THE UNIVERSITY OF NGAOUNDERE

FACULTY OF SCIENCE

DEPARTMENT OF COMPUTER

UNIVERSITE DE NGAOUNDERE

FACULTE DES SCIENCES

DEPARTEMENT DE L'INFORMATIQUE

RAPPORT / PROJET N°7 / TECHNIQUE DE SIMULATION

THEME : **Performances evaluation in a wireless network using UDP sockets:**
Avec 2 et puis 50 machines.

LA LISTE DES PARTICIPANTS

N°	Noms et prénoms	Matricules
1	VARMANTCHAONALA MOUDINA CHARLES	16B015FS
2	TEMANI TEGABWE KAKMO	18B729FS
3	TCHUIGANG DERIC	16A998FS
4	TCHOUKOUEGNO DE MOFO GABBI EVRARD	16B140FS
5	TCHOBYANG ALBERT	14A345FS
6	ZOUFANE YOUGOUDA EMMANUE	16A014FS
7	YCHO GUEIME LOWOL	17B527FS
8	YIMBOU WAKAM FABIOLA	16B143FS
9	TSASSE FOTIO CARIN	15A480FS

TP Technique de Simulation
Réseaux.
M1SLED, Semestre 2, 2019/2020
Sous la Supervision de :
l'enseignante Mme Zongo

Année académique 2019/2020

Vendredi le 19 Juin 2020

I- La procédure de communication entre 2 sockets UDP :

◦ Ci-dessous, nous donnons le principe de fonctionnement d'une socket UDP

✓ Le module **udpApp** du framework INET contient déjà :

- Le module qui implemente le **socket udp** que nous voulons utiliser ;
- **Alors pour créer une socket**, il nous suffit de preciser dans le module **udpApp** les éléments suivants :

udpApp[*].destAddresses ='' ici on met l'adresse de destination '' et udpApp.destPort= ici on met le port de destination ou d'écoute

▪ Création de la socket

Nous montrons ici comment on crée la socket sous omnet++.

▪Une socket de façon simple est un couple :

(Adresse IP d'une machine ; le port d'écoute de la machine)

▪Sous omnet++ dans le projet pour mettre en place notre socket,

nous créons un fichier **.cc** dans lequel devons mettre le code suivant :

```
#include <omnetpp.h>
#include "UDPAppBase.h"
#include "UDPSocket.h"
#include "UDPControlInfo_m.h"

void UDPAppBase::bindToPort(int port)
{
    EV << "Binding to UDP port " << port << endl;

    // TODO UDPAppBase should be ported to use UDPSocket sometime, but for now
    // we just manage the UDP socket by hand...
    cMessage *msg = new cMessage("UDP_C_BIND", UDP_C_BIND);
    UDPControlInfo *ctrl = new UDPControlInfo();
    ctrl->setSrcPort(port);
    ctrl->setSockId(UDPSocket::generateSocketId());
    msg->setControlInfo(ctrl);
    send(msg, "udpOut");
}

void UDPAppBase::sendToUDP(cPacket *msg, int srcPort, const IPvXAddress& destAddr, int destPort)
{
    // send message to UDP, with the appropriate control info attached
    msg->setKind(UDP_C_DATA);

    UDPControlInfo *ctrl = new UDPControlInfo();
    ctrl->setSrcPort(srcPort);
    ctrl->setDestAddr(destAddr);
    ctrl->setDestPort(destPort);
    msg->setControlInfo(ctrl);

    EV << "Sending packet: ";
    printPacket(msg);

    send(msg, "udpOut");
}
```

✓ Description du code

Ce bout de code est un code qui permet de créer **une socket UDP**

précisant l'adresse IP : **10.0.0.2** de la machine et son port

d'écoute :**1000** et envoie un message à la machine sur cette socket.

