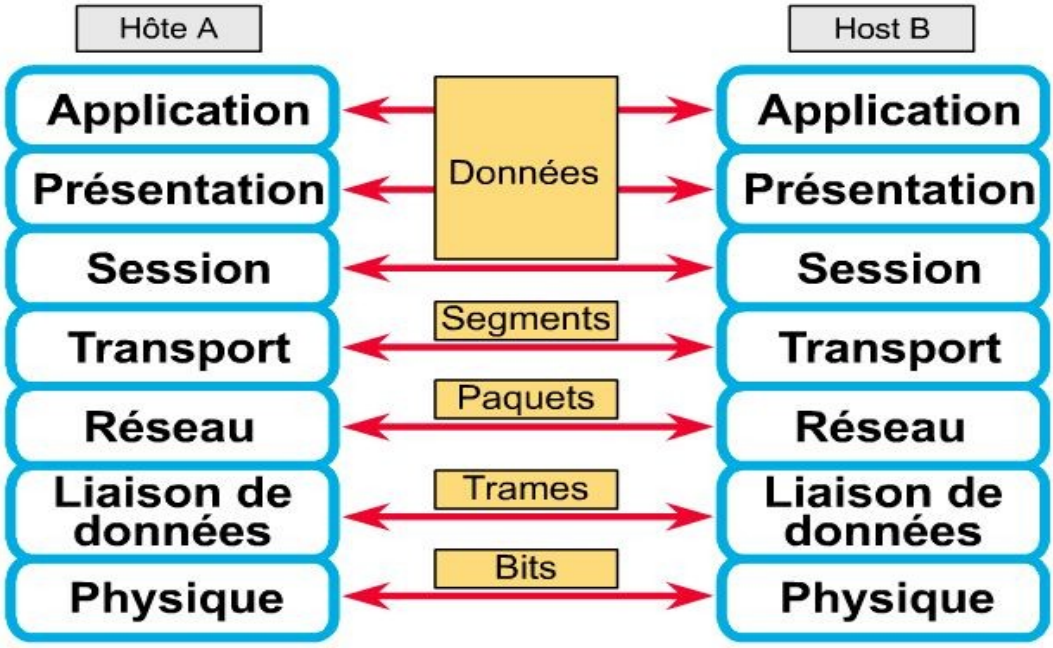
TP2. Réseaux (Structuration de trames (Couche 2))



Lors du TP 1, nous avons mis en pratique des solutions techniques pour communiquer au **niveau 1 de la représentation OSI**. En particulier, nous avons écrit des fonctions qui permettent, via des signaux, (1) à un émetteur d'envoyer une séquence de bits à un récepteur et (2) à un récepteur de reconstituer la séquence de bits envoyée par un émetteur. Nous avons vu les limites de cette technique face aux aléas du monde réel et proposé des solutions pour réduire les erreurs (i.e. faire en sorte que l'émetteur et le récepteur partagent des séquences de bits aussi proches que possibles). Nous avons étudié en particulier les échecs dus aux désynchronisations d'horloges, mais il en existe d'autres, comme des signaux corrompus en chemin ou des canaux bruités.

Sur ce TP, nous allons mettre en pratique des solutions techniques communiquer via la **couche 2 du modèle OSI**. La couche 2 vise à échanger des *trames de données* en s'appuyant sur la couche 1. On passe d'un signal brut (i.e. la séquence de ± sur le premier TP) à des messages structurés, dont on peut reconnaître le début et la fin, l'émetteur et le destinataire. La couche 2 couvre aussi le transfert correct de ces trames de données *au niveau local* (sans intermédiaires identifiés) : vérifier que les messages ont bien été reçus, corriger les erreurs, partager les canaux de communication etc. Sur ce TP, nous allons mettre en pratique la structuration des trames pour couvrir les besoins de la couche 2. Sur le prochain TP, nous étudierons des protocoles pour contrôler l'échange de messages de la couche 2.

Nous aurons à notre disposition le contenu du niveau 1 du modèle OSI, développé pendant le TP1 : on peut échanger des bits de donnée via une transformation en signal avec relativement peu d'erreurs.

Simuler un réseau simple

Afin de bien effectuer le lien avec le premier TP, on va commencer par mettre en place un environnement de simulation qui distingue les différents éléments d'un réseau. Cet environnement de simulation est disponible sur :

https://tinyurl.com/reseau-tp2

- *Etudiez l'algorithme « Simulation »*. Cet algorithme simule le transfert complet d'un message en passant par la couche 2 puis la couche 1 du modèle OSI. Cet algorithme contient trois parties : *émission*, qui simule le travail de l'émetteur ; *transmission*, qui simule les effets causés par la transmission (ex : bruit sur le canal) ; *réception*, qui simule le travail du récepteur
- « Emission » prend en entrée en entrée un **message** (message représenté par « o » et « i » où « o » est représenté par « 0 » et « i » est représenté par un « 1 ») ; **il effectue les opérations relatives à la couche 2 : transformer le message en une séquence de bits** de sorte à ce que le récepteur puisse, à partir de cette même séquence de bits récupérer le message initial. Puis, **il effectue les opérations relatives à la couche 1 : transformer la séquence de bits en un signal,** de sorte à ce que le récepteur puisse, à partir de cette même séquence de signaux récupérer le message initial. Cette seconde fonction correspond au codage NRZ, rédigé pendant le TP1
- « Transmission » simule les effets causés par la transmission : le fait qu'il y ait déjà des signaux qui résident sur le canal, les erreurs possibles
- « Réception » simule le travail du récepteur. Il effectue les opérations inverses de l'émetteur : **transformer signaux en une séquence de bits (couche 1) ; transformer une séquence de bits en un message (couche 2)**

