une ATL ou l'autre. Ces commandes sont indiquées dans le plan de reconfiguration, à chaque étape. La commande *nextStep* du protocole de reconfiguration permet donc de passer à la commande suivante à exécuter, quelle que soit celle-ci. Le protocole de reconfiguration devient ainsi scalable vis-à-vis du nombre de composants protocolaires possibles. La figure 3.5 montre le déroulement d'une reconfiguration dans le cas où un des deux *Autonomic Manager* orchestre l'opération.

### 3.2.1.3 Déroulement selon les cas

Nous allons ici décrire le déroulement de chacun des quatre cas de reconfiguration :

1. comportemental ;
2. par ajout ou retrait d'un composant protocolaire dans un *Channel* existant ;
3. par déploiement d'un nouveau *Channel* en série ;
4. par déploiement d'un nouveau *Channel* en parallèle.

**Reconfiguration comportementale** Le processus de décision distribué entre les *Autonomic Manager* a statué pour une reconfiguration comportementale, c'est-à-dire pour la reconfiguration d'un ou plusieurs paramètres au sein d'un ou plusieurs composants protocolaires du *Channel* en cours d'exécution.

C'est à ce processus de décision de décider si les actions à entreprendre sur ces différents paramètres doivent respecter un certain ordre d'exécution s'il y en a plusieurs, pour des raisons de dépendance par exemple.

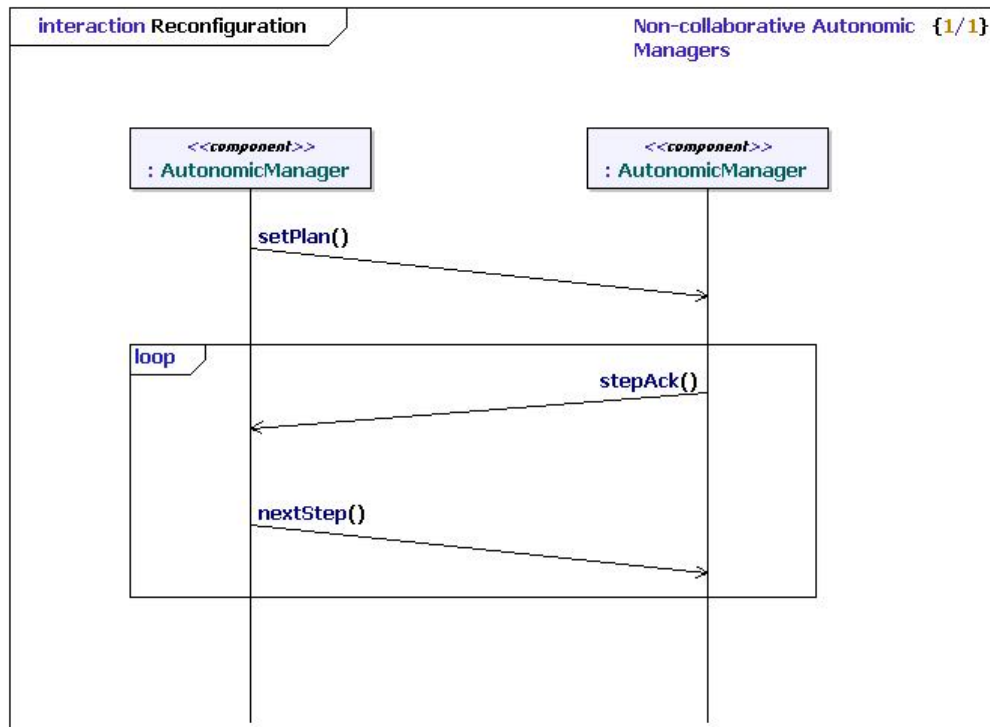


FIGURE 3.5: Suivi du plan de reconfiguration

Chaque paramètre sera modifié via l'invocation d'une méthode appropriée du *Channel Manager*. Celui-ci, sur cet ordre, invoque la procédure de modification correspondante publiée par le composant à modifier.

Nous pouvons ici remarquer qu'une modification de paramètre du point de vue des *Autonomic Manager* peut aboutir à une reconfiguration structurelle interne du composant protocolaire ciblé. En effet, si le composant est lui même un ensemble composite, sa composition représente un de ses paramètres. Modifier celle-ci revient à modifier ce paramètre, sans modifier la structure protocolaire du *Channel*, du point de vue de l'ATL, même s'il s'agit, du point de vue du composant concerné, d'une restructuration interne.

L'utilisation d'ETP comme protocole au sein du *Channel* est un exemple de cette situation. En effet, modifier la composition du service interne d'ETP s'effectue en changeant le fichier de description de composition de ce service, donc via un changement de paramètre, du point de vue de l'*Autonomic Manager*. Cette modification va cependant initier une procédure de reconfiguration au sein d'ETP,

qui possède un tel système de synchronisation avec l'entité ETP distante. Cette reconfiguration prenant du temps, il peut être nécessaire d'attendre qu'elle soit terminée avant d'effectuer l'action de reconfiguration (de niveau ATL) suivante.

**Retrait/ajout d'un composant protocolaire au sein d'un *Channel* existant** Afin de modifier la composition protocolaire d'un *Channel*, par ajout et/ou retrait d'un composant, il est nécessaire de mettre ce *Channel* en pause le temps d'effectuer la modification. La première opération à effectuer sera donc le gel des *Channel*, à commencer par le *Channel* émetteur de données, puis par le récepteur. Si les deux *Channel* émettent chacun, alors ils doivent mettre en pause leur capacité d'émission mais toujours être capables de recevoir des données.

Pour cela, les *Channel Manager* ont la possibilité d'ordonner le gel de la transmission des messages depuis les buffers vers le *Channel* concerné aux multiplexeurs/démultiplexeurs du *Data Plan*. Ceux-ci vont alors cesser de transmettre les messages depuis les buffers vers les *Channel*, permettant à ceux-ci de vider les messages en cours de traitement.

Par la suite, les *Autonomic Manager* vont retirer ou ajouter les différents composants protocolaires prévus, éventuellement de manière synchronisée, via une méthode d'ajout ou de retrait fournie par le *Channel Manager*. Celui-ci retire le composant protocolaire concerné, ou insère une nouvelle instance du composant protocolaire demandé à l'emplacement désiré.

Ces modifications effectuées, les *Autonomic Manager* synchronisent le redémarrage des capacités réceptrices de leurs *Channel*, puis, par la suite, leurs capacités émettrices, par le déblocage de l'activité des multiplexeurs/démultiplexeurs.

**Déploiement d'un nouveau *Channel* en série** Cette opération consiste à déployer un service composite au sein d'un nouveau *Channel* après avoir détruit le *Channel* actuellement en cours d'exécution.

Tout d'abord, les deux *Autonomic Manager* synchronisent l'arrêt de leur *Channel* respectif, en mettant la communication en pause, comme précédemment. A ce moment, les *Channel* doivent terminer leurs envois et éventuelles retransmissions. Afin que cette étape ne s'éternise pas au détriment de la QoS demandée, il est important que l'*Autonomic Manager* puisse signifier un temps maximum d'attente au delà duquel la destruction du *Channel* est effectuée, travail terminé ou non. Ce temps doit donc être calculé en fonction de l'impact négatif causé par un arrêt prématuré du *Channel*, et indiquant le moment où cet impact est plus faible que celui causé par l'attente de l'arrêt total de celui-ci.

Une fois le travail du *Channel* terminé (ou le temps d'attente maximal atteint), celui-ci est détruit par le *Channel Manager*, qui le remplace ensuite par un nouveau *Channel*, contenant le nouveau service composite. De manière synchrone

à nouveau, les deux *Autonomic Manager* vont redémarrer la communication, les messages étant alors traités par le nouveau *Channel*.

**Déploiement d'un nouveau *Channel* en parallèle** Cette opération similaire à la précédente, déploie le nouveau *Channel* et le démarre en parallèle de l'ancien qui finit son travail. Chaque *Autonomic Manager* doit synchroniser le numéro de séquence à partir duquel les messages doivent transiter par le nouveau *Channel*, par exemple en l'incluant dans le plan de reconfiguration. Si le nouveau *Channel* n'est pas encore déployé de chaque côté alors que ces messages doivent être envoyés, ils doivent être mis en tampon dans les buffers en attendant le démarrage du *Channel*.

Les *Autonomic Manager* doivent en parallèle gérer la création du dit nouveau *Channel*, puis, de manière synchrone, le démarrer. Tout nouveau message dont le numéro de séquence est supérieur au numéro de séquence limite doit être traité par ce nouveau *Channel*. L'ancien *Channel* continue pendant ce temps d'effectuer son travail, et donc les éventuelles retransmissions. Une fois le travail de celui-ci effectué, il est détruit.

## 3.3 Traitement des retransmissions lors des transitions

Toute reconfiguration réclame une certaine continuité dans le service fourni. Pour cela, le nouveau service déployé doit démarrer dans un état correspondant à celui du service qu'il remplace, au moment de la prise de relai de ce dernier. Nous allons illustrer cette problématique dans cette partie à travers l'exemple de la retransmission de messages, initialement émis avant la reconfiguration mais dont la retransmission devrait être effectuée après celle-ci.

Pour ceci, nous caractérisons l'environnement à un instant donné par la paire  $E = \{B, C\}$  où  $B$  représente l'ensemble des besoins en qualité de service demandés par l'application, et  $C$  les contraintes imposées par le réseau. Afin de répondre à cet environnement, l'ATL va mettre en place un service  $S$  correspondant.

### Tous les messages doivent-ils cependant transiter par le même service ?

Un message est émis avec les besoins de l'application au moment où celle-ci le soumet à l'émission, mais avec les contraintes du moment où il est réellement transféré au réseau. Ceci signifie que l'environnement donné pour un message est défini en deux temps : les besoins  $B$  au moment où l'application le soumet à l'ATL, et les contraintes  $C$  par la suite lorsque l'ATL le retire des buffers pour le faire transiter dans le *Channel*. Ainsi, si les besoins correspondant à ce message ne vont pas varier entre la transmission initiale et la retransmission du message,



les contraintes peuvent, elles, différer entre ces deux instants, les conditions du réseaux étant dynamiques.

**Cas d'une retransmission idéale** Nous discutons ici de la solution idéale à adopter, dans le cas de systèmes sans limite de ressources et où les temps de reconfiguration sont négligeables.

Du paragraphe précédent, on déduit que ce sont les raisons du changement d'environnement qui vont conditionner le service qui servira à retransmettre les messages le nécessitant. Bien sûr, les messages dont la première émission prend place après la reconfiguration, seront retransmis (si tant est que le nouveau service le prévoit) par le nouveau service. Seuls sont concernés les messages initialement transmis par l'ancien service.

**Changement de besoins** Avant reconfiguration, nous nous trouvons dans la situation  $E_1 = \{B_1, C_1\}$ . Le changement de besoins modifie donc l'environnement en  $E_2 = \{B_2, C_1\}$  les contraintes n'ayant pas changé. Les messages ayant été initialement transmis par le service  $S_1$ , répondant à l'environnement  $E_1$ , ont donc été soumis à l'émission par l'application durant l'environnement  $E_1$ . Les besoins qui leur sont associés sont donc les besoins  $B_1$ .

Les contraintes n'ayant pas changé et étant toujours  $C_1$ , ces messages, au moment de leur émissions correspondront toujours à l'environnement  $E_1$ . Ils devraient donc dans l'idéal être retransmis avec l'ancien service,  $S_1$ . Ceci est possible dans l'idéal, car le système, n'ayant pas de limite de ressources, peut déployer  $S_2$  en parallèle de  $S_1$ .

**Changement de contraintes** Avant reconfiguration, nous nous trouvons dans la situation  $E_1 = \{B_1, C_1\}$ . Le changement de contraintes modifie donc l'environnement en  $E_2 = \{B_1, C_2\}$  les besoins n'ayant pas changé. Les messages ayant été initialement transmis par le service  $S_1$ , répondant à l'environnement  $E_1$ , ont donc été soumis à l'émission par l'application durant l'environnement  $E_1$ . Les besoins qui leur sont associés sont donc les besoins  $B_1$ .

Les besoins n'ayant pas changé et étant toujours  $B_1$ , ces messages, au moment de leur émissions correspondront maintenant à l'environnement  $E_2$ . Ils devraient donc dans l'idéal être retransmis avec l'ancien service,  $S_2$ . On peut donc dans ce cas opter pour une reconfiguration par modification du service en place, le transformant de  $S_1$  à  $S_2$ , l'ancien service n'ayant plus d'utilité.

**Changement de besoins et de contraintes** Nous ne considérons ici que le changement de besoins et de contraintes peut-être simultané. Dans les faits, il peut

être considéré comme tel à cause de la latence induite par la fonction de Monitoring. Avant reconfiguration, nous nous trouvons dans la situation  $E_1 = \{B_1, C_1\}$ . Le changement de situation modifie donc l'environnement en  $E_2 = \{B_2, C_2\}$  les besoins et les contraintes ayant changé. Les messages ayant été initialement transmis par le service  $S_1$ , répondant à l'environnement  $E_1$ , ont donc été soumis à l'émission par l'application durant l'environnement  $E_1$ . Les besoins qui leur sont associés sont donc les besoins  $B_1$ .

Les contraintes ont cette fois changé et sont à présent  $C_2$  ainsi ces messages, au moment de leur émission correspondront toujours à un environnement  $E_3 = \{B_1, C_2\}$  pour lequel il n'existe pas de service. Ils devraient donc dans l'idéal être retransmis avec un service,  $S_3$  temporaire uniquement dédié à leur retransmission. Ceci est possible dans l'idéal, car le système, n'ayant pas de limite de ressources, peut déployer  $S_3$  en parallèle de  $S_2$  et détruire  $S_1$ , devenu inutile, ou modifier ce dernier en  $S_3$  si possible.

**Considérations sur les cas réels** Bien évidemment, les systèmes possèdent des limites et les temps de reconfiguration selon les réseaux considérés, ne pourront pas toujours être vus comme négligeables. Ainsi, ces trois solutions sont une simple base sur laquelle construire des prises de décisions plus poussées, prenant en compte des variables plus complexes. En effet, si le déploiement de deux services en parallèle est impossible, que faire de ces messages? Doit-on les envoyer avec le nouveau service, et donc avec un service n'étant pas forcément parfaitement approprié? Ou doit-on considérer leur retransmission d'une importance moindre dans le nouveau contexte et l'abandonner, tout en détruisant l'ancien service?

Ces questions ne sont que quelques exemples de la complexité de la question de la prise de décisions, centrale dans un système comme l'ATL. Ce thème de la décision réclame à lui seul une étude ultérieure poussée et exhaustive.

## 3.4 Conclusion

Dans ce chapitre nous avons présenté le comportement des différents composants de l'ATL dans deux cas particuliers : l'ouverture d'un point d'accès et la reconfiguration d'un service au sein d'un *Flow*. Ceci nous a permis de mettre en lumière différents cas possibles en fonction des systèmes, des applications et des choix faits ou faisables par l'ATL, offrant ainsi tout un panel de solutions pour répondre à un maximum de situations.

Nous avons ensuite, à travers l'exemple de la retransmission de messages, illustré la problématique de la transition d'un service à un autre et levé des interrogations concernant la complexité du système de décision que réclame un système tel que l'ATL.

