Rapport de projet EvalPerf – Partie 1 (Flux TCP)

Jusqu'à la fin de la section 1.2

Nom Prénom :
N° étudiant :

14 décembre 2025

Résumé

Ce projet a pour but d'étudier le comportement de TCP en présence d'un goulot d'étranglement (bottleneck) et d'une hétérogénéité de RTT, en utilisant le simulateur NS-2. Dans ce rapport, jusqu'à cette étape, nous avons implémenté le scénario de base (section 1.1) puis, en section 1.2, nous avons exécuté deux cas : *avec congestion* et *sans congestion* sur le lien `core`, afin de préparer l'extraction des résultats (débits/throughput et équité).

Table des matières

1 Introduction

Dans la Partie 1 du projet, on s'intéresse aux flux TCP et au contrôle de congestion. L'idée générale est de construire une topologie simple avec un lien central (*core link*) jouant le rôle de goulot d'étranglement, puis de faire passer plusieurs flux TCP avec des RTT différents afin d'observer l'impact du RTT sur le partage de la bande passante et la notion d'équité (*fairness*).

2 Environnement et outils

— Simulateur : NS-2
— Langage des scénarios : Tcl
— Sorties : fichier `trace` (pour l'analyse) et fichier `NAM` (visualisation, si nécessaire)
— Outil d'analyse simple : AWK pour calculer le throughput moyen de chaque flux depuis le `trace`

3 Section 1.1 – Mise en place du scénario de base

3.1 Objectif

L'objectif de la section 1.1 est de construire un scénario de base correct et exécutable, qui sera réutilisé dans les sections suivantes. À ce stade, il n'est pas demandé de produire des analyses poussées ou des statistiques avancées ; il faut surtout que la topologie, les flux et l'hétérogénéité de RTT soient bien définis.

3.2 Topologie

La topologie contient 8 nœuds :
— Deux routeurs centraux : `r0` et `r1`
— Trois émetteurs à gauche : `s0, s1, s2`
— Trois récepteurs à droite : `d0, d1, d2`

Les deux routeurs `r0` et `r1` sont connectés par un lien central (`core`) qui est destiné à devenir le *bottleneck*. Les nœuds `s0,s1,s2` sont connectés à `r0`, et les nœuds `d0,d1,d2` sont connectés à `r1`.

3.3 Flux

Conformément à l'énoncé, nous créons 6 flux TCP/FTP :
— Pour chaque paire émetteur/récepteur (`s0`→`d0`, `s1`→`d1`, `s2`→`d2`), on crée deux flux indépendants, soit 6 flux au total.
— Chaque flux est composé d'un agent `Agent/TCP` côté émetteur, d'un `Agent/TCPSink` côté récepteur, et d'une application `Application/FTP` au-dessus de TCP.
— Les instants de démarrage des flux sont légèrement décalés afin d'éviter une synchronisation parfaite.

3.4 Hétérogénéité des RTT

Pour obtenir des RTT hétérogènes, on modifie les délais des liens d'accès (access links) selon la paire considérée :

— `s0`→`d0` : faible délai (RTT faible)
— `s1`→`d1` : délai moyen (RTT moyen)
— `s2`→`d2` : délai élevé (RTT élevé)

Comme le RTT dépend approximativement de la somme des délais sur le chemin aller-retour (liens d'accès + lien `core`), ce choix garantit une différence nette de RTT entre les trois paires, ce qui sera utile pour étudier l'équité en présence d'un goulot d'étranglement.

3.5 Sorties produites

La simulation de la section 1.1 produit typiquement :
— `out.tr` (ou nom similaire) : fichier `trace` pour analyse
— `out.nam` (optionnel) : visualisation dans NAM

4 Section 1.2 – Cas avec congestion et cas sans congestion

4.1 Objectif

La section 1.2 consiste à exécuter deux expériences :

1. **Cas congestion** : le lien `core` est un bottleneck et reste congesté suffisamment longtemps pour obtenir des résultats exploitables.
2. **Cas sans congestion** : le lien `core` n'est pas un bottleneck (et la fenêtre de réception n'est pas limitante), puis on observe ce qui change.