En suite ferme la socket avec socket.close() ; Ici il s'agit d'attacher une socket à une machine mais le fichier udp qui implémente la socket est **UDPsocket.h** que nous avons importé

- ✓ Sachant déjà comment on crée une socket sous omnet++, la suite de notre travail va se subdiviser en deux parties :

II- Simulation d'une socket UPD avec **deux machines** : Un Client et un Server ;
III- Simulation avec **50 machines** : Un **Sender et les autres 49 réceptrices**

<u>Note</u> : Au lieu de recréer la socket avec un fichier .cc nous avons directement importé dans notre projet le Framework INET pour deux objectifs :

1. **D'abord pour réutiliser le modèle de la socket UDP déjà implémenté par ce Framework ;**
2. **Pour bénéficier du service du module qui implémente le protocole UDP car l'objectif du TP est aussi de simulation la communisation avec ce protocole ;**

II- Simulation d'une socket UPD avec deux machines : Un Client et un Server ;

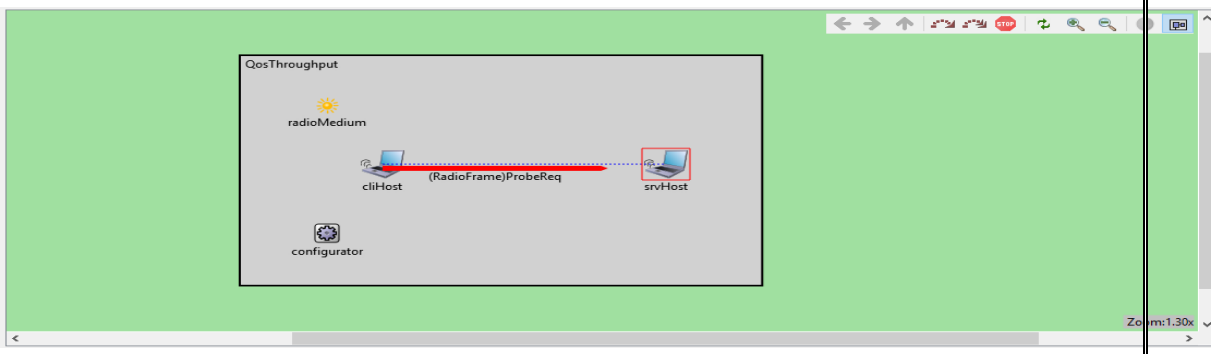
- ✓ Dans cette partie nous allons faire la simulation entre deux machines donc : donc une machine envoie et l'autre reçoit dans un intervalle de temps régulier défini et nous allons aussi présenter et commenter les différents résultats de notre simulation.

- ✓ Le projet que nous allons créer des maintenant dans le cadre de **la simulation de 2 machines** sera exploiter dans la deuxième partie qui consistera **à simuler n=50 machines**. Alors il nous suffira **de créer une variable :**

  Int numHosts ;

  // cette variable nous permettra de conserver le même code et la même topologie réseau et ne modifier dans notre environnement de simulation que le nombre de machines **numHosts à simuler**. Alors nous commencerons par fixer la variable **numHosts= 2** ; pour cette première partie de notre simulation et en suite nous fixerons la variable **numHosts=50 ; pour la simulation avec 50 machines**

- ✓ Pour cette première partie de notre simulation notre fichier de topologie le **.ned** se présentera comme suit en mode Design



✓ Nous remarquons bien sur notre figure qu'en realité la machine qui envoie le packet **UDPbasicAppData ici le 27eme paquet** a besoin de connaitre la socket c'est-à-dire **l'adresse ip : 10.0.0.1** et **le port de réception : 100** de la machine qui récoit, precisé dans notre fichier de configuration **.ini** comme suit :

```
**.sender.numUdpApps = 1
**.sender.udpApp[0].typename = "UDPBasicApp"
**.sender.udpApp[*].destAddresses =  "10.0.0.1"
**.sender.udpApp[0].destPort = 1000
**.sender.udpApp[0].messageLength = 100B
**.sender.udpApp[0].startTime = 10s
**.sender.udpApp[0].sendInterval = 1s

**.numhosts = 2
**.host[*].numUdpApps = 1
**.host[*].udpApp[0].typename = "UDPSink"
**.host[*].udpApp[0].localPort = 1000
```

✓ Cette petite capture nous permet de récuper les informations suivantes :

- Les deux lignes suivantes constituions notre socket udp :

```
**.sender.udpApp[*].destAddresses =  "10.0.0.1"
**.sender.udpApp[0].destPort = 1000
```

