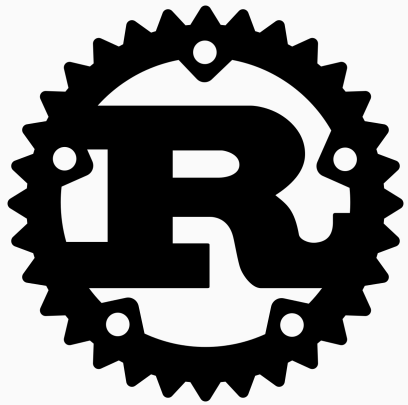
Fearless and blazingly fast async HDLC

Etienne Collin, Emeric Laberge

3 décembre 2024

Plan de la présentation

Architecture
Implémentation
Client
Server
Utils
CRC
Conclusion

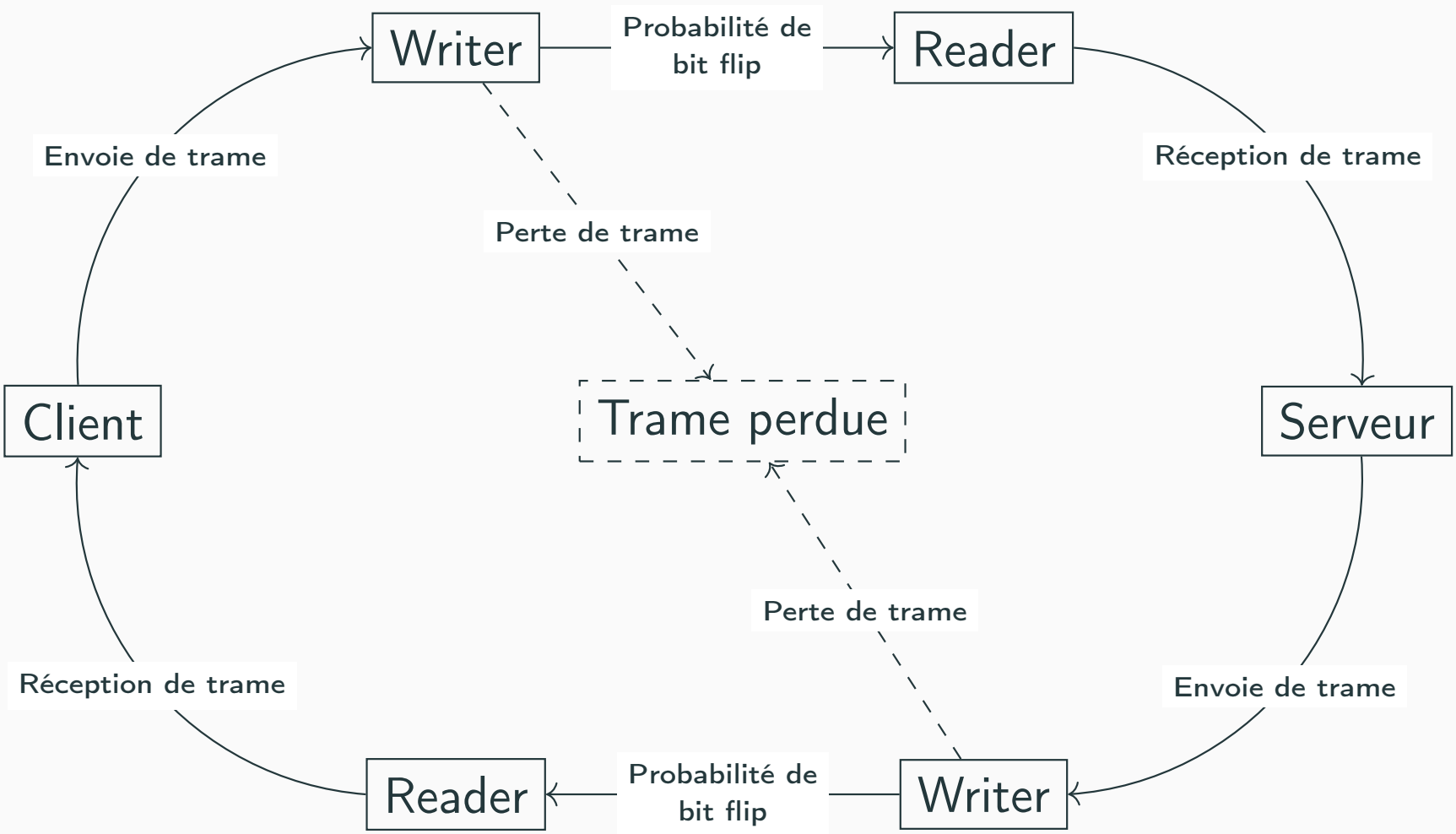


Architecture

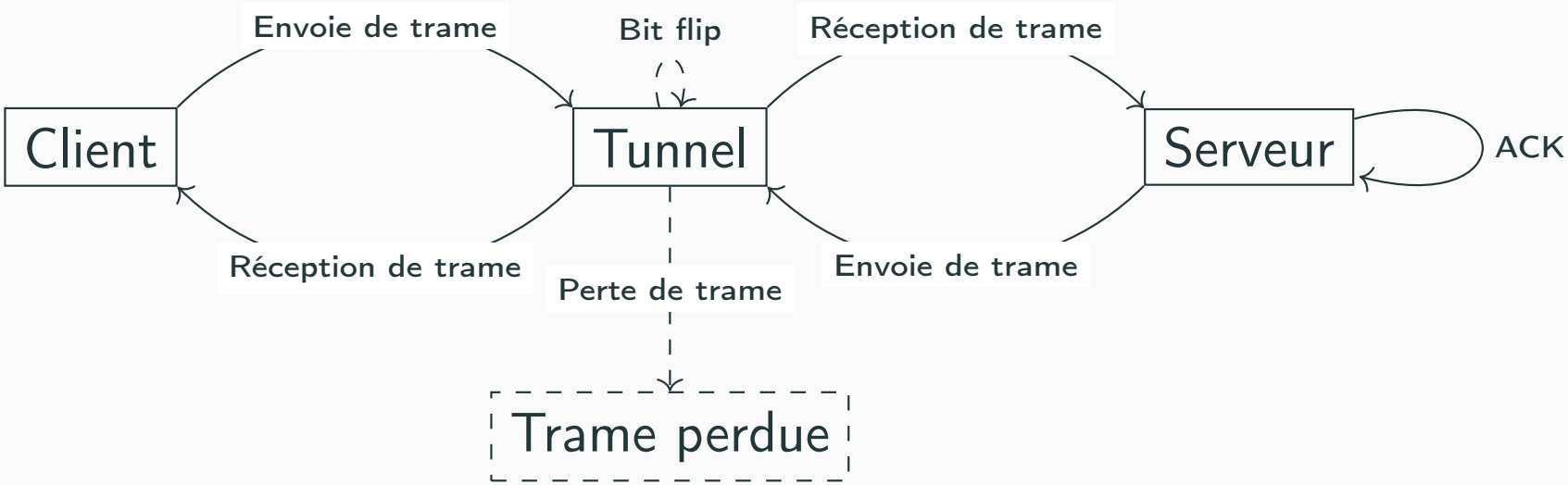
Architecture du Projet

- Client
  - Reader
  - Writer
  - Send File
- Server
  - Reader
  - Writer
  - Assembler
- Tunnel ?

