Équipe 11

PolyPaint™
Protocole de communication

Version 1.3

Historique des révisions

Date	Version	Description	Auteur
2018-01-17	1.0	Élaboration des protocoles de communication.	Équipe 11
2018-01-30	1.1	Diagramme de la section 2	Équipe 11
2018-02-09	1.2	Révision finale	Équipe 11
2018-04-15	1.3	Révision pour deuxième remise	Équipe 11

Table des matières

Protocole de communication

1. Introduction

Le présent document sert à décrire le fonctionnement des communications entre les différents composants du logiciel PolyPaint. La section 2 de ce document porte sur la description de la communication entre les différents composants du produit tel que la communication client léger et serveur. La partie 3 contient toute l'information sur le contenu de tous les paquets qui seront utilisés dans la communication. La section 4 indique le protocole lors d'une détection d'erreur de transmission. La section suivante indique le protocole de délai et lors de perte d'informations. Finalement, la section 6 explique le protocole d'authentification pour les utilisateurs.

2. Communication client-serveur

Les clients communiqueront avec le serveur à l'aide de sockets. Ceux-ci devront donc préciser l'adresse IP du serveur et le port sur lequel il écoute. La communication se fera sous l'architecture suivante :

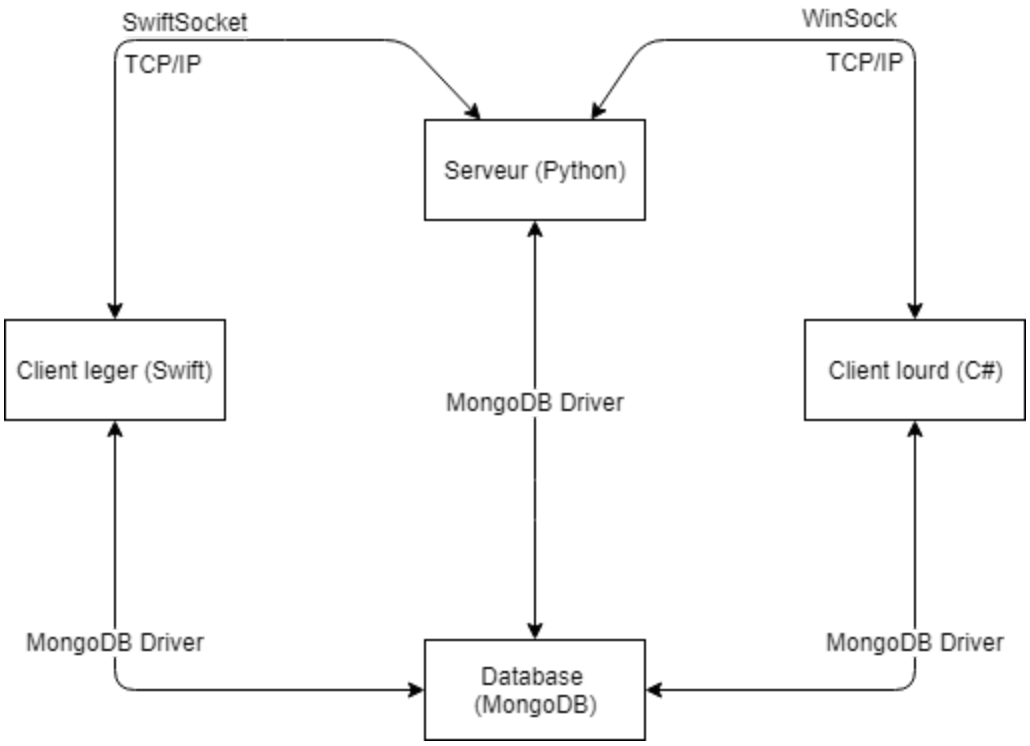

Figure 1 : Architecture de communication du système complet.

Les 2 clients communiqueront sur le protocole TCP/IP, chacun avec leur libraire respective. Les clients lourds qui seront exécutés sur Windows utiliseront Winsock pour la gestion de sockets. En ce qui concerne l'application IOS, elle utilisera la librairie *open source* SocketSwift qui peut être téléchargée directement de Github. Dans les 2 situations, les communications seront gérées par des threads.

Le serveur utilisera les sockets fournis par défaut par Python et de la librairie asyncio. D'ailleurs, la communication avec le serveur de MongoDB se fera avec la librairie pymongo qu'il faudra télécharger dans le système.

