Projet 2 (ns-3) — Analyse comparative UDP vs TCP
Goodput et RTT sur une topologie Wi-Fi + CSMA

Contexte et objectif. L'objectif de ce travail est d'évaluer, sur une même topologie hybride *Wi-Fi + CSMA*, les performances d'un trafic *UDP* (débit offert imposé) et d'un trafic *TCP* (contrôle de congestion), en se concentrant sur deux indicateurs opérationnels : le *goodput* (débit utile réellement reçu par l'application) et la *latence aller-retour* (*RTT*) mesurée *sous charge*. Au-delà d'une simple comparaison de moyennes, l'enjeu est de caractériser la dynamique temporelle (séries temporelles) et la variabilité (CDF/percentiles), afin d'expliquer les comportements observés.

Topologie, paramétrage et déroulement expérimental. La topologie comporte trois nœuds : `STA` (client), `AP` (point d'accès) et `Server`. Le lien `STA`↔`AP` est un lien Wi-Fi (802.11ax), et le lien `AP`↔`Server` est un lien CSMA. Deux sous-réseaux IPv4 sont utilisés : `10.1.0.0/24` côté Wi-Fi et `10.2.0.0/24` côté CSMA. Les tables de routage globales sont générées après l'affectation IP, ce qui garantit la connectivité `STA`→`Server` à travers les deux segments. La mobilité est fixe : AP en $(0, 0, 0)$, STA en $(d, 0, 0)$ et Server en $(0, 1, 0)$, avec $d = 5.000\,\text{m}$. La simulation dure `simTime` $= 20.000\,\text{s}$ et le trafic principal démarre à `appStart` $= 2.000\,\text{s}$, ce qui fixe la durée utile à

$$T_{\text{utile}} = \texttt{simTime} - \texttt{appStart} = 18.000\,\text{s}.$$

Deux exécutions sont réalisées : (i) UDP, `udpRate` $= 50.000\,\text{Mbps}$, `pktSize` $= 1200.000\,\text{B}$; (ii) TCP, `tcpMaxBytes` $= 0$ (illimité), `pktSize` $= 1200.000\,\text{B}$. Les graines RNG sont fixées (`seed/run`) afin de rendre les résultats reproductibles.

Mesures : goodput et RTT sous charge. Le *goodput global* est calculé en fin de simulation à partir du nombre total d'octets reçus par le `PacketSink` :

$$\text{goodput (bps)} = \frac{8 \cdot \texttt{rxBytes}}{T_{\text{utile}}}.$$

Cette définition est essentielle : moyenner sur `simTime` au lieu de T_{utile} sous-estimerait artificiellement le débit utile (période sans trafic). En complément, une *série temporelle* de goodput est obtenue par échantillonnage périodique de `GetTotalRx()` (pas $\Delta t = 0.500\,\text{s}$) ; le goodput instantané est estimé à partir de la variation Δ`rxBytes` sur Δt. Le RTT est mesuré pendant la période de trafic principal via une sonde à faible fréquence ($\approx 5.000\,\text{Hz}$). Lorsque l'outil `V4Ping` n'est pas disponible dans le build, une sonde *echo* horodatée (timestamp + numéro de séquence) fournit une mesure équivalente : la sonde envoie un paquet léger, le serveur le renvoie, et le client calcule RTT $= t_{\text{rx}} - t_{\text{tx}}$. Les valeurs RTT sont exportées en CSV et peuvent être affichées dans la console.

Résultats quantitatifs (synthèse). Le Tableau 1 récapitule les métriques principales issues des sorties `console`, `stats` et `FlowMonitor` (UDP). On observe deux signatures nettes : (i) UDP atteint un débit utile $\approx 7.000\,\text{Mbps}$ relativement stable, tandis que (ii) TCP converge vers un débit utile plus faible ($\approx 2.260\,\text{Mbps}$) et présente une latence beaucoup plus variable (queueing tail marqué).

