UMONS
Université de Mons
Faculté des Sciences

Projet de réseaux informatiques
Modélisation d'un protocole de routage à vecteurs de distances

Étudiant: Mohammed KHAMLICHI, Matricule : 220416
Étudiant: Gédéon DASSI KOUAM, Matricule : 233363
Professeur: Bruno QUOITIN
Assistant: Valentin DUSOLLIER

26 mai 2025

Table des matières

Introduction

Le routage constitue le mécanisme fondamental qui permet aux paquets IP de circuler d'un point A vers un point B à travers un réseau maillé. Parmi les protocoles de routage dynamique, ceux basés sur les *vecteurs de distances* (Distance Vector) présentent l'avantage d'une mise en œuvre relativement simple, reposant sur l'échange périodique de courtes tables de coûts entre voisins. Toutefois, leur convergence peut être ralentie par les délais inhérents aux liens et ils sont sujets au phénomène du *count-to-infinity* lorsque les liens changent d'état.

Dans ce projet, nous avons développé un simulateur en Python capable de :

— **Charger** une topologie au format JSON, décrivant chaque lien par sa longueur, sa vitesse de propagation, son débit de transmission et son coût administratif.

— **Modéliser** l'envoi asynchrone de paquets DV entre routeurs, en calculant précisément les délais de propagation et de transmission en fonction de la taille du vecteur et des caractéristiques physiques des liens.

— **Gérer** dynamiquement des événements — modification de coût ou coupure de lien — à tout instant de la simulation, via un ordonnanceur d'événements.

— **Tracer** dans des logs horodatés l'ensemble des envois, réceptions et mises à jour de routes, puis afficher en fin d'exécution les tables de routage finales (destination, coût, next-hop) pour chaque routeur.

Nous appliquerons ce simulateur à trois scénarios clés :

1. *Validation fonctionnelle* sur une topologie de 5 routeurs, pour comparer automatiquement les routes calculées et les résultats attendus.
2. *Impact des délais*, afin de quantifier comment les différences de débit influent sur la chronologie de convergence sans modifier les routes optimales.
3. *Comptage à l'infini*, pour illustrer la fragilité du protocole pur DV face à un changement brutal de coût, et présenter la technique du *hold-down timer* comme solution partielle.

Cette approche nous permettra à la fois de vérifier la robustesse de notre implémentation et de mieux comprendre les compromis inhérents aux protocoles Distance Vector.“‘

1 Exécution

1.1 Structure des fichiers

Notre arborescence suit les consignes :

```
projet-reseaux-2025-gr14/
  rapport.pdf
  src/
    main.py
    simulateur_dv/
      __init__.py
      simulateur.py
      lien.py
      routeur.py
      table_routage.py
      message_dv.py
      evenement.py
    scenarios/
      scenario_validation.json
      scenario_delays.json
      scenario_count_to_infinity.json
```

1.2 Comment lancer une simulation

1. Placer les fichiers `.json` dans `scenarios/`.
2. Se positionner dans le dossier `src/` et exécuter, par exemple :

```
python main.py scenarios/scenario_validation.json
```

3. Observer les **logs** dans la console (envois et réceptions de DV, changements de routes, etc.).
4. En fin de simulation, les tables de routage sont imprimées.

2 Implémentation

2.1 Structure générale

Notre dépôt contient un point d'entrée `main.py` et un paquet `simulateur_dv/` :

— `simulateur.py` — classe **Simulateur**
  — file d'évènements (tas *min-heap*) ;
  — horloge simulée et exécution des actions ;
  — instanciation des **Routeur** et **Lien** à partir du JSON ;
  — logger horodaté pour toutes les traces.
— `evenement.py` — classe **Evenement** : enregistrement (instant_ms, compteur, action) trié par `heapq`.
— `lien.py` — classe **Lien** : longueur, vitesse de propagation, débit, coût administratif ; calcule $\Delta t_{\text{prop}} + \Delta t_{\text{trans}}$ et gère la mise hors service ($+\infty$).
— `routeur.py` — classe **Routeur** : conserve une **TableRoutage**, stocke les DV voisins, envoie / reçoit les **MessageDV** et journalise les raisons de chaque changement.
— `table_routage.py` — classe **TableRoutage** : applique l'algorithme DV et fournit un diff pour le logger.
— `message_dv.py` — structure **MessageDV** : vecteur {dest → coût} et `taille_bits` proportionnelle au nombre d'entrées.

Au lancement (`t=0`) :

1. chaque routeur diffuse un DV initial à tous ses voisins ;
2. les évènements (réceptions de DV, `cost_change`, etc.) sont exécutés par ordre chronologique jusqu'à vidage de l'agenda ;
3. les tables finales `dest cost next-hop` sont imprimées pour chaque routeur.

2.2 Détails sur le code et la logique

Distance Vector Le calcul suit la formule classique

$$\mathrm{DV}[X](D) = \min_{v \in \mathcal{N}(X)} \big(\mathrm{Cost}(X, v) + \mathrm{DV}[v](D)\big).$$

Toute amélioration déclenche un nouvel envoi de DV via `Routeur.envoyer_dv`.

Prise en compte des délais Pour un DV de n entrées :

$$\Delta t = \frac{\text{distance}}{\text{vitesse\_prop}} + \frac{n \times 32 \text{ bits}}{\text{débit}},$$

calculé par `Lien.delai_transmission` puis converti en millisecondes avant planification.

Évènements de coût Un extrait typique de fichier JSON :

```
{
  "type": "cost_change",
  "time": 15000,
  "link": [1, 3],
  "new_cost": 12
}
```

déclenche `Simulateur._planifier_changement_cout` : le coût est modifié (ou $>$ `1e9` si lien coupé) puis chaque extrémité renvoie immédiatement un DV.

Arrêt de la simulation L'exécution se termine dès que l'agenda est vide ; il n'y a plus de compteur `MAX_DV_UPDATES`.

2.3 Difficultés rencontrées et état final

— **Horodatage précis** : injection du temps simulé dans le logger via l'adapteur interne `Adapt` de `simulateur.py`.
— **Count-to-infinity** : un scénario dédié illustre la divergence ; le log permet d'identifier la boucle.
— **Format JSON** : Les fichiers de scénarios respectent scrupuleusement le format JSON imposé par l'énoncé , y compris l'utilisation d'entiers pour toutes les grandeurs physiques spécifiées (vitesses, distances, coûts). De plus, `json.load` garantit le respect de la grammaire syntaxique.

État final : toutes les fonctionnalités du contrat sont opérationnelles ; les trois scénarios fournis convergent ou illustrent correctement le phénomène attendu, et les tables finales correspondent aux coûts administratifs minimaux.

3 Simulations

Nous avons testé trois scénarios :

3.1 Validation de l'implémentation

Topologie : 5 routeurs, 7 liens Pour valider notre implémentation, nous utilisons une topologie spécifiquement conçue à cet effet, comprenant 5 routeurs interconnectés par 7 liens. Les caractéristiques de ces liens (notamment leurs coûts administratifs) sont définies dans le fichier `scenario_validation.json` et sont les suivantes :

— (R_1, R_2) : coût 1
— (R_1, R_3) : coût 2
— (R_2, R_4) : coût 2
— (R_2, R_5) : coût 3
— (R_3, R_5) : coût 5
— (R_4, R_5) : coût 1
— (R_4, R_1) : coût 4

La vitesse de propagation pour tous les liens est de 2×10^8 m/s, la vitesse de transmission de 10^6 bps et la distance de 1000 m.

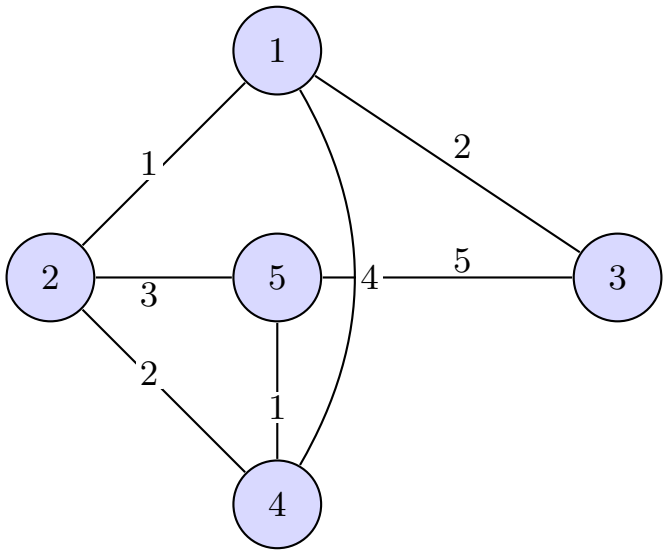


FIGURE 1 – Topologie de validation (5 routeurs, 7 liens) et coûts administratifs des liens.

Convergence et Tables de Routage Finales Le simulateur exécute l'algorithme Distance-Vector : les routeurs échangent initialement leurs vecteurs de distances avec leurs voisins directs. Chaque réception d'un vecteur de distances peut amener un routeur à mettre à jour sa propre table de routage si une meilleure route (coût plus faible) est découverte. Si sa table est modifiée, le routeur propage à son tour son nouveau vecteur de distances à ses voisins. Ce processus itératif se poursuit jusqu'à ce qu'aucun routeur n'ait plus de mise à jour à effectuer, indiquant que le système a convergé et que les routes les

plus courtes (selon les coûts administratifs) ont été établies. Les tables de routage finales obtenues par le simulateur pour cette topologie sont présentées ci-dessous :

TABLE 1 – Tables de routage finales (dump complet) après convergence pour `scenario_validation.json`

Routeur 1

dest	cost	nh
1	0	1
2	1	2
3	2	3
4	3	2
5	4	2

Routeur 2

dest	cost	nh
1	1	1
2	0	2
3	3	1
4	2	4
5	3	4

Routeur 3

dest	cost	nh
1	2	1
2	3	1
3	0	3
4	5	1
5	5	5

Routeur 4

dest	cost	nh
1	3	2
2	2	2
3	5	2
4	0	4
5	1	5

Routeur 5

dest	cost	nh
1	4	2
2	3	2
3	5	3
4	1	4
5	0	5

Analyse et Comparaison Manuelle des Résultats Les tables de routage finales sont obtenues après que le simulateur a exécuté l'algorithme Distance-Vector, permettant aux routeurs d'échanger leurs vecteurs de distances jusqu'à ce qu'un état stable (convergence) soit atteint, où plus aucune amélioration de route n'est possible. Pour valider ces résultats, nous les comparons aux routes optimales calculées manuellement en appliquant l'algorithme de Bellman-Ford ou par simple inspection des plus courts chemins sur le graphe pondéré de la topologie.

Prenons l'exemple de la détermination du chemin optimal de R_1 à R_5 :

- Chemin direct (s'il existait) : N/A.
- Via R_2 :
  - $R_1 \to R_2 \to R_5$: coût $\text{Cost}(1,2) + \text{Cost}(2,5) = 1 + 3 = 4$.
  - $R_1 \to R_2 \to R_4 \to R_5$: coût $\text{Cost}(1,2) + \text{Cost}(2,4) + \text{Cost}(4,5) = 1 + 2 + 1 = 4$.
- Via R_3 :
  - $R_1 \to R_3 \to R_5$: coût $\text{Cost}(1,3) + \text{Cost}(3,5) = 2 + 5 = 7$.
- Via R_4 (directement depuis R_1) :
  - $R_1 \to R_4 \to R_5$: coût $\text{Cost}(1,4) + \text{Cost}(4,5) = 4 + 1 = 5$.

Le calcul manuel montre que le coût minimal pour atteindre R_5 depuis R_1 est de 4. Il existe deux chemins de coût 4 : $R_1 \to R_2 \to R_5$ et $R_1 \to R_2 \to R_4 \to R_5$. La table de routage du simulateur pour R_1 indique bien un coût de 4 vers R_5, avec R_2 comme prochain saut ('next-hop'). Cela correspond à l'un des chemins optimaux identifiés.

Des vérifications similaires ont été effectuées pour d'autres paires source-destination, confirmant que les tables de routage générées par le simulateur correspondent aux routes de moindre coût déterminées manuellement. Le simulateur est donc jugé conforme sur ce scénario.