4.2 Cas "avec congestion" (Congestion case)

Pour provoquer une congestion sur le lien `core`, on fixe une faible capacité sur ce lien (par exemple `2Mb`) et une limite de file relativement petite, afin d'observer des pertes et le fonctionnement du contrôle de congestion TCP. La durée de simulation est assez longue pour ignorer un temps de "warm-up" et observer un comportement plus stable.

Fichier scénario : `part1_2_congest.tcl`
Sorties :
— `p12_congest.tr`
— `p12_congest.nam` (optionnel)

4.3 Cas "sans congestion" (No-congestion case)

Dans ce cas, on configure le lien `core` de sorte qu'il ne soit plus le goulot d'étranglement. Une remarque pratique : si on met une capacité énorme sur le `core` tout en gardant `trace-all`, la simulation peut devenir très lente, car énormément de paquets sont générés et tracés. Pour garder une exécution raisonnable, on utilise des débits "mis à l'échelle" (tout en conservant l'idée que `core` n'est pas limitant).

Fichier scénario : `part1_2_nocongest.tcl`
Sorties :

— `p12_nocongest.tr`
— `p12_nocongest.nam` (optionnel ; si la simulation est lente, on peut désactiver NAM)

4.4 Extraction du throughput par flux

Pour obtenir un résultat simple et directement exploitable dans le rapport, on calcule le throughput moyen de chaque flux à partir du fichier `trace`. Le principe est de sommer les octets TCP reçus au niveau des récepteurs sur un intervalle de temps de mesure, puis de convertir en Mbps.

Script : `throughput.awk`

Exemples d'exécution :

— Cas congestion :

```
gawk -f throughput.awk -v t0=20 -v t1=180 p12_congest.tr
```

— Cas sans congestion :

```
gawk -f throughput.awk -v t0=10 -v t1=50 p12_nocongest.tr
```

4.5 Sorties attendues pour la section 1.2

Pour répondre correctement à la section 1.2, on vise au minimum :
— Les deux fichiers `trace` (congestion / no-congestion)
— Une table de throughput par flux (6 flux) pour chaque cas
— Un commentaire qualitatif : partage de bande passante et équité en fonction du RTT (cas congestion), puis description de ce qui change lorsque le `core` n'est plus un bottleneck

4.6 Tableaux de résultats (gabarit)

4.6.1 Throughput – cas congestion

TABLE 1 – Throughput moyen par flux (cas congestion) sur l'intervalle `t0`–`t1`

Flux (fid)	Chemin	RTT (qualitatif)	Throughput (Mbps)
1	s0→d0	faible	
2	s0→d0	faible	
3	s1→d1	moyen	
4	s1→d1	moyen	
5	s2→d2	élevé	
6	s2→d2	élevé	

4.6.2 Throughput – cas sans congestion

TABLE 2 – Throughput moyen par flux (cas sans congestion) sur l'intervalle `t0–t1`

Flux (fid)	Chemin	RTT (qualitatif)	Throughput (Mbps)
1	s0→d0	faible	
2	s0→d0	faible	
3	s1→d1	moyen	
4	s1→d1	moyen	
5	s2→d2	élevé	
6	s2→d2	élevé	

4.7 Discussion préliminaire (jusqu'ici)

Dans le cas congestion, on s'attend généralement à ce que des flux avec RTT plus faible obtiennent une part plus importante du goulot, car la dynamique AIMD (augmentation additive / diminution multiplicative) est liée au RTT (rythme des ACK, vitesse d'augmentation de la fenêtre de congestion, etc.). Dans le cas sans congestion (lien `core` non limitant et fenêtre de réception non limitante), les pertes sur le `core` deviennent très rares, et les limitations observées proviennent davantage des liens d'accès et du partage local. L'effet du RTT sur le partage d'un *bottleneck commun* est donc en général moins visible. Les sections suivantes permettront de confirmer cela avec des valeurs mesurées (throughput) et, si nécessaire, des courbes.

5 Suite du rapport (à compléter)

5.1 Section 1.3

Comparaison de différentes variantes TCP (agressivité, équité, réglages).

5.2 Section 1.4

Impact de la taille/type de file d'attente sur le goulot.

5.3 Section 1.5

Scalabilité, trafic de fond (*background traffic*) et analyse des retransmissions.

Annexe

Fichiers

- `part1_1.tcl`
- `part1_2_congest.tcl`
- `part1_2_nocongest.tcl`
- `throughput.awk`