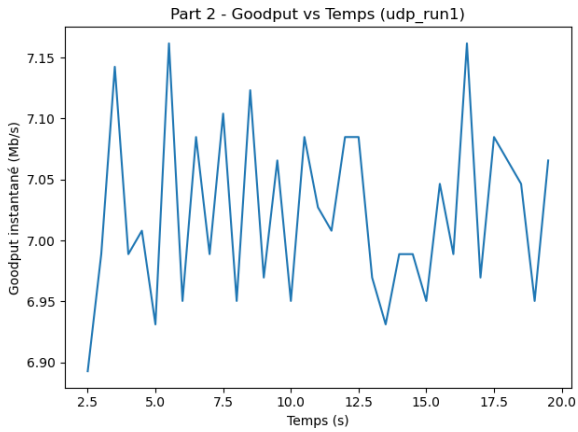
TABLE 1 – Synthèse des résultats (d=5.000 m, `simTime`=20.000 s, `appStart`=2.000 s).

Transport	Goodput global	Goodput moyen (TS)	RTT moyen	RTT p95 / p99
UDP (run1)	7.022 Mbps	7.023 Mbps	138.640 ms	143.000 ms / 144.000 ms
TCP (run2)	2.260 Mbps	2.259 Mbps	141.080 ms	198.800 ms / 240.100 ms

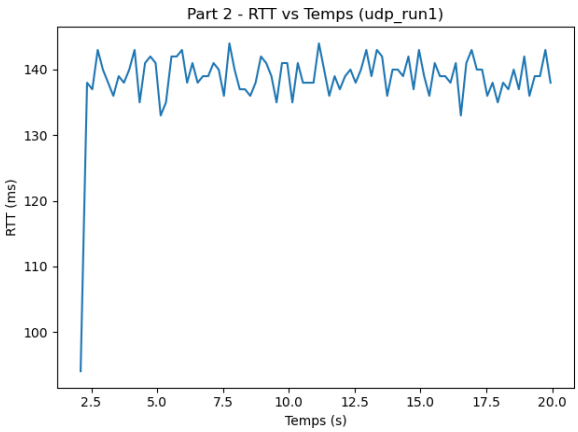
Analyse UDP : débit stable mais saturation et pertes massives. Les Figures 1 et 2 montrent une stabilité remarquable du goodput instantané autour de 7.000 Mbps ainsi qu'un RTT relativement concentré (CDF très abrupte). Cependant, l'analyse `FlowMonitor` révèle que cette stabilité n'implique pas un fonctionnement "confortable" du réseau. En effet, le débit offert UDP (50.000 Mbps) est largement supérieur à la capacité effectivement soutenable dans ce scénario, ce qui force une saturation : le flux principal (destination `port` = 5000) subit une *forte perte en paquets* (environ 75% des paquets transmis sont perdus). En parallèle, les paquets qui *arrivent* peuvent accumuler un retard très important : le délai moyen (one-way) reporté par FlowMonitor est de l'ordre de plusieurs secondes sur le flux principal, alors que la sonde RTT (port `9000`) conserve une latence bien plus faible et stable. Ce contraste s'explique par deux mécanismes qui peuvent coexister : (i) l'existence d'une *queueing tail* extrême sur une fraction des paquets du flux saturant (quelques paquets très retardés suffisent à faire exploser la moyenne), et (ii) la nature *faible débit / petits paquets* de la sonde RTT, qui échantillonne le réseau sans contribuer significativement à l'encombrement et peut traverser les périodes de congestion avec une pénalisation plus limitée. D'un point de vue réseau, le résultat essentiel n'est donc pas seulement "UDP a plus de débit", mais "UDP peut forcer le réseau en saturation", au prix d'un compromis sévère : pertes élevées et délais très dégradés pour le trafic de données.

Analyse TCP : débit inférieur mais variabilité de latence marquée. Les Figures 3 et 4 mettent en évidence une dynamique typiquement TCP. Le goodput converge vers ≈ 2.300 Mbps après une phase transitoire, et l'on observe des épisodes de baisse suivis de récupération. Dans le même temps, le RTT présente des *pics* pouvant atteindre ≈ 240.000 ms et une CDF nettement plus étalée que pour UDP : alors que la médiane TCP reste proche de 135.000 ms, les percentiles élevés (p95 ≈ 199.000 ms, p99 ≈ 240.000 ms) révèlent un *tail* de latence important. Ce comportement est cohérent avec le couplage entre le contrôle de congestion TCP (croissance/réduction de fenêtre, réactions à la perte ou au délai) et l'occupation des files : lors des périodes où la file se construit, la latence augmente ; après un événement de congestion (perte ou signal équivalent), la fenêtre se réduit, le débit chute temporairement et la file se draine, ce qui provoque une baisse rapide du RTT. La signature conjointe "pic de RTT + perturbation du goodput" est précisément ce qui distingue TCP d'un trafic UDP à débit imposé.

