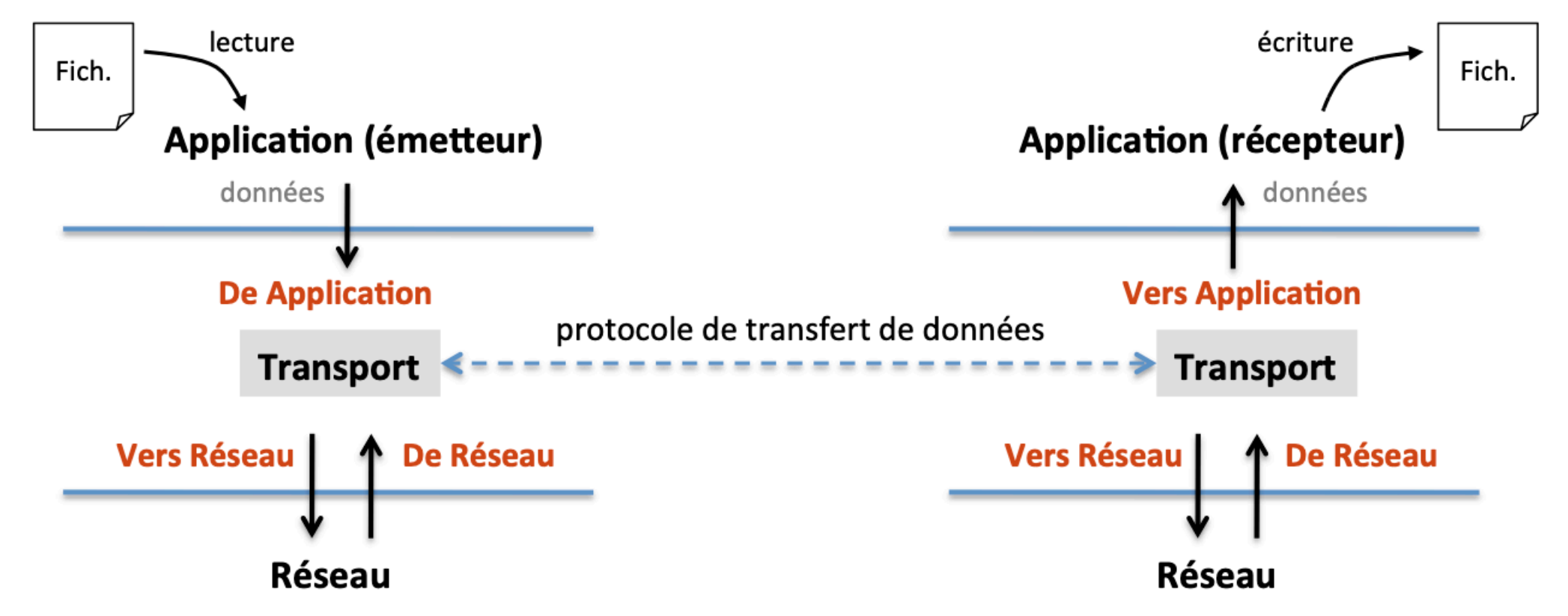


L'objectif de ces TP est de développer (en langage C) des protocoles de transfert de données mettant en oeuvre différents mécanismes de gestion de la fiabilité (contrôle d'erreurs, contrôle de flux, retransmissions).

Ces protocoles de transfert de données seront vus dans le contexte d'une couche transport (même scénario qu'en Cours/TD). Dans la suite du sujet, nous considérerons uniquement 3 couches : application, transport et réseau. La figure ci-dessous illustre l'articulation de ces trois couches :



- La **couche application** intègre un protocole de transfert de fichiers entre un émetteur et un récepteur. Cette couche a déjà été développée, elle vous est fournie sous la forme d'un module `application.h`.
- La **couche transport** constitue le cœur du travail. Vous devrez développer dans cette couche plusieurs protocoles de transfert de données (unidirectionnels pour les données) supportant différents types de services détaillés dans la section suivante.
- La **couche réseau** a également déjà été implémentée et vous est fournie sous la forme d'un module `services_reseau.h`. Elle peut fonctionner soit en mode local (émetteur et récepteur sur la même machine), soit en mode réparti (émetteur et récepteur sur des machines distinctes).

