INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

# Rapport BE QoS

**ANSER Yacine**
**BOURZAT Franck**
**DJEBAR Loïc**
**MANDOUJ Lisa-Marie**
**TISSOT Evan**
**4IR-SC**

28/05/2019

# Sommaire

# Introduction

La qualité de service désigne la capacité d'un service à répondre par ses caractéristiques aux différents besoins de ses utilisateurs ou consommateurs.

Afin de mieux appréhender les enjeux de la QoS, ils nous a été demandé de concevoir et de mettre en oeuvre, au travers d'un bureau d'étude, un système de réservation dynamique de ressources en bande passante nécessaires à la mise en oeuvre d'applications interactives de type voix sur IP en présence de trafic de type transfert de fichier. Ce système prendra place sur un domaine distribué sur 3 sites distants interconnectés par un réseau de coeur.

Une phase de préparation a donc été nécessaire notamment pour comprendre le protocole SIP et son implémentation logicielle OpenJSIP, l'outil TC ( Trafic Control ), le choix de la solution implanté au niveau du réseau ainsi qu'à la mise en place du Bandwidth Broker.

Ce rapport est donc la synthèse de tous ces choix et des résultats obtenus.

# I/ Architecture réseau

L'architecture réseau mise en place est celle décrite sur la figure ci-dessous. Chaque site est relié au réseau de coeur du prestataire de service à l'aide d'un routeur tournant sur une machine Linux (représenté en orange sur la figure).
Le réseau de coeur est maillé et c'est un réseau MPLS. Le prestataire de service offre un service VPN de niveau 3 au client. Cela lui permet d'avoir un réseau IP pour chaque site et d'avoir une interconnexion entre chaque site.
Le prestataire de service fait du policing à l'entrée de son réseau en se basant sur les SLA négociés pour chaque site. Une politique de QoS est également à déployer à l'intérieur de son réseau. La classification du trafic se fait sur le champs EXP (que le client devra marqué par lui même). Ainsi le prestataire de service pourra différencier le trafic voix prioritaire et le trafic best effort et allouer de la bande passante en fonction de la classification.

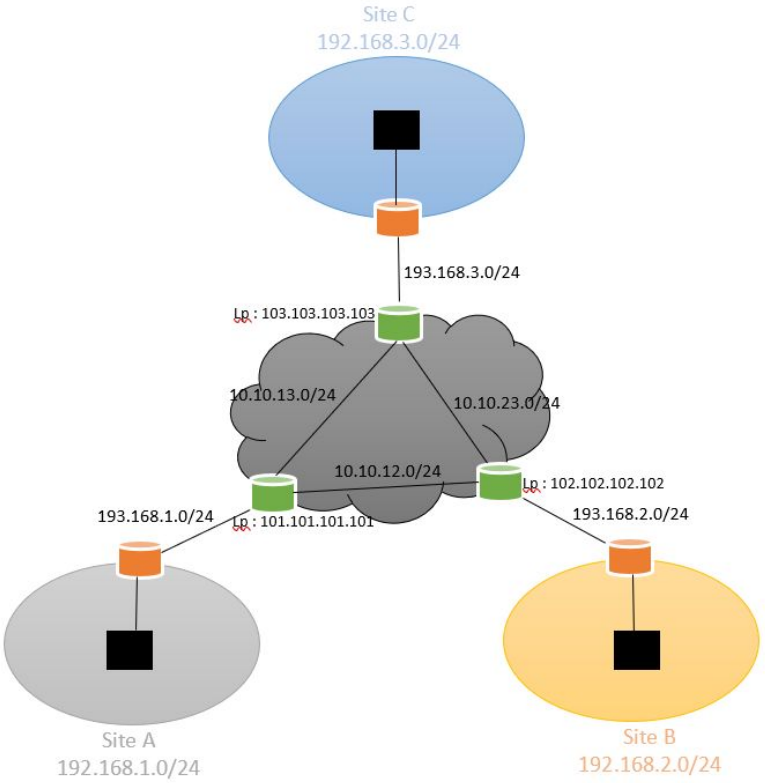


Figure 1 : Architecture du réseau

La mise en place de cette structure a été plus longue que prévu durant le TP, cela est notamment dû à des problèmes imprévus et aléatoires (port sérial de certains routeur qui se déconnecte / reconnecte fréquemment par exemple), perte de temps sur des problèmes facilement contournables. Ainsi nous n'avons pas eu le temps de déployer le policing et la QOS sur le réseau de coeur.