# Expérimentations et mesures

Dans ce chapitre, nous détaillons des expériences en simulation et émulation qui ont permis la mise en évidence de l'intérêt d'une solution tel que l'ATL, ainsi que son coût potentiel.

Dans une première partie, nous décrivons une simulation dans laquelle nous combinons MPTCP avec différents mécanismes de fiabilité partielle afin de tirer partie des capacités multi-chemins du protocole à des fins d'amélioration d'une transmission vidéo. Cette simulation nous permet ainsi de confirmer l'intérêt de combiner un protocole donné avec des mécanismes externes afin de moduler le service offert par celui-ci.

Dans une deuxième partie, nous décrivons une simulation dans laquelle un changement d'environnement survient en milieu de communication — une transmission vidéo — à laquelle le système réagit en modifiant la composition protocolaire, basée sur MPTCP. Les résultats sont ensuite comparés au cas témoin où la composition protocolaire reste inchangée, afin de montrer le bénéfice induit.

Enfin, la troisième partie de ce chapitre présente une illustration du coût d'une solution telle que l'ATL, en étudiant le cas du protocole de synchronisation d'ETP, dont les similitudes architecturales avec l'ATL ont été montrées au chapitre 2.

## Sommaire

4.1	De la composabilité des protocoles . . . . .	<b>92</b>
4.1.1	But de l'expérience . . . . .	92
4.1.2	Description de l'expérience . . . . .	93
4.1.3	Analyse des résultats . . . . .	97
4.1.4	Conclusion . . . . .	101
4.2	De l'intérêt de l'adaptation dynamique . . . . .	<b>102</b>
4.2.1	But de l'expérience . . . . .	103

4.2.2	Description de l'expérience . . . . .	103
4.2.3	Analyse des résultats . . . . .	104
4.2.4	Conclusion . . . . .	105
4.3	De l' <i>overhead</i> de la signalisation . . . . .	<b>106</b>
4.3.1	But de l'expérience . . . . .	106
4.3.2	Description de l'expérience . . . . .	107
4.3.3	Résultats attendus . . . . .	114
4.3.4	Conclusion . . . . .	116
4.4	Conclusion . . . . .	<b>117</b>

---

## 4.1 De la composabilité des protocoles

Cette section du chapitre propose d'étudier l'intérêt de coupler protocoles existants et micro-protocoles afin de permettre une meilleure satisfaction des besoins en qualités de service de l'application, par rapport à l'utilisation du protocole seul. Pour ceci nous prenons l'exemple d'une application de streaming vidéo communiquant via un réseau de type 3G sur des terminaux comportant plusieurs interfaces. Afin d'en tirer parti, nous utilisons un protocole multi-domicilié, MPTCP, et le composons avec des mécanismes de fiabilité partielle.

### 4.1.1 But de l'expérience

Le but de cette expérience est de démontrer l'intérêt de coupler des mécanismes protocolaires dédiés à certaines tâches, et génériques, afin d'être réutilisables, avec des protocoles standards. Le but n'est pas d'étudier l'efficacité d'une composition de tels mécanismes avec un protocole comme le fait l'ATL, par rapport à une intégration d'un mécanisme au sein d'un protocole particulier, mais d'étudier l'intérêt d'un tel couplage, quelle que soit la méthode retenue.

MPTCP utilise les différentes interfaces présentes sur le système, de manière concurrente, afin d'augmenter la capacité réseau disponible et réduire le délai de transmission car, étant basé sur TCP, celui-ci est lié à cette capacité. Cependant, pour cette même raison, le service offert par MPTCP est parfaitement fiable et parfaitement ordonné, rendant ainsi ses avantages moins intéressants pour des applications de type streaming multimédia, qui peuvent tolérer une certaine quantité de pertes.

Le but de cette expérience est de démontrer l'intérêt de combiner des micro-protocoles de fiabilité partielle avec MPTCP, afin de tirer parti de ses avantages tout en améliorant la qualité de service envers des applications pour lesquelles ce protocole ne serait pas totalement adéquat.

Le caractère générique, c'est-à-dire indépendant d'un protocole particulier, de ces micro-protocoles leur permet d'être associé à d'autres protocoles, potentiellement en cours de communication (cf. 4.2). Ceci permet ainsi de mettre en évidence l'utilité de l'ATL, afin de gérer ce type de situation de manière autonome.

L'expérience est menée en simulation, grâce au simulateur ns-2.

## 4.1.2 Description de l'expérience

### 4.1.2.1 Choix de l'application

L'application choisie est une application de vidéo interactive basée sur le codec H.264. Ce type d'application tolère un certain niveau de perte d'information, mais requiert un délai de transit borné : les paquets dont le délai dépasse 400 ms seront considérés comme obsolètes et inutiles.

Pour simuler cette application, nous utilisons une vidéo de 400 images, encodées en YUV au format QCIF, soit une résolution de 176x144. La vidéo dure 13,6 s et est émise à un rythme de 30 images par seconde. Du côté du récepteur, l'adaptation effectuée consiste à écarter les paquets dont le délai a dépassé les 400 ms. La vidéo, encodée en H.264, est composée d'images de type I, P ou B.

**Images I :** Ces images sont autonomes. Elles peuvent être décodées indépendamment des autres.

**Images P :** Ces images encodent uniquement les informations qui ont été modifiées au sein de l'image par rapport à l'image I ou P précédente qu'elle prend pour référence. Ainsi, si une image I, ou P, est perdue, toutes les images P successives qui en dépendent seront mal décodées, provoquant des erreurs d'affichage.

**Images B :** Ces images se basent sur le même principe que les images P mais utilisent en référence une image I ou P précédente et une image I ou P suivante. Ces images sont donc fortement dépendantes du décodage correct des autres images du flux.

Ainsi, toutes les images ne revêtent pas la même importance au sein du flux vidéo. La perte d'une image B sera moins pénalisante que la perte d'une image P, elle-même moins handicapante que la perte d'une image I. Notons également que selon les choix à l'encodage de la vidéo, certaines images B peuvent également servir de référence, introduisant ainsi une hiérarchie à quatre niveaux et non plus trois.

L'utilisation d'une interface spécifique de type IPMH permet l'assignation de différents niveaux de priorité à chaque message applicatif, permettant à MPTCP et aux micro-protocoles de différencier le traitement qui sera appliqué à chacun.

### 4.1.2.2 Micro-protocoles utilisés

Pour réaliser cette simulation, les mécanismes suivants ont été implémentés dans ns-2, au sein du modèle ns-2 de MPTCP, recommandé par le working group MPTCP de l'IETF [Nis]. Chacun de ces micro-protocoles implémente un concept de fiabilité partielle dans le but d'améliorer le délai de transit afin de correspondre aux exigences de l'application, en tirant parti de la tolérance aux pertes de celle-ci.

**Adaptation à l'émission : le *Selective Discarding*** Le *Selective Discarding* a été introduit dans [LBL89] et permet une adaptation du flux à l'émission. Ce mécanisme consiste à éliminer les paquets de moindre importance lorsque le réseau est saturé.

Grâce à l'interface, MPTCP peut récupérer les informations relatives à la priorité de chaque type d'image. En utilisant cette information, notre implémentation du *Selective Discarding* écarte systématiquement les images de type B avant leur émission, car ce sont les images les moins importantes. Nous nous attendons ainsi à avoir une qualité de service améliorée, car le non-envoi d'un paquet conduit à moins de congestion et donc de perte sur le réseau et également à un délai moindre pour les images I et P, qui sont les images les plus importantes pour le décodage de la vidéo du côté du récepteur.

### **Adaptation à la réception : le *Time-Constrained Partial Reliability***

L'autre implémentation réalisée du concept de fiabilité partielle intègre l'aspect temporel de la contrainte. Puisque MPTCP utilise plusieurs chemins pour accroître la capacité réseau disponible, le risque de paquets arrivant dans le désordre s'en trouve augmenté, ces chemins possédant des caractéristiques différentes. MPTCP offrant un service d'ordre total, celui-ci place en buffers les messages arrivant dans le désordre jusqu'à réception des paquets manquants, afin de délivrer le tout dans l'ordre à l'application. Ces paquets en attente peuvent devenir obsolètes à cause de l'attente des paquets en retard, alors qu'ils sont disponibles et pourraient être utiles à l'application. Ainsi, le mécanisme a pour but que ces paquets soient délivrés avant leur obsolescence, considérant les paquets en retard comme perdus et à présent inutiles, en acquittant leur perte auprès de l'émetteur.

L'implémentation proposée calcule à tout instant le délai de transit applicatif des paquets dans les buffers de réception de l'hôte puits de la communication, c'est-à-dire le délai écoulé depuis leur soumission à l'émission par l'application émettrice. Lorsque ce délai se rapproche des 400 ms de délai maximum, les paquets concernés sont délivrés à l'application et les paquets manquants, considérés à présent obsolètes, sont déclarés perdus et acquittés auprès de l'émetteur, comme si ceux-ci avaient été correctement reçus. Ainsi, l'émetteur les croyant arrivés à

destination ne les réémettra pas. Si ceux-ci arrivent par la suite, ils seront tout simplement écartés.

Notons que la simulation permet d'éluder la question de la synchronisation des deux hôtes. En expérimentation réelles, celle-ci devrait être prise en compte via des solutions comme NTP [Mil10] ou toute autre solution offrant une précision appropriée. Le caractère du mécanisme protocolaire resterait alors local, mais sa dépendance envers un tel système de synchronisation, par nature distribué, rendrait la composition protocolaire globalement distribuée. Rappelons enfin que le but de cette expérience est d'étudier le couplage de mécanismes protocolaires avec un protocole standard, et non l'étude du mécanisme en lui-même.

#### 4.1.2.3 Simulation

Les deux mécanismes précédents permettent trois combinaisons :

1. MPTCP-SD qui couple MPTCP au *Selective Discarding* ;
2. MPTCP-PR qui couple MPTCP à la *Time-Constrained Partial Reliability*.
3. MPTCP-PR-SD qui couple MPTCP à la *Time-Constrained Partial Reliability* et au *Selective Discarding*.

Chacune d'elles a été testée sur l'environnement conseillé par le working group MPTCP. Celui-ci est représenté figure 4.1.

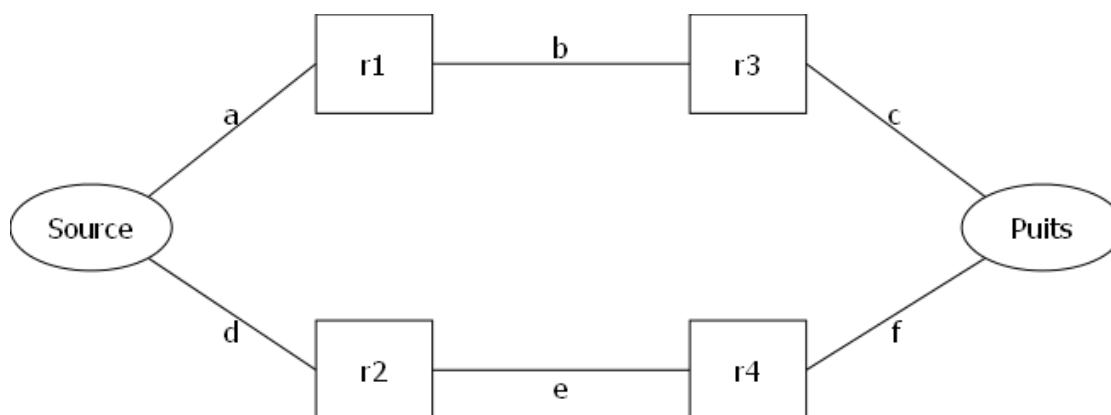


FIGURE 4.1: La topologie du réseau simulé

La source, abrégée S, représente le système hébergeant l'application émettrice de données, et le puits, abrégé P, le système hébergeant l'application réceptrice. Les machines r1 à r4 sont des routeurs intermédiaires. Nous disposons ainsi de deux chemins distincts utilisables par MPTCP. Ces chemins sont donc :

- S, a, r1, b, r3, c, P ;
- S, d, r2, e, r4, f, P.

#### 4. EXPÉRIMENTATIONS ET MESURES

---

Afin d'évaluer les performances des différentes combinaisons de mécanismes avec MPTCP, nous simulons un transfert vidéo entre S et P.

Les capacités des liens d'accès a, d, c et f ont été fixés soit à 1,8 Mbps, soit à 3,6 Mbps et les liens du cœur de réseau, b et e, sont fixés à 1 Mbps ou 0,5 Mbps. Ces valeurs ont été choisies proches de débits théoriques utilisables sur des liens WWAN HSDPA. Elles ont également été choisies afin de créer des situations dans lesquelles MPTCP devra réagir à un manque de capacité. Aucun trafic concurrent n'est injecté dans le réseau simulé. Les délais des différents liens ont été fixés à 5 ms ou 10 ms. Sept scénarios ont été définis, et sont résumés dans les tableaux 4.1 à 4.3.

	1	2
a et c	3,6 Mbps ; 10 ms	3,6 Mbps ; 10 ms
b et e	1 Mbps ; 5 ms	1 Mbps ; 5 ms
d et f	3,6 Mbps ; 10 ms	1,8 Mbps ; 10 ms
Congestion	faible	faible

TABLE 4.1: Résumé des scénarios 1 et 2

	3	4	5
a et c	3,6 Mbps ; 5 ms	1,8 Mbps ; 5 ms	1,8 Mbps ; 10 ms
b et e	0,5 Mbps ; 5 ms	1 Mbps ; 5 ms	0,5 Mbps ; 5 ms
d et f	3,6 Mbps ; 5 ms	1,8 Mbps ; 5 ms	1,8 Mbps ; 10 ms
Congestion	moyenne	moyenne	moyenne

TABLE 4.2: Résumé des scénarios 3 à 5

	6	7
a et c	3,6 Mbps ; 10 ms	1,8 Mbps ; 10 ms
b et e	0,5 Mbps ; 5 ms	0,5 Mbps ; 5 ms
d et f	3,6 Mbps ; 10 ms	1,8 Mbps ; 10 ms
Congestion	forte	forte

TABLE 4.3: Résumé des scénarios 6 et 7

**Métrique d'évaluation** La qualité est évaluée selon un calcul par image du Peak Signal to Noise Ratio (PNSR). Nous utilisons Evalvid [KRW03], un ensemble d'outils permettant d'évaluer la qualité d'une transmission vidéo sur réseaux réels ou simulés. Evalvid permet notamment la mesure de différents paramètres de QoS



tels que le taux de perte ou le délai de bout en bout. Il existe une correspondance entre le PNSR et une autre métrique d'évaluation vidéo, subjective celle-ci, le *Mean Opinion Score* (MOS), résumée dans le tableau 4.4.

PNSR (dB)	MOS
> 37	Excellent
31 à 37	Bon
25 à 31	Moyen
20 à 25	Faible
< 20	Pauvre

TABLE 4.4: Les correspondances entre PNSR et MOS

### 4.1.3 Analyse des résultats

#### 4.1.3.1 Comparaison des performances de MPTCP avec TCP et UDP

Dans cette partie, nous comparons la qualité vidéo, caractérisée par le PNSR, lorsque la transmission est effectuée avec MPTCP et lorsqu'elle est effectuée TCP ou avec UDP. Les calculs du PNSR sont effectués après réception des données et rejet de celles ayant un délai de transmission applicatif supérieur à 400 ms.

