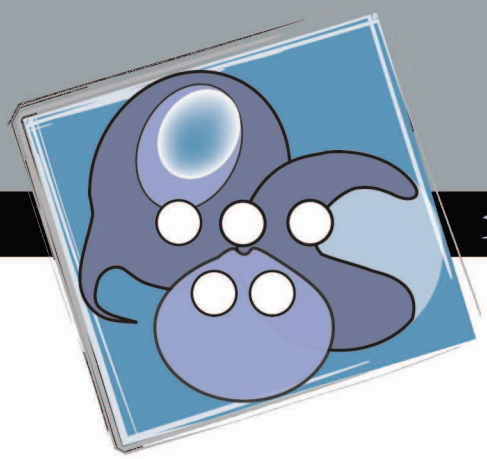


rapport de projet "tutoré" 2005

INSA TOULOUSE

Nicolas Van Wambeke

Digitally signed by Nicolas Van Wambeke
DN: CN = Nicolas Van Wambeke, SN = Van Wambeke, G = Nicolas
Reason: I am the co-author of this document
Location: Toulouse, FRANCE
Date: 2005.07.16 23:09:39 +01'00'

Introduction à la qualité de service sur Internet
Application de visioconférence sur un réseau IntServ/DiffServ

INSA Toulouse 4°année RT

Van Wambeke Nicolas, Molinier Nicolas, Fok Frederic &
Richasse Nicolas, Martelli Nicolas

Tuteurs: M. Chassot Christophe & M. Auriol Guillaume

rapport de projet “tutoré” 2005

Introduction à la qualité de service sur Internet
Application de visioconférence sur un réseau IntServ/DiffServ

INSA Toulouse 4°année RT

Van Wambeke Nicolas, Molinier Nicolas, Fok Frederic &
Richasse Nicolas, Martelli Nicolas

Tuteurs: M. Chassot Christophe & M. Auriol Guillaume

De nombreuses personnes ont contribué à la réalisation de ce projet.
Nous tenons à remercier en particulier :

Messieurs Christophe Chassot et Guillaume Auriol pour les nombreuses heures passées avec nous afin de nous aiguiller et nous permettre d'éclairer les notions parfois obscures qui régissent le fonctionnement des systèmes abordés

SOMMAIRE

INTRODUCTION

La montée en puissance d'Internet et l'augmentation de capacité des micro-ordinateurs a permis à de nouvelles applications multimédia de voir le jour. La visioconférence sur Internet (l'IP Conferencing) fut le fruit des développements de laboratoires de recherche dans l'objectif de réaliser des outils de travail coopératif entre scientifiques internationaux. A ses débuts, la visioconférence était difficilement portable mais sont intérêt s'est révélé avec l'arrivée d'Internet pour le grand public. L'un des premiers logiciels à utiliser la visiophonie sur ordinateur a été CU-Seeme de l'université de Cornell aux Etats-Unis. La qualité de l'image restait médiocre et la compression MJPEG utilisée n'était pas adaptée aux réseaux bas débits. Malgré l'évolution des normes (la ratification de la norme H323 en 1996, extension de H320) et l'augmentation des bandes passantes, Internet n'est pas dimensionné pour la visioconférence. Des problèmes, tels que la congestion, montrent l'insuffisance des protocoles existants pour ce genre d'applications. Afin de gérer le transport de données multimédia, de nouveaux protocoles Temps Réel doivent être mis en place. Il s'agit des protocoles RTP (Real Time Protocol) et RSVP (Ressources Reservation Protocole). Toutefois, la pénurie en bande passante reste toujours un problème.
Nous avons ainsi été amené à une première approche de la qualité de service sur Internet en travaillant sur une plateforme hétérogène composés d'un domaine IntServ et d'un domaine DiffServ (techniques proposant des mécanismes de garantie de la Qualité de Service différentes).

L'objectif de ce projet tutoré, sous la tutelle de Messieurs C.Chassot et G.Auriol, est la mise–en place d'un protocole de communication et la gestion de la réservation de ressources dans le cadre d'une application de visioconférence.

Le travail a été réparti entre deux équipes, l'une chargée de concevoir le système de réservation de ressources et l'autre s'occupant de l'application de visioconférence.

Nous verrons dans un premier temps les développements menés autours du système de réservation de ressources avant de présenter dans un deuxième temps la conception de l'application de visioconférence.

I - Le système de réservations de ressources

A. Présentation

1. Le besoin de garantir la qualité de service sur Internet

Le problème de congestion prend de plus en plus d'importance dans un environnement gourmand en bande passante. Les données transmises sur Internet sont de plus en plus volumineuses, surtout avec l'intégration des services multimédia aux applications (visioconférence, radio Internet, vidéo temps réel, streaming …). Les besoins sont aussi différents : transmettre un flux vidéo nécessite de garantir un débit, une gigue, (…) constante, l'application est dite temps réel. Aujourd'hui, les normes évoluent, et de nouveaux protocoles comme RTP ou RTCP permettent d'améliorer la qualité des services fournis. Toutefois, les problèmes liés à l'utilisation du protocole IP, comme les congestions des routeurs intermédiaires ou la pénurie en bande passante, restent difficile à résoudre.
Ainsi, comment dans un système hétérogène peut on parvenir à proposer à l'utilisateur un service fiable et de qualité ?

Deux techniques permettent de répondre à ce besoin à savoir les approches IntServ et DiffServ.

2. Les deux grandes approches de la QoS : IntServ/DiffServ

IntServ a été au cœur du projet de l'année précédente. Or nous étudions un réseau IntServ/DiffServ, c'est-à-dire deux domaines interconnectés proposant deux approches différentes pour gérer la qualité de service. En effet, alors qu'IntServ intègre la notion de flux applicatif et propose pour ces flux la garantie de la Qualité de Service (QoS), DiffServ traite les paquets IP par classe.

IntServ fourni 3 services au niveau IP (BE, GS et CL[1]) et le contrôle d'admission[2] pour chacun de ces flux impliquent les hôtes et les routeurs sur le chemin concerné. Chaque point du réseau réserve les ressource correspondant aux flux qui les traversent. Avant chaque émission de nouveau flux, les routeurs intermédiaires sont informés, et réservent si possible les ressources nécéssaires.

[1] BE = Best Effort ; GS = Guaranted Service ; CL = Controlled Loaded
[2] Le contrôle d'admission consiste en plusieurs étapes nécessaires à l'établissement de la réservation RSVP (Ressource Reservation Protocol).

L'approche DiffServ génère moins de trafic car peu de signalisation est nécessaires allégeant ainsi la charge au niveau des routeurs (il n'existe pas de soft state par flux mais un champs DSCP associé au flux). Il convient d'expliquer le fonctionnement de DiffServ qui n'a pas été traité l'année précédente.

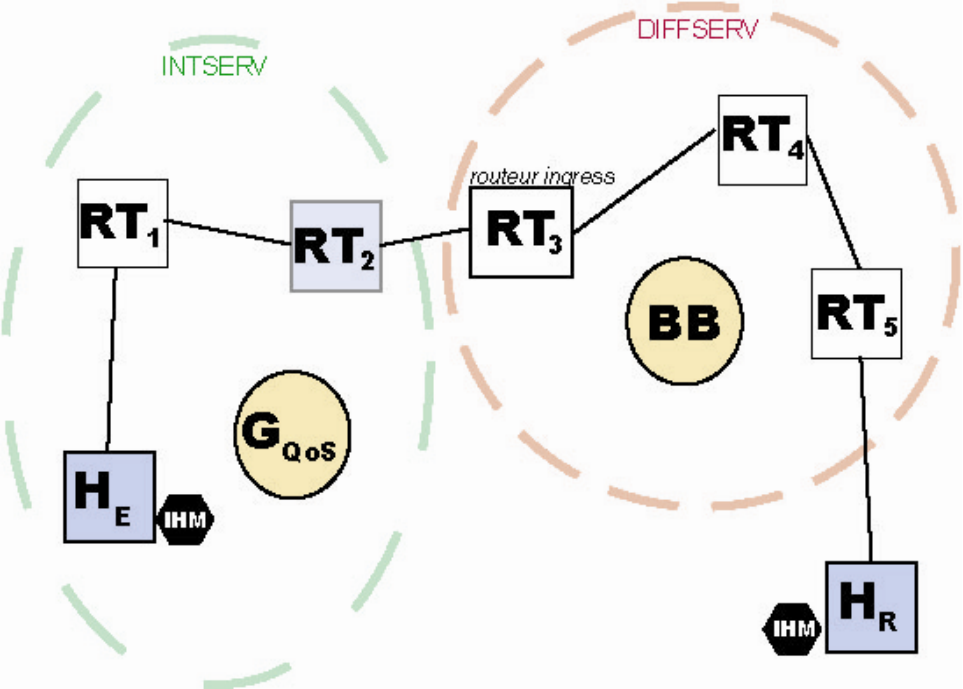
Le trafic entrant dans un domaine DiffServ est classé, marqué (marquage DSCP) en bordure du domaine avant d'y être introduit. Tandis que le traitement des paquets dans le modèle IntServ se fait à tous les nœuds de ce réseau, il ne se fait plus qu'en bordure dans le domaine DiffServ. Ainsi, à l'intérieur du réseau DiffServ et pour chaque classe de trafic, les routeurs appliquent la même politique de traitement des paquets, c'est le contrat SLA (Service Level Agreement). Toutefois, l'approche fonctionne à condition que le domaine soit suffisamment approvisionné en bande passante.

La notion de SLA qui est négocié statistiquement entre les domaines adjacents définit les services de routage IP dont va disposer le client ainsi que la façon dont va être conditionné le trafic en entrée (Traffic Conditionning Agreement).
Il est important de comprendre le comportement des routeurs de bordure et notamment d'entrée (ingress). En effet, ils se chargent de classer le trafic IP entrant. Ce classement peut se faire soit selon les champs d'entêtes (classification multi field), soit selon le champ TOS (classification « Behavior Agregate »). C'est ce type de classement qui a été retenu pour simuler le fonctionnement de DiffServ. Nous avons ainsi du mettre en place le protocole de communication en tenant compte du rôle du routeur ingress qui conditionne aussi le trafic en appliquant un policing en cas de non-respect.

Ces techniques coexistant de nos jours sur Internet, l'enjeu est donc d'analyser le fonctionnement de chacune d'entre elles et de mettre en place un système de réservation de ressources capable d'opérer pour des flux amenés à traverser les deux domaines successivement. L'année dernière, l'objectif consistait à effectuer une réservation sur le domaine IntServ. Nous avons étendu le système à une plateforme plus complexe et implanté un protocole capable d'effectuer une « réservation transversale ».

3. Le réseau hétérogène étudié

Le réseau étudié est le suivant :



Sur ce diagramme, on retrouve les différentes composantes logiques du réseau hétérogène sur lequel nous allons développer le système de réservation de ressources entre un hôte émetteur noté He et un récepteur noté Hr.
Un premier réseau (à gauche), implémente une politique d'allocation des ressources basée sur la logique IntServ, orientée connexion. Tous les routeurs traversés par un flux donné ont un rôle de régulation. Un élément-clé du réseau IntServ est le gestionnaire de QoS (GQoS).

Cette entité n'est **pas** nécessaire pour des réservations internes au domaine IntServ. En revanche, elle joue un rôle dans l'établissement de la réservation inter-domaine. En effet, elle a une connaissance globale du domaine (routeurs de bordures par exemple) ainsi qu'une connaissance plus ou moins importante des domaines limitrophes et leurs politiques de gestion de la QoS (Adresse du Bandwidth Broker par exemple). C'est ce Gestionnaire qui représentera le domaine IntServ dans notre protocole de réservation à savoir qu'il sera le seul à pouvoir échanger des informations sur la nature des flux traversés avec les domaines adjacents.

Le second réseau du diagramme (à droite) implémente quant à lui la logique DiffServ pour sa politique d'allocation des ressources. Toute la régulation est faite par les routeurs d'entrée dans le domaine (routeur ingress, dans notre cas RT3). A noter également la présence d'un Bandwidth Broker (BB) qui a pour vocation de centraliser toutes les demandes de Qualité de Service. En fonction des capacités du réseau qui lui sont connues, il informe ou non les routeurs ingress de la mise en place d'une certaine politique pour un flux donné.

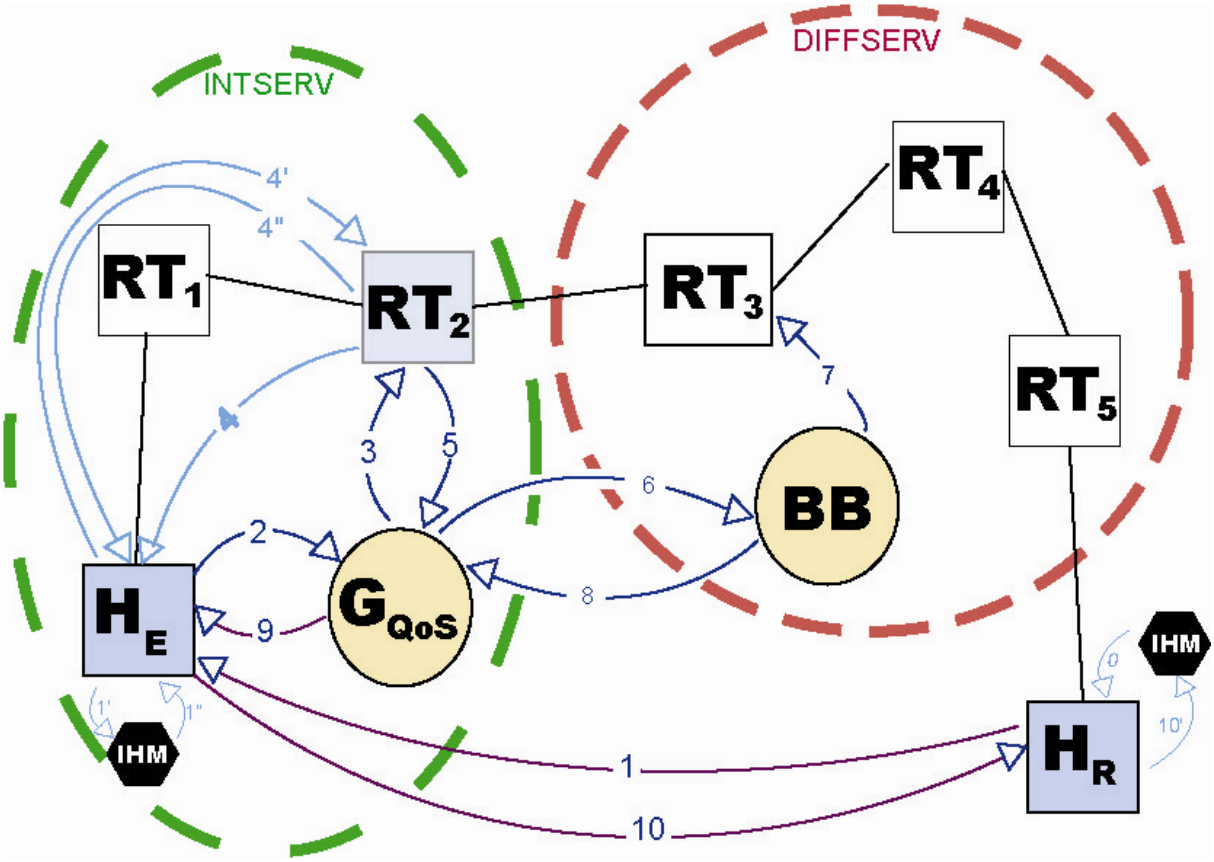
Enfin, il faut noter que l'hôte récepteur se trouve sur un troisième réseau qui sera, pour les besoins de l'exercice, considéré comme surdimensionné en bande passante (cas typique d'un accès haut débit à l'heure actuelle).

4. Présentation de la démarche de travail

La description des différents acteurs du système ainsi faite, il apparaît clairement que le système est relativement complexe. Pour pouvoir mettre en place un protocole de réservation de ressources permettant de créer un canal logique strictement dimensionné et spécifié entre l'hôte émetteur et hôte récepteur, il est nécessaire de décomposer le problème par étape. Pour chacune de ces étapes intermédiaires, il faut spécifier les données nécessaires pour qu'elle puisse être menée à bien. Une fois cette première étape terminée, l'enjeu consiste à proposer une façon d'acheminer ces données sur les différents hôtes dans un ordre cohérent afin d'effectuer la réservation.

Nous présenterons dans un premier temps une solution générale qui permet de résoudre le problème de l'acheminement des données sur les différents hôtes du système. Dans un deuxième temps, nous nous intéresserons aux différentes approches possibles pour l'implémentation de certaines fonctions en détaillant les divers arguments qui nous ont poussé à faire le choix final que nous expliquerons en détails.

B. Spécification du protocole de réservation



Le schéma logique a été enrichi des messages échangés résultant de la mise en place du protocole de communication. Nous allons décrire la réflexion menée qui nous a permis d'aboutir à cette vue logique ainsi que la signification et le rôle de chacun des messages intervenant.
Nous avons d'abord été amené à comprendre et à nous approprier les concepts propres aux approches DiffServ et IntServ. De plus, le projet étant dans la continuité de celui réalisé l'année précédente, il nous a aussi fallu nous mettre à jour sur les mécanismes déjà implémentés et les adapter à notre réseau.

1. Adaptation de la réservation IntServ existante

La réservation qui avait été implémentée l'année précédente s'effectuait entre deux hôtes appartenant à un réseau IntServ. Or dans notre cas, la réservation s'effectue

entre un hôte récepteur hors du réseau IntServ et un hôte émetteur situé dans le réseau IntServ. Donc, elle ne fonctionnera pas dans notre réseau.
Comme nous souhaitons réutiliser le logiciel de réservation de ressources fait l'année dernière sans avoir à le modifier, il est nécessaire que, vis-à-vis du système de réservation déjà établi, les deux hôtes se trouvent sur le même réseau. Afin que cela leur soit transparent nous mettrons en place un proxy en RT2 (Les détails de l'implémentation de ce proxy seront explicités ultérieurement).

2. Mise en place de la réservation sur DiffServ

La réservation de ressources dans le domaine IntServ ne suffit pas, il faut aussi la mettre en place dans le domaine DiffServ. Or les mécanismes à mettre en œuvre sont tout à fait différents, puisque le principal traitement à faire se situe au niveau de RT3 (routeur de bordure ingress du domaine DiffServ). Contrairement au domaine IntServ, nous avons entièrement mis en place le protocole.
Ainsi nous avons géré chaque étape de la communication, aussi bien sur la spécification du comportement du BandWidth Broker que sur le routeur ingress.

3. Description des messages du protocole de communication [3]

PDU 0 : Ihm_Hr_ReqQos

Il s'agit de la requête IHM. Elle est engendrée par l'utilisateur (récepteur pour le flux) qui remarque une dégradation de la qualité du flux et qui souhaite donc obtenir une garantie de qualité. Ce flux se caractérise par le numéro de port et l'adresse IP de l'émetteur et du récepteur du flux (ID_Flux = @IPSrc, @IPDest, Psrc, Pdest).

↳ **QoS, @IPdest, Portdest + @IPsrce, Portsrce**

PDU 1 : Hr_He_ReqQos

Il s'agit de la requête qui va de HR à HE. Nous avons besoin, pour cette requête, de la qualité de service souhaitée ainsi que de l'identifiant du flux tel que défini précédemment, c'est l'étape 0 qui permet de recueillir les informations nécessaires.

↳ **QoS, @IPdest(HR), Portdest(HR), @IPsrce(HE), Portsrce(HE)**

[3] Un tableau récapitulatif est donné en annexes

PDU 1' et 1'' : He_Ihm_AffichSpec & Ihm_He_InfoSpec

Lors de la réception d'une demande de réservation provenant de Hr, l'émeteur du flux est sollicité à travers une IHM afin de préciser les caractéristiques permettant de dimensionner le flux qu'il est en train d'émettre. Ceci permet d'anticiper sur la réservation RSVP qui sera effectuée au sein du domaine IntServ.
Pour des raisons liées à l'implémentation, il faudra mettre en place un proxy au niveau de RT2. Cela se justifie pleinement du fait que RT2 est un routeur de bordure et donc sépare deux réseaux de nature différente. Les utilisateurs du domaine IntServ susceptibles, pendant leur communication, d'avoir besoin d'une garantie de la qualité de service seront invités à se connecter via le proxy *(son fonctionnement est expliqué plus loin).*

Dans cette étape, deux PDU apparaissent : l'un de l'entité de protocole vers l'IHM et l'autre de l'IHM à l'entité de protocole.

- **1' He_Ihm_AfficheSpec : @IPdest, portDest, @IPsrc, PortSrc**
- **1'' Ihm_He_InfoSpec : @IPdest, Portdest, @IPSrc, PortSrc, TSPEC**

PDU 2 : He_Gqos_InfoSpec

On informe le GQoS, en fournissant notre @IPsource, Portsource, et en précisant l'@IPdest, le port dest, la QoS souhaité et les caractéristiques du flux pour lequel on désire réserver des ressources (TSPEC) que nous a fourni l'utilisateur à l'étape précédente.

***Rappel** : r,b,p sont les paramètres qui caractérisent l'allocation de ressources demandée par l'utilisateur HE.*
On a : b = taille max d'une rafale ; r = débit moyen ; p = débit crête.

La QoS souhaitée est la même que celle demandée initialement. Les données du TSPEC ne sont pas utilisées par le Gestionnaire de QoS pour l'établissement d'une réservation dans le domaine IntServ, elles sont destinées à être communiquées au Bandwidth Broker du domaine DiffServ une fois la réservation dans le domaine IntServ réussie.

- **@IPsrc(HE), PortSrc, @IPdest, PortDest, QoS_requis, rbp (=TSPEC)**

PDU 3 : Gqos_Rt2_ReqResa

Une fois que le gestionnaire de QoS est informé qu'une demande de réservation est en cours, il doit déterminer le routeur de sortie du domaine IntServ auquel il doit

s'adresser afin de lui propager la demande. En effet RT2 devra, conformément au modèle du projet de l'année 2003-2004, s'adresser à l'émetteur du flux pour l'établissement d'une réservation RSVP.

↳ **QoS_requis, @IPsrce(HE), portsrce(HE),@IPdest(HR), Portdest(HR).**

Sur réception du message venant de GQoS, RT2 doit envoyer un message à HE via une requête de signalisation pour lui demander de lancer la réservation RSVP. Celle-ci se fera entre HE et RT2 (ce qui est logique vu le rôle de proxy joué au niveau du domaine IntServ par le routeur de bordure RT2 comme exposé précédemment). Ceci est réalisé par le logiciel Résa 2004 duquel notre application est simplement cliente.

PDU 4 , 4'P et 4'R :

Il s'agit des PDU générés par le service proposé par l'application Résa 2004. Afin de démarrer cette réservation, RT2 devra générer un PDU de type GENERE_PATH à l'attention du logiciel Résa2004. Celui-ci contactera ensuite l'entité de protocole que nous mettons en place sur He (via l'instance de RESA en He) afin d'obtenir les données de TSPEC correspondant au flux. Ces données sont connues de He puisqu'elles ont été demandées à l'utilisateur au début de la réservation. La requête émise par Résa2004 à destination de He et la réponse respective portent pour nom :

AFFICHE_REQ_RESA_2004 et REQ_RESA_IHM_2004.

PDU 5: Rt2_Gqos_ResaIntservOk

Sur réception d'un PDU AFFICHE_RESA_2004, l'émetteur et le récepteur savent que la réservation a réussi. Si c'est le cas, RT2 peut générer un paquet RT2_GQoS_ResaIntServOK au gestionnaire de QoS pour signaler que la partie INTSERV est prête et que les routeurs intermédiaires ont alloués les ressources nécessaires. Par ailleurs, HE a lui aussi connaissance du succès de la réservation (dans l'implémentation actuelle, cette information n'est pas exploitée).

↳ **[OK, @Dest,@Src, PSrce, PDest, QoS]**

Le Gestionnaire de QoS enregistre alors le fait que la réservation est effectuée pour le flux ID_FLUX de QoS et TSPEC souhaitées.

PDU 6 : GQoS_BB_ReqAlloc

Une fois la réservation sur le domaine IntServ réussie, le Gestionnaire de QoS va devoir signaler au Bandwidth Broker (BB) du domaine DiffServ la qualité de service souhaitée ainsi que les ressources qu'il souhaite réserver. En effet, on a « préparé le

terrain » côté IntServ, mais encore faut il garantir que les paquets ne seront pas éliminés après RT2 en accord avec le policing en vigueur.
Le Gestionnaire de QoS envoie donc une requête d'allocation de ressources pour le flux dans le domaine DiffServ. Pour ce flux, RT2 se place en tant qu'émetteur et Hr reste lui le récepteur de ce flux.
Le format du PDU est le suivant :

- **QoSrequis, @IPSrc, PortSrc(utilité future…), @IPdest, PortDest, rbp(=TSPEC)**

Explications :
TSPEC avait été gardé en mémoire par GQoS, donc on peut envoyer cette donnée.
La QoS souhaitée est présente ainsi que les caractéristiques et les identifiants du flux afin que le Bandwidth Broker puisse être informé de la QoS à mettre en œuvre dans son domaine DiffServ.

Comment traiter le port source pour un usage futur éventuellement ?
Supposons que la réservation ait été effectuée avec succès. Alors, RT2 retransmet les paquets qui lui sont adressés sur le port du proxy à destination de HR sur le port de réception des paquets de HR. Or si un intrus s'insère dans le réseau IntServ, il lui suffit d'envoyer les paquets à destination de RT2 sur le port du proxy. RT2 retransmet alors les paquets en spécifiant que @IPsrce=@IPRT2 ! Si on ne précise pas le port source des paquets, l'intrus peut profiter de la réservation pour communiquer de façon privilégiée avec Hr ou simplement nuire à la bonne qualité de la transmission en générant un fort traffic !! Or un comportement envisageable du proxy serait qu'il utilise pour émettre les données le même port que celui depuis lequel ont été émis les paquets qu'il reçoit, de cette façon, un attaquant devrait également connaître le port source à utiliser afin d'arriver à ses fins.
Ceci dit, en dehors de tout aspect sécurité, le port source ne semble pas utile pour le moment.

PDU 7 : Bb_Rt3_DiffServReserv

- **QoS requise, Portdest, @IPdest, @IPsrc, PortSrc, TSPEC)**

Le BandWidth Broker (BB) envoie une requête au routeur de bordure (ingress, RT3) pour le configurer afin qu'il applique un marquage différentiel en fonction de la QoS demandée (ID_Flux, TSPEC). Ainsi les paquets à destination de HR auront des marquages pour le domaine DiffServ permettant de les différentier en leur affectant une classe de service.

PDU 8 : Bb_Gqos_ReqDiffServOk

\ reponse=OK, @IPsrce =@IPRT2, Portsrce= ??, @IPdest=@IPHR, Portdest

Les requêtes pour DiffServ sont centralisées au niveau de BB. Ainsi, c'est lui seul qui décide où non d'accepter la demande de réservation de ressources en accord avec les différentes règles de policing en vigueur et les ressources disponibles. Il envoie alors un message à destination de GQoS, initiateur de la réservation, pour l'informer du succès ou de l'échec de celle-ci.