## Travail à réaliser

Le travail demandé consiste à implémenter les protocoles décrits ci-dessous. Pour chaque cas, vous développerez un programme principal pour l'émetteur et un programme principal pour le récepteur, qui devront respecter la convention de nommage suivante : `proto_tdd_vX_emetteur.c` et `proto_tdd_vX_recepteur.c` où `X` est le numéro de protocole (0, 1, 2, etc.). Vous placerez vos fonctions utilitaires dans `couche_transport.c`.

💡 **Aidez-vous des algorithmes vus en Cours/TD.**

Pour implémenter vos protocoles, vous vous appuyerez sur le type de données `paquet_t` spécifié dans `couche_transport.h` et rappelé ici :

```
typedef struct paquet_s {
    uint8_t type;           /* type de paquet (DATA, ACK, CON_REQ...) */
    uint8_t num_seq;        /* numéro de séquence */
    uint8_t lg_info;        /* longueur du champ info */
    uint8_t somme_ctrl;      /* somme de contrôle */
    unsigned char info[MAX_INFO]; /* données utiles du paquet */
} paquet_t;
```

Pour simplifier, le même format de paquet sera utilisé pour tous les protocoles. Certains paquets n'utiliseront pas tous les champs de la structure. Par exemple, un paquet d'acquittement ne contiendra pas de données utiles, le champ `lg_info` devra donc être initialisé à 0.

### Test du protocole v0 : transfert de données sans garantie

À titre d'exemple, nous vous fournissons ce protocole qui offre un service sans aucune garantie sur la délivrance des données (des erreurs ou des pertes peuvent se produire). Aucun mécanisme de contrôle de flux, ni de contrôle et reprise sur erreurs n'est implémenté.

Consultez le code source de `proto_tdd_v0_emetteur.c` et `proto_tdd_v0_recepteur.c` pour comprendre le fonctionnement de ces programmes.

Pour tester :

1. `make tdd0` pour générer les exécutables du protocole v0 dans le dossier `bin/`
2. Dans un terminal, lancez le récepteur `bin/recepteur` puis, dans un autre terminal, lancez l'émetteur `bin/emetteur`
3. Vérifiez si le transfert s'est bien effectué (résultat dans dossier `fichiers/`, le nom du fichier transmis est spécifié dans `config.txt`)
4. Modifiez le taux d'erreurs et de pertes dans `config.txt` et refaites les étapes 2 et 3.

💡 **Aide :**

- [Paramètres du fichier `config.txt`](#)
- [Utilisation du `Makefile`](#)

## Protocole v1 : « Stop-and-Wait » avec acquittement négatif, mode non connecté

Cette version de protocole rajoute deux mécanismes par rapport à la v0 :

- un contrôle de flux en mode « Stop-and-Wait » ;
- une détection d'erreurs basée sur une somme de contrôle. Cette somme sera calculée en appliquant l'opérateur « ou exclusif » (XOR) sur trois octets de l'en-tête (`type`, `num_seq`, `lg_info`) ainsi que sur tous les octets de données du paquet. Si aucune erreur n'est détectée par le récepteur alors un acquittement positif sera renvoyé (type `ACK`), sinon un acquittement négatif (type `NACK`).

On ne considérera dans cette version que des erreurs bits sur des paquets de données (type `DATA`), pas sur les paquets d'acquittement.

💡 **Aidez-vous de l'algorithme 1 vu en CTD.**

## Protocole v2 : « Stop-and-Wait » ARQ, mode non connecté

La **gestion des pertes** sera rajoutée dans ce protocole (en plus des erreurs bits), avec une reprise sur erreurs « Stop-and-Wait ». Contrairement à la version précédente, la retransmission se fera suite à l'expiration d'un temporisateur (pas d'acquittement négatif).

Concernant les erreurs bits, comme précédemment, on ne considérera que des erreurs sur des paquets de données (type `DATA`), pas sur les paquets d'acquittement.

💡 **Aidez-vous de l'algorithme 2 vu en CTD.**

## Protocole v3 : « Go-Back-N » ARQ, mode non connecté

Dans ce protocole, vous devez implémenter **une fenêtre d'anticipation avec une stratégie de retransmissions de type « Go-Back-N »**.