Comparaison et interprétation globale. Au niveau *débit utile*, UDP domine dans ce scénario (≈ 7.000 Mbps contre ≈ 2.260 Mbps pour TCP), mais cette domination est largement artificielle dans le sens où elle est obtenue en poussant un débit offert (50.000 Mbps) incompatible avec la capacité du lien : le système se stabilise sur un débit reçu plafonné, en sacrifiant de très nombreux paquets. TCP, à l'inverse, obtient un débit inférieur mais impose une régulation : il ne cherche pas à maintenir un débit offert constant, et sa performance est intimement liée à la dynamique de congestion (variations de fenêtre, occupation/désoccupation de file), ce qui se traduit par une latence plus variable et un tail plus lourd. Du point de vue QoS, la comparaison la plus informative est celle des distributions : UDP présente une latence RTT très concentrée pour la sonde (p99 $\approx$ 144.000 ms), tandis que TCP présente une dégradation marquée des percentiles (p99 ≈ 240.000 ms). Cela signifie qu'une application sensible aux *pires cas* (gaming, VoIP, contrôle industriel) percevra davantage l'instabilité TCP dans ce scénario précis, même si la moyenne de RTT reste similaire.



(a) Goodput vs temps (UDP).



(b) RTT vs temps (UDP).

FIGURE 1 – Séries temporelles UDP : plateau de goodput ≈ 7.000 Mbps et RTT relativement stable après le démarrage du trafic.

Exploitation des figures et recommandations de présentation. Pour rendre l'analyse reproductible, les figures à inclure dans le dossier `docs/` sont : (i) goodput vs temps, (ii) RTT vs temps, (iii) RTT CDF, pour UDP et TCP. La présentation la plus efficace consiste à juxtaposer UDP et TCP sur des figures comparables, puis à commenter : stabilité du plateau de débit, apparition des pics de RTT, et lecture de la CDF (pente/étalement). Les fichiers `PCAP` permettent, si nécessaire, de confirmer la saturation (rafales, retransmissions MAC, pertes) via Wireshark.

Limites et perspectives. Deux limites doivent être explicitement mentionnées. Premièrement, si `V4Ping` n'est pas disponible dans le build, une sonde *echo* horodatée constitue un substitut valide pour mesurer le RTT sous charge, mais il faut l'indiquer clairement pour éviter toute ambiguïté de conformité. Deuxièmement, une analyse encore plus complète consisterait à exporter systématiquement `FlowMonitor` et les CSV pour *les deux* transports (UDP et TCP) afin de quantifier pertes et délais applicatifs de manière strictement comparable. Enfin, des campagnes paramétriques (variation de distance, débit offert, taille de paquet, MCS/PHY) permettraient de relier les observations à des seuils (transition non saturée $\rightarrow$ saturée) et d'établir des courbes de performance.