PDU 9 : Gqos_He_RepReservOk

\ < Réussite ? >, @IPsrce = @IPHE, PortSrce = PortHe, @IPdest = @IPHR, PortDest

Une fois les demandes de réservations pour les deux réseaux IntServ et DiffServ finies, GQoS sait si la réservation a réussi ou non. GQoS doit alors propager ce résultat vers l'hôte émetteur.

PDU 10 : He_Hr_ResaOk

\ < Réussite ? >, @IPsrce = @IPHE, PortSrce = PortHe, @IPdest = @IPHR, PortDest

Suite à la réponse provenant de GQoS sur la réussite de la réservation ou non, HE envoie une réponse à la requête envoyée par HR (req1) concernant la demande de réservation de ressources.

De plus, l'issue de la réservation est signalée à l'utilisateur via l'IHM. C'est le rôle des PDUs He_Ihm_ResultResa et Hr_Ihm_ResultResa qui ne figurent pas sur le schéma par soucis de clarté.

Une grande réflexion a été menée au niveau de l'interconnexion des routeurs RT2 et RT3 pour savoir comment nous allions effectuer l'implémentation vis-à-vis des mécanismes de réservations. Or pour établir une qualité de service, il faut tout d'abord sélectionner les flux qu'on veut différencier. Ainsi pour chaque flux, il faut établir des correspondances sur la qualité de service pris en charge dans le domaine IntServ et celle prise en charge dans le domaine DiffServ. Nous avons réfléchies à deux approches que nous développerons par la suite.

C. Les choix d'implémentation

1. L'implémentation d'un proxy en RT2

Sur le domaine IntServ, la réservation de ressources utilise le service fourni par l'application de réservation de ressources développée en 2004 par nos prédécesseurs. Afin de pouvoir la conserver et réutiliser le service fourni dans nos applications, il est donc nécessaire que les données du flux mentionnent une machine du réseau IntServ comme destinataire (ici RT2 – routeur de sortie). C'est pourquoi l'application qui effectue une réservation devra adresser ses paquets à RT2. Afin de délivrer les données à l'hôte récepteur, un proxy applicatif (relayant simplement les paquets UDP en changeant l'adresse de destination) sera mis en place de façon statique avant d'initier la communication.

2. L'implémentation en RT3: multiplexage de port vs. marquage TOS

Le routeur d'entrée sur le domaine DiffServ (RT3) a pour rôle de marquer les paquets qu'il reconnaît comme appartenant à un flux faisant l'objet d'une réservation (qui lui aura été indiquée par BB).

Afin d'effectuer ce marquage, il existe plusieurs possibilités. Une première approche que nous avons envisagé concerne un multiplexage différentiel par port source. En effet, le routeur modifierait le port source des paquets qui le traversent afin d'identifier leur classe d'appartenance. (Le principe est détaillé en Annexe)
Cette méthode présente deux inconvénients majeurs. Tout d'abord, il faut être certain que l'application ne se base pas sur le port source des paquets reçus pour envoyer une réponse car sinon celle-ci serait adressée à un port erroné. Ce comportement peut être écarté du fait que l'on travaille principalement avec des applications utilisant le protocole RTP/UDP pour véhiculer des flux multimédias (ne nécessitant pas de réponse).

La seconde difficulté résulte de l'implantation. En effet, dans une première approche, nous pensions à la mise en place d'un proxy qui aurait le rôle d'effectuer ce multiplexage, ce qui signifie que les paquets auraient du être adressés à RT3 et remontés au niveau applicatif. Fort heureusement, cette difficulté a été levée par l'utilisation d'iptables (voir Annexe sur l'utilisation d'iptables) qui permet de

modifier le numéro de port source des paquets « à la volé » sans remonter au niveau applicatif.
Enfin, ayant découvert la possibilité d'utiliser iptables pour changer les données de l'en-tête de niveau 4 d'un paquet, il semblait naturel de s'intéresser aux champs disponibles au niveau IP et plus particulièrement aux champs ToS (Terms of Service) et DSCP (Differentiated Services Code Point). Il s'est avéré simple en effet de modifier le champ ToS par l'utilisation des fonctionnalités du noyau linux. Cette approche nous permet d'utiliser des fonctionnalités fournies par le noyau tout en se rapprochant plus des standards DiffServ.

3. Policing sur routeur de bordure (ingress)

La question d'établir un policing permettant de définir les différentes classes de service ainsi que leurs spécifications respectives a également été abordée. La méthode retenue pour cela repose sur l'utilisation de « TC » (Traffic Control), utilitaire permettant de configurer les mécanismes d'ordonnancement des paquets du noyau linux. Une autre approche étudié repose sur l'utilisation de « ipfw », utilitaire BSD afin d'effectuer le même contrôle. (Voir Annexe sur l'utilisation de TC)

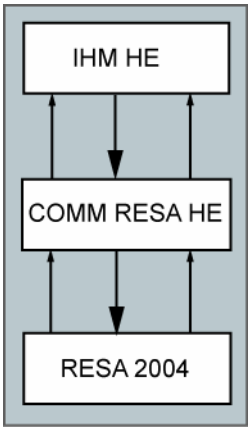
4. Utilisation de Java

Dans le contexte de développement d'une application distribuée et afin de complémenter les enseignements reçus cette année, nous avons décidé de l'utilisation du langage Java pour la programmation. En effet, l'approche objet du développement rend tout où partie du code facilement réutilisable soit dans un projet similaire, soit pour la mise en place de nouveaux services basés sur ceux fournis par l'application.

5. Architectures

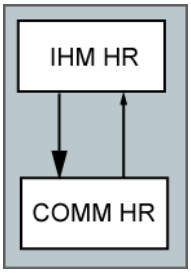
Dans l'objectif, calqué sur le modèle OSI, de rendre notre application modulaire, nous avons mis en place une architecture en couches pour laquelle nous avons défini les interfaces. L'adoption de ce choix d'architecture permet de fournir un service de réservation réutilisable par d'autres applications.
Nous allons présenté pour chaque nœud du réseau, les couches mises en place ainsi que leur finalité.

- **HE**



Sur l'hôte émetteur, cœxistent trois instances logicielles. Du bas vers le haut, on retrouve Resa 2004 qui n'est autre que le produit du projet de l'année 2003-2004 permettant d'effectuer une réservation sur le réseau IntServ. CommResaHe est une entité chargée de la communication avec les autres éléments du système de réservation que nous avons mis en place (GQoS et Hr). De plus, elle est chargée de relayer les informations à l'IHM qui permet de solliciter l'utilisateur afin qu'il saisisse les données nécessaires aux réservations et de l'informer de l'avancement de celles-ci.

- **HR**

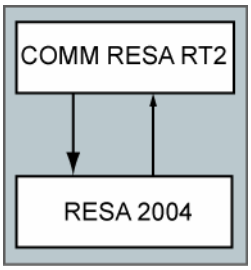


Sur l'hôte récepteur se trouvent deux instances logicielles. La première consiste en une IHM permettant à l'utilisateur d'effectuer les demandes de réservation. Cette entité ne dialogue qu'avec CommHr, la deuxième entité.

CommHr est l'entité de réservation proprement dite, elle se charge de toute la communication avec le système de réservation de ressources et informe l'IHM des événements significatifs.

Remarque : Ces deux entités logicielles, bien qu'elles ont vocation à s'éxécuter sur la même machine peuvent s'exécuter sur deux hôtes différents. CommHr devenant alors serveur et répondant potentiellement aux requêtes de QoS provenant de plusieurs clients (plusieurs IHM sur le réseau).

- **RT2**

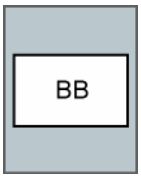


RT2 possède deux couches logicielles. Resa 2004 correspond au logiciel développé lors du Projet Tutoré 2003-2004 et rends un service de réservation sur le réseau IntServ.
CommResaRT2 est le client du service fourni par Résa, il communique également avec GQoS afin de recevoir les requêtes et l'informer de leur issue.

- **GQOS**

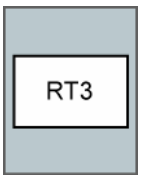
Le gestionnaire de QoS lui ne possède qu'une seule entité logicielle, elle est chargée de la coordination des différents éléments d'une réservation et contacte le Bandwidth Broker quand l'avancement d'une réservation est suffisant.

- **BB**



Le BandWidth Broker ne possède qu'une seule entité logicielle, elle est chargée de la gestion du domaine DiffServ. Elle répond aux requêtes de QoS émises par GQoS, configure les routeurs de bordure le cas échéant et informe GQoS de l'issue de ses requêtes.

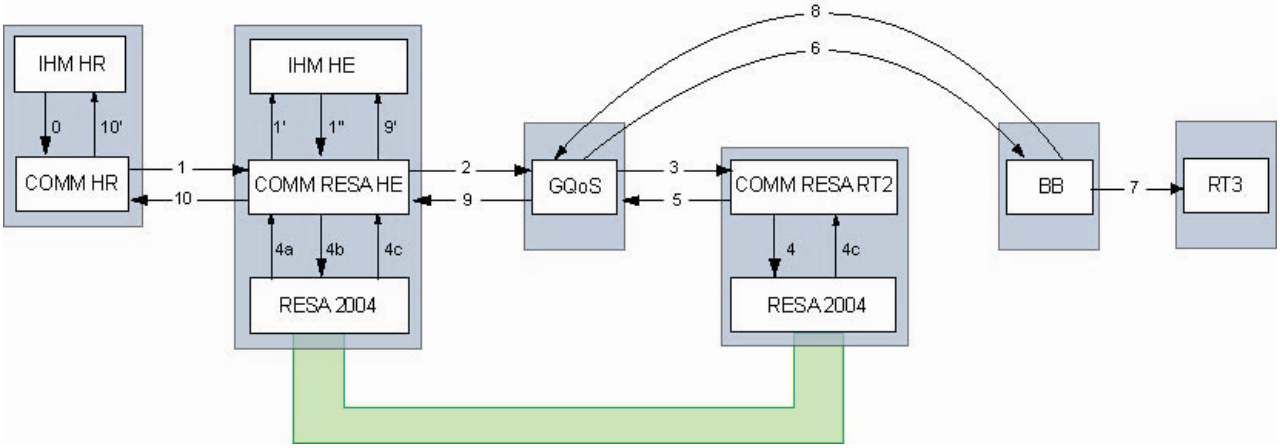
- **RT3**



RT3 se compose d'une entité logicielle. Elle a pour rôle d'exécuter les requêtes du BandWidth Broker afin de rendre effectif le marquage des paquets. Ceci est réalisé par l'exécution d'une commande iptables visant à mettre en place le marquage à la volée.

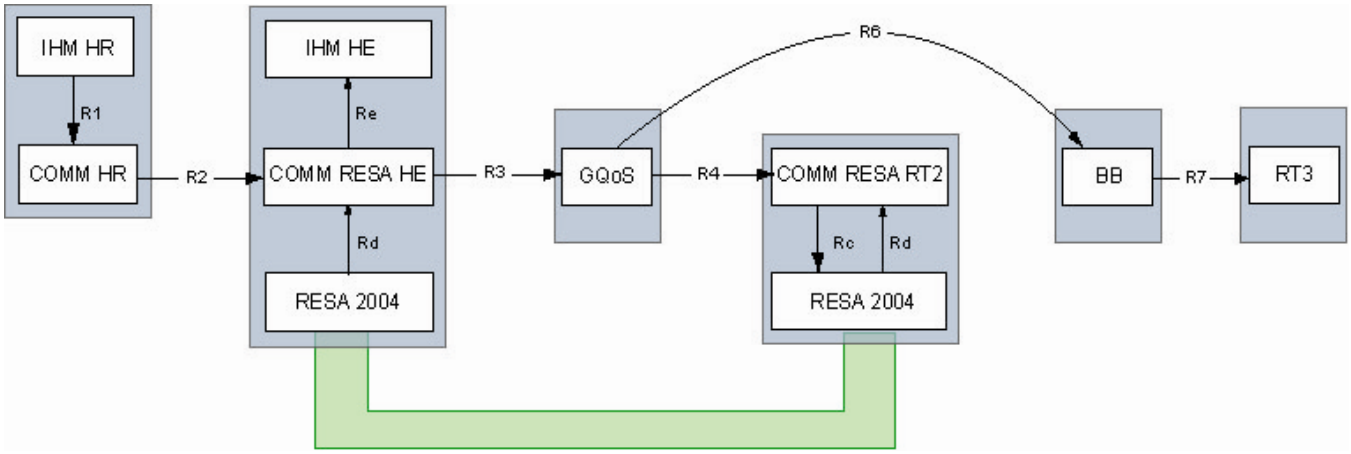
o *Fonctionnement général*

1. Réservation réussie



0 – Ihm_Hr_ReqQoS
1 – Hr_He_ReqQoS
1' – He_Ihm_AfficheSpec
1'' – Ihm_He_InfoSpec
2 – He_GQoS_InfoSpec
3 – GQoS_RT2_ReqResa
4 – Generate_Path_2004
4a – Affiche_Req_Resa_2004
4b – Req_Resa_Ihm_2004
4c – Affiche_Resa_2004
5 – RT2_GQoS_ResaIntServOK
6 – GQoS_BB_ReqAlloc
7 – BB_RT3_DiffServReserv
8 – BB_GQoS_ReqDiffServOK
9 – GQoS_He_ResaOK
9' – He_Ihm_ResultResa
10 – He_Hr_ResaOK
10' – Hr_Ihm_ResultResa

2. Réservation relâchée



R1… R7 + Rc – Relache_Resa
Rd – Desaffiche_Resa_2004

Si aucune signalisation n'est émise après qu'une réservation ait abouti, la réservation reste active. Lorsqu'un relâchement à lieu, le gestionnaire GQoS est alors mis au courant, il se chargera d'invalider la réservation sur le domaine IntServ et de prévienir le Bandwidth Broker du domaine DiffServ qui fera la même chose sur son domaine.

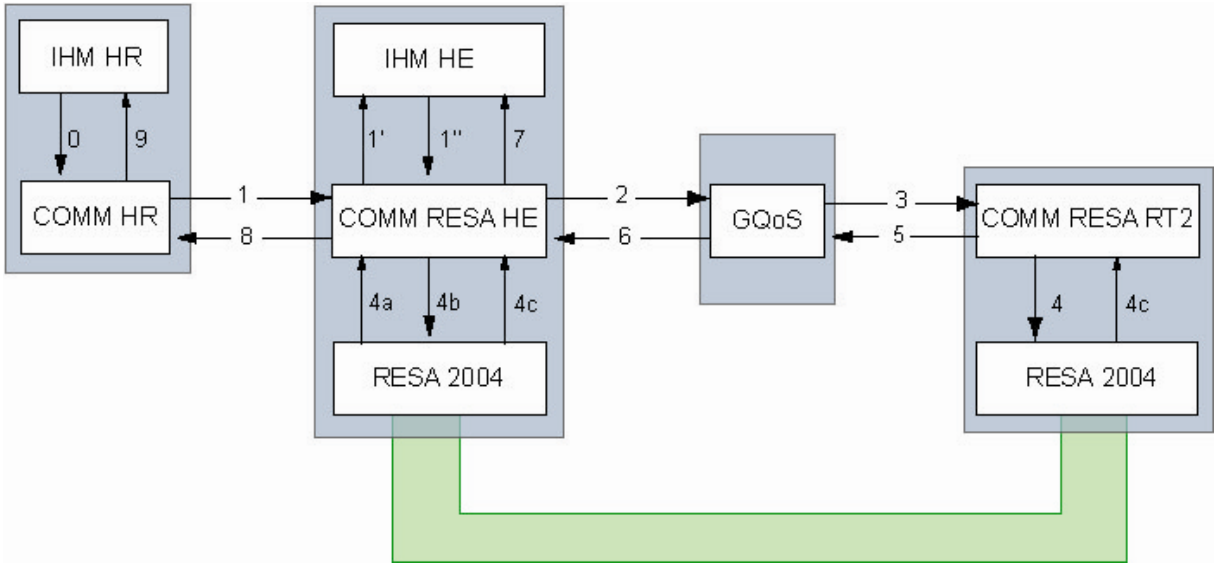
Nous avons basé l'implantation de notre protocole sur l'utilisation du protocole de transport UDP. Nous aurions pu choisir des communications en TCP, qui reposent sur un mécanisme de connexion. Ce choix aurait allégé le protocole pour le relâchement de la réservation. Néanmoins, il n'est pas plus compliqué de travailler avec des flux UDP, d'autant plus que l'on s'affranchit des phases de connexions qui engendrent un fort surdébit compte tenu de la taille de nos PDUs.

L'échec de la réservation est bien entendu signalé aux hôtes d'extrémités concernés, de même pour le relâchement.

Nous détaillons ci-dessous les différents cas d'échec envisageables et la façon adoptée pour assurer la cohérence des états (typiquement, éviter qu'une réservation soit active sur le réseau IntServ alors qu'elle est considérée comme impossible sur le réseau DiffServ et que les hôtes d'extrémités considèrent que la réservation a échouée et vice-versa).

3. Cas d'échec

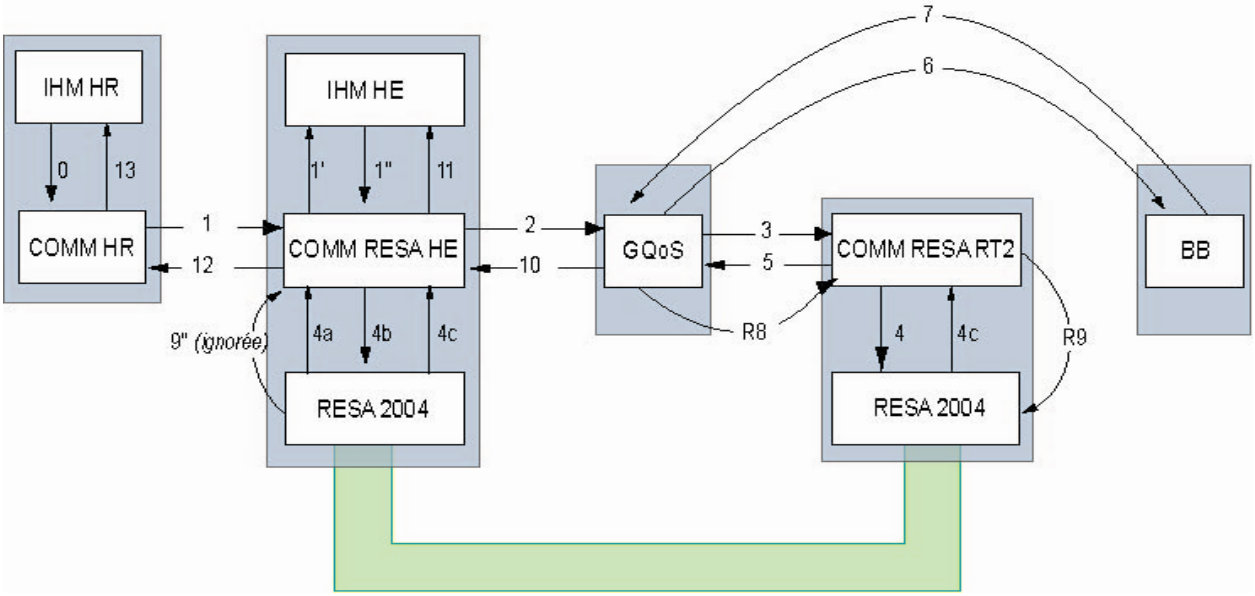
- **Echec IntServ :**



0 – Ihm_Hr_ReqQoS
1 – Hr_He_ReqQoS
1' – He_Ihm_AfficheSpec
1'' – Ihm_He_InfoSpec
2 – He_GQoS_InfoSpec
3 – GQoS_RT2_ReqResa
4 – Generate_Path_2004
4a – Affiche_Req_Resa_2004
4b – Req_Resa_Ihm_2004
4c – Affiche_Echec_Resa_2004
5 – RT2_GQoS_ResaIntServOK (false)
6 – GQoS_He_ResaOK (false)
7 – He_Ihm_ResultResa (false)
8 – He_Hr_ResaOK (false)
9 – Hr_Ihm_ResultResa (false)

Si la réservation du côté IntServ échoue, il n'est pas nécessaire de propager la demande de réservation dans le domaine DiffServ.

- **Echec DiffServ :**



0 – Ihm_Hr_ReqQoS
1 – Hr_He_ReqQoS
1' – He_Ihm_AfficheSpec
1'' – Ihm_He_InfoSpec
2 – He_GQoS_InfoSpec
3 – GQoS_RT2_ReqResa
4 – Generate_Path_2004
4a – Affiche_Req_Resa_2004
4b – Req_Resa_Ihm_2004
4c – Affiche_Resa_2004
5 – RT2_GQoS_ResaIntServOK (true)
6 – GQoS_BB_ReqAlloc
7 – BB_RT3_DiffServReserv (false)
8 – Relache_Resa_2004
9 – Relache_Resa_2004
10 – GQoS_He_ResaOK (false)
11 – He_Ihm_ResultResa (false)
12 – He_Hr_ResaOK (false)
13 – Hr_Ihm_ResultResa (false)

Si la réservation a été réussie dans le domaine IntServ mais échoue dans le domaine DiffServ, il est nécessaire que lorsque GQoS en est informé, ce dernier demande la relâche de la réservation InServ à RT2.

II. Application de Visioconférence

Le but de notre partie était de développer en Java une application de visioconférence de type unidirectionnelle et interactive, intégrant des mécanismes de caractérisation du trafic générés et présentant de façon graphique les résultats en temps réel à l'écran. Notre démarche a été dans un premier temps d'étudier les protocoles devant être utilisés, à savoir RTP et RTCP, à l'aide de leur RFC respective.

Le protocole RTP (Real-time Transport Protocol) est intéressant pour nous puisqu'il fournit un service de bout en bout approprié aux applications manipulant des données en temps réel, comme de la vidéo, de l'audio, ou des données de simulation par exemple, sur des réseaux unicast ou multicast. RTP n'offre aucune garantie de service (donc n'assure pas le délai de transfert des données par exemple), n'effectue aucune réservation de ressource, mais permet un monitoring des paquets via un protocole RTCP (Real-time Transfert Control Protocol).

RTCP est quant à lui basé sur une transmission périodique de paquets de contrôle pour tous les participants d'une session RTP, utilisant le même mécanisme de distribution des paquets de données.

Ces deux protocoles RTP et RTCP sont donc complémentaires et vont permettre d'assurer la cohérence des données audio et vidéo lors de la vidéoconférence. A noter qu'ils peuvent être utilisés au-dessus de n'importe quel protocole de transport et de réseau, mais les applications les utilisent généralement sur de l'UDP/IP (ce que nous ferons aussi).

Dans un second temps, nous avons étudié la librairie JMF (Java Media Framework) qui permet principalement de gérer la lecture de médias, leur capture et les outils de transfert sur un réseau. Le point intéressant pour nous est qu'elle fournit une implémentation des protocoles RTP et RTCP, avec possibilité de récupérer un certain nombre de statistiques issues du protocole RTCP.

A. De la documentation à l'implémentation

1. Le cahier des charges

Notre objectif était de développer une application de visioconférence de type unicast et intéractive, intégrant des mécanismes de caractérisation du trafic générés et présentant de façon graphique les résultats en temps réel à l'écran. Nous devions de préférence utiliser le langage de programmation orienté objet JAVA avec entre autre, l'utilisation de l'API (Application Programming interface) JMF (Java Media Framework).

2. La phase de documentation

Notre démarche à été dans un premier temps d'étudier le protocole devant être utilisé, à l'aide de la documentation (RFC : Request For Comment) associée aux protocoles RTP/RTCP. Une fois les mécanismes et la compréhension des champs acquis, nous sommes passés à une phase de lecture de la JAVADOC ainsi que de documents associés à l'API JMF. C'est elle qui permet d'utiliser les protocoles RTP/RTCP. L'objectif était de connaître les possibilités et les limites qu'offrait cette API en terme d'utilisation des protocoles RTP/RTCP, avec notamment la récupération des données qu'offre RTCP en terme de caractérisation de flux.

3. La conception et l'implémentation du logiciel final

Une fois la phase de documentation finie, nous sommes passés au développement du logiciel. Il nous a été demandé de reprendre les logiciels développés à l'issue d'un projet tutoré de 4RT effectué en 2003-2004 intitulé « Développement en JAVA d'une application de visioconférence et mise en œuvre sur une plate forme RSVP ». Le travail effectué par nos prédécesseurs a été la création d'un logiciel utilisant le protocole RTP pour créer une visioconférence unidirectionnelle mais sans caractérisation du flux à l'aide du protocol RTCP. Nous devions donc reprendre les sources de ce logiciel et y incorporer les mécanismes de caractérisation du trafic. Un des aspects important souligné par notre tuteur était de concevoir ces mécanismes de caractérisation de manière à les rendre le plus clair et le plus agréable possible pour les utilisateurs. Les statistiques ne devaient donc pas être simplement récupérées et affichées, mais demandaient un gros travail sur leurs présentations avec l'utilisation de graphiques et de listings . Nous devons préciser que la présence des graphiques n'a pas qu'un simple but d'embellir notre interface, mais surtout d'interpréter de manière significative et juste les données rendues disponibles associées au flux.

Nous avons donc commencé par regarder le travail de nos prédécesseurs. L'interface qu'ils avaient proposée était simple et efficace dans le cadre de leur projet, mais en ce qui concerne nos demandes, elle n'en devenait pas suffisamment adaptée. Effectivement nous ne voulons pas rajouter une simple fenêtre au logiciel mais créer une interface permettant d'observer dans une seule fenêtre la vidéo reçue (pour le récepteur) ou émise pour (l'émetteur) ainsi que les éléments de caractérisation. A cela s'est ajouté une lecture des sources qui ne nous ont pas satisfaits de part leur structuration et leur manque de commentaires. Nous avons donc, avec l'accord de notre tuteur, décidé de reprendre le travail à zéro, en développant un nouveau logiciel.

