T. CALONNE
T. JUILLARD

30 mars 2020

PROTOCOLE TOTO

Statut de ce mémo

Cette RFC spécifie un protocole de service de chat utilisant les normes TCP. Ce protocole est Open source.

Description

TOTO est un protocole simple utilisé pour réaliser un service de discussion et d'échange de fichiers. Le document décrit le protocole et ses types de paquets. Ce document explique les raisons de certaines décisions de conception. Il a été mis en oeuvre afin qu'il puisse être utilisé pour le service de chat. La première étape est la connexion, puis une fois connecté via une base de donnée ou sans, un utilisateur pourra envoyer des messages aux autres utilisateurs connectés.

Remerciements

Le protocole TOTO a été conçu par Thomas Calonne et Thomas Juillard dans le cadre d'un projet d'étude sur la programmation réseau. Nous tenons à remercier les enseignants Arnaud Carayol, Etienne Duris et Yann Feugeu de nous donner l'opportunité de réaliser le projet ChatHack ainsi que leur investissement.

Avant propos

Dans le cas où une trame reçue par un serveur ou un client ne serait pas conforme, la trame ne sera pas traitée. Le format Big-endian sera utilisé à chaque fois que l'on parlera d'Integer ou de Long dans cette RFC.

Aperçu du protocole

1. Protocole de connexion initiale

Le protocole TOTO propose deux modes d'authentification qui sont avec mot de passe et sans mot de passe. Cette étape est indispensable pour envoyer et recevoir des messages.

a. Connexion avec mot de passe

Le serveur a accès à une base de données de pseudonymes et de mots de passe.

i. Demande de connexion d'un client vers le serveur

1 byte	1 Integer	String	1 Integer	String
Opcode	SizeName	Login	SizePwd	Pwd

Le protocole avec mot de passe permet de se connecter à la base de données. Il faut donc renseigner:

- Opcode : contient la valeur 13
- SizeLogin : contient la taille de Name
- Login : le contenu de Login est encodé en UTF-8 (Byte * SizeLogin)
- SizePwd : contient la taille de Pwd
- Pwd : le contenu de Pwd est encodé en UTF-8 (Byte * SizePwd)

ii. Réponse du serveur

1 byte	1 byte
Opcode	Ack

- Opcode : contient la valeur 14
- Ack : 0 si la connexion a été établie, 1 sinon.

Cas de refus de connexion :

- Le nom d'utilisateur est déjà pris
- Nom d'utilisateur, mot de passe incorrect

b. **C**onnexion sans mot de passe

i. **D**emande de connexion d'un client vers le serveur

La trame suivante est envoyée au serveur :

1 byte	1 Integer	String
Opcode	SizeLogin	Login

Le protocole sans mot de passe ne permet pas de se connecter à la base de données. Il faut donc renseigner :

- Opcode : contient la valeur 2.
- SizeLogin : contient la taille de Login encodé en UTF-8
- Login : Login est encodé en UTF-8 (Byte * SizeLogin)

ii. **R**éponse du serveur

1 byte	1 Integer
Opcode	Ack

- Opcode : contient la valeur 1.
- Ack : 0 si la connexion a été établie, 1 sinon.

Le serveur envoie un paquet contenant un seul Byte. Le serveur met l'Ack à 0 si la connexion a été établie, 1 sinon.

Cas de refus de connexion :

- Le nom d’utilisateur est déjà pris par un utilisateur de la base de données.
- Le nom d’utilisateur est déjà pris par un autre utilisateur connecté.

2. Envoi d'un message public

Les trames suivantes seront utilisées lorsqu'un client voudra parler dans le chat public à tous les utilisateurs connectés.

a. Client vers serveur

1 byte	1 Integer	String
Opcode	SizeMsg	Msg

- Opcode : contient la valeur 3.
- SizeMsg : contient la taille de Msg
- Msg : le message est encodé en UTF-8 (Byte * SizeMsg)

b. Le serveur envoie un message

1 byte	1 Integer	String	1 Integer	String
Opcode	SizeLogin	Login	SizeMsg	Msg

- Opcode : contient la valeur 4.
- SizeLogin : contient la taille de Login encodé en UTF-8
- Login : l'identifiant de l'expéditeur est encodé en UTF-8 (Byte * SizeLogin)
- SizeMsg : contient la taille de Msg
- Msg : le message est encodé en UTF-8 (Byte * SizeMsg)

3. Demande de connexion privée

a. Client 1 vers serveur

Le Exp envoie une trame pour pouvoir dialoguer avec une autre personne Dest au serveur.
La trame suivante est envoyée :

1 byte	1 Integer	1 String	1 Long
Opcode	SizeDest	Dest	Token

- Opcode : contient la valeur 5.
- SizeDest : taille de Name2
- Dest : nom du client 2 encodé en UTF-8 (Byte * SizeName2)
- Token : le token que devra utiliser Client1 pour s'authentifier au serveur Dest.

b. Le serveur interroge le destinataire (client 2), pour savoir s'il veut bien dialoguer avec le client 1

i. Le client existe

La trame envoyée au client 2 depuis le serveur :

1 byte	1 Integer	String	Long
Opcode	SizeExp	Exp	Token

- Opcode : contient la valeur 6.
- SizeExp : la taille de Exp correspond au client1
- Exp : le nom du client 1 est encodé en UTF-8 (Byte * SizeExp)
- Token : token que devra utiliser Client1 pour s'authentifier au serveur Client2

ii. Le client n'existe pas

Dans le cas où le client n'existe pas, la trame suivante sera renvoyée au client 1 :

