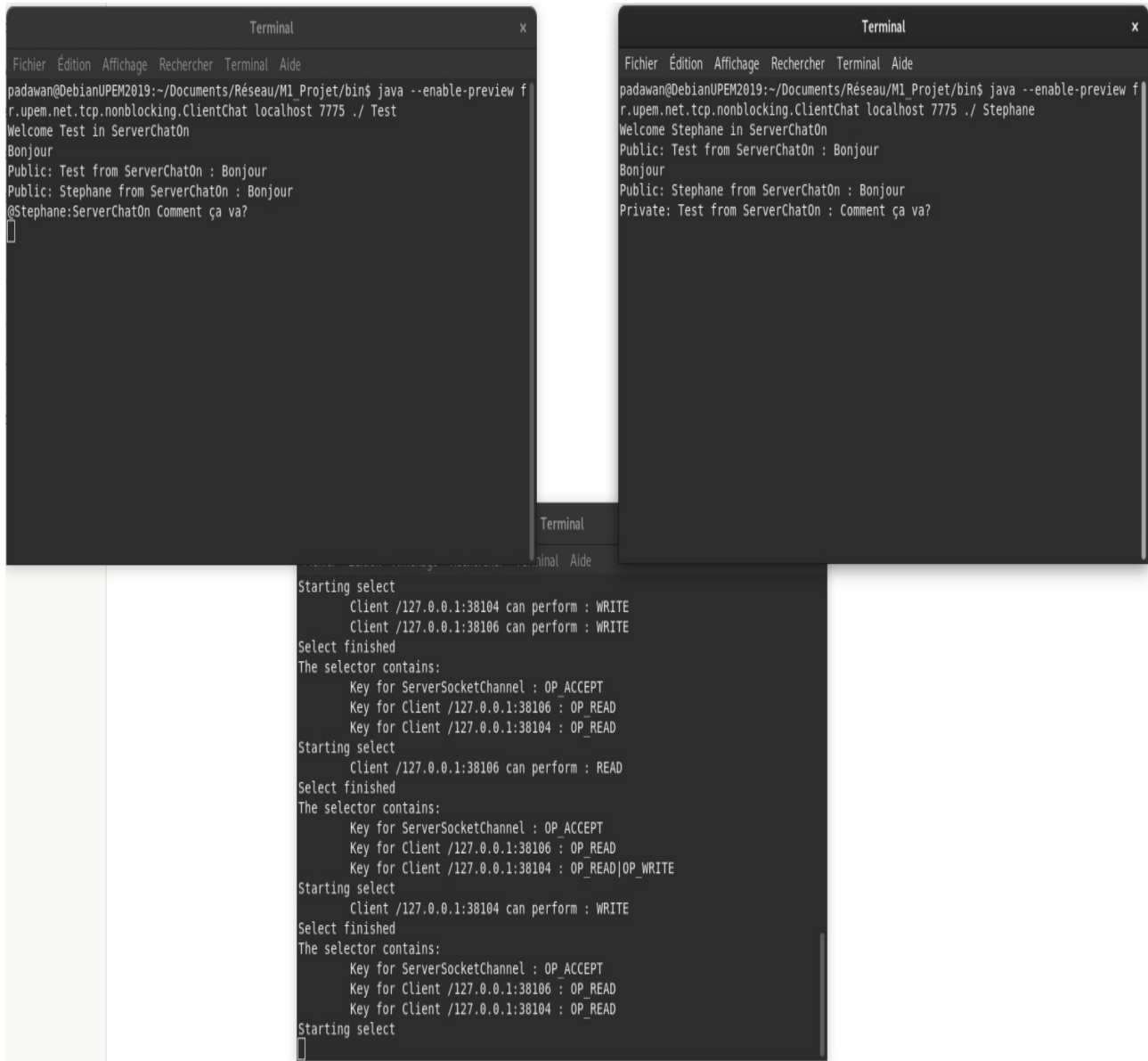


Rapport Réseau



The image displays three terminal windows illustrating a network chat application. The top-left window shows the server's perspective, the top-right window shows one client's perspective, and the bottom window shows the server's internal state using the `select` system call.