B. Présentation du logiciel développé

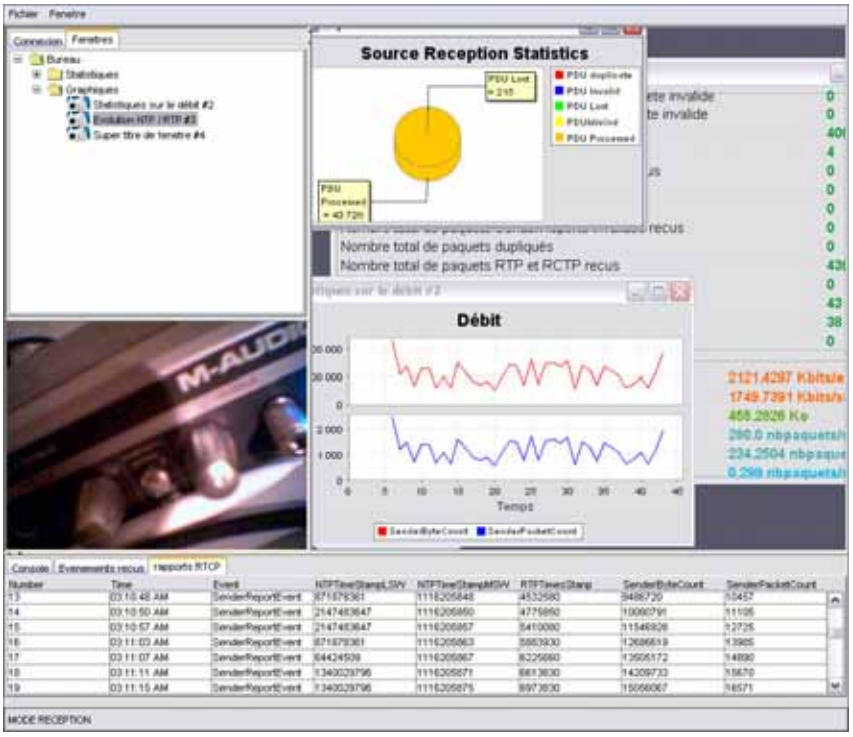
Contrairement à ce que peut laisser supposer le titre du projet, l'application développée n'a pas pour unique fonction d'assurer une vidéoconférence. Elle doit surtout mettre en évidence des statistiques variées sur le trafic, afin de permettre une analyse en temps réel grâce à plusieurs graphiques. La vidéoconférence n'est alors que le support de tous les calculs à effectuer. Précisons que la vidéo est le seul média à utiliser (le son pourrait tout de même être rajouté par multiplexage grâce aux fonctionnalités de la JMF, mais nous ne l'utiliserons pas ici), et que la conférence doit seulement être unidirectionnelle (caractéristique demandée par l'architecture sous-jacente).

1. Structure de l'interface graphique

L'application développée doit mettre en évidence les éléments suivants :

- La vidéo reçue dans le cas du récepteur
- Un panneau de contrôle de codec chez l'émetteur
- Un panneau de connexion simple
- La liste des événements reçus
- Des statistiques sur le déroulement de la visioconférence

Voici un aperçu de l'interface graphique côté récepteur lors d'une session de vidéoconférence :



Voici un aperçu de l'interface graphique côté émetteur lors d'une session de vidéoconférence :

2. Utilisation du logiciel

Nous avons fait en sorte que l'application soit la plus simple possible, malgré le grand nombre de composants a y insérer.
Tout d'abord, le panneau de connexion propose 2 modes :

- un mode Local permettant de tester l'application sur un seul poste.
- un mode Distant permettant de tester l'application sur n'importe quel interréseau.

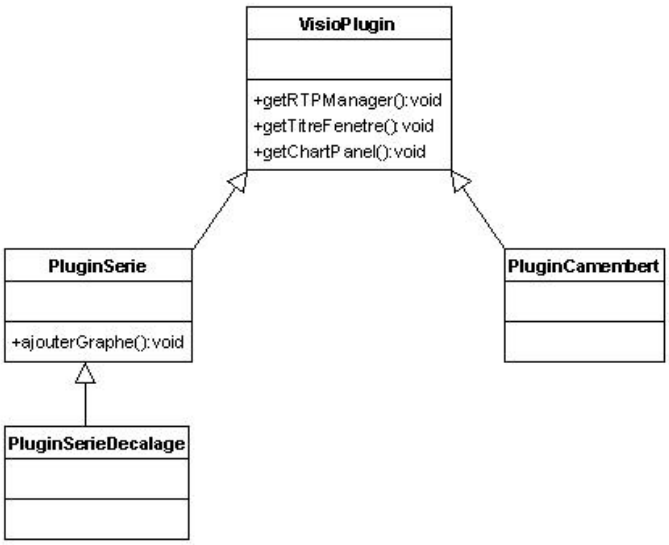
Puis, si l'on se positionne en tant qu'émetteur de vidéo, un sélecteur permet de choisir la webcam a utiliser pour la transmission.

Une fois la connexion établie, des panneaux recensant tous les événements reçus lors de la session vont se présenter dans les onglets situés en bas de l'application. Ces événements permettent d'avoir une trace de tous ce qui se passe durant la conférence.
Dans le bureau situté au centre de l'application vont aussi venir s'afficher plusieurs fenetres de statistiques diverses, mettant chacune en évidence sous forme de graphe ou de camembert l'évolution d'une donnée particulière.

3. Extensibilité de l'application

Le cahier des charges ne précisait pas le type de statistiques désiré, c'est pourquoi nous avons créé un mécanisme de plugin permettant à tout utilisateur de mettre facilement en évidence ses propres statistiques.
Pour cela, il suffit simplement de placer une archive Java (un fichier d'extension .JAR) dans le répertoire « plugin/repository », et toutes les classes étendant « VisioPlugin » (ou ses classes filles) se chargeront automatiquement au démarrage de l'application.

Voici le diagramme de classe du système de plugins :



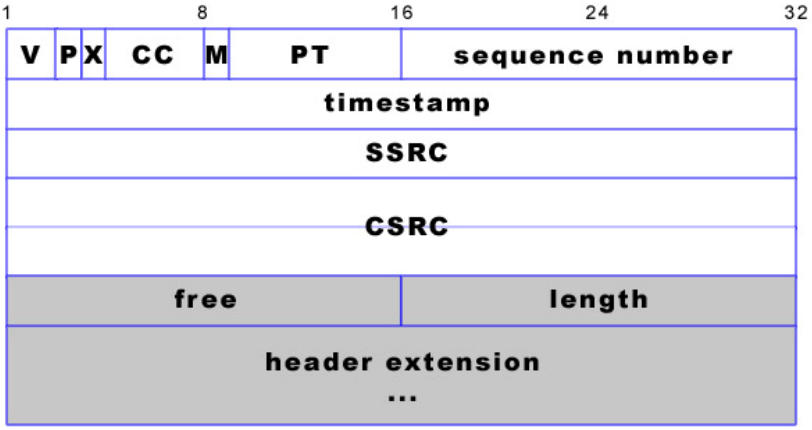
- VisioPlugin est la classe mère de tous les plugins graphiques, elle propose des fonctionnalités de base.
- PluginSerie est la classe mère des plugins mettant en évidence des statistiques sous forme de courbes (avec mémorisation de l'ensemble des données).
- PluginSerieDecalage est la classe mère des plugins mettant en évidence des statistiques sous forme de courbes sur une fenetre de temps de largeur fixe.
- PluginCamembert est la classe mère des plugins mettant en évidence des statistiques sous forme de camembert.

C. Présentation des protocoles RTP et RTCP

Le protocole RTP (Real-time Transport Protocol) fournit un service de bout en bout approprié aux applications manipulant des données en temps réel, comme de la vidéo, de l'audio, ou des données de simulation par exemple, sur des réseaux unicast ou multicast.
RTP n'offre aucune garantie de service (donc n'assure pas le délai de transfert des données par exemple), et n'effectue aucune réservation de ressource, mais permet un monitoring des paquets via un protocole RTCP (Real-time Transfert Control Protocol). C'est donc à l'aide des informations des entêtes RTP et des paquets RTCP que les données pourront être reconstituées correctement.
Les protocoles RTP et RTCP peuvent être utilisés au-dessus de n'importe quels protocoles de transport et de réseau, mais les applications les utilisent généralement sur de l'UDP/IP.

1. Le protocole RTP

Un paquet RTP est composé d'une entête et d'une charge utile (payload) transportant une donnée multimedia. L'entête contient des informations sur son payload, et peut aussi posséder une extension permettant d'apporter des informations complémentaires.
Les paquets RTP transitent lorsqu'une session est créée, une session étant identifiée par une adresse réseau ainsi que 2 ports (un pour RTP, un autre pour RTCP). Généralement, on associe une session à chaque équipement multimédia, afin d'éviter les problèmes de multiplexage de paquets RTP de sources différentes.
Une source de paquets RTP (donc un équipement multimédia en général), aussi appelée source de synchronisation, est identifiée par un numéro unique codé sur 32 bits, et est nommée SSRC. Ce numéro est alloué au hasard, et son unicité est assurée par un algorithme d'évitement de collisions.



Voici un descriptif des champs composant l'entête des paquets RTP :

- **Version (V)** : 2 bits
  - Ce champ identifie la version de RTP.

- **Padding (P)** : 1 bit
  - Permet de savoir si le paquet contient des octets additionnels de remplissage a la fin. Le dernier octet de remplissage contient le nombre d'octets de remplissage qui doivent être ignorés. Ces octets de remplissage peuvent être utilisés par exemple par certains algorithmes de cryptage utilisant des tailles de blocs fixes, ou encore pour le multiplexage mis en œuvre par les protocoles des couches inférieures.

- **Extension (X)** : 1 bit
  - Permet de savoir si entête est munie d'une extension qui lui est concaténée (donc après le champ CSRC).
    Cette extension permet de rajouter des informations complémentaires aux paquets RTP. entête de cette extension est composée de 32 bits :
    - ♣ les 16 derniers sont destinés a coder le nombre de mots de 32 bits que contient l'extension (sans compter les 32 bits de entête).
    - ♣ les 16 premiers, quant à eux, n'ont pas de signification précise, et peuvent être utilisés librement (en tant que flags, ou pour coder un checksum par exemple).

- **CSRC count (CC)** : 4 bits
  - Contient le nombre d'identificateurs CSRC listés dans entête

- **Marker (M)** : 1 bit
  - L'interprétation de ce flag est définie par un profil. Ce flag peut par exemple servir a identifier des événements significatifs tels le débit d'images ou autres.
    Un profil peut définir des flags additionnels ou préciser qu'il n'y en a aucun, via le champ Payload type.

- **Payload type (PT)** : 7 bits
  - Ce champ permet d'identifier le format du payload RTP

- **Sequence number** : 16 bits
  - Permet d'identifier chaque paquet, afin d'assurer le séquencement et de repérer les pertes éventuelles. A noter que ce nombre est choisi aléatoirement lors de la création du premier paquet RTP, pour des raisons de sécurité.

- **Timestamp** : 32bits
  - Permet de synchroniser les données reçues, mais aussi de faire des calculs de gigue.

- **SSRC** : 32 bits
  - Identifie la source de synchronisation. Ce nombre est choisi aléatoirement par le participant pour chacune de ses sources de synchronisation, et son unicité est assurée par un algorithme d'évitement de collisions entre participants.

- **CSRC** : de 0 a 15 entrées de 32 bits chacune

- Permet d'identifier les sources de synchronisation contribuant a ce payload. Si plus de 15 sources apportent leur contribution, seules 15 d'entre elles seront citées. Ce champ est rempli par les mixers.

2. Le protocole RTP Control Protocol (RTCP)

RTCP est basé sur une transmission périodique de paquets de contrôle pour tous les participants d'une session, utilisant le même mécanisme de distribution des paquets de données.
Le protocole sous-jacent doit fournir le multiplexage des données et le contrôle des paquets, par exemple en utilisant d'autres numéros de port avec UDP.
RTCP présente quatre principales fonctions :

- La première fonction est d'informer sur la qualité de transmission des données, qui est une partie du rôle de RTP, en tant que protocole de transport. Cette fonction est liée au flux et aux fonctions de contrôle de congestions d'autres protocoles de transport. L'envoi des rapports sur la qualité de réception à tous les participants permet à ceux qui rencontrent des problèmes d'évaluer si ceux-ci sont locaux ou globaux.

- RTCP possède un identifiant sur le niveau de transport de la source RTP associée. Cette information est appelée le nom canonique (CNAME : Canonical NAME). Dans la situation où un conflit est découvert, ou un programme est remis en marche, l'identifiant associé au SSRC pourrait changer. Il faut alors que les hôtes récepteurs obtiennent un CNAME afin de conserver tous les participants de la session en cours. Le CNAME permet aussi aux hôtes récepteurs d'associer plusieurs flux de données pour un participant donné, en un ensemble de sessions RTP, afin par exemple de synchroniser l'audio et la vidéo.

- Les deux premières fonctions permettent à tous les participants d'envoyer des paquets RTCP, donc le taux de transfert doit être connu afin que RTP puisse évaluer le nombre maximal de participants possibles. En ayant chaque participant qui envoie ses paquets de contrôle à tous les autres, chacun peut indépendamment observer le nombre de participants. Ce nombre permet aussi de calculer le taux de transfert des paquets.

- La quatrième fonction est facultative. Elle donne les paramètres minimaux de la session de contrôle, par exemple l'identification de participants doit être affichée dans l'interface utilisateur. C'est le paramètre le plus susceptible d'être utile en sessions "contrôle des pertes" sans quoi les participants entrent

et partent sans contrôle d'adhésion et négociation de paramètres. RTCP sert de canal pour atteindre tous les participants, mais il n'est pas forcément nécessaire d'assurer le contrôle de toutes les communications d'une application.

On constate que les trois premières fonctions sont utilisées dans un environnement IP multicast, mais elles sont aussi recommandées pour les autres environnements afin de ne pas limiter les applications RTP à des échanges unicast.

i. Rapports d'émission ou de réception (Sender Reports, Receiver Reports)

Les hôtes récepteurs RTP fournissent des rapports sur leur qualité de réception en utilisant les paquets de rapport RTCP (RTCP report paquets). Ces paquets peuvent prendre des formes différentes en fonction du fait que l'hôte récepteur est ou pas aussi un hôte émetteur. La seule différence entre la forme d'un rapport d'émission (SR) et d'un rapport de réception (RR), outre le code indiquant le type du paquet, est le fait que le rapport d'émission inclut une section d'information d'émission de 20 octets à l'usage des émetteurs actifs. Un SR est généré si un hôte a envoyé des données dans un intervalle de temps donné, sinon un RR est publié.

On peut avoir plusieurs blocs de rapport de réception dans un SR ou un RR (voire aucun). Chaque bloc correspond à une source de synchronisation avec laquelle l'hôte récepteur a reçu des paquets de données RTP depuis le dernier rapport. Les rapports ne sont pas publiés pour contribuer à l'énumération de la liste des sources présentes dans les CSRC. En effet, chaque bloc de rapport de réception fournit des statistiques au sujet des données reçues, provenant de la source indiquée dans ce bloc. Comme au maximum 31 blocs de rapport de réception peuvent être contenus dans un paquet SR ou RR, les paquets additionnels RR peuvent être empilés après le paquet initial SR ou RR, afin de contenir un rapport de réception pour chaque source entendue depuis le dernier rapport.

a. SR : Paquet RTCP relatif au rapport d'émission

Il est composé de trois sections, avec optionnellement quatre sections d'extension pour profiles spécifiques.
La première section, l'entête, est de 8 octets. Les champs sont les suivants :

- **version (V)** : 2 bits

  - Permet d'identifier la version RTP, qui doit la même que celle données dans l'entête des paquets RTP associé au paquet.

- **padding (P)** : 1 bit
  - Le principe de rembourrage est le même qu'avec RTP.
  - Dans un paquet RTCP composé, le rembourrage doit seulement être obligatoire dans le dernier paquet individuel, car un paqué composé est crypté comme un tout.

- **reception report count (RC)** : 5 bits
  - Le nombre de rapports de réception contenu dans le paquet. le nombre zéro est valide.

- **packet type (PT)** : 8bits
  - contient la constante 200 pour identifier un paquet RTCP SR.

- **length** : 16 bits
  - La longueur d'un paquet RTCP est compté en nombre de mots de 32-bits moins un. L'entête et le rembourrage doivent être comptés.

- **SSRC** : 32 bits
  - L'identifiant de la source de synchronisation pour l'initiateur du paquet SR.

La seconde section, contenant des informations sur l'émetteur, est de 20 octets et est présente dans tous les paquets de rapport d'émission. Cette section résume les transmissions des données produites par l'émetteur. Les champs sont les suivants :

- **NTP timestamp** : 64 bits
  - Il indique le temps aboslu relevée quand le rapport à été envoyé.(le temps absolu, plus communément appelé 'wallclock time', correspond au nombre de secondes écoulées depuis le 1 janvier 1900 à 0h00). Cette valeur peut être employé en combinaison avec des estampilles temporelles ('timestamps') retournées dans les rapports de réception d'autres récepteurs pour mesurer la propagation aller-retour à travers ces récepteurs. Il est permis d'utiliser l'horloge d'échantillonnage pour estimer le temps écoulé. Un expéditeur qui n'a aucune notion de temps absolu écoulé peut placer l'estampille NTP à zéro.

- **RTP timestamp** : 32 bits

  - Correspond au même temps que l'estampille NTP (ci-dessus), mais avec les mêmes unités et la même position aléatoire que les estampilles des paquets de données RTP. Cette correspondance peut être employée pour la synchronisation intra-médias et d'inter-médias pour les sources dont les estampilles NTP sont synchronisés. Elle peut aussi être employée par les récepteurs indépendants de médias pour estimer la fréquence du temps nominal de base RTP. Notez que dans la plupart des cas cette estampille ne sera égale à aucune estampille RTP d'un paquet de données adjacent.

- **sender's packet count** : 32 bits
  - C'est le nombre total de paquets de données RTP transmis par la source depuis le début de la transmission jusqu'à que ce paquet soit généré. Le comptage est initialisé à zéro quand l'identifiant SSRC de la source est modifié.

- **sender's octet count** : 32 bits
  - C'est le nombre exact de données 'payload' en octets.(cela n'inclut pas l'entête et le rembourrage) transmis dans les paquets de données RTP par la source depuis le début de la transmission jusqu'à la génération du paquet. Le comptage est aussi initialisé à zéro lors d'une modification de l'identifiant SSRC de la source. Ce champs peut être utilisé pour estimer le taux moyen de données 'payload'.

La dernière section contient de zéro à plusieurs blocs de rapport de réception, selon le nombre des sources entendues par l'émetteur traité, depuis le dernier rapport. Chaque bloc achemine des statistiques de réception sur les paquets RTP d'une seule source de synchronisation. Les récepteurs ne reportent pas lors statistiques pour une source correspondante qui a changé son identifiant SSRC. Les statistiques sont :

- **SSRC_n (source indentifier)** : 32 bits
  - L'identifiant SSRC de la source à laquelle correspond le bloc de rapport d'émission.

- **fraction lost** : 8 bits
  - La fraction des paquets de données RTP de la source SSRC_n perdus depuis le précédent paquet envoyé de type SR ou de RR. Elle est notée de façon équivalente à prendre la partie entière après multiplication de la fraction par 256. Cette fraction est définie pour être le nombre de paquets perdus divisés par le nombre de paquets prévus. Si la perte est négative (à cause des duplications de paquets), la fraction perdue est placée à zéro. Notez qu'un récepteur ne peut pas indiquer si des paquets ont été perdus

après le dernier reçu, et qu'il n'y aura aucun bloc de rapport de réception publié pour une source si tous les paquets de cette source envoyée pendant la dernière intervalle de temps ont été perdus.

- **cumulative number of packets lost** : 24 bits
  - Le nombre total de paquets de données RTP de la source SSRC_n perdus depuis le commencement de la réception. Ce nombre est défini pour être le nombre de paquets attendus moins le nombre de paquets actuellement reçus, sachant que le nombre de paquets reçus inclut les paquets reçus en retard et les paquets dupliqués. Ainsi les paquets arrivant en retard ne sont pas comptés comme perdu, et la perte peut devenir négative si il y a des duplications.

- **extended highest sequence number received** : 32 bits
  - Les 16 bits de poids faible contiennent le numéro de séquence le plus grand reçu dans un paquet de donnée RTP par la source SSRC_n, et les 16 bits les plus signifiants qui prolongent ce numéro de séquence correspondent au nombre de cycles en relation avec ce numéro de séquence.

- **last SR timestamp (LSR)** : 32 bits
  - Les 32 bits du milieu des 64 de l'estampille NTP, reçu comme élément du paquet de rapport d'expéditeur RTCP le plus récent (SR) de la source SSRC_n. Si aucun SR n'a été encore reçu, le champ est placé à zéro.

- **delay since last SR (DLSR)** : 32 bits
  - Le retard, exprimé par unité de 1/65536 seconde, correspondant au temps entre la réception du dernier paquet SR de la source SSRC_n et l'envoi de ce bloc de rapport de réception. Si aucun paquet de SR n'a été reçu pour un SSRC_n donné, le champ de DLSR est placé à zéro.

b. RR : Paquet RTCP relatif au rapport de réception

Le format des paquets de rapport de réception (RR) est le même que ceux de rapport d'émission, à l'exception que leur champ type est une constante de valeur 201, et que les cinq informations de rapport d'émission sont omises. Les champs restant ont la même signification que dans les paquets de rapport d'émission.

A noter que des profils supplémentaires peuvent être définis avec des extensions dans les rapports d'émission ou de réception dans le cas où des informations supplémentaires doivent être rapportées régulièrement.

D. Présentation de la JMF

Dans un premier temps nous allons vous présenter les différentes étapes nécessaires à la transimission et à la réception d'un flux RTP. A noter que cette section a pour simple but de vous aider à assimiler les mécanismes utilisés, pour de plus amples informations nous vous invitons fortement à consulter l'aide en ligne http://java.sun.com/.
Dans un second temps vous verrez les outils de la JMF permettant d'utiliser le protocole RTCP. Cette présentation sera aussi l'occasion de confronter la JMF aux protocoles RTP/RTCP, et ainsi de pouvoir justifier notre choix pour cette librairie.

1. Description de la transmission et de la réception RTP avec la JMF

i. Capture de flux vidéo et audio

- Accéder à un dispositif de capture (Capture Devices):

Pour capturer un flux vidéo à l'aide de la JMF, il faut pouvoir sélectionner le périphérique de capture. CaptureDeviceManager est une sorte de registre central pour tous les dispositifs de capture disponibles et fonctionnant sous la JMF. On peut obtenir une liste de ces dispositifs en appelant la méthode getDeviceList(). En mettant null en argument de cette méthode, celle-ci renvoie la liste complète des périphériques de capture. Chaque dispositif est représenté par un objet de type CaptureDeviceInfo. Pour obtenir un objet CaptureDeviceInfo associé à un dispositif particulier, il suffit de rentrer la ligne suivante :

```
CaptureDeviceInfo deviceInfo =
CaptureDeviceManager.getDevice(« nomDuDispositif ») ;
```

- Capturer des données Multimédia (Media Data):

Pour capturer des données Multimédia à partir d'un dispositif particulier, il suffit d'obtenir son MediaLocator associé à partir de son objet CaptureDeviceInfo. On peut ensuite utiliser ce MediaLocator pour constuire un Player ou un Processor directement, ou utiliser ce MediaLocator pour construire un DataSource qu'on peut mettre en entrée d'un Player ou d'un Processor.

Dans le cadre de notre projet, nous devions développer un logiciel afin que celui-ci puisse être optimisé pour les systèmes d'exploitations de type Windows. Or très peu de webcams sont compatibles avec la JMF fonctionnant sous Windows. Il se trouve que les webcams mises à notre disposition n'en font pas partie, nous ne pouvions donc les reconnaître avec CaptureDeviceManager. Nous avons alors directement localisé la source à l'aide d'un MediaLocatore contenant l'URL d'un matériel de capture vidéo : « vfw://0 ». Ceci ne pose aucun problème dans le bon déroulement de notre logiciel. La seule différence est dans le cas où plusieurs matériels de capture vidéo sont disponibles sur une machine, le choix du périphérique ne pourra être fait qu'avec les interfaces de Windows.

ii. Transmission du RTP avec l'aide de RTPManager.

Nous ne nous intéressons ici que du cas d'une transmission unicast.
Pour pouvoir transmettre un flux à partir d'un DataSource il faut créer un Processor et le faire passer par différentes étapes :

- la configuration qui obtient des informations sur le format des données d'entrée. Dans le cas où la vidéo et le son sont envoyés dans un même flux il faut aussi démultiplexer le flux d'entrée.
- formater les pistes du Processor.
- La réalisation du processor.
- Démarrer le processus.

- Configuration :

Il suffit simplement d'appeler la méthode configure() du Processor. Cette méthode n'est pas bloquante et n'attend donc pas que le Processor soit configuré pour continuer, ce qui induit qu'on pourrait obtenir des erreurs si par la suite d'autres méthodes font appel à ce même Processor. On va pour cela utiliser la méthode getState() associée à un mécanisme d'attente passive, qui est de synchroniser la méthode configure() avec un mutex.

- Mettre les pistes dans un format supporté :

A noter que nous aurons au maximum deux pistes (une vidéo et une audio)
Il faut mettre chaque piste du Processor dans un format supporté, dans le cas contraire nous pourrions avoir des erreurs et le programme ne pourra pas transmettre le flux.

Pour cela, il suffit d'appeler la méthode setFormat sur chaque piste, et de mettre en paramètre un objet étendant la classe format (objet VideoFormat ou AudioFormat).

Il existe 4 formats audio standards RTP :

- ULAW_RTP = " JAUDIO_G711_ULAW/rtp "
- DVI_RTP = "dvi/rtp"
- G723_RTP = "g723/rtp"
- GSM_RTP = "gsm/rtp"

Il existe de même 3 formats vidéo standards RTP :

- JPEG_RTP = " jpeg/rtp "
- H261_RTP = "h261/rtp"
- H263_RTP = "h263/rtp"

- Réalisation :

Il faut faire de même que la configuration avec la méthode realize() qui n'est pas bloquante.

- Démarrage du Processus :

On utilise la méthode start() du Processor pour le démarrer.

2. La JMF et le protocole RTCP

L'API JMF (Java Media Framework) de Java gère donc les protocoles RTP et RTCP, et permet de récupérer les statistiques liées à ces deux protocoles lorsqu'une session est en cours.

- Création du RTPManager :

Pour cela on instancie un nouveau RTPManager, en appelant la méthode newInstance() du RTPManager. Ensuite, de la même manière que sur le serveur, on appelle les méthodes initialize() et addTarget() pour le configurer. Enfin, on utilise la méthode addReceiveStreamListener pour définir la classe qui réceptionnera l'événement ReceiveStreamEvent.

- Gestion de l'événement ReceiveStreamEvent :

Cet événement est produit lorsque l'on reçoit un flux. Il faut dans un premier temps s'assurer que c'est un nouveau flux. Ensuite, on récupère le DataSource de ce flux et

on crée un Player avec. Enfin si, c'est un flux vidéo, il ne nous reste plus qu'à afficher le composant visuel.

La classe **javax.media.rtp.RTPManager** va fournir deux méthodes permettant d'accéder à ces statistiques :

- `getGlobalReceptionStats()`
  Cette méthode permet d'obtenir les statistiques sur la réception des paquets.

- `getGlobalTransmissionStats()`
  Cette méthode permet d'obtenir les statistiques sur la transmission des paquets.

Intéressons nous maintenant au retour de ces fonctions.

- La méthode `getGlobalReceptionStats()` retourne un objet de type GlobalReceptionStats qui fournit des informations via les méthodes suivantes :

  - `int` **`getBadRTCPPkts`**`()`
    Le nombre total de paquets RTCP reçus dont l'entête était incorrecte

  - `int` **`getBadRTPkts`**`()`
    Le nombre total de paquets RTP reçus dont l'entete était incorrecte

  - `int` **`getBytesRecd`**`()`
    Le nombre total d'octets reçus sur la session RTP avant la validation des paquets.

