Babel - RFC

Simon Ninon – Navid Emad – Alain Nguyen – Davy Tran – Pierre Guego

Dernière mise à jour : 4 novembre 2014

Préambule

Le Babel est un programme de communication vocal (VOIP).

Ce document explique comment les clients communiquent avec le serveur et comment les clients communiquent entre eux.

Communication Client - Client

a. *Préambule*

La communication “client - client” est utilisée dans le cadre de l’envoi de paquet de sons.
La communication se fait en **protocole UDP** afin que la communication audio soit plus performante.

b. *SoundPacket*

Les paquets audio sont formatés grâce à une structure “*SoundPacket*”:

```
#pragma pack(push, 1)
struct SoundPacket {
    int         magic_code;
    int64_t     timestamp;
    int         soundSize;
    char        sound[500];
};
#pragma pack(pop)
```

Type	Nom	Utilité
int	magic_code	Permet de vérifier la validité du paquet. Le magic code doit valoir: **0x150407CA**
int64_t	timestamp	Date de l’envoie du paquet de son. Si la date est antérieure au paquet précédemment reçu, le client peut ne pas traiter le paquet afin de gagner en performance.
int	soundSize	taille du paquet de son dans le buffer “sound”
char[500]	sound	Buffer de son. Ce buffer contient un champ de bytes qui correspond à du son encodé par le codec “opus”

Il est important de noter dans l’exemple la présence du “*#pragma pack*”. Cette directive permet de forcer le compilateur à ne pas faire de padding afin de ne pas avoir de soucis lors de la récupération et de l’envoie des paquets.

c. *Comportement*

L’émetteur d’un paquet ne se soucie pas d’un éventuel problème de communication et de réseau et n’intègre pas de gestion de timeout.
Tout paquet ne comportant pas le bon magic code ou ayant un timestamp inférieur au précédent paquet reçu ne doit pas être traité.
Le son transmis doit obligatoirement avoir été encodé au préalable.

Communication Client - Serveur

a. Préambule

La communication client - serveur est au cœur du Babel. C'est le serveur qui gère les comptes utilisateurs, qui centralise les informations et transmets des informations entre les contacts.
La communication client - serveur se fait par le **protocole TCP** afin d'assurer une bonne transmission des commandes.

b. Formatage des commandes

Toutes les commandes, qu'elles soient envoyées par un client ou par un serveur, possèdent le même header.

```
#pragma pack(push, 1)
struct Header {
    int magicCode;
    int instructionCode;
};
#pragma pack(pop)
```

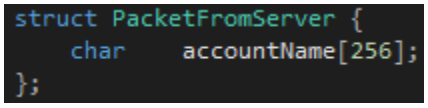
Type	Nom	Utilité
int	magic_code	Permet de vérifier la validité du paquet. Le magic code doit valoir: **0x150407CA**
int	instructionCode	Permet d'identifier la commande envoyée.

Même si les paquets possèdent tous le même header, ils possèdent en revanche un "*body*" différent: chaque commande possède sa propre structure, adaptée à ses besoins.
C'est grâce à l'instruction code que le client ou le serveur sera en mesure de déterminer ce qu'il va recevoir comme body.

c. Formatage des commandes envoyées par le serveur

i. add

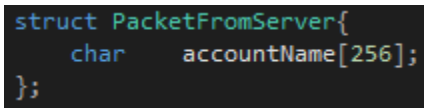
Utilisée pour notifier un client qu'un autre client souhaite l'ajouter en contact.



Type	Nom	Utilité
char[256]	accountName	Nom de compte de la personne qui envoie la demande

ii. call

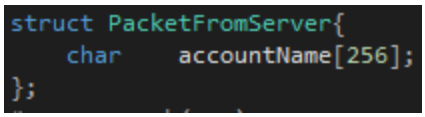
Utilisée pour notifier un client qu'un de ses contacts souhaite l'appeler.



