



RAPPORT DE TP

EVALUATION DES PERFORMANCES DES RÉSEAUX

Effectué par :

BOU SABA Elie

William Staël KAYO

Sous la direction de : M. Wahabou ABDOU

Table des matières

<i>TP 1 : Modèles de propagation radio</i>	2
I. Introduction:.....	2
II. Analyse de fichiers de traces:.....	2
III. Conclusion	6
<i>TP 2: Simulation de réseaux ad hoc</i>	6
I. Introduction:.....	6
II. Travail à faire:.....	6
III. Conclusion	10
<i>TP 3: Routage ad hoc</i>	10
I. Introduction :.....	10
II. Protocole DSDV:	10
III. Routage multi-sauts:	12
<i>TP 4: Routage ad hoc proactif</i>	15
I. Introduction:.....	15
II. Protocole DSR :	15
III. Routage multi-sauts:	16
<i>TP 5: Routage AODV et trafic FTP</i>	18
I. Introduction:.....	18
II. Travail à faire	18
III. Conclusion	19

Liste de figures

Figure 1 : Script Shell qui calcule la distance d'un nœud vers le nœud 0 et le nombre de trames reçues	3
Figure 2 : Script AWK	3
Figure 3 : Courbe du taux de réception pour le Free Space	4
Figure 4 : Courbe du taux de réception pour le Two-ray ground.....	5
Figure 5 : Courbe du taux de réception pour le Shadowing.....	5
Figure 6 : Calcul des trames émises, reçues et perdues	6
Figure 7 : Distance et taux de réception pour le mode Free Space	7
Figure 8 : Graphe représentant le taux de réception par rapport à la distance inter-nœud pour le mode Free Space	7
Figure 9 : Distance et taux de réception pour le mode Two-Ray Ground	8
Figure 10 : Graphe représentant le taux de réception par rapport à la distance inter-nœud pour le mode Two-Ray Ground	8
Figure 11 : Distance et taux de réception pour le mode Shadowing.....	9
Figure 12 : Graphe représentant le taux de réception par rapport à la distance inter-nœud pour le mode Shadowing.....	9
Figure 13 : les 20 premières lignes du fichier de traces	11
Figure 14 : Nombre de paquets émis et reçu pour t=5.0s	12
Figure 15 : Nombre de paquets émis et reçu pour t=0s	12
Figure 16 : Génération de 4 nœuds	12
Figure 17 : Nombre de trames émises.....	13
Figure 18 : Script AWK calculant le temps d'acheminement moyen des paquets.....	14
Figure 19 : Nombre de paquets émises et reçues	15
Figure 20 : Nombre de paquets émises et reçues pour 5.4Mbit/s	16
Figure 21 : Fichier de traces après déplacement d'un nœud	16
Figure 22 : Nombre de trames émises.....	17
Figure 23 : Script AWK calculant le temps d'acheminement moyen des paquets.....	17
Figure 24 : Noeud origine de la requete de decouverte de route	18
Figure 25 : Noeud origine de la reponse de decouverte de route.....	19

TP 1 : Modèles de propagation radio

I. Introduction:

Dans ce premier TP, nous allons exploiter les modèles de propagation des ondes radio (Free Space, Two-ray Ground et Shadowing) en étudiant les fichiers de traces obtenues après l'exécution des simulations correspondantes et pouvoir par la suite tracer les courbes du taux de réception des paquets en fonction de la distance.

II. Analyse de fichiers de traces:

1) Pour pouvoir récupérer les différents fichiers de traces, nous nous sommes partagés cela par clé USB car il n'y avait pas la possibilité de les récupérer via la machine de l'enseignant.

2) La récupération des lignes du fichier de traces FreeSpace.tr concernant la réception de message est obtenue avec la commande grep :

```
grep ^r FreeSpace.tr | less
```

Dans ce cas, on filtre la recherche en prenant que les lignes commençant par la lettre 'r' et donc que les messages de réception.