Terminal 1 (Server):

```
Fichier Édition Affichage Rechercher Terminal Aide
padawan@DebianUPEM2019:~/Documents/Réseau/M1_Projet/bin$ java --enable-preview fr.upem.net.tcp.nonblocking.ClientChat localhost 7775 ./ Test
Welcome Test in ServerChat0n
Bonjour
Public: Test from ServerChat0n : Bonjour
Public: Stephane from ServerChat0n : Bonjour
@Stephane:ServerChat0n Comment ça va?
```

Terminal 2 (Client):

```
Fichier Édition Affichage Rechercher Terminal Aide
padawan@DebianUPEM2019:~/Documents/Réseau/M1_Projet/bin$ java --enable-preview fr.upem.net.tcp.nonblocking.ClientChat localhost 7775 ./ Stephane
Welcome Stephane in ServerChat0n
Public: Test from ServerChat0n : Bonjour
Bonjour
Public: Stephane from ServerChat0n : Bonjour
Private: Test from ServerChat0n : Comment ça va?
```

Terminal 3 (Server State):

```
Starting select
Client /127.0.0.1:38104 can perform : WRITE
Client /127.0.0.1:38106 can perform : WRITE
Select finished
The selector contains:
Key for ServerSocketChannel : OP_ACCEPT
Key for Client /127.0.0.1:38106 : OP_READ
Key for Client /127.0.0.1:38104 : OP_READ
Starting select
Client /127.0.0.1:38106 can perform : READ
Select finished
The selector contains:
Key for ServerSocketChannel : OP_ACCEPT
Key for Client /127.0.0.1:38106 : OP_READ
Key for Client /127.0.0.1:38104 : OP_READ|OP_WRITE
Starting select
Client /127.0.0.1:38104 can perform : WRITE
Select finished
The selector contains:
Key for ServerSocketChannel : OP_ACCEPT
Key for Client /127.0.0.1:38106 : OP_READ
Key for Client /127.0.0.1:38104 : OP_READ
Starting select
```

Vong Stéphane - Groupe 2 – Université Gustave Eiffel
Diagne Ben – Groupe 2 – Université Gustave Eiffel

Architecture

Nous nous sommes principalement basés sur les derniers TPs traitant les clients et les serveurs TCP non-bloquants pour faire ce projet. On est parti sur cette voie puisque les derniers Tps avaient déjà un Reader basique qui était crucial pour la lecture des paquets dans le projet, en plus d'avoir un squelette du client et du serveur déjà toute faite. En ce qui concerne les modifications, nous sommes partis sur deux maps stockant les noms de serveur en tant que clé et sa SelectionKey en valeur. Cela nous permettra de pouvoir regrouper tous les clients et serveurs qu'on ouvre/crée, ce qui, en retour, nous permettra d'envoyer des messages privés aux clients par exemple. On s'est aussi aidé de visiteurs pour raccourcir les fonctions de process. En effet, la fonction de processIn doit évaluer tous les types de paquet, donc le switch finira par être trop gros. On utilise donc les visiteurs pour au moins éviter les duplications dans le cas des paquets désignant les messages publiques et privées.

Les choses qui fonctionnent

- La connection du serveur au client
- L'envoi des messages publiques
- L'envoi des messages privées
- Une partie de la fusion

Les choses qui ne marchent pas

- L'envoi des fichiers
- L'autre partie de la fusion

Difficultés rencontrés

La principale difficulté qu'on a eu avec le projet en général était de débbugger. Le projet étant assez gourmand en quantité de classe, il était difficile de cerner la plupart des problèmes sur lequel on tombait. De plus, puisque le programme a des classes qui dépendent beaucoup entre elles, se répartir les tâches et travailler de notre côté était compliqué. Il était mieux de travailler ensemble directement pour éviter à devoir attendre que la tâche du binôme soit fini.

