# Rapport de projet

## Description des expériences

Nous avons conduit l'expérience suivante sous 4 scénarios différents :

- Définir un nombre de nodes N à contacter.
- Commencer les tests avec W=1.
- Compter combien de fois la requête réussit sur un nombre prédéfini de tests (toujours 100 dans notre cas). La clé est différente pour chaque test, testant donc différents *hashs* afin de pouvoir tester différents serveurs.
- Incrémenter W de 1 jusqu'à ce que l'on arrive à la fin ou, de manière plus probable, jusqu'à ce que l'on n'ait plus aucun succès lors de 3 valeurs de W différentes de suite.

Nous avons donc défini les scénarios suivants, qui seront par la suite nommés par leur numéro :

1. N=174 (le nombre total de serveurs). Cela permet donc de s'abstraire du hash pour tester uniquement le réseau (tous les serveurs sont de toute façon contactés).
2. N=174 tout en ajoutant une fonction « wait » entre chaque envoi de paquet qui permet au buffers de paquets de « respirer » entre chaque envoi.
3. N=50 sans « wait ». Ce scénario permet de plus se concentrer sur les rings à proprement parler, en faisant un sélection a priori des serveurs, et toujours différente pour chaque test effectué.
4. N=50 avec « wait ».

La fonction wait est très simple et permet d'obtenir des résultats satisfaisants, testés avec pps-list-nodes notamment, qui était très aléatoire sans cette fonction (notamment avec beaucoup de nodes) :

```
unsigned int wait = -1;
while (wait != 0) { --wait; }
```

La façon dont les tests ont été implémentés est expliquée dans le README. Ils ont été lancés grâce aux exécutables « test-servers » et « test-parser », qui lui calcule les moyennes et écarts-types du temps passé pour les succès et échecs (moyennes séparées!).

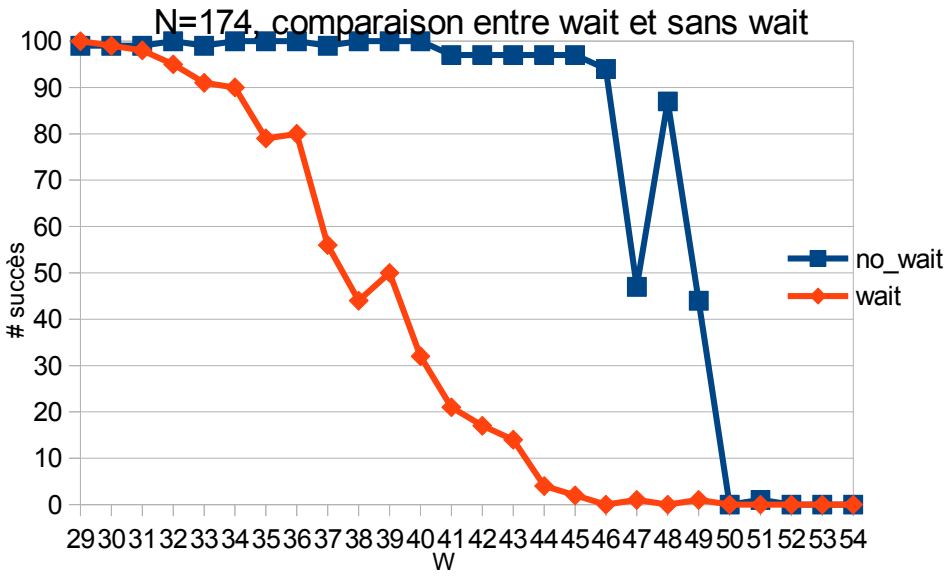
## Résultats

Pour les résultats, nous avons décidé de nous concentrer sur les valeurs temporelles des succès, les échecs nous intéressant bien moins. Les résultats en tant que tels se trouvent dans le dossier `resultats`.