3) Le filtrage de manière à ne conserver que les lignes relatives à la couche 2 du nœud 5 est réalisé en enchainant plusieurs opérations grep :

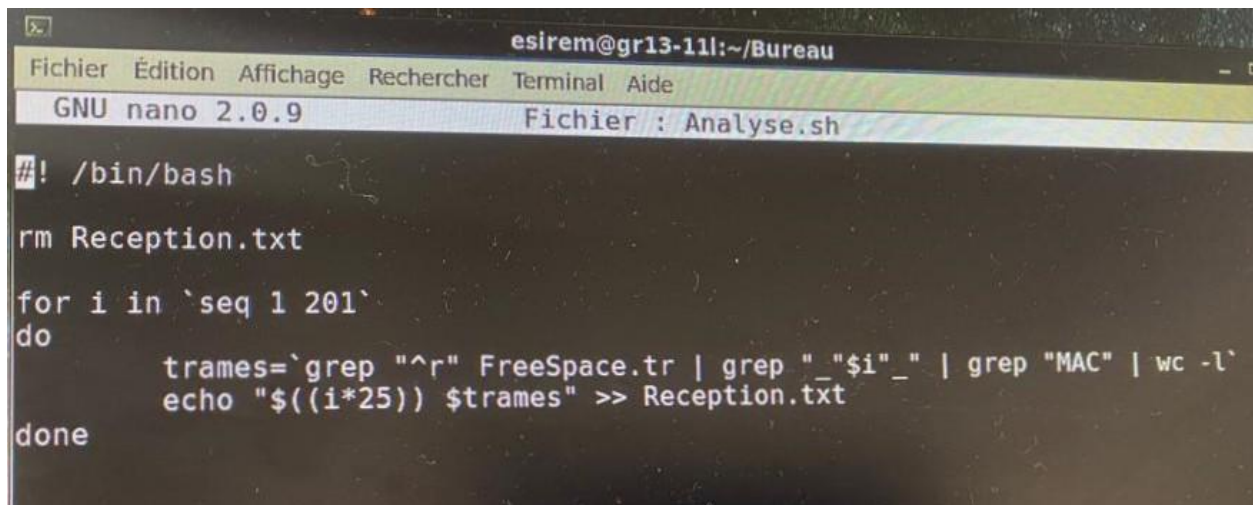
```
grep ^r FreeSpace.tr | grep _5_ | grep MAC | less
```

4) Pour trouver le nombre de trames qui ont été reçus par le nœud 5 on utilise la commande « wc -l » qui va nous compter le nombre de lignes trouvés par la commande exécutée avant. Pour se faire :

```
grep ^r FreeSpace.tr | grep _5_ | grep MAC | wc -l
```

En fait, cette ligne de commande a retourné 1000

5)

A screenshot of a terminal window with a menu bar at the top containing 'Fichier', 'Édition', 'Affichage', 'Rechercher', 'Terminal', and 'Aide'. Below the menu bar, it says 'GNU nano 2.0.9' and 'Fichier : Analyse.sh'. The terminal content shows a shell script starting with a shebang, removing a file, and then looping from 1 to 201 to calculate the number of frames received for each node.

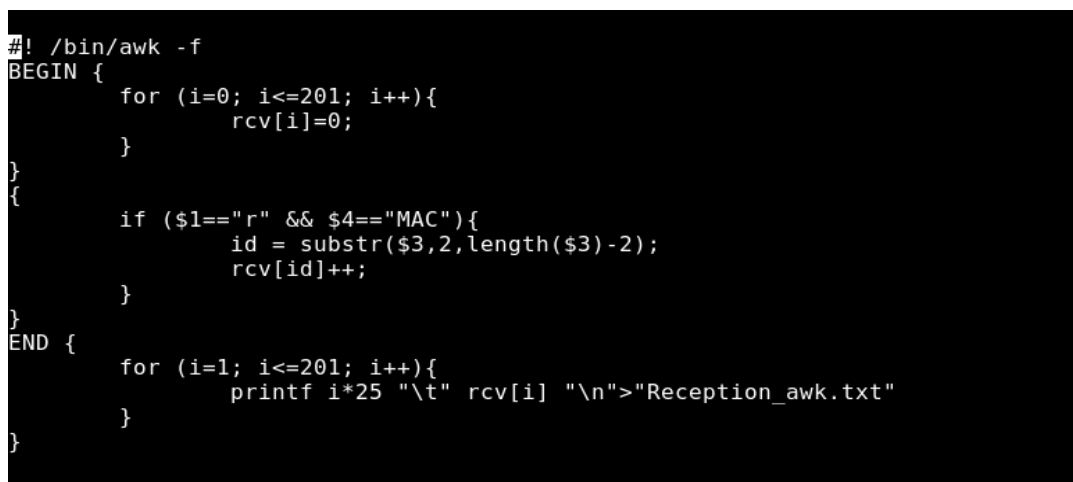
```
#!/bin/bash
rm Reception.txt
for i in `seq 1 201`
do
    trames=`grep "^r" FreeSpace.tr | grep "_$i_" | grep "MAC" | wc -l`
    echo "$((i*25)) $trames" >> Reception.txt
done
```