Les tableaux 4.5 et 4.6 montrent les bénéfices engendrés sur le PNSR par l'utilisation de MPTCP. On peut remarquer que pour UDP et TCP, la colonne 2 est vide : comme deux chemins avec des caractéristiques différentes sont utilisés par MPTCP, la comparaison avec UDP ou TCP, tous deux mono-chemins, pour ces caractéristiques est effectuée dans les cas 1 et 5. Ces chemins sont utilisés dans les simulations 1 et 4 pour TCP et UDP. Le PNSR pour TCP est parfois à 0 dB car tous les paquets, dans ces situations, ont dépassé le délai maximum de 400 ms et ont ainsi été rejetés car considérés comme obsolètes.

Scénario	1	2	3	4
UDP	13,98		14	4,27
TCP	4,66		4,64	0
MPTCP	24,96	25,59	22,88	22,51

TABLE 4.5: PNSR moyen en dB des simulations 1 à 4

Les tableaux 4.7 et 4.8 montrent que le délai est également globalement réduit grâce à l'utilisation de MPTCP.

Les tableaux 4.9 et 4.10 montrent quant à eux l'apport de MPTCP sur le taux de pertes pour chaque type d'image.

#### 4. EXPÉRIMENTATIONS ET MESURES

---

Scénario	5	6	7
UDP	4,26	4,25	4,13
TCP	0	0	0
MPTCP	23,05	19,15	17,14

TABLE 4.6: PNSR moyen en dB des simulations 5 à 7

Scénario	1	2	3	4
UDP	432		475	474
TCP	419		481	465
MPTCP	334	346	296	358

TABLE 4.7: Délai moyen en millisecondes des simulations 1 à 4

Scénario	5	6	7
UDP	516	575	616
TCP	527	581	629
MPTCP	319	403	428

TABLE 4.8: Délai moyen en millisecondes des simulations 5 à 7

Scénario	1	2	3	4
I	UDP	95,5		95,5
	TCP	100		100
	MPTCP	55,5	57,7	77,7
P	UDP	71,9		73,03
	TCP	23,59		23,59
	MPTCP	6	5,61	14,6
B	UDP	72,93		73,3
	TCP	0,75		14,41
	MPTCP	7,8	6,3	6,3

TABLE 4.9: Taux de pertes des simulations 1 à 4

##### 4.1.3.2 Comparaison de MPTCP avec et sans mécanismes couplés

Nous allons à présent comparer la qualité de la vidéo reçu via un transfert utilisant MPTCP et utilisant les différentes combinaisons de mécanismes :

- MPTCP-SD ;
- MPTCP-PR.

La comparaison a été effectuée dans le cadre du scénario 4 décrit dans le tableaux 4.2.

Scénario	5	6	7
I	UDP	100	100
	TCP	100	100
	MPTCP	55,5	80
P	UDP	89,88	93,25
	TCP	100	100
	MPTCP	11,2	15,7
B	UDP	86,09	91,72
	TCP	100	100
	MPTCP	1,12	19,17

TABLE 4.10: Taux de pertes des simulations 5 à 7

Si les résultats décrits 4.1.3 nous ont permis d'évaluer les bénéfices de MPTCP par rapport à TCP ou UDP, nous pouvons remarquer sur la figure 4.2 que le PNSR moyen pour MPTCP est de 22,5 dB, montrant que la qualité moyenne de la vidéo n'est pas suffisante pour une utilisation confortable.

En effet, le tableau 4.4 montre que le PNSR doit être supérieur à 25 dB pour avoir une qualité vidéo satisfaisante. On remarque également sur la figure 4.2 que le PNSR descend sous 22,5 dB pendant un temps prolongé, avant l'image n° 200 et aux alentours de l'image n° 300. Ces résultats se confirment à la visualisation de la vidéo reconstituée après réception, et motivent l'étude de la combinaison de MPTCP avec les mécanismes précédemment introduits.

Les figures 4.3 et 4.4 montrent le PNSR image par image pour l'utilisation de MPTCP-PR et MPTCP-SD respectivement, dans le contexte du scénario 4.

Nous remarquons immédiatement sur la figure 4.3 que le PNSR moyen obtenu est meilleur que celui observé avec MPTCP seul. En effet, celui-ci atteint à présent 24,74 dB. Cette moyenne est cependant toujours sous la barre des 25 dB, bien que proche. Nous pouvons cependant remarquer que plusieurs fois la vidéo reconstituée s'approche d'un PNSR de 35 db, ce qui est un très bon score. Malheureusement nous notons également des performances réduites durant la seconde moitié de la vidéo (entre la 230<sup>e</sup> et la 340<sup>e</sup> image). Ceci peut s'expliquer par la perte d'informations importantes, telle qu'une image I, déclarée perdue alors que celle-ci pouvait encore être utile, non plus pour son affichage, mais pour servir de référence pour des images P et B ultérieures. Ceci pourrait être amélioré en permettant un plus grand paramétrage du mécanisme, afin de moduler la date d'obsolescence en fonction du type de l'image, et non plus utiliser une date fixe.

Malgré ce léger écueil, et bien que la différence dans les chiffres puisse sembler mineure (10 %), l'amélioration est nettement visible au visionnage et on s'approche d'une utilisation confortable.

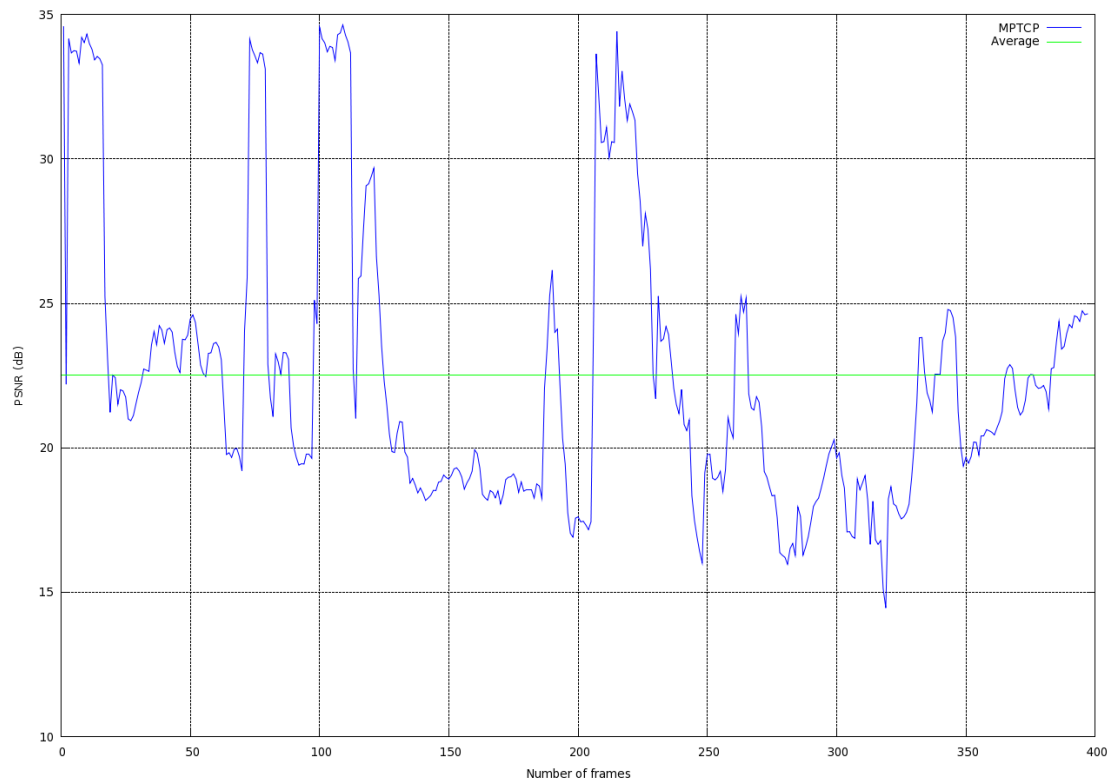


FIGURE 4.2: PNSR par image en utilisant MPTCP, scénario 4

La figure 4.4 représente l'évolution du PNSR par image avec utilisation de MPTCP-SD. On voit ici que le PNSR moyen dépasse 25 dB, avec une valeur de 26,04 dB. Les oscillations sont dues au fait qu'à chaque image B, le PNSR chute, en raison du rejet systématique de celle-ci. Cependant, celui-ci étant plus élevé pour les images P et I, la moyenne s'en trouve relevée.

Notons malgré tout une chute du PNSR aux alentours de la 200<sup>e</sup> image. Celle-ci est due à la perte d'une image I ou P servant donc de référence à plusieurs images qui la suivent. Celles-ci, ne pouvant se baser sur la bonne référence, seront décodées et affichées incorrectement, créant des différences trop fortes avec la vidéo d'origine, traduites par ce creux dans la courbe.

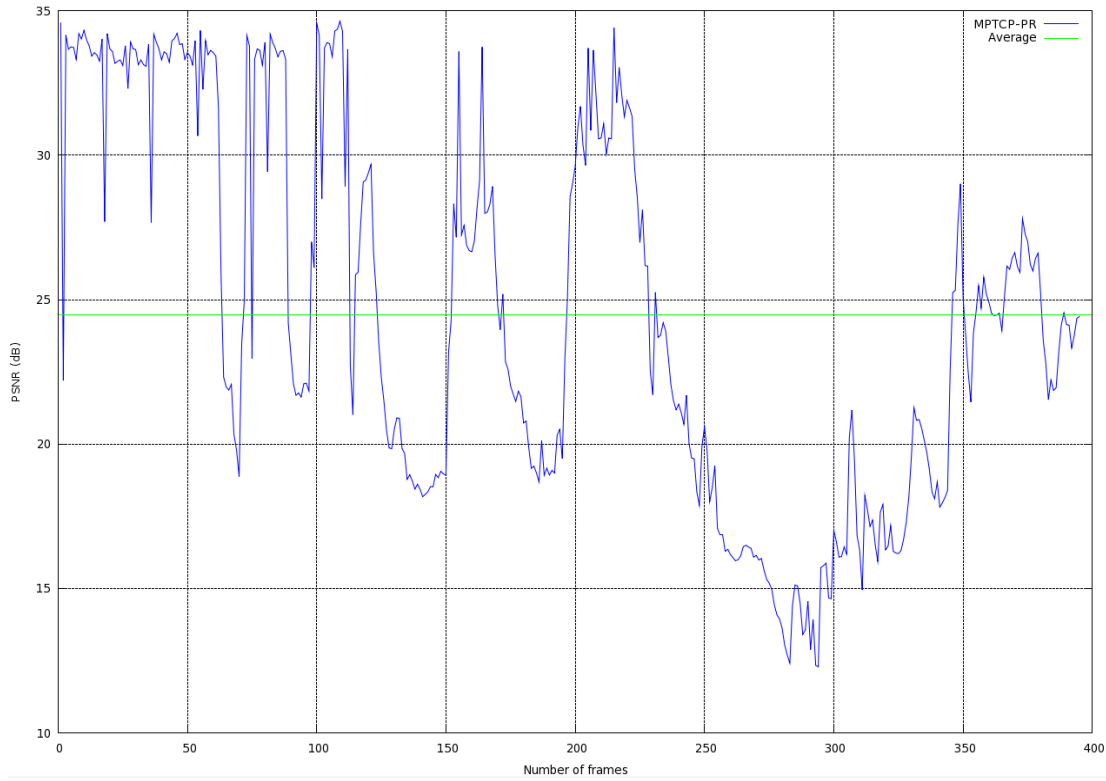


FIGURE 4.3: PSNR par image en utilisant MPTCP-PR, scénario 4

#### 4.1.4 Conclusion

L'expérience présente a permis de démontrer les gains en Qualité de Service que l'on peut obtenir via la combinaison de protocoles avec des mécanismes externes. Ces bénéfices pourraient encore être améliorés, notamment par l'optimisation des mécanismes, en permettant un paramétrage plus fin de ceux-ci par exemple. En effet, ici le *Selective Discarding* rejette systématiquement toute image de type B. Le même mécanisme pourrait être amélioré en permettant de modifier le taux de rejet pour chaque type d'image indépendamment ou en fonction des relations de dépendance d'une image à l'autre.

Un paramétrage plus fin, cependant, requerrait une adaptation dynamique afin de pouvoir tirer au mieux parti de celui-ci. Ces décisions et adaptations ne peuvent pas être requises de la part de l'utilisateur, ni de l'application. En effet, celles-ci sont trop complexes, et réclament une connaissance poussée du réseau et des protocoles. L'ATL pourrait cependant, parfaitement remplir ce rôle. La section

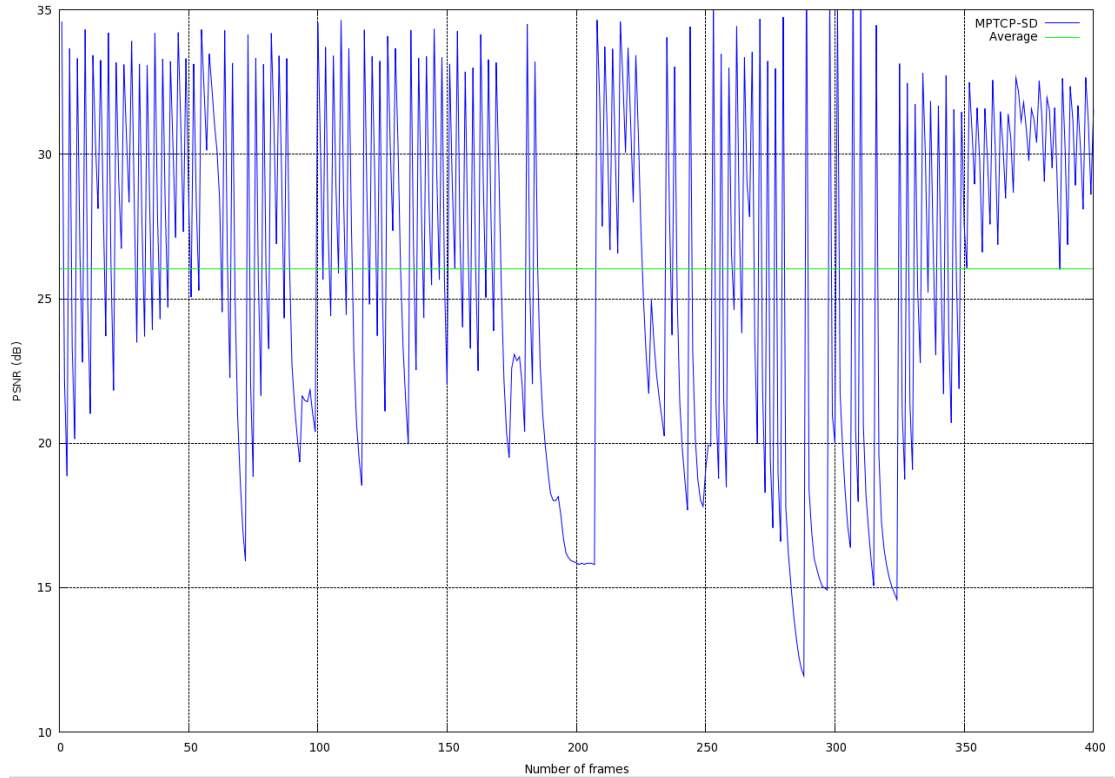


FIGURE 4.4: PSNR par image en utilisant MPTCP-SD, scénario 4

suivante étudie ce type d'adaptation dynamique.