Les clients pourront tous les deux communiquer directement avec la base de données à l'aide d'un driver MongoDB.

3. Description des paquets

Toute la communication réseau en TCP/IP doit être encodée en petit boutiste.

Tous les paquets en générale devront respecter la structure suivante:

Description:	**Taille**	**Type ID**	**Date**	**Taille nom usager**	**Nom**	**Donnée**
Taille en octets	2	1	4	1	Taille nom usager	Le reste (taille variable).

Tableau 1. Paquet message en TCP/IP.

Description:

- Taille: La taille totale du paquet en octets.
- Type ID: Contient un identifiant du type de paquet (message, trait, pixel, etc.). Elle peut prendre une valeur de 0 à 255. Ces types seront identifiés dans le code source.
- Date: Contient la date de l'émission du paquet. Celle-ci est enregistrée par le client lors de l'envoi du paquet au serveur. Le type de valeur que doit prendre ce champ de 4 octets doit être une valeur UTC en secondes.
- Taille nom usager: Contient la quantité de caractères du nom de l'usager. Les valeurs possibles sont de 0 à 255. Toutefois, cette plage de valeur sera limitée dans les clients.
- Nom: Un identifiant du client qui sera son nom choisi de son profil.
- Donnée: Cette section sera utilisée différemment pour chaque type de paquet afin de convenir au besoin.

3.1 Les Type ID utilisés

Les Types ID devront être contenus dans un enum dans le code. Il est très important que leurs valeurs soient identiques chez le client lourd, le client léger et le serveur afin que la communication soit possible.

Nom dans l'enum	**ID**	**Description**
MESSAGE	0	Déprécié
LINE_MODE	1	Déprécié.
PIXEL_MODE	2	Déprécié.
CONNECTION	3	Requête d'authentification entre le client et le serveur.
LOBBY_CREATION	4	Créer une nouvelle session pour dessiner.
LOBBY_FETCH	5	Récupérer une liste de tous les lobbys existants du serveur.

LOBBY_CONNECT	6	Se connecter à une session existante de dessin.
CHANNEL_CREATION	7	Créer un nouveau canal de discussion.
CHANNEL_CONNECTION	8	Se connecter à un canal de discussion existants.
CHANNEL_DISCONNECT	9	Quitter un canal de discussion.
CHANNEL_MESSAGE	10	Envoyer un message dans un canal de discussion.
CHANNEL_FETCH	11	Récupérer une liste de tous les canaux de discussion existants.
ADD_STROKE	12	Ajouter un nouveau trait, mode par traits.
REMOVE_STROKE	13	Enlever un trait du canvas, mode par traits.
INSERT_STROKE	14	Insérer un trait dans la collection de traits à un certain index, mode par traits.
SELECTED_STROKE	15	Sélectionner un trait, mode par traits.
UNSELECTED_STROKE	16	Désélectionner un trait, mode par traits.
REPLACE_STROKE	17	Remplacer un trait par son déplacement ou sa modification, mode par traits.
SEND_POINTS	18	Envoyer une liste de points à dessiner, mode par pixels.
CREATE_SELECTION	19	Créer une nouvelle sélection et extraire les pixels dans la sélection, mode par pixels.
REMOVE_SELECTION	20	Supprimer une sélection et écrire les pixels de la sélection dans le canevas principal, mode par pixels.
UPDATE_SELECTION	21	Déplacer, faire une mise à l'échelle ou tourner une sélection existante, mode par pixels.
APPLY_FILTER_SELECTION	22	Appliquer un effet de filtre dans la sélection, mode par pixels.
RESIZE_CANVAS	23	Redimensionner le canevas avec des données du type double.
RESET_CANVAS	24	Enlever tous les éléments sur un canevas et retourner à un canevas blanc.
BUCKET_FILL	25	Appliquer l'outil de remplissage à partir d'une certaine position, mode par pixels.
UPDATE_DATABASE	26	Mettre à jour la représentation du lobby dans la base de données.
LOBBY_DISCONNECT	27	Se déconnecter d'un lobby.
LOBBY_FREEZE_STATE	28	Geler toutes modification dans un certain lobby pour donner le temps à un client de charger les informations de la session de dessin.

FFETCH_DATABASE	29	Dire à un client de charger un lobby de la base de données.
CREATE_ACCOUNT	30	Créer un nouveau utilisateur pour l'application.
LOBBY_INFO_CHANGE	31	Changer le mode de protection d'un lobby.
LOBBY_PLAYER_KICK	32	Expulser un utilisateur suite à un changement de protection du lobby.
RESIZE_CANVAS_INT	33	Redimensionner le canevas avec des données du type integer.