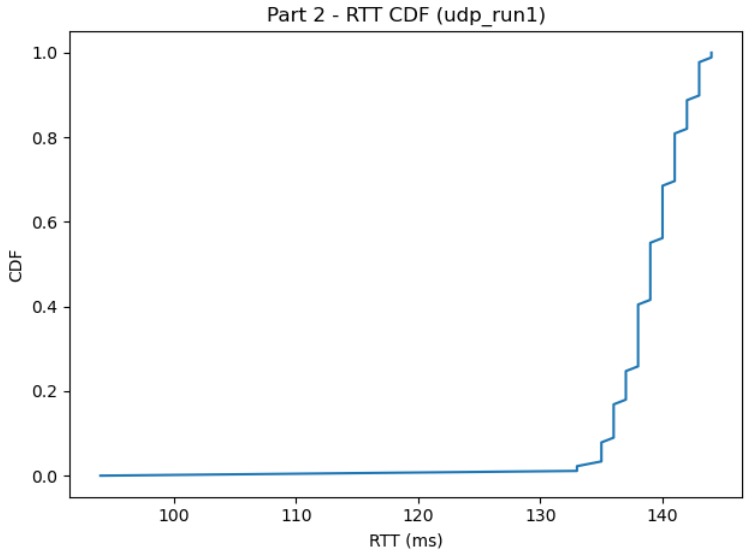
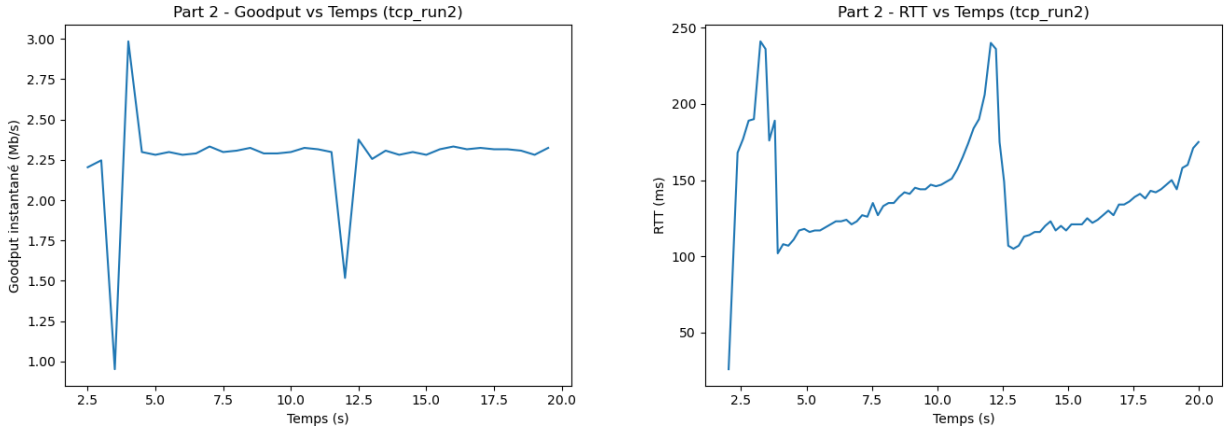


FIGURE 2 – CDF du RTT (UDP). La distribution est très concentrée (faible dispersion), p99 ≈ 144.000 ms.



(a) Goodput vs temps (TCP).

(b) RTT vs temps (TCP).

FIGURE 3 – Séries temporelles TCP : goodput plus faible et épisodes de congestion visibles (pics de RTT et perturbations du débit).

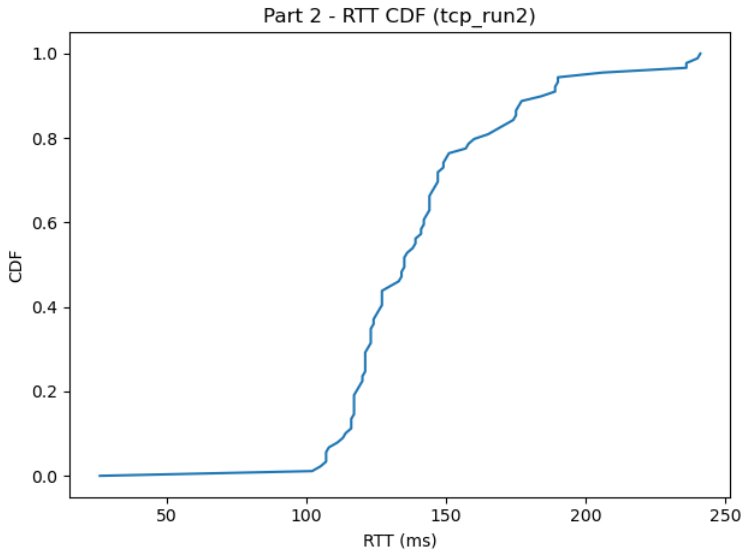


FIGURE 4 – CDF du RTT (TCP). La dispersion est plus forte que pour UDP : tail marqué (p95 ≈ 199.000 ms, p99 ≈ 240.000 ms).