

Jos Zigabe (63271000)
Justine Doutreloux (32431200)

Octobre 2016
LINGI1341

RÉSEAUX INFORMATIQUES

RAPPORT DU PROJET 1

1 Architecture

Notre programme est divisé en 7 parties :

- `receiver` : gère la réception des packets;
- `sender` : gère l'envoi des packets;
- `packet_implem` : implémente les packets et les fonctions nécessaires à leur utilisation;
- `create_socket` : gère la création d'un socket;
- `read_write_loop` : gère la lecture et l'écriture des packets sur `stdin/stdout` ou depuis/vers un fichier;
- `real_address` : gère la résolution du hostname;
- `wait_for_client` : bloque tant qu'un message n'a pas été reçu.

2 Choix d'implémentations

2.1 Contenu et utilité de `TimeStamp`

Nous avons décidé que le champ `timestamp` contiendrait une représentation de l'heure d'envoi du packet. Nous l'obtenons grâce à la fonction `time()`. Ce champ nous est utile pour gérer les timeout. Quand l'heure actuelle diminuée du `timestamp` est supérieure à la valeur de notre `timestamp`, nous renvoyons le packet.

2.2 Choix de la valeur de `timeout`

Nous avons choisi la valeur du `timeout` de façon expérimentale : en utilisant la commande `ping` dans le terminal, nous avons constaté que le temps maximum était de $0.214ms$ donc, nous avons choisi une valeur de `timeout` de `0.5`. Cette valeur vaut le double de la valeur expérimentale car nous voulions être sûrs de ne pas déclencher le `timeout` trop vite.

```

PING 192.168.1.55 (192.168.1.55): 56 data bytes
64 bytes from 192.168.1.55: icmp_seq=0 ttl=64 time=0.065 ms
64 bytes from 192.168.1.55: icmp_seq=1 ttl=64 time=0.071 ms
64 bytes from 192.168.1.55: icmp_seq=2 ttl=64 time=0.174 ms
64 bytes from 192.168.1.55: icmp_seq=3 ttl=64 time=0.087 ms
64 bytes from 192.168.1.55: icmp_seq=4 ttl=64 time=0.154 ms
64 bytes from 192.168.1.55: icmp_seq=5 ttl=64 time=0.161 ms
64 bytes from 192.168.1.55: icmp_seq=6 ttl=64 time=0.214 ms
64 bytes from 192.168.1.55: icmp_seq=7 ttl=64 time=0.065 ms
64 bytes from 192.168.1.55: icmp_seq=8 ttl=64 time=0.135 ms
64 bytes from 192.168.1.55: icmp_seq=9 ttl=64 time=0.151 ms
64 bytes from 192.168.1.55: icmp_seq=10 ttl=64 time=0.142 ms
64 bytes from 192.168.1.55: icmp_seq=11 ttl=64 time=0.132 ms
64 bytes from 192.168.1.55: icmp_seq=12 ttl=64 time=0.138 ms
64 bytes from 192.168.1.55: icmp_seq=13 ttl=64 time=0.129 ms
^C
--- 192.168.1.55 ping statistics ---
14 packets transmitted, 14 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.065/0.130/0.214/0.042 ms

```

3 Efficacité

4 Test unitaires

Nous avons 8 tests unitaires :

- `testPkt_new()` : teste l'initialisation d'un packet;
- `void testPkt_decode()` : teste le décodage d'un packet;
- `void testPkt_encode()` : teste l'encodage d'un packet;
- `void testPkt_getsSets()` : teste les getteurs et les setteurs des packets;
- `void test_real_address()` : teste la fonction `real_address` avec des arguments corrects;
- `void test_real_address1()` : teste la même fonction mais avec des arguments incorrects;
- `void test_create_socket1()` : teste la fonction `create_socket` avec des arguments corrects;
- `void test_create_socket2()` : teste la même fonction mais avec des arguments incorrects.

Notre code passe la plupart de nos tests. Certains assert dans `testPkt_decode()` ne passent pas car certaines fonctionnalités ne sont pas encore implémentées dans notre code.