Réseaux et Systèmes Avancés

Rapport

MyTwitter

Auteurs :
Laury DE DONATO
Clément JOLY

Enseignants référents :
Rémi BADONNEL
Isabelle CHRISMENT

Mai 2019

Introduction

Dans le cadre de ce projet de RSA, nous réalisons une application client/serveur présentant des fonctionnalités proches de Twitter[1].

Ce document présente les caractéristiques de notre travail et une capture des échanges client/serveur avec l'outil Wireshark[2]. Il se termine par la spécification du protocole utilisé pour la communication client/serveur.

Caractéristiques de notre projet

Techniques

Nous avons utilisé une base de données et le format de données JSON pour la communication client/serveur. Ceci a été l'occasion d'apprendre à utiliser de manière plus approfondie des bibliothèques spécifiques en C comme `sqlite3` ou `json-c`. En effet, les projets précédents n'autorisaient pas forcément l'utilisation de bibliothèques externes.

Notre projet présente les fonctionnalités facultatives suivantes :

- Sauvegarde des tweets (appelés par la suite gazouillis) et des comptes utilisateurs (ils ne sont pas perdu au redémarrage du serveur) ;
- Fonction « retweet » : un utilisateur peut relayer un gazouilli, éventuellement plusieurs fois. Il apparaît alors pour ses propres abonnés, inséré parmi les autres gazouillis dans l'ordre chronologique.

Enfin, dans un souci d'optimisation de la charge pour le serveur, l'ensemble des gazouillis récents sont envoyés au client lorsque celui-ci le lui demande. Le serveur fournit le nombre de tweet demandé par le client, ce qui simplifie l'implémentation côté client. Par ailleurs, le client n'a ainsi rien à stocker.

Des précautions ont été prises afin d'éviter les injections SQL. Cependant, les mots de passe sont stockés et transmis en clair, comme autorisé par le sujet.

Gestion de projet

Nous avons établi la spécification avant de commencer à écrire le client ou le serveur, comme en atteste les commits du fichier `spec_protocol.md`. La spécification comprend les primitives de service nécessaires à la réalisation de l'application ainsi que la liste et le format des Application Data Units (APDUs), c'est-à-dire la liste et le format des messages qui vont transiter entre le client et le serveur au niveau applicatif.

Écrire en premier lieu la spécification avant de commencer à proprement parler l'implémentation nous a permis par la suite de travailler de manière séparée sur le client et sur le serveur, mais aussi sur des environnements de développement différents. Ce travail préalable nous a grandement facilité le développement de l'application.

Pour les ajustements éventuels qui se sont avérés nécessaires, la personne souhaitant effectuer une modification de la spécification la proposait dans une Merge-Request sur l'instance Gitlab de l'école. Elle était alors discutée puis acceptée ou rejetée.

[1] https://twitter.com/
[2] https://www.wireshark.org/

Capture Wireshark

Nous avons effectué le schéma suivant et capturé les paquets à l'aide Wireshark (dans la fichier `capture_wireshark.pcapng`) :

1. Le client n°1 lance l'application et se connecte en IPv6 (trame 1). Le serveur accepte la connexion (trames 2 et 3).
2. Le client n°1 crée un compte pour l'utilisateur « @alice » avec le mot de passe « alice » (trame 4). Le serveur traite la requête (trame 6).
3. Le client n°1 se connecte avec l'utilisateur « @alice » et le mot de passe « alice » (trame 10). Le serveur traite la requête (trame 11).
4. Le client n°2 lance l'application et se connecte en IPv4 (trame 15). Le serveur accepte la connexion (trames 16 et 17).
5. Le client n°2 crée un compte pour l'utilisateur « @bob » avec le mot de passe « bob » (trame 18). Le serveur traite la requête (trame 20).
6. Le client n°2 se connecte avec l'utilisateur « @bob » et le mot de passe « bob » (trame 24). Le serveur traite la requête (trame 25).
7. Le client n°1 envoie un gazouilli dont le contenu est « Je teste #MyTwitter » (trame 29). Le serveur traite la requête (trame 31).
8. Le client n°2 envoie un gazouilli dont le contenu est « J'aime #MyTwitter » (trame 35). Le serveur traite la requête (trame 36).
9. Le client n°2 demande à suivre le tag « MyTwitter » (trame 40). Le serveur traite la requête et lui indique que le tag est bien suivi (trame 41).
10. Le client n°2 demande à relayer un gazouilli. Pour cela, le logiciel client récupère d'abord les gazouillis relayables (trame 45 et 46) et les affiche. Le client n°2 choisit de relayer le gazouilli dont l'id est 1 (trame 50). Le serveur traite la requête et lui indique que le gazouilli est bien relayé (trame 51).
11. Le client n°2 demande l'affichage des gazouillis auxquels il est abonné (trame 55). Le serveur traite la requête et lui renvoie les gazouillis correspondants (trame 56).
12. Le client n°2 demande à se déconnecter du serveur (trame 60). Le serveur traite la requête (trame 61).
13. Le client n°1 ferme l'application. Il ferme donc la connexion TCP (trames 71 et 72). Le serveur répond et ferme la connexion (trames 76 et 77).
14. Le client n°2 ferme l'application. Il ferme donc la connexion TCP (trames 78 et 79). Le serveur répond et ferme la connexion (trames 83 et 84).

