Rapport - Devoir 2 : Protocole de Liaison de Données

Équipe : Nomena Andre Willis 20213162

1. Conception du Protocole

Format de la Trame

Définition d'une structure de trame inspirée de HDLC :
`[FLAG | TYPE (1B) | SEQ (4B) | LEN (2B) | DATA (N) | CRC (4B) | FLAG]`

- **FLAG** : `0x7E` (01111110) pour délimiter le début et la fin.
- **TYPE** : 0 pour Données, 1 pour ACK.
- **SEQ** : Numéro de séquence sur 4 octets (entier non signé) pour gérer l'ordre et les pertes.
- **LEN** : Longueur des données utiles (2 octets).
- **DATA** : Charge utile (max 100 octets).
- **CRC** : CRC-32 sur l'en-tête et les données pour la détection d'erreurs.

Mécanisme de Contrôle : Go-Back-N

Choisi **Go-Back-N** pour sa simplicité d'implémentation par rapport à Selective Repeat.

- **Taille de fenêtre** : 5. Cela permet d'envoyer plusieurs trames sans attendre (pipeline) tout en restant simple à gérer.
- **Timeout** : Configurable (0.5s par défaut). Suffisant pour couvrir le RTT simulé tout en réagissant aux pertes.

Détection d'Erreurs

- **CRC-32** : Choisi pour sa robustesse standard (utilisé dans Ethernet, HDLC, etc.) et sa faible probabilité de collision par rapport à CRC-16.
- **Bit-stuffing** : Insertion d'un `0` après cinq `1` consécutifs dans le corps de la trame (après calcul du CRC) pour assurer la transparence des données vis-à-vis du Flag.

2. Émulation du Canal

Le module `canal.py` simule :

- **Erreurs de bits** : Inversion aléatoire de bits selon `probErreur`.
- **Pertes** : Suppression de trames selon `probPerte`.
- **Délai** : Retard aléatoire (`0` à `delaiMax`) implémenté via des `threading.Timer`.

3. Résultats Expérimentaux

Scénario 1 : Canal Parfait

- **Paramètres** : Err=0, Perte=0, Délai=0
- **Résultats** :
  - Frames envoyées : 49
  - Frames retransmises : 0
  - ACK reçus : 49
  - Durée : 2.16 s
- **Interprétation** : Fonctionnement nominal. Aucune perte ni erreur.

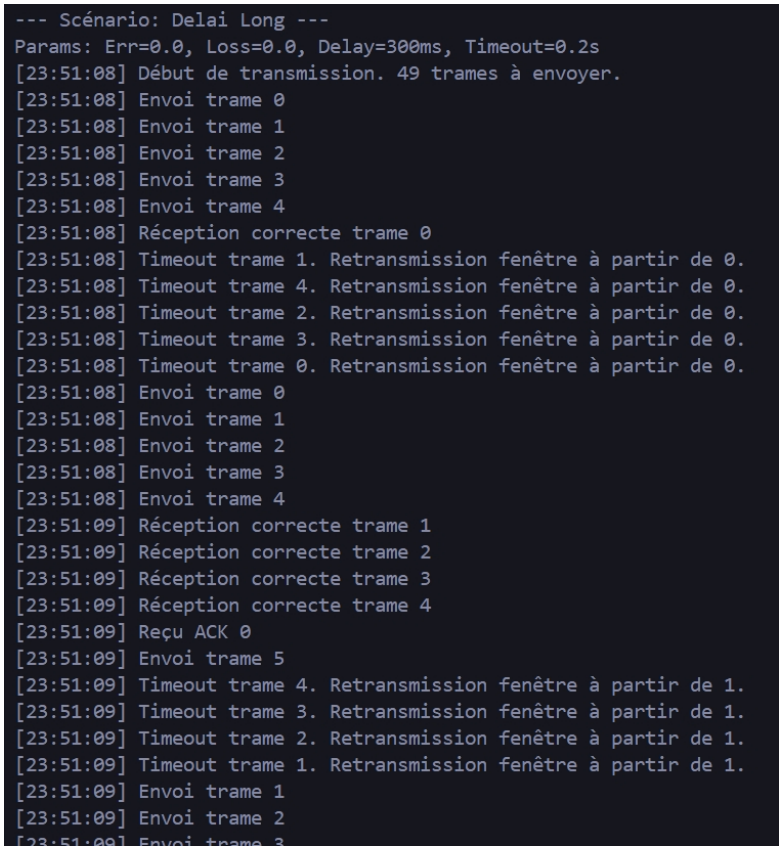
Scénario 2 : Canal Bruité

- **Paramètres** : Err=0.05, Perte=0.1, Délai=200ms
- **Résultats** :
  - Frames envoyées : 103
  - Frames retransmises : 25
  - ACK reçus : 79
  - Durée : 8.88 s
- **Interprétation** : Le taux d'erreur et de perte force des retransmissions, mais le protocole récupère efficacement.