- La ligne suivante constituira l'intervalle de temps de l'envoie periodique du message sinon du packet udp :

```
**.sender.udpApp[0].sendInterval = 1s
```

donc apres chaque 1seconde

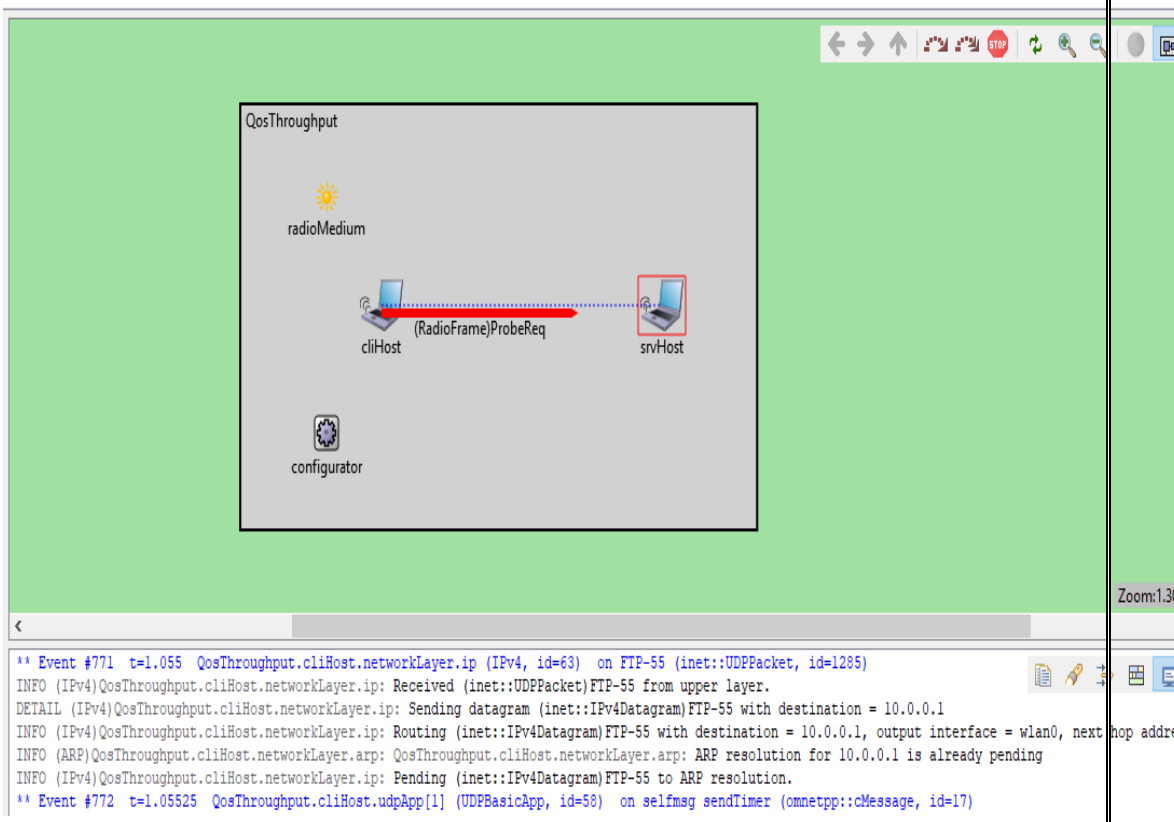
- Les autres lignes nous donnent les informations comme : la taille du message à envoyer et le début de l'envoi ;

✓ Ainsi, après avoir configure notre projet avec le fichier **.ned** et le fichier **.ini nous obtenons les résultats suivants :**

<u>NB</u> : nous rappelons que si nous n'avons pas parler des fichiers .cc qui implémentent les fonctionnalités de la socket et autres c'est par ce que le module 'udpApp' du protocole udp implémente déjà ces fichiers .cc et nous nous n'avons eu qu'à importer le framework INET dans notre projet après l'avoir créé.