Figure 1 : Script Shell qui calcule la distance d'un nœud vers le nœud 0 et le nombre de trames reçues

Pour écrire le script Shell, on a créé une boucle allant de 1 à 201 car on a 201 nœuds. Dans cette boucle, on a défini une variable qui va contenir le nombre de trames (MAC) contenues pour chaque nœud et c'est '\$i' qui correspond à quel nœud on fait le calcul et cela est incrémenté de 1 à chaque itération.

Ensuite, on va insérer et afficher à chaque itération de la boucle dans le fichier Reception.txt une ligne qui va contenir la distance du nœud courant vers le nœud 0 et pour y avoir ce résultat on multiplie '\$i' par 25 car entre un nœud et l'autre il y a 25 mètres.

6)

A screenshot of a terminal window showing an AWK script. The script initializes an array 'rcv' for nodes 0 to 201, then processes a file 'FreeSpace.tr' to count MAC frames for each node, and finally writes the results to 'Reception_awk.txt' with a distance multiplier of 25.

```
#!/bin/awk -f
BEGIN {
    for (i=0; i<=201; i++){
        rcv[i]=0;
    }
}
{
    if ($1=="r" && $4=="MAC"){
        id = substr($3,2,length($3)-2);
        rcv[id]++;
    }
}
END {
    for (i=1; i<=201; i++){
        printf i*25 "\t" rcv[i] "\n">"Reception_awk.txt"
    }
}
```

Figure 2 : Script AWK

Dans la partie BEGIN du script AWK, on a initialisé un tableau 'rcv' ayant comme taille le nombre de nœuds d'où 201 et on a rempli chaque case du tableau par 0. Dans le second bloc, on a utilisé '\$1' et '\$4' qui correspondent respectivement au 1^{er} et 4^{ème} élément de la ligne traitée et donc ici on vérifie si la ligne débute par la lettre 'r' qui convient aux messages reçus et on vérifie aussi si le '\$4' correspond au 'MAC'. Ensuite on récupère l'identifiant qui va nous permettre de savoir où on doit remplacer dans le tableau donc quel nœud. Enfin, on va afficher chaque nœud qu'on récupère du tableau 'rcv' avec la distance du nœud vers le nœud 0.

7) Avec l'outil xgraph on trace la courbe de Free Space :