## 4.2 De l'intérêt de l'adaptation dynamique

Dans cette deuxième partie, nous nous intéressons à l'effet de l'adaptation dynamique de la composition protocolaire, c'est-à-dire sa modification en cours de communication. Pour ceci, nous utilisons le même contexte que l'expérimentation précédente, en utilisant un nouveau mécanisme pour MPTCP : un *load balancer* chargé de répartir les paquets à envoyer sur les différents chemins possibles, en fonction de la priorité des messages et de la qualité de chaque chemin.

### 4.2.1 But de l'expérience

Le but de cette expérience est de démontrer l'intérêt de procéder à des modifications structurelles de la composition protocolaire en cours de communication. En effet, les adaptations comportementales, *i.e.* les modifications de paramètres, existent déjà dans les protocoles actuels. par exemple, TCP adapte la taille de ses fenêtres de contrôle de congestion et de contrôle de flux, en fonction de l'état de congestion du réseau, et de la saturation du système récepteur des données.

Ici, nous nous intéressons à l'adjonction d'un nouveau mécanisme à une composition protocolaire existante. Contrairement à la section 4.1, le mécanisme est pensé pour les protocoles multi-domiciliés. Ceci n'est pas contradictoire vis-à-vis de l'ATL. En effet, celle-ci est pensée pour accueillir des composants pouvant avoir des conditions de dépendance envers d'autres composants.

Comme pour la section 4.1, l'expérience est réalisée en simulation, grâce au simulateur ns-2.

### 4.2.2 Description de l'expérience

#### 4.2.2.1 Choix de l'application

L'application choisie est la même que pour l'expérience précédente. Sa description peut donc être trouvée en 4.1.2.1.

#### 4.2.2.2 Micro-protocoles

Le mécanisme implémenté ici est un *load-balancer*, ou répartiteur de charge. Dans le contexte des protocoles multi-domiciliés, son rôle est de répartir les paquets sur les différents chemins. Dans MPTCP, il est appelé *packet scheduler*. Notre implémentation de ce répartiteur se base, comme pour les mécanismes précédents, sur la différence de priorité des différentes images.

Notre implémentation va donc diriger les paquets sur l'un ou l'autre chemin en fonction de l'importance de chacun. Le but étant toujours de faire en sorte que les images les plus importantes aient le plus de chances d'arriver intactes à destination, le répartiteur va diriger les paquets vers le chemin le plus sûr pour les plus importants, et le moins sûr pour les moins importants.

La notion de chemin sûr est ici à préciser. Celle-ci implique bien évidemment un monitoring du réseau constant permettant une modélisation de chaque chemin. Afin de définir ensuite le plus sûr, il faut décider selon quelles caractéristiques un chemin sera qualifié de sûr : son taux de congestion, sa capacité, son délai de transit. . .

La répartition doit ensuite être décidée entre les différents niveaux de priorité qui peuvent être plus ou moins nombreux que le nombre de chemins. Un para-

métrage plus fin peut même permettre de faire passer un certain pourcentage de chaque priorité sur chaque chemin.

Dans notre implémentation et pour cette expérience, les images I et P sont dirigées vers le meilleur chemin, les images B vers le moins bon.

#### 4.2.2.3 Simulation

L'environnement réseau employé est le même qu'en 4.1 et est donc représenté sur la figure 4.1. De même, nous utilisons le PNSR comme mesure de la qualité de service.

Deux environnements différents ont été définis. ils sont résumés dans le tableau 4.11.

	1	2
a et c	1,8 Mbps ; 10 ms	1 Mbps ; 10 ms
b et e	1 Mbps ; 5 ms	1 Mbps ; 5 ms
d et f	3,6 Mbps ; 10 ms	3,6 Mbps ; 10 ms

TABLE 4.11: Résumé des scénarios

Pour chacun de ces scénarios, nous avons simulé le transfert de la vidéo en utilisant MPTCP et MPTCP couplé à notre *load-balancer*, indiqué si après MPTCP-LB, afin de déterminer quelle composition obtient les meilleures performances pour chaque environnement.

Dans un second temps, nous avons effectué la simulation du transfert avec modification de l'environnement. A mi-chemin du transfert, l'environnement change de 1 vers 2, et la composition protocolaire s'adapte en conséquence. Nous pouvons ainsi déterminer l'apport de la reconfiguration dynamique par rapport à un transfert qui emploierait le même protocole de bout en bout, comme c'est le cas actuellement.

### 4.2.3 Analyse des résultats

La première série de simulation compare MPTCP et MPTCP-LB sur chacun des deux environnements. Le tableau 4.12 résume ces résultats.

Simulation	1	2
MPTCP	25,59	21,75
MPTCP-LB	24,56	24,20

TABLE 4.12: PNSR en dB



Nous pouvons voir que sur le premier environnement, le PNSR obtenu avec MPTCP est meilleur que ce lui obtenu avec MPTCP-LB. En effet, les deux chemins étant très similaires, le répartiteur de charge ne peut exprimer pleinement son potentiel, et la répartition classique effectuée par MPTCP mène à de meilleurs résultats. Cependant, dans le second environnement, nous pouvons remarquer que les bénéfices du *load-balancer* peuvent s'exprimer, en raison de la grande dissimilarité des chemins.

Ces résultats nous poussent à nous poser la question suivante : quels bénéfices pourrait-on tirer de l'activation dynamique du *load-balancer* selon la différence de caractéristiques entre les chemins ?

Cette simulation reprend les caractéristiques de l'environnement n° 1 puis, au milieu du transfert, change les conditions pour celles de l'environnement n° 2. Dans cet environnement changeant, nous effectuons deux simulations. La première avec MPTCP seul, et la seconde en intégrant dynamiquement notre répartiteur de charge lors du changement d'environnement, celui-ci donnant de meilleures performances globales dans ces nouvelles conditions.

Les résultats obtenus sont indiqués dans le tableau 4.13.

Simulation	MPTCP seul	MPTCP puis MPTCP-LB
PNSR	21,77	22,56

TABLE 4.13: PNSR obtenu sans et avec adaptation dynamique

Nous pouvons remarquer une amélioration de 0,79 dB grâce à l'intégration dynamique du *load-balancer*. Cette valeur amène les commentaires suivants.

1. Ce bénéfice, si petit soit-il, est-il significatif, particulièrement pour un PNSR d'environ 21 dB ? En effet, en dessous de 25 dB la qualité vidéo doit être améliorée. Cependant, le visionnage de la vidéo nous permet déjà d'observer une amélioration.
2. Ces améliorations sont obtenues dans le cas idéal où le changement de composition protocolaire s'effectue sans délai par rapport au changement d'environnement. Le problème du coût de cette adaptation en conditions réelles se pose donc et doit être évalué.

#### 4.2.4 Conclusion

Dans cette section, l'expérience menée a permis de mettre en évidence les bénéfices qui peuvent être apportés par une adaptation dynamique de la composition protocolaire vis-à-vis du contexte. Ces bénéfices sont cependant à mettre en perspective du déroulement de la simulation : en effet celle-ci met en place un cas

idéal dans lequel le temps d'adaptation au contexte est nul. Les bénéfices seront probablement donc moindres en situation réelle.

Le paramétrage du mécanisme utilisé ici, cependant, et comme dans la section 4.1, devrait pouvoir être modifiable, afin de décider quelle proportion de chaque type d'image doit passer par chaque chemin. La décision de recomposition et le paramétrage doivent alors être pris en charge par un système de décision et de gestion autonome tel que l'ATL. Celui-ci devra également, dans ses décisions, prendre en compte les modifications de performances induites par le temps de reconfiguration. La section suivante s'intéresse aux temps de reconfiguration.

### 4.3 De l'*overhead* de la signalisation

La troisième partie de ce chapitre étudie le coût de la signalisation sur le déroulement de la communication. En effet parmi les différentes fonctionnalités de l'ATL, certaines requièrent une synchronisation entre les deux pairs de la communication. Afin d'accomplir cette synchronisation, il est nécessaire d'avoir des échanges de messages de contrôle entre les *Autonomic Manager*. Cette section tente d'évaluer l'impact des échanges de signalisation liés à la reconfiguration comme décrite en section 3.2.

#### 4.3.1 But de l'expérience

Cette expérience a pour but de déterminer le coût induit par les besoins en signalisation. Pour ce faire, nous utilisons ETP comme *framework* de composition protocolaire.

En effet, la structure d'ETP étant proche de celle de l'ATL, ce dernier pouvant être vu comme un *Data Plan* d'un point de vue structurel, son protocole de signalisation pour la négociation initiale et la recomposition dynamique peut être utilisé comme base de réflexion sur le coût de ces actions. En effet, la simple adjonction d'un *Autonomic Manager* à ETP permettrait de lui conférer toutes les caractéristiques d'un *Flow* comme l'accueil et la composition de composants protocolaires ou le multiplexage de plusieurs *Channel*.

ETP est capable d'accueillir des composants appelés *Processing Modules*. Ceux-ci sont une décomposition des micro-protocoles qui prennent place au sein du *framework*. Afin de déployer les compositions à mettre en place, ETP intègre un protocole de synchronisation permettant aux deux entités pairs de négocier la composition à mettre en place. Afin de permettre sa reconfiguration dynamique, ETP intègre un système de déploiement correspondant au schéma de reconfiguration structurelle en parallèle décrit en 3.2.1.3. Celui-ci est accompagné d'une variation du protocole de synchronisation initiale permettant la reconfiguration.

Nous utilisons donc ETP comme base de réflexion sur le coût de la signalisation dans le cadre de l'ATL, celle-ci étant proche structurellement nous permettant d'avoir une évaluation du coût de la signalisation liée à la reconfiguration.

### 4.3.2 Description de l'expérience

#### 4.3.2.1 Plateforme de test

L'expérience est ici réalisée en émulation, sur la plateforme laasNetExp hébergée au LAAS-CNRS. Celle-ci implémente une topologie représentée en figure 4.5.

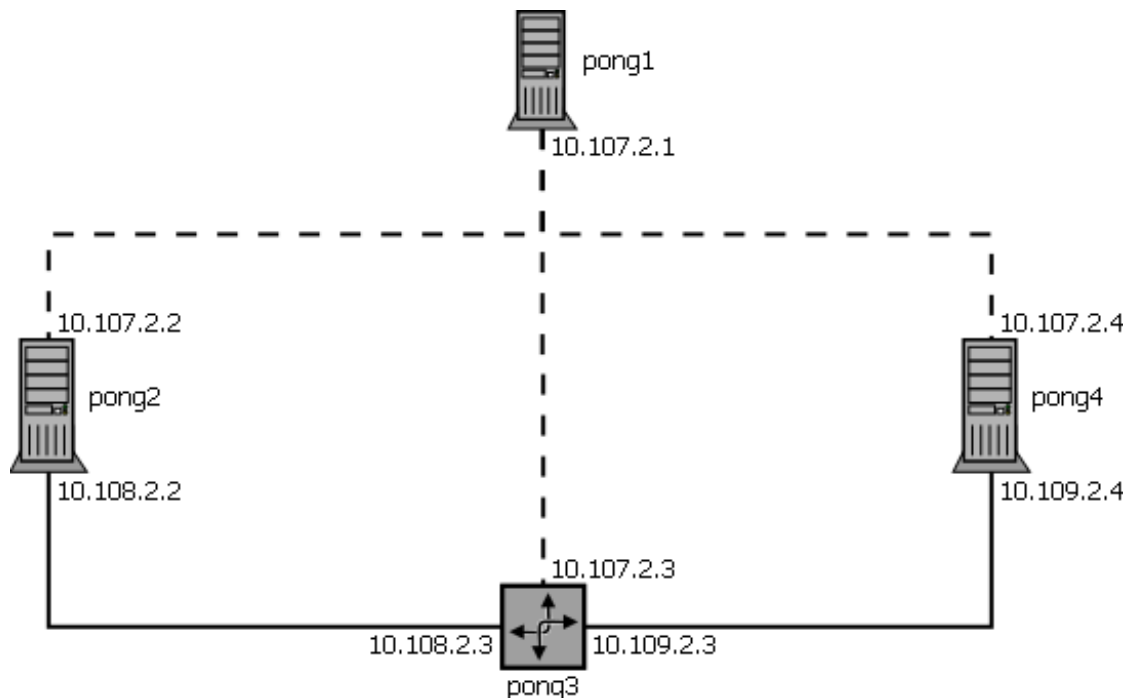


FIGURE 4.5: La topologie utilisée pour les tests

L'accès aux différents systèmes se fait via la machine **pong1**. Celle-ci peut ensuite commander les différents hôtes présents sur le réseau via le réseau de commande 10.107.2.0. L'hôte **pong2** accueille l'application source de données et **pong4** l'application puits. Ces deux applications communiquent via les interfaces adressées 10.108.2.2 pour **pong2** et 10.109.2.4 pour **pong4**, et à travers **pong3**, jouant le rôle de routeur et instanciant un émulateur de réseaux, *netem*. Celui-ci permet de modifier le comportement aux interfaces du système, afin de mettre en place un délai, un taux de perte, des duplications et les lois de probabilité régissant l'apparition de ces événements.

Les définitions de ces paramètres et leur modification dynamique, ainsi que les démarrages des applications puits et source sont centralisés sur pong1, qui utilise le réseau de commande pour effectuer ces actions.

### 4.3.2.2 Protocoles de signalisation

ETP possède deux protocoles de signalisation pour la gestion structurelle de la composition protocolaire. Le premier est utilisé pour la synchronisation initiale et le second pour la reconfiguration.

#### Synchronisation

**Cas d'utilisation** Le protocole de synchronisation tient compte de plusieurs paramètres. Afin de l'établir, plusieurs cas d'utilisation ont été définis et sont représentés sur la figure 4.6.

