IFT585 - Télématique

Simulateur de communication réseau

Date 19 mai 2022

1 Programme du Simulateur

Le programme permet de simuler un ensemble d'ordinateurs connectés sur le même sous-réseau. Il est possible de modifier plusieurs paramètres afin de voir leurs effets sur le transfert des données entre les ordinateurs simulés.

Le simulateur peut recevoir soit les trois premiers paramètres suivants lors de son démarrage, soit le dernier pour exécuter le test de correction et de détection d'erreur :

Paramètre	Valeur par défaut	Description
-c	2	Nombre d'ordinateurs dans la simulation
-f	1	Numéro du premier ordinateur
-g	Chaîne vide	Nom du fichier de configuration global du réseau
-t	Aucun*	Exécuter en mode test pour la correction d'erreur

*Le paramètre n'a pas de valeur par défaut. Vous devez spécifier 1 pour tester l'implémentation du code CRC ou 2 pour l'implémentation du code de Hamming.

Chaque ordinateur est connecté à un hub central qui redistribue l'ensemble des données reçues d'un ordinateur à tous les autres ordinateurs connectés.

Tous les fichiers de configuration doivent être dans le répertoire d'exécution du simulateur. Le simulateur s'occupera de les lire et de créer le sous-réseau approprié.

Note : Tous les paramètres dans les fichiers de configurations qui suivent sont écrits au format suivant (un par ligne) : NomDuParamètre=ValeurDuParamètre

2 Hub central

Le **hub central** est paramétrable afin de simuler des erreurs de transmission. Le tableau suivant donne les paramètres possibles à mettre dans le fichier de configuration global et leur valeur par défaut :

Paramètre	Valeur par défaut	Description
TransmissionHubBufferSize	500000	Nombre d'octets du tampon interne du hub de transmission
TransmissionHubNoise	1	Type d'interférence à appliquer sur les données (0 : aucune interférence, 1 : interférence aléatoire)

TransmissionHubNoiseFrequency	10	Pourcentage des données qui transitent par le hub sur lesquelles de l'interférence doit être appliquée
TransmissionHubNoiseByteErrorFrequency	1	Pourcentage des octets qui seront modifiés advenant une interférence

3 Ordinateurs

Chaque ordinateur est composé d'une carte réseau sur laquelle un petit pilote s'exécute. Le pilote est composé de trois couches, soit la sous couche liaison basse (*LinkLayerLow*), la sous couche liaison haute (*LinkLayer*) et la couche réseau (*NetworkLayer*).

Chaque ordinateur est paramétrable. Les paramètres de chacun des ordinateurs doivent se retrouver dans un fichier nommé « **ComputerX.txt** » où X est le numéro de l'ordinateur. Les numéros sont assignés séquentiellement à partir de la valeur de l'argument –f lors du démarrage du simulateur (voir la première table). Chaque ordinateur peut aussi spécifier l'ensemble des envois de fichiers à effectuer dans un fichier de configuration nommé « ComputerXFiles.txt » où X est encore une fois le numéro de l'ordinateur. Le fichier de configuration de chaque ordinateur doit contenir au minimum l'information sur l'adresse MAC qui y sera associée. La table suivante indique le nom des paramètres pour spécifier cette adresse. Les valeurs qui y sont associées doivent être indiquées en hexadécimal :

Paramètres	Description
`MacAddressByte1`	Premier octet de l'adresse (le plus à gauche)
`MacAddressByte2`	Deuxième octet de l'adresse
`MacAddressByte3`	Troisième octet de l'adresse
`MacAddressByte4`	Quatrième octet de l'adresse
`MacAddressByte5`	Cinquième octet de l'adresse
`MacAddressByte6`	Sixième octet de l'adresse (le plus à droite)

Les noms des autres paramètres de configurations possibles pour un ordinateur sont donnés dans les tables suivantes associées à chacune des couches du pilote. Le format du fichier contenant les fichiers à transmettre est le suivant :

```
NomDuFichier=Adresse MAC de destination (au format hexadécimal AB:CD:EF:98:76:54)
```

4 Sous couche liaison basse

La sous couche liaison basse s'occupe d'encoder et de décoder des chaines d'octets reçues à l'aide d'algorithme de détection/correction d'erreur. La sous couche s'occupe ensuite d'envoyer les données à la couche liaison haute ou sur le câble qui connecte l'ordinateur au hub central. L'annexe 1 indique le fonctionnement de certaines parties du code et ce qui doit être modifié par les étudiants.