Tableau 2. Les Type ID actuellement utilisés avec leurs descriptions.

3.2 Le champ données des paquets

Cette section décrira le contenu que chaque type de paquet va contenir dans la section donnée.

3.2.1 Paquet MESSAGE

Déprécié

3.2.2 Paquet LINE_MODE

Déprécié

3.2.3 Paquet PIXEL_MODE

Déprécié

3.2.4 Paquet CONNECTION

Il y a deux types de valeurs possibles pour ce type de paquet:

1) Lorsqu'un client veut se connecter au serveur, la section donnée est une chaîne de caractère en UTF-8 qui correspond au mot de passe du compte de l'utilisateur.

2) Lorsque le serveur envoie ce paquet aux clients, la section donnée est un nombre entier de 32 bits pour représenter un état:

Identifiant	**Valeur**	**Description**
ACCEPTED	0	La connection au serveur est acceptée.
USER_ALREADY_CONNECTED	1	La connection est refusée. Il y a déjà un utilisateur connecté dans le serveur.
BAD_PASSWORD	2	La connection est refusée. Le mot de passe est incorrecte.

Tableau 3. Valeurs possibles suite à une tentative de connection au server.

3.2.5 Paquet LOBBY_CREATION

Il y a deux types de données possibles pour ce packet:

1) Si ce message est envoyé par le client vers le serveur, le contenu du message est une chaîne de caractère qui respecte le format JSON. Il est important qu'il y a les 5 attributs suivant dans le message:

Nom	**Valeur possibles**	**Description**
Filename	Chaîne de caractères du nom de l'image.	Le nom de l'image.
Owner	Chaîne de caractères du nom du créateur de l'image.	Le créateur de l'image.
Mode	Chaîne de caractères possible pour le mode: ● stroke ● pixel	Le mode de dessinage.
Protection	Chaîne de caractère possibles pour la protection: ● public ● private ● protected	Le type de protection utilisé par le créateur.
Passcode	Chaîne de caractères	Le mot de passe pour la protection protected. Ce champ est obligatoire.

Tableau 4. Structure du JSON contenant les informations d'un lobby à créer

2) Si ce paquet est envoyé par le serveur au clients, le contenu du message sera un nombre naturel de 32 bits:

Identifiant	**Valeur**	**Description**
CREATED	0	La session a été créé.
CONNECTED	1	Pas utilisé dans ce contexte.
EXISTS	2	La session existe déjà dans le serveur.
DOES_NOT_EXIST	3	Pas utilisé dans ce contexte.

Tableau 5. Description de la valeur de retours par le serveur.

3.2.6 Paquet LOBBY_FETCH

Le contenu du message est une chaîne de caractère qui respecte le format JSON si elle provient du serveur. Un client qui envoie ce paquet au serveur n'a pas de contenu, puisqu'il s'agit d'une simple requête.
Le format JSON de ce paquet va contenir les informations sur les sessions actives dans le serveur. Ce JSON sera une liste d'objets ayant ce format:

Nom	**Valeur possibles**	**Description**
Filename	Chaîne de caractères du nom de l'image.	Le nom de l'image.

Owner	Chaîne de caractères du nom du créateur de l'image.	Le créateur de l'image.
Mode	Chaîne de caractères possible pour le mode: ● stroke ● pixel	Le mode de dessinage.
Protection	Chaîne de caractère possibles pour la protection: ● public ● private ● protected	Le type de protection utilisé par le créateur.
Passcode	Chaîne de caractères	Le mot de passe pour la protection protected. Ce champ est obligatoire.
Clients	Une list de chaine de caractère.	Contient les noms d'utilisateurs qui sont actuellement connecté dans la session.

Tableau 6. Structure du JSON contenant les informations des lobbys

3.2.7 Paquet LOBBY_CONNECT

Il y a deux types de données possibles pour ce packet:

1) Si ce message est envoyé par le client vers le serveur, le contenu du message est une chaîne de caractère qui respecte le format JSON. Il est important qu'il y a les 5 attributs suivant dans le message:

Nom	**Valeur possibles**	**Description**
Filename	Chaîne de caractères du nom de l'image.	Le nom de l'image.
Owner	Chaîne de caractères du nom du créateur de l'image.	Le créateur de l'image.
Mode	Chaîne de caractères possible pour le mode: ● stroke ● pixel	Le mode de dessinage.
Protection	Chaîne de caractère possibles pour la protection: ● public ● private ● protected	Le type de protection utilisé par le créateur.
Passcode	Chaîne de caractères	Le mot de passe pour la protection protected. Ce champ est obligatoire.