  - `int` **`getLocalColls`**`()`
    Le nombre total de collisions locales identifiées par le RTPSM

  - `int` **`getMalformedBye`**`()`
    Le nombre total de paquets RTCP "Bye" invalides reçus par le RTPSM.

  - `int` **`getMalformedRR`**`()`
    Le nombre total de paquets RR invalides reçus par le RTPSM.

  - `int` **`getMalformedSDES`**`()`
    The Le nombre total de paquets SDES invalides reçus par le RTPSM.

  - `int` **`getMalformedSR`**`()`
    Le nombre total de paquets SR invalides reçus par le RTPSM.

- `int` **`getPacketsLooped`**`()`
  Le nombre total de paquets bouclés identifiés par le RTPSM.

- `int` **`getPacketsRecd`**`()`
  Le nombre total de paquets RTP et RTCP reçus sur la session RTP avant la validation.

- `int` **`getRemoteColls`**`()`
  Le nombre total de collisions distantes identifiées par le RTPSM.

- `int` **`getRTCPRecd`**`()`
  Le nombre total de paquets RTCP reçus sur la session RTP avant la validation d'entete.

- `int` **`getSRRecd`**`()`
  Le nombre total de SR reçus sur la socket RTCP.

- `int` **`getTransmitFailed`**`()`
  Le nombre de paquets qui ne sont pas arrivés.

- `int` **`getUnknownTypes`**`()`
  Le nombre total de paquets RTCP non reconnus par le RTPSM.

ϖ La méthode `getGlobalTransmissionStats()` retourne un objet de type GlobalTransmissionStats qui fournit des informations via les méthodes suivantes :

- `int` **`getBytesSent`**`()`
  Le nombre total d'octets envoyés sur la socket de session RTP.

- `int` **`getLocalColls`**`()`
  Le nombre total de collisions locales identifiées par le RTPSM.

- `int` **`getRemoteColls`**`()`
  Le nombre total de collisions distantes identifiées par le RTPSM.

- `int` **`getRTCPSent`**`()`
  Le nombre total de paquets RTCP envoyés par le RTPSM.

- `int` **`getRTPSent`**`()`
  Le nombre total de parquets RTP transmis sur la socket de session RTP.

`int` **`getTransmitFailed`**`()`
Le nombre total de paquets qui n'ont pas pu être transmis pour X raison.

CONCLUSION

A travers cet exercice, nous avons pu avoir un aperçu de l'impact réel de la réservation de ressource sur Internet et de comprendre les mécanismes associés. Assurer un service temps réel de bonne qualité et de bout en bout sur un réseau aussi vaste que l'Internet est un problème délicat. Cela dit, des solutions existent et s'interconnectent avec plus ou moins de facilité. Ce que nous avons réalisé sur deux réseaux de deux machines chacun, fonctionnera selon le même principe sur un une multitudes de réseaux de plusieurs centaine de machines.

Par les mesures de l'application de visioconférence, nous avons pu quantifier réellement les impacts de la réservation sur des flux de différentes natures. Si la pénurie en bande passante pose toujours le principal problème, des moyens de le contourner pour assurer certains services existent et sont mis en place aujourd'hui.

Il faut toutefois souligner que les applications multimédias sur l'Internet restent encore en retrait, et sont utilisées par le grand public. Une fois que tout le monde y aura recours, les solutions en place seront-elle toujours suffisantes ?

BIBLIOGRAPHIE

Rapports :

- F. Armando, P. Casenove, S. Falquier, L. Fert – *Développement en Java d'une appliaction de visioconférence et mise en œuvre sur une plate forme RSVP* – 2004
- *RTP: A Transport Protocol for Real-Time Applications* (1996) – RFC 3550
- *RTP: A Transport Protocol for Real-Time Applications* (2003) – RFC 1889

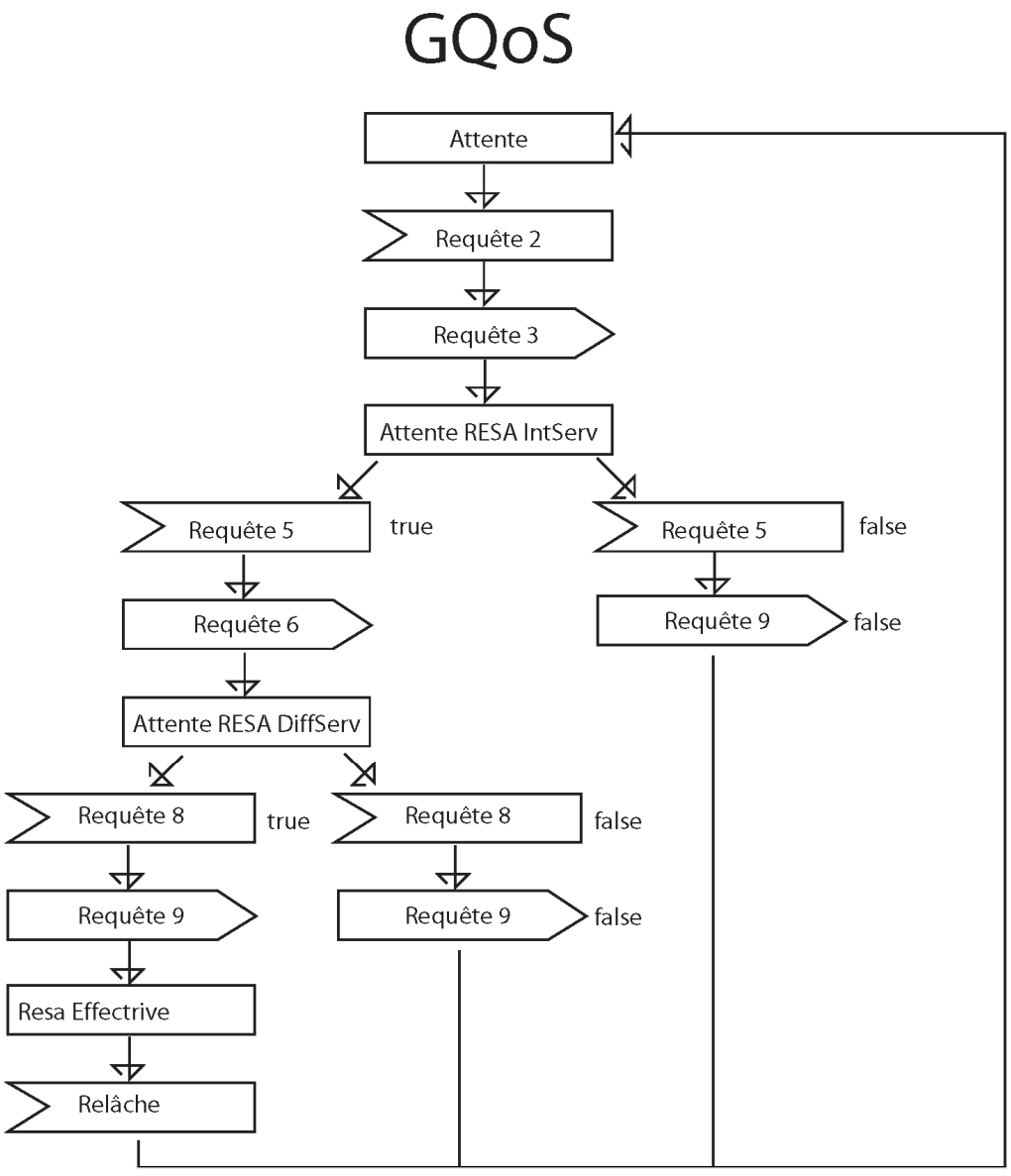
Internet :

- *Linux Advanced Routing & Traffic Control*
  http://lartc.org/
- *Routage avancé et du contrôle de trafic sous Linux* -
  http://www.linux-france.org/prj/inetdoc/guides/Advanced-routing-Howto/
- *RSVP protocol and QoS APIs*
  http://publib.boulder.ibm.com/iseries/v5r2/ic2924/index.htm?info/rzak8/rzak8rsvp.htm
- IETF : DiffServ Charter
  http://www.ietf.org/html.charters/diffserv-charter.html
- Differentiated Services on Linux
  http://diffserv.sourceforge.net/
- Le Modèle DiffServ
  http://www-l2ti.univ-paris13.fr/~fourmaux/stages/DESS_cisco/index2.htm
- API de la JMF version2.1.1
  http://java.sun.com/products/java-media/jmf/2.1.1/apidocs/
- API de JFreeChart
  http://www.jfree.org/jfreechart/javadoc/
- Ensemble de cours sur la JMF
  http://www.laas.fr/~gauriol/

ANNEXES

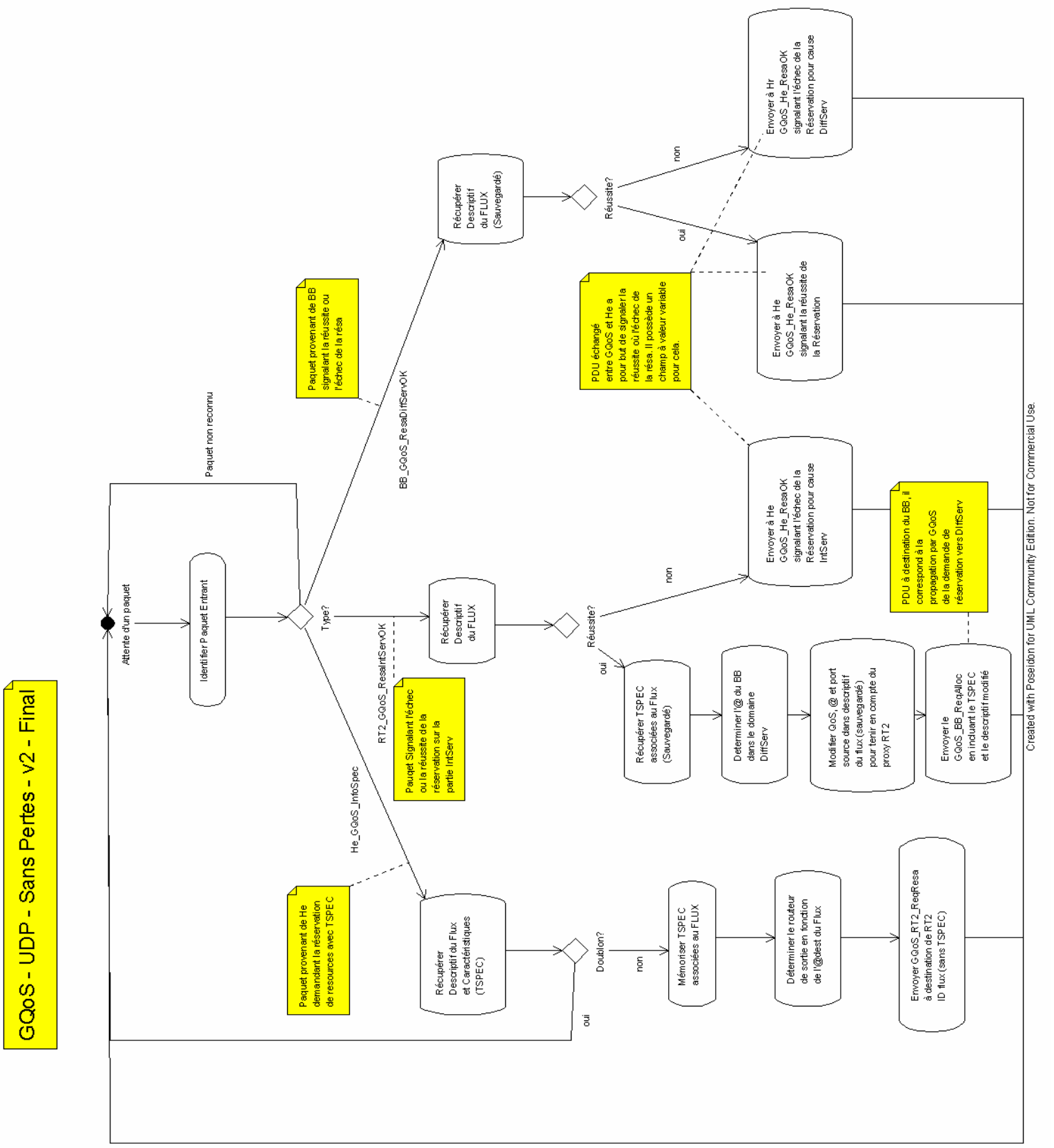
Table des Annexes

1. Machine à Etat GQoS

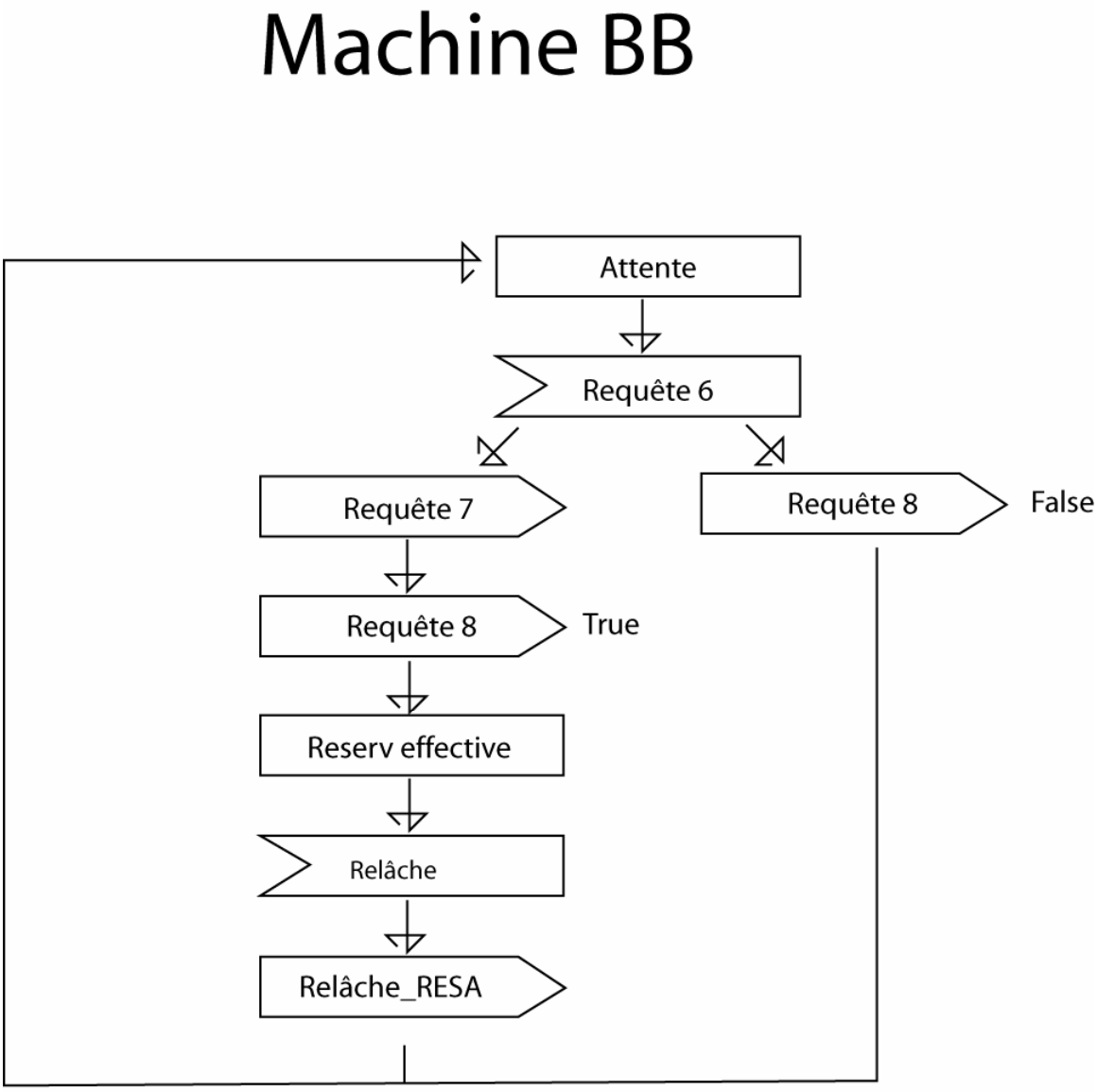


Requête 2	He_Gqos_InfoSpec
Requête 3	Gqos_Rt2_ReqResa
Requête 5	Rt2_Gqos_ResaIntservOk
Requête 6	Gqos_Bb_ReqAlloc
Requête 8	Bb_Gqos_ReqDiffServOk
Requête 9	Gqos_He_RepReservOk

1. Diagramme d'activité GQoS

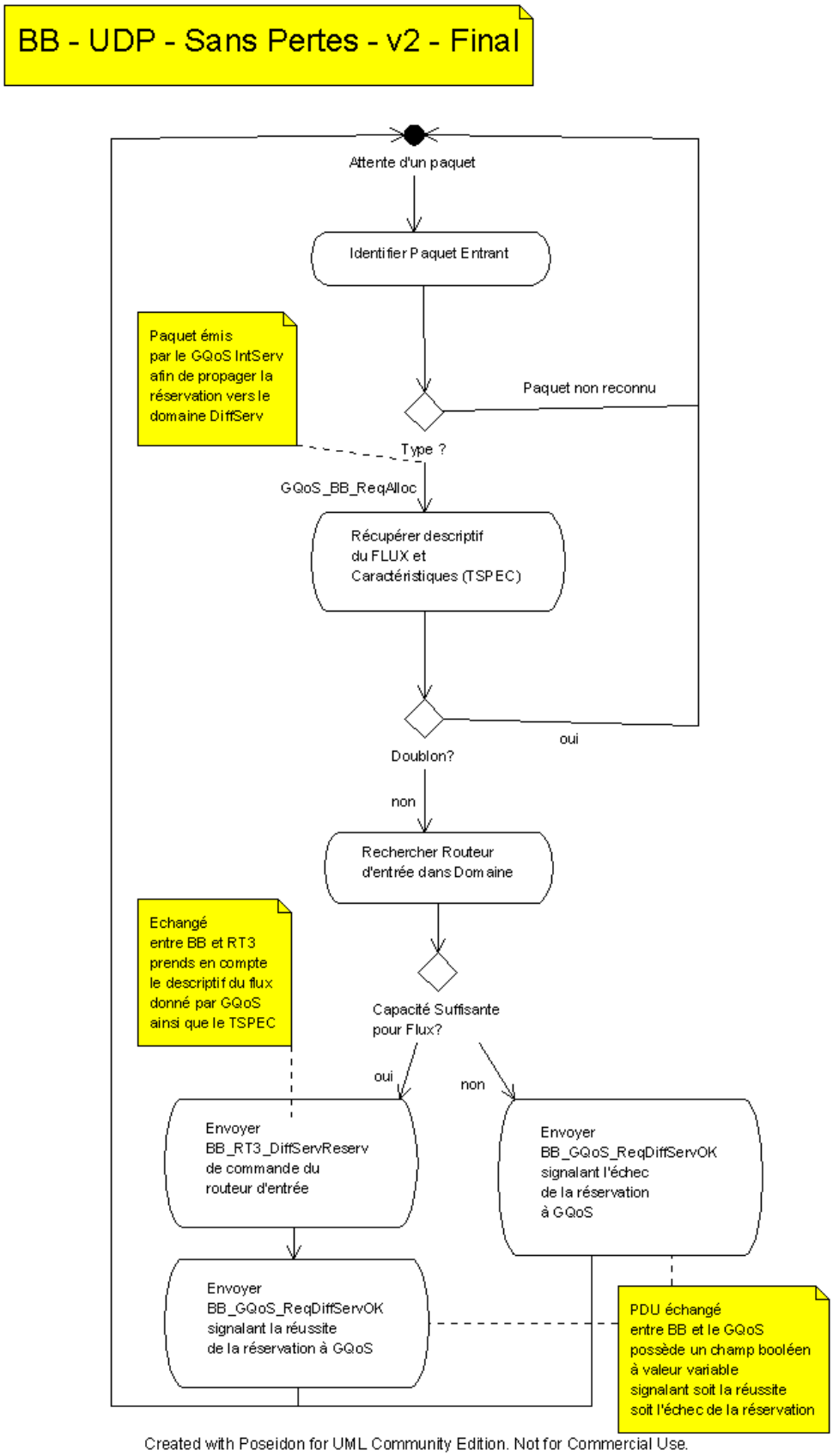


Machine à Etat BandWidth Broker

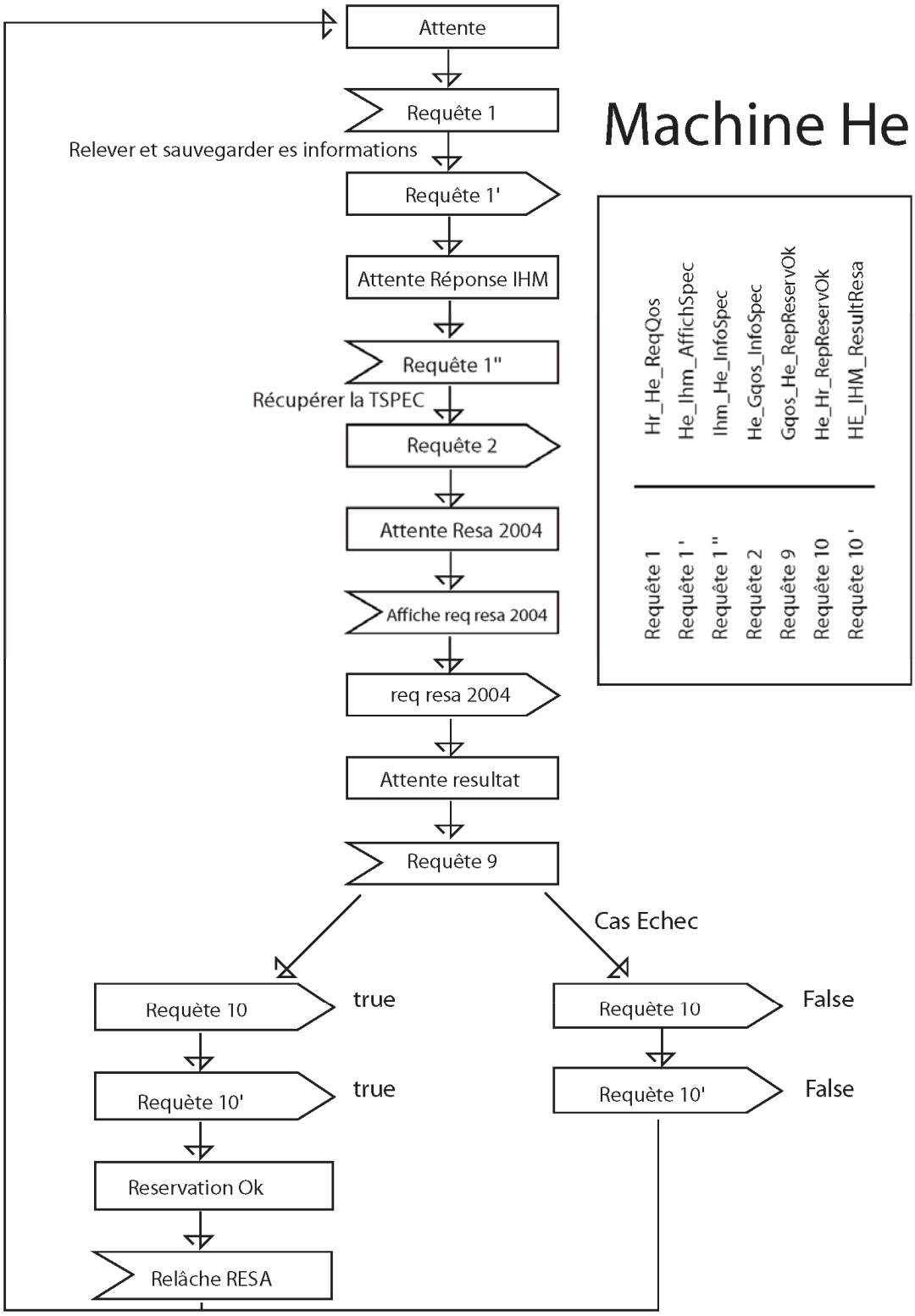


Requête 6	Gqos_Bb_ReqAlloc
Requête 7	Bb_Rt3_DiffServReserv
Requête 8	Bb_Gqos_ReqDiffServOk

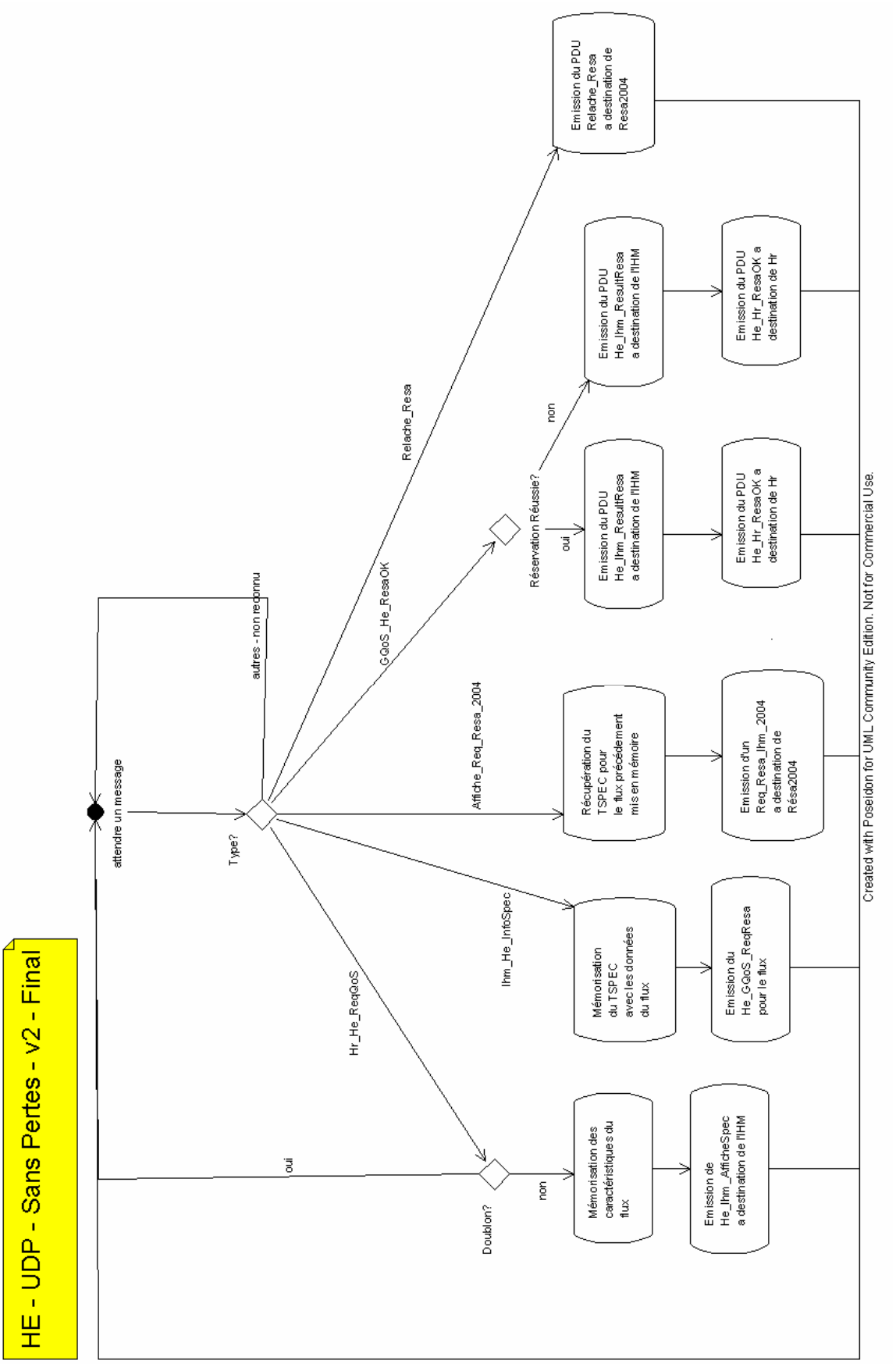
3. Diagramme d'activité BandWidth Broker