Nous avons vérifié que les APDUs soient conformes à la spécification. Dans un souci de longueur, nous n'avons pas fait apparaître sur la capture toutes les fonctionnalités proposées par MyTwitter.

Spécification du protocole de communication client/serveur

Version: 2.0

Introduction

Ce document décrit le protocole de communication entre un client et un serveur de ce service.

Définitions

Dans cette partie nous définissons tous les termes utiles à notre application.

Gazouilli

Message publié par un utilisateur. C'est un message textuel d'un maximum de 140 caractères. Le message peut référencer une thématique particulière, mais pas un autre utilisateur. Les thématiques sont comptées dans la limite des 140 caractères. Les caractères seront encodés en ASCII.

Le nom d'auteur du gazouilli ne fait pas partie de la limite des 140 caractères d'un gazouilli.

Thématique

Mot d'un message commençant par un `#`. Ces thématiques doivent servir au regroupement des messages ayant un thème commun. Deux thématiques sont séparées par un espace. Les thématiques sont séparées du corps du message par un espace. Les thématiques peuvent se trouver à n'importe quelle position dans le message. Deux thématiques identiques n'ayant pas la meme casse seront considérées comme deux thématiques différentes. Ainsi : `#Twitter` et `#twitter` seront considérées comme deux thématiques différentes.

Chaque caractère de la thématique compte pour un caractère du gazouilli.

Relayer un gazouilli

Retransmettre le message d'un autre utilisateur à ses abonnés. La mention `relayé par <nom_relayeur>` est alors indiquée.

Abonné

Utilisateur qui suit un autre utilisateur. Un abonné recevra tous les gazouillis postés par les utilisateurs qu'il suit.

Fonctionnalités :

- Fonctionnalité 1 : Une personne doit pouvoir s'inscrire sur MyTwitter, elle devient alors un utilisateur.
- Fonctionnalité 2 : Un utilisateur doit pouvoir se connecter sur MyTwitter.
- Fonctionnalité 3 : Un utilisateur connecté doit pouvoir envoyer un gazouilli.
- Fonctionnalité 4 : Un utilisateur connecté doit pouvoir demander de suivre un autre utilisateur.
- Fonctionnalité 5 : Un utilisateur connecté doit pouvoir demander de suivre une thématique donnée.
- Fonctionnalité 6 : Un utilisateur connecté doit pouvoir afficher la liste des utilisateurs qu'il suit
- Fonctionnalité 7 : Un utilisateur connecté doit pouvoir afficher la liste des thématiques qu'il suit
- Fonctionnalité 8 : Un utilisateur connecté doit pouvoir afficher la lsite des utilisateurs qui le suivent
- Fonctionnalité 9 : Un utilisateur connecté doit pouvoir recevoir à tout moment les messages liés aux utilisateurs et/ou aux thématiques qu'il suit.
- Fonctionnalité 10 : Un utilisateur connecté doit pouvoir relayer un message d'un autre utilisateur qu'il a reçu à ses propres abonnés
- Fonctionnalité 11 : Un utilisateur connecté doit pouvoir se déconnecter

Forme des APDU

Pour coder les APDU, nous avons tout d'abord envisagé d'utiliser le format binaire msgpack[3]. Il présente l'avantage d'être compact, tout en étant relativement facile à déboguer, puisqu'il peut être converti dans les deux sens en JSON[4].

Cependant, après réflexion, nous avons décidé de travailler directement avec un format de données JSON, dans un souci de simplicité de débuggage. C'est un format moins compact mais qui peut être débuggé directement (sans avoir à être converti).

La spécification des requêtes est inspirée du JSON-RPC[5].

La taille maximum d'une APDU est de **6000 octets**. En l'absence de paramètres utilisateurs particuliers, le port par défaut est le port **1234**.