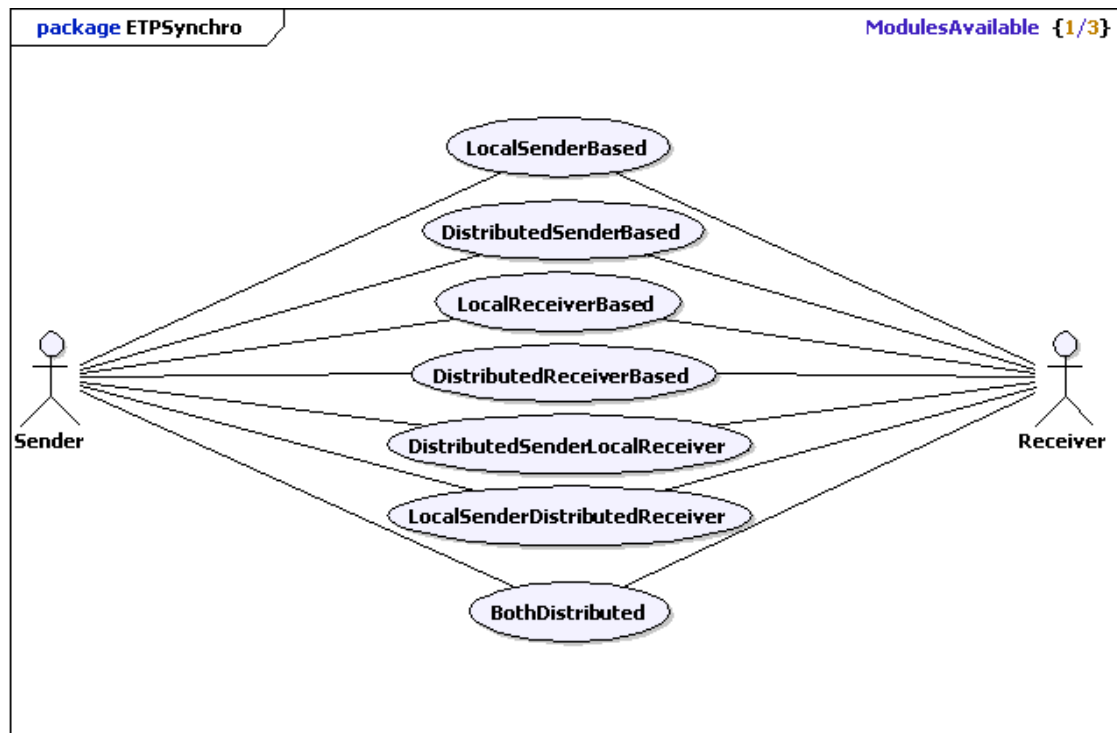


FIGURE 4.6: Cas d'utilisation du protocole de synchronisation d'ETP

ETP ne possède pas de notion de client ou de serveur. A la place, la différence de comportement s'effectue en fonction du rôle de l'application en tant qu'émetteur ou récepteur de données. Ce caractère est identifiable par les options utilisées à

l'ouverture du socket ETP. En effet, une application désirant émettre des données devra indiquer l'adresse IP à laquelle elle souhaite envoyer ces données. L'application est alors identifiée comme émettrice par ETP.

Afin de simplifier la modélisation du protocole de synchronisation, nous considérons la communication comme unidirectionnelle. Une application source, uniquement émettrice de données, souhaite envoyer des messages à une application puits, uniquement réceptrice.

Le second paramètre important est celui du caractère local ou distribué de la composition protocolaire. La composition qui est échangée entre deux entités ETP, est détaillée et précise exactement quel *Processing Module* doit prendre place dans quel conteneur, et à quel niveau de la pile de module hébergée par ce conteneur.

Avec ETP, l'application a la charge de la gestion du protocole. C'est donc elle qui doit définir cette composition de *Processing Modules*. Ainsi, une application souhaitant un service de Transport n'assurant que le minimum, à l'instar d'UDP, demandera une composition vide, n'instanciant aucun micro-protocole et donc aucun module. Une telle composition est donc considérée comme nulle. Dans le cas contraire, deux cas peuvent se poser. En effet, certains micro-protocoles nécessitent l'instanciation de *Processing Modules* de part et d'autre de la communication, et sont dits distribués. D'autres en revanche ne nécessitent de module que sur l'émetteur, ou sur le récepteur, et sont dits locaux.

Si une composition comporte au moins un micro-protocole distribué, celle-ci est également dite distribuée. Un système d'acquiescement sélectif est un exemple d'un tel micro-protocole. En effet, les acquiescements ont besoin d'être envoyés par le récepteur et traités par l'émetteur, impliquant un besoin de modules de chaque côté de la communication.

Si une composition ne comporte que des micro-protocoles locaux, celle-ci est dite locale également. Un *traffic shaper* n'utilisant aucune information explicite donnée par le récepteur est un exemple de micro-protocole local.

Les sept cas d'utilisation représentés sur la figure 4.6 prennent en compte les combinaisons de ces différents paramètres afin de déterminer le comportement du protocole. Cette liste ne se veut pas exhaustive, l'hypothèse d'une communication unidirectionnelle étant déjà restrictive, mais a pour but d'offrir une réflexion suffisamment complète permettant de dériver les autres cas de ceux présents ici.

Nous présentons le cas *Distributed Sender Based* utilisé dans cette expérience. En effet, comme décrit en 4.3.2.3, la présente expérience est basée sur une précédente étude dans laquelle ce type de déploiement est le plus adapté.

**Composition distribuée à la source, nulle au puits** Le diagramme de la figure 4.7 décrit les échanges de messages entre les entités source et puits de la

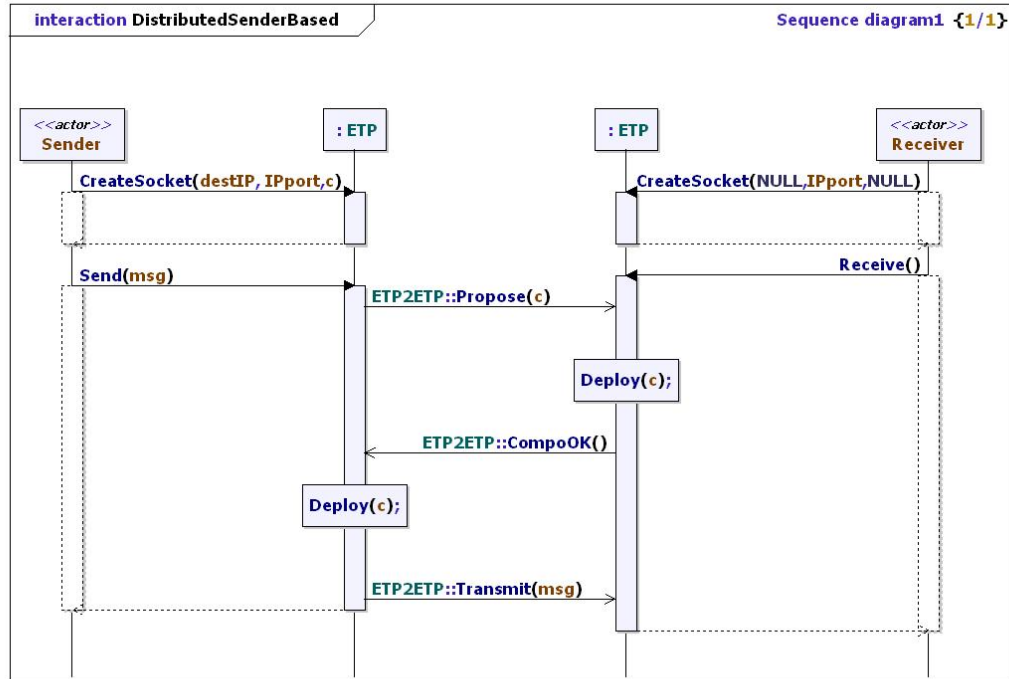


FIGURE 4.7: Diagramme de séquence du cas d'utilisation Distributed Sender Based

communication, lorsque la source prévoit l'utilisation d'une composition distribuée et le puits d'une composition nulle.

A la création du socket ETP, l'application source renseigne le protocole sur les coordonnées du puits et lui passe la composition micro-protocolaire à utiliser. Celle-ci est analysée par ETP et s'avère être distribuée, donc doit être communiquée à l'entité réceptrice. Cette dernière est prête à recevoir des messages. En effet lorsque l'application puits a ouvert son socket, celle-ci a précisé l'utilisation d'une composition nulle. L'entité ETP réceptrice est donc immédiatement disponible.

L'entité ETP émettrice va ainsi envoyer une proposition contenant la composition distribuée. L'entité réceptrice la déploie immédiatement après réception. En effet, cette dernière n'ayant aucune composition déployée peut déployer celle demandée sans conflit avec une composition déjà existante. Après déploiement, celui-ci est acquitté auprès de l'émetteur, qui effectue à son tour le déploiement. En effet, si le récepteur avait eu un conflit entre la proposition de l'émetteur et sa propre composition, il aurait dû effectuer une contre-proposition. Attendre la réponse du récepteur avant déploiement permet ainsi d'éviter un double déploiement par l'émetteur en cas de contre-proposition de la part du récepteur.

Une fois la composition instanciée de chaque côté, la communication peut commencer.

**Reconfiguration** La reconfiguration au sein d'ETP utilise le protocole de synchronisation existant. Ce dernier se déclenche lors de la demande d'émission du premier message. Le but du protocole de reconfiguration est donc de préparer ETP à effectuer une seconde synchronisation, c'est-à-dire, comme évoqué précédemment, de préparer un déploiement en parallèle d'une nouvelle composition. La figure 4.8 montre le déroulement de cet échange.

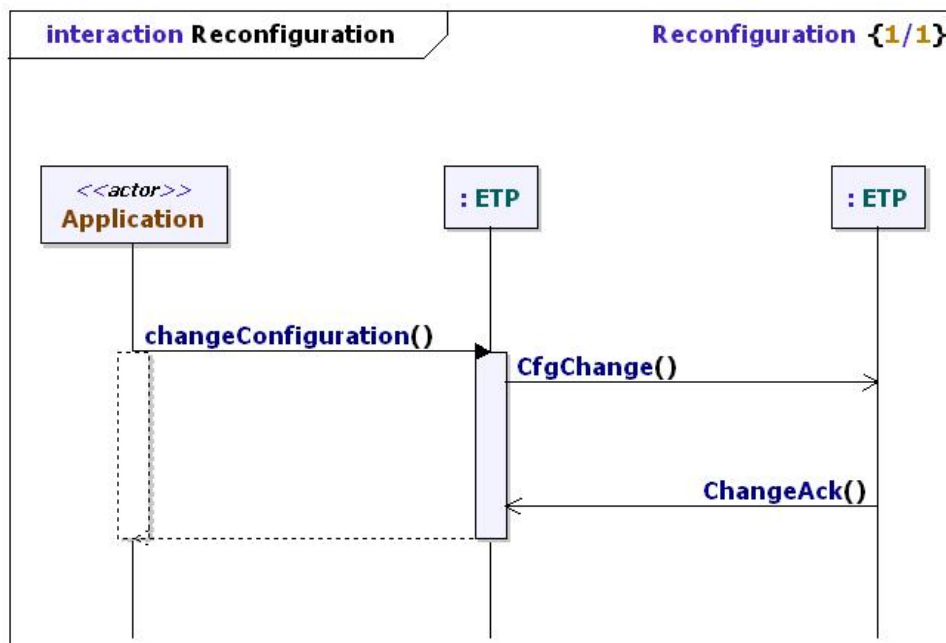


FIGURE 4.8: Préparation à la reconfiguration

L'échange se fait en deux temps. Lorsqu'une application, qu'il s'agisse de la source ou du puits, décide de changer la composition utilisée, l'entité ETP associée envoie à son pair un message indiquant qu'il doit se préparer à un changement de configuration. Une fois la nouvelle composition prête à être mise en place, un acquittement est envoyé vers l'initiateur de l'échange qui peut se préparer à son tour.

Par la suite, une synchronisation sera effectuée dans le cadre de cette nouvelle structure d'accueil, quiinstanciera la nouvelle composition en parallèle de la précédente.

**Estimation du temps de synchronisation** D'après les spécifications précédentes, on peut établir une première estimation du temps de synchronisation en fonction du RTT, donc du délai du réseau. Dans le cas d'étude décrit en 4.3.2.2, on remarque que l'échange se compose uniquement d'un message de proposition et de son acquittement, créant donc un RTT de délai. De plus, chaque système doit déployer la composition. Le système récepteur, cependant, effectue son déploiement entre la réception de la composition et l'émission de son acquittement ce qui, en toute rigueur, l'inclue dans le RTT. Enfin nous admettons que les temps de déploiement sur chacun des hôtes sont équivalents car :

- les systèmes utilisés dans le cadre de cette expérience sont les mêmes ;
- le temps de déploiement est supposé être court, voire négligeable, face au délai du réseau.

Le temps de synchronisation est donc estimé à :

$$T_s = RTT + T_d \quad (4.1)$$

où  $T_s$  représente le temps de synchronisation et  $T_d$  le temps de déploiement d'une composition sur un système.

Une reconfiguration demande tout d'abord sa préparation, décrite en 4.3.2.2, puis une synchronisation de la nouvelle composition. Le temps d'une reconfiguration sera donc :

$$T_r = T_p + T_s \quad (4.2)$$

où  $T_r$  est le temps de reconfiguration total et  $T_p$  est le temps préparation. Or :

$$T_p = RTT \quad (4.3)$$

ce qui nous permet de conclure que le temps total de reconfiguration  $T_r$  devrait s'exprimer ainsi :

$$T_r = 2RTT + T_d \quad (4.4)$$

### 4.3.2.3 Protocole expérimental

L'expérience réalisée est basée sur des mesures réalisées dans [VW09]. Un transfert de cinq minutes est réalisé, entre une application source située sur la machine



pong2 et une application puits située sur pong4, via pong3, émulant le réseau. Ce transfert doit respecter les objectifs de qualité de service suivant :

- délai maximum de 150 ms ;
- taux de pertes maximum de 10 %.

Durant ce transfert, chaque minute, les taux de pertes et délai de l'environnement réseau sont modifiés. Les travaux réalisés dans [VW09] associent à chaque environnement, une composition protocolaire pour ETP ayant la plus forte probabilité de respecter ces objectifs en qualité de service. La succession de contextes réseau et les compositions associées sont résumées sur la figure 4.9.

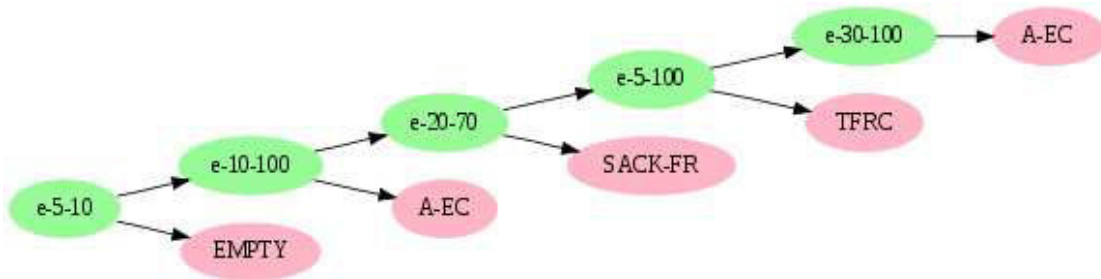


FIGURE 4.9: Caractéristiques des cinq environnements réseau et compositions associées

### Description des compositions

**EMPTY** : La composition *EMPTY* représente une composition « vide », ce qui dans le cas spécifique d'ETP correspond à offrir un service équivalent à celui d'UDP.

**SACK-FR** : Le *Selective Acknowledgement with Full Reliability* (SACK-FR), est un mécanisme de contrôle des pertes par acquittements sélectifs, équivalent à l'option SACK utilisée par le protocole TCP [Mat96]. Le mécanisme SACK disponible dans ETP permet un paramétrage du taux de pertes maximum acceptable. Nous l'utilisons ici avec une tolérance de 0 %, interdisant donc toute perte.

**A-EC** : L'*Autonomic Error Control* (A-EC) adjoint un gestionnaire autonome au SACK permettant de modifier à la volée la tolérance aux pertes en fonction d'un double objectif de taux de pertes maximum d'une part et de délai de transit maximum d'autre part, tentant d'équilibrer au mieux ces deux critères.