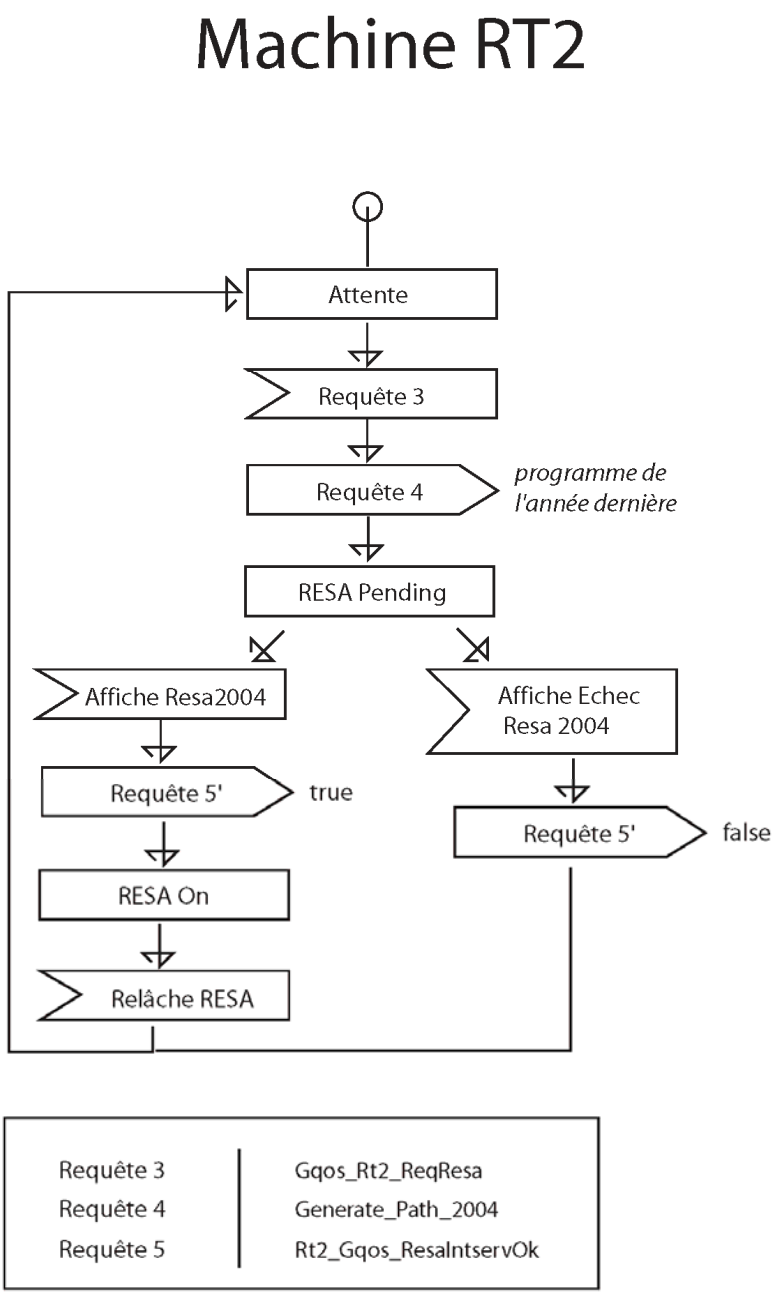
4. Machine à Etat He



5. Diagramme d'activité He

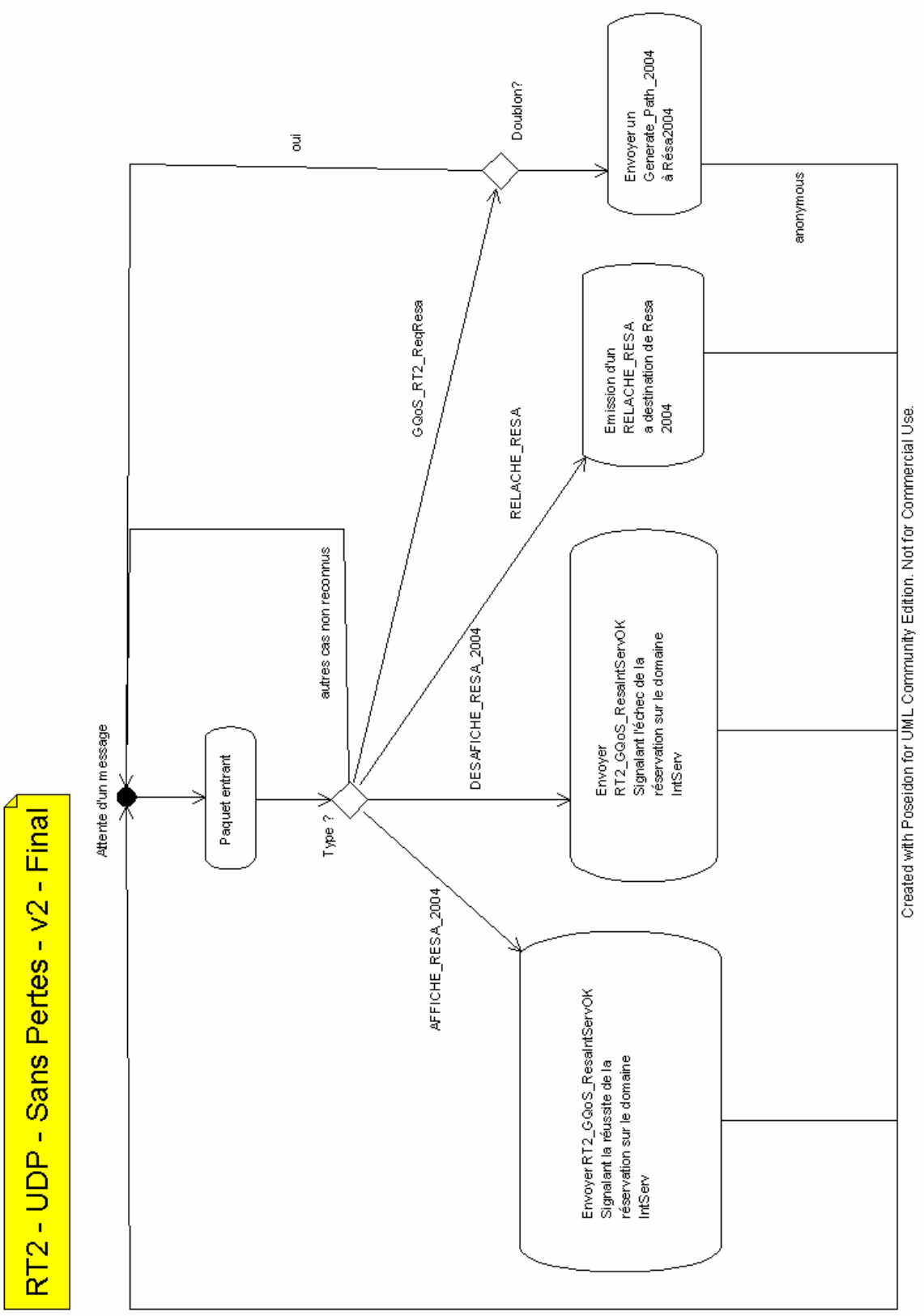


6. Machine à Etat RT2

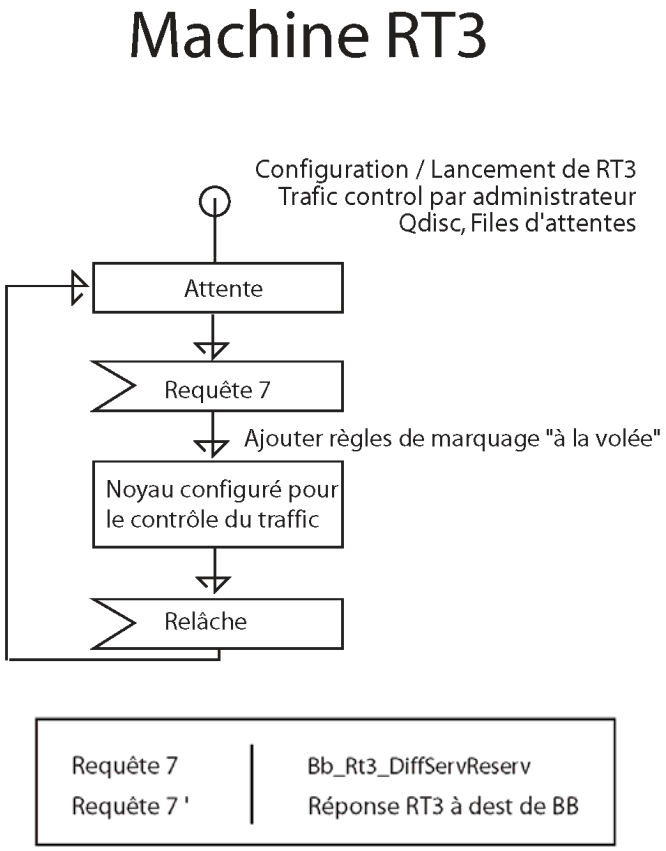


Requête 3	Gqos_Rt2_ReqResa
Requête 4	Generate_Path_2004
Requête 5	Rt2_Gqos_ResaIntservOk

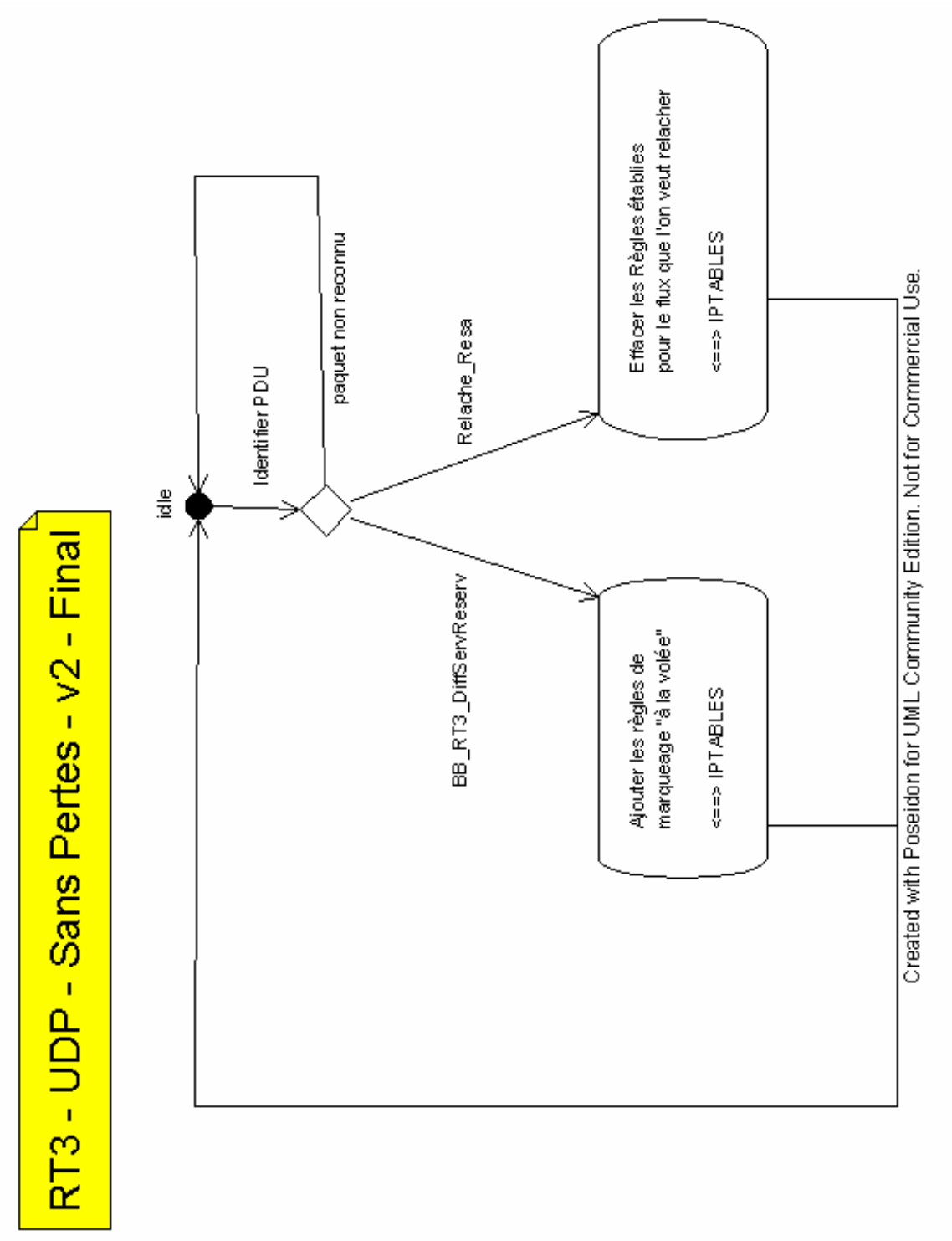
7. Diagramme d'activité RT2



8. Machine à Etat RT3



9. Diagramme d'activité RT3



10. Les différentes implantation en RT2/RT3

L'IMPLEMENTATION DU PROXY UDP EN RT2/RT3 : Via l'utilisation des ports

Première Réflexion

« Pour établir une qualité de service il faut tout d'abord sélectionner les flux qu'on veut différencier. »

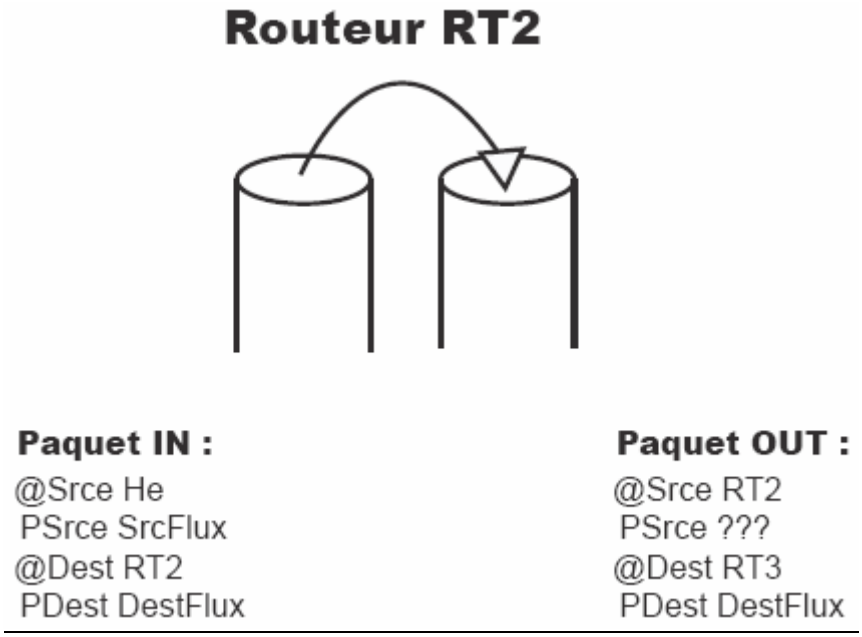
Le proxy est configuré pour que sur réception d'un paquet sur le numéro de port correspondant à un flux donné, on aiguille les paquets sur un port source de numéro fixé par la qualité de service souhaitée et à destination de HR. Par défaut, si aucune réservation n'est trouvable, on aiguille sur le port Best Effort (BE) par exemple 8000. Si GS est demandé, on aiguille le paquet sur le numéro de port 8001 par exemple, etc…

A- En distinguant RT2 de RT3 :

[SCHEMA]

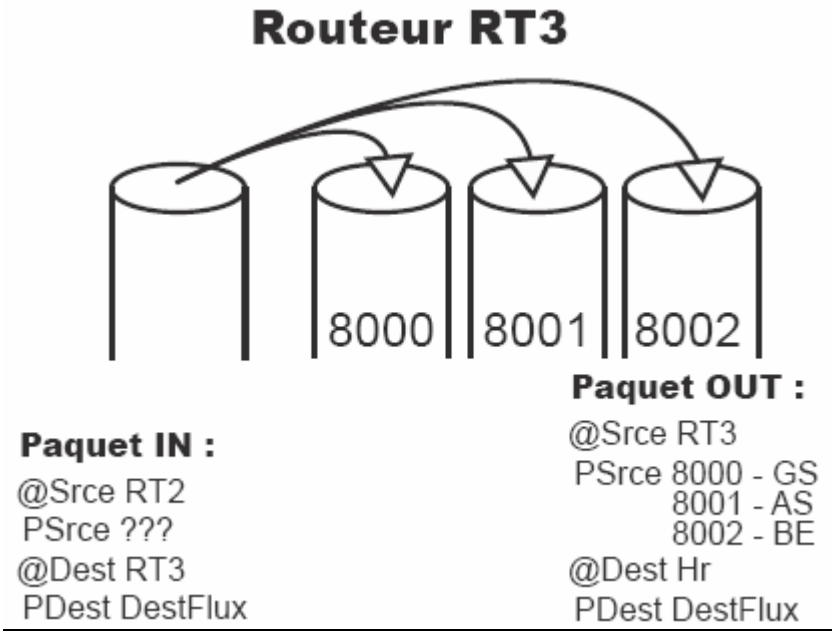
Tout d'abord, rappelons que l'on caractérise un **flux** par les @IPdest et @IPsrc ainsi que par les ports dest et srce (**@IPdest,@IPsrce,Pdest,Psrce**). Ce flux reflète les données que l'on souhaite acheminer de l'émetteur vers le récepteur.

En RT2 :



Comme on l'a vu, RT2 doit jouer le rôle de proxy afin de se faire passer pour l'hôte HR dans le but d'effectuer une réservation de ressources (RSVP) entre HE et RT2 sur le domaine IntServ.
Lors de l'envoie des données, HE envoie les données à destination de HR sur le port = PortHR de réception des paquets. Ainsi *: PortDest = Port_HR et RT2 se met en attente sur ce numéro de port*.
RT2 bascule les paquets sur l'interface de sortie du domaine IntServ. Ces paquets ont pour destinataire : @IP_RT3 et Port_HR.

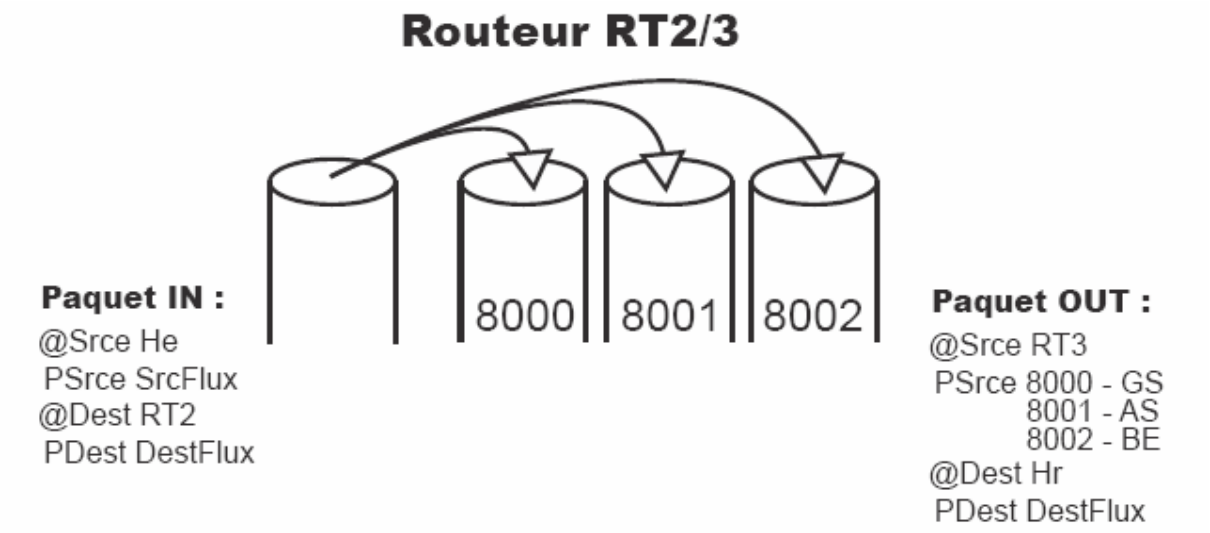
<u>En RT3 :</u>



Les paquets provenant de RT2 étant à destination de HR, RT3 écoute sur un port = Port_HR.
Selon la qualité souhaitée, on aiguille les paquets sur un port d'émission correspondant à la qualité de service réservée pour ce flux dans le domaine DiffServ. Ce numéro de port est déterminé par l'administrateur du proxy et reste fixe ensuite, en effet, les routeurs effectuant un ordonnancement sélectif basé sur ce numéro de port distinctif, le modifier demanderait la modification non seulement du proxy mais aussi de tous les routeurs. Donc, si on suppose que le port 8001 est réservé pour une très bonne qualité, et si HR souhaite une très bonne qualité pour deux flux différents (audio et vidéo), alors au niveau de RT3, on bascule les paquets de ces flux sur le port d'émission 8001. En conséquence, on se rend compte que le buffer du socket d'émission va être partagé entre plusieurs flux potentiels. Il faudra donc mettre en place un système de contrôle du buffer (ipc, verrous ou autres).

Au final, les paquets seront toujours à destination de HR et le fait d'assigner des ports sources en fonction de qualité de service permettra, le marquage, et ainsi l'ordonnancement sélectif des flux dans le domaine DiffServ.

B- En réunissant RT2 et RT3 en un seul proxy :



Le principe reste le même que précédemment. On réunit simplement les deux fonctions en une seule. Donc au lieu de faire basculer les paquets sur l'interface de sortie de RT2, on les redirige directement en fonction de la qualité de service vers un socket d'émission particulier.

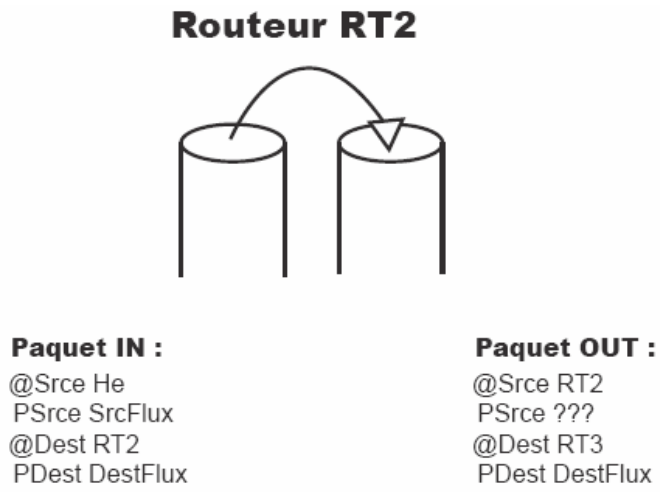
EN BREF …

RT2/RT3 étant routeur de bordure de DiffServ (ingress en l'occurrence), reçoit les paquets sur un port d'écoute. Il analyse les paquets, et selon le numéro de port de provenance des paquets, il déduit la qualité de service à fournir. En effet, on a réservé

des numéros de port par flux, donc les paquets sont redirigés sur un numéro de port source correspondant à qualité de service à fournir dans Diffserv.
Pour le fonctionnement dans DiffSerc, il faudra marquer les paquets en entrée du domaine de telle sorte qu'ils soient interprétables par tous les routeurs du domaine DiffServ.

L'IMPLEMENTATION DU PROXY UDP EN RT2 ET MARQUAGE « A LA VOLEE »
Seconde réflexion

Mise en place d'un proxy UDP en RT2 :



Le principe est le même que précédemment. L'émetteur HE devra s'adresser à RT2 et les paquets pour le flux (He/Hr) seront aiguillés sur l'interface de sortie de RT2.
Les paquets sont donc à destination de RT2 qui écoute sur le port = PortDest du flux en question. Le proxy se charge d'aiguiller les paquets à destination de HR sur ce même port.

Le marquage « à la volée » en RT3 et la mise en place du Policing :

Contrairement à la précédente réflexion, on ne redirige plus les paquets en fonction du port selon lequel on distinguait la qualité de service. Ici, on effectue un marquage « à la volée » car on modifie directement les paquets entrant sur l'interface d'entrée du routeur RT3 (routeur ingress). Ce marquage permet d'affecter une valeur au champ TOS.
Il faut en plus définir un « policing » pour caractériser et mettre en place la gestion des files d'attente sur l'interface de sortie du routeur RT3 (interface connecté au réseau DiffServ).
Nous avons orienté nos recherches sur TC, et nous avons étudié les différents outils à notre disposition pour effectuer le marquage que nous désirons.
Nous avons remarqué l'existence du mkfifo fast, nous permettant de gérer les trois qualités de service que nous nous sommes fixés.

LE RELACHEMENT D'UNE RESERVATION

Dans le domaine Insterv, la réservation est maintenue par l'envoie régulier de PDU PATH. Lorsqu'un relâchement à lieu, le gestionnaire GQoS est alors mis au courant et prévient le BB (gestionnaire pour le domaine DiffServ).

Nous avons adopté la gestion des PDU selon le protocole UDP. Nous aurions pu choisir des communications en TCP, qui reposant sur un mécanisme de connexion, aurait allégé le protocole pour la considération du relâchement de la réservation. Néanmoins, il n'est pas plus compliqué de travailler avec des flux UDP. Il nous suffit de spécifier le protocole à mettre en place lors d'un relâchement de réservation. Dans un premier temps, on considérera que le relâchement a lieu sur demande explicite de BB. Dans une version ultérieure, il nous sera possible d'ajouter un timer en RT3 suite à une réservation enclenchée et de relâcher cette réservation si le BB ne renvoie pas un PDU pour maintenir la réservation avant la fin du TTL.

La fin de la réservation est bien entendu signalée aux hôtes d'extrémités, concernés par la réservation.

