**Partie 1**

Question 1.

Les résultats présentés ci-haut ont été obtenu en calculant la moyenne, pour chaque taille de paquet, du temps nécessaire avant d’obtenir une réponse vide du serveur. Une exécution consiste en l’envoie de 7 paquets, de taille 101 à 107 octets. Pour les appels normaux, on remarque une baisse du temps nécessaire entre les paquets de 10 octets et ceux de 100. Nous croyons qu’il s’agit d’une latence dû à la première exécution d’une fonction : latence causée par la JVM qui invoque pour la première fois une nouvelle fonction. Par la suite, on observe une hausse pour chaque augmentation de taille du paquet, s’approchant des appels locaux. En ce qui concerne les appels locaux, et distants, on observe également une baisse du temps nécessaire durant les premiers paquets. Nous supposons que la cause est similaire à celle des appels normaux, mais additionné à la latence de la communication réseau. Pour des tailles de paquets plus grosses, on observe une latence plus élevée pour les appels distants en comparaison aux appels locaux, ce qui est normal puisque l’appel se fait sur une plus grande distance.

Un avantage important de Java RMI est qu’il permet l’appel de fonction à distance comme s’il s’agissait de fonctions locales, malgré un délai plus important. Cette intégration avec le langage permet de faire abstraction de la communication entre les différentes composantes (serveurs et clients), et de programmer comme s’il s’agissait d’une seule grosse composante. Puisque RMI abstrait les appels distants, on profite donc d’une plus basse verbosité dans le code.

Le principale désavantage que nous avons remarqué est les délais d’exécution, et une configuration plus compliquée de la communication entre les composantes, en comparaison à un API web (par exemple, REST).

Question 2.

Au lancement du serveur, le serveur s’inscrit dans un registre RMI qui existe sur sa machine (fonction *run* de la classe *Server*). Par la suite, lorsque le client tente d’accéder au serveur, il communique avec le registre RMI de la machine distance (fonction *loadServerStub* de la classe *Client*) et y récupère une référence à l’instance du serveur, de type *ServerInterface*. Par la suite, les appels aux fonctions du serveur sont abstraits par Java RMI qui s’occupe de la traduction des références entre la machine locale et la machine distante.