# II/ Conception et implémentation du Bandwidth Broker

La conception ainsi que la mise en oeuvre du Bandwidth Broker étant libre, il fallu faire des choix. Nous avons choisi d'utiliser le langage de programmation Java car nous l'avons beaucoup utilisé ce semestre notamment en programmation réseau. Il a fallu ensuite définir l'architecture du programme.

## Conception

Avant de commencer la programmation, nous avons d'abord choisi de réaliser un diagramme UML de notre architecture Java. Celle-ci a beaucoup changé au cours du projet, voici la version finale :

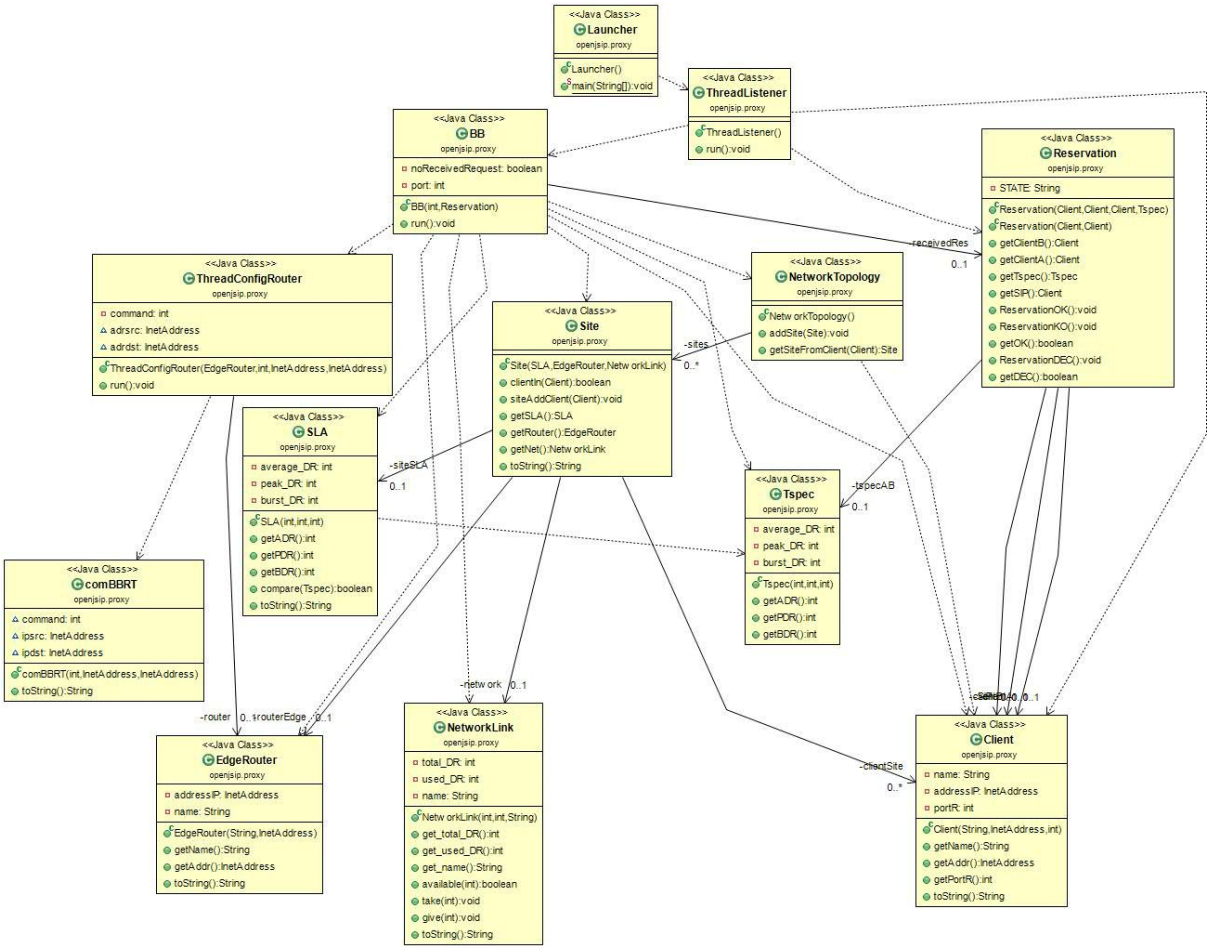


Figure 2 : Diagramme de classe UML du code Java du Bandwidth Broker

Le fonctionnement général du programme est le suivant :

A l'execution du programme, la classe Launcher ( Main ) lance le thread ThreadListener qui va écouter sur le port 1234 toutes les connexions entrantes en provenance du proxy SIP. Lorsqu'une réservation est  reçue, un thread BB est créé avec comme paramètres la réservation reçue ainsi qu'un port de communication avec le proxy SIP.