La sous couche liaison basse peut aussi être paramétrée. Le tableau suivant donne les paramètres possibles pour cette couche :

Paramètre	Valeur par défaut	Description
LinkLayerLowReceivingBufferSize	500000	Nombre d'octets dans le buffer de réception
LinkLayerLowSendingBufferSize	500000	Nombre d'octets dans le buffer d'envoi

LinkLayerLowDataEncoderDecoder	0	Type d'encodeur de données à utiliser pour la correction ou détection des erreurs (0 : Aucun encodage/décodage ; 1 : Code de Hamming, 2 : Code CRC)

5 Sous couche liaison haute

La sous couche liaison s'occupe de l'envoi sous forme de trame des paquets produits par la couche réseau vers la sous couche liaison basse. C'est dans cette couche que le protocole d'envoi doit être implémenté. L'annexe 2 donne une description des différentes fonctions utilisables afin de s'assurer que les deux fils d'exécution de la sous couche (réception et envoi) puissent se parler afin de gérer adéquatement leur partie de l'algorithme.

La sous couche liaison peut elle aussi être paramétrée. Le tableau suivant donne les paramètres possibles :

Paramètre	Valeur par défaut	Description
LinkLayerReceivingBufferSize	500000	Nombre d'octets du buffer de réception des données à partir de la couche liaison basse
LinkLayerSendingBufferSize	500000	Nombre d'octets du buffer d'envoi des données vers la couche liaison basse
LinkLayerMaximumBufferedFrame	4	Nombre maximum de trames envoyé gardé en cache (taille de la fenêtre d'envoi)
LinkLayerTimeout	1000	Nombre de millisecondes avant de réenvoyer une trame déjà envoyée si aucun accusé de réception n'a été reçu

6 Couche réseau

La couche réseau s'occupe d'envoyer un fichier en le découpant en petit morceau (paquet) qui sont ensuite envoyés à la sous couche liaison haute. La couche réseau s'occupe aussi de recevoir des paquets en provenance de la sous couche liaison haute et de reconstruire le fichier reçu. Les informations et les données sur le fichier reçu et envoyé par la couche réseau sont organisées dans le format suivant :

Taille de l'information en octet	Description	Type de valeur
4 octets	Nombre de caractère dans le nom du fichier	UInt32
X octets	Chacun des caractères du nom du fichier	Suite de Char
8 octets	Nombre d'octets dans le fichier	UInt64
Y octets	Les octets de données du fichier	Suite de UInt8

Lorsque la couche réseau reçoit un fichier, le nom du fichier produit sera « From_X_To_Y_Z » où X est l'adresse MAC de la source d'envoi au format hexadécimal « AB-CD-EF-98-76-54 », Y est l'adresse MAC de la destination au même format que la source et Z est le nom du fichier d'origine.

La couche réseau, tout comme les autres couches, est aussi paramétrable. Le tableau suivant indique les paramètres possibles :

Paramètre	Valeur par défaut	Description
NetworkLayerReceivingBufferSize	500000	Nombre d'octets du buffer de réception des données à partir de la couche liaison
NetworkLayerSendingBufferSize	500000	Nombre d'octets du buffer d'envoi des données vers la couche liaison
NetworkLayerDataSize	50	Nombre d'octets de donnée dans un paquet

Annexes

Annexe 1 Encodeur et décodeur de la couche liaison basse

Il y a trois types d'encodeur et décodeur pour les données de la couche liaison basse. Le premier ne fait rien (*PasstroughDataEncoderDecoder*). Les méthodes *encode* et *decode* s'occupent d'encoder et décoder le *buffer* de données reçu en paramètre.

La méthode *decode* retourne une paire comprenant une valeur booléenne indiquant si les données décodées sont valides et peuvent être utilisées.

> Pour l'implémentation du code de Hamming, la valeur devrait toujours être vraie puisque les données peuvent être corrigées.
>
> Pour l'implémentation du code CRC, la valeur pourrait être fausse si le CRC calculé n'est pas bon.

Les méthodes *encode* et *decode* sont appelées automatiquement par la couche liaison basse. **Vous n'avez qu'à écrire le code des méthodes.**

Annexe 2 : Méthodes de la classe *LinkLayer*

La présente annexe donne de l'information sur les différentes méthodes de la classe *LinkLayer* afin d'aider les étudiants à bien implémenter le protocole de communication demandé en utilisant adéquatement les outils fournis.

La classe *LinkLayer* exécute deux fils d'exécution en parallèle.

> Le premier s'occupe de la lecture des données de la couche réseau et envoi des données sur le buffer d'envoi de la couche liaison en suivant les étapes du protocole. La méthode *senderCallback* s'exécute sur ce fil d'exécution. **Vous devez remplacer le code de cette méthode par le code qui gère l'envoi du protocole à implanter**.