> *Remarque : dans l'implémentation de DiffServ, dès lorsqu'un flux est accepté, le BB lui affecte une durée de cession déterminée. Du coup, si le flux continue de passer, on lui affectera non plus la priorité qui lui avait été affecté mais une priorité de Best Effort !!*
> *Il est donc important de mettre en place un système de mise à jour qui maintienne le flux. Dans IntServ, on envoie des pdu path à intervalles régulier pour maintenir le flux, et si on ne reçoit plus rien, la réservation est relâchée. Dans notre modèle, on pourrait faire suivre la requête PATH, c'est-à-dire, que sur réception d'un PDU PATH de maintien pour IntServ, on envoie un message au BB pour lui signaler que l'on veut encore émettre pour le flux. Donc, il faut effectuer une demande pour qu'il réinitialise le compteur pour le flux demandé.*

11. Documentation sur l'utilisation de TC

NOTE – UTILISATION DE QDISC, PRIO | HTB, FILTRE TC

CONTROL DU TRAFIC AU NIVEAU DE RT3 SUR L'INTERFACE CONNECTE AU RESEAU DIFFSERV

Recherche et synthèse de ce rapport d'après Fok Frederic, complément à la synthèse « iptables-netfilter-tc » d'après VanWambeke Nicolas pour l'équipe Réseau :
Fok Frederic, Van Wambeke Nicolas, Molinier Nicolas

NOTE – UTILISATION DE QDISC, PRIO | HTB, FILTRE TC

CONTROL DU TRAFIC AU NIVEAU DE RT3 SUR L'INTERFACE CONNECTE AU RESEAU DIFFSERV

SOMMAIRE :

OBJECTIFS ET CONTEXTE

Au niveau du routeur Ingress, routeur de bordure du réseau DiffServ, nous devons mettre en place la réservation pour ce réseau. Pour cela, nous avons marqué les paquets en fonction du champ TOS en entrée du routeur RT3 (Marquage par le champ TOS). L'objectif est alors de contrôler le trafic entrant dans le réseau DiffServ en fonction de sa nature et de ses caractéristiques. Il s'avère ainsi nécessaire de gérer les priorités selon la qualité de service du flux demandé. Or, nous avons défini trois types de services dont BE qui est le service de priorité la plus faible.
Il existe plusieurs gestions des files d'attentes possibles avec ou sans classe. Nous allons donc expliquer les étapes (commandes sous Linux) à suivre pour configurer nos files d'attente dans l'objectif de distinguer les flux et leur priorité.
Dans ce rapport, j'aborde les notions du contrôle de trafic en rapport avec notre projet. Pour plus de clarté et afin de faciliter la compréhension des termes employés dans cette synthèse, les notions particulière à ce domaine sont données en annexe. J'ai tenté, pour chaque commande utilisée sous linux, d'expliquer au mieux l'action que chacune d'elles effectue.

1- ETAPE D'INITIALISATION

Avant de paramétrer les gestionnaire, il est important de supprimer (*del*) toutes les règles. Pour cela, il suffit simplement de supprimer la discipline à la racine :

<tc qdisc del root dev eth0>

Explication: ***qdisc***[*] est un gestionnaire de file d'attente. Ici, on précise qu'il s'agit du gestionnaire root et on précise l'interface (*dev*) utilisé par le gestionnaire : eth0.

2- TC, NOTION DE CLASSE ET MISE EN FILE D'ATTENTE

2.1- HIERARCHIE DE CLASSES [Un exemple d'arbre est représenté en 3.2.2.]

Il ne faut *pas* imaginer le noyau comme étant au sommet de l'arbre et le réseau en dessous, ce qui n'est justement pas le cas. Les paquets sont mis et retirés de la file d'attente à la racine du gestionnaire, qui est le seul élément avec lequel le noyau dialogue.
Ces descripteurs sont constitués de deux parties : un nombre majeur et un nombre mineur : **<major>:<minor>**. Il est habituel de nommer le gestionnaire racine `1:`, ce qui est équivalent à `1:0`. Le nombre mineur d'un gestionnaire de mise en file d'attente est toujours 0.
Les classes doivent avoir le même nombre majeur que leur parent. Le nombre majeur doit être unique à l'intérieur d'une configuration egress ou ingress.

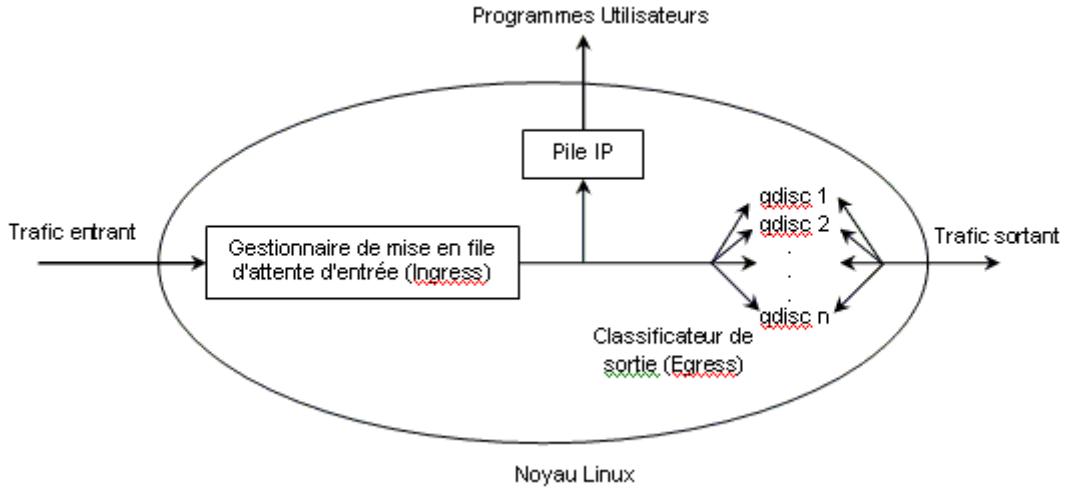
2.2- L'UTILISATION DE TC

L'utilitaire `tc` permet de transmettre au noyau les paramètres que l'on souhaite mettre en place.
C'est un utilitaire de la famille `iproute2`. La syntaxe utilisée par ce programme est :

tc [OPTIONS] OBJECT { COMMAND | help }
where OBJECT := { qdisc | class | filter }
OPTIONS := { -s[tatistics] | -d[etails] | -r[aw] | -b[atch] file }

Le principe est le suivant :
1. Définition d'une architecture d'ordonnancement sur le périphérique
2. Si l'architecture comporte des classes, définitions des classes
3. Classification des paquets en flux.
4. Affectation des flux de paquets aux classes

2.3- PLUS EN PROFONDEUR : COMPRENDRE LA MISE EN FILE D'ATTENTE



Le trafic du réseau qui entre dans le routeur passe dans le gestionnaire de mise en file d'attente d'entrée ingress qui peut appliquer des filtres à un paquet et décider, le cas échéant, de le supprimer.
Si le paquet est autorisé à continuer, il peut être destiné à une application locale. Il entrera alors dans la couche IP pour être traité et délivré à un programme utilisateur. Le paquet peut être également transmis sans entrer dans une application. Il sera alors destiné au classificateur de sortie egress. Les programmes utilisateurs peuvent délivrer des données qui seront alors transmises et examinées par Egress.
Au niveau du classificateur, le paquet est examiné et mis en file d'attente vers un certain nombre de gestionnaires de mise en file d'attente (qdisc 1, qdisc2, …, qdisc i, …, qdisc n). Par défaut, il n'y a qu'un seul gestionnaire egress installé, pfifo_fast, qui reçoit tous les paquets. C'est ce qu'on appelle la **mise en file d'attente** (*enqueuing*).
Le paquet réside maintenant dans le gestionnaire de mise en file d'attente et attend que le noyau le réclame pour le transmettre à travers l'interface réseau. Le fait de retirer le paquet de la file d'attente est encore appelé *dequeueing*.

Quand le noyau décide qu'il doit extraire des paquets pour les envoyer vers l'interface, le gestionnaire racine `1:` reçoit une requête `dequeue`, qui est transmise à `1:1` et qui, à son tour, est passée à `10:`, `20:` et `30:`, chacune interrogeant leurs descendances qui essaient de retirer les paquets de leur file d'attente. Dans ce cas, le noyau doit parcourir l'ensemble de l'arbre, car seul `30:2` contient un paquet en suposant que 30 : possède une classe fille 30 :2.

En résumé, les classes « emboîtées » parlent uniquement à leur gestionnaire de mise en file d'attente parent ; jamais à une interface. Seul la file d'attente du gestionnaire racine est vidée par le noyau !
Ceci a pour résultat que les classes ne retirent jamais les paquets d'une file d'attente plus vite que ce que leur parent autorise. Et c'est exactement ce que nous voulons : de cette manière, nous pouvons avoir SFQ dans une classe interne qui ne fait pas de mise en forme, mais seulement de l'ordonnancement, et avoir un gestionnaire de mise en file d'attente extérieur qui met en forme le trafic.

3- NOTION DE FILTRAGE

3.1- Utilité et Utilisation :

Le filtrage est un moyen de diriger les flux dans vers des classes définies préalablement ou des files d'attente. On va pouvoir ainsi gérer le trafic et ensuite l'ordonnancer.
Il faut noter que le noyau examine les règles pour chaque paquet. Donc, si on disposait d'un très grand nombre de règles, par exemple 2000, alors il faudrait faire 2000 contrôles pour chaque paquet. Pour des questions de performance, on pourrait mettre des filtres hachés (existence de clés de hachage pour accélérer la recherche). Dans le cadre de notre projet, cette fonction n'est pas utile puisque nous ne définissons pas un grand nombre de règles.

A chaque fois que l'on voudra effectuer un filtrage, on commencera par la ligne de commande suivante : ***tc filter add dev eth0 parent 1:0***
Il existe une panoplie conséquente de règles de filtrage. On pourra filtrer suivant une adresse IP, un port source, une route, une entête tcp ou udp, une classe, etc…

3.2- Utilisation de quelques filtres

- **Filtrage d'une adresse IP source :**

La commande suivante crée un filtre sur l'interface eth0 à la classe parente 1 :0. Elle agit sur le protocol ip et redirige les paquets, provenant de @IP source ou destinataire, vers 1 :1.

```
tc filter add dev eth0 parent 1 :0 \
protocol ip prio 1 u32 match ip <src/dest> <@IP> flowid 1:1
```

- **Filtrage par rapport à l'entête tcp :**

La commande suivante redirige les paquets vers 1 :1 en effectuant la correspondance avec l'entête tcp :

```
tc filter add dev eth0 parent 1:0 \
protocol ip prio 1 u32 match tcp src 80 flowid 1:1
```

- **Filtrage par rapport à un port :**

La commande suivante redirige les paquets à destination du port 22 vers la classe 1 :1.

```
tc filter add dev eth0 parent 1:0 \
protocol ip prio 1 u32 match ip dport 22 0xffff flowid 1:1
```

- **Filtrage sur le champ TOS**

La commande suivante filtre suivant le champ tos et dirige les paquets dont le champ TOS a pour valeur : 0x10 vers la classe 1 :1.

```
tc filter add dev eth0 parent 1:0 \
protocol ip prio 1 u32 match ip tos 0x10 0xff flowid 1:1
```

4- EN UTILISANT LE QDISC PRIO.

4.1- UTILITE

PRIO (gestionnaire de mise en file d'attente) permet de classer les flux selon leur priorité. Cet Ordonnanceur envoie les paquets d'un flux de plus forte priorité avant ceux d'un flux de priorité inférieure. Donc, tant qu'il y a des paquets en attente dans ce flux de forte priorité, on ne tient pas compte des flux suivants.
Dans le cadre de notre étude, PRIO semble bien adapté à la distinction des flux selon la qualité de service. Néanmoins, PRIO peut avoir des effets excessifs puisque rien ne pourra être envoyé dans une classe tant qu'une classe de priorité plus haute aura des paquets à émettre ! De plus, il est possible d'aboutir à des dépassements de capacités ce qui induirait des pertes.
Quand un paquet est mis en file d'attente dans le gestionnaire PRIO, une classe est choisie en fonction des filtres que nous avons placés. Par défaut, trois classes sont créées. Ces classes contiennent par défaut de purs gestionnaires de mise en file d'attente FIFO sans structure interne, mais on peut les remplacer par n'importe quels gestionnaires disponibles.

Pour visualiser l'ordonnancement, on ne peut pas le vérifier avec une commande de traçage sur une classe telle que :

<watch tc –s class dev $DEV> \ ***segmentation fault***

En revanche on peut effectuer un traçage sur les files d'attentes :

<watch tc -s qdisc dev $DEV>

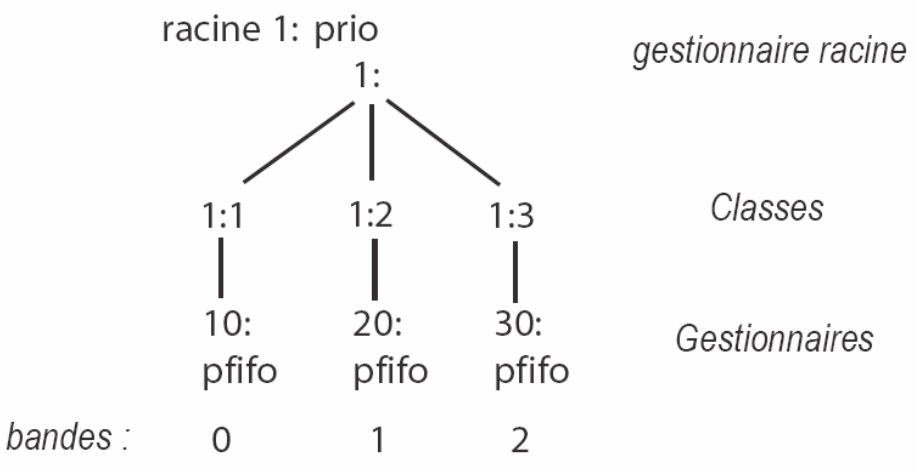
4.2- CONFIGURATION CHOISIE POUR CE GESTIONNAIRE PRIO AVEC FILTRES TC

4.2.1- CODE

```
tc qdisc add dev eth0 root handle 1: prio
tc qdisc add dev eth0 parent 1:1 handle 10: pfifo
tc qdisc add dev eth0 parent 1:2 handle 20: pfifo
tc qdisc add dev eth0 parent 1:3 handle 30: pfifo
tc filter add dev eth0 parent 1 :0 protocol ip prio 1 u32 match ip tos 0x10 0xff flowid 1:1
tc filter add dev eth0 parent 1 :0 protocol ip prio 2 u32 match ip tos 0x8 0xff flowid 1:2
tc filter add dev eth0 parent 1 :0 protocol ip prio 3 u32 match ip tos 0x0 0xff flowid 1:3
```

4.2.2- EXPLICATIONS

Les classes sont organisées par priorité (1 est la priorité la plus forte). Ainsi, la classe 1 :1 aura une priorité plus forte que la classe 1 :2. Pour chaque classe créée, on assigne un gestionnaire en FIFO. Donc le premier paquet entré sera le premier paquet sorti.




```
tc qdisc add dev eth0 root handle 1: prio
```

On attache un gestionnaire de mise en file d'attente (QDISC) ***PRIO*** sur ROOT que l'on nomme 1:0
Cela crée automatiquement les classes 1 :1, 1 :2 et 1 :3

```
tc qdisc add dev eth0 parent 1:1 handle 10: pfifo
```

Un QDISC de type FIFO est attaché à la classe 1 :1 (flux de classe 1) sur l'interface Eth0. On nomme ce gestionnaire 10 : et son parent est 1 :1.

```
tc qdisc add dev eth0 parent 1:2 handle 20: pfifo
```

Un QDISC de type FIFO est attaché à la classe 1 :2 (flux de classe 2) sur l'interface Eth0. On nomme ce gestionnaire 20 : et son parent est 1 :2.

```
tc qdisc add dev eth0 parent 1:3 handle 30: pfifo
```

Un QDISC de type FIFO est attaché à la classe 1 :3 (flux de classe 3) sur l'interface Eth0. On nomme ce gestionnaire 30 : et son parent est 1 :3.

Ajout des filtres :

On rajoute des filtres pour aiguiller les paquets dans les classes correspondant à leur flux. On filtre les paquets selon le champ TOS.

Règle de filtrage pour aiguiller les paquets de forte priorité dans la classe 1.

```
tc filter add dev eth0 \
              parent 1 :0 \
              protocol ip \
              prio 1 \
              u32 \
              match ip tos 0x10 0xff \
              flowid 1:1 \
```

Explications:

Instruction	signification
dev	interface utilise pour l'application du filtre
parent 1 :0	le filtre est associé à la classe racine
protocol ip	les paquets concernés possèdent une entête ip
prio 1	on affecte une priorité 1 au paquet que l'on redirige
u32	base la décision sur les champs à l'intérieur du paquet ip *En générale, il classe les paquets selon les adresses ip, les ports (destination ou source) tc ou udp, le ToS, le protocole.*
match ip tos 0x10 0xff	on vérifie que le paquet possède un champ tos = 0x10
flowid 1 :1	On place le paquet qui répond au critère du filtre dans la file d'attente correspondant à la classe 1 :1

Règles de filtrage pour aiguiller les paquets de moyenne et faible priorités dans la classe 2 et 3

```
tc filter add dev eth0 parent 1 :0 protocol ip prio 2 u32 match ip tos 0x8 0xff flowid 1:2
tc filter add dev eth0 parent 1 :0 protocol ip prio 3 u32 match ip tos 0x0 0xff flowid 1:3
```

4.3- GESTIONNAIRE PRIO SANS FILTRE, EN MODIFIANT LA PRIOMAP

4.3.1- CODE

```
tc qdisc add dev eth0 root handle 1: prio      bands      3      priomap
2212220222222222
tc qdisc add dev eth0 parent 1:1 handle 10: pfifo
tc qdisc add dev eth0 parent 1:2 handle 20: pfifo
tc qdisc add dev eth0 parent 1:3 handle 30: pfifo
```

Pour visualiser (traçage sur les file d'attente), il suffit d'appliquer la commande:

```
watch tc -s qdisc dev eth0
```

4.3.2- EXPLICATION

La priomap (carte des priorités) assigne des classes aux priorités

pfifo_fast utilise la carte de priorité par défaut qui permet d'associer à une valeur du champ TOS une classe. Cette carte par défaut est la suivante et représente les 16 priorités possibles du champ TOS : `1, 2, 2, 2, 1, 2, 0, 0 , 1, 1, 1, 1, 1, 1, 1, 1`

On peut faire correspondre l'une des 16 valeurs de ce champ à une file en fonction du marquage du champ tos.

Interprétation

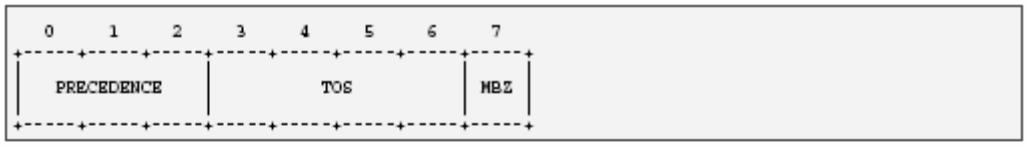
Par défaut, trois classes seulement sont crées (0,1,2). Si on ajoute des classes, il faut alors changer la priomap.
La priorité 0 est relié à la classe 1.
La priorité 2 est relié à la classe 2. …
La priorité 4 est relié à la classe 1. …
La priorité 6 est relié à la classe 0. … ainsi de suite…

Utilité

Nous avions remarqué que le service Best Effort (BE) a une priorité de 0 et cette priorité est reliée à la bande 1 (classe1). Or nous voulions lui donner la priorité la plus faible. On peut donc changer la carte des priorités pour assigner à la priorité 0 une bande supérieure. On prendra 2 comme valeur de bande.
Ce qui donne une carte des priorités : 2, 2, 1, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2

Cette nouvelle carte des priorités permet les associations suivantes :
Priorité 0 = BE bande 2
Priorité 2 = masse bande 1
Priorité 6 = interactive bande 0

Ci-dessous la représentation du champ tos et son interprétation



Les quatre bits TOS (le champ TOS) sont définis comme suit :

Binaire	Décimal	Signification
1000	8	Minimise le Délai (Mininize delay) (md)
0100	4	Maximalise le Débit (Maximize throughput) (mt)
0010	2	Maximalise la Fiabilité (Maximize reliability) (mr)
0001	1	Minimalise le Coût Monétaire (Minimize monetary cost) (mmc)
0000	0	Service Normal

Comme il y a 1 bit sur la droite de ces quatre bits, la valeur réelle du champ TOS est le double de la valeur des bits TOS. `tcpdump -v` -v fournit la valeur de tout le champ TOS, et non pas seulement la valeur des quatre bits. C'est la valeur que l'on peut voir dans la première colonne du tableau suivant :

TOS	Bits	Signification	Priorité Linux	Bande
0x0	0	Service Normal	0 Best Effort	1
0x2	1	Minimise le Coût Monétaire (mmc)	1 Filler	2
0x4	2	Maximalise la Fiabilité (mr)	0 Best Effort	1
0x6	3	mmc+mr	0 Best Effort	1
0x8	4	Maximalise le Débit (mt)	2 Masse	2
0xa	5	mmc+mt	2 Masse	2
0xc	6	mr+mt	2 Masse	2
0xe	7	mmc+mr+mt	2 Masse	2
0x10	8	Minimise le Délai (md)	6 Interactive	0
0x12	9	mmc+md	6 Interactive	0
0x14	10	mr+md	6 Interactive	0
0x16	11	mmc+mr+md	6 Interactive	0
0x18	12	mt+md	4 Int. Masse	1
0x1a	13	mmc+mt+md	4 Int. Masse	1

5- EN UTILISANT HTB (Hierarchical Token Bucket).

5.1- PRINCIPE D'HTB

HTB est en quelque sorte un *Token Bucket Filter* basé sur des classes, d'où son nom.
Ce contrôle hiérarchique permet de diviser une bande passante entre les différents éléments et chacun de ces éléments aura une bande passante garantie, avec la possibilité de spécifier la quantité de bande passante qui pourra être empruntée.
Dans ce mécanisme, chacune des classes possède un seau de jetons. Lorsque le seau d'une classe fille devient vide, cette classe emprunte un jeton à sa classe mère s'il lui reste des jetons. Dans le cas contraire, la classe mère demande à son tour à sa propre classe mère (si elle en a) et ainsi de suite. Si une classe et tous ses ancêtres ont momentanément épuisés leurs seaux respectifs, le flux correspondant à la classe fille est bloqué.

5.2- CODE :

```
tc qdisc add dev eth0 parent root handle 1:0 htb default 30

tc class add dev eth0 parent 1: classid 1:1 htb rate 10mbps ceil 10mbps
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 7mbps ceil 10mbps prio 1
tc class add dev eth0 parent 1:1 classid 1:20 htb rate 2mbps ceil 10mbps prio 2
tc class add dev eth0 parent 1:1 classid 1:30 htb rate 1mbps ceil 1mbps prio 3

tc qdisc add dev eth0 parent 1:10 handle 20: sfq perturb 10
tc qdisc add dev eth0 parent 1:20 handle 30: sfq perturb 10
tc qdisc add dev eth0 parent 1:30 handle 40: sfq perturb 10

tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip tos 0x10 0xff flowid
1:1
tc filter add dev eth0 parent 1:0 protocol ip prio 2 u32 match ip tos 0x8  0xff flowid
1:2
tc filter add dev eth0 parent 1:0 protocol ip prio 3 u32 match ip tos 0x0  0xff flowid
1:3
```

5.3- EXPLICATIONS

Mise en place d'un qdisc en root

```
tc qdisc add dev eth0 parent root handle 1:0 htb default 30
```

Il faut nécessairement mettre en place un gestionnaire pour gérer les files d'attente. Ce gestionnaire de mise en file d'attente (qdisc) est attaché à root et on désigne (*handle*) par 1 :0 le qdisc mis en place.
<default 30> signifie que tous les trafics qui ne sont pas classifiés seront acheminés dans la classe fille 1 :30.

Mise en place d'une classe root

```
tc class add dev eth0 parent 1: classid 1:1 htb rate 10mbps ceil 10Mbps
```

Une **`classe root`** est une classe avec un qdisc htb comme sa mère. Elle peut, comme toutes les classes sous un qdisc, autoriser ses enfants à partager les ressources entre elles.
On aurait très bien pu créer directement les classes filles sans crée de classe root, ***mais*** l'excès en bande passante d'une des classes filles n'aurait pas pu être accessible par les autre classes filles ! (La classe root est noté par **1 : ou 1 :0).**

Le champ **`classid`** indique le handle de la classe en train d'être ajoutée.
Le champ **`parent`** indique le handle de la classe parente.
`Rate 10Mbps` : défini la vitesse de sortie ou le débit en sortie des paquets. Comme il s'agit de la classe root mère, la vitesse maximale est la même que celle en sortie sur l'interface.
`Ceil 10 Mbps` : la bande passante maximum utilisable pour cette classe est de 10Mbps

Mise en place des classes filles

```
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 7mbps ceil 10mbps prio 1
tc class add dev eth0 parent 1:1 classid 1:20 htb rate 2mbps ceil 10mbps prio 2
tc class add dev eth0 parent 1:1 classid 1:30 htb rate 1mbps ceil 1mbps prio 3
```

Nous avons trois classes filles dont le parent est 1:1. Comme précédemment, classid 1 :10, 1 :20 et 1 :30 indique le handle de la classe en train d'être rajoutée. On précise pour chacune d'elle le débit utilisable et le maximum de bande passante utilisable.
Le **`ceil`** peut être plus grand (ou plus petit) que le rate, ce qui permet si une classe présente un excès de bande passante d'en partager avec ses soeurs.
Dans notre cas, on décide de donner une bande passante pour un flux en Best Effort de 1mbps et le flux ne pourra pas bénéficier d'une bande plus grande.
En revanche les flux demandant des qualités de services plus importantes pourront bénéficier des 10 mbits si un de ces 2 flux n'utilisent pas de bande passante !
`Prio 1` : permet de rajouter une priorité aux classes pour privilégier les paquets assignés à cette classe.

Remarque : *on peut aussi définir un burst pour chacune de ces classes filles :*

`burst` *= taille du seau. Elle est égale à la quantité maximale de données pouvant être envoyée d'une classe avant de servir une autre classe.*

On aurait :
```
tc class add dev eth0 parent 1:1 classid 1:10 htb \
rate 7mbit ceil 10mbit burst 20k
```

Mise en place d'un gestionnaire pour les classes filles

```
tc qdisc add dev eth0 parent 1:10 handle 20: sfq perturb 10
tc qdisc add dev eth0 parent 1:11 handle 30: sfq perturb 10
tc qdisc add dev eth0 parent 1:12 handle 40: sfq perturb 10
```

Ces instructions ne sont pas obligatoires. Néanmoins, elles permettent de spécifier comment sont les files d'attentes des classes filles. Les classes se voient alors dotées d'un ordonnancement en SFQ (round robin, plusieurs files d'attentes gérées pour un flux et notion de probabilité relative).
Pourquoi un SFQ ? *(voir annexe pour la définition)*

On peut définir la gestion des files d'attentes pour les classes filles. Ceci est appréciable et nous permet de rendre la gestion des files d'attentes plus équitables pour les flux. Par exemple, si un hôte fait la demande de réservation pour un flux en spécifiant une qualité de service, le débit des paquets en sortie du routeur va être égale à celle que l'on a spécifié pour ce flux. Mais si un deuxième flux demandant une même qualité de service arrive, la bande passante qui lui attribuée n'est pas égale à celle du premier flux. Pourtant, ils demandent tous les deux une même qualité de service. SFQ permet de gérer cela en effectuant un round robin sur plusieurs files attribuées à une même qualité de service.