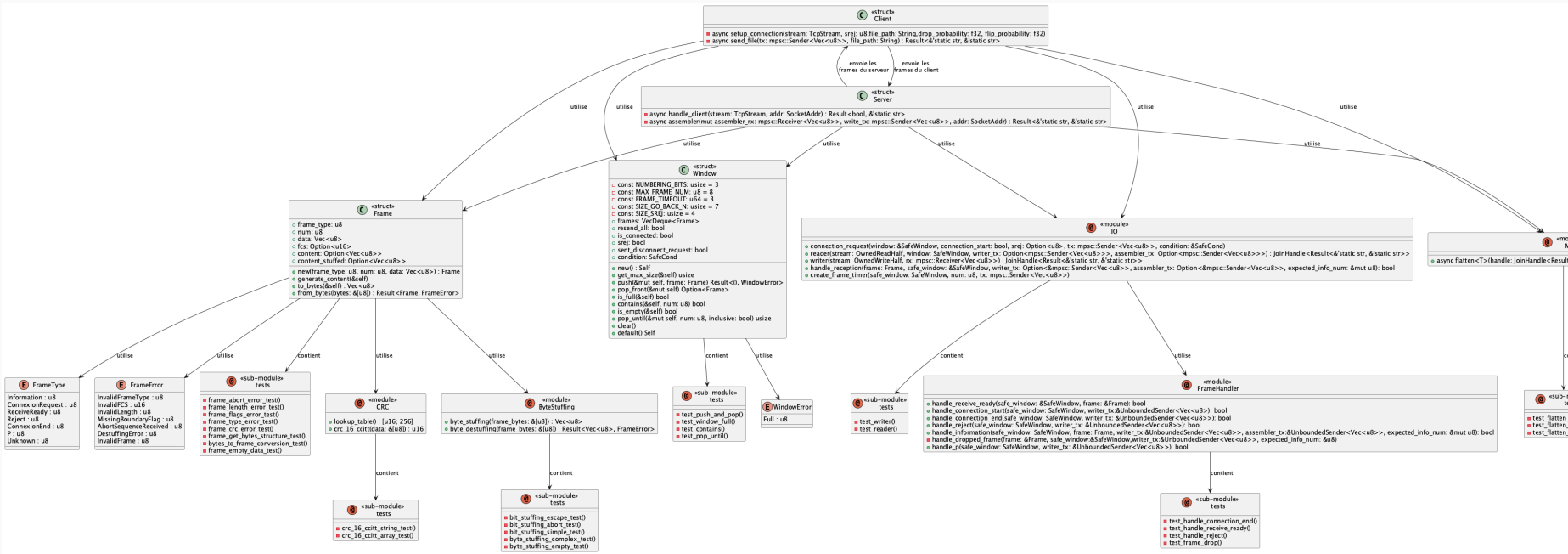
Architecture sans Tunnel



Architecture avec Tunnel



Visualisation détaillée de l'architecture



Implémentation

Implémentation

- Rust
  - Memory Safe (même sans GC)
  - Press Release : Future Software Should Be Memory Safe
  - FAANG, Microsoft, Linux, NSA
- Design Asynchrone
  - Synchronisation
  - MPSC (Multi-Producer Single Consumer)
  - Thread Safe
- Utilisation directe des bytes
  - Versus string (espace x8-x32)

Client

Fonctionnalités du client

- Établissement de la connexion
- Séparation des données en trames
- Utilisation des utils
- Gestion de la fenêtre d'envoi
- Fermeture de la connexion

Séparation des données en trames

Voici le format des trames que nous avons utilisé :

Flag	**Type**	**Num**	**Données**	**CRC**	**Flag**
1 octet	1 octet	1 octet	1 à 64Ko	2 octets	1 octet

Il est important de noter que les données ajoutées par le byte stuffing sont incluses dans le champ données.

Scénario :

Taille des données à envoyer : 128Ko

Taille des données après byte stuffing : 140Ko

1. Première trame : contient les 64 premiers Ko
2. Deuxième trame : contient les 64 Ko suivants
3. Troisième trame : contient les 12 Ko restants

Server

Fonctionnalités du Récepteur

Les fonctionnalités du récepteur sont les suivantes :

- Établissement de la connexion
- Utilisation des utils
- Assemblage des trames reçues
- Fermeture de la connexion

Utils

Utils

Design modulaire !

Fonctionnalités

- Gestion des trames
  - Création
  - Conversion bytes $\rightarrow$ trame
    - Vérification intégrité
  - Conversion trame $\rightarrow$ bytes
    - Calcul CRC et byte stuffing
- Gestion de la fenêtre
  - Création
  - Push & Pop
  - Temporisateurs async
  - Condvar

Fonctionnalités

- Envoi des trames
  - Byte stuffing
  - Byte destuffing
  - Attente de l'acquittement
  - Trames perdues
  - Trames corrompues
- Réception et traitement des trames
  - Vérification de l'intégrité des données
  - Gestion des erreurs ou des pertes de trames
  - Gestion de la fenêtre de réception
  - Envoi d'acquittements

Gestion de la fenêtre de réception

- `NUMBERING_BITS` : Nombre de bits pour numéroter les trames
  - ☞ `NUMBERING_BITS` $\rightarrow 3$
- `MAX_FRAME_NUM` : Valeur maximale du numéro d'une trame
  - ☞ `MAX_FRAME_NUM` $\rightarrow 1 \ll$ `NUMBERING_BITS` ou $2^{\texttt{NUMBERING\_BITS}}$
- `SIZE_GO_BACK_N` : Taille maximale de la fenêtre pour le protocole Go-Back-N
  - ☞ `SIZE_GO_BACK_N` $\rightarrow (1 \ll$ `NUMBERING_BITS`$) - 1$

