RFC ChatVaBien

Résumé

ChatVaBien est un protocole applicatif permettant à ses utilisateurs d'envoyer et de recevoir des messages et des fichiers en privé ou d'envoyer des messages publiques à tous les utilisateurs.

Ce protocole propose un mode de communication en peer-to-peer pour les messages privés et avec un serveur permettant de contacter tous les clients connectés pour les messages publiques.

Il est important de noter que le protocole ChatVaBien utilise le protocole de transport TCP (RFC 793) pour échanger des données sur le réseau via des paquets.

1. Introduction

1.1. Objectifs

L'objectif de cette RFC est de fournir les principes et les règles pour permettre une implémentation du protocole ChatVaBien.

1.2. Terminologie

client : une instance d'application connectée au serveur, un client doit toujours avoir un port d'écoute ouvert

serveur : un logiciel centralisant les connexions des clients

message privé : un message envoyé entre deux clients sans passer par le serveur

message publique : un message envoyé par un client à tous les autres clients en passant par le serveur

réseau : un ensemble de clients connecté au même serveur font partie d'un même réseau

2. Principes fondamentaux

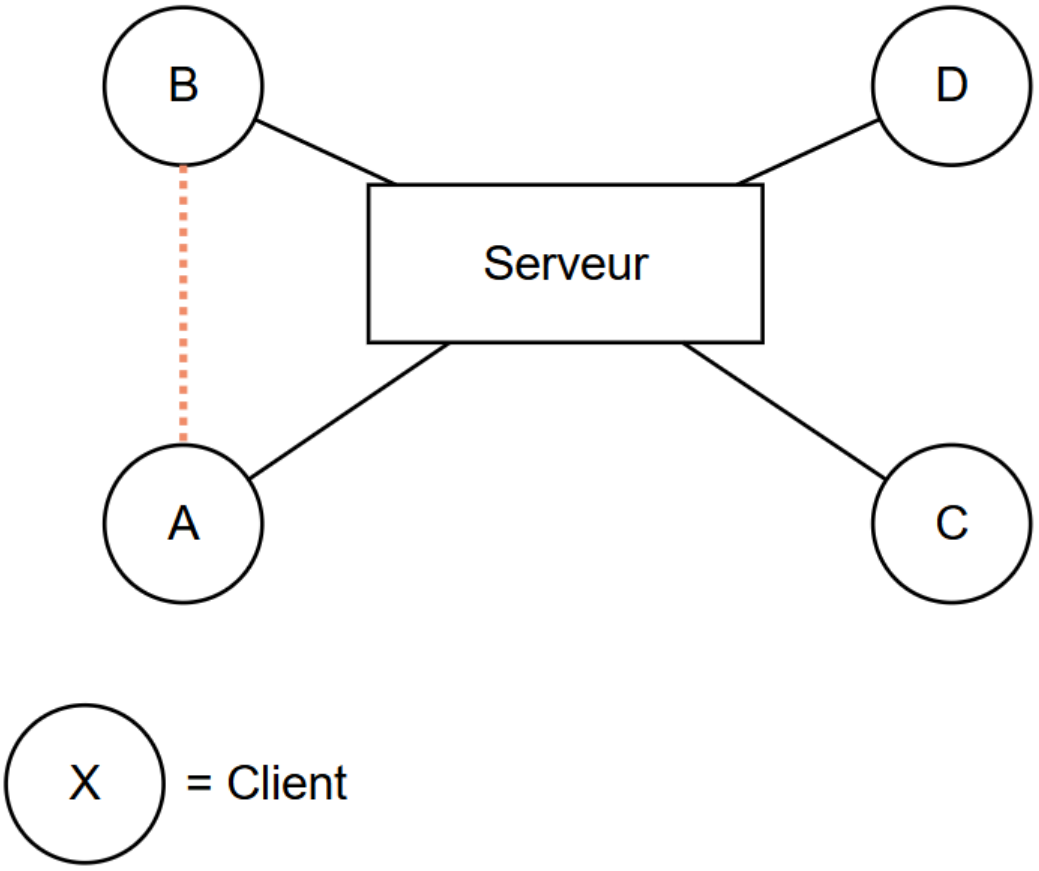
Le principe fondamental de ChatVaBien est de permettre l'envoi de messages privés ou publiques entre les clients d'un même réseau. Les clients se connectent à un serveur. Chaque client connecté est identifié par un pseudonyme. Le protocole doit permettre deux formes d'accès aux clients : un accès authentifié un par mot de passe et un accès sans mot de passe. On supposera que le serveur a accès sous une forme ou une autre à une base de données de pseudonymes et de mots de passe. Le protocole ne s'occupe pas de la création et de la mise à jour de cette base. Le serveur doit garantir que deux clients ne peuvent pas avoir le même pseudonyme et qu'un client authentifié sans mot de passe n'utilise pas le pseudonyme d'un utilisateur de la base de données. Une fois connectés et identifiés par un pseudonyme, les clients peuvent :