Le thread va vérifier que toutes les conditions nécessaires à l'établissement de la connexion entre les deux entités sont satisfaites. C'est à dire que les SLAs des deux sites acceptent le Tpsec de la communication.
Si les conditions ne sont pas vérifiées, il renvoie au proxy le message réservation mais avec la variable STATE  = "KO" ( auparavant STATE = "REQ", pour request) puis s'arrête.
Sinon il envoie l'objet Réservation avec le STATE = "OK", puis il effectue les modifications nécessaire dans les variables des classes de réseau pour qu'elles prennent en compte le nouveau trafic créé (exemple, soustraction sur la variable bande passante du débit accepté).
Enfin il crée deux threads ThreadConfigRouteur afin de configurer les deux routeurs de bordures de chaque sites. Une fois cela fait, il se met en attente d'un message de déconnexion pour actualiser les variables des classes réseaux et supprimer les règles des routeurs.

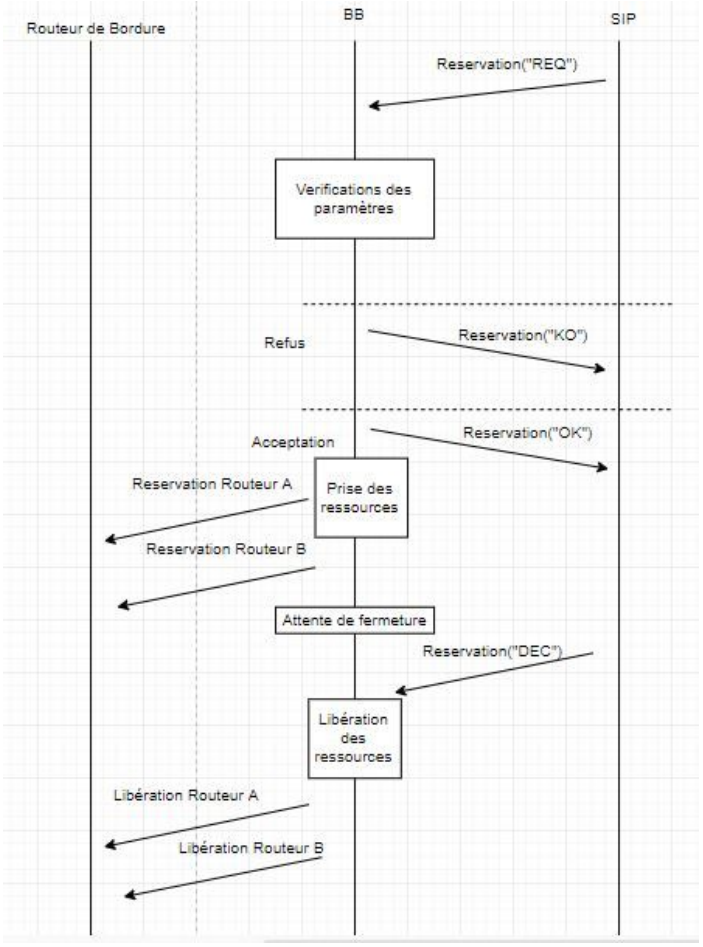


Figure 3 : Diagramme de séquence de la communication entre le Bandwidth Broker et le proxy SIP