Tableau 7. Structure du JSON contenant les informations du lobby que l'on souhaite se connecter

2) Si ce paquet est envoyé par le serveur au clients, le contenu du message sera un nombre naturel de 32 bits:

Identifiant	**Valeur**	**Description**
CREATED	0	Pas utilisé dans ce contexte.

CONNECTED	1	L'utilisateur est connecté dans la session.
EXISTS	2	Pas utilisé dans ce contexte.
DOES_NOT_EXIST	3	La session que l'utilisateur veut se connecter n'existe pas.

Tableau 8. Description de la valeur de retours par le serveur.

3.2.8 Paquet CHANNEL_CREATION

Le contenu de la section donnée est une chaîne de caractères. Elle va contenir le nom du canal à créer.

Description	**Nom du canal à créer**
Octet	Taille du nom

Tableau 9. La description de la structure de donnée du paquet création de canal de discussion

3.2.9 Paquet CHANNEL_CONNECT

Le contenu de la section donnée est une chaîne de caractères. Elle va contenir le nom du canal à se connecter.

Description	**Nom du canal à connecter**
Octet	Taille du nom

Tableau 10. La description de la structure de donnée du paquet connection à un canal de discussion

3.2.10 Paquet CHANNEL_DISCONNECT

Le contenu de la section donnée est une chaîne de caractères. Elle va contenir le nom du canal à se déconnecter.

Description	**Nom du canal à déconnecter**
Octet	Taille du nom

Tableau 11. La description de la structure de donnée du paquet déconnection à un canal de discussion

3.2.11 Paquet CHANNEL_MESSAGE

Le contenu de la section donnée est une chaîne de caractères. Elle va contenir le contenu du message que le client souhaite envoyer. Il est à noter que le message suit cette structure: <NOM_DU_CANAL> ; <CONTENU>

Description	**Message à envoyer** ("<NOM_DU_CANAL> ; <CONTENU>")
Octet	Taille du message

Tableau 12. La description de la structure de donnée du paquet d'envoi de message.

3.2.12 Paquet CHANNEL_FETCH

Le contenu du message est une chaîne de caractère qui respecte le format JSON si elle provient du serveur. Un client

qui envoie ce paquet au serveur n'a pas de contenu, puisqu'il s'agit d'une simple requête.
Le format JSON de ce paquet va contenir les informations sur les sessions actives dans le serveur. Ce JSON sera une liste d'objets ayant ce format:

Nom	**Valeur possibles**	**Description**
Name	Chaîne de caractères du nom du canal.	Le nom du canal.
Clients	Liste de chaîne de caractères du nom de chaque client connectés à ce canal	Nom de chaque client connectés au canal de discussion.

Tableau 13. Structure du JSON contenant les informations des canaux de discussion

3.2.13 Paquet ADD_STROKE

Le contenu de la section donnée est un format sérialisé sur toute l'information nécessaire pour la création d'un trait.

Description	**Index**	**RGBa**	**Longueur**	**Largeur**	**Points**
Octet	4	4	4	4	Nombre de points * 8 octets

Tableau 14. La description de la structure de donnée du paquet mode par trait pour l'envoi d'un trait ajouté.

La description de chacune des champs:

- Index: L'index du trait dans la collection de trait.
- RGBa: Contient la couleur avec l'opacité du trait.
- Hauteur: La hauteur du point du crayon de ce trait
- Largeur: La largeur du point du crayon de ce trait.
- Points: Contient une paire de points en x et y de type float.

3.2.14 Paquet REMOVE_STROKE

Le contenu de la section donnée est une liste de nombres entiers de 32 bits qui représente les index des traits à effacer. Il est très important que cette liste est trié du plus grand nombre au plus petit.

3.2.15 Paquet INSERT_STROKE

Le contenu de la section donnée est un format sérialisé sur toute l'information nécessaire pour la création d'un trait.

Description	**Index**	**RGBa**	**Longueur**	**Largeur**	**Points**
Octet	4	4	4	4	Nombre de points * 8 octets

Tableau 15. La description de la structure de donnée du paquet mode par trait pour l'envoi d'un trait en insertion.