- envoyer des messages publiques, qui seront transmis à tous les clients connectés.
- envoyer des messages privés et des fichiers à un autre client.

Par rapport à un serveur de chat standard (type IRC), la particularité de ce protocole est que tous les messages privés (dont les fichiers) sont envoyés par une connexion directe entre les deux clients: le serveur permet juste dans ce cas de s'échanger les adresses des clients.

Le protocole permettra aux clients de faire des demandes de communication privée et d'accepter/refuser la demande de communication privée. Entre deux clients, il ne devra y avoir au maximum une connexion privée. Une fois que la communication privée entre deux clients est établie, l'envoi/réception de fichiers ne doit pas bloquer les messages entre ces deux clients. Par exemple, si un client A envoie en privé un fichier de 4 Go à un client B. Le client A doit pouvoir envoyer et

recevoir des messages du client B.



3. Fonctionnement

3.1. Protocole de transport

L'implémentation du protocole devra être réalisée en utilisant le protocole TCP pour le transport des paquets et la connexion.

3.2. Charset

Le charset utilisé pour l'encodage du texte est **UTF-8**.

3.3. Nombres entiers

L'utilisation des types entiers est en notation **BIG ENDIAN**.

3.4. Limite de taille d'un texte dans un paquet

Un texte dans un paquet ne doit en aucun cas dépasser une taille fixée à **1024 octets** et doit être précédé de sa taille une fois encodé en UTF-8. Cette taille ne peut donc pas dépasser 1024.

4. Cas d'utilisations

4.1. La connexion d'un client au serveur

Dès lors qu'un serveur est démarré, des clients peuvent s'y connecter pour rejoindre le réseau.
Le client voulant se connecter va utiliser le protocole TCP pour initier la connexion vers le serveur puis envoyer un paquet de connexion via le protocole comme défini :

Il y a deux possibilités :

- **Connexion sans authentification**

byte	short	text
1	taillePseudo	pseudo

- **Connexion avec authentification**

byte	short	text	short	text
2	taillePseudo	pseudo	taillePassword	password

4.2. La réponse du serveur à la connexion d'un client

byte	byte
3	statusCode

statusCode peut prendre plusieurs valeurs selon la réussite ou non de la connexion :

- 0 -> succès
- 1 -> pseudo ou mot de passe incorrect
- 2 -> pseudo est déjà utilisé et ne peut être réutilisé

idée pour éviter DoS : timeout pour les clients non authentifiés -> kick
voir sujet de TP clients TCP non bloquants exo 2 (partie optionnelle)

4.3. Envoi d'un message public par le client connecté

4.3.1. Envoi du message

byte	short	text
4	tailleContenu	contenu

Il n'est pas nécessaire d'envoyer le pseudo, le serveur reconnait la provenance du message et peut retrouver cette information.

