**Documentation Babel**

Table des matières

[I/ Introduction 3](#_Toc433179208)

[Système d’interconnexion 3](#_Toc433179209)

[Application : 3](#_Toc433179210)

[Présentation : 3](#_Toc433179211)

[Session : 3](#_Toc433179212)

[Transport: 3](#_Toc433179213)

[II/ Les besoins technique 4](#_Toc433179214)

[Sockets 4](#_Toc433179215)

[Librairies multimédia 4](#_Toc433179216)

[Codec de compression 4](#_Toc433179217)

[III/ Ordonnancement des communications 5](#_Toc433179218)

[Connexion 5](#_Toc433179219)

[La négociation de démarrage d’activité 5](#_Toc433179220)

[Pendant l’activité 5](#_Toc433179221)

[Fin de l’activité 5](#_Toc433179222)

[IV/ Le protocole commun client / serveur 6](#_Toc433179223)

[Généralité 6](#_Toc433179224)

[Les codes d’instructions 6](#_Toc433179225)

[Les types de données 6](#_Toc433179226)

[Le header 6](#_Toc433179227)

[V/ Le protocole client-client 7](#_Toc433179228)

[getSound/sendSound 7](#_Toc433179229)

[VI/ Le protocole client-serveur 8](#_Toc433179230)

[connect 8](#_Toc433179231)

[disconnect 8](#_Toc433179232)

[contactList 8](#_Toc433179233)

[addContact 8](#_Toc433179234)

[delContact 8](#_Toc433179235)

[askCall 8](#_Toc433179236)

[rcvCall 9](#_Toc433179237)

[callEstablished 9](#_Toc433179238)

[callFailed 9](#_Toc433179239)

[endCall 9](#_Toc433179240)

# I/ Introduction

Ce protocole vise à décrire la façon dont plusieurs clients Babel communiqueront entre eux. Il décrira les bibliothèques nécessaires, la façon dont elles doivent être mises en œuvre, et comment les clients doivent gérer les connexions, la réception et l’expédition de données. Un client Babel doit mettre en œuvre plusieurs activités de base concernant la VOIP à partir de protocole SIP : Discussion audio, conférence, flux audio, etc. Un client Babel doit être en mesure de commencer chaque activité avec plusieurs autres clients simultanément.

## Système d’interconnexion

### Application :

**Client Babel avec Interface Utilisateur**

Etabli la connexion avec un autre client Babel appartenant à la liste de contact.

**Serveur Babel**

Proxy de liaison des différents clients

Gestion de la liste de contacts

### Présentation :

**Conversion des données**

Sérialisation dans des « paquets » avec mise en place d’un magic, d’un type ainsi que d’une size du paquet. Méthode de sérialisation et desérialisation faite main.

Données agrégée en vecteur de unsigned char.

**Compression des données**

Pour le son il s’agit d’Opus

### Session :

**Ouverture, maintenance et terminaison de session:**

Système de socket.

**Support de session:**

Nom d’utilisateur rattaché à une ipv4

Assurance de sécurité par un code

### Transport:

UDP pour les communications Client to Client

TCP pour la relation Client to Host

# II/ Les besoins technique

## Sockets

Pour communiquer entre eux, les clients Babel devront configurer une connexion UDP binaire, cela induit l'ouverture d'un socket pour chaque activité et flux de médias (avec un socket pour chaque sens de flux).

**Exemple :**

Si le client A veut démarrer un appel vocal avec le client B, deux socket sont établis:

* Le premier pour le transport audio de A vers B
* Le second pour le transport audio de B vers A

## Librairies multimédia

Portaudio est la bibliothèque qui doit être utilisée pour le flux audio entre les clients Babel. Tous les flux de médias doivent être compressés afin d'optimiser l'utilisation du réseau.

## Codec de compression

Afin de réduire les besoins en bande passante, les flux audio (et vidéo) doivent être compressés. Pour ce faire, le codec de compression Opus sera toujours utilisé lors d'une communication.

# III/ Ordonnancement des communications

## Connexion

Une communication entre deux clients commence par la création d'un binaire UDP socket. Le port par défaut pour la connexion est 4242, mais le port peut être différent. Dès que la connexion est faite, le client demandant enverra un en-tête décrivant l’activité qu’il veut commencer.

## La négociation de démarrage d’activité

Dans le cas où le client demandant refuse le démarrage d'une activité, il ferme le socket sans envoyer de paquet. Ceci est la seule façon pour le client demandant de savoir si l’activité a été refusée par le client recevant. Différemment, le client recevant renvoie "OK" et l'activité peut commencer. Si l'activité proposée est un appel vocal, un second socket est créé par le client demandé afin de pouvoir renvoyer son flux.

