

```
#contentWrapper #fs, #sidebarContent #fs, #contentWrapper div [id * = 'myExtraContent'], #sidebarContent  
div [id * = 'myExtraContent'] {display: block;}
```

Kermith's workshop ([https://translate.googleusercontent.com/translate\\_c?depth=1&hl=en&prev=search&pto=aue&rurl=translate.google.com](https://translate.googleusercontent.com/translate_c?depth=1&hl=en&prev=search&pto=aue&rurl=translate.google.com))

The other way to see supervision ...

## Network sockets with LiveCode



Sockets allow communication between various processes using the TCP / IP layer. Communication can take place in the machine itself with the local loop of the network interface or through the physical network between several machines. Historically, sockets have been implemented in Berkeley UNIX distributions, they are sometimes called BSD (Berkeley Software Distribution)

sockets.

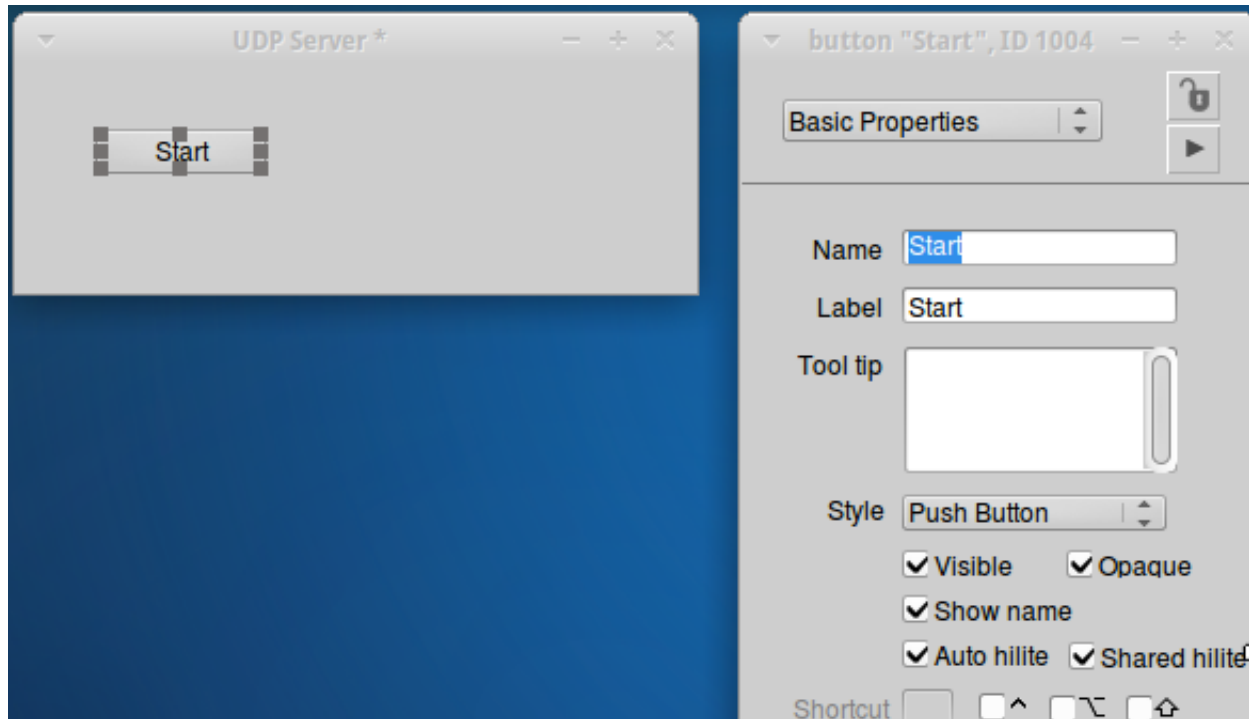
There are two types of communication:

The connected mode using the TCP protocol. This connection mode is comparable to telephone communication. A lasting connection is established between the transmitter (client) and the receiver (server). The destination address is only required when initializing the connection.