**TFRC** : Le mécanisme TFRC, pour *TCP Friendly Rate Control*, offre une implémentation pour ETP de l'algorithme TFRC [Flo08].

Le premier environnement, par exemple, est caractérisé par un taux de perte de 5 % et un délai de transit de 10 ms. La composition associée est une composition vide, les caractéristiques du réseau étant conformes aux objectifs en qualité de service. Après la première minute de simulation, l'environnement est modifié avec un taux de perte de 10 % et un délai de transit de 100 ms. La composition associée est alors l'A-EC, un système de reprise des pertes basé sur des acquittements sélectifs à fiabilité partielle.

L'expérience originale considérait la reconfiguration comme immédiate, les deux compositions étant déployées en même temps sur chaque système. Cependant, un temps de monitoring et de décision de 5 s était déjà présent, seul le déploiement de ces décisions étant alors considéré comme immédiat.

Le but de notre expérience est d'introduire le protocole de reconfiguration, afin de prendre en compte le temps de reconfiguration  $T_r$  effectif et de mesurer son impact sur les performances.

### 4.3.3 Résultats attendus

Les résultats attendus par l'expérience ont été extrapolés à partir des résultats originaux. Pendant le temps de monitoring de l'expérience originale, l'ancien service continue de fonctionner sur le nouvel environnement. Nous avons appliqué une régression sur ce laps de temps, et l'avons étendu durant  $T_r$  après le monitoring.

**Taux de pertes** La première métrique observée est le taux de pertes. Celui-ci a été mesuré au niveau de l'environnement réseau, et au dessus du service de Transport. Ainsi, nous pouvons observer l'impact du système de reconfiguration sur les performances du système. La figure 4.10 montre l'évolution du taux de pertes mesuré au niveau du réseau tout au long de l'expérience.

On remarquera sur la courbe 4.11 que le taux de pertes observé reste globalement sous la barre demandée des 10 %. L'introduction du système de reconfiguration ne modifie d'ailleurs que partiellement les résultats obtenus lors de l'expérience originale, comme observé sur la courbe 4.12.

En effet seul un court pic de pertes supplémentaire lors de la dernière reconfiguration est introduit dans les nouveaux résultats. Celui-ci semble être un effet de bord de notre méthode de prédiction, dû à l'application de la régression sur l'échelon observé vers 230 s, et ne devrait donc pas apparaître en situation réelle.

**Délai** La seconde métrique observée est le délai de transit des paquets. Comme pour le taux de pertes, celui-ci a été mesuré au niveau de l'environnement et du service de Transport afin d'observer l'impact du système de reconfiguration sur les performances.

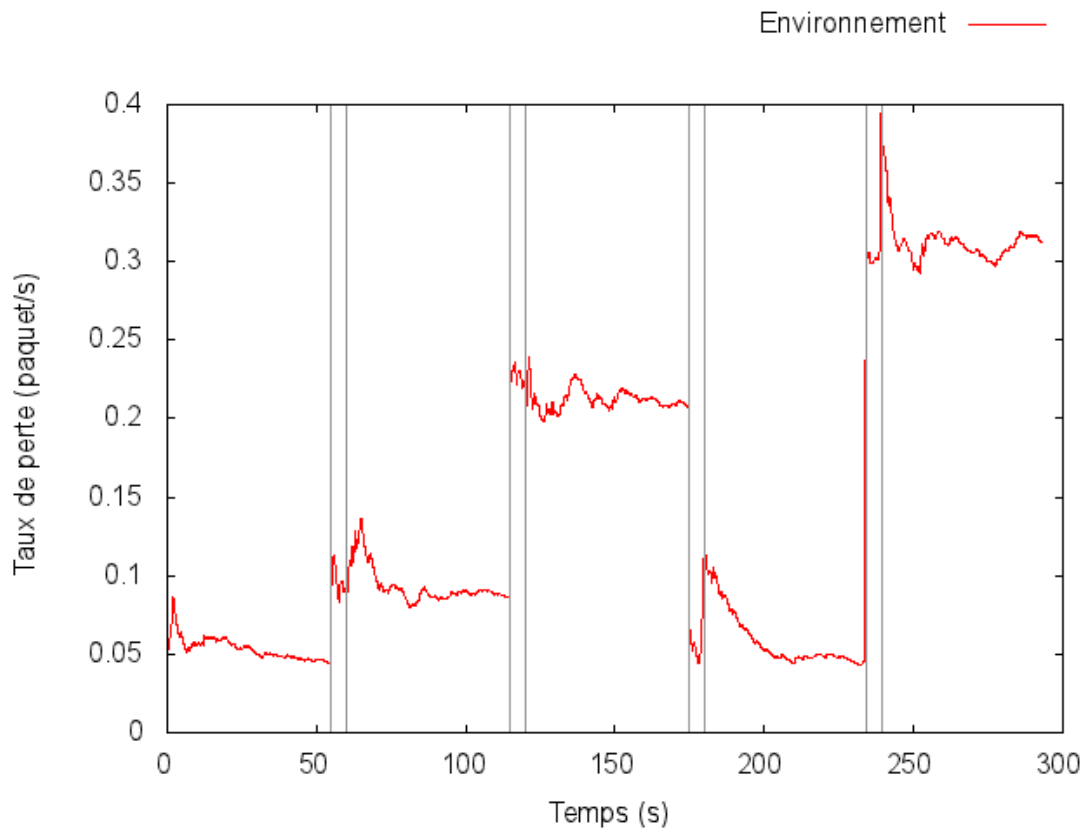


FIGURE 4.10: Taux de pertes de l'environnement

Comme pour le taux de pertes, le délai est peu impacté par le système de reconfiguration. Les prévisions de résultats de la courbe 4.14 n'indiquent en effet aucune différence par rapport aux résultats de l'expérience originale vus sur la courbe 4.15.

**Analyse** Ces résultats étaient prévisibles. En effet, l'expérience originale prévoyait un temps de monitoring et de prise de décision assez long de 5 s, devant lequel les 200 ms de déploiement maximum du système de reconfiguration sont faibles. L'*overhead* induit est donc négligeable.

Un temps de reconfiguration total long comme celui-ci est cependant acceptable du point de vue de l'ATL. En effet, une telle reconfiguration n'aura lieu qu'initiée par l'*Autonomic Manager*. Or, les décisions de ce dernier ont trait à la gestion de la communication, le contrôle étant laissé à la charge des composants protocolaires ; les actions de gestion sont de plus entreprises avec une granularité temporelle longue. L'impact sur les performances, induit par cette reconfiguration sera donc

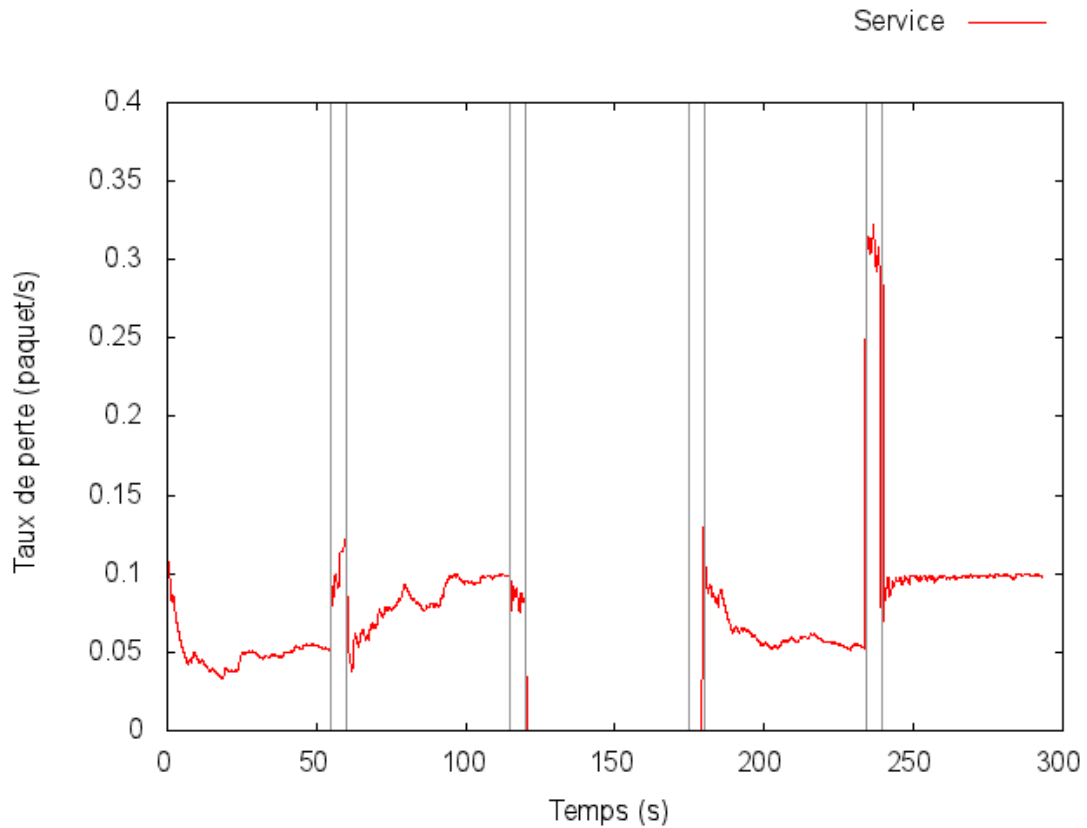


FIGURE 4.11: Taux de pertes vu par le service de Transport

suffisamment espacé dans le temps pour limiter la dégradation moyenne à long terme de la communication.

#### 4.3.4 Conclusion

Dans cette section nous avons observé le comportement du protocole de synchronisation d'ETP dont les caractéristiques structurelles sont proches de l'ATL. Celui-ci nous permet d'observer une tendance des dégradations de performances induites dans des systèmes tels que l'ATL. Bien que les résultats soient prometteurs, ceux-ci devront être testés sur un prototype réel, et étendus afin d'observer plus en détails les relations entre temps de reconfiguration et fréquence de reconfiguration, les variations en fonction de type d'environnement ou l'impact des caractéristiques du réseau sur une reconfiguration donnée.

Ce type d'observations nous permettra alors d'enrichir la connaissance des *Autonomic Manager* de l'ATL afin de leur permettre un choix plus fin dans les multiples décisions auxquelles ils auront à faire face. En effet, les décisions prises

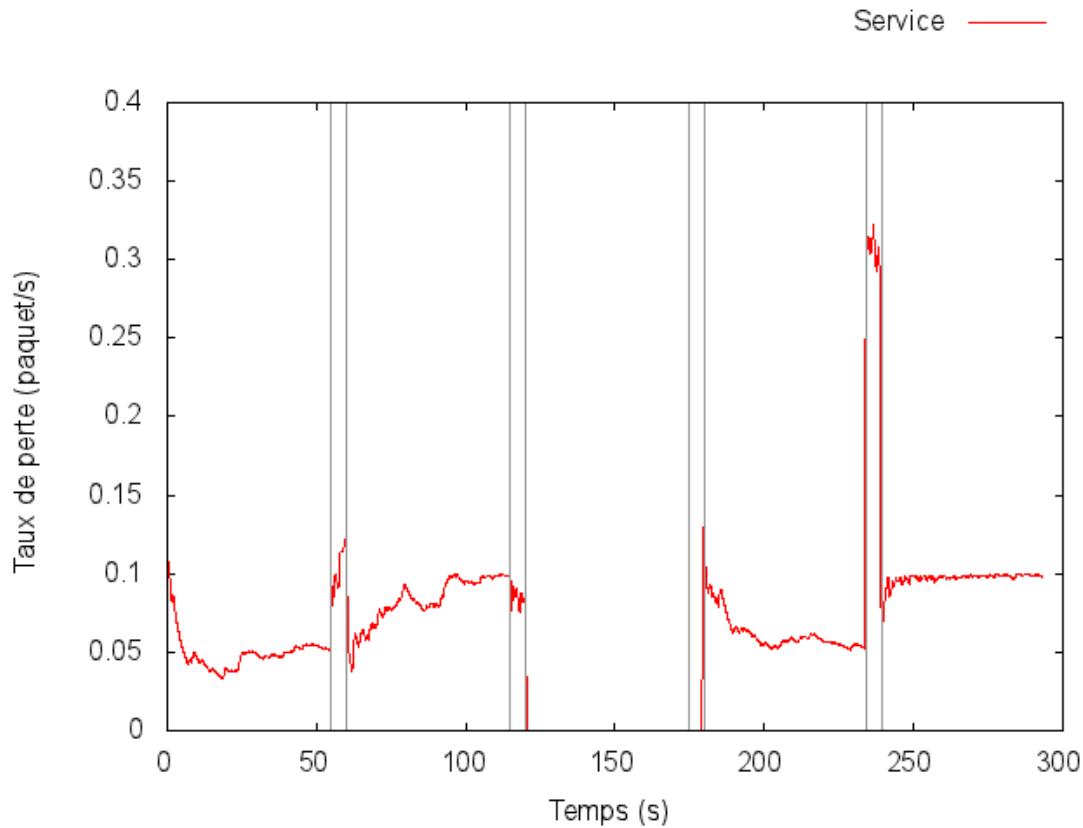


FIGURE 4.12: Taux de pertes vu par le service sans système de reconfiguration

dans un contexte idéal peuvent avoir à être modifiées en fonction du contexte réel. L'étude des relations entre les différentes métriques et les différentes situations permettront de prendre ces cas en compte.

## 4.4 Conclusion

Ce chapitre a montré par l'expérience l'intérêt que peut avoir un système tel que l'ATL. La première partie se base sur MPTCP pour montrer l'intérêt de composer des micro-protocoles avec des protocoles existants afin d'élargir le service offert par ces derniers sans nécessairement les modifier. Elle a en outre permis de mettre en avant le besoin d'autonomie dans la prise de décisions concernant la configuration et le paramétrage de chaque composant, celui-ci se révélant bien trop lourd pour l'utilisateur ou l'administrateur au cas par cas.