Mise en place des filtres pour diriger les paquets dans les files d'attente

```
tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip tos 0x10 0xff flowid 1:1
tc filter add dev eth0 parent 1:0 protocol ip prio 2 u32 match ip tos 0x8  0xff flowid 1:2
tc filter add dev eth0 parent 1:0 protocol ip prio 3 u32 match ip tos 0x0  0xff flowid 1:3
```

La mise en place des filtres s'effectue de la même façon que précédemment. On effectue un filtrage sur le champs TOS et on redirige les paquets vers les classes en fonction de la valeur du tos.
Le paramètre **`u32`** est un nom de *classeur* (*classifier*), ici, le classeur basé sur le contenu de l'entête IP.
Les paramètres **`match`** qui le suivent précisent la sélection effectuée par le classeur. Le paramètre **`flowid`** est le handle de la classe destination des paquets sélectionnés par le classeur.

5.4- DES COMMANDES UTILES POUR L'ANALYSE ET LA SIMULATION

Pour visualiser les qdisc : **tc -s -d qdisc show dev eth0**
Pour visualiser les classes : **tc -s -d class show dev eth0**
Pour visualiser les filters : **tc –s –d filter ls fev eth0**

Pour pfifo_fast, on peut effectuer un traçage sur les files d'attentes :
Watch tc –s qdics dev eth0

ANNEXE : Terminologies des termes spécifiques au traitement de TC

PRIO : PRIO permet de classer les flux selon leur priorité. Cet Ordonnanceur envoie les paquets d'un flux de plus forte priorité avant ceux d'un flux de priorité inférieure. Donc, tant qu'il y a des paquets en attente dans ce flux de forte priorité, on ne tient pas compte des flux suivants.

SFQ : (Stochastic Fair Queuing), sépare un flux en plusieurs files d'attentes et chaque flux possède un poids. Le paquet est choisi au hasard dans une file d'attente du flux avec une probabilité égale au poids de sa file d'attente. Cela permet de définir des priorités relatives, les flux étant équilibrés suivant leur poids respectif.
Ainsi, Si notre lien est vraiment saturé et que l'on veut être sûr qu'aucune session ne va accaparer la bande passante vers l'extérieur, on peut utiliser le *Stochastical Fairness Queueing*.

TBF : Token Bucket Filter permet de ralentir le trafic en sortie du routeur.

Qdisc (*Queueing Discipline* ou *gestionnaire de mise en file d'attente*) :
Algorithme qui gère la file d'attente d'un périphérique, soit pour les données entrantes (*ingress*), soit pour les données sortantes (*egress*).

Classes
Un gestionnaire de mise en file d'attente peut être basé sur des classes et en posséder un grand nombre. De plus, on peut ajouter d'autres classes à une classe. Ceci implique qu'une classe peut avoir comme parent soit le gestionnaire, soit une classe.
Une classe terminale : classe ne possédant pas de classe enfant \ un seul gestionnaire peut lui être attaché.
Le gestionnaire de mise en file d'attente par défaut est le gestionnaire fifo. Si on ajoute une classe enfant, le gestionnaire est supprimé. Il est possible de remplacer ce gestionnaire fifo par un gestionnaire de mise en file d'attente basé sur des classes de sorte à pouvoir rajouter des classes supplémentaires.

Classificateur (*classifier*) :
Pour un gestionnaire de mise en file d'attente basé sur des classes, il permet de déterminer vers quelles classes on doit envoyer un paquet.

Filtre (*filter*) :
Il permet de classifier et est composé d'un certains nombre de conditions pour réaliser le filtrage.

Ordonnancement (*scheduling*)
C'est l'action de décider que des paquets doivent sortir plus tôt que d'autres.
exemple : mécanisme implémenté par pfifo_fast.

Mise en forme (*Shaping*)
Processus consistant à retarder l'émission des paquets sortants pour avoir un trafic conforme à un débit maximum configuré. La mise en forme est réalisée sur *egress*.
On emploi aussi le terme pour dire que l'on rejette des paquets afin de ralentir le trafic.

Réglementation (*Policing*)
C'est le fait de retarder ou jeter des paquets dans le but d'avoir un trafic restant en dessous d'une bande passante configurée. Dans Linux, la réglementation ne peut que jeter un paquet, et non le retarder dans la mesure où il n'y a pas de « file d'attente d'entrée » (*ingress queue*).

Bibliographie :

Sources Internet

- http://www.linux-france.org/prj/inetdoc/guides/Advanced-routing-Howto/lartc.qdisc.html
- « HOWTO du routage avancé et ducontrôle de trafic sous Linux »,
  - Bert Hubert & Laurent Foucher
- « HTB Linux queuing discipline manual - user guide »,
  - Martin Devera aka devik
- « Introduction à la qualité de service sous Linux »,
  - Éric LEBLOND
- « Architectures de réseaux et systèmes associés »,
  - Olivier Aumage, Pierre-André Wacrenier, Laboratoire Bordelais de Recherche en Informatique
- "tc - traffic control, Linux QoS control tool" 2nd December 2001,
  - Milan P. Stanic
- « *sch_prio* », 2001-09-12, sch_prio.htm
  - Emmanuel Lochin

12. Documentation sur l'utilisation d'iptables

NOTE – UTILISATION DE NETFILTER / IPTABLES

Bref résumé des fonctionnalités offertes par le noyau linux pour l'identification de flux via l'intermédiaire du logiciel ipTables et de NetFilter.

Recherche et synthèse de ce rapport par Nicolas Van Wambeke pour équipe Réseau :
Fok Frederic, Van Wambeke Nicolas, Molinier Nicolas

Introduction Générale

Ce document a pour but de présenter de façon simple, les possibilités offertes par les différentes composantes du noyau Linux (composantes réseau) ainsi que les différents utilitaires de l'espace utilisateur dans le cadre de la réalisation d'un système automatisé de réservation de ressources pour une application de visioconférence interactive (Projet Tutoré 2004-2005).

Nous verrons dans un premier temps, les fonctionnalités offertes netfilter et iptables en termes de marquage/filtrage/modification de paquets TCP/IP, UDP/IP ou IP. Ensuite, nous évaluerons les possibilités offertes par l'utilitaire de l'espace utilisateur TC (Traffic Control) en termes de contrôle de flux. Notamment tout ce qui a trait aux mécanismes de mise en forme du trafic (Shaping).

NetFilter & iptables

A. Introduction

NetFilter et iptables sont respectivement la partie du noyau Linux (pour les versions 2.4.x et 2.6.x) et l'utilitaire de l'espace utilisateur permettant d'effectuer du filtrage de paquets, de la translation de ports et adresses ainsi qu'une quantité de modifications aux paquets reçus, traversant ou émis depuis l'hôte linux.

iptables est le successeur respectivement de ipchains et de ipfwadm des version 2.2.x et 2.0.x du noyau linux.

B. Marquage différentiel

Il est possible grâce aux deux outils précédents de marquer les paquets de plusieurs façons. Effectivement, dans le domaine DiffServ, les flux doivent être 'taggés' en fonction de la qualité de service qui leur est associée, afin d'être reconnaissables par les différents routeurs (modélisés dans notre approche par un seul routeur émulateur DiffServ).

Marquage par numéro de port source

Une première approche de marquage possible consiste à effectuer une différentiation des flux vis-à-vis du numéro de port source. Ceci peut être fait grâce à la fonctionnalité de Translation de ports source de iptables, en effet, les flux étant tous en udp, ils sont de nature unidirectionnelle et la modification du numéro de port source des paquets émis par l'une des deux entités participant à l'échange du flux ne dégrade en rien la faculté de

réponse de l'autre utilisateur qui ne se fiera normalement pas aux informations contenues dans le paquet IP afin de répondre à l'émetteur.

Cette approche est intéressante car elle permet de s'affranchir d'un éventuel proxy UDP qui aurait la même fonction mais qui demanderait que les paquets soient émis à destination de l'adresse IP de la machine sur laquelle le proxy UDP s'exécute sur son port d'écoute. L'approche iptables permet de traiter les paquets, même si ceux-ci ne sont pas adressés à l'hôte qui effectue le traitement, il suffit simplement que cet hôte se trouve sur le chemin des données. Le routeur devient alors un **Proxy Transparent**.

Syntaxe :

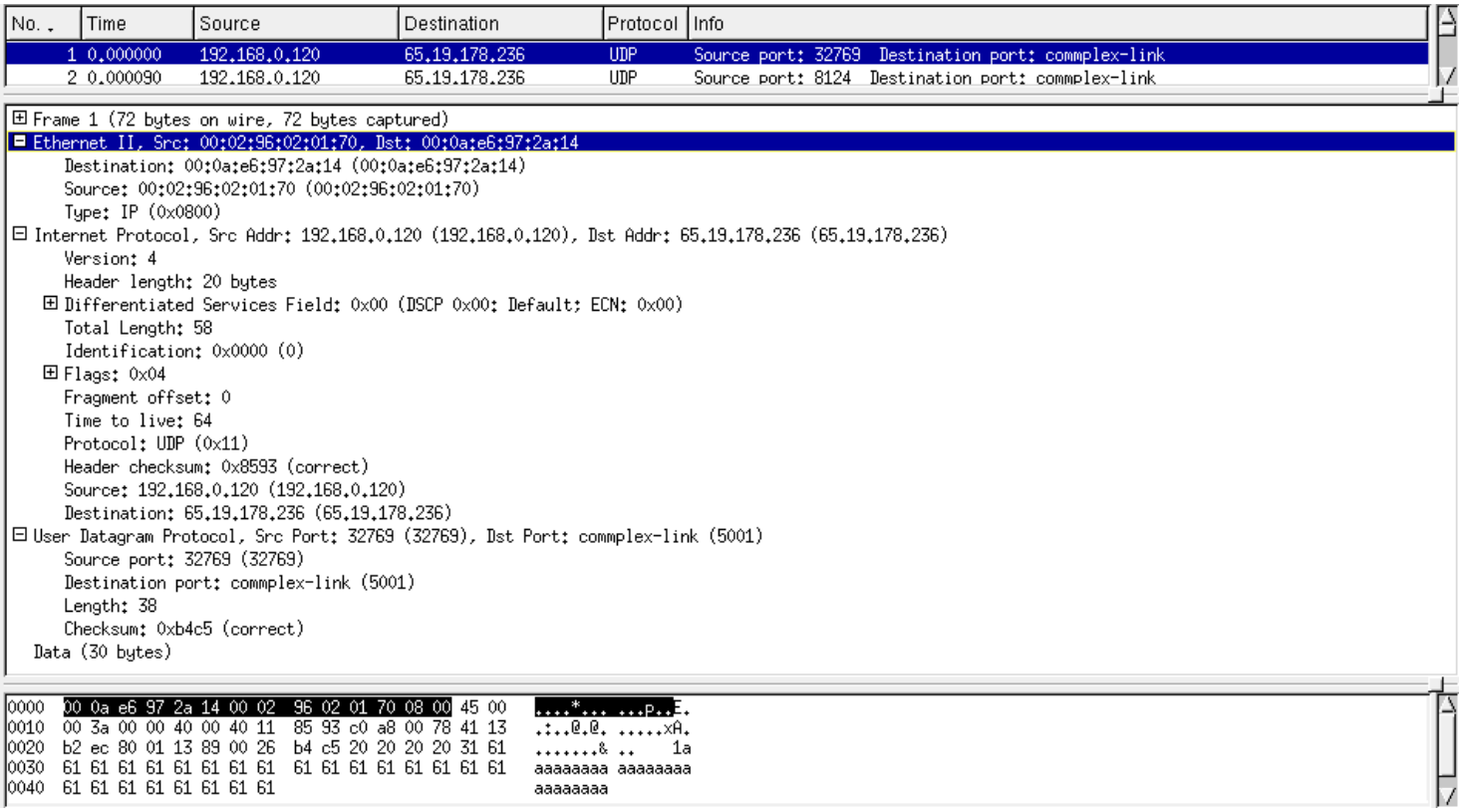
```
La commande suivante de IPTABLES modifie le port source <portSrc> d'un
paquet UDP provenant de l'@IPa qui a pour port de destination le
<portA> sur une machine quelconque.

iptables -t nat -A POSTROUTING -p udp -s <@IPa> --dport <portA> -j SNAT
            --to-source <@IPa>:<portSrc>
```

Explication :

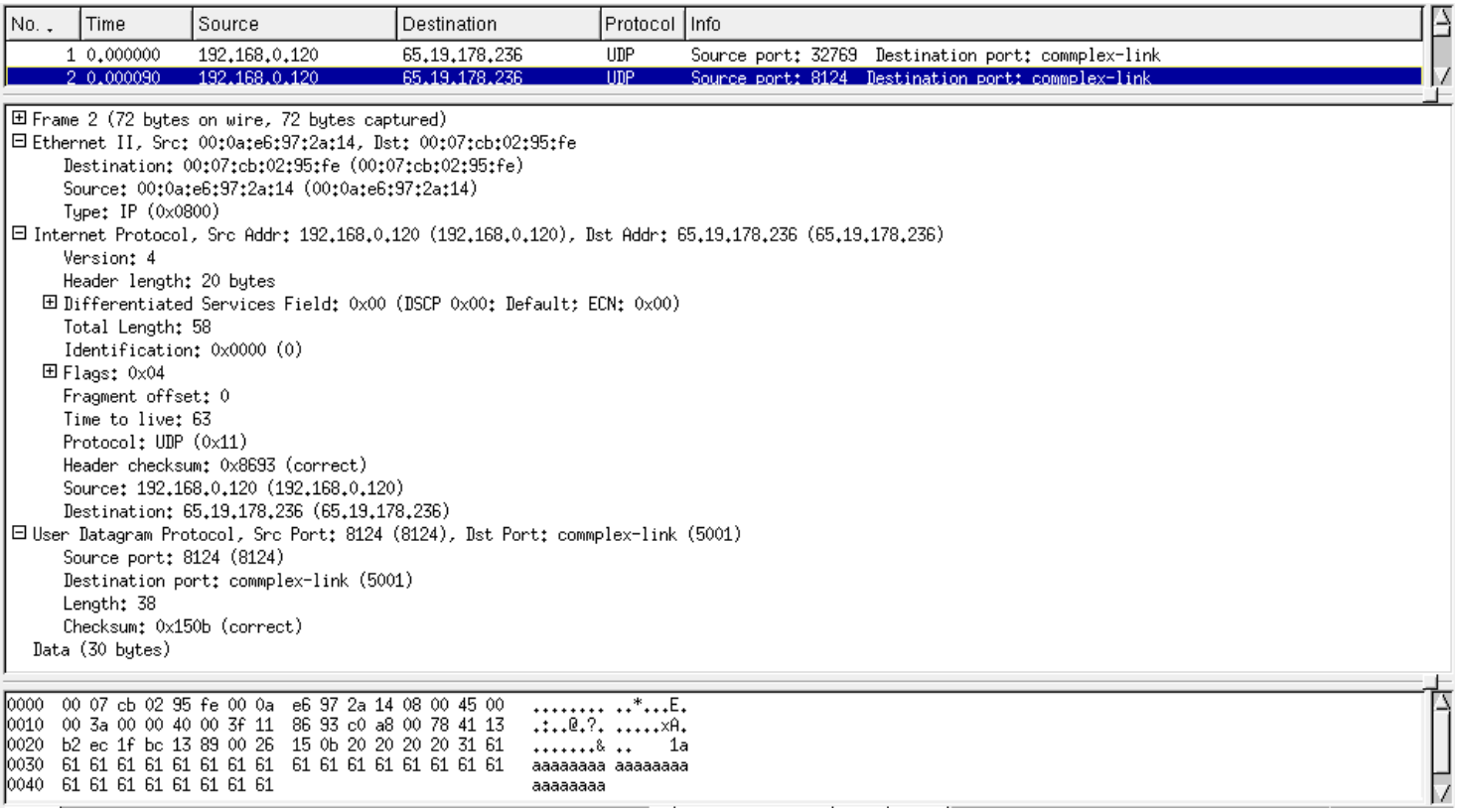
- La table dans laquelle on ajoute une règle est la table de NAT, cette règle doit être appliquée avant que l'on ne traite le paquet par l'algorithme de routage (dans notre cas, cela n'a pas d'importance).
- On spécifie que cette règle ne s'applique qu'aux paquets transportant de l'UDP et donc l'adresse IP source est <@IPa> et le port de destination est le <portA>.
- La décision à prendre sur l'occurrence d'un tel paquet est un NAT de Source vers <@IPa> avec modification du numéro de port vers <portSrc> correspondant à la catégorie de trafic à laquelle appartient le flux.
- La commande iptables permet de spécifier plus d'options pour caractériser un flux, on y retrouve notamment l'adresse IP de destination, le port source parmi d'autres options valides. Pour faire simple, nous avons proposé une solution sans trop de détails.

Vérification :

Afin de vérifier le bon fonctionnement de la technique, nous avons procédé à des captures de trafic sur un routeur aménagé sur lequel s'exécute le noyau netfilter et sur lequel iptables a été configuré de façon à faire une modification du numéro de port source, les résultats sont représentés sur la figure suivante :



Capture du premier paquet, arrivant au routeur depuis une station de travail (@src 192.168.0.120), On observe que le numéro de port source est 32769



Capture du second paquet, quittant le routeur vers l'Internet. On observe que le numéro de port source est 8124, il y a donc bien eu modification comme prévu.

Cette approche, bien qu'intéressante, ne nous a pas satisfait de par le fait qu'il en existe un autre, plus simple et qui touche directement aux fonctionnalités prévues par le protocole IP, il s'agit du champ ToS (Type of Service) qui a pour but principal d'effectuer la distinction entre différentes classes de service.

Marquage par le champ ToS

Une deuxième approche de l'identification des paquets appartenant à un flux de données consiste à effectuer un marquage distinctif par l'intermédiaire de la modification du champ ToS (type of service) contenu dans l'entête du paquet IP.

La table de règles nommée Mangle (en anglais littéralement « essoreuse » à voir comme un synonyme de « bidouille ») permet d'effectuer une série intéressante de modifications de l'entête des paquets IP qui traversent notamment les champs TOS/DSCP/ECN.

Pour notre approche de la qualité de service, nous utiliserons le marquage du champ ToS. Il est possible avec iptables, pour un flux identifié par au moins l'un des quatre paramètres suivants :

- Un protocole,
- l'@IP source,
- l'@IP de destination,
- le port source
- le port de destination

de spécifier une valeur à attribuer au champ ToS de tous les paquets lui appartenant. Ceci s'effectue avec néanmoins une contrainte sur la valeur que peut prendre le champ ToS. Cette valeur doit être comprise parmi celles définies dans le fichier ip.h du répertoire include/linux dans l'arbre des sources du noyau linux compilé sur la machine. Pour la version 2.6.x du noyau, ces valeurs sont :

```
#define IPTOS_LOWDELAY          0x10
#define IPTOS_THROUGHPUT        0x08
#define IPTOS_RELIABILITY       0x04
#define IPTOS_MINCOST           0x02
```

Syntaxe :

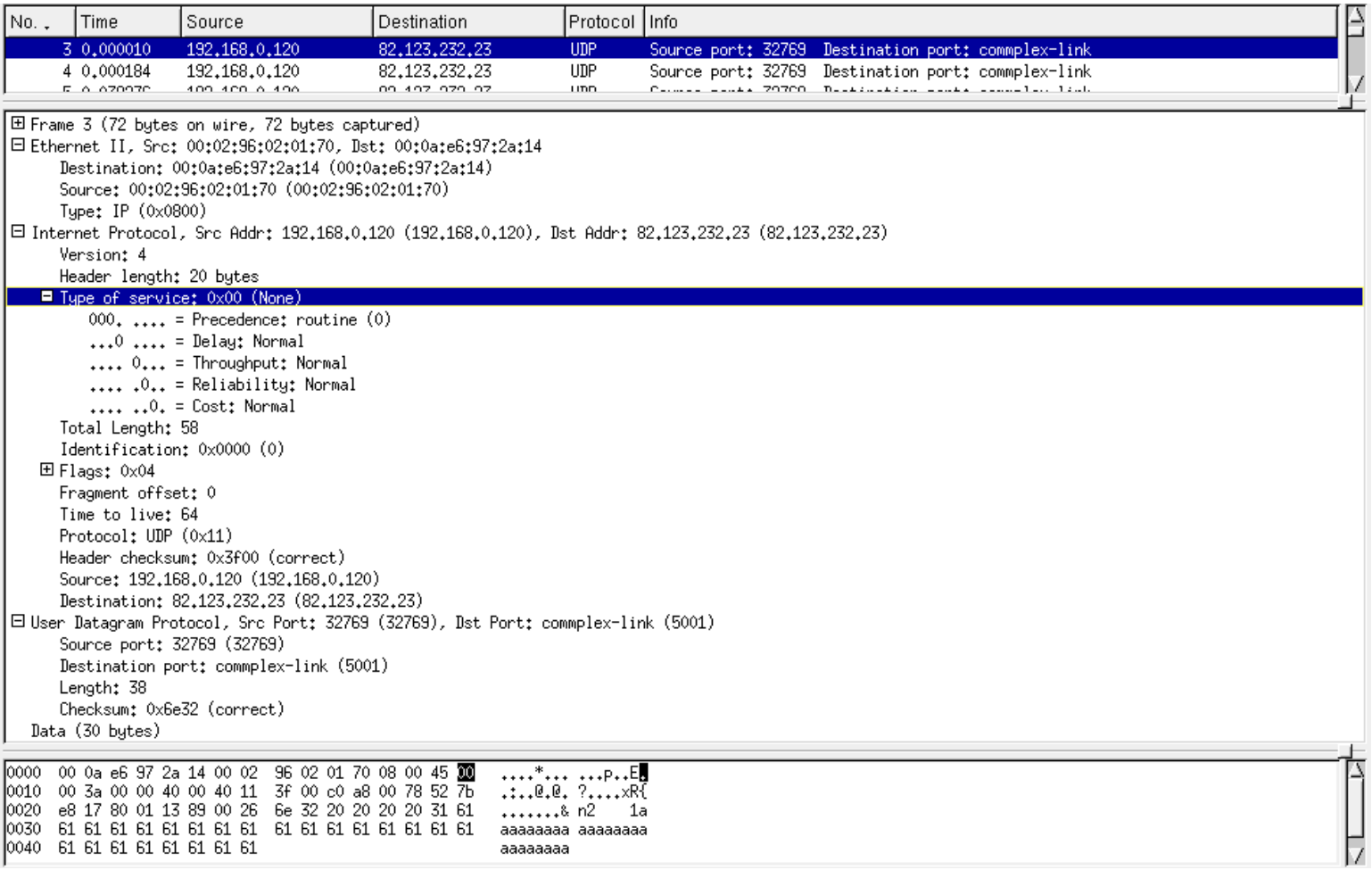
```
Il est possible de modifier le champ ToS des paquets IP passant par le
routeur à la volée. Ceci est réalisé par l'utilisation de la commande:

iptables -t mangle -A POSTROUTING -p udp -s <@IPa> --dport <portA>
           -j TOS --set-tos <VALEUR>
```

Explications :

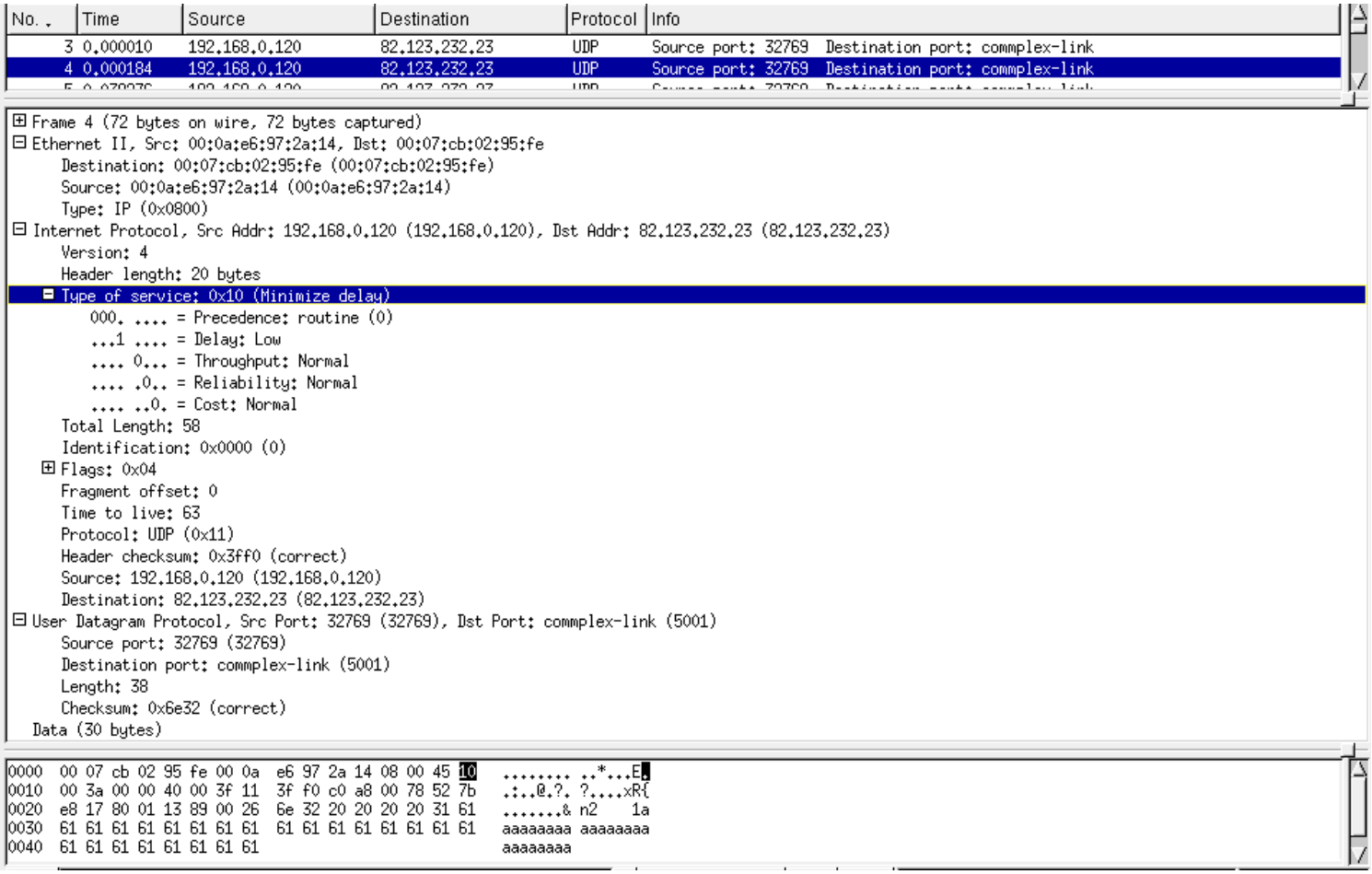
- On travaille ici sur la table mangle, en POSTROUTING (après routage), on pourrait également travailler en PREROUTING (avant routage) pour notre utilisation puisque l'on travaille sur le champ TOS.
- Le flux est identifié par son protocole (udp), l'adresse IP de l'émetteur du flux ainsi que le port de destination.
- La décision est ici une altération du champ ToS qui prendra la valeur spécifiée.