1 byte	1 byte	1 integer	String
Opcode	WichError	SizeName	Name

- Opcode : Contient la valeur 15
- WichError : Contient un byte permettant de déterminer de quelle Erreur il s'agit : si c'est un 1 le destinataire n'existe pas, si c'est un 2 le client ne peut pas communiquer avec lui-même (donc deux erreurs)
- SizeName : Correspond à la taille de Name
- Name : Le nom encodé en UTF-8 du client (Byte * SizeName)

c. **L**e destinataire (client 2) répond au serveur

i. **L**e destinataire accepte d'échanger en privé avec le client 1

Le client 2 devra créer un serveur sur l'un de ses ports libres et donner son adresse IP ainsi que le port sur lequel le serveur a été créé. Le client se servira de ce serveur et de ce port pour toutes les demandes de connexions privées qu'il acceptera. Pour différencier les utilisateurs qui utiliseront ce serveur, chaque utilisateur possédera un token qui permettra de s'identifier et donc aussi régler certaines questions de sécurité. Une fois le token obtenu, les deux clients pourront dialoguer ensemble sans demander de nouveau un token.

Voici la trame de ce que le client 2 enverra au serveur:

1 byte	1 Integer	String	1 Integer
Opcode	SizeDest	Dest	portExp

- Opcode : contient la valeur 7.
- SizeDest : contient la taille de Name1 encodé en UTF-8 correspond au destinataire
- Dest: contient le nom du client 1 encodé en UTF-8 (Byte * SizeDest)
- PortExp : contient le port de connexion de l'expéditeur

ii. Le destinataire ne veut pas échanger en privé avec le client 1

Si le client 2 n'accepte pas la connexion, il devra envoyer au client 1 via le serveur son refus de connexion privée.

La trame suivante est envoyée au serveur

1 byte	1 Integer	String
Opcode	SizeDest	Dest

- Opcode : contient la valeur 8.
- SizeDest : contient la taille de Name1 encodé en UTF-8 correspond au destinataire
- Dest: contient le nom du client 1 encodé en UTF-8 (Byte * SizeDest)

d. Le serveur transfert au client 1 la réponse à la demande de chat privé du client 2

i. Réponse positive du serveur à la demande de chat privé du client 2

1 byte	1 Integer	String	1 Integer	1 Integer ou 2 Long	1 Integer
Opcode	SizeName2	Name2	WichIp	IpAdrCli2	PrtCli2

- Opcode : contient la valeur 9
- Name2 : contient le nom du client 2 encodé en UTF-8 (Byte * SizeName)
- SizeName2 : contient la taille de Name2 encodé en UTF-8 correspond au client2
- wichIp : permet d'identifier s'il s'agit d'une adresse en IPv4 ou IPv6 en renvoyant 4 ou 16 octets
- IpAdrCli2 : contient l'adresse IP du client2. Dans le cas où l'IP est en IPv4 on utilisera un Integer, si l'adresse IP est en IPv6 alors on utilisera 2 Long
- PrtCli2 : contient le port de connexion du client 2

Le client 2 devra donc créer un serveur sur l'un de ses ports disponibles et ce sera ce port-là qu'il enverra au client. Si le client 2 possède déjà un serveur, il donnera juste un identifiant unique au client 1 pour discuter. Client 2 aura donc un token différent pour chacune des ses connexions privées, ce qui permettra de sécuriser ses échanges.

ii. **R**éponse négative du serveur à la demande de chat privé du client 2

La trame suivante est envoyée du serveur au client 1 :

1 byte	1 Integer	String
Opcode	SizeName2	Name2

- Opcode : contient la valeur 10
- SizeName2 : contient la taille de Name2, qui correspond au nom du client2
- Name2 : contient le nom du client 2 encodé en UTF-8 (Byte * SizeName2)

4. Connexion privée entre X et Y

a. Authentification de X auprès de Y

X envoie directement un message à Y dès qu'il a reçu son acceptation pour la demande de chat privé. En s'identifiant, Y saura qui est X.

Une fois que le X a bien reçu la réponse positive, il va envoyer une trame au serveur de Y pour qu'il puisse envoyer un message en premier.

Voici la trame qui est envoyée :

1 byte	Long
Opcode	token

- Opcode : contient la valeur 11
- token : Le token d'identification

b. Échange privé de messages entre deux clients X et Y

Une fois que l'étape d'autorisation requise pour les échanges privés est terminée, les clients communiqueront directement entre eux. Ce sont les mêmes trames, que le client soit serveur ou non.

Voici la trame qui sera envoyée par le client X au client Y :
C'est la même trame qui est utilisé lors d'un envoi de message publique. c'est un rappel de la trame avec l'opecode 3.

1 byte	1 Int	1 String
Opcode	SizeMsg	Msg

- Opcode : contient la valeur 3
- SizeMsg : contient la taille de Msg
- Msg : message encodé en UTF-8 (Byte * SizeMsg)

c. **E**nvoi d’un fichier

Le Protocole TOTO permet également de réaliser des envois de fichiers. Nous allons l’utiliser dans le projet ChatHack. Admettons que le client 1 souhaite envoyer un fichier au client 2. Pour cela, il faut que le client 1 et le client 2 aient validé l’étape de messages privés (voir 3. de la RFC).

Voici la trame qui sera envoyée par le client 1 vers le client 2 lors d’un envoi de fichier :

1 byte	1 Integer	String	1 Long	bytes
Opcode	SizeName	Name	SizeFile	File

- Opcode : contient la valeur 12
- SizeName : contient la taille de Name correspond au nom du fichier
- Name : contient le nom du fichier encodé en UTF-8 (Byte * SizeName)
- SizeFile : contient la taille du fichier
- File : Fichier encodé en UTF-8 (bytes * SizeMsg)