La deuxième partie étend la première en introduisant le concept d'adaptation structurelle dynamique. En effet, ici la composition protocolaire change en cours

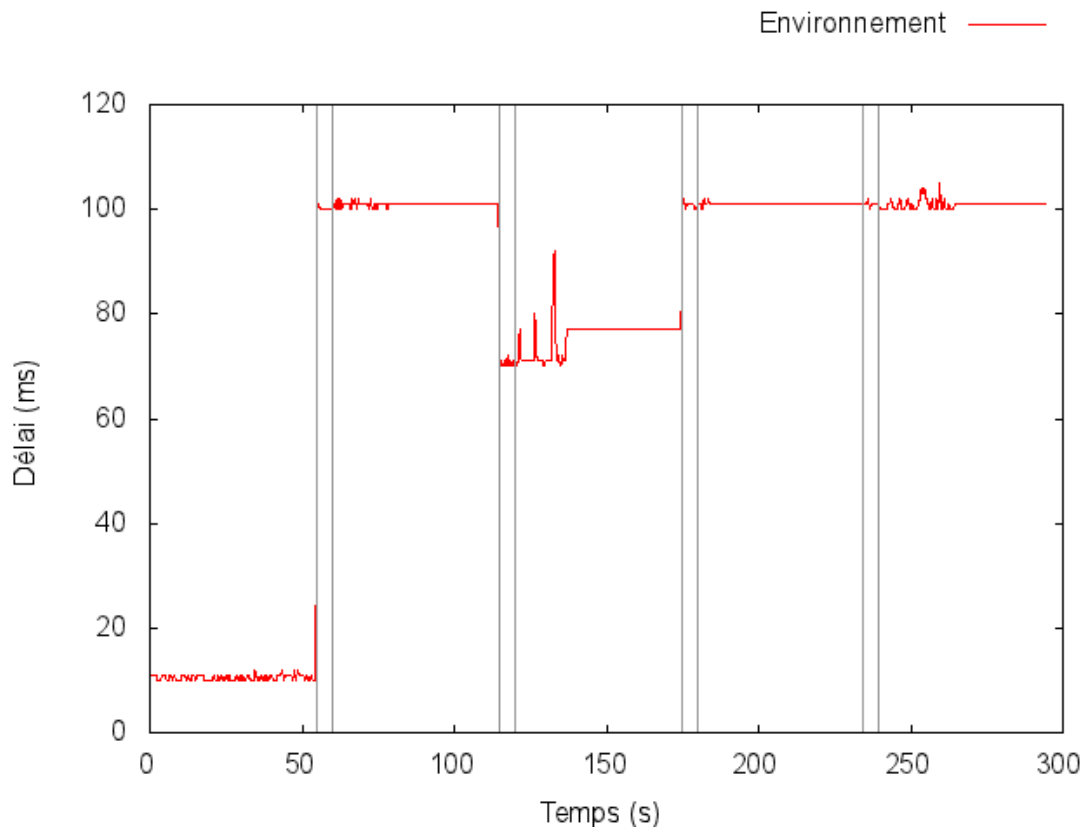


FIGURE 4.13: Délai induit par l'environnement

de communication pour s'adapter aux variations de l'environnement. Les gains tirés de cette adaptation étant positifs, nous remarquons tout de même que de tels changements sont complexes et doivent se faire sans intervention extérieure à la couche de Transport, ce qui implique à nouveau un besoin d'autonomie.

La troisième partie se concentre sur le coût d'un système de reconfiguration. En effet, un système comme l'ATL doit disposer d'un protocole de signalisation adéquat permettant la reconfiguration dynamique. L'expérience ayant été soumise à des difficultés, les résultats présentés sont déduits du modèle théorique du coût du protocole de signalisation. On observe que ces résultats sont acceptables et permettent de respecter la qualité de service demandée par l'application.

Des expériences plus poussées seront nécessaires sur un prototype concret de l'ATL, afin notamment d'observer les limites de celui-ci ainsi que les améliorations à y apporter. Celles-ci devront être complètes et poussées afin d'observer les effets de toutes les interactions possibles des multiples paramètres pouvant varier dans un système de cette complexité.

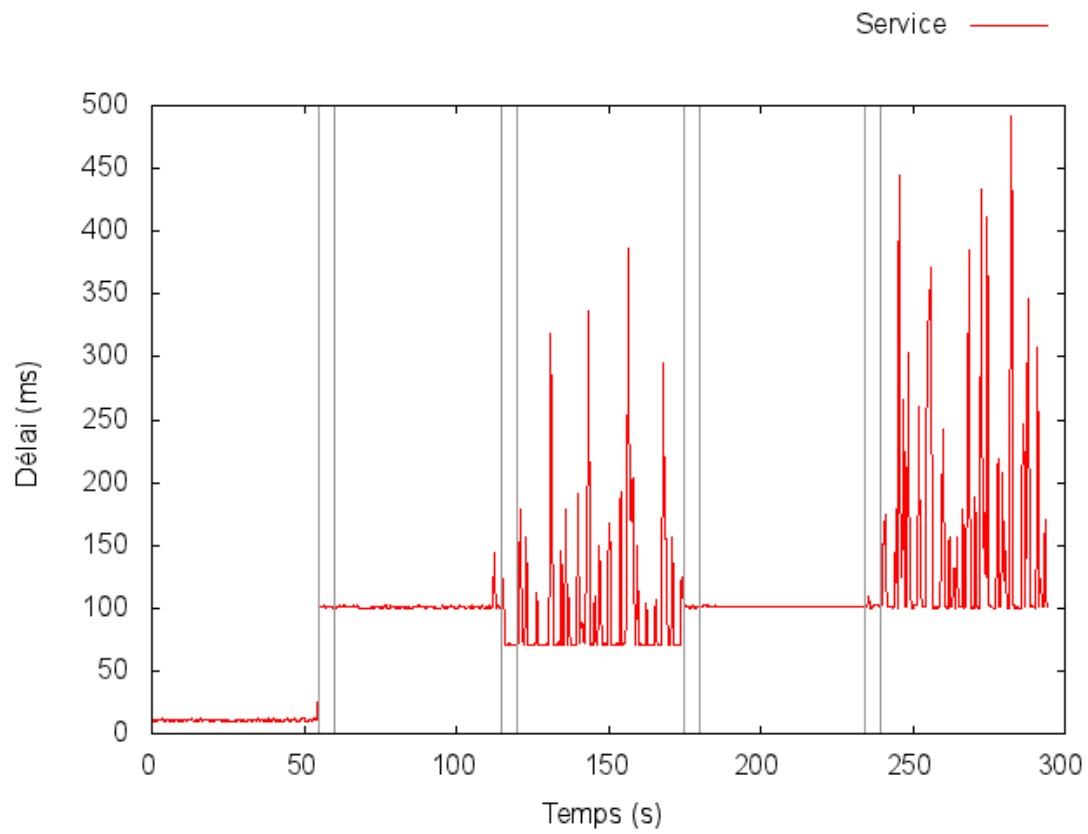


FIGURE 4.14: Délai vu par le service

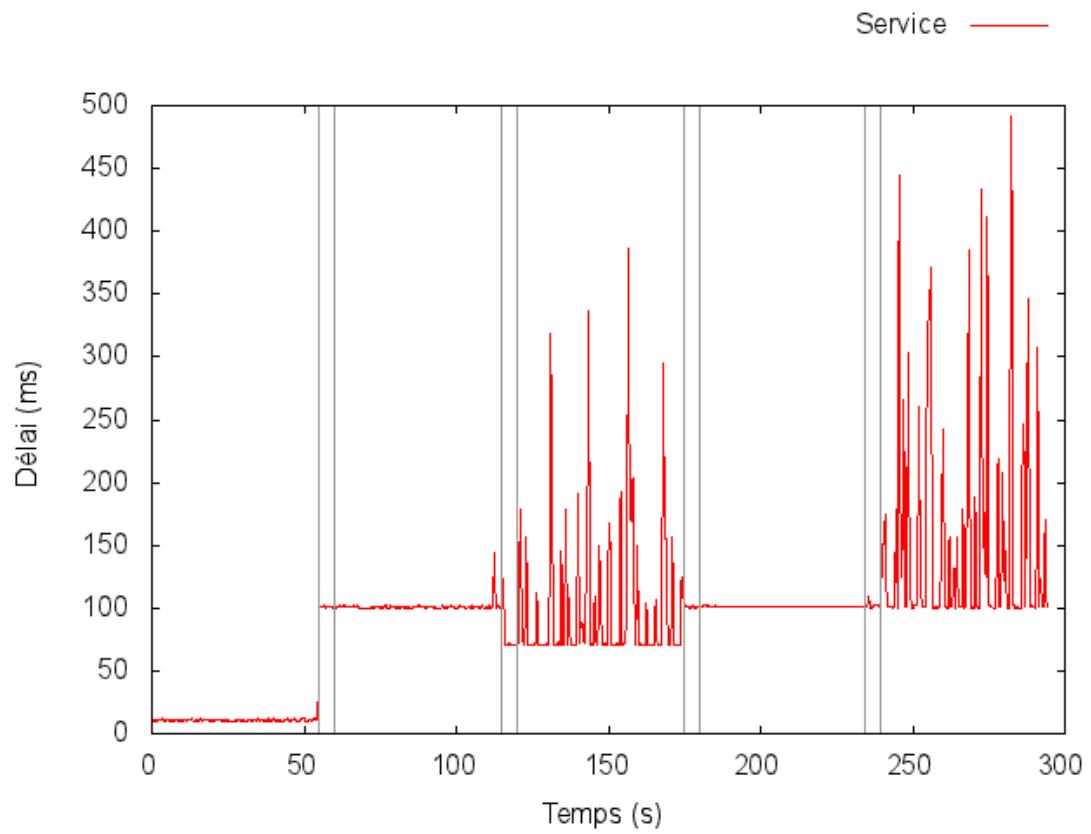


FIGURE 4.15: Délai vu par le service sans système de reconfiguration



# Conclusion générale

## Rappel des contributions

La thèse soutenue dans ce manuscrit part du constat que l'évolution majeure de l'Internet, de ses applications, des équipements terminaux et des réseaux physiques sous jacents, ont conduit à une multiplication des propositions d'évolution ou d'adaptation des protocoles de Transport.

Le chapitre 1 a décrit ce contexte et en a déduit cinq points de problématique relatifs à la complexité de choix et d'utilisation des protocoles, à la configurabilité des protocoles, à l'extensibilité des protocoles, à l'assujettissement des applications aux solutions protocolaires et au déploiement de nouveaux protocoles.

Ces différents problèmes nous ont ensuite conduits à analyser les limites des solutions de Transport actuelles face au contexte moderne et de poser les bases d'une redéfinition de la couche Transport, l'*Autonomic Transport Layer* (ATL). L'approche soutenue est que cette dernière doit offrir aux applications supportées des points d'accès génériques leur permettant de requérir des services et non des solutions protocolaires. Les services requis sont ensuite réalisés via une composition protocolaire dont le choix revient à l'ATL de manière autonome et dynamique. Une telle couche Transport de ce type permet également l'intégration progressive et transparente de nouvelles solutions protocolaires ainsi que la combinaison de plusieurs mécanismes en un seul service composite afin de satisfaire le plus finement possible les souhaits exprimés par l'application.

Le chapitre 2 a posé les bases de la structure de l'ATL. Tout d'abord, nous avons posé les objectifs du système et les paradigmes des architectures orientées service et basées composants sur lesquels reposent la construction de l'ATL. Nous avons ensuite défini une classification concernant les types d'actions à considérer (gestion, contrôle et données), les applications à prendre en compte (*legacy*, *ATL-aware* et *smart*) et les systèmes avec lesquels interagir (*legacy* et *ATL-enabled*). Ces classifications nous ont contraint dans la construction de l'ATL pour à aboutir une solution complète pouvant s'intégrer de manière transparente dans le paysage actuel tout en permettant une évolution future souple. Le rôle des différents com-

posants constituant l'ATL ainsi que leur organisation les uns par rapport aux autres ont finalement été décrits.

Le chapitre 3 a complété la description établie dans le chapitre 2 en détaillant le comportement des différents éléments de l'ATL dans deux cas d'utilisations particuliers. Le premier cas d'utilisation concerne l'ouverture d'un point d'accès au service de Transport. Ce type d'action diffère de l'ouverture d'un socket classique. Il doit en effet prendre en compte le fait que l'application l'effectuant ainsi que l'application distante à contacter peuvent être *legacy*, *ATL-aware* ou *smart* et que cette dernière peut être hébergée sur un système *legacy* ou *ATL-enabled*. Nous avons décrit les différentes phases de découverte et de négociation des services qui doivent avoir lieu pour mettre en place le service composite adéquat. Dans le second cas d'utilisation, nous avons décrit le comportement de l'ATL lors d'un changement de contexte en cours de communication. Le redéploiement du service composite durant le transfert des données pose en effet des problèmes de continuité du service rendu et du coût de ce changement. Nous avons clos le chapitre par des considérations sur la traduction de l'état de l'ancien service vers le nouveau en prenant l'exemple de la problématique de retransmission des messages initialement émis avant une reconfiguration.

Le chapitre 4 a présenté différents résultats obtenus au cours d'expériences menées en simulation et en émulation. La première expérience, menée en simulation, a combiné *Multipath* TCP (MPTCP) à des mécanismes génériques orientés QoS afin d'améliorer la qualité d'un transfert vidéo interactif en altérant le comportement du protocole sans modifier l'implémentation de celui-ci. Par cette expérience, nous avons cherché à démontrer l'intérêt de l'utilisation de services composites utilisant des mécanismes génériques avec des protocoles traditionnels. Les résultats étant prometteurs, nous avons poursuivi ce travail par de nouvelles simulations portant sur l'adaptation dynamique de tels services. Le but était ici de montrer le gain perçu par le changement de composition en réponse aux changements de conditions réseaux en cours de communication. Les gains induits nous ont amené à nous interroger sur le coût induit par la signalisation inhérente à de telles actions. La troisième expérience a tenté d'établir une estimation de ce coût en observant l'impact du protocole de synchronisation utilisé dans le contexte d'une implémentation d'un protocole (ETP) aux principes proches de l'ATL. Cette étude a permis de conclure que ce coût était négligeable face à la fréquence faible de mise en œuvre des actions de gestion pour lesquelles cette signalisation est nécessaire.

## Perspectives

De nombreuses perspectives de travail peuvent être envisagées à partir du travail décrit dans ce document. Nous les décrivons ci-après.

---

## Flux collaboratifs

La première possibilité d'extension de ce travail consiste à l'intégrer dans la problématique des flux collaboratifs. Ce concept consiste à établir une hiérarchie d'importance dans les différents flux au sein d'une machine voire d'un réseau que plusieurs flux partagent, afin que chacun se fixe potentiellement une borne supérieure sur différents critères de qualité de service.

Prenons l'exemple d'un réseau local domestique sur lequel un ordinateur de bureau télécharge via BitTorrent la dernière version d'un système Linux pendant qu'un ordinateur de salon diffuse un film obtenu en vidéo à la demande sur un service de *streaming*. Actuellement, le risque est que les différents flux BitTorrent non seulement saturent le flux entrant de l'ordinateur qui effectue le téléchargement, mais saturent également l'accès Internet de tout le réseau domestique, dégradant du même coup le visionnage en *streaming*.

Le fait d'utiliser le paradigme de l'*Autonomic Computing* pour gérer chaque flux permet d'exposer une interface de contrôle similaire à celle exposée par le *Touchpoint* afin de diriger l'*Autonomic Manager* (cf. 2.2.3.2). Cette interface, notamment utilisée par les applications *smart* ou les applications de gestion réservées aux utilisateurs et administrateurs du système, permettrait également à un *Autonomic Manager* hiérarchiquement supérieur d'exprimer des préférences découlant d'une priorisation des flux sur la machine mais également sur le réseau. D'une telle utilisation de cette interface découlerait alors une répartition automatique des ressources réseau afin de donner la priorité à certains flux sur les différents critères de qualité de service. Dans notre exemple, le téléchargement pourrait automatiquement se limiter afin de permettre aux autres applications s'exécutant sur l'ordinateur d'avoir accès au réseau, mais également au flux vidéo d'être diffusé sans accroc.

On peut également imaginer que de telles mesures puissent être mises en œuvre par des *Autonomic Manager* dont la responsabilité se porte sur la gestion de l'énergie du système, par exemple.