Unconnected mode using UDP protocol. This connection mode is comparable to sending a letter and requires no acknowledgment of receipt. The destination address is required at each connection.

## Creation of our first UDP server

To build our UDP server, we will use a virtual machine on Linux. This will have the IP address 172.16.209.176. We will need a battery containing a button named "Start" and that's it. Name your UDP Server stack immediately.



UDP Server interface

Right click the button to get the Object Inspector. Enter Start in Name and Label. Then right click the button again to open a window of the code editor. First, we will enter the code associated with the Start button. **on mouseUp – we check the button label – start: server start – stop: server stop if the label of me = "Start" then – we accept UDP connections on port 9997 – the dataReceived procedure is initialized accept datagram connections on port 9997 with message**

"dataReceived"

**– we modify the label of the button to stop set the label of me to "Stop" else – the UDP connection is stopped close socket 9997 – we modify the label of the button to start set the label of me to " Start " end if end mouseUp** Explanation: We accept incoming UDP connections on port 9997. For each successful connection, we will execute the procedure (Message Handler) dataReceived. You have to create the rest of the code. **on dataReceived pSocket, pMsg**

```
-- affichage boite de dialogue message UDP reçu  
answer pSocket & cr & pMsg  
end dataReceived
```

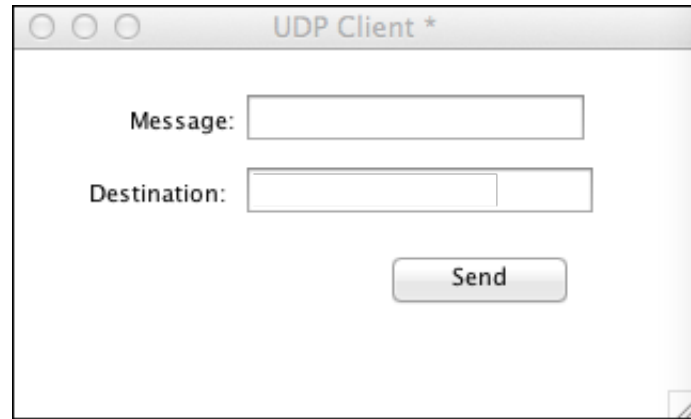
Explication : Le serveur reçoit la trame UDP et la traite par cette procédure. On récupère l'adresse de la machine distante et le message.

```
on socketError  
-- affichage boite de dialogue erreur flux UDP  
answer "Failed to open server socket"  
end socketError
```

Explication : la procédure socketError est activée lorsqu'un problème survient lors de l'établissement de la connexion UDP.

## Création du programme client UDP

Le programme client va être réalisé dans un environnement Mac. Nous avons besoin d'une pile contenant deux champs texte "txtMessage" et "ipDestination" et d'un bouton nommé "Send". Nommez votre projet UDP client.



Pile Socket Client

Cliquez sur le bouton Send servira à initialiser la connexion TCP. Voici le code.

```

on mouseUp
  -- vérification d'un message
  if fld "txtMessage" is not empty then
    local ipDest
    -- initialisation de la variable ipDest
    -- ipDest adresse de destination du serveur UDP accompagné du numéro de port
    put fld "ipDestination" into ipDest
    put ":9997" after ipDest
    -- ouvre un flux UDP
    open datagram socket to ipDest
    if the result is not empty then
      -- pb réseau on envoie le résultat
      answer "Socket: " & the result
    end if
    -- on envoie le message sans accusé de reception
    write fld "txtMessage" to socket ipDest
    -- on clot la connexion UDP
    close socket ipDest
    -- on efface le message
    put empty into fld "txtMessage"
  else
    beep
  end if
end mouseUp

```

Explication : on vérifie l'existence d'un message dans le champ texte, puis on crée la

connexion UDP et on envoie directement le message sur le port 9997. on clos la communication puisqu'on n'attend pas d'accusé de réception du serveur. A la fin, on efface le message.