Format du dialogue client/serveur

Requêtes Une requête doit avoir le format suivant :

Champs	Rôle
`request`	Identifie la méthode appelée
`params`	Paramètres de la méthode, nommés ou non, voir plus loin
`id`	Champs généré par le client, unique pour une requête. Il permet au client d'associer la réponse à la requête initiale

Une réponse correcte à une requête doit avoir le format suivant :

Champs	Rôle
`result`	Nom de la méthode réalisée avec succès
`params`	Objet contenant les informations de la réponse
`id`	Comme précédemment

Une réponse d'erreur à une requête doit avoir le format suivant :

Champs	Rôle
`error`	Nom de la méthode réalisée avec erreur
`error_code`	Numéro de l'erreur, est toujours un entier. Il est unique pour une méthode donnée
`id`	Comme précédemment

Si le nombre de paramètre est connu, on pourra nommer directement les paramètres.

```
{"request": "nom_methode", "params": {"param1": "val1", "param2": "val2"}, "id": 3}
```

Si le nombre de paramètre est inconnu, on pourra utiliser un tableau de paramètres.

```
{"resquest": "nom_methode", "params": [...], "id": 9}
```

On préférera nommer explicitement les paramètres, dans un soucis de lisibilité.

[3] https://msgpack.org/
[4] http://json.org/
[5] https://www.jsonrpc.org/specification

Réponse sans erreur

```
{"result": "nom_methode", "params": {"param1": "val1", "param2": "val2"}, "id": 3}
```

Réponse d'erreur

```
{ "error": "nom_methode", "error_code": 1234, "id": 3}
```

Notes : - Si la requete possède un format incorrect, un message d'erreur sera renvoyé. Par exemple, si l'id n'est pas fourni par le client, la requête est incorrecte et on produit un réponse d'erreur avec un champs `id` contenant la valeur null - Le `code` de l'erreur doit toujours être associé à la même erreur. On définit : - Le code `10` pour un message dont le format n'est pas correct (par exemple s'il est trop long ou ne respecte pas le format JSON) - Le code `11` pour une erreur interne du côté serveur - Le code `12` pour une erreur interne du côté client - Le code `13` pour une erreur liée à une fonctionnalité non implémentée côté client ou serveur. - Le code `14` pour une erreur liée à un cookie invalide.

Objets généraux

Gazouilli

Voici un exemple de la structure JSON d'un Gazouilli. Elle comporte trois champs : `id` qui est un entier, `content` qui est un String et `tags` qui est un tableau de String. Chaque `id` sera unique.

```
{
    "id": 3,
    "content": "Bonjour #MyTwitter et #HelloWorld !",
    "author": "JeanDupont",
    "retweeter": "AnneONyme",
    "retweet_date": "2019-03-18T17:15:00",
    "list_of_tags": ["MyTwitter", "HelloWorld"],
    "date": "2019-03-18T17:15:00"
}
```

Utilisateur

Voici un exemple de la structure JSON d'un utilisateur. Elle comporte un champs : `username`.

```
{
    "username": "LouisSchmit"
}
```

Méthodes

Création de compte : `create_account`

Sens : Client - Serveur

Paramètres :

Nom	Type	Description
`username`	String	Nom d'utilisateur
`password`	String	Mot de passe

Retour sans erreur : Paramètres : aucun

Retour avec erreur : Valeur des codes d'erreur :

Valeur	Description
1	Nom d'utilisateur déjà pris

Autoriser une connexion : `connect`

Sens : Client - Serveur

Paramètres :

Nom	Type	Description
`username`	String	Nom d'utilisateur
`password`	String	Mot de passe

Retour sans erreur : Paramètres :

Nom	Type	Description
`cookie`	int	Identifiant de connexion unique, valable tout le temps de la connexion

Dans le cas où l'utilisateur est déjà connecté, on définit un nouveau cookie de manière à ne pas bloquer un utilisateur qui aurait mal fermé une session précédente.

Retour avec erreur : Valeur des codes d'erreur :

Valeur	Description
1	Nom d'utilisateur non enregistré
2	Mot de passe incorrect

Envoi d'un gazouilli : `send_gazou`

Sens : Client - Serveur

Paramètres :

Nom	Type	Description
`gazouilli`	Gazouilli	Objet de type Gazouilli
`cookie`	int	Authentifie l'utilisateur

Les champs `id`, `retweeter`, `retweet_date` et `author` seront ignorés puisqu'ils sont définis par le serveur.

Un message sans thématique aura le tableau de tags vide, et ne sera envoyé qu'aux abonnés de l'utilisateur envoyant le gazouilli.

Un même tag présent X fois dans le gazouilli ne provoquera qu'un seul envoi de ce gazouilli pour cette thématique.

Retour sans erreur : Paramètres : aucun

Retour avec erreur : Valeur des codes d'erreur :

Valeur	Description
1	Message comportant un/des caractère(s) non supporté(s)
2	Message trop long

Relayer un gazouilli : `relay_gazou`

Sens : Client - Serveur

Paramètres :

Nom	Type	Description
`id_gazouilli`	entier	Identifiant unique du gazouilli
`retweet_date`	string	Date du relayage, au même format que pour un objet Gazouilli
`cookie`	int	Authentifie l'utilisateur

Note : Un gazouilli peut-être relayé plusieurs fois.