# Implémentation

Lorsque nous sommes arrivé en salle de TP, le code était codé à 90 % cependant, comme expliqué précédemment, afin que notre programme Java fonctionne, il nécessite la connaissance de l'architecture du réseau ( que nous ne connaissions pas avant ). Nous avons donc choisi de la coder en dur dans notre programme à l'aide des classes que nous avons créé pour modéliser l'architecture réseau :

```
//CLIENTS
C_S1 = new Client("Client1S1",InetAddress.getByName("192.168.1.1"),2155);
C_S2 = new Client("Client1S2",InetAddress.getByName("192.168.2.1"),2155);
C_S3 = new Client("Client1S3",InetAddress.getByName("192.168.3.1"),2155);


// EDGE ROUTER
rt_S1 = new EdgeRouter("RouterS1",InetAddress.getByName("192.168.1.254"));
rt_S2 = new EdgeRouter("RouterS2",InetAddress.getByName("192.168.2.254"));
rt_S3 = new EdgeRouter("RouterS3",InetAddress.getByName("192.168.3.254"));
```

Figure 4 : Codage de l'architecture du réseau en Java

Les routeurs de bordures et les machines clients de chaque site on été créé. Puis on ajoute chaque entitée à son site correspondant.

```
// CrÃ©ation sites
Site s1 = new Site(sla_s1, rt_S1, nw_s1);
s1.siteAddClient(C_S1);

Site s2 = new Site(sla_s2, rt_S2, nw_s2);
s2.siteAddClient(C_S2);

Site s3 = new Site(sla_s3, rt_S3, nw_s3);
s3.siteAddClient(C_S3);


NetworkTopology network = new NetworkTopology();
network.addSite(s1);
network.addSite(s2);
network.addSite(s3);
```

Figure 5 : Codage de l'architecture du réseau en Java

A partir de cette structure, à la réception de la Réservation, le Bandwidth Broker est capable de déduire à quel site appartient chaque client, et donc d'accéder aux SLAs des deux sites clients pour vérifier le respect des règles. Le reste du code correspond à la partie Conception.

Nous avons perdu beaucoup de temps à débugger le code durant le TP car nous n'avions pas pu tester ensemble l'ensemble des différentes parties ( proxy SIP, Bandwidth Broker, programme Java sur les routeurs de bordures ). Cependant, nous avons tout de même réussi à fournir une démonstration fonctionnelle à nos encadrants.

# III/ Configuration des routeurs de bordure

## Modélisation

Il est nécessaire de configurer les routeurs de bordure pour prioriser le trafic de la voix. Nous avons utilisé l'outil tc qui permet de modifier l'ordonnancement des paquets sortants. Il existe plusieurs algorithmes d'ordonnancement et nous avons utilisé le hierarchical token bucket qui est particulièrement bien adapté à la QOS. Son fonctionnement peut être représenté sous la forme d'un arbre. La représentation du HTB dans notre cas est représenté sur la figure X. Il nous permet de définir un débit pour l'ensemble des communications voix (DR_Voix) et un débit pour tout le reste (DB_BE). Chaque communication est une fille de la classe voix et a un débit associé. Séparer les communications nous permet de s'assurer qu'un utilisateur ne monopolisera pas tout le débit voix.

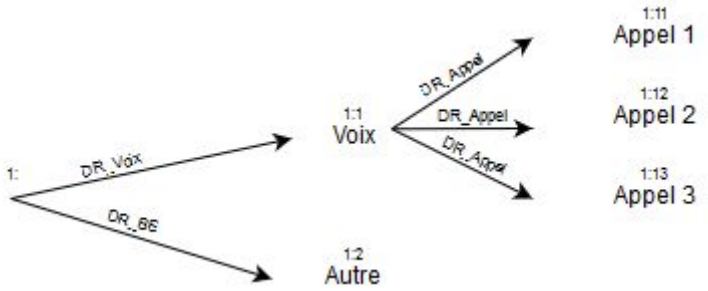


Figure 6 : Représentation de notre modèle du HTB

## Mise en oeuvre

Pour configurer les routeurs de bordure, nous avons décidé de développer un programme en Java qui tourne sur chaque routeur de bordure.