CRC

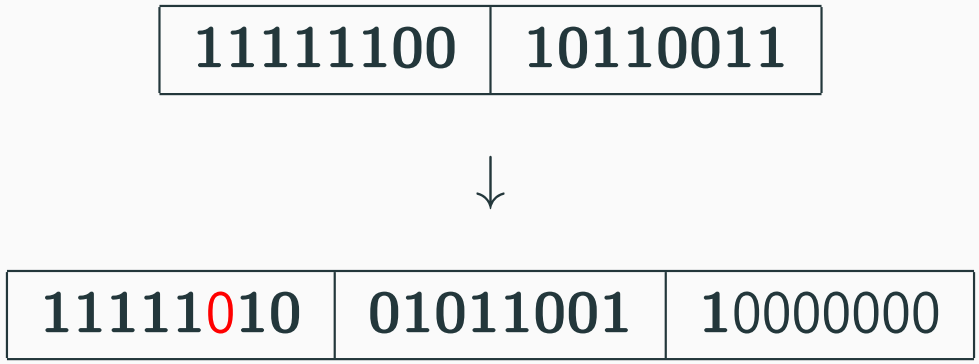
Implémentation

- LUT
  - Systèmes embarqués
  - Rapide ! ! !
  - Taille en mémoire 16x16

Implémentation du CRC

```rust
/// CRC-16 CCITT implementation.
/// This function computes the CRC-16 CCITT checksum for the given data.
pub fn crc_16_ccitt(data: &[u8]) -> PolynomialSize {
    let mut remainder = INITIAL_VALUE;
    // For each byte in the data, find its corresponding value
    // in the lookup table and XOR it with the remainder.
    // Then, shift the remainder 8 bits to the left.
    data.iter().for_each(|byte| {
        // Isolate the upper byte of the current remainder
        let processed_byte = *byte ^ (remainder >> (POLYNOM_WIDTH - 8)) as u8;
        // XOR the remainder with the value from the lookup table
        remainder = LUT[processed_byte as usize] ^ (remainder << 8);
    });
    remainder ^ FINAL_XOR
}
```

Problème du bit stuffing



On ne sait pas quels bits sont ajoutés comme padding (sans modifier la structure de la trame) !

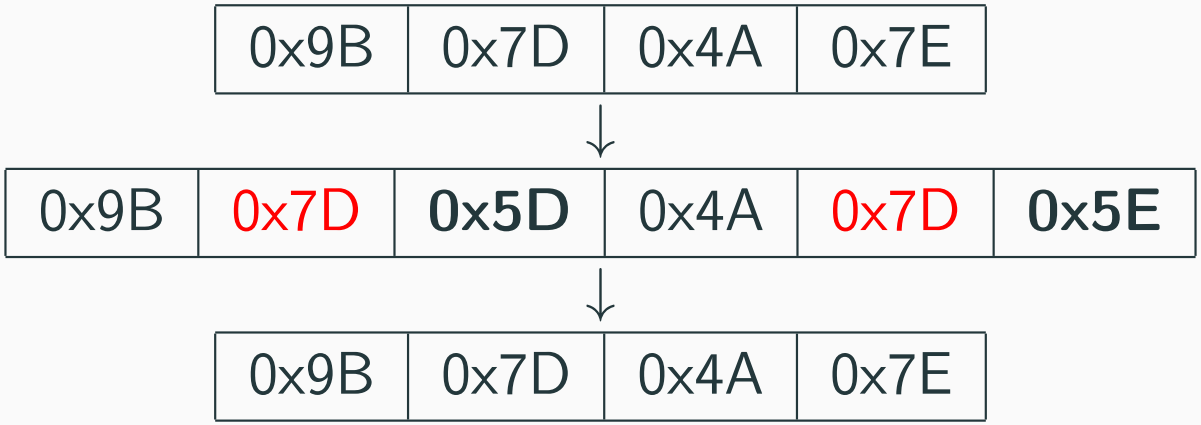
Comment byte stuffing règle cela

- ESCAPE_FLAG $\rightarrow$ **0x7D**
- BOUNDARY_FLAG $\rightarrow$ **0x7E**
- REPLACEMENT_BYTE $\rightarrow$ **0x20**

bits	bytes
10011011 01111101 01001010 01111110	0x9B 0x7D 0x4A 0x7E

Byte stuffing :

bytes	bytes après stuffing
ESCAPE_FLAG	ESCAPE_FLAG ESCAPE_FLAG $\oplus$ REPLACEMENT_BYTE
BOUNDARY_FLAG	ESCAPE_FLAG BOUNDARY_FLAG $\oplus$ REPLACEMENT_BYTE



Conclusion

Conclusion

- Optimisations
- Ajout de fonctionnalités

Questions ?