**on** socketError

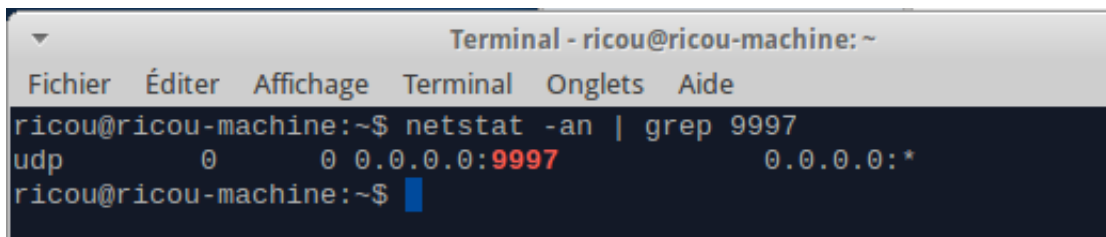
**answer** "Failed to open socket"

**end** socketError

Comme précédemment, cette procédure intercepte les erreurs du flux TCP

## Fonctionnement

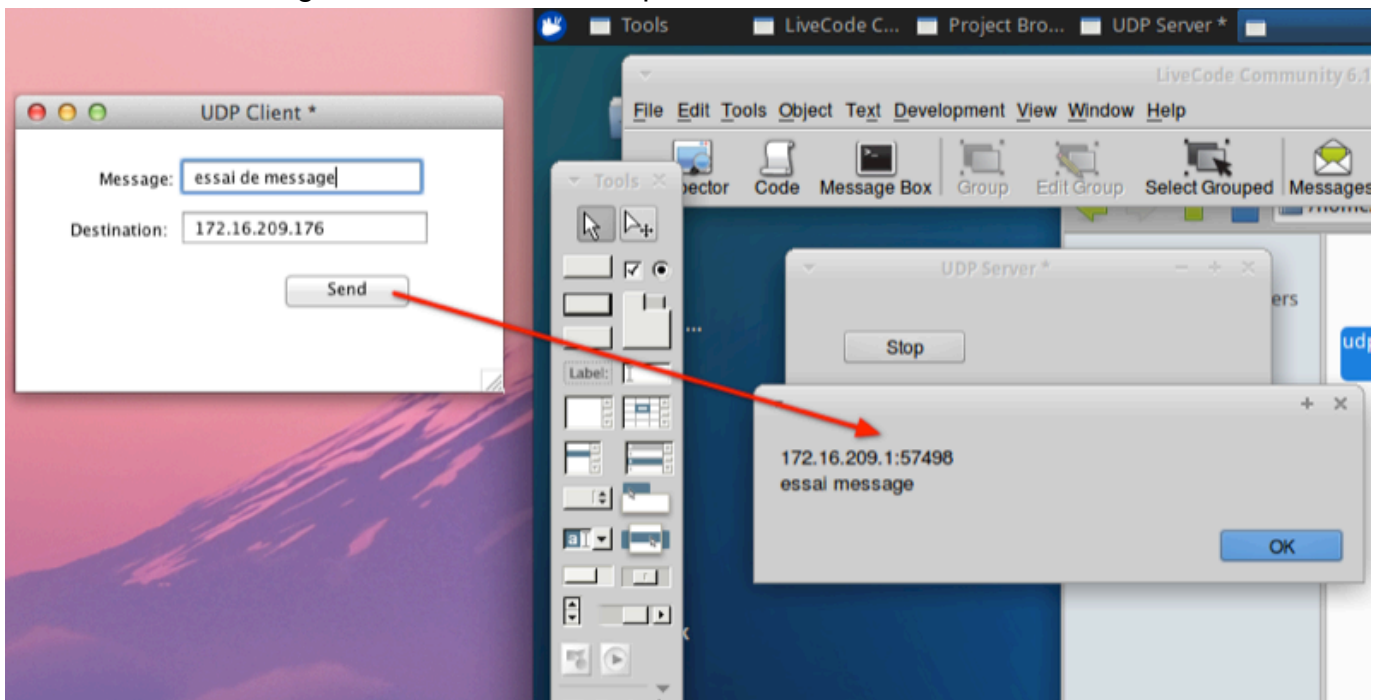
Il faut démarrer notre serveur UDP sur Linux en cliquant sur le bouton Start. Afin de vérifier son bon fonctionnement, utilisez la commande **netstat -an | grep 9997** en ligne de commande.



```
Terminal - ricou@ricou-machine: ~
Fichier  Éditer  Affichage  Terminal  Onglets  Aide
ricou@ricou-machine:~$ netstat -an | grep 9997
udp        0      0 0.0.0.0:9997          0.0.0.0:*
ricou@ricou-machine:~$
```

Le port UDP 9997 est en écoute.