Scénario 3 : Canal Instable

- **Paramètres** : Err=0.10, Perte=0.15, Délai=300ms
- **Résultats** :
  - Frames envoyées : 138
  - Frames retransmises : 48
  - ACK reçus : 86
  - Durée : 11.69 s

- **Interprétation** : Conditions difficiles. Le protocole surmonte les pertes et erreurs fréquentes grâce aux retransmissions.

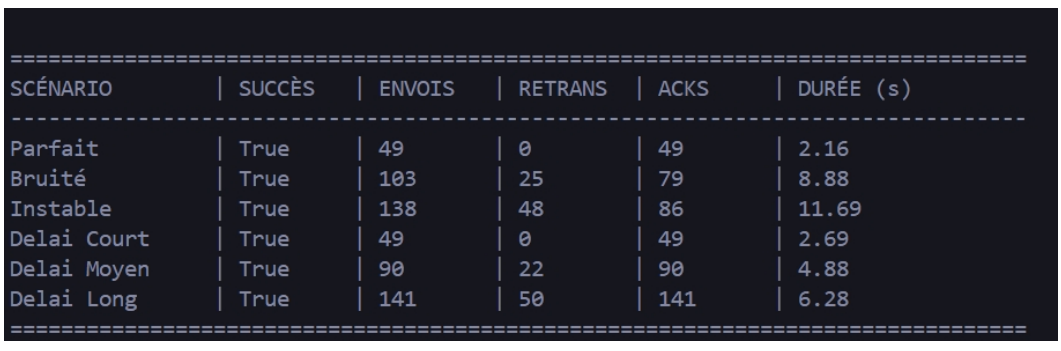


```
--- Scénario: Delai Long ---
Params: Err=0.0, Loss=0.0, Delay=300ms, Timeout=0.2s
[23:51:08] Début de transmission. 49 trames à envoyer.
[23:51:08] Envoi trame 0
[23:51:08] Envoi trame 1
[23:51:08] Envoi trame 2
[23:51:08] Envoi trame 3
[23:51:08] Envoi trame 4
[23:51:08] Réception correcte trame 0
[23:51:08] Timeout trame 1. Retransmission fenêtre à partir de 0.
[23:51:08] Timeout trame 4. Retransmission fenêtre à partir de 0.
[23:51:08] Timeout trame 2. Retransmission fenêtre à partir de 0.
[23:51:08] Timeout trame 3. Retransmission fenêtre à partir de 0.
[23:51:08] Timeout trame 0. Retransmission fenêtre à partir de 0.
[23:51:08] Envoi trame 0
[23:51:08] Envoi trame 1
[23:51:08] Envoi trame 2
[23:51:08] Envoi trame 3
[23:51:08] Envoi trame 4
[23:51:09] Réception correcte trame 1
[23:51:09] Réception correcte trame 2
[23:51:09] Réception correcte trame 3
[23:51:09] Réception correcte trame 4
[23:51:09] Reçu ACK 0
[23:51:09] Envoi trame 5
[23:51:09] Timeout trame 4. Retransmission fenêtre à partir de 1.
[23:51:09] Timeout trame 3. Retransmission fenêtre à partir de 1.
[23:51:09] Timeout trame 2. Retransmission fenêtre à partir de 1.
[23:51:09] Timeout trame 1. Retransmission fenêtre à partir de 1.
[23:51:09] Envoi trame 1
[23:51:09] Envoi trame 2
[23:51:09] Envoi trame 3
```

Scénario 4 : Influence du Délai (Timeout = 200ms)

Scénario	Délai Max	Frames Envoyées	Retransmissions	Durée
Court	50 ms	49	0	2.69 s
Moyen	180 ms	90	22	4.88 s
Long	300 ms	141	50	6.28 s

- **Interprétation** :
  - **Délai Court (50ms < 200ms)** : **0 retransmission**. C'est le comportement attendu. Le RTT est toujours inférieur au timeout.
  - **Délai Moyen (180ms ~ 200ms)** : Quelques retransmissions (22). Le délai aléatoire dépasse parfois le timeout.
  - **Délai Long (300ms > 200ms)** : Nombreuses retransmissions (50) car le timeout expire systématiquement avant l'arrivée de l'ACK.



```
================================================================================
SCÉNARIO        | SUCCÈS   | ENVOIS   | RETRANS  | ACKS     | DURÉE (s)
--------------------------------------------------------------------------------
Parfait         | True     | 49       | 0        | 49       | 2.16
Bruité          | True     | 103      | 25       | 79       | 8.88
Instable        | True     | 138      | 48       | 86       | 11.69
Delai Court     | True     | 49       | 0        | 49       | 2.69
Delai Moyen     | True     | 90       | 22       | 90       | 4.88
Delai Long      | True     | 141      | 50       | 141      | 6.28
================================================================================
```

```
PS C:\Users\willi\OneDrive\Bureau\New folder\code> python .\stuffing.py
Input:     01111110111110111111110111110
Stuffed:   011111010111110011111010111110 0
Expected:  011111010111110011111010111110 0
Destuffed: 01111110111110111111110111110
```

4. Bit-stuffing

Le test de validation a été effectué.

- **Input** : `01111110111110111111110111110`
- **Output (Algorithme)** : `011111010111110011111010111110 0`
- **Note** : L'algorithme insère un `0` après chaque suite de cinq `1`. Le dé-stuffing restitue parfaitement la chaîne originale..

5. Discussion

1. **Importance de l'Ordre** : Nous avons observé que si le canal ne préserve pas l'ordre (FIFO), le protocole Go-Back-N (qui attend des séquences strictes) échoue massivement, provoquant des boucles de retransmission. L'utilisation d'une file prioritaire dans la simulation a résolu ce problème.
2. **Perte d'ACK** : Lorsqu'un ACK est perdu, l'émetteur retransmet après timeout. Le récepteur détecte le doublon et renvoie l'ACK.
3. **Optimisations possibles** :
    - **Selective Repeat** : Pour éviter de retransmettre toute la fenêtre en cas d'erreur unique.
    - **Timeout Adaptatif** : Ajuster le timeout en fonction du RTT mesuré (comme TCP) pour gérer les délais variables (Scénario "Moyen").

```
[23:51:08] Envoi trame 0
[23:51:08] Envoi trame 1
[23:51:08] Envoi trame 2
[23:51:08] Envoi trame 3
[23:51:08] Envoi trame 4
[23:51:09] Réception correcte trame 1
[23:51:09] Réception correcte trame 2
[23:51:09] Réception correcte trame 3
[23:51:09] Réception correcte trame 4
[23:51:09] Reçu ACK 0
[23:51:09] Envoi trame 5
[23:51:09] Timeout trame 4. Retransmission fenêtre à partir de 1.
[23:51:09] Timeout trame 3. Retransmission fenêtre à partir de 1.
[23:51:09] Timeout trame 2. Retransmission fenêtre à partir de 1.
[23:51:09] Timeout trame 1. Retransmission fenêtre à partir de 1.
[23:51:09] Envoi trame 1
[23:51:09] Envoi trame 2
[23:51:09] Envoi trame 3
[23:51:09] Envoi trame 4
[23:51:09] Envoi trame 5
[23:51:09] Trame dupliquée 0. Renvoi ACK.
[23:51:09] Trame dupliquée 1. Renvoi ACK.
[23:51:09] Trame dupliquée 2. Renvoi ACK.
[23:51:09] Trame dupliquée 3. Renvoi ACK.
[23:51:09] Trame dupliquée 4. Renvoi ACK.
[23:51:09] Réception correcte trame 5
[23:51:09] Trame dupliquée 1. Renvoi ACK.
[23:51:09] Reçu ACK 1
[23:51:09] Reçu ACK 2
[23:51:09] Envoi trame 6
[23:51:09] Envoi trame 7
[23:51:09] Reçu ACK 3
[23:51:09] Reçu ACK 4
```