Documentation du protocole xXGonzoChat420Xx

Principes de base

Le protocole de communication TCP/UDP xXGonzoChat420Xx est un protocole avant-gardiste qui a comme objectif de révolutionner la communication entre les habitants de la Terre. Il supporte actuellement la communication avec des ordinateurs en réseau local. Il permet de **connecter un ordinateur à plusieurs autres en UDP dans un chat global, ou de connecter un ordinateur à un autre en TCP (messages cryptés à l'aide d'une clé AES 128bits).**

Toutes les communications sont envoyées en UTF-8. Les chiffres sont également représentés en UTF-8. **Les communications UDP entrantes sont toujours écoutées sur le port 42069**, tandis que **les communications TCP entrantes se font sur le port 42070.** Les utilisateurs sont identifiés à l'aide de leur adresse IP.

Toutes les communications **contiennent un entête directement après l'entête TCP/UDP**. L'entête va toujours comme suit:

GONZO_*COMMANDE*

Où *COMMANDE* représente une commande dans la liste des commandes. Le message (si applicable) se trouve entre la deuxième série des caractères **{** et **}**. Si la commande ne nécessite pas de message, il faut tout de même envoyer les caractères **{** et **}** à la fin de l'entête.

Le format d'une communication sera donc toujours sous cette forme:

GONZO_*COMMANDE*{*message*}

Le message peut contenir les caractères { ou }. Le message commence alors au premier caractère { trouvé et se termine au dernier caractère } trouvé (le client peut utiliser la regex {(.*)} pour trouver le message). Le message devrait avoir une longueur d'environ 200 caractères maximum, pour ne pas dépasser la taille maximale d'un datagramme UDP.

Liste des commandes

Format de la commande	Description de la commande	Note
`GONZO_CYA_NERDS{}`	Commande envoyée pour signifier que l'utilisateur quitte le programme de chat. Lors de la réception de la commande, le client devrait supprimer la source de sa liste de clients connectés.	Cette commande doit être envoyée en UDP en *broadcast*.
`GONZO_IM_THERE {`*`Username`*`}`	Commande qui sert à signifier la présence du client aux autres clients et, par le fait même, envoyer son *username* au client. Ceci redémarre le temps de vie de l'utilisateur chez le client, qui est de 12 secondes. Après 12 secondes sans nouvelle réponse, on peut considérer que l'utilisateur a quitté.	Cette commande est envoyée en UDP directement à l'IP qui a envoyé une commande `GONZO_WHOS_THERE`. S'il n'y a aucun *username*, la communication est ignorée.
`GONZO_INITIATE_SECRET_ COMMUNICATION{`*`cléAES128bits`*`}`	Commande envoyée à un utilisateur pour amorcer une communication privée. Lors de la réception de la commande, le client devrait stocker la *`cléAES128bits`*.	Cette commande doit être envoyée en TCP à l'IP de l'utilisateur(port 42070).
`GONZO_SEND_SECRET_MEME {`*`messageCrypté`*`}`	Commande envoyée pour envoyer un message privé à un utilisateur.	Cette commande doit être envoyée en TCP à l'utilisateur concerné et le message doit être crypté avec la clé AES 128 bits préalablement reçue ou générée.
`GONZO_SPAM_GROUP_CHAT {`*`message`*`}`	Commande envoyée pour partager un message dans le chat global. Lors de la réception de la commande, le client devrait afficher le message dans le chat global.	Cette commande doit être envoyée en UDP individuellement à chaque utilisateur connecté.
`GONZO_TERMINATE_SECRET _COMMUNICATION{}`	Commande envoyée pour signifier que l'utilisateur termine sa communication privée avec l'utilisateur de destination.	Cette commande doit être envoyée en TCP à l'utilisateur concerné. La connexion TCP entre les deux clients devra par la suite être fermée.
`GONZO_WHOS_THERE {`*`sonUsername`*`}`	Commande envoyée pour avoir la liste des utilisateurs connectés. Lors de la réception de cette commande, le client doit envoyer à l'IP qui a envoyé cette commande la commande `GONZO_IM_THERE`. *sonUsername* ne sert qu'à identifier les `GONZO_WHOS_THERE{}` qui proviennent de soi-même, au cas où.	Cette commande est envoyée en UDP en broadcast sur le réseau.

Utilisations typiques

Considérons client1, client2 et client3. Ce cas d'utilisation décrit les commandes envoyées quand client1 se connecte, puis client2, puis client3, et quand client2 envoie un message et quand client1 se déconnecte. Dans cet exemple, toutes les communications sont en UDP. Prendre note que le programme doit également envoyer la commande `GONZO_WHOS_THERE{}` une fois par 5 secondes pour corriger la liste d'utilisateurs connectés (si un client avait perdu la connexion).

client1 -> `GONZO_WHOS_THERE{}` <- Envoyé en broadcast. Aucune réponse puisqu'il est le premier client connecté.

client2 -> `GONZO_WHOS_THERE{}` <-Envoyé en broadcast.

client1 -> `GONZO_IM_THERE{client1}` <- envoyé à client2. client2 vérifie que son nom d'utilisateur n'est pas client1. s'il l'est, il demande à l'utilisateur d'entrer un autre nom d'utilisateur. client2 ajoute client1 à sa liste de clients connectés.

(après 2 secondes)

client2 -> `GONZO_IM_THERE{client2}` <- envoyé à client1. client1 ajoute client2 à sa liste de clients connectés.

client3 -> `GONZO_WHOS_THERE{}` <-Envoyé en broadcast.

client1 -> `GONZO_IM_THERE{client1}` <- envoyé à client3. Même vérification qu'à la 3e commande.

client2 -> `GONZO_IM_THERE{client2}` <- envoyé à client3. Même vérification qu'à la 3e commande.

(après 2 secondes)

client3 -> `GONZO_IM_THERE{client3}` <- envoyé à client1. client1 ajoute client3 à sa liste de clients connectés.

client3 -> `GONZO_IM_THERE{client3}` <- envoyé à client2. client2 ajoute client3 à sa liste de clients connectés.

client2 -> `GONZO_SPAM_GROUP_CHAT{test hihihihi}` <- envoyé à client1.

client2 -> `GONZO_SPAM_GROUP_CHAT{test hihihihi}` <- envoyé à client3.

client1 -> `GONZO_CYA_NERDS{}` <- Envoyé en broadcast. client2 et client3 enlèvent client1 de leur liste de clients connectés.

Considérons maintenant qu'il ne reste que client2 et client3. Ce cas d'utilisation décrit client2 qui commence une communication privée avec client3, client2 qui envoie un message privé à client3 puis client3 qui termine la communication privée. Toutes ces communications sont envoyées en TCP.

client2 -> `GONZO_INITIATE_SECRET_COMMUNICATION{`*`cléAES128bitsGénérée`*`}` <- Envoyé à client3. client2 stocke la clé, client3 aussi.

client2 -> `GONZO_SEND_SECRET_MEME{un meme crypté}` <- Envoyé à client3. "un meme crypté" est crypté avec la clé AES préalablement envoyée.

client3 -> `GONZO_TERMINATE_SECRET_COMMUNICATION{}` <- Envoyé à client2. La connexion TCP se termine et la clé peut être détruite.