Dès son lancement, ce programme exécute un script bash INIT.
Ce script va créer la discipline et initialiser les classes de bases DR_Voix et DR_BE.
Puis il se met en écoute d'un ordre du bandwidth broker. Lors de la réception d'un ordre de nouvel appel, un script bash CREATE est exécuté. Ce script va créer une classe fille pour l'appel, marquer le trafic et l'assigner à la classe fille précédemment créée, marquer le champs DSCP pour la QOS dans le réseau de coeur de l'opérateur.
Lors de la réception d'un ordre de fin d'appel, un script bash DEL est exécuté. Il supprime les règles mises en place sur le routeur lors de l'établissement de l'appel.

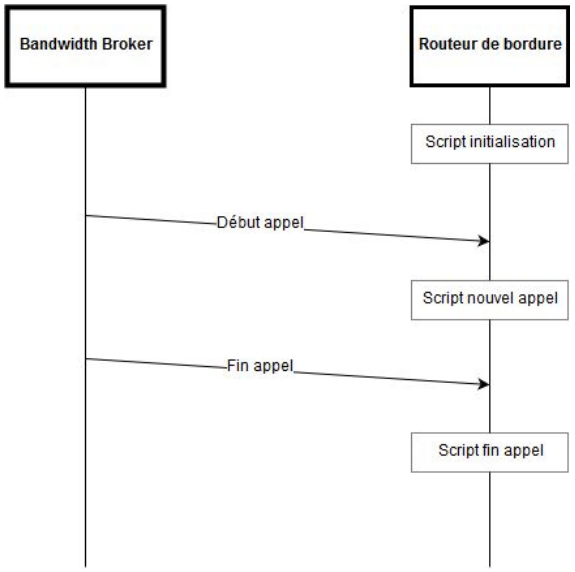


Figure 7 : Diagramme de séquence représentant la communication en le Bandwidth Broker et un routeur de bordure

## Guide des commandes

**#Création de la discipline queuing racine sur l'interface eth0**
**# htb : hierarchy token bucket**

```
tc qdisc add dev eth0 root handle 1: htb default 20
```

**# Création d'une classe**
**# id : 1:1 - avec un débit garanti de 500kbit et possibilité d'emprunter 70kbit aux**
**# classes de même niveau**

```
tc class add dev eth0 parent 1: classid 1:1 htb rate 500kbit ceil 70kbit
```

**# Marquage du trafic pour traitement interne**

```
iptables -A PREROUTING -t mangle -s $1 -d $2 -p tcp --dport 5001 -j MARK
--set-mark $3
```

**# Marquage du trafic DSCP pour QOS**

```
iptables -A PREROUTING -t mangle -s $1 -d $2 -p tcp --dport 5001 -j DSCP
--set-dscp-class EF
```

**# Classification du trafic dans les classes associées en fonction du marquage**
**# handle : marquage par iptables - flowid : numéro de la classe**

```
tc filter add dev eth0 parent 1:1 protocol ip prio 1 handle $3 fw flowid 1:$3
```

# IV/ Compréhension et gestion du Proxy SIP

## Approche théorique

Pour permettre la mise en place d'un contrôle d'admission par flux au sein d'un réseau de site, il nous est nécessaire de récupérer des informations qui vont permettre au Bandwidth broker de gérer les requêtes de QoS formulées par les deux hôtes du réseau et de réserver les ressources nécessaires. Nous avons comme décrit précédemment, un site constitué d'application de VoIP, d'un proxy SIP et d'un Bandwidth broker , ce qui nous intéresse ici c'est de comprendre le fonctionnement du proxy SIP qui sert d'intermédiaire entre les deux hôtes afin de pouvoir récupérer les informations qui nous semblent importantes.