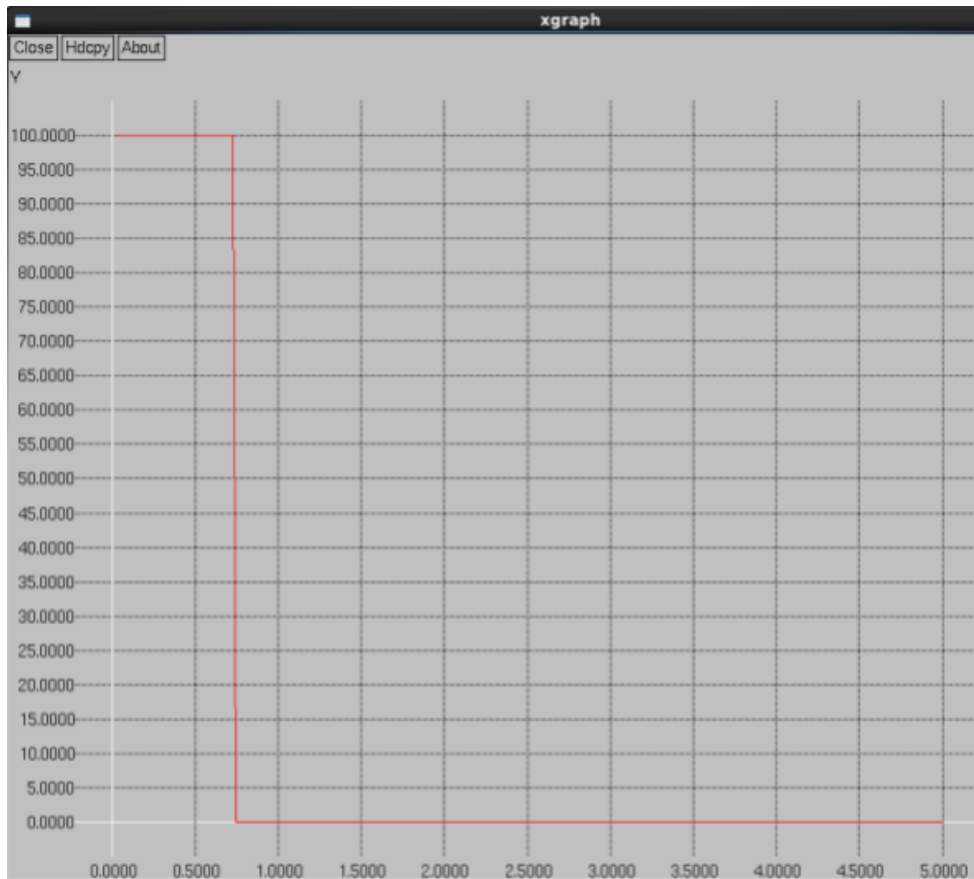


Figure 3 : Courbe du taux de réception pour le Free Space

8)



Figure 4 : Courbe du taux de réception pour le Two-ray ground

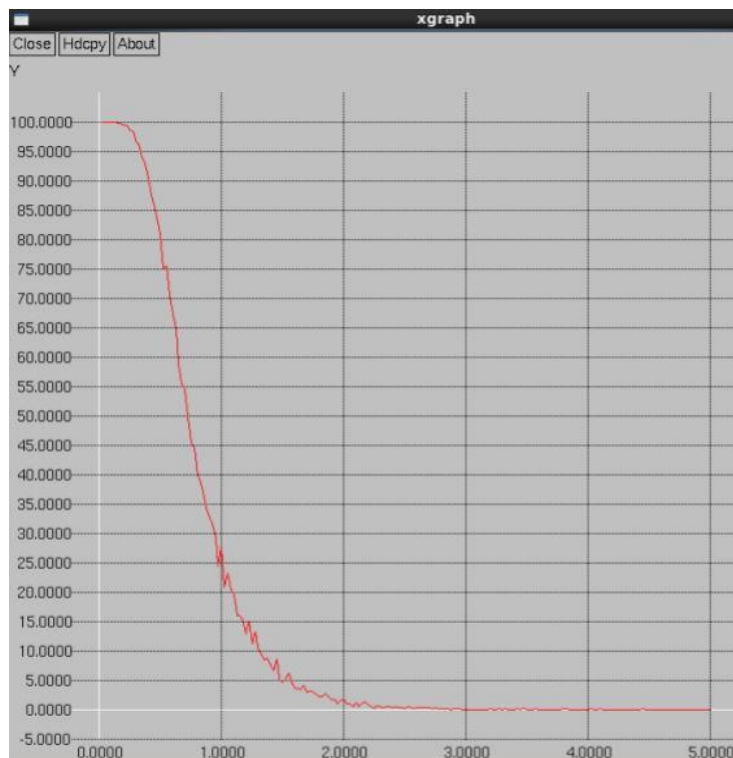


Figure 5 : Courbe du taux de réception pour le Shadowing

III. Conclusion

Tous les résultