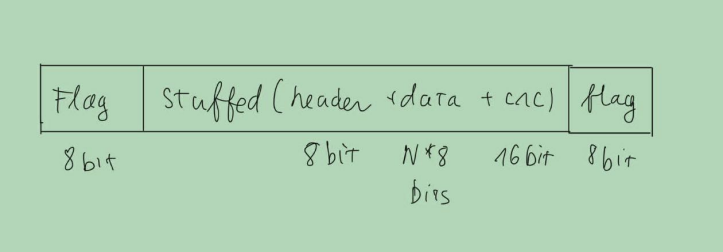
Rapport Devoir 2 - IFT 3325

Auteurs : Paul Gayout - 20264358, Trung Nguyen - 20238006

1 Conception du protocole

1. Format de la trame



Justification :
Flag : délimiter le début et la fin d'une trame. **Header** : numéroter les trames. **Data** : contient le message qu'on veut transmettre (avec N étant le nombre de caractères du message). **CRC** : CRC pour détection d'erreur. Les champs header + data + crc vont être "stuffed" aussi

2. Mécanisme de contrôle
On a choisit **Go-Back-N** parce que c'est simple à coder : un seul timer, pas de tamponnage complexe côté réception. **Taille de fenêtre** = 4 pour bien surveiller les cibles des simulations. **Timeout** = 250 ms - suffisamment long pour éviter des retransmissions inutiles, suffisamment court pour réagir à une perte.

3. Mécanisme de contrôle
Je choisit CRC-16 comme c'est suffisant pour détecter la plupart des erreurs simples et burst < 16 bits.

2 Implémentation du protocole

Nous avons implémenté le programme avec le langage Python (version 3.11.12). Voici le flôt l'entrée - sortie du programme avec un message de < 1 Ko

```
Trung@Trung MINGW64 /e/devoir2_IFT3325_PaulGayout_TrungNguyen (master)
$ E:/devoir2_IFT3325_PaulGayout_TrungNguyen/.venv/Scripts/python.exe e:/devoir2_IFT3325_PaulGayout_TrungNguyen/code/protocole.py
[16:59:47.549] Sending frame 0
[16:59:47.549] Sending frame 1
[16:59:47.550] ACK received for 0
[16:59:47.550] ACK received for 1
[16:59:47.551] Transmission finished.
[16:59:47.551] Frames sent: 2
[16:59:47.551] Frames retransmitted: 0
[16:59:47.551] ACKs received: 2
[16:59:47.551] Total time: 0.002s
(devoir2_IFT3325_PaulGayout_TrungNguyen)
```

Avec un système de ACK cumulatifs (si on reçoit un ACK pour 40, cela signifie qu'on a recu tous les ACK jusqu'à 40, même si on n'a pas vu de message ACK pour 39)

3 Test expérimentaux

Pour les tests ci-dessous, nous avons utilisé un message de 5 Ko

1. **Canal parfait** (probErreur = 0, probPerte = 0, gbnTimeout = 250 ms)

```
[16:04:20.492] Transmission finished.
[16:04:20.492] Frames sent: 51
[16:04:20.494] Frames retransmitted: 0
[16:04:20.494] ACKs received: 51
[16:04:20.494] Total time: 0.047s
(devoir2_IFT3325_PaulGayout_TrungNguyen)
```

Pour ce canal, la transmission de messages est très rapide, car il n'y a rien qui obstrue le flôt

2. **Canal bruité** (probErreur ≈ 0.05, probPerte ≈ 0.1, gbnTimeout = 250 ms)

```
[16:09:22.977] Transmission finished.
[16:09:22.977] Frames sent: 52
[16:09:22.977] Frames retransmitted: 1
[16:09:22.977] ACKs received: 42
[16:09:22.977] Total time: 0.297s
(devoir2_IFT3325_PaulGayout_TrungNguyen)
```

Pour ce canal, il y a des chances d'obstruction, mais cest relativement petit, donc c'est aussi vite

3. **Canal instable** (probErreur ≈ 0.10, probPerte ≈ 0.15, delaiMax = 300 ms, gbnTimeout = 280 ms)

```
[16:03:11.108] Transmission finished.
[16:03:11.108] Frames sent: 173
[16:03:11.108] Frames retransmitted: 122
[16:03:11.108] ACKs received: 39
[16:03:11.108] Total time: 17.393s
[16:03:11.299] Ignored duplicate ACK for 49
(devoir2_IFT3325_PaulGayout_TrungNguyen)
```

Pour ce canal, le délai maximum est plus grand que le timeout, donc parfois les ACKs arrivent plus tard que le timeout. Ce qui va beaucoup ralentir la transmission de messages.

4. **Influence du délai sur les temporisations**

probErreur et probPerte vont être 0 pour ces tests, on change seulement le delaiMax. Et on fixe timeout = 200 ms

— **delaiMax = 50 ms**

```
[16:23:51.544] Transmission finished.
[16:23:51.544] Frames sent: 51
[16:23:51.544] Frames retransmitted: 0
[16:23:51.544] ACKs received: 51
[16:23:51.544] Total time: 1.304s
(devoir2_IFT3325_PaulGayout_TrungNguyen)
```

Le programme est très vite comme on a peu de delaiMax.

— **delaiMax = 180 ms**

```
[16:24:30.851] Transmission finished.
[16:24:30.851] Frames sent: 51
[16:24:30.851] Frames retransmitted: 0
[16:24:30.851] ACKs received: 51
[16:24:30.851] Total time: 4.157s
(devoir2_IFT3325_PaulGayout_TrungNguyen)
```

Le programme est un peu plus lent comme on a plus de delaiMax

— **delaiMax = 300 ms**

```
[16:55:51.010] Transmission finished.
[16:55:51.010] Frames sent: 504
[16:55:51.010] Frames retransmitted: 453
[16:55:51.010] ACKs received: 24
[16:55:51.010] Total time: 28.431s
[16:55:51.199] Ignored duplicate ACK for 21
(devoir2_IFT3325_PaulGayout_TrungNguyen)
```

Pour ce test, nous avons dû implémenter un mécanisme de retryMax = 10 pour chaque trame comme le delaiMax est beaucoup plus grand que timeout. Et on voit beaucoup de retransmission, trames envoyées pour ce test. Donc on peut conclure que ca prend énormement de temps (même impossible) pour le transmission du message dans ce cas

4 Test de validation du bit-stuffing

```
$ E:/devoir2_IFT3325_PaulGayout_TrungNguyen/.venv/Scripts/python.exe e:/devoir2_IFT3325_PaulGayout_TrungNguyen/code/stuffing.py
Original bits:      01111110111110111110111110
After stuffing:     011111010111110011111010111100
After destuffing:   01111110111110111110111110
CRC(original):      1011111000010010
CRC(destuffed):     1011111000010010
test 1 passed: Everything works accordingly

Original bits:      01111110111110111110111110
After stuffing:     011111010111110011111010111100
After destuffing:   01111110111110111110111110
After corruption:   00111110111110111110111110 (in destuffed seq)
CRC(original):      1011111000010010
CRC(destuffed):     0101001101111010
test 2 passed: CRC detected bit flip
(devoir2_IFT3325_PaulGayout_TrungNguyen)
```

5 Discussion

— **Réaction aux ACK perdus**

Notre programme va faire un timeout et essayer de retransmettre les trames

```
[17:12:19.309] Sending frame 0
[17:12:19.310] Sending frame 1
[17:12:19.310] ACK received for 0
[17:12:19.532] Timeout! Retransmitting window starting at 1
[17:12:19.532] Resending frame 1 (retry 1)
[17:12:19.747] Timeout! Retransmitting window starting at 1
[17:12:19.747] Resending frame 1 (retry 2)
[17:12:19.747] ACK received for 1
[17:12:19.747] Transmission finished.
```

— **Différences avec le comportement théorique**

Premièrement, notre structure de trame est beaucoup plus simple que celui de HDLC (moins de champs et la majorité des opérations de notre programme est sur le numéro de séquence de la trame)

— **Optimisation de la fiabilité**

Une optimisation qu'on considère est l'implémentation de timeout adaptif qui prend en compte le RTT et calcule un timeout approprié au RTT et le delaiMax du canal. Ce qui peut éviter la boucle infinie dans le cas où delaiMax est plus grand que timeout