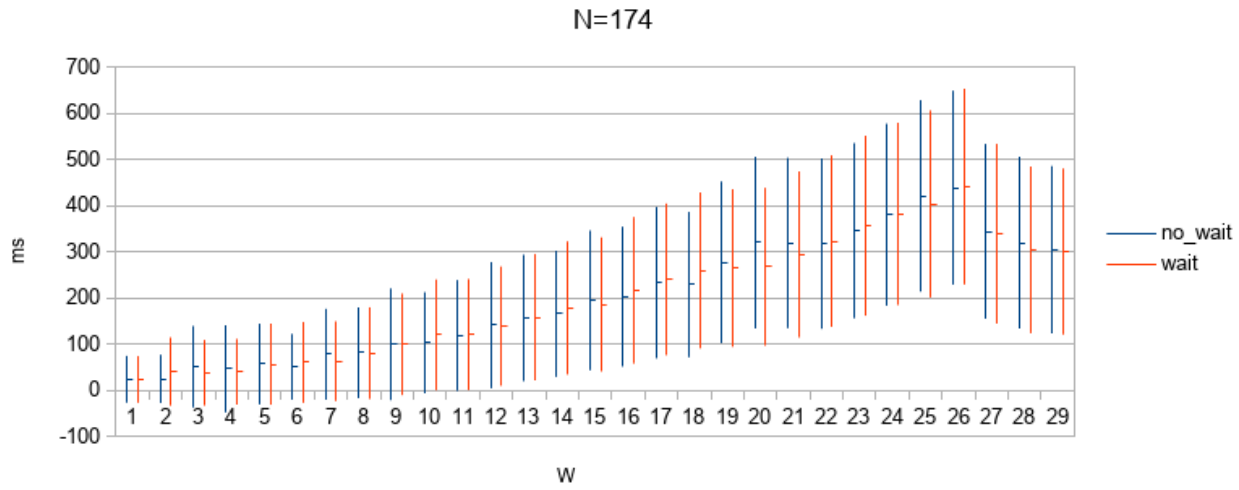
### Scénarios 1 et 2

On remarque que le scénario 2 (celui ayant une attente) commence à observer des échecs beaucoup plus tôt que le scénario 1 (celui sans attente). Cela est proablement dû au fait qu'il a été exécuté après

le 1er, lors d'une période où les serveurs ont été mis à rude épreuve et où ils ont commencé à s'éteindre peu à peu.
La remarque intéressante à faire est que le nombre d'échecs augmente de façon bien plus graduelle que lorsqu'il n'y a aucune attente entre les envois. Cela peut cependant être aussi dû à l'instabilité du système de serveurs plutôt qu'à une éventuelle stabilité développée par ce scénario.
Le nombre de succès était très proche de ou égal à 100 avant W=29 pour ces deux scénarios.

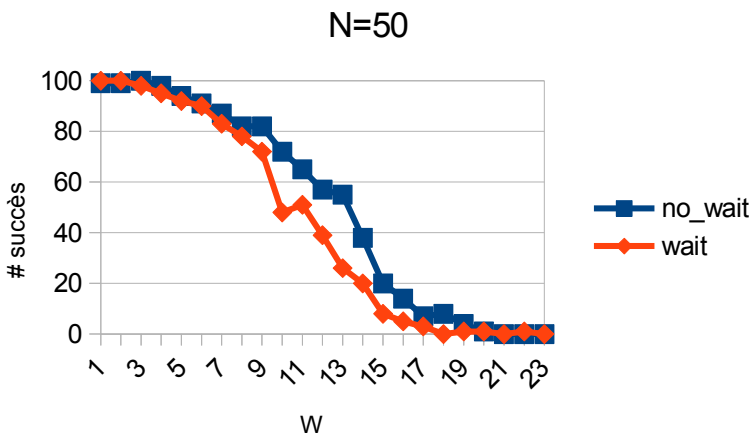



En ce qui concerne les temps, on remarque une différence quasi nulle entre le scénario avec et sans attente, ce qui semble logique au vu du code très léger qui fait attendre le programme. Si le fait d'attendre est réellement bénéfique pour l'envoi des paquets, alors il faut absolument l'implémenter puisqu'il n'y a aucune répercussion temporelle (la barre d'erreur **en entier** représente un écart-type). Ici, nous nous sommes concentrés sur les temps lors des succès probants (autour de 100).