Vérification :

Afin de vérifier le bon fonctionnement de la technique, nous avons procédé à des captures de trafic sur un routeur aménagé sur lequel s'exécute le noyau netfilter et sur lequel iptables a été configuré de façon à faire une modification du champ ToS pour le position à 0x10 (Minimize-Delay), les résultats sont représentés sur la figure suivante :



Capture du premier paquet, arrivant au routeur depuis une station de travail (@src 192.168.0.120). On observe que le champ ToS vaut 0x00



Capture du second paquet, quittant le routeur vers l'Internet.
On observe que le champ ToS vaut 0x10, comme attendu.

Bibliographie :

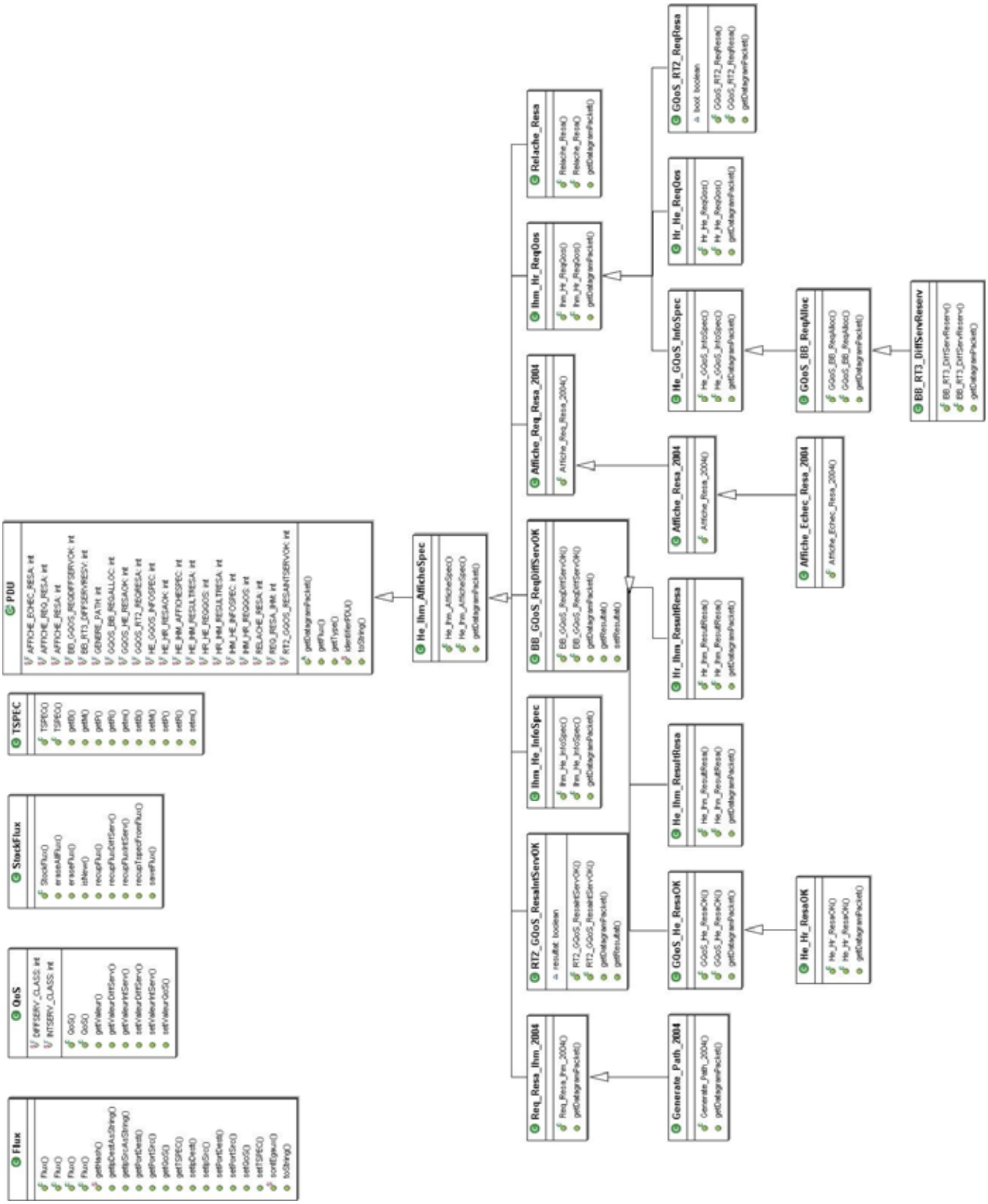
- Linux Advanced routing and Traffic Control HOWTO
(http://www.linux-france.org/prj/inetdoc/guides/lartc/ version en français)
(http://lartc.org/howto/ version en anglais)

- Linux NetFilter hacking HOWTO
(http://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.html)

- IpRoute2
(http://developer.osdl.org/dev/iproute2/)

- NetFilter
(http://www.netfilter.org/)

13. Librairie des PDUs



14. Note de conception de l'IHM

A. Introduction

L'IHM (Interface Homme Machine) est la couche de plus haut niveau de l'application de réservation de ressources. Elle est constituée de deux entité, une entité émettrice (qui sera placée sur He) et une entité réceptrice (sur Hr).
C'est l'entité Réceptrice qui sera à l'origine de toutes les réservations. Elle est en contact direct avec l'utilisateur. Pour effectuer une réservation, il ne manipulera que l'IHM. Il est donc nécessaire de souligner l'aspect graphique de cette application.

B. Objectifs

Coté Hr :

L'IHM est chargée de récupérer les informations sur le flux fournies par l'utilisateur, de les formater dans un langage compréhensible par la machine et de fournir ses informations à la couche RESA sous-jacente.
Elle est aussi chargée d'informer l'utilisateur du résultat de la demande de réservation.
Enfin, c'est elle qui est à l'origine de la relâche de réservation sur abandon de l'utilisateur.

Coté He :

L'IHM est chargée de demander des informations sur la qualité de réservation qui va être effectuée, et d'informer l'utilisateur de la réussite de la réservation.

C. Existant

Pour dialoguer avec les couches sous-jacentes, l'application manipulera la bibliothèque de PDU programmée préalablement. Ce n'est que sur réception de ces PDUs que l'application réagira, et elle n'émettra que des PDUs de la librairie. C'est ce qui fait la force d'ailleurs de l'intégralité du projet, c'est que l'intégralité du protocole de communication repose sur ces PDUs, il est aisé d'en rajouter, et ont tous une structure telle qu'il est facile des les reconnaître parmi les autres paquets.

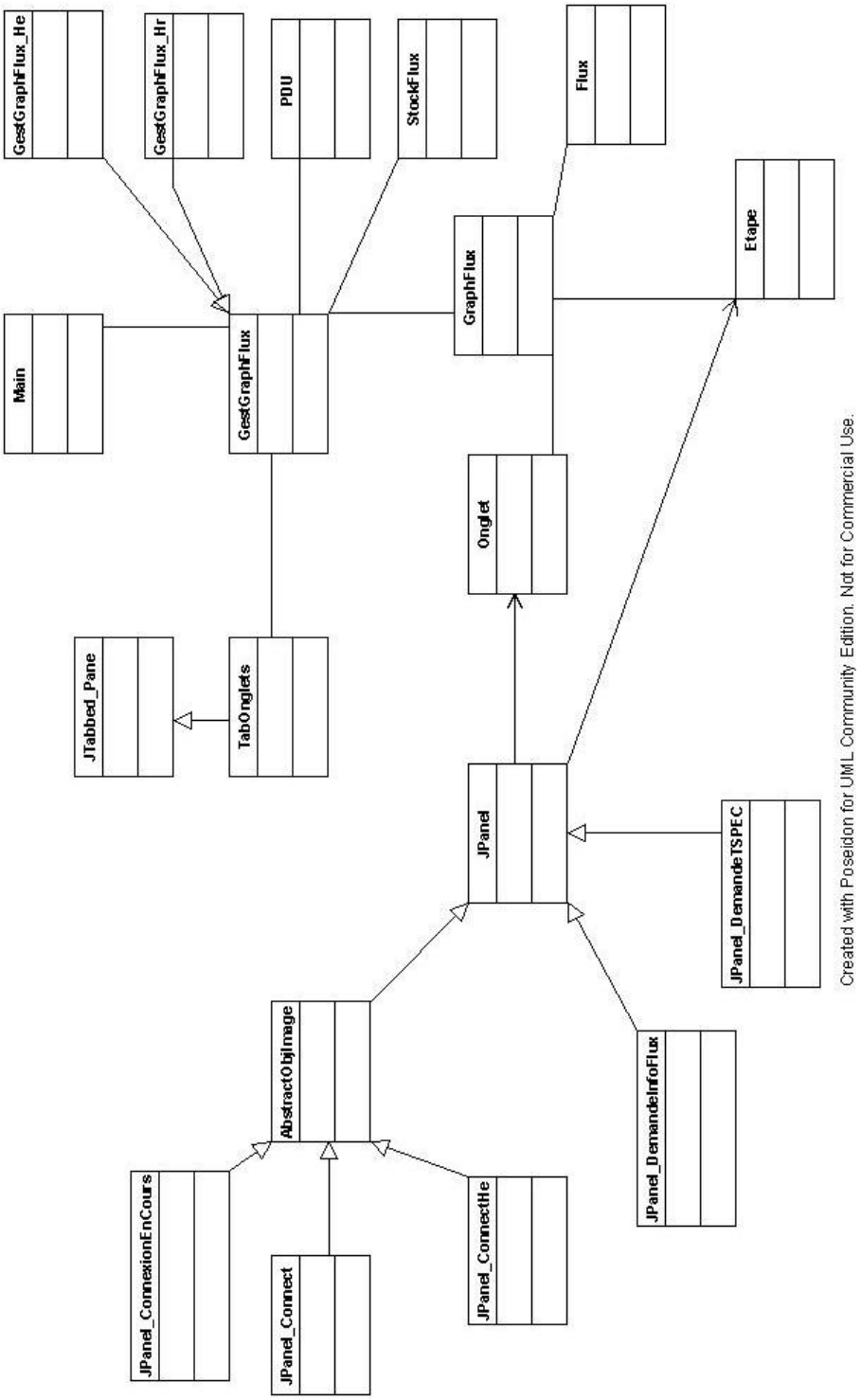
D. Cahier des charges

L'application doit être interactive, graphique, et facile d'utilisation, même pour un novice.
Elle doit manipuler uniquement la bibliothèque de PDUs existante.

Une seule application doit servir aussi bien à He et à Hr, tout en distinguant bien les deux entités.
Enfin, l'application doit être écrite en Java, pour pouvoir manipuler facilement aussi bien les outils graphiques associés que le bibliothèque PDU Programmée en Java.

E. Choix de conceptions

Classes choisies



Le schéma ci-dessus, est très simple, il ne s'agit pas d'un schéma UML conventionnel, il est juste là pour donner un aperçu de la structure de l'IHM, et je vais fortement m'y appuyer pour présenter le programme :
L'objectif est de faire un tableau d'onglets, ou chaque Onglet correspondrait à un flux, affichant l'avancement de la réservation d'un flux.
Prenons maintenant chaque classe et expliquons succinctement son rôle :

La classe Main :

Elle contient le main() principal et unique de l'appli, elle lance le Gestionnaire de flux graphiques (GestGraphFlux), He ou Hr, en fonction du choix de l'utilisateur de manipuler un émetteur ou un récepteur, nous allons détailler par la suite son rôle. Elle lance la fenêtre principale, dans laquelle se trouve le un TabOnglet.

La classe GestGraphFlux :

Il s'agit d'une classe abstraite. GestGraphFlux_He et GestGraphFlux_Hr héritent toutes deux de cette classe. Son Objectif est de gérer l'intégralité des GraphFlux. C'est elle qui est chargée de l'émission et la réception de tous les PDUs. Sur la réception d'un PDU, la classe doit retrouver le GraphFlux correspondant, pour pouvoir le faire évoluer. Pour se faire elle compare le flux arrivé par le PDU à ceux contenu dans chaque GraphFlux.

La classe GraphFlux :

Il s'agit de la classe gérant un onglet, le détruire lorsqu'il na plus lieu d'être ou le construire en cas de besoin, à cette classe est associé un Flux, qui est une classe programmé dans la librairie de PDUs qui permet de stocker les informations concernant un Flux (Adresses IP, Nos de port, QoS, TSPEC). Enfin elle contrôle l'évolution de chaque Onglet, par la classe Etape.

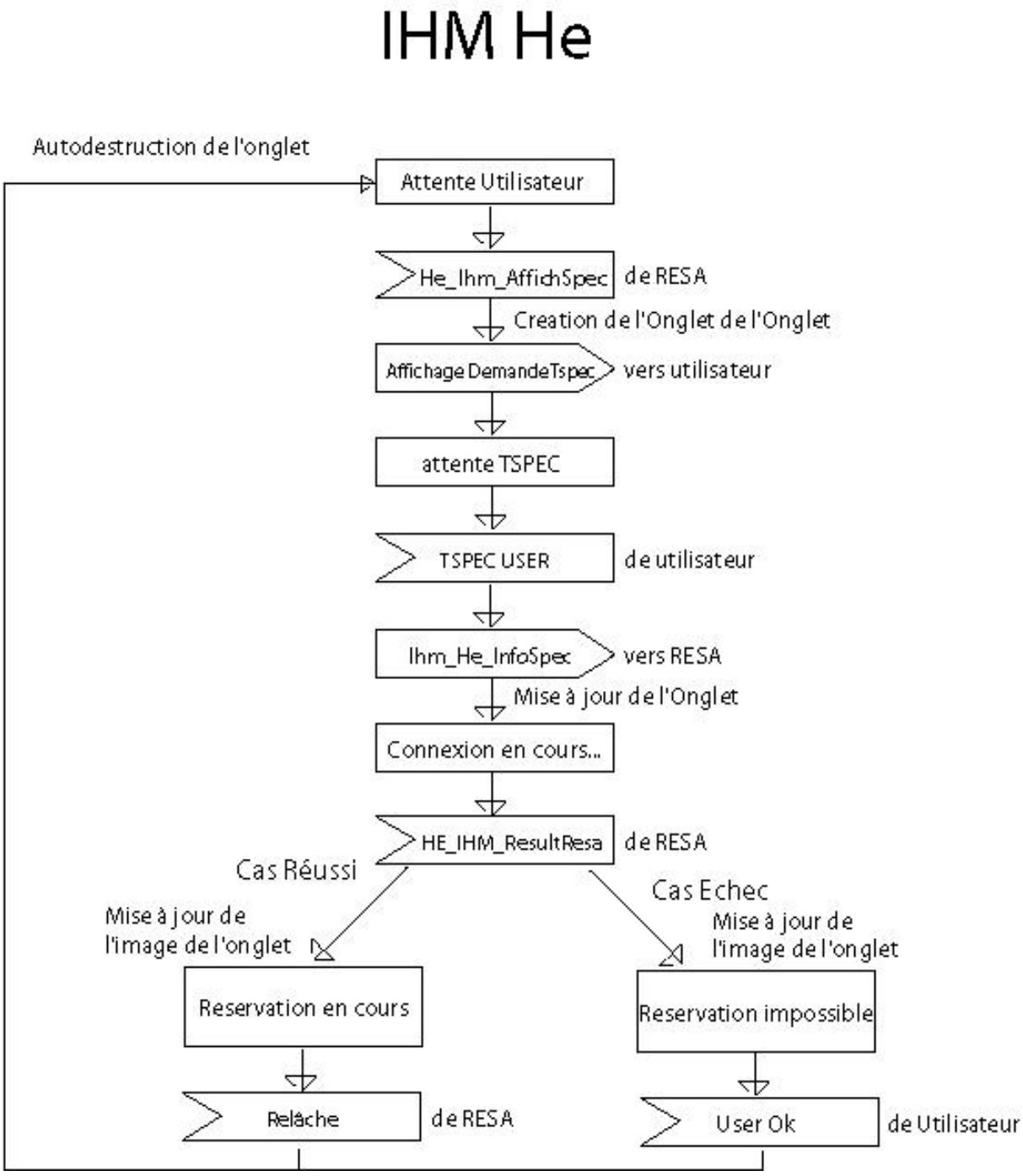
La classe Etape :

Cette classe gère le contenu d'un JPanel contenu dans un Onglet. Lors de la demande de changement d'une étape, en fonction de l'étape précédente, il affiche l'état d'un Flux. Par exemple, si on lance changerEtape(Vrai) alors que l'on était à l'étape connexion en cours, on va tomber sur l'étape connexion réussie.

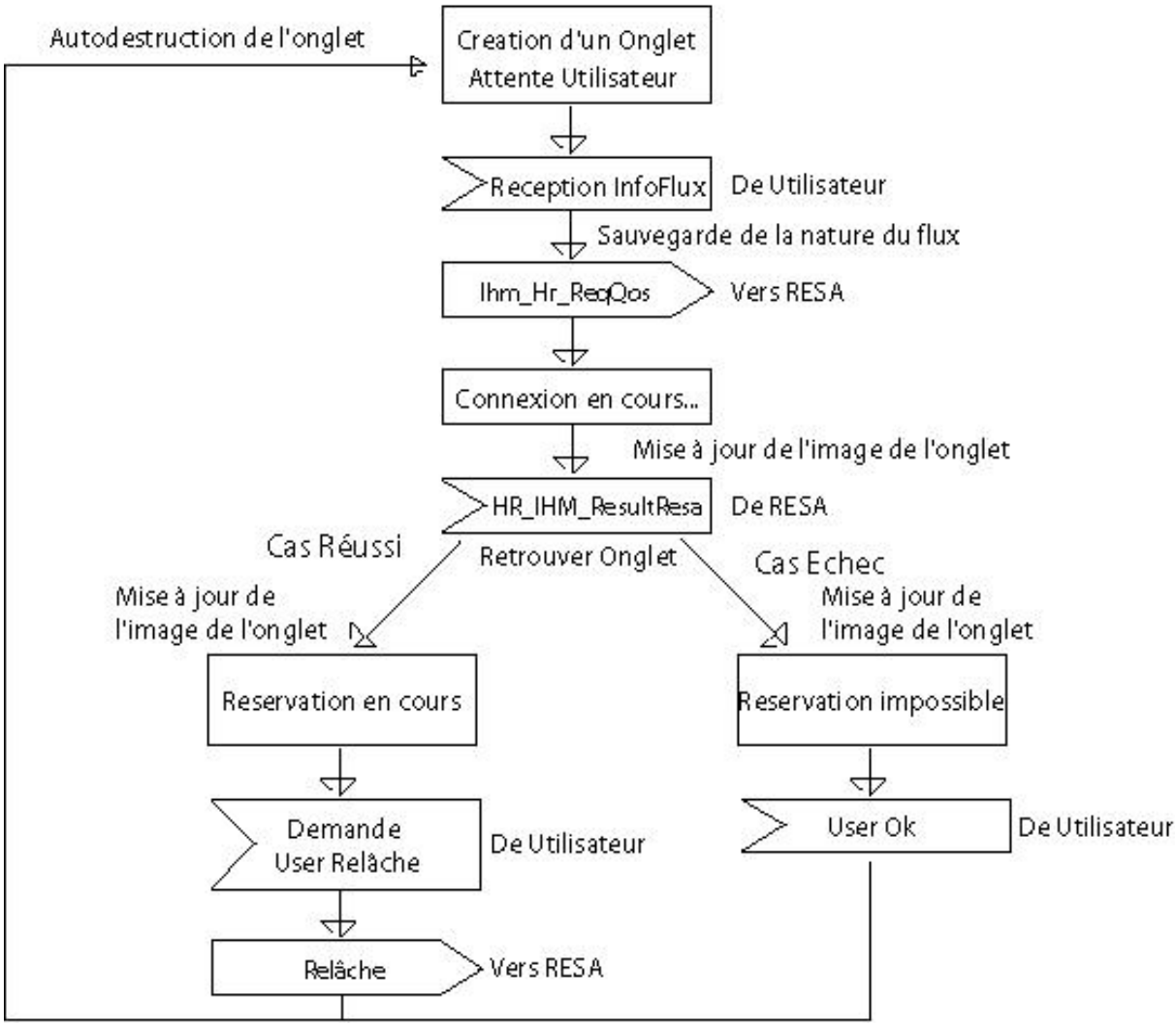
La classe AbstractObjImage :

Utilitaire permettant de créer un JPanel contenant une image seulement et actif sur un clic de souris.

Présentations des étapes :



IHM Hr


Autodestruction de l'onglet
Creation d'un Onglet
Attente Utilisateur
Reception InfoFlux
De Utilisateur
Sauvegarde de la nature du flux
Ihm_Hr_ReqQos
Vers RESA
Connexion en cours...
Mise à jour de l'image de l'onglet
HR_IHM_ResultResa
De RESA
Cas Réussi
Retrouver Onglet
Cas Echec
Mise à jour de
l'image de l'onglet
Mise à jour de
l'image de l'onglet
Reservation en cours
Reservation impossible
Demande
User Relâche
De Utilisateur
User Ok
De Utilisateur
Relâche
Vers RESA

15. Comptes Redus de Réunion

COMPTE RENDU DE REUNION DU 26 JANVIER 2005

Participants :

- M. Chassot
- Fok Frédéric
- Molinié Nicolas
- Van Wambeke Nicolas

Ordre du jour:

- iptables/netfilter pour RT3
- policing
- relachement de réservation
- traitement des erreurs

iptables/netfilter pour RT3

suite à la discussion, nous avons dégagé que l'utilisation de netfilter serait optimale pour le marquage du champ ToS des paquets composant les flux pour lesquelles on effectue une réservation, il faut néanmoins vérifier que ce marquage est possible.

Le programme d'écoute des messages de signalisation qui est à coder devra donc exécuter la commande de configuration du noyau netfilter à savoir **iptables** (comme l'année passé l'IHM Java qui exécutant le programme de résa codé en C)

L'unification des fonctions RT2 et RT3 devient alors moins naturelle car RT2 est toujours un proxy applicatif et RT3 se voit attribué un rôle de marqueur des paquets. La machine destinée au routeur émulateur servira donc de routeur RT3 et effectuera un ordonnancement différentiel des paquets sur son interface de sortie dans le domaine DiffServ.

En ce qui concerne RT2, il est codé et fonctionnel.

policing

pour ce qui est du policing, c'est-à-dire, la vérification que les flux ne dépassent pas les caractéristiques qui ont été réservées, il faudra creuser l'approche utilisant TC (Traffic Control), utilitaire linux permettant une configuration des interfaces de sortie pour effectuer un ordonnancement différentiel. Eventuellement, cet ordonnancement devra se faire en tenant compte des classes de trafic selon le modèle WFQ.

relâchement de réservation

la procédure de relâchement de réservation a été abordée et deux approches ont été dégagées. Soit la réservation est maintenue par l'envoi de paquets de signalisation à intervalles réguliers afin d'effectuer un rafraîchissement périodique de

la réservation auquel cas, l'arrêt de cet envoi systématique entraîne le relâchement de la réservation.

Une autre approche consiste à conserver la réservation jusqu'à la réception d'un message de relâchement, il faudra décider d'un modèle à implémenter pour prendre en compte le relâchement dans le domaine DiffServ.

traitement des erreurs

afin de palier aux éventuelles pertes de paquets de signalisation, il a été évoqué la possibilité d'utiliser le protocole TCP pour véhiculer les paquets de contrôle. Dans cette optique, les problèmes complexes liés à la perte de paquets dans les mécanismes évoqués précédemment lors du relâchement de la réservation ne sont plus d'actualité.

Il faut également mettre en place le protocole à suivre en cas de refus de réservation par l'un des domaines concernés et définir la cascade de messages afin de mettre au courrant les deux intervenant du flux. *(diagrammes d'états uml, …)*

vrac et conclusion

d'autres thèmes abordés pendant la réunion tenaient à la numérotation des messages sur les documents précédents, il doit être corrigé dans l'optique de respecter la règle : « Toute requête demande acquittement par l'hôte récepteur de la requête » (messages 2-5)

De plus, il faudra noter que le PDU de réservation IntServ réussie ne devra pas déclencher l'affichage chez l'utilisateur d'une réussite de réservation (éventuellement, une réussite partielle pourra être signalée mais en aucun cas la totalité) et les affichages sur l'IHM devront être inhibés au niveau de l'hôte RT2.

COMPTE RENDU DE REUNION DU 9 FEVRIER 2005

Participants :

- M. Chassot
- Fok Frédéric
- Molinié Nicolas
- Van Wambeke Nicolas

Ordre du jour:

- utilisation de tc en sortie RT3
- protocole des messages de sig
- langage de programmation
- machines à état

utilisation de tc en sortie de RT3

suite aux recherches effectuées par le groupe sur l'utilisation de tc et de sa configuration dans le cadre du projet, la discussion a porté principalement sur la politique d'ordonnancement à appliquer pour émuler au mieux le comportement dans un système DiffServ tout en n'ayant pas recours au RTémul.

dans un premier temps, l'établissement d'une politique de qdisc en pfifo_fast avec distinction de deux classes de trafic a été envisagée. Une extension qui porterai sur l'utilisation de CBQ a aussi été discutée et doit être approfondie. Des tests sont à réaliser pour vérifier le bon fonctionnement de ces mécanismes.

il a aussi été discuté la séparation à faire entre policing et scheduling, en effet, même si le qdisc CBQ permet de faire les deux opérations en même temps, il est nécessaire d'adresser les problèmes séparément lors de l'analyse.

protocole des messages de sig

une discussion portant sur le protocole de niveau transport à utiliser a abouti sur le besoin d'une étude des contraintes que représenterai l'utilisation d'UDP vis-à-vis d'une implémentation en TCP. Pour des raisons de compatibilité avec le projet de l'année dernière, il serait favorable d'utiliser UDP.

dans le cadre d'une utilisation de ce protocole, il a été défini que l'implémentation pourrait dans un premier temps considérer une communication sans pertes.

un nommage des différents messages de signalisation est à proposer pour validation.

langage de programmation

afin de garder une certaine uniformité avec ce qui avait été fait l'an dernier, l'implémentation sera faite en Java. Cette approche permettra un approfondissement de nos connaissances en ce langage de programmation.

machines à état

une illustration du formalisme de schéma sous forme de machines à états a été expliquée afin de permettre la réalisation, pour chaque fonction du système de réservation, d'une machine à état décrivant son comportement vis-à-vis des différents messages qui lui sont adressés.

conclusion

une modification de la documentation produite précédemment pour prendre en compte les derniers changements apportés à l'implémentation est attendue. Ceci dans le but d'une validation par le tuteur.

des tests d'utilisation de tc sur la plateforme de projet seront réalisés afin de vérifier le bon fonctionnement de celui-ci.

les machines à état des différentes fonctions à implémenter pour le système sont à réaliser en vue d'une validation.