Retour sans erreur : Paramètres : aucun

Retour avec erreur : Valeur des codes d'erreur :

Valeur	Description
1	Id de gazouilli invalide
2	Gazouilli déjà relayé

Demander à suivre un utilisateur : `follow_user`

Sens : Client - Serveur

Paramètres :

Nom	Type	Description
`username`	String	Nom d'utilisateur à suivre
`cookie`	int	Authentifie l'utilisateur

Retour sans erreur : Paramètres : aucun

Retour avec erreur : Valeur des codes d'erreur :

Valeur	Description
1	Nom d'utilisateur inconnu
2	Déjà abonné

Demander à suivre une thématique : `follow_tag`

Sens : Client - Serveur

Paramètres :

Nom	Type	Description
`tag`	String	Nom de la thématique à suivre
`cookie`	int	Authentifie l'utilisateur

Retour sans erreur : Paramètres : aucun

Retour avec erreur : Valeur des codes d'erreur :

Valeur	Description
1	Déjà abonné

Demander à ne plus suivre un utilisateur : `unfollow_user`

Sens : Client - Serveur

Paramètres :

Nom	Type	Description
`username`	String	Nom de l'utilisateur à ne plus suivre
`cookie`	int	Authentifie l'utilisateur

Retour sans erreur : Paramètres : aucun

Retour avec erreur : Valeur des codes d'erreur :

Valeur	Description
1	Non abonné

Demander à ne plus suivre une thématique : `unfollow_tag`

Sens : Client - Serveur

Paramètres :

Nom	Type	Description
`tag`	String	Nom de la thématique à ne plus suivre
`cookie`	int	Authentifie l'utilisateur

Retour sans erreur : Paramètres : aucun

Retour avec erreur : Valeur des codes d'erreur :

Valeur	Description
1	Non abonné

Lister les utilisateurs suivis : `list_followed_users`

Sens : Client - Serveur

Paramètres :

Nom	Type	Description
`cookie`	int	Authentifie l'utilisateur

Retour sans erreur : Paramètres :

Nom	Type	Description
`list_of_users`	Tableau d'utilisateurs	Liste d'objets de type utilisateur qui référence tous les utilisateurs suivis

Retour avec erreur : Valeur des codes d'erreur : aucun, hormis les codes globaux définis plus haut

Lister les thématiques suivies : `list_followed_tags`

Sens : Client - Serveur

Paramètres :

Nom	Type	Description
`cookie`	int	Authentifie l'utilisateur

Retour sans erreur : Paramètres :

Nom	Type	Description
`list_of_tags`	Tableau de String	Liste des thématiques suivies

Retour avec erreur : Valeur des codes d'erreur : aucun, hormis les codes globaux définis plus haut

Lister les abonnés d'un utilisateur : `list_my_followers`

Sens : Client - Serveur

Paramètres :

Nom	Type	Description
`cookie`	int	Authentifie l'utilisateur

Retour sans erreur : Paramètres :

Nom	Type	Description
`list_of_followers`	Tableau d'utilisateurs	Liste d'objets de type utilisateur qui référence tous les utilisateurs auquels on est abonné

Retour avec erreur : Valeur des codes d'erreur : aucun, hormis les codes globaux définis plus haut

Déconnexion du client : `disconnect`

Sens : Client - Serveur

Paramètres :

Nom	Type	Description
`cookie`	int	Authentifie l'utilisateur

Retour sans erreur : Paramètres : aucun

Retour avec erreur : Valeur des codes d'erreur : aucun, hormis les codes globaux définis plus haut

Récupérer les messages destinés à l'utilisateur : `get_gazou`

Sens : Client - Serveur

Information : si un utilisateur receveur du gazouilli est à la fois un abonné de l'utilisateur auteur du gazouilli et abonné d'au moins une thématique contenue dans le gazouilli, le serveur devra veiller à n'envoyer le gazouilli qu'une et une seule fois.

Paramètres :

Nom	Type	Description
`nb_gazou`	int	Nombre d'objets gazouillis à recevoir
`cookie`	int	Authentifie l'utilisateur

On récupère au plus `nb_gazou` gazouillis.

Retour sans erreur : Paramètres :

Nom	Type	Description
`list_of_gazous`	Tableau de gazouillis	Liste d'objets de type gazouilli, ordonnés chronologiquement (le plus récent en premier)

Retour avec erreur : Valeur des codes d'erreur : aucun, hormis les codes globaux définis plus haut