Détermination du chemin à partir des tables Pour reconstruire le chemin complet emprunté par un paquet de données d'un routeur source A à un routeur destination B en utilisant uniquement les tables de routage finales :

1. Consulter la table de routage du routeur actuel (initialement A). Chercher l'entrée pour la destination B et identifier le prochain saut (`next_hop`), notons-le N_1. Le chemin est pour l'instant $A \to N_1$.
2. Si N_1 est différent de B, consulter la table de routage de N_1. Chercher l'entrée pour la destination B et identifier le prochain saut, N_2. Le chemin devient $A \to N_1 \to N_2$.
3. Répéter ce processus, en passant au routeur `next_hop` et en consultant sa table, jusqu'à ce que le `next_hop` identifié soit la destination B elle-même.

C'est le principe classique du routage par sauts successifs (*next-hop forwarding*). Par exemple, pour le chemin de R_1 à R_5 (coût 4) :

- Table R_1 [dest R_5] → next-hop R_2. Chemin : $R_1 \to R_2$.
- Table R_2 [dest R_5] → next-hop R_4. Chemin : $R_1 \to R_2 \to R_4$.
- Table R_4 [dest R_5] → next-hop R_5. Chemin : $R_1 \to R_2 \to R_4 \to R_5$. La destination est atteinte.

3.2 Impact des délais

Topologie Le fichier `scenario_delays.json` modélise six routeurs reliés par sept liens. La longueur des liens est identique ($L = 1000\,\text{m}$) ainsi que la vitesse de propagation ($v_{\text{prop}} = 2{\times}10^8\,\text{m/s}$) ; en revanche, le débit (`transmission_speed`) et le coût administratif (`cost`) diffèrent d'un lien à l'autre :

— (R_1, R_2) : coût 1, débit 16 kbps
— (R_1, R_3) : coût 15, débit 20 kbps
— (R_2, R_4) : coût 3, débit 20 kbps
— (R_3, R_5) : coût 2, débit 20 kbps
— (R_4, R_5) : coût 1, débit 1 kbps
— (R_4, R_6) : coût 4, débit 1 kbps
— (R_5, R_6) : coût 1, débit 22 kbps

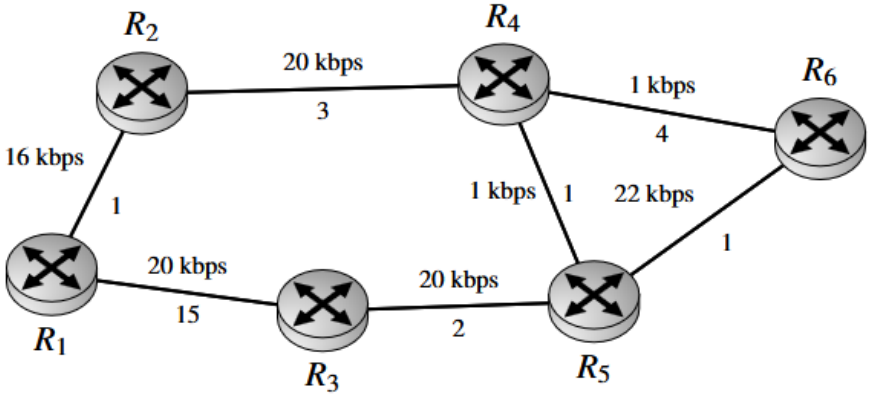


FIGURE 2 – Topologie du scénario `scenario_delays.json`.