4.3.2. Transmission du message aux clients connectés

byte	short	text	short	text
5	tailleContenu	contenu	taillePseudo	pseudo

C'est un broadcast à tous les clients connectés. Ici le pseudo de l'émetteur est renseigné par le serveur pour indiquer aux client qui a envoyé le message.

4.4. Demande d'envoi d'un message privé

4.4.1. L'émetteur envoie un paquet de demande de MP au serveur :

byte	short	text
6	taillePseudoDestinataire	pseudoDestinataire

4.4.2. Le serveur transmet la demande au destinataire

- **Si le destinataire n'est pas connecté au réseau, le protocole du serveur se charge de refuser**

byte	short	text
6	taillePseudoEmetteur	pseudoEmetteur

4.4.3. Acceptation ou refus de la demande

Si une connexion est déjà établie avec l'émetteur de la demande, le protocole du destinataire se charge de refuser

- en cas de refus du destinataire :

byte	short	text	byte
7	taillePseudoEmetteur	pseudoEmetteur	0

et le serveur se charge de transmettre l'information à l'émetteur :

byte	short	text	byte
7	taillePseudoDestinataire	pseudoDestinataire	0

- en cas d'acceptation du destinataire :

byte	short	text	byte	long	address
7	taillePseudoEmetteur	pseudoEmetteur	1	***nonce***	***adresseDestinataire***

où ***nonce*** est un entier long tiré aléatoirement par le destinataire qui sera associé à un unique pseudoEmetteur en particulier

où ***adresseDestinataire*** aura le format suivant :

- Si l'adresse IP est en IPV4 avec le format suivant `a.b.c.d:port`, alors le paquet contiendra les champs suivants :

byte	byte	byte	byte	byte	short
0	a	b	c	d	port

- Sinon, si l'adresse IP est en IPV6 avec le format suivant `[ab:cd:ef:gh:ij:kl:mn:op]:port`, alors le paquet contiendra les champs suivants :

byte	byte * 16	short
1	IPV6	port

Le champ contenant l'IPV6 contient les 16 octets au complet non tronqués.

et le serveur transmettra à l'émetteur :

byte	short	text	byte	long	address
7	taillePseudoDestinataire	pseudoDestinataire	1	***nonce***	***adresseDestinataire***

4.4.4 En cas de réponse positive

L'émetteur se connecte à destinataire avec l'adresse reçue via le serveur et envoie un paquet de connexion :

byte	short	text	long
8	taillePseudoEmetteur	pseudoEmetteur	nonce

Si le destinataire ne reconnait pas le pseudo qui se connecte ou que le nonce donné ne correspond pas à celui donné au préalable pour ce pseudo, il coupe la connexion.

4.5. Envoi d'un message privé

Il y a deux possibilités :

Cas du message

byte	short	text
9	tailleContenu	contenu

Cas du fichier

1. On commence par prévenir le serveur qu'un fichier d'une taille arbitraire va être envoyé avec un paquet de *header* :

byte	short	text	int
10	tailleNomFichier	nomFichier	tailleFichier

2. Puis on envoie le contenu fichier en le découpant si nécessaire en de multiples paquets de *Contenu* dont la taille de la section `contenu` ne doit pas dépasser **8192 octets** :

byte	short	binary
11	tailleContenu	contenu

Pour signaler la fin du fichier, le serveur doit recevoir un ou plusieurs paquets *Contenu* de sorte à ce que la somme de leur taille soit égale à la taille totale du fichier indiquée dans le paquet *header* au début.

> **Remarque**
>
> Afin que l'envoi d'un paquet ne monopolise pas la connexion entre clients, il faut un mécanisme de priorité pour privilégier l'envoi d'un message si les deux actions venaient à se chevaucher.
>
> #### Exemple :
>
> A envoie un fichier à B
> Pendant l'envoi du fichier, A envoie un message à B
> L'envoi du message interrompt momentanément l'envoi du fichier pour prendre la priorité
> L'envoi du fichier reprend dès que le message est envoyé