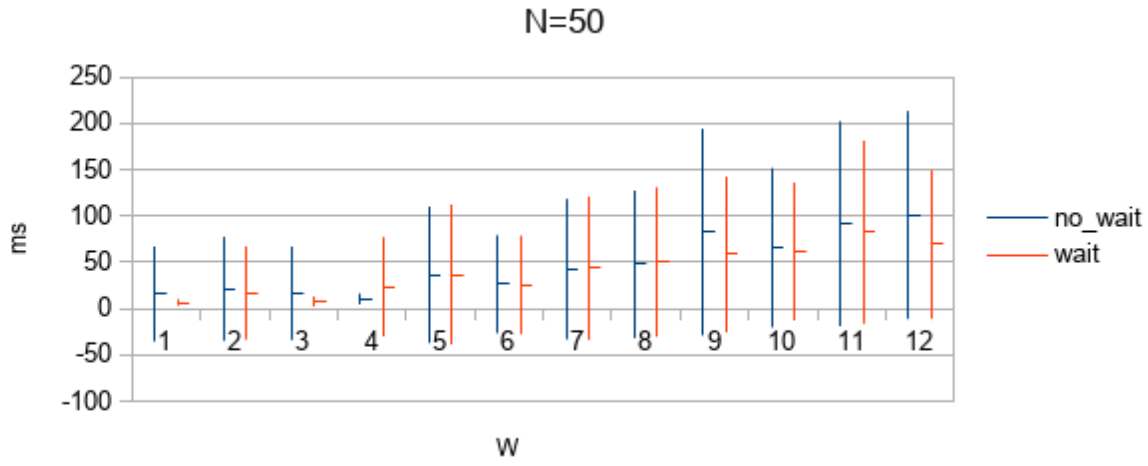

## Scénarios 3 et 4

Ici, la différence est beaucoup plus subtile, et certainement uniquement due au fait que le système des serveurs était instable au moment des tests. La différence



pourrait aussi s'expliquer par le fait qu'il y a moins de messages à envoyer qu'auparavant (50 vs 174), donc la fonction wait a beaucoup moins d'impact que tout à l'heure puisque les paquets UDP ne s'écrasent plus.

Encore une fois, nous n'observons aucune différence de temps significative entre usage ou non de l'attente.




# Conclusion

Nous observons que nous passons en-dessous de 50 succès pour chaque scénario avec les ratios W/N suivants :

1. 49/174 = 0.282
2. 40/174 = 0.230
3. 14/50 = 0.280
4. 12/50 = 0.240

Le ratio W/N montre à partir de quelle proportion de serveurs sur la liste initiale il devient difficile d'opérer des demandes d'écritures. Si l'on demande qu'un trop grand nombre de serveurs soit disponible, alors on échouera presque systématiquement, comme montré dans nos graphiques.

Ce qui veut dire que lorsque l'on essayait de contacter plus du quart des serveurs listés (que ce soit 50 ou 174), un échec était plus probable qu'un succès. Cela est clairement dû au fait que la plupart des serveurs ne répondaient pas lors des tests, mais correspond très bien au nombre de serveurs actifs à l'époque.

On peut donc voir dans nos exemples que le nombre de serveurs contactés n'influence pas le succès ou non du quorum, ce qui d'un point de vue probabiliste est tout à fait cohérent.

Le fait d'attendre entre chaque envoi de paquet pourrait être bénéfique (on évite de les écraser si on en envoie beaucoup), mais cela a été difficile à discerner à cause des problèmes structurels mentionnés.