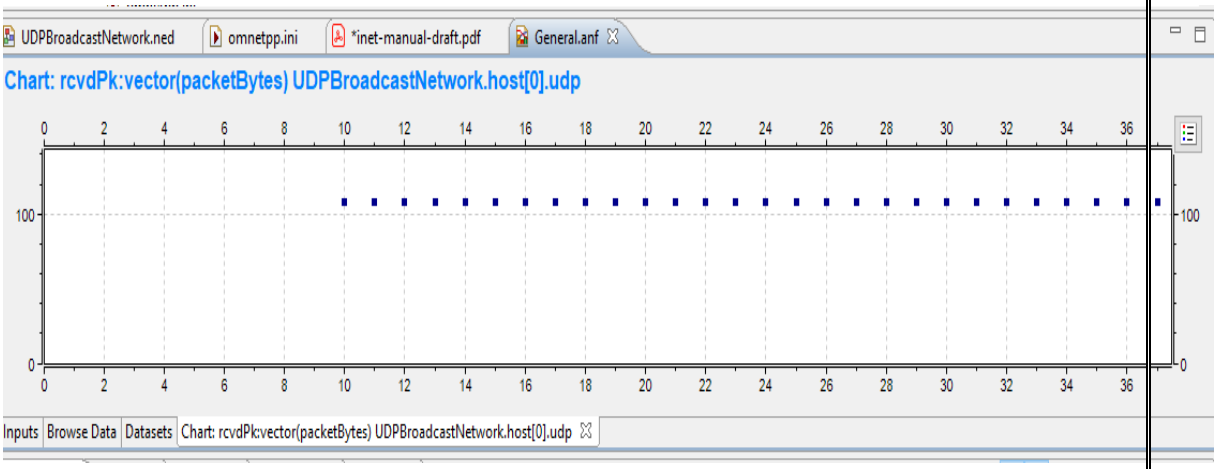
✓ **Résultats :**

- **Temps nécessaire à un client pour envoyer un message au serveur : le Latency, définit ici comme le temps nécessaire pour partir du poste client au poste serveur.**



- Les deux dernières lignes de la simulation montrent que le message est envoyé à partir du **'sender'** au **temps t=1.055** et il arrive en destination à l'Host au temps **t=1.05525 donc on peut voir : Latency=1.05525-1.055 ;**

-pourcentage de réception d'un message ou Packet Delivery Ratio (PDR), diffusé par une machine


UDPBroadcastNetwork.ned
omnetpp.ini
*inet-manual-draft.pdf
General.anf
Chart: rcvdPk:vector(packetBytes) UDPBroadcastNetwork.host[0].udp
Inputs
Browse Data
Datasets
Chart: rcvdPk:vector(packetBytes) UDPBroadcastNetwork.host[0].udp