La capacité de numérotation sera de 16 et la fenêtre d'émission aura une taille de 7 par défaut. Cette taille pourra être modifiée à l'appel du programme `emetteur` en passant la taille de la fenêtre en paramètre (qui devra dans tous les cas être strictement inférieure à la capacité de numération). Exemple avec une fenêtre d'émission de taille de 4 :

```
$ bin/emetteur 4
```

Rappel des points importants sur le « Go-Back-N » :

Pour l'émetteur :

- un seul temporisateur est suffisant pour toute sa fenêtre ;
- à la réception d'un ACK sans erreur, si le numéro d'ACK est compris dans la fenêtre, l'émetteur décale sa fenêtre de manière cumulative ;
- lors d'un *timeout*, l'émetteur retransmet toute sa fenêtre (jusqu'à son pointeur courant).

Pour le récepteur :

- il n'accepte que les paquets en séquence (fenêtre de réception=1) ;
- chaque paquet reçu sans erreur génère un ACK numéroté. Si le paquet reçu est en séquence alors on acquitte ce paquet, sinon on acquitte le dernier paquet reçu correctement.

💡 **Aidez-vous de l'algorithme 3 vu en CTD.** De plus, nous vous fournissons déjà la fonction `dans_fenetre()` pour vérifier si un numéro de séquence est inclus dans la fenêtre d'émission (cf. `couche_transport.h`).

## Protocole v4 : « Selective Repeat » ARQ, mode non connecté

Dans ce protocole, vous devez implémenter **une fenêtre d'anticipation avec une stratégie de retransmissions de type « Selective Repeat »**.

La capacité de numérotation sera de 16 et la fenêtre d'émission aura une taille maximale de la moitié de cette capacité. La taille de la fenêtre d'émission pourra être définie à l'appel du programme `emetteur`. De manière similaire, la taille de la fenêtre de réception pourra être définie à l'appel du programme `recepteur` (il est recommandé de prendre la même taille de fenêtre côté émetteur et récepteur).

Rappels des points importants sur le « Selective Repeat » :

Pour l'émetteur :

- un temporisateur devra être utilisé pour chaque paquet envoyé ;
- à la réception d'un ACK sans erreur inclus dans la fenêtre, on ne décale la fenêtre que si cela est possible (cf. diapo 20 dans le support CTD sur « sliding window ») ;
- lors d'un *timeout*, l'émetteur ne retransmet que le paquet pour lequel le délai a expiré.

Pour le récepteur :

- il peut accepter des paquets hors séquence mais dans la fenêtre, dans ce cas ces paquets sont « bufferisés » ;
- chaque paquet reçu sans erreur génère un ACK. On acquitte le paquet qu'il soit en séquence ou hors séquence.


# Évaluation

Vous disposez de **5 séances** de TP. La validation de votre travail se fera au fur et à mesure de votre avancement avec votre enseignant de TP (vous serez donc amenés à lui faire une démonstration du fonctionnement de vos programmes). Par ailleurs, **une archive de votre code source devra être déposée sur Moodle, cf. consignes ci-dessous**.

À titre indicatif, le barème sera le suivant :

- Protocoles: 16 pts (v1 = v2 = v3 = v4 = 4 pts)
- Suivi: 2 pts (assiduité, avancement au cours des séances)
- Code: 2 pts (lisibilité, modularité, commentaires)

## Intégrité académique

 Le travail doit être réalisé individuellement. Vous pouvez discuter du sujet avec vos camarades mais il est strictement interdit de copier-coller du code. L'équipe pédagogique utilise des outils de mesures de similarité de codes sources. **Tout plagiat sera sévèrement sanctionné.**

# Consignes de dépôt du code source

Une archive de votre code source doit être déposée sur Moodle.

1. Renseignez votre nom et prénom (sans espace) dans le `Makefile` (variable `NOM_ETU`)
2. Éditez le fichier `README.txt` si vous souhaitez notifier l'enseignant d'un point particulier (protocoles fonctionnels, bug connu...)
3. Générez l'archive à l'aide de la commande `make deliver`
4. Déposez l'archive sur Moodle **au plus tard le 29/03/2024 à 23h59**