Type	Nom	Utilité
char[256]	accountName	Nom de compte de la personne qui envoie la demande

iii. close_call

Utilisée pour notifier un client que l'appel en cours est terminée.



Type	Nom	Utilité
char[256]	accountName	Nom de compte de la personne avec qui le client est en communication

iv. del

Utilisée pour notifier un client qu'un autre client l'a supprimé de sa liste de contact.

```
struct PacketFromServer{
    char    accountName[256];
};
```

Type	Nom	Utilité
char[256]	accountName	Nom de compte de la personne qui a supprimé le client de sa liste de contact

v. show

Utilisée pour donner des informations à un client concernant un utilisateur.

```
struct PacketFromServer{
    char    accountName[256];
    char    pseudo[256];
    char    status;
    char    isConnected;
};
```

Type	Nom	Utilité
char[256]	accountName	Nom de compte de la personne dont on transmet les informations
char[256]	pseudo	Pseudo de la personne dont on transmet les informations
char	status	Statuts de la personne dont on transmet les informations. Ce statuts correspond à un enum définit plus loin dans cette documentation
char	isConnected	Indique si la personne est actuellement connectée. Cet attribut correspond à un booléen et contiendra 0 ou 1 (1 signifiant que l'utilisateur est connecté)

vi. send

Utilisée pour transmettre un message d'un utilisateur à un autre utilisateur.

```
struct PacketFromServer{
    char    accountName[256];
    char    textMessage[4096];
};
```

Type	Nom	Utilité
char[256]	accountName	Nom de compte de la personne qui envoie le message
char[4096]	textMessage	message

vii. err

Utilisée pour notifier un utilisateur que la dernière action effectuée s'est bien déroulée ou non.

```
#pragma pack(push, 1)
struct PacketFromServer{
    int instructionCode;
    int errorCode;
};
#pragma pack(pop)
```

Type	Nom	Utilité
int	instructionCode	numéro de l'instruction concernée. Cet attribut correspond à un enum définit plus loin dans cette documentation
int	errorCode	Code d'erreur. Cet attribut correspond à un enum définit plus loin dans cette documentation

viii. accept_call

Cette commande permet 2 choses:

- notifier à un utilisateur que le contact qu'il a essayé d'appeler à accepter, ou non, la demande d'appel
- Informer les 2 contacts qui souhaitent communiquer à quelle IP ils doivent d'adresser.

```
struct PacketFromServer{
    char    accountName[256];
    char    host[15];
    char    hasAccepted;
};
```

Type	Nom	Utilité
char[256]	accountName	Nom de compte du contact avec qui le client va communiquer
char[15]	host	Adresse IP du contact avec qui le client va communiquer
char	hasAccepted	Indique si la personne qui a reçu la demande d'appel à accepter ou non la demande

d. Formatage des commandes envoyées par le client

i. reg

Utilisée pour créer un compte.

```
#pragma pack(push, 1)
struct PacketFromClient{
    ICommand::Header    header;
    char                accountName[256];
    char                pseudo[256];
    char                password[256];
};
#pragma pack(pop)
```

Type	Nom	Utilité
char[256]	accountName	Nom de compte
char[256]	pseudo	pseudo
char[256]	password	mot de passe

ii. log

Utilisée pour s'authentifier et se connecter.

```
#pragma pack(push, 1)
struct PacketFromClient{
    ICommand::Header    header;
    char                accountName[256];
    char                password[256];
};
#pragma pack(pop)
```

Type	Nom	Utilité
char[256]	accountName	Nom de compte
char[256]	password	mot de passe

iii. list

Utilisée pour demander la liste de contact. Cette commande ne prend pas de paramètres.

```
#pragma pack(push, 1)
struct PacketFromClient{
    ICommand::Header    header;
};
#pragma pack(pop)
```

iv. show

Utilisée pour demander des informations sur un contact.

```
struct PacketFromClient{
    ICommand::Header    header;
    char                accountName[256];
};
```