Convention : le routeur A mentionné dans l'énoncé correspond ici au routeur R_1 de la figure 2.

```
% R1 démarre et envoie ses DV initiaux
@0.000 s Routeur 1 envoie DV a 2: {1: 0}        % lien 1-2 16kbps (rapide)
@0.000 s Routeur 1 envoie DV a 3: {1: 0}        % lien 1-3 20kbps (rapide)

% Les voisins directs R2 et R3 reçoivent rapidement l'info sur R1
@0.002 s Routeur 2 recoit DV de 1: {1: 0}
@0.002 s Routeur 3 recoit DV de 1: {1: 0}

% Propagation rapide de l'info sur R1 vers R4 via R2
@0.002 s Routeur 2 envoie DV a 4: {2:0, 1:1}    % R2 annonce R1 (cout 1) à R4
                                                % (lien 2-4 20kbps (rapide))
@0.005 s Routeur 4 recoit DV de 2: {2:0, 1:1, 4:3}% R4 apprend R1 via R2 (cout 1+3=4)

% Propagation de l'info sur R1 vers R5 via R3
@0.002 s Routeur 3 envoie DV a 5: {3:0, 1:15}   % R3 annonce R1 (cout 15) à R5
                                                % (lien 3-5 20kbps (rapide))
@0.005 s Routeur 5 recoit DV de 3: {3:0, 1:15}  % R5 apprend R1 via R3 (cout 2+15=17)

% R5 propage l'info sur R1 à R4 via le lien lent (R5-R4 @ 1kbps)
@0.005 s Routeur 5 envoie DV a 4: {5:0,1:17,3:2,6:1}% 4 entrées (128 bits)
@0.133 s Routeur 4 recoit DV de 5: {5:0,1:17,3:2,6:1}
```

Les logs ci-dessus illustrent la différence notable dans la chronologie de propagation des informations concernant le routeur R_1 vers le routeur R_4 :

— R_4 obtient une première information sur R_1 par le chemin rapide ($R_1 \to R_2 \to R_4$) dès `@0.005 s`. Le coût appris est $1 + 3 = 4$.
— L'information transitant par $R_1 \to R_3 \to R_5 \to R_4$ (lien lent R_5–R_4 @ 1 kbps) n'arrive qu'à `@0.133 s` avec un coût annoncé de $1 + 17 = 18$.

La différence d'horodatage (`0.005 s` vs `0.133 s`) met en évidence l'impact des débits sur la vitesse de propagation des mises à jour.

Justification numérique des délais La durée totale d'acheminement d'un paquet DV sur un lien est

$$\Delta t = \underbrace{\frac{\text{distance}}{v_{\text{prop}}}}_{\text{propagation}} + \underbrace{\frac{\text{taille(bits)}}{\text{débit(bps)}}}_{\text{transmission}}.$$

— **Paramètres communs.** Tous les liens mesurent $L = 1000\,\text{m}$ et $v_{\text{prop}} = 2{\times}10^8\,\text{m/s}$, d'où $\Delta t_{\text{prop}} = 5\,\mu\text{s}$ (négligeable). Le premier vecteur échangé (par R_1) ne contient qu'une entrée soit 32 bits.

— **Liens 16–22 kbps.**

$$\frac{32}{16\,000} \simeq 2.0\,\text{ms}, \qquad \frac{32}{22\,000} \simeq 1.5\,\text{ms}.$$

Avec la propagation : $\Delta t \approx 1.5$–$2.0\,\text{ms}$.

— **Liens 1 kbps.**

$$\Delta t_{\text{trans}} = \frac{32}{1000} = 32\,\text{ms}.$$

— **Rapport des délais.**

$$\frac{32\,\text{ms}}{2\,\text{ms}} \approx 16.$$

Un DV minimal se propage $\sim 16\times$ plus lentement sur un lien 1 kbps que sur un lien 16 kbps. Les horodatages relevés (`@0.002 s`, `@0.133 s`, etc.) confirment ces ordres de grandeur.

Effet sur la convergence Les écarts de délai retardent la convergence ($\approx 0.112\,\text{s}$: premier DV $\rightarrow$ dernier changement concernant la destination 1) mais n'influent pas la route finale, qui dépend uniquement des coûts cumulés.

TABLE 2 – Tables de routage finales — `scenario_delays.json` (R1–R6)

Routeur 1			**Routeur 2**			**Routeur 3**			**Routeur 4**			**Routeur 5**			**Routeur 6**		
dest	cost	nh	dest	cost	nh	dest	cost	nh	dest	cost	nh	dest	cost	nh	dest	cost	nh
1	0	1	1	1	1	1	7	5	1	4	2	1	5	4	1	6	5
2	1	2	2	0	2	2	6	5	2	3	2	2	4	4	2	5	5
3	7	2	3	6	4	3	0	3	3	3	5	3	2	3	3	3	5
4	4	2	4	3	4	4	3	5	4	0	4	4	1	4	4	2	5
5	5	2	5	4	4	5	2	5	5	1	5	5	0	5	5	1	5
6	6	2	6	5	4	6	3	5	6	2	5	6	1	6	6	0	6

Par exemple, pour joindre R_3, R_1 passe par R_2 (coût total $1 + 6 = 7$), indépendamment des latences.