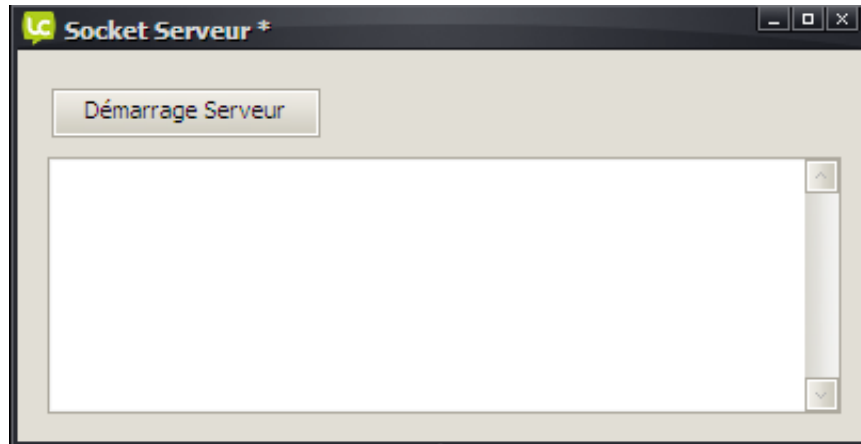
Ensuite, saisissez un message dans le programme UDP client et cliquez sur le bouton send. Une boîte de dialogue doit s'afficher sur le poste Linux.



Fonctionnement de la connexion UDP

## Création de notre premier serveur TCP

Une fois n'est pas coutume, nous utiliserons une machine sous Windows pour réaliser notre premier serveur. Celui-ci aura pour adresse IP 172.16.209.176. Nous aurons besoin d'une pile contenant un bouton et une liste appelée LstMessage. Nommez de suite notre pile Socket Serveur.



Carte de notre serveur

Cliquez droit sur le bouton pour ouvrir une fenêtre de l'éditeur de code. Tout d'abord, nous saisisons le code associé au bouton Démarrage Serveur.

**on** mouseUp

**accept** connections **on** port 20001 with message "MachineConnecte"

**end** mouseUp

Explication : On accepte les connexions TCP entrantes sur le port 20001. Pour chaque connexion réussie, on exécutera la procédure (Message Handler) MachineConnecte. Il faut créer le reste du code.

**on** MachineConnecte ipMachine

**read** from socket ipMachine until **return** with message "nouveauMessage"

**end** MachineConnecte

Explication : La connexion est établie, on initialise la procédure nouveauMessage. Le flux TCP utilisera cette procédure tant que la connexion est active.

**on** nouveauMessage ipMachine theMessage

**put** ipMachine && ":" && theMessage & **return** after field "LstMessage"

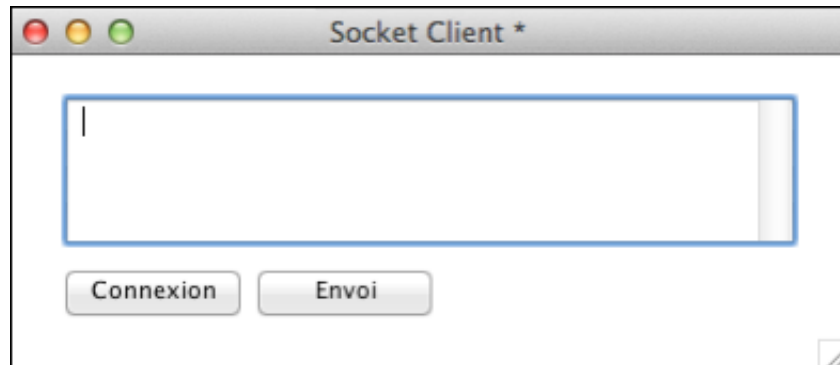
**read** from socket ipMachine until **return** with message "nouveauMessage"