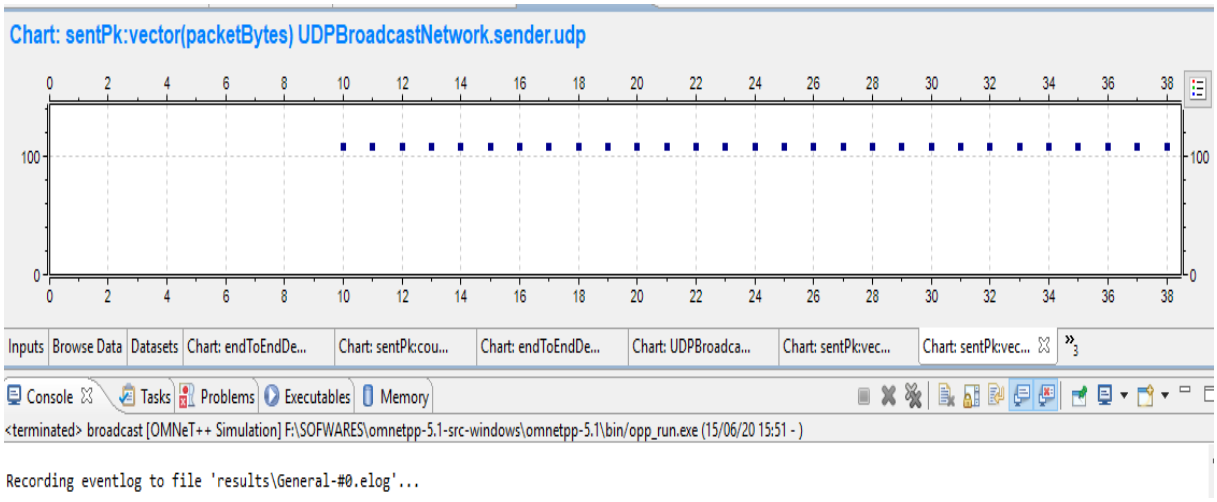
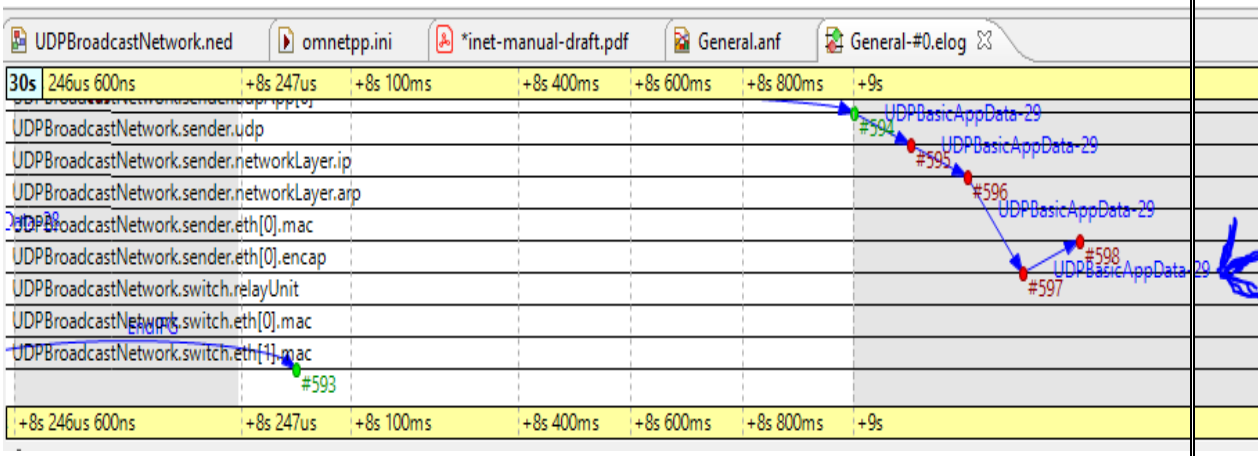
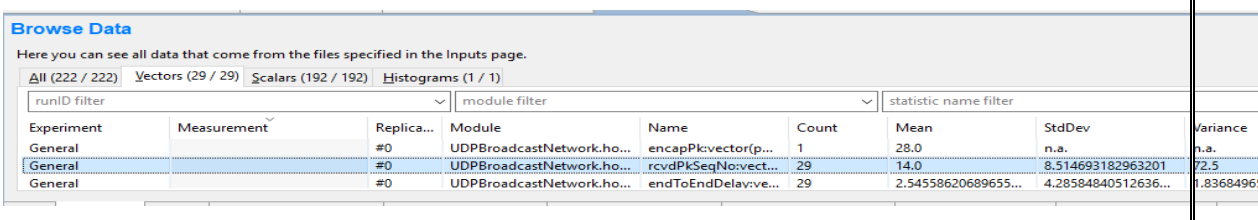
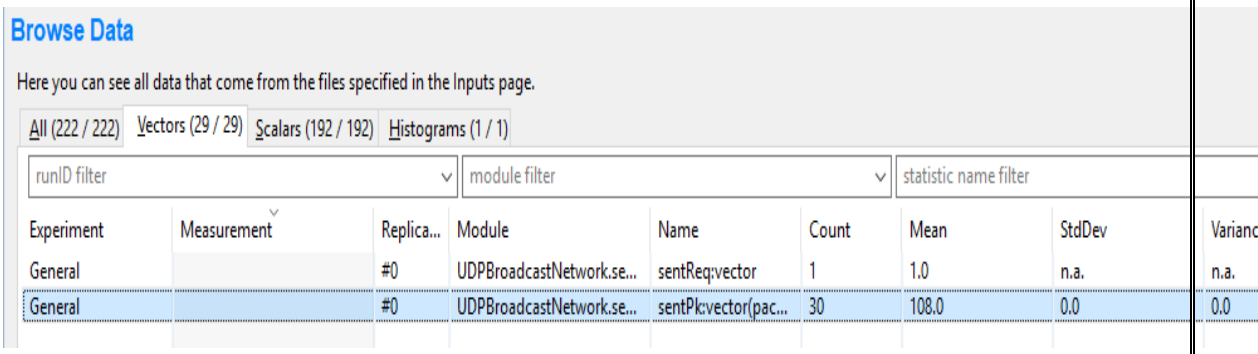