Type	Nom	Utilité
char[256]	accountName	Nom de compte de la personne dont on souhaite obtenir des informations

v. add

Permet d'envoyer une demande d'ajout de contact.

```
struct PacketFromClient {
    ICommand::Header    header;
    char                accountName[256];
};
```

Type	Nom	Utilité
char[256]	accountName	Nom de compte de la personne que l'on souhaite ajouter

vi. accept_add

Permet de répondre à une demande d'ajout de contact.

```
#pragma pack(push, 1)
struct PacketFromClient {
    ICommand::Header    header;
    char                accountName[256];
    char                hasAccepted;
};
#pragma pack(pop)
```

Type	Nom	Utilité
char[256]	accountName	Nom de compte de la personne qui souhaite nous ajouter en contact
char	hasAccepted	Réponse à la demande

vii. call

Utilisée pour envoyer une demande d'appel à un contact.

```
struct PacketFromClient{
    ICommand::Header    header;
    char                accountName[256];
};
```

Type	Nom	Utilité
char[256]	accountName	Nom de compte de la personne que l'on souhaite appeler

viii. accept_call

Utilisée pour répondre à une demande d'appel.

```
struct PacketFromClient{
    ICommand::Header    header;
    char                accountName[256];
    char                hasAccepted;
};
```

Type	Nom	Utilité
char[256]	accountName	Nom de compte de la personne qui envoie la demande
char	hasAccepted	Réponse à la demande

ix. close_call

Utilisée pour mettre fin à un appel en cours.

```
struct PacketFromClient{
    ICommand::Header    header;
    char                accountName[256];
};
```

Type	Nom	Utilité
char[256]	accountName	Nom de compte de la personne avec qui l'on est en communication

x. del

Utilisée pour supprimer un contact de sa liste de contact.

```
struct PacketFromClient{
    ICommand::Header    header;
    char                accountName[256];
};
```

Type	Nom	Utilité
char[256]	accountName	Nom de compte de la personne que l'on souhaite supprimer de sa liste de contact

xi. exit

Utilisée pour se déconnecter. Cette commande ne prend pas de paramètres.

```
#pragma pack(push, 1)
struct PacketFromClient{
    ICommand::Header    header;
};
#pragma pack(pop)
```

xii. update

Utilisée pour mettre à jour les informations du client actuellement connecté.

```
#pragma pack(push, 1)
struct PacketFromClient{
    ICommand::Header    header;
    char                accountName[256];
    char                pseudo[256];
    char                password[256];
    char                status;
};
#pragma pack(pop)
```

Type	Nom	Utilité
char[256]	accountName	Nom de compte de la personne qui souhaite changer ses informations. Le nom de compte ne peut pas être modifié
char[256]	pseudo	nouveau pseudo
char[256]	password	nouveau mot de passe
char	status	nouveau status. Le status correspond à un enum définit plus bas dans cette documentation

xiii. send

Cette commande est utilisée pour envoyer un message à un contact.

```
struct PacketFromClient{
    ICommand::Header    header;
    char                accountName[256];
    char                textMessage[4096];
};
```

Type	Nom	Utilité
char[256]	accountName	Nom de compte de la personne à qui on souhaite envoyer un message
char[4096]	textMessage	Message à envoyer

e. Cas d'utilisations et réponses attendues

i. Inscription

Client 1 envoie la commande "reg" au serveur.
Le serveur renvoie la commande "err" au Client 1.

ii. Connexion

Client 1 envoie la commande "log" au serveur.
Le serveur renvoie la commande "err" au Client 1.
En cas de succès, le serveur envoie la commande "show" à tous les contacts de Client 1.
En cas de succès, le serveur renvoie une boucle de "show" à Client 1 pour lui lister ses contacts.

iii. Ajout d'un contact

Client 1 envoie la commande "add" au serveur.
Le serveur renvoie "err" au Client 1.
En cas de succès, le serveur envoyé la commande "accept_add" au Client 2.