**end** nouveauMessage

Explication : les données du client sont récupérées par la procédure `nouveauMessage`. On récupère l'adresse IP et le message dans la liste `LstMessage`. Chaque message est délimité par un retour chariot. La dernière ligne permet de traiter les messages suivants.

## Création de notre programme client TCP

Le programme client va être réalisé dans un environnement Mac. Nous avons besoin d'une pile contenant deux boutons et une liste. Nommez notre projet `Socket client`.



Pile Socket Client

### Bouton connexion

Ce bouton servira à initialiser la connexion TCP. Voici le code.

```
on mouseUp
  open socket to "172.16.209.176:20001"
  if the Result <> "" then
    put "result:" && the result
  end if
end mouseUp
```

Explication : la commande `open socket` tente l'ouverture d'une connexion TCP avec le port 20001 et l'adresse du serveur 172.16.209.176. Si la connexion est déjà active, on affiche le résultat dans la fenêtre `MessageBox`.

### Bouton envoi

Ce bouton servira à envoyer le message stocké dans la liste `LstMessage`. Voici le code.

```
on mouseUp
  write field "LstMessage" & return to socket "172.16.209.176:20001"
```

**end mouseUp**

Explication : le contenu de LstMessage est envoyé vers le serveur 172.16.209.176 avec le port 20001.

## Fonctionnement

Il faut démarrer notre serveur en cliquant sur le bouton Démarrage Serveur. Afin de vérifier son bon fonctionnement, utilisez la commande netstat -an en ligne de commande.

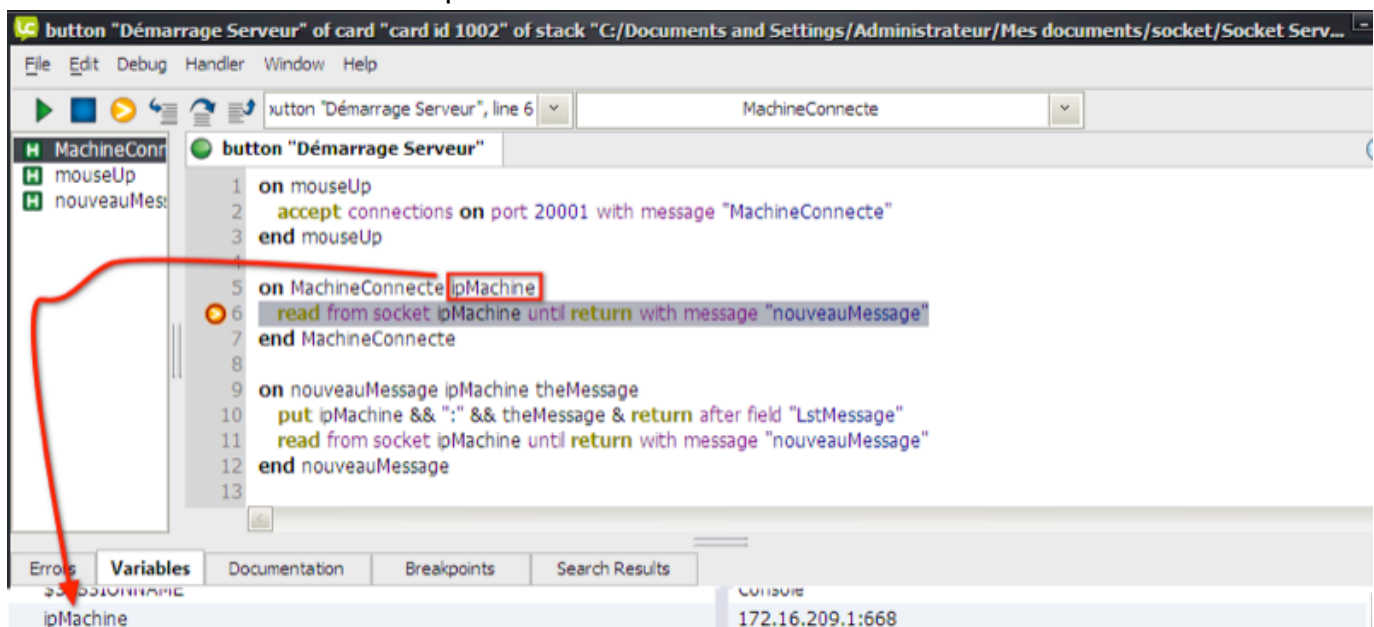
```
C:\Documents and Settings\Administrateur>netstat -an

Connexions actives

Proto  Adresse locale      Adresse distante     État
TCP    0.0.0.0:990         0.0.0.0:0            En écoute
TCP    0.0.0.0:20001       0.0.0.0:0            En écoute
TCP    127.0.0.1:5679      0.0.0.0:0            En écoute
TCP    127.0.0.1:7438      0.0.0.0:0            En écoute
TCP    172.16.209.176:139  0.0.0.0:0            En écoute
UDP    172.16.209.176:137  *:*
```