## Pendant l’activité

Toutes les communications via un socket sont à sens unique. Cela s’explique par la conception asynchrone des sockets UDP, ce serait un gâchis d'avoir à attendre que des paquets soit envoyé pour recevoir et traiter d’autres paquets.

## Fin de l’activité

Quand un client cesse une activité, il ferme son socket d’envoi. L'interruption de la communication est explicite, afin que le client correspondant ferme son propre socket d'envoi. Ceci est la façon normale pour arrêter les communications.

# IV/ Le protocole commun client / serveur

## Généralité

Nous allons utiliser une class Packet qui permet l’ajout d’un header aux données.

Le header est formé de 4 unsigned int suivi des données.

Le premier unsigned int correspond à un magic number suivi du code de l’instruction, et enfin le type et la taille des données contenue dans la suite du paquet.

## Les codes d’instructions

|  |  |
| --- | --- |
| Instruction | Code |
| connect | 001 |
| disconnect | 002 |
| contactList | 003 |
| addContact | 004 |
| delContact | 005 |
| askCall | 006 |
| rcvCall | 007 |
| callEstablished | 008 |
| callFailed | 009 |
| endCall | 010 |
| getSound | 011 |
| sendSound | 012 |

## Les types de données

Chaque type de donnée possède un unsigned int correspondant au type de donnée.

|  |  |
| --- | --- |
| Type | Code |
| String | 0xb16b00b5 |
| IntVector | 0x42424242 |

## Le header

Chacun des paquets généré par la class Packet seront formé d’un header suivi des données.

|  |  |
| --- | --- |
| Variables | Description |
| Unsigned int magic | Permet de vérifier que le packet reçu est bien valide |
| Unsigned int instruction | Instruction contient le code de l’instruction, dans le cas de l’envoie de son nous avons 011(getSound) ou 012(sendSound) |
| Unsigned int type | Le code correspondant au type de données contenues dans le paquet |
| Unsigned int size | Taille des données qui suivent le header |
| Char\* data | Les données du paquet |

# V/ Le protocole client-client

La communication client-client est établie en UDP afin d’améliorer les performances sonores.

### getSound/sendSound

Voici les données envoyé lors d’une communication client-client :

|  |  |
| --- | --- |
| Variables | Description |
| Time\_t time | Utilisation d’un timestamp afin d’eviter l’utilisation d’un paquet plus ancien que le précedent |
| Unsigned int bufSize | Contient la taille du buffer son encoder qui suit |
| Char \*soundBuf | Buffer contenant le son encodé par Opus |

# VI/ Le protocole client-serveur

### connect

Permet l’authentification et la mise en place de la connexion d’un client au serveur.

|  |  |
| --- | --- |
| Variables | Description |
| Char username[64] | Contient le nom de l’utilisateur souhaitant se connecter |

### disconnect

Permet la notification au serveur de la déconnection d’un client.

|  |  |
| --- | --- |
| Variables | Description |
| Char username[64] | Contient le nom de l’utilisateur souhaitant se deconnecter |

### contactList

Permet au client de récupérer la liste des clients connectés.

|  |  |
| --- | --- |
| Variables | Description |
| Unsigned int nbClient | Contient le nombre de contact actuellement en ligne |
| Char\* clientList | Buffer contenant tous les utilisateurs actuellement connectés, ils sont séparés par un délimiteur |

### addContact

Ajoute un contact à la liste du client

|  |  |
| --- | --- |
| Variables | Description |
| Char username[64] | Contient le nom du contact à ajouter à la liste |

### delContact

Supprime un contact de la liste du client

|  |  |
| --- | --- |
| Variables | Description |
| Char username[64] | Contient le nom du contact à supprimer de la liste |

### askCall

Envoie au serveur une demande d’appel à un contact.

|  |  |
| --- | --- |
| Variables | Description |
| Char username[64] | Contient le nom du contact à appeler |

### rcvCall

Le client reçoit une requête pour débuter une conversation

|  |  |
| --- | --- |
| Variables | Description |
| Char username[64] | Contient le nom du contact appelant |

### callEstablished

Confirme aux différents clients que la communication a réussi et est en place.

Cette commande n’a pas de paramètres.

### callFailed

Annonce aux différents clients que la mise en place de la communication a échouée.

Cette commande n’a pas de paramètres.

### endCall

Notifie les différents clients de la fin de la communication.

Cette commande n’a pas de paramètres.