iv. Accepter une invitation de contact

Client 1 envoie la commande "accept_add" au serveur.
Le serveur renvoie "err" au Client 1.
En cas de succès, le serveur envoie la commande "show" aux deux clients.

v. Supprimer un contact

Client 1 envoie la commande "del" au serveur.
Le serveur renvoie la commande "err" au client 1.
En cas de succès, le serveur envoie la commande "del" au 2 clients.

vi. Appeler

Client 1 envoie la commande "call" au serveur.
Le serveur renvoie la commande "err" au client 1.
En cas des succès, le serveur envoie la commande "accept_call" au client 2.

vii. Accepter un appel

Client 1 envoie la commande "accept_call" au serveur.
Le serveur renvoie la commande "err" au client 1.
En cas de succès, 2 possibilités sont possibles:

- client 1 a accepté: le serveur envoie la commande “accept_call” aux deux clients
- client 1 a refusé: le serveur envoie la commande “accept_call” à client 2.

viii. Raccrocher

Client 1 envoie la commande “close_call” au serveur.
Le serveur renvoie la commande “err” à client 1.
En cas de succès, le serveur envoie la commande “close_call” aux 2 clients.

ix. Envoyer un message

Client 1 envoie la commande “send” au serveur.
Le serveur renvoie la commande “err” à client 1.
En cas de succès, le serveur envoie la commande “send” à client 2.

x. Se déconnecter

Client 1 envoie la commande “exit” au serveur.
Le serveur renvoie la commande “err” à client 1.
En cas de succès, le serveur envoie la commande “show” à tous les contacts de client 1.

xi. Modifier ses informations

Client 1 envoie la commande “update” au serveur.
Le serveur renvoie la commande “err” à client 1.
En cas de succès, le serveur envoie la commande “show” à tous les contacts de client 1.
En cas de succès, le serveur envoie la commande “show” à client 1.

Annexes

a. Instructions codes

Code	Instruction
0x01	reg
0x02	log
0x03	list
0x04	show
0x05	add
0x06	accept_add
0x07	del
0x08	exit
0x09	update
0x0A	send
0x0B	call
0x0C	accept_call
0x0D	close_call
0x0E	err

b. Errors codes

Code	Erreur
0x00	Everything is OK MODAFUCKER
0x01	The impossible happened
0x02	FAIL_INIT_SOCKET
0x03	FAIL_INIT_AUDIO
0x04	WRONG_PACKET_STRUCT
0x05	REGISTER_ACC_ALREADY_USED
0x06	LOGIN_WRONG_PASSWORD
0x07	UNKNOWN_ACCOUNT
0x08	ACTIONS_TO_OFFLINE_ACCOUNT
0x09	CANNOT_DO_CALL_MULTIPLE
0x0A	CANNOT_ACCEPT_CALL_FROM_NOT_A_CALLER
0x0B	CANNOT_CLOSE_CALL_WHEN_YOU_ARENT_CALLING
0x0C	LOGIN_ON_ALREADY_LOGGED_ACCOUNT
0x0D	CANNOT_ADD_YOURSELF
0x0E	CANNOT_CALL_YOURSELF
0x0F	YOU_ARE_NOT_LOGGED
0x10	ALREADY_CONNECTED
0x11	ALREADY_IN_YOUR_CONTACT_LIST
0x12	NOT_IN_YOUR_CONTACT_LIST
0x13	BUSY_CONTACT_CANNOT_REPLY
0x14	NOT_IN_COMMUNICATION_WITH_HIM
0x15	CANNOT_LOAD_DATA
0x16	CANNOT_SAVE_DATA
0x17	YOU_CANNOT_UPDATE_OTHER_ACCOUNT
0x18	INVALID_STATUS_ID

c. Statuts

Code	Statuts
0x00	Connecté
0x01	Déconnecté
0x02	Occupé
0x03	Absent
0x04	Kipour
0x05	Grasse mat’ 10h
0x06	Ramadan
0x07	Sport
0x08	Petit coin
0x09	YOLO
0x0A	Parti bouder