Le port TCP 20001 est en écoute (LISTEN)

Ensuite, cliquez sur le bouton connexion du client. Un point d'arrêt dans la procédure MachineConnecte nous indique l'établissement de la connexion.



Etablissement de la connexion TCP

Nous pouvons vérifier la connexion TCP côté serveur :



```
>netstat -an | grep 20001
```

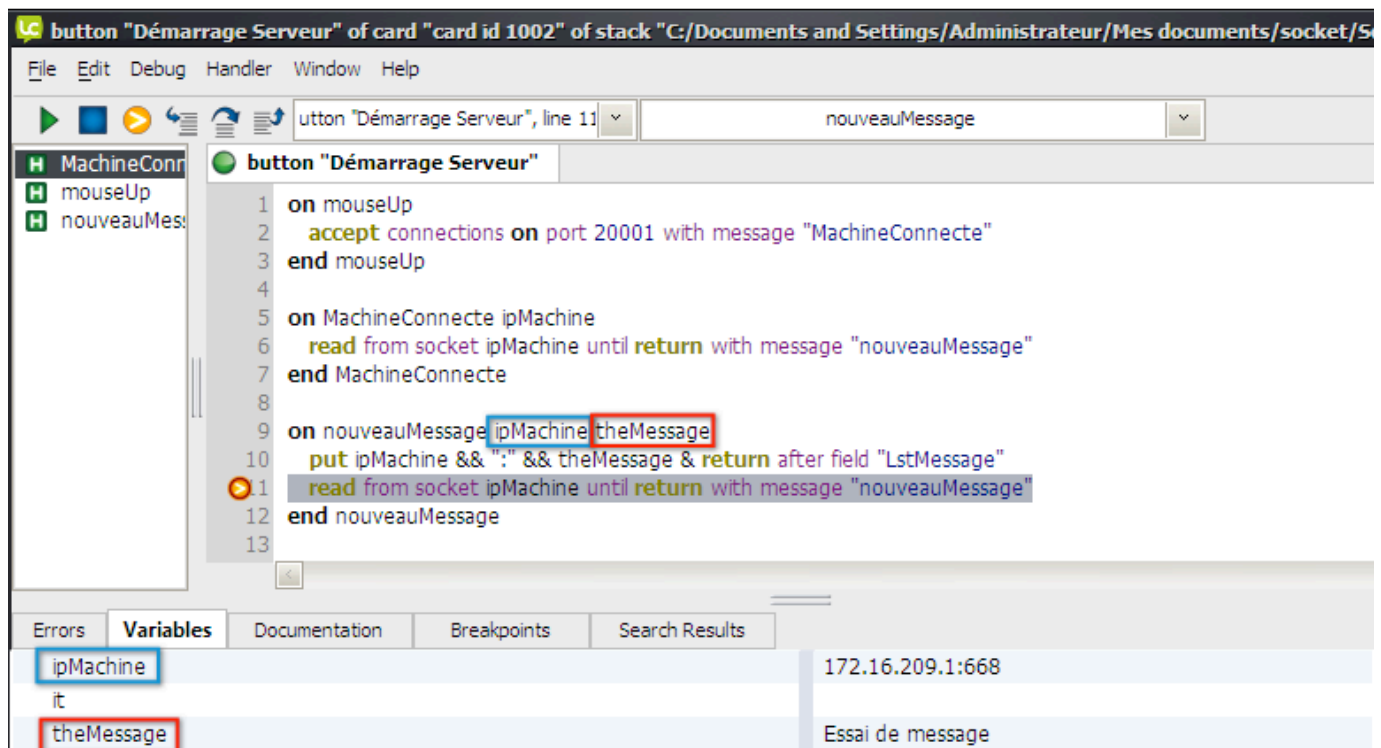
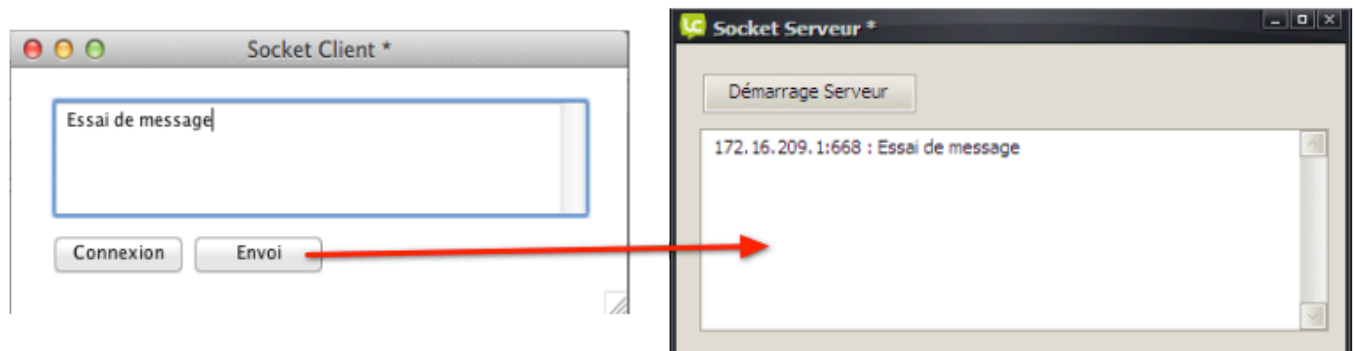
```
TCP 172.16.209.176:20001 172.16.209.1:51545 Établie
```

et côté client :

```
>netstat -an | grep 20001
```

```
tcp4 0 0 172.16.209.1:51545 172.16.209.176:20001 ESTABLISHED
```

Continuons en envoyant un texte au serveur. Côté client, nous saisissons "essai de message" et cliquons sur le bouton envoyer.



### Fonctionnement du flux TCP




Vous avez vu un bref aperçu de ce que l'on peut faire avec Livecode et les sockets BSD.

N'hésitez pas à améliorer ces petits programmes. A vos claviers !

J'ai rajouté un article sur le cryptage des données UDP que vous trouverez [ici](#)

(../../../../code/LiveCodeSocket/LiveCodeSocket\_Secure/)

0 Commentaires Sugarbug  Règles de confidentialité de Disqus  1 S'identifier ▼

 Recommander  Tweet  Partager Les meilleurs ▼



Commencer la discussion...

S'IDENTIFIER AVEC

OU INSCRIVEZ-VOUS SUR DISQUS 

Nom

Soyez le premier à commenter.

 S'abonner  Ajoutez Disqus à votre site web !Ajouter DisqusAjouter

© 2020 Eric Coquard [Contact me \(mailto:eric.coquard@sugarbug.fr\)](mailto:eric.coquard@sugarbug.fr) Last modified: 06/30/2020