Le processus démarre durant la phase d'initiation de la session de VoIP à l'aide du protocole SIP, conçu pour établir et terminer des sessions multimédia, il se poursuit comme indiqué sur la figure suivante :

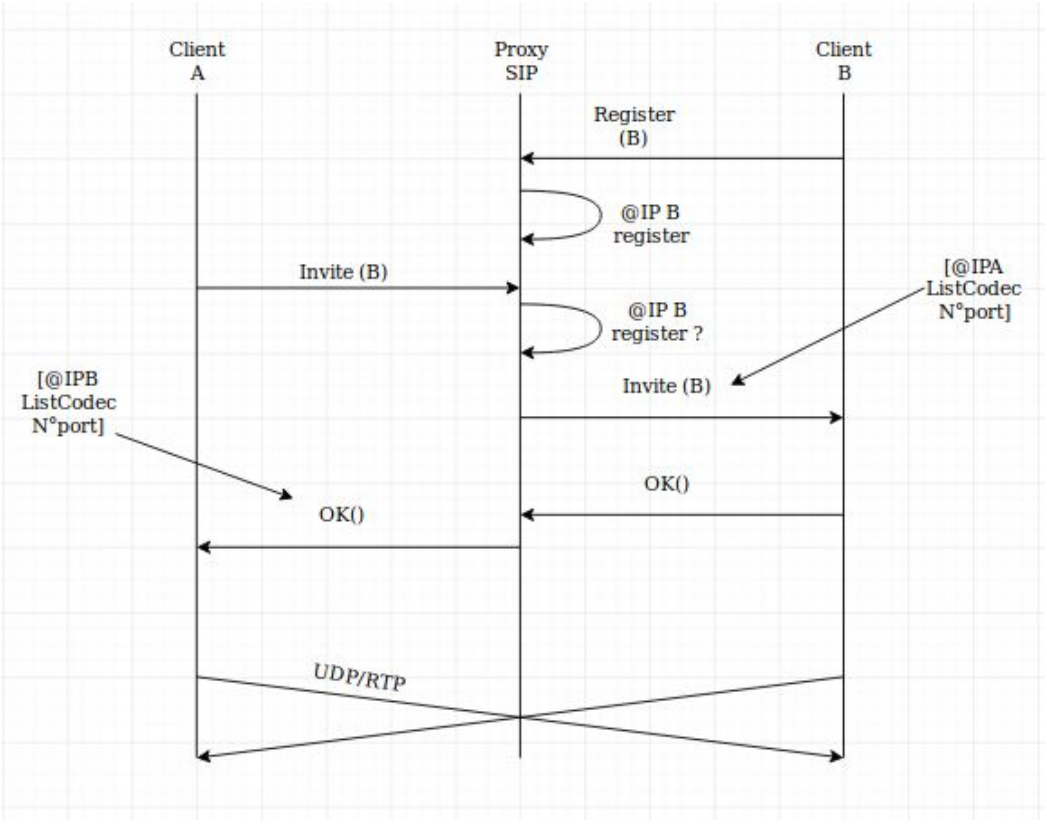


Figure 8 :Diagramme de séquence représentant le protocole SIP

Le client A envoie une invitation au client B, celle ci passe d'abord par le proxy SIP qui va la rediriger vers la bonne personne, Si le client B accepte de communiquer avec le client A, celui ci envoie un PDU OK qui passe aussi par le serveur SIP, de là les deux clients peuvent commencer à échanger des paquet UDP/RTP .

A noter que le protocole SIP est chargé de la négociation sur les types de média (codec) utilisables par les deux hôtes  en encapsulant des messages SDP.

## Implémentation du modèle

De ce qui suit nous considérons les termes suivant : Le client SIP correspond à l'entité appelante et le serveur SIP à l'entité appelée .

Pour la réalisation il nous a été fourni une instance de jitsi qui est un softphone basée sur java et qui permet les appels vocaux basés sur SIP. En ce qui concerne le proxySIP, il nous été imposé l'utilisation de OpenJSIP.

