Université de Caen Normandie
U.F.R des sciences
L2 – informatique

Réseaux 1 - TP3
Couche liaison de données

Introduction

Nous allons implémenter, dans ce TP, des solutions pour communiquer via la couche 1 et 2 du modèle **OSI** en développant un simulateur d'un réseau de communication. 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 couche2.

Nous aurons à notre disposition le contenu du niveau 1 du modèle OSI, qui reprend les travaux du TP1. On peut échanger des bits de donnée via une transformation en signal avec relativement peu d'erreurs.

Simulateur d'un réseau de communication

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 Ecampus.

- Étudiez 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 aléatoire sur le canal**); **réception**, qui simule le travail du récepteur.
- «Émission» 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 ceque 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)

Détection et Correction d’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 même réparer ces erreurs. La fonction «SignalSwapError» prend une séquence de signaux (ou de bits) et un entier n afin d’altérer le n-ième signal (ou bit) de la séquence.

Vous allez remarquer que la fonction «SignalSwapError» est appliquée dans la fonction transmission avant le rajout du bruit, plus précisément sur la séquence de signaux qui code le message (e.g. le 28ème). Dans cette simulation, nous allons pas essayer sur un bit de l'entête (les erreurs sur l'entête sont détectées autrement, notamment parce que l'émetteur ne reçoit jamais l'**ACK**)

1. Ré-exécutez la simulation pour transférer «1111011100001011010», constater qu'une erreur s'est glissée ! La fonction existante «parity_bit» est utilisée pour détecter l'erreur. Si l’erreur est détectée, elle retourne «erreur de transfert» au lieu du code à transférer. Notez que la fonction est utilisée à l'envoi en rajoutant un bit à la fin de la trame et aussi à la réception.
2. Rajoutez une autre erreur dans le signal avec «SignalSwapError» (e.g. position 26ème). Constater que «parity_bit» échoue à trouver l'erreur.
3. Complétez l'algorithme «double_parity» pour détecter les erreurs. Comparez le résultat avec «parity_bit» sur le même code transféré.
4. Rédigez l'algorithme «CRC» pour détecter les erreurs. Comparez avec le « double_parity »
5. Enlevez le second SignalSwapError de la question 2.
6. Rédigez l’algorithme de correction de Hamming pour réparer les erreurs
7. Comparer les différentes solutions en termes de coût d'envoi et d'efficacité à détecter les erreurs