Chart: sentPk:vector(packetBytes) UDPBroadcastNetwork.sender.udp
Inputs
Browse Data
Datasets
Chart: endToEndDe...
Chart: sentPk:cou...
Chart: endToEndDe...
Chart: UDPBroadca...
Chart: sentPk:vec...
Chart: sentPk:vec...
Console
Tasks
Problems
Executables
Memory
<terminated> broadcast [OMNeT++ Simulation] F:\SOFWARES\omnetpp-5.1-src-windows\omnetpp-5.1\bin/opp_run.exe (15/06/20 15:51 -)
Recording eventlog to file 'results\General-#0.elog'...


UDPBroadcastNetwork.ned
omnetpp.ini
*inet-manual-draft.pdf
General.anf
General-#0.elog
30s
246us 600ns
+8s 247us
+8s 100ms
+8s 400ms
+8s 600ms
+8s 800ms
+9s
UDPBroadcastNetwork.sender.udp
UDPBroadcastNetwork.sender.networkLayer.ip
UDPBroadcastNetwork.sender.networkLayer.arp
UDPBroadcastNetwork.sender.eth[0].mac
UDPBroadcastNetwork.sender.eth[0].encap
UDPBroadcastNetwork.switch.relayUnit
UDPBroadcastNetwork.switch.eth[0].mac
UDPBroadcastNetwork.switch.eth[1].mac
#593
#594
#595
#596
#597
#598
UDPBasicAppData-29


Browse Data
Here you can see all data that come from the files specified in the Inputs page.
All (222 / 222)
Vectors (29 / 29)
Scalars (192 / 192)
Histograms (1 / 1)
runID filter
module filter
statistic name filter
Experiment
Measurement
Replica...
Module
Name
Count
Mean
StdDev
Variance
General
#0
UDPBroadcastNetwork.ho...
encapPk:vector(p...
1
28.0
n.a.
n.a.
General
#0
UDPBroadcastNetwork.ho...
rcvdPkSeqNo:vect...
29
14.0
8.514693182963201
72.5
General
#0
UDPBroadcastNetwork.ho...
endToEndDelay:ve...
29
2.54558620689655...
4.28584840512636...
1.8368496