Après un déploiement fastidieux de l'architecture proxySIP (OpenJSIP) plus Softphone (Jitsi), nous avons identifié, via wireshark , les champs des messages SIP nécessaires à la réalisation du contrôle de flux qui sont : le numéros de port , l'adresse IP du client SIP et du serveur SIP et le codec utilisé. En plus de cela il serait intéressant de recupere le CALL-ID de l'appel afin de le sauvegarder et de l'associer avec les informations decrites precedemment. Cela permettra de réaliser plusieurs appels en même temps et de les clore correctement .

Pour nous faciliter le travail on a considéré que le codec indiqué par le client SIP est disponible côté serveur SIP . Cela reste assez proche de la réalité même si le codec est normalement négocié par les deux clients au cours de leurs interactions avec le proxy.

<u>Allocation des ressources :</u>

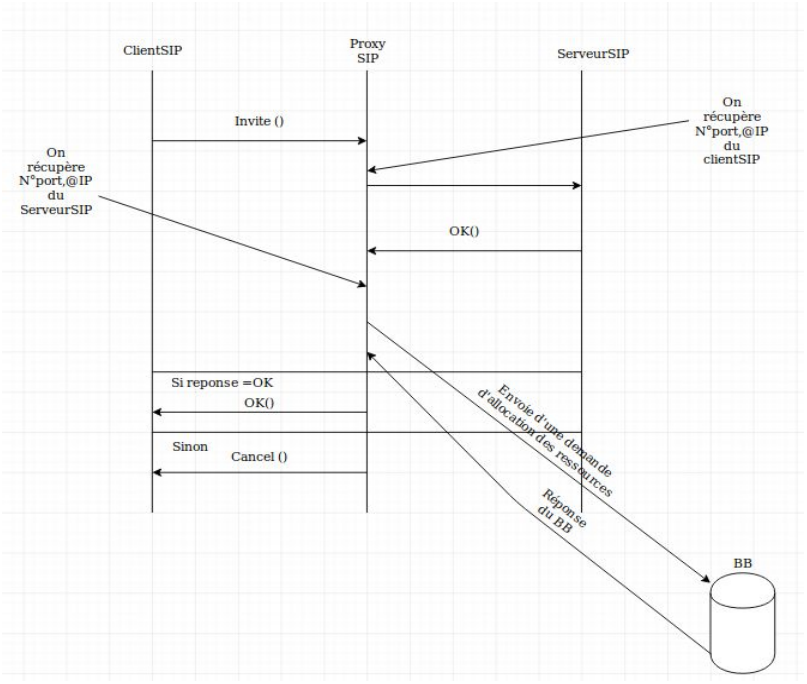


Figure 9: Interaction du BB avec le Proxy SIP

L’allocation des ressources se fait comme indiqué dans la figure 9 .On récupère les données du ClientSIP (N°port , adresse IP,codec ) en interceptant tous les PDU SIP “INVITE” .De la même manière mais cette fois-ci en précisant que le status code est égal à 200,nous récupérons les informations du ServeurSIP présentes dans le PDU SIP “OK” émis par ce dernier dans le cas d’une acceptation de la connexion. Par la suite nous nous créons un objet Réservation que nous l'envoyons au bandwidth broker sur le port 1234 à l’aide de la sérialisation Java , cet objet prend comme paramètre, les informations du ClientSIP ( N°port , adresse IP ) ,du ServeurSIP ( N°port , adresse IP) et le débit nécessaire à la communication (average_DR ,peak_DR, burst_DR ) celui ci est calculé à l’aide du document suivant :

https://www.cisco.com/c/fr_ca/support/docs/voice/voice-quality/7934-bwidth-consume.pdf

Ce document, nous donne une correspondance entre le codec et la bande passante a alloué pour qu’une communication se passe bien .

En plus de cela nous avons créé une hashmap qui prend comme clé le call-ID et comme valeur l’objet réservation, celle-ci va nous permettre d’avoir plus d’un appel en même temps tout en sauvegardant les donnés des appelés précédent . Pour savoir si les ressources ont été bien alloué nous attendant la réponse du BB en écouter sur le port 1235 , celui-ci enverra l’objet Réservation mais avec la variable STATE = “OK” si il accepte l’allocation de ressources sinon il envoie l’objet Réservation avec le STATE = “KO”.