La description de chacune des champs:

- Index: L'index du trait dans la collection de trait.
- RGBa: Contient la couleur avec l'opacité du trait.
- Hauteur: La hauteur du point du crayon de ce trait

- Largeur: La largeur du point du crayon de ce trait.
- Points: Contient une paire de points en x et y de type float.

3.2.16 Paquet SELECTED_STROKE

Le contenu de la section donnée est une liste de nombres entiers de 32 bits qui représente les index des traits qui sont sélectionnées. L'ordre de ces nombres n'a pas d'importance.

3.2.17 Paquet UNSELECTED_STROKE

Le contenu de la section donnée est une liste de nombres entiers de 32 bits qui représente les index des traits qui ne sont plus sélectionnées. L'ordre de ces nombres n'a pas d'importance.

3.2.18 Paquet REPLACE_STROKE

Le contenu de la section donnée est un format sérialisé sur toute l'information nécessaire pour la création d'un trait.

Description	**Index**	**RGBa**	**Longueur**	**Largeur**	**Points**
Octet	4	4	4	4	Nombre de points * 8 octets

Tableau 16. La description de la structure de donnée du paquet mode par trait pour l'envoi d'un trait en insertion.

La description de chacune des champs:

- Index: L'index du trait dans la collection de trait.
- RGBa: Contient la couleur avec l'opacité du trait.
- Hauteur: La hauteur du point du crayon de ce trait
- Largeur: La largeur du point du crayon de ce trait.
- Points: Contient une paire de points en x et y de type float.

3.2.19 Paquet SEND_POINTS

Ce paquet contient une liste de tous les points modifiés par un traits de crayon ou efface dans le mode par pixels. De plus, il contient la taille du crayon à appliquer ainsi que la couleur.

Description	**Taille du crayon**	**RGBa**	**Points**
Octet	4	4	Taille de la liste * 2 coordonnées * 4

Tableau 17. La description de la structure de donnée du paquet mode par pixel pour l'envoi de pixels

3.2.20 Paquet CREATE_SELECTION

Ce paquet contient les information nécessaire pour créer une nouvelle sélection. Ainsi, on peut y retrouver la postion (X,Y) du coin supérieur gauche, sa longueur, sa hauteur et son angle. Il est à noter que l'angle à la création sera toujours 0. À la réception de ce paquet, le client va créer un sélection et insérer un tableau de pixels dans celle-ci, de plus, les pixels qui ont été extraits dans la sélection seront enlevés du canevas principal.

Description	X	Y	Longueur	Largeur	Angle
Octet	4	4	4	4	4

Tableau 18. La description de la structure de donnée du paquet mode par pixel pour l'envoi d'une sélection

3.2.21 Paquet REMOVE_SELECTION

Ce paquet est vide, mais il permet d'indiquer au clients de réinsérer le tableau de pixels contenu dans la sélection du client qui a envoyé ce paquet dans le canevas principal.

3.2.22 Paquet UPDATE_SELECTION

Ce paquet contient les informations nécessaire à un client pour mettre à jour une sélection qui a été soit: déplacée, mise à l'échelle ou tournée à un certain angle.

Description	X	Y	Longueur	Largeur	Angle
Octet	4	4	4	4	4

Tableau 19. La description de la structure de donnée du paquet mode par pixel pour l'envoi d'une sélection

3.2.23 Paquet APPLY_FILTER_SELECTION

Ce paquet contient le nom du filtre à appliquer sur une sélection.

Description	Nom du filtre
Octet	Taille du nom

Tableau 20. La description de la structure de donnée du paquet mode par pixel pour l'application d'un filtre

3.2.24 Paquet RESIZE_CANVAS

Ce paquet contient les nouvelles dimensions du canevas, ce paquet est envoyé lorsqu'un client a terminé de redimensionner le canevas.

Description	Longueur	Largeur
Octet	8	8

Tableau 21. La description de la structure de donnée du paquet pour redimensionner le canevas

3.2.25 Paquet RESET_CANVAS

Ce paquet est vide, mais il permet d'indiquer au clients de soit enlever tous les Stroke pour le mode par traits, ou de réinitialiser le tableau de pixels à blanc pour le mode par pixels.

3.2.26 Paquet BUCKET_FILL

Ce paquet contient la position de départ pour l'opération de remplissage par région par pixels. Il contient aussi la nouvelle couleur que l'on souhaite appliquer.