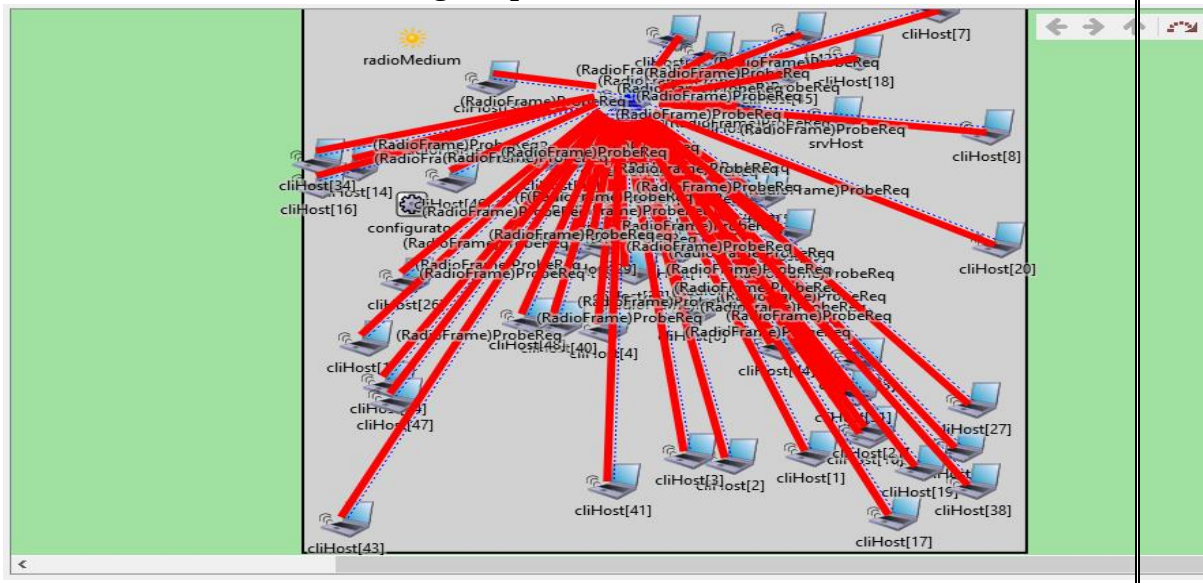


✓ **Ces dernières captures nous montrent clairement (Les deux lignes soulignées) que sur les 30 paquets envoyés entre les deux machines on a reçu 29 paquets.**

III- Simulations avec cinquante (50) machines

✓ **Dans cette deuxième partie où nous simulons notre réseau avec 50 machines, notre réseau du coté code ne changera vraiment pas mais n'aurons seulement qu'a modifier la valeur de la variable numHosts=2 à numHosts=50 simplement et notre réseau de 50 machine est constitué .**

✓ **Notre fichier .ned en mode Design se présentera comme suit :**



- ✓ **La seule modification à faire est la suivante : au niveau du fichier .ini nous le modifions comme suit :**

```
**.sender.numUdpApps = 1
**.sender.udpApp[0].typename = "UDPBasicApp"
**.sender.udpApp[*].destAddresses =  "10.0.0.1 10.0.0.3 10.0.0.4 10.0.0.5 10.0.0.6 10.0.0.7 10.0.0.8 10.0.0.9 10.0.0.10 10.0.0.11 10.0.0.12\
                                      10.0.0.13 10.0.0.14 10.0.0.15 10.0.0.16 10.0.0.17 10.0.0.18 10.0.0.19 10.0.0.20 10.0.0.21 10.0.0.23 \
                                      10.0.0.24 10.0.0.25 10.0.0.26 10.0.0.27 10.0.0.28 10.0.0.29 10.0.0.30 10.0.0.31 10.0.0.32 10.0.0.33\
                                      10.0.0.34 10.0.0.35 10.0.0.35 10.0.0.36 10.0.0.37 10.0.0.38 10.0.0.39 10.0.0.40 10.0.0.41 10.0.0.42\
                                                10.0.0.43 10.0.0.44 10.0.0.45 10.0.0.46 10.0.0.47 10.0.0.48 10.0.0.49 10.0.0.50"
**.sender.udpApp[0].destPort = 1000
**.sender.udpApp[0].messageLength = 100B
**.sender.udpApp[0].startTime = 10s
**.sender.udpApp[0].sendInterval = 1s
```

En fait la modification se trouve seulement au niveau de la ligne :

```
**.sender.udpApp[*].destAddresses =  "10.0.0.1 10.0.0.3 10.0.0.4 10.0.0.5 10.0.0.6 10.0.0.7 10.0.0.8 10.0.0.9 10.0.0.10 10.0.0.11 10.0.0.12\
                                      10.0.0.13 10.0.0.14 10.0.0.15 10.0.0.16 10.0.0.17 10.0.0.18 10.0.0.19 10.0.0.20 10.0.0.21 10.0.0.23 \
                                      10.0.0.24 10.0.0.25 10.0.0.26 10.0.0.27 10.0.0.28 10.0.0.29 10.0.0.30 10.0.0.31 10.0.0.32 10.0.0.33\
                                      10.0.0.34 10.0.0.35 10.0.0.35 10.0.0.36 10.0.0.37 10.0.0.38 10.0.0.39 10.0.0.40 10.0.0.41 10.0.0.42\
                                                10.0.0.43 10.0.0.44 10.0.0.45 10.0.0.46 10.0.0.47 10.0.0.48 10.0.0.49 10.0.0.50"
```