Réponses aux questions de l'énoncé

1. **Les délais influencent-ils la propagation ?** Oui : ils modifient l'ordre et l'instant d'arrivée des DV.
2. **Influent-ils les routes finales ?** Non : la sélection du *next-hop* dépend uniquement des coûts administratifs.

Remarque pédagogique [1] Dans l'exemple théorique, on suppose que tous les voisins traitent *simultanément* le DV reçu ; le nombre d'itérations est donc minimal. Notre simulation, elle, tient compte des délais de propagation, de transmission et de mise en file : un même routeur peut être réactivé plusieurs fois, ce qui augmente le nombre total de messages. Le résultat final (routes optimales) reste inchangé — seuls les horodatages diffèrent.

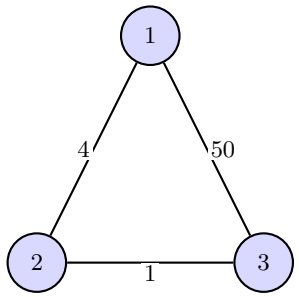
1. Adapté de la diapositive du cours Réseaux « Distance-Vector : example discussion », p. 181.

3.3 Comptage à l'infini

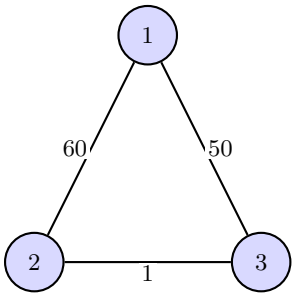
Explication fine du phénomène Dans un protocole *Distance Vector* (DV), chaque routeur annonce périodiquement à ses voisins le **coût minimal** dont il dispose pour joindre chaque destination. S'il apprend d'un voisin un chemin *apparemment* plus court, il met sa table à jour et relaie aussitôt cette nouvelle métrique :

$$\mathrm{DV}[X](D) = \min_{v \in \mathcal{N}(X)} \big\{ \mathrm{Cost}(X, v) + \mathrm{DV}[v](D) \big\}.$$

Quand un lien rompt ou voit son coût exploser, on s'attend à ce que la métrique vers la destination augmente d'un coup. Malheureusement, si deux voisins dépendent l'un de l'autre pour cette destination (par exemple R_2 atteint D via R_3 et R_3 via R_2), ils risquent de **se ré-annoncer en boucle** une route qui passe… par l'autre, chacun croyant que l'autre dispose encore d'un chemin valide et court. À chaque tour, la métrique progresse de quelques unités : c'est le *count-to-infinity*. Sans contre-mesure (split horizon, poison reverse, hold-down timer…), le coût peut diverger jusqu'à la valeur symbolisant l'infini (16 pour RIP) ou jusqu'à ce qu'une autre route (moins affectée) soit finalement préférée.



(a) Avant modification : après les premiers échanges, R_3 apprend un coût 5 vers R_1 via R_2 ($1 + 4 = 5$).



(b) Après : le lien 1–2 passe à 60.

FIGURE 3 – Topologie illustrant le comptage à l'infini, avant et après la hausse du coût du lien 1–2.

Topologie et scénario déclencheur Le fichier `scenario_count_to_infinity.json` contient :

- trois routeurs R_1, R_2, R_3 ;
- coûts initiaux : $\mathrm{Cost}(1,2) = 4$, $\mathrm{Cost}(2,3) = 1$, $\mathrm{Cost}(1,3) = 50$;
- à $t = 100\,\mathrm{ms}$ un événement `cost_change` fait passer $\mathrm{Cost}(1,2)$ de 4 à 60.

Chronologie détaillée de la convergence et du comptage

$t = 0$ **Convergence initiale.**

- R_1 connaît R_2 à 4 (direct) et R_3 à 50 (direct).
- R_2 connaît R_1 à 4 et R_3 à 1.
- R_3 connaît R_2 à 1 et R_1 à 50.
- Après échange : R_3 choisit $R_3 \to R_2 \to R_1$ ($1 + 4 = 5$) ; R_1 choisit $R_1 \to R_2 \to R_3$ ($4 + 1 = 5$).