Description	X	Y	RGBa
Octet	4	4	4

Tableau 22. La description de la structure de donnée du paquet mode par pixel pour le remplissage par région

3.2.27 Paquet UPDATE_DATABASE

Le contenu est vide de ce paquet. Il est seulement utilisé afin de notifier un client de sauvegarder le contenu actuel du canvas dans la base de donnée.

3.2.28 Paquet LOBBY_DISCONNECT

Le contenu est vide de ce paquet. Il est seulement utilisé par les client pour signaler au serveur qu'il est sorti de la session de dessin.

3.2.29 Paquet LOBBY_FREEZE_STATE

Le paquet va contenir un nombre naturel de 32 bits et il est seulement envoyé par le serveur. Il y a deux types de valeurs possibles:

- 0: La session n'est pas en état d'attente. Les utilisateurs de la session peuvent dessiner durant ce temps.
- 1: La session est en état d'attente. Les utilisateurs de la session ne peuvent pas dessiner durant ce temps.

3.2.30 Paquet FETCH_DATABASE

Le contenu est vide de ce paquet. Si le serveur envoie ce paquet vers un utilisateur, ce dernier va chercher dans la base de données l'état actuel du canvas. L'utilisateur va ensuite envoyer ce paquet au serveur pour confirmer qu'il a fini d'avoir les données de la base de données.

3.2.31 Paquet CREATE_ACCOUNT

Le contenu de ce paquet est une chaîne de caractères qui va contenir le mot de passe de l'utilisateur. Le format devra respecter le standard UTF-8.

3.2.32 Paquet LOBBY_INFO_CHANGE

Le contenu de ce paquet est une chaîne de caractères qui devra respecter le format JSON. Il est nécessaire d'avoir les informations suivantes:

Nom	Valeur possibles	Description
Filename	Chaîne de caractères du nom de l'image.	Le nom de l'image.
Owner	Chaîne de caractères du nom du créateur de l'image.	Le créateur de l'image.
Protection	Chaîne de caractère possibles pour la protection: ● public ● private ● protected	Le type de protection utilisé par le créateur.
Passcode	Chaîne de caractères	Le mot de passe pour la protection

		protected. Ce champ est obligatoire.

Tableau 23. Structure du JSON contenant les informations du lobby que l'on souhaite modifier.

3.2.33 Paquet LOBBY_PLAYER_KICK

Le contenu de ce paquet est une chaîne de caractères qui va contenir un message de la justification de l'expulsion. Il est seulement utilisé par le serveur pour expulser un utilisateur de sa session.

3.2.34 Paquet RESIZE_CANVAS_INT

Ce paquet contient les nouvelles dimensions du canevas, ce paquet est envoyé lorsqu'un client a terminé de redimensionner le canevas. Ce paquet diffère du paquet RESIZE_CANVAS, car les données sont des integer au lieu d'être des doubles.

Description	**Longueur**	**Largeur**
Octet	4	4

Tableau 24. La description de la structure de donnée du paquet pour redimensionner le canevas

4. Détection d'erreurs de transmission

Le protocole TCP/IP se chargera de toutes les pertes de données lors du transport des paquets.

Le serveur doit en tout temps valider que le paquet soit conforme à la structure imposée. Si le paquet n'est pas acceptable, le serveur doit le rejeter.

5. Perte d'information - délais d'attente et tentatives

Le serveur doit en tout temps mettre un délai d'attente lorsque le client souhaite s'authentifier. Ce délai sera d'une seconde. Si le client n'a pas pu envoyer un paquet durant ce temps, sa connexion sera fermée. La raison d'une telle implémentation est de ne pas laisser d'un utilisateur mal intentionné de bloquer la réception de nouvelles connexions.

En ce qui concerne des clients déjà connectés, il n'y a pas de temps limite d'inactivité.

Il n'y a pas de limite de tentative qu'un client peut faire pour se connecter au serveur.

6. Authentification

L'authentification d'un compte utilisateur se fait avec un paquet ayant un type id de valeur 3, soit celle qui représente la demande de connexion. Lorsqu'un utilisateur se connecte au serveur, il doit envoyer directement ce paquet avec son mot de passe contenu dans la section donné. Le serveur devra alors vérifier dans la base de données si cet utilisateur existe et que le mot de passe est identique. Ensuite, le serveur envoie un paquet qui informe l'utilisateur de l'état de l'authentification. Enfin, si l'authentification a échoué, le serveur ferme la connexion avec ce client.