

Rapport de Simulation

Glineur Pierre
Matricule1

Tiako Ngouadje Cardin Patson
Matricule2

11 janvier 2026

Table des matières

1	Exécution	2
1.1	Structure du projet	2
1.2	Instructions de reproduction	2
2	Implémentation	2
2.1	Approche choisie	2
2.2	Difficultés rencontrées	2
2.3	État final	2
3	Simulations	3
3.1	Analyse des résultats et réponses aux questions de l'énoncé	3
3.1.1	Stop and Wait (Modes 2 & 3)	3
3.1.2	Pipelining avec fenêtre fixe (Mode 4)	3
3.1.3	Pipelining avec fenêtre dynamique (Mode 5)	3
4	Conclusion	3

1 Exécution

Cette section détaille la structure du projet et la marche à suivre pour reproduire nos résultats.

1.1 Structure du projet

Le projet est organisé autour d'un moteur de simulation discret (Simulator.py) et d'entités réseau (Host.py, Router.py, NIC.py, Link.py).

1.2 Instructions de reproduction

Pour reproduire les scénarios décrits dans ce rapport, exécutez les scripts suivants à la racine du projet :

- Mode 1 : `python mode1NO_RELIABILITY.py`
- Mode 2 : `python mode2acquittements.py`
- Mode 3 : `python mode3avecetretransmission.py`
- Mode 4 : `python mode4pipelining.py`
- Mode 5 : `python mode5pipeliningavecfenetredynamique.py`
- Analyse Globale : `python run_all_scenarios.py` (Script maître avec paramètres de perte éditables).

2 Implémentation

2.1 Approche choisie

Notre implémentation repose sur l'extension de la classe Host pour gérer les mécanismes de fiabilité décrits dans le chapitre 3 du cours (Transport Layer). Nous avons mis en place :

- Le Frigo (`_unacked_packets`) : Gestion de la fenêtre glissante.
- Le Buffer (`_buffer`) : File d'attente logicielle pour le stockage des paquets en attente de fenêtre libre.
- Le Timer unique : Implémentation du protocole rdt 3.0 avec un décompte pour le paquet le plus ancien.

2.2 Difficultés rencontrées

- Gestion des ACKs cumulatifs : S'assurer que la réception d'un ACK libère correctement tous les paquets précédents du "frigo".
- Synchronisation du Timer : Éviter les redondances de timers lors de l'envoi rapide de plusieurs paquets en mode Pipelining.

2.3 État final

L'implémentation supporte les 5 modes demandés. Les modes 2 et 3 fonctionnent en Stop-and-Wait, tandis que les modes 4 et 5 exploitent le Pipelining pour optimiser l'utilisation de la bande passante.

3 Simulations

3.1 Analyse des résultats et réponses aux questions de l'énoncé

3.1.1 Stop and Wait (Modes 2 & 3)

- Mode 2 (ACK) : Sans timer, nous observons que la simulation se bloque indéfiniment en cas de perte (Lien L1 avec `lost_prob = 0`). Cela illustre les limites du protocole rdt 2.2 décrit aux pages 34-40 du cours.
- Mode 3 (Retransmission) : Le log montre @0.500000 Timer expired! Retransmitting. L'émetteur détecte la perte et survit à l'incident.
- Temps moyen de transmission : [Insérez votre calcul ici : Moyenne des temps entre sends et received ACK].

3.1.2 Pipelining avec fenêtre fixe (Mode 4)

- Observations : Contrairement au mode 3, l'émetteur envoie plusieurs paquets (fenêtre de 4) dès $t=0.000000$ sans attendre les premiers ACKs.
- Comparaison : Le temps moyen de transmission diminue drastiquement car le lien ne reste pas inactif durant le temps de propagation (RTT).

3.1.3 Pipelining avec fenêtre dynamique (Mode 5)

- Comportement attendu : La fenêtre commence à 1, augmente à chaque ACK, et chute à 1 en cas de timeout.
- Analyse du Goulot : Avec un lien L2 à 500 kbps et une file au routeur de 10, nous observons une congestion.
- Débit moyen effectif : il faudra Insérer le calcul : $(\text{Paquets} * \text{Taille} * 8) / \text{Temps total}$.
- Justification : L'ajustement dynamique permet de trouver le débit maximal du lien critique (500 kbps) sans saturer indéfiniment le routeur.

4 Conclusion

La comparaison entre les modes fixe et dynamique montre que le mode dynamique (inspiré de TCP) est plus résilient et efficace pour s'adapter aux capacités variables des nœuds intermédiaires (routeurs).