Pour le moment, ce code est incomplet : le message reçu ne correspond pas au message envoyé. Le travail de ce TP consiste à étendre les fonctions couche2 et couche2R afin de faire correspondre ces deux messages.

Mettre des bords

Une première étape pour transformer une séquence de bits en une trame de donnée consiste à définir les bordures du message. Ainsi, il devient possible (1) de distinguer un message utile d'un bruit aléatoire et (2) de distinguer plusieurs messages émis.

- Étendre couche2
  - Écrire un algorithme qui ajoute la bordure « 01111110 » au début du message à
  - Écrire un algorithme qui ajoute au début du message la taille du message à envoyer (en binaire sur 8 bits)
- Étendre couche2R

    - Écrire un code qui, à partir d'un message enrichi d'une bordure de départ et d'une taille de message, permet d'obtenir le message initial.
- Appliquez cet algorithme pour échanger le message : «oooiii»
- Afficher la quantité de signaux échangés par rapport à la taille du message initial

La puissance du modèle OSI

Permuter la couche 1 du modèle OSI : utiliser le codage NRZI ou Manchester au lieu du codage NRZ du côté de l'émetteur que du récepteur. Observer que le message est envoyé sans accroc (si les fonctions de codage/décodage du niveau 1 sont correctes, bien entendu).

Quel sont les bénéfices ? Imaginez qu'au lieu d'envoyer des « +- », vous envoyez des impulsions (ex : signal WiFi), quelles parties auriez-vous à ré-écrire ? Qu'est-ce que cela implique pour les couches supérieures ?

Are you talking to me?

Il arrive fréquemment que plusieurs systèmes écoutent le même canal de communication, ex : ondes écoutées par plusieurs systèmes, répétiteurs qui retransmettent tous les signaux à toutes les machines connectées.

Le niveau 2 du modèle OSI est en charge de déterminer qui envoie un message et pour qui, en local (connexion directe entre l'émetteur et le récepteur, les connexions qui demandent de passer par des intermédiaires identifiés sont gérées par la couche 3).

- Écrire un code « AddIdentification » qui ajoute l'adresse de l'émetteur et du récepteur en binaire, sur quatre bits par adresse. A partir de maintenant, l'émetteur est la machine 1 et le récepteur est la machine 3
- Écrire un algorithme côté récepteur « WhoIsTheSender » qui indique le nom de l'émetteur. Afficher cette information quand un message est reçu
- Écrire un algorithme « IsAddressedToMe » qui vérifie si un message reçu est bien adressé au récepteur. Si ce n'est pas le cas, retourner « message ignoré »
- Envoyer un message vers la machine 2. Constater que le message est bien envoyé par l'émetteur et ignoré par le récepteur. Envoyer le même message sur la machine 3, constater qu'il est bien intégré par le récepteur.

Détecter et supprimer les erreurs

Les séquences de bits provenant de la couche 1 sont parfois erronées. Afin de transmettre des données fiables, la couche 2 doit détecter, voire réparer ces erreurs.

- Écrire un code « SignalSwapError » qui prend une séquence de signaux et un entier *n* et qui altère le n-ième signal de la séquence

- Dans Simulation, appliquer « SignalSwapError » entre le signal émis et le signal reçu, sur un signal de la séquence de signaux qui code le message(ex : le 30ème). Ne pas essayer cela sur un message de l'entête (les erreurs sur l'entête sont détectées autrement, notamment parce que l'émetteur ne reçoit jamais l'ACK)
- Ré-exécuter le code pour transférer « 1111011100001011010 », constater qu'une erreur s'est glissée
- Appliquer l'algorithme « parity bit » pour détecter l'erreur. Si une erreur est détectée, retourner « erreur de transfert » au lieu du code à transférer. Notez qu'il faut appliquer l'algorithme à l'envoi et à la réception et rajouter un bit à la trame
- Rajouter une autre erreur dans le signal avec « SignalSwapError ». Constater que « parity bit » échoue à trouver l'erreur.
- Appliquer l'algorithme « double parity » pour détecter les erreurs. Comparer avec « parity bit »
- Appliquer l'algorithme « CRC » pour détecter les erreurs. Comparer avec le double parity
- Appliquer le code de correction de Hamming pour réparer les erreurs
- Comparer les différentes solutions en termes de coût d'envoi et d'efficacité à détecter les erreurs