$t = 100\,\mathrm{ms}$ **Hausse du coût du lien 1–2.**

- *Réaction de R_1.* Route alternative : $R_1 \to R_3 \to R_2 = 50 + 1 = 51$. Coût vers R_2 devient 51 (via R_3).
- *Réaction de R_2.* Avant de savoir que R_3 dépend de lui, R_2 voit R_3 annoncer R_1 à 5. Il calcule $1 + 5 = 6$ et passe son coût à 6 (via R_3).

Ping-pong des annonces. R_3 reçoit 6 → annonce 7 ; R_2 reçoit 7 → annonce 8 ; etc.

Stabilisation. Quand R_2 annonce 49, R_3 obtient $1 + 49 = 50$ et reprend son lien direct à 50. R_3 annonce 50, R_2 calcule $1 + 50 = 51$ et s'arrête.

```
@0.100s Link [1,2] cost 4 -> 60
@0.100s R2 cost to 1: 4 -> 6    % via R3
@0.100s R3 cost to 1: 5 -> 7    % via R2
@0.100s R2 cost to 1: 6 -> 8
@0.100s R3 cost to 1: 7 -> 9
...
```

```
@0.104s R2 cost to 1: 46 -> 48
@0.104s R3 cost to 1: 45 -> 47
@0.105s R3 cost to 1: 49 -> 50  % reprend le lien direct
@0.105s R2 cost to 1: 50 -> 51
```

Conséquences observées

- **Stabilisation des tables.** $R_3 \to R_1 = 50$ (lien direct) ; $R_2 \to R_3 \to R_1 = 51$.
- **Boucle transitoire.** Entre `0.100 s` et `0.105 s`, les paquets destinés à R_1 peuvent tourner entre R_2 et R_3.
- **Coût final augmenté** pour R_2 : $4 \to 51$.

3.4 Solution alternative au comptage à l'infini : le *Hold-Down Timer*

Le problème du comptage à l'infini (*count-to-infinity*) se produit lorsque deux routeurs s'échangent en boucle des informations de routage incorrectes, incrémentant progressivement le coût d'une route vers une destination désormais injoignable (ou devenue très coûteuse). Pour y remédier, on connaît plusieurs approches (split horizon, route poisoning, etc.). Nous allons détailler ci-dessous la technique des *hold-down timers*.

3.4.1 Principe du *Hold-Down Timer*

Le *hold-down timer* est un mécanisme qui, lorsqu'un routeur détecte qu'une route est soudainement injoignable (ou si son coût devient trop élevé), place cette route dans un état « hold-down ». Concrètement :

1. Le routeur constate qu'un lien ou une route devient invalide (coût infini ou considérablement accru).
2. Il invalide alors la route (ou la considère comme très coûteuse), et il enclenche un **chronomètre** (le *hold-down timer*).
3. Pendant la durée du *hold-down timer*, le routeur **rejette** (ou ignore) les annonces de ses voisins prétendant disposer d'une route attractive (plus courte) vers cette destination. L'idée est d'éviter de se fier aux voisins qui, eux aussi, auraient éventuellement appris cette route via ce routeur.
4. Au terme du *hold-down timer*, le routeur vérifie s'il a reçu des informations de routage fiables, ou s'il doit continuer à annoncer la destination comme injoignable (coût = ∞).

Cette période de gel (*hold-down*) empêche la propagation réciproque de routes invalides. Ainsi, lorsque le voisin s'appuie en réalité sur un chemin qui repasse par le routeur lui-même (boucle), l'annonce est ignorée durant le *hold-down*, et la boucle n'a pas l'occasion de faire gonfler le coût à l'infini.

3.4.2 Avantages et limites

Avantages

- **Brise les boucles de mise à jour** : en empêchant temporairement la réintroduction d'une route soudainement marquée invalide, on évite l'effet « ping-pong ».
- **Implémentation relativement simple** : on ajoute un champ de « compteur de temps » par route problématique, sans devoir réécrire toute la logique DV.
- **Moins de messages erronés** : les voisins ne peuvent pas reconvaincre le routeur d'accepter la route invalidée pendant cette fenêtre temporelle.