- ✓ Cette modification est du au fait que, comme nous l'avons déjà souligner ci-haut, pour qu'une machine puis envoyer un message dans notre réseau il faut qu'elle sache qu'elle est l'adresse IP et le port d'écoute de la machine à laquelle elle veut envoyer le message.
- ✓ Alors pour que notre machine **'sender'** puisse envoyé un message à toute nos cinquante (50) machines elles doit connaitre toutes les adresses des cinquante machines aussi.
- ✓ Pour ce qui est des ports, la ligne suivante nous permet de dire que toutes les machines ont des adresses IP différentes mais ont le même port d'écoute = 1000 ;

```
**.sender.udpApp[0].destPort = 1000
```

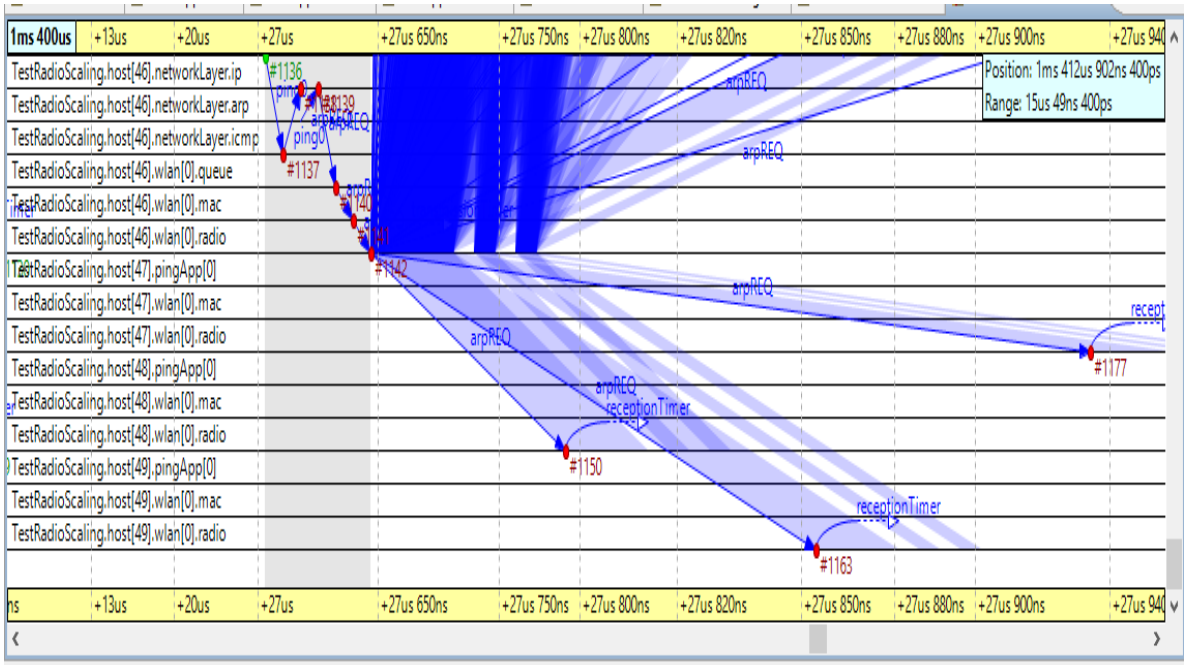
- ✓ Alors le sender prend chaque adresse ci-haute citéé et envoie le message à cette machine sur le port 100

- ✓ **<u>Résultats de la simulation du réseau avec les 50 machines</u>** :

- Temps nécessaire à un client pour envoyer un message au serveur : le *Latency*, définit ici comme le temps nécessaire pour partir du poste client au poste serveur.

- ✓ Sur cette capture ci-dessus, on voit bien qu'au début de la communication le temps de simulation au niveau de chaque récepteur est initialisé à **t=0**
- ✓ **Et** la dernière ligne en bleu, nous indique le temps de la réception **t=0.000536440382**
- ✓ **Donc le temps de latence est = 0.000536440382-0=0.000536440382**

-pourcentage de réception d'un message ou Packet Delivery Ratio (PDR), diffusé par une machine



- ✓ Sur cette capture chacune des machines réceptrices host[1] jusqu'au host[49] a reçu le message provenant du Sender (le host[0])
- ✓ Donc on a un **PDR de 100%**