**M2M** Le LAAS-CNRS et notamment le groupe SARA, travaille sur les problématiques de communication entre objets intelligents, en particulier dans le contexte du *machine to machine*, ou M2M, dans lequel la problématique des flux collaboratifs peut s'avérer prometteuse. En effet un réseau domestique faisant cohabiter des systèmes tels que les ordinateurs actuels, des capteurs et des robots, pourrait avantageusement tirer parti d'une répartition dynamique des ressources selon les flux.

## Implémentation

Dans le chapitre 4 nous avons mentionné l'absence de prototype de l'ATL comme première limite aux expérimentations possibles. Cette limite doit être levée. Un travail de précision du modèle dans une optique d'implémentation est obligatoire et prioritaire si nous voulons faire de l'ATL un projet sur lequel mener des études approfondies sur des problématiques complexes. Ce travail doit respecter deux objectifs.

Le premier est un objectif de recherche. L'implémentation de l'ATL doit respecter son caractère modulable afin de permettre l'étude séparée de chacune de ses fonctionnalités et son influence sur les autres sans modification de ces dernières. Ainsi, l'étude de la prise de décision de la fonction de *Planning* au sein de l'*Autonomic Manager* pourrait être menée en parallèle de celle sur le *Monitoring*, chaque étude ne modifiant que la fonctionnalité sur laquelle elle porte sans devoir se préoccuper de l'autre.

Le second objectif porte sur les performances et l'adoption du modèle. En effet, afin que celui-ci gagne en visibilité et qu'il suscite l'intérêt dans la communauté internationale, l'implémentation du modèle doit être faite de manière réaliste. Le prototype doit donc être spécifié et implémenté dans le but d'intégrer parfaitement un système d'exploitation comme un projet de production et pas uniquement d'étude. Motiver l'intérêt et l'adoption permettra d'ouvrir ce projet et favorisera la collaboration avec d'autres équipes ce qui lui apportera un poids supplémentaire et multipliera les études à son sujet.

## Décision et sécurité

La prise de décision et la sécurité de l'ATL sont les deux études principales à mener.

Nous avons vu au sein de ce document que le nombre de paramètres qui doivent être pris en compte dans les décisions de chaque *Autonomic Manager* peut être important. Une étude poussée sur l'incidence de ces paramètres sur les performances s'avère ainsi pertinente. Le système et l'algorithme de décision en lui-même, sujets non abordés dans le présent document, sont également des points indispensables à étudier et définir afin de donner une réalité à l'ATL.

La sécurité représente l'autre point essentiel. Si l'ATL veut devenir un modèle valable, elle ne peut se permettre de représenter une faille dans un système d'exploitation. La sécurisation de l'architecture et des échanges qu'elle effectue ainsi que sa tolérance aux fautes représentent donc deux axes d'étude majeurs qui doivent être pris en compte afin de permettre à l'ATL d'évoluer et devenir un modèle dont la réalisation et l'intégration dans le paysage informatique se concrétiseraient.

## Publications de l'auteur

### A.1 Revue internationale avec actes et comité de lecture

- [1] C. Diop, G. Dugué, C. Chassot, E. Exposito, and J. Gomez. Qos-aware and autonomic-oriented multipath tcp extensions for mobile and multimedia applications. *International Journal of Pervasive Computing and Communications*, 8(4) :306–328, 2012.

### A.2 Conférences et workshop internationaux avec actes et comité de lecture

- [2] C. Diop, G. Dugué, C. Chassot, and E. Exposito. Qos-aware multipath-tcp extensions for mobile and multimedia applications. In *10th International Conferences on Advances in Mobile Computing and Multimedia (MoMM 2011)*, MoMM '11, pages 139–146, New York, NY, USA, 2011. ACM.
- [3] C. Diop, G. Dugué, C. Chassot, and E. Exposito. Qos-oriented mptcp extensions for multimedia multi-homed systems. In *2nd International Workshop on Protocols and Applications with Multi-Homing Support (PAMS 2012)*, WAINA '12, pages 1119–1124, Washington, DC, USA, 2012. IEEE Computer Society.
- [4] G. Dugué, C. Diop, C. Chassot, and E. Exposito. Towards autonomic multipath transport for infotainment-like systems. In *3rd IEEE*

- International Workshop on Smart Communications in Network Technologies (SaCoNeT 2012)*, pages 6453–6457, June 2012.
- [5] C. Diop, J. Gomez-Montalvo, G. Dugué, C. Chassot, and E. Exposito. Towards a semantic and mptcp-based autonomic transport protocol for mobile and multimedia applications. In *3rd International Conference on Multimedia Computing and Systems, May 10-12, 2012 (ICMCS 2012)*, pages 496–501, May 2012.
- [6] G. Dugué, M. Oulmahdi, and C. Chassot. Design principles of a service-oriented and component-based autonomic transport layer. In *4th Track on Adaptive and Reconfigurable Service-oriented and component-based Applications and Architectures*, June 2014.
- [7] M. Oulmahdi, G. Dugué, and C. Chassot. Towards a service-oriented and component-based transport layer. In *IEEE 5th International Conference on Smart Communications in Network Technologies (SaCoNeT 2014)*, June 2014.

# Acronymes

**A-EC** *Autonomic Error Control.*

**AC** *Autonomic Computing.*

**API** *Application Programming Interface.*

**ATL** *Autonomic Transport Layer.*

**CTP** *Configurable Transport Protocol.*

**DCCP** *Datagram Congestion Control Protocol.*

**ETP** *Enhanced Transport Protocol.*

**MPTCP** *Multipath TCP.*

**PNSR** *Peak Signal to Noise Ratio.*

**QoS** *Qualité de Service (Quality of Service).*

**SACK** *Selective Acknowledgement.*

**SACK-FR** *Selective Acknowledgement with Full Reliability.*

**SCTP** *Stream Control Transmission Protocol.*

**TCP** *Transmission Control Protocol.*

**Tng** *Transport Next Generation.*

**UDP** *User Datagram Protocol.*





# Bibliographie

- [AGM<sup>+</sup>10] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). *SIGCOMM Comput. Commun. Rev.*, 40(4) :63–74, August 2010.
- [BP95] L.S. Brakmo and L.L. Peterson. Tcp vegas : end to end congestion avoidance on a global internet. *Selected Areas in Communications, IEEE Journal on*, 13(8) :1465–1480, 1995.
- [BWH<sup>+</sup>07] Patrick G. Bridges, Gary T. Wong, Matti Hiltunen, Richard D. Schlichting, and Matthew J. Barrick. A configurable and extensible transport protocol. *IEEE/ACM Trans. Netw.*, 15(6) :1254–1265, December 2007.
- [Car02] B. Carpenter. Middleboxes : Taxonomy and Issues. RFC 3234, RFC Editor, February 2002.
- [CF04] Carlo Caini and Rosario Firrincieli. Tcp hybla : a tcp enhancement for heterogeneous networks. *INTERNATIONAL JOURNAL OF SATELLITE COMMUNICATIONS AND NETWORKING*, 22, 2004.
- [Con94] Connolly, T. and Amer, P. and Conrad, P. An Extension to TCP : Partial Order Service. RFC 1693, RFC Editor, November 1994. Obsoleted by RFC 6247.
- [DABR12] Thomas Dreibholz, Hakim Adhari, Martin Becke, and Erwin Paul Rathgeb. Simulation and Experimental Evaluation of Multipath Congestion Control Strategies. In *Proceedings of the 2nd International Workshop on Protocols and Applications with Multi-Homing Support (PAMS)*, Fukuoka/Japan, March 2012. ISBN 978-0-7695-4652-0.
- [DBA13] Thomas Dreibholz, Martin Becke, and Hakim Adhari. SCTP Socket API Extensions for Concurrent Multipath Transfer. Internet Draft Version 06, IETF, Network Working Group, July 2013. draft-dreibholz-tsvwg-sctpsocket-multipath-06.txt, work in progress.

- [Die08] Dierks, T. and Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, RFC Editor, August 2008.
- [Egg11] Eggert, L. Moving the Undeployed TCP Extensions RFC 1072, RFC 1106, RFC 1110, RFC 1145, RFC 1146, RFC 1379, RFC 1644, and RFC 1693 to Historic Status. RFC 6247, RFC Editor, May 2011.
- [ETS13] ETSI. Digital video broadcasting (dvb) ; second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications (dvb-s2). Technical report, European Telecommunications Standards Institute, March 2013.
- [Exp03] Ernesto Exposito. *Specification and implementation of a QoS oriented transport protocol for multimedia applications*. PhD thesis, Université de Toulouse, Institut National Polytechnique de Toulouse, 2003.
- [FI08] Bryan Ford and Janardhan Iyengar. Breaking up the transport logjam. In *In HOTNETS*, 2008.
- [Flo03] Floyd, S. HighSpeed TCP for Large Congestion Windows. RFC 3649, RFC Editor, December 2003.
- [Flo06a] Floyd, S. and Kohler, E. Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2 : TCP-like Congestion Control. RFC 4341, RFC Editor, March 2006.
- [Flo06b] Floyd, S. and Kohler, E. and Padhye, J. Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3 : TCP-Friendly Rate Control (TFRC). RFC 4342, RFC Editor, March 2006.
- [Flo07] Floyd, S. and Kohler, E. TCP Friendly Rate Control (TFRC) : The Small-Packet (SP) Variant. RFC 4828, RFC Editor, April 2007.
- [Flo08] Floyd, S. and Handley, M. and Padhye, J. and Widmer, J. TCP Friendly Rate Control (TFRC) : Protocol Specification. RFC 5348, RFC Editor, September 2008.
- [Flo09] Floyd, S. and Kohler, E. Profile for Datagram Congestion Control Protocol (DCCP) Congestion ID 4 : TCP-Friendly Rate Control for Small Packets (TFRC-SP). RFC 5622, RFC Editor, August 2009.
- [For11] Ford, A. and Raiciu, C. and Handley, M. and Barre, S. and Iyengar, J. Architectural Guidelines for Multipath TCP Development. RFC 6182, RFC Editor, March 2011.
- [Fox89] Fox, R. TCP Big Window and Nak Options. RFC 1106, RFC Editor, June 1989. Obsoleted by RFC 6247.

- [GMPG00] T. Goff, J. Moronski, D.S. Phatak, and V. Gupta. Freeze-tcp : a true end-to-end tcp enhancement mechanism for mobile environments. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1537–1545 vol.3, 2000.
- [Hen12] Henderson, T. and Floyd, S. and Gurtov, A. and Nishida, Y. The NewReno Modification to TCP’s Fast Recovery Algorithm. RFC 6582, RFC Editor, April 2012.
- [HRX08] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic : a new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5) :64–74, July 2008.
- [IEE12a] IEEE. Ieee standard for ethernet. *IEEE Std 802.3-2012 (Revision to IEEE Std 802.3-2008)*, pages 1–0, 2012.
- [IEE12b] IEEE. Ieee standard for information technology – telecommunications and information exchange between systems local and metropolitan area networks – specific requirements part 11 : Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pages 1–2793, 2012.
- [ITU99a] ITU. G.992.1 : Asymmetric digital subscriber line (adsl) transceivers. Technical report, International Telecommunication Union, July 1999.
- [ITU99b] ITU. G.992.2 : Splitterless asymmetric digital subscriber line (adsl) transceivers. Technical report, International Telecommunication Union, July 1999.
- [ITU07] ITU. G.707 : Network node interface for the synchronous digital hierarchy (sdh). Technical report, International Telecommunication Union, January 2007.
- [ITU11a] ITU. G.1010 : End-user multimedia qos categories. Technical report, International Telecommunication Union, January 2011.
- [ITU11b] ITU. G.993.2 : Very high speed digital subscriber line transceivers 2 (vdsl2). Technical report, International Telecommunication Union, December 2011.
- [Jac88] Jacobson, V. and Braden, A. TCP Extensions for Long-Delay Paths. RFC 1072, RFC Editor, October 1988. Obsoleted by RFC 1323, RFC 2018, RFC 6247.
- [Kho06] Kholer, E. and Handley, M. and Floyd, S. Datagram Congestion Control Protocol (DCCP). RFC 4340, RFC Editor, March 2006.

- [KRW03] Jirka Klaue, Berthold Rathke, and Adam Wolisz. Evalvid - a framework for video transmission and quality evaluation. In *In Proc. of the 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 255–272, 2003.
- [LBL89] M.M. Lara-Barron and G.B. Lockhart. Selective discarding procedure for improved tolerance to missing voice packets. *Electronics Letters*, 25(19) :1269–1271, Sept 1989.
- [Mat96] Mathis, M. and Mahdavi, J. and Floyd, S. and Romanow, A. TCP Selective Acknowledgment Options. RFC 2018, RFC Editor, October 1996.
- [Mil10] Mills, D. and Martin, J. and Burbank, J. and Kasch, W. Network Time Protocol Version 4 : Protocol and Algorithms Specification. RFC 5905, RFC Editor, June 2010.
- [Nag84] J. Nagel. Congestion Control in IP/TCP Internetworks. RFC 896, RFC Editor, January 1984.
- [Nis] Yoshifumi Nishida. Ns2 implementation for mtcp. <https://code.google.com/p/multipath-tcp/>.
- [Phe12] Phelan, T. and Fairhurst, G. and Perkins, C. DCCP-UDP : A Datagram Congestion Control Protocol UDP Encapsulation for NAT Traversal. RFC 6773, RFC Editor, November 2012.
- [Pos80] J. Postel. User Datagram Protocol. RFC 768, RFC Editor, August 1980.
- [Pos81] J. Postel. Transmission Control Protocol. RFC 793, RFC Editor, September 1981.
- [Pre02] Presuhn, R. and Case, J. and McCloghrie, K. and Rose, M. and Wald-busser, S. Management Information Base (MIB) for the Simple Network Management Protocol (SNMP). RFC 3418, RFC Editor, December 2002.
- [Sch13] Scharf, M. and Ford, A. Multipath TCP (MPTCP) Application Interface Considerations. RFC 6897, RFC Editor, March 2013.
- [STBT08] M. Sridharan, K. Tan, D. Bansal, and D. Thaler. Compound TCP : A New TCP Congestion Control for High-Speed and Long Distance Networks. Internet Draft Version 02, IETF, Network Working Group, November 2008. draft-sridharan-tcpm-ctcp-02.txt, expired.

- [Ste04] Stewart, R. and Ramalho, M. and Xie, Q. and Tuexen, M. and Conrad, P. Stream Control Transmission Protocol (SCTP) Partial Reliability Extension. RFC 3758, RFC Editor, May 2004.
- [Ste07] Stewart, R. and Ed. Stream Control Transmission Protocol. RFC 4960, RFC Editor, September 2007.
- [VW09] Nicolas Van Wambeke. *Une approche pour la composition autonome de services de communication orientés QoS*. PhD thesis, Université de Toulouse, 2009.
- [WJLH06] D.X. Wei, Cheng Jin, S.H. Low, and S. Hegde. Fast tcp : Motivation, architecture, algorithms, performance. *Networking, IEEE/ACM Transactions on*, 14(6) :1246–1259, 2006.
- [XHR04] Lisong Xu, K. Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2514–2524 vol.4, 2004.