Limites

- **Délai d'acceptation d'alternatives légitimes** : si une routeur *valide* pour la destination existe réellement, le *hold-down timer* va parfois retarder la prise en compte de cette nouvelle route, augmentant le temps de convergence.
- **Paramétrage délicat** : un *hold-down timer* trop court n'empêche pas le bouclage, trop long ralentit globalement la convergence.

3.4.3 Exemple d'utilisation

Considérons la même topologie minimaliste où un lien se coupe soudainement (coût passe à ∞). Sans *hold-down timer*, nous avons vu que deux routeurs (A et B) s'échangent leurs routes et s'incrémentent mutuellement le coût, conduisant à un « comptage à l'infini ».

Avec un *hold-down timer* :

1. Routeur A détecte la rupture. Il invalide la destination (ex. route X) et démarre un *hold-down timer* (par ex. 30 secondes).
2. Pendant ces 30 secondes, A **refuse** toute annonce d'un voisin prétendant avoir un meilleur chemin vers X, car A soupçonne que l'information « plus courte »découle de sa propre route désormais invalide.
3. Si la route redevient valide (via un autre itinéraire) ou si la situation se confirme, A ajuste définitivement ses annonces au terme du *hold-down*. Ainsi, B ne peut pas « réintroduire » la route défaillante chez A en la lui renvoyant, brisant ainsi la boucle.

Le *hold-down timer* est donc une **solution partielle** et complémentaire à d'autres (tel que *split horizon*) pour éviter le phénomène de *count-to-infinity*. Dans la plupart des implémentations de *Distance Vector*, on utilise un mélange de techniques (hold-down timers, poison reverse, etc.) pour améliorer la robustesse et accélérer la convergence.

4 Conclusion

Dans l'ensemble, notre implémentation du protocole *Distance Vector* répond pleinement aux objectifs fixés. La structure en classes permet de modéliser finement les **routeurs**, leurs **tables de routage**, et les **liens** qui relient ces derniers, tout en gérant de manière réaliste les **délais de propagation et de transmission**. Grâce à ce simulateur :

- Nous avons **validé** notre approche sur une topologie de test simple (5 routeurs, 7 liens). Les résultats obtenus sont cohérents avec le calcul manuel, attestant du bon fonctionnement du calcul de routes et de la mise à jour asynchrone des DV.
- Nous avons **étudié l'impact des délais** en observant un scénario où la vitesse de transmission varie sensiblement entre les liens. Cela a montré que si ces délais influent sur la *chronologie* de la convergence, ils n'affectent pas la *qualité* finale des routes (les coûts administratifs restent déterminants).
- Nous avons enfin **mis en évidence le problème de comptage à l'infini**, typique des protocoles à vecteur de distance, en démontrant l'inflation sans borne des coûts lorsque deux routeurs se renvoient mutuellement une route invalide. Nous avons discuté d'une *solution alternative* au poison reverse (le *hold-down timer*), capable de limiter ou d'éviter ce phénomène.

Ainsi, notre simulateur illustre **concrètement** le principe du routage par vecteur de distances, sa **facilité** de mise en œuvre, mais aussi ses **défauts inhérents**, parmi lesquels la lenteur de convergence et le risque de « compter à l'infini ». Les *logs* produits, ainsi que l'ensemble des scénarios menés, fournissent une vision très claire du fonctionnement et de la dynamique de ce protocole : à travers ces expériences, nous avons acquis une compréhension approfondie du *Distance Vector* et de ses mécanismes de mise à jour, tout en mesurant l'importance des techniques visant à améliorer sa robustesse.

Références

[1] C. Hedrick, *Routing Information Protocol*, RFC 1058, November 1988.

[2] G. Malkin, *RIP Version 2*, RFC 2453, November 1998.

[3] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5^{e} éd., Pearson, 2011.

[4] J. F. Kurose and K. W. Ross, *Computer Networking : A Top-Down Approach*, 7^{e} éd., Pearson, 2016.

[5] Cisco Systems, *Implementing IP Routing (ROUTE) Foundation Learning Guide*, Cisco Press, 2006.