Le deuxième fil d'exécution s'occupe de lire les données reçues sur le buffer de réception qui proviennent de la couche liaison basse, s'occupe du traitement associé au protocole et envoi éventuellement les données reçues sur le buffer de réception de la couche réseau. La méthode *receiverCallback* s'exécute sur ce deuxième fil d'exécution. **Vous devez remplacer le code de cette méthode par le code qui gère la réception du protocole à implanter**. Dans le protocole à implémenter, lors de la réception de certaines données, il faut répondre en envoyer d'autres données. Comme les deux fils d'exécution sont indépendants, il n'est pas possible (voir dangereux) de faire l'envoi directement dans le fil d'exécution de la réception. Un ensemble de fonctionnalités a été ajouté afin de vous permettre d'échanger des « messages » entre les deux fils d'exécution. Il vous appartiendra ensuite de gérer les messages dans chacune des deux méthodes principales (*senderCallback* et *receiverCallback*). Les méthodes *getNextSendingEvent*() et *getNextReceivingEvent*() vous permettent chacune de récupérer respectivement le prochain événement à gérer en lien avec l'envoi ou la réception. Si aucun événement n'est à gérer, ces méthodes retourneront un événement de type *Invalid*. La liste des méthodes suivante vous permet d'ajouter un nouvel événement à gérer dans les files d'événements respectifs de l'envoi et de la réception :

sendAck(const MACAddress& to, NumberSequence ackNumber)

Demande à ce qu'une trame de type ACK soit envoyée à l'adresse indiquée avec le numéro spécifié. Cette méthode devrait être appelée par la méthode qui reçoit des données. Vous devriez gérer l'événement produit (SEND_ACK_REQUEST) dans la méthode d'envoi des données.

sendNak(const MACAddress& to, NumberSequence nakNumber)

Demande à ce qu'une trame de type NAK soit envoyée à l'adresse indiquée avec le numéro spécifié. Cette méthode devrait être appelée par la méthode qui reçoit des données. Vous devriez gérer l'événement produit (SEND_NAK_REQUEST) dans la méthode d'envoi des données.

notifyNAK(const Frame& frame)

Envoi un événement qui indique qu'on a reçu une trame NAK. Cette méthode devrait être appelée par la réception des données pour indiquer à celle qui envoi qu'on a reçu un NAK. L'événement produit est NAK_RECEIVED.

notifyACK(const Frame& frame, NumberSequence piggybackAck)

Envoi un événement qui indique qu'on a reçu une trame et qu'il y a probablement un ACK en plus dans cette trame. Cette méthode permet aussi d'envoyer le prochain numéro de trame attendu en *piggybacking*. L'événement produit est ACK_RECEIVED.

notifyStopAckTimers(const MACAddress& to)

Envoi une demande pour arrêter le minuteur des ACK pour une adresse de réception particulière. Cette méthode enverra un événement de type STOP_ACK_TIMER_REQUEST à la file d'événements de la méthode de réception. Elle devrait être appelée dans la méthode d'envoi.

startAckTimer(size_t existingTimerID, NumberSequence ackNumber)

Cette méthode permet de démarrer un minuteur pour l'envoi d'un ACK. Il faut lui envoyer en paramètre le numéro du minuteur courant pour les ACK afin de le redémarrer s'il existe déjà. Ce minuteur enverra éventuellement un événement ACK_TIMEOUT à la file d'événements de la méthode de réception. Vous devrez donc gérer cet événement pour notifier la méthode d'envoi qu'il doit envoyer un ACK avec le numéro de trame approprié.

size_t startTimeoutTimer(NumberSequence number)

Démarre un nouveau minuteur pour réenvoyer une trame déjà envoyé si aucune réponse n'est reçue. Ce minuteur ajoutera éventuellement l'événement SEND_TIMEOUT à la file d'événements de la méthode d'envoi afin que la trame indiquée par le minuteur soit envoyée à nouveau.

bool sendFrame(const Frame& frame)

Cette méthode envoie une trame vers la couche liaison basse. Elle se bloque si la couche liaison basse est pleine. Elle retourne toujours vraie lorsque la trame a été transférée à la couche inférieure. Si la méthode retourne faux, c'est qu'il y a eu une demande pour arrêter le simulateur et vous devez donc quitter aussi vous-même la fonction qui l'a appelé (seule la méthode *senderCallback* devrait envoyer des trames à la couche liaison basse).

Événements

Les événements reçus possèdent tous un champ *Number* qui est associé au numéro reçu en paramètre lors de sa création. Lorsque pertinents, les champs *Address* et *TimerID* sont aussi remplis. Le champ *Next* est utilisé seulement pour l'événement ACK_RECEIVED et devrait indiquer le prochain numéro de ACK à envoyer en *piggybacking* sur les trames.