Une autre difficulté était de cibler les serveurs. De base, on ne savait pas comment récupérer le numéro d'un serveur, donc essayer de cibler un serveur était dur. On a

alors eu l'idée de créer une map des serveurs avec leur nom au lieu de leur numéro de port en clé, ce qui nous a facilité la tâche. Cette difficulté est similaire à ce qu'on a eu pour chercher comment envoyer des messages privés. Pour la solution, on a procédé de la même manière, qui était de créer une map des clients marchant de la même façon que la map des serveurs.

En ce qui concerne la fusion, elle est incomplète pour une raison, qui est qu'après avoir reçu le paquet 9, on fait un `logger.info("FUSION DONE")`, mais ce msg s'affiche à l'infini. Nous avons malheureusement pas trouvé de solution à ce problème.

Améliorations faites après la soutenance bêta

"Quand un login n'est pas valide, il faut que le serveur nous relance et pas fermer la connexion."

Une chose qu'on faisait est que dans le `processIn`, nous fermons la connection dans le cas où le type de paquet était de 0 (le tout premier paquet qui fait la connection du client au serveur via authentification). On a tout simplement supprimé l'appel à `silentlyClose()` qu'on faisait dans ce cas là, et on l'a remplacé par un `return`.

"Ne pas oublier de vérifier que le buffer soit plein quand on passe par `processOut(int opcode)`."

Avant la soutenance, on avait oublié de vérifier une condition de `processOut`, qui est le cas où le buffer est plein. On a juste ajouté ce bout de code pour y palier.

```
private void processOut(int opcode) {  
    if (bufferOut.remaining() < Integer.BYTES) {  
        return;  
    }  
    if(!bufferOut.hasRemaining()) {  
        return;  
    }  
    if (opcode == 2) {  
        var serv = queue.poll();  
        if (serv == null) {  
            return;  
        }  
        var encodedServ = StandardCharsets.UTF_8.  
    }  
}
```

"Le switch du `processIn()` de `ClientChat` et `ServeurChatOn` est trop volumineux."

Avant la soutenance, on avait un switch très grand pour les deux cas. Pour résoudre ce problème pour le cas du serveur, on a rentré le contenu des cas du switch dans des méthodes séparés qu'on a regroupé dans la même classe. En ce qui concerne le client, on a implémenté des visiteurs pour les messages publiques et privées.

Serveur:

```
private void processIn() {
    bufferIn.flip();
    var tmp = bufferIn.getInt();
    bufferIn.compact();
    while(true) {
        switch (tmp) {
            case 0 :
                if(processInConnection() == 0)
                    silentlyClose();
            case 4 :
                if(processInPublicMessage() == 0)
                    silentlyClose();
            case 5 :
                if(processInPrivateMessage() == 0)
                    silentlyClose();
            default:
                return;
        }
    }
}
```

```
private int processInPublicMessage() {
    Reader.ProcessStatus status;
    status = publicReader.process(bufferIn);
    switch (status) {
        case DONE:
            var value = publicReader.get();
            server.broadcast(value, 4);
            publicReader.reset();
            break;
        case REFILL:
            break;
        case ERROR:
            return 0;
    }
    return 1;
}
```

Client:

```
while(true) {
    ProcessStatus status;
    switch (opCode) {
        case 2 :
            status = stringReader.process(bufferIn);
            switch (status) {
                case DONE:
                    nameServer = stringReader.get();
                    System.out.println("Welcome " + login + " in " + nameServer);
                    stringReader.reset();
                    break;
                case REFILL:
                    return;
                case ERROR:
                    silentlyClose();
                    return;
            }
            break;
        case 3 :
            System.out.println("Login failed");
            silentlyClose();
            break;
        case 4 :
            PublicVisitor publicVisitor = new PublicVisitor();
            if(publicReader.accept(publicVisitor, bufferIn) == 0)
                silentlyClose();
        case 5 :
            PrivateVisitor privateVisitor = new PrivateVisitor();
            if(privateReader.accept(privateVisitor, bufferIn) == 0)
                silentlyClose();
        default:
            return;
    }
}
```

La fonction visit() des visiteurs contiennent les contenus des cas de switch.