Cette partie a été réalisé avec succès au cours de la démonstration qui c’est déroulé la fin du TP .

Désallocation des ressources :

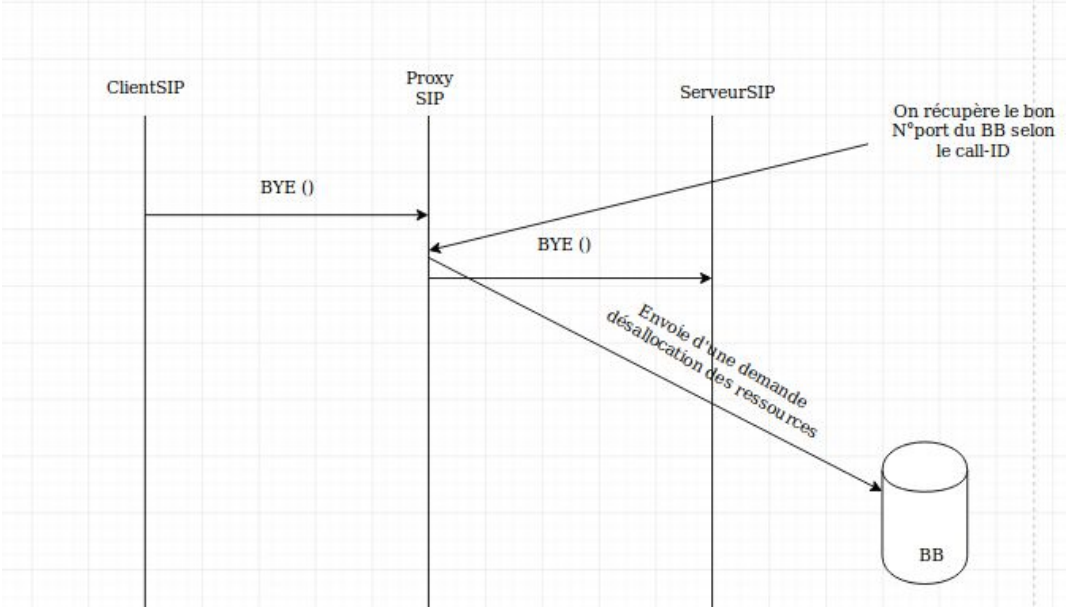


Figure 10: Interaction du BB avec le Proxy SIP

A la fin d'un appel, le ProxySIP se doit de prévenir le bandwith broker qu'il n'est plus nécessaire de garder les ressources alloué, pour ce faire nous nous sommes basé sur le call-id de l'appel que nous avons stocké précédemment , celui-ci va nous permettre d'identifier le N°port pour qui il faut envoyer la demande de désallocation,pour être plus précis , à chaque demande d'allocation un thread BB est créé, celui-ci écoute sur un N°port que le proxy connaît grâce à un mécanisme d'addition séquentielle , ce N°port et ensuite associé au call-id qui lui correspond grâce à une hashmap , de là nous pouvons identifier le bon N°port .

Malheureusement, cette étape n'a pas pu être vérifié au cours du TP par manque de temps .

# Conclusion

Ce  bureau d'étude nous a permis de mobiliser plusieurs connaissances rencontrées au cours de l'année scolaire : QOS, Java, MPLS… L'objectif concret du projet (pouvoir fiabiliser une communication sur un réseau) a fait ressortir l'utilité et la nécessité des concepts cités précédemment.
Nous avons également pu avoir un aperçu de la réalité du métier d'ingénieur et notamment de la difficulté de tenir des délais. En effet, même si nous nous étions préparé à l'avance afin d'être le plus efficace possible durant cette journée, les imprévus, les prises de décisions et la pression liée à l'avancement du temps ont rendu notre tâche beaucoup plus difficile que prévu. C'est pourquoi nous regrettons de ne pas avoir pu réaliser tout ce que nous avions prévu mais gardons néanmoins un très bon souvenir de cette expérience qui nous a notamment préparé pour nos stages.