TP n°3 : Client/Serveur UDP	Durée : 8 heures
Principale compétence terminale concernée	Savoir et savoir faire associés
C4.1 Câbler et/ou intégrer un matériel C4.1 Adapter et/ou configurer un matériel C4.4 Développer un module logiciel C5.2 Mettre en œuvre une solution matérielle/logicielle en situation	SF35. Réaliser la mise en situation et interconnecter du matériel. S7.7 Programmation réseau S7.7.1 Concept Client/Serveur S7.7.2 Sockets POSIX

Niveau	2ere année BTS SN IR
Configuration du poste	Possède un système Linux avec le compilateur gcc
Données fournies	Cours Réseau API Socket
Prérequis	Maîtrise du langage C

1] Client/Serveur UDP squelette : programme srv_udp_sq et clt_udp_sq

Tester les programmes fournis client/serveur squelettes.
a) Test en local et b) Test avec une machine distante

2] Client/Serveur UDP simplifié : programme srv_udp_simple et clt_udp_simple

Coder les programmes client/serveur simplifiés.
Le serveur reçoit une trame, l'affiche et la renvoie ceci tant que la requête est différente de "fin".
Le client envoie des trames saisies au clavier tant que différent de "fin".
Simplifié signifie tout dans le main() avec le minimum de fonctions.
a) Test en local et b) Test avec une machine distante

3] Création d'outils pour observer les caractéristiques UDP

Copier et modifier les programmes squelettes pour répondre aux exigences suivantes.

3.1] Le programme client s'exécutera ainsi : $ **clt_udp_cpt** nom_machine n°port nombre_paquet taille_paquet
Cette application permet donc d'envoyer des paquets dont le nombre et la taille sont paramétrables.

Exemple : $ clt_udp_cpt p1.tsii.lml 23456 1024 128

L'application commence par vérifier si elle peut obtenir le n°IP de la machine cible, en cas de succès elle alloue dans cet exemple, 128 octets et initialise cette zone avec des '0'.
Le paquet étant construit, elle l'envoie, dans cet exemple, 1024 fois au serveur.
Le premier paquet envoyé contiendra donc des '0', au deuxième envoi ce sera des '1', etc jusqu'à '9' puis on reviendra sur le chiffre '0'.
En fin de programme on affichera le nombre d'octets transmis.

3.2] Le programme serveur s'exécutera de la façon suivante : $ **srv_udp_cpt** n°port tps_charge delai
Cette application permet donc de recevoir des paquets sur un n° de port fixé avec possibilité d'insérer des temps d'attente (temps de charge) entre chaque lecture ainsi que de gérer un timeout (sortie automatique après une durée sans réception de données)

Exemple : $ srv_udp_cpt 23456 1 5

Cette application affiche les données et la taille de chaque paquet reçu.
Entre chaque réception on endort dans cet exemple le serveur 1 seconde pour simuler une charge système.
La fin de réception est réalisée au moyen d'un "timeout" de 5 secondes dans cet exemple, activé dès la 1iere réception.
Avant de se terminer, afficher le nombre total de paquets reçus et la taille totale d'octets reçus.
Le serveur affichera en plus des informations demandées, le n°IP et le n° de port du client.

3.3] Comment fonctionne la transmission UDP (paquet ou flux ?)

Faire en sorte que le client envoie plusieurs trames avant que le serveur puisse commencer à lire. Exemple

```
$ srv_udp_cpt 23456  3  5
$ clt_udp_cpt localhost  23456  10  20
```

Q1 : Le serveur peut-il lire en 1 coup plusieurs paquets ?

Q2 : Conclusion UDP est un mode de transmission par paquet ou flux ?

3.4] Test de perte de paquet

Commencer par lancer le serveur sans temps de charge et envoyer une petite quantité de données (Ex:10 x 128o). Comparer le nombre d'octets et de paquets transmis et reçus.

Augmenter progressivement le nombre de paquet et laisser la taille à 128 octets, on peut aussi jouer sur le temps de charge du serveur. Comparer à nouveau les valeurs.

Q3 : Constatation, peut-on perdre des données en mode UDP ?

3.5] Fragmentation IP

Utiliser **clt_udp_cpt** pour envoyer 1 paquet de 2048 octets au serveur **srv_udp_cpt**

A l'aide de l'outil "Wireshark", capturer toutes les trames échangées entre le client et le serveur qui sont en rapport avec votre transfert puis sauvegarder votre capture.

a) Test en local => fichier de capture : transfert_udp_local.pcap

b) Test avec machine distante => fichier de capture : transfert_udp_distant.pcap

Placer le filtre d'affichage suivant dans "Wireshark" : ip.flags.df == 0 sur vos 2 captures

Q4 : Comparer les résultats, interpréter et expliquer.

Q5 : Remplir le tableau (voir annexe) qui illustre le transfert des 2048 octets avec la machine distante.

Q6 : Préciser quelles autres trames (autre que UDP) ont été nécessaires au transfert ?

Remarques :

- **Ces applications devront gérer tous les cas d'erreurs possibles et donc préciser l'origine d'un problème !**

Annexes (Document réponse 3.5 Q5)

Doc réponse Q4	Valeurs	Signification	Valeurs	Signification
Entête de type :		**Ethernet : trame1**		**Ethernet : trame2**
@ destination				
@ source				
Protocole				
Entête de type :		**IP : trame1**		**IP : trame2**
Version				
Lg de l'entête				
Type de service				
Lg du paquet				
N° de séquence				
Drapeaux				
Place segment				
Durée de vie				
Protocole				
Checksum				
@ source		n° IP :		n° IP :
@ destination		n° IP :		n° IP :
Entête de type :		**UDP :** Transfert de 2048 octets		Transfert de 2048 octets suite
Port source				
Port destination				
Lg du message				
Checksum				
données				

Pour la correction

Doc réponse Q4	Valeurs	Signification	Valeurs	Signification
Entête de type :		**Ethernet : trame1**		**Ethernet : trame2**
@ destination				
@ source				
Protocole				
Entête de type :		**IP : trame1**		**IP : trame2**
Version				
Lg de l'entête				
Type de service				
Lg du paquet				
N° de séquence				
Drapeaux				
Place segment				
Durée de vie				
Protocole				
Checksum				
@ source		n° IP :		n° IP :
@ destination		n° IP :		n° IP :
Entête de type :		**UDP :** Transfert de 2048 octets		Transfert de 2048 octets suite
Port source				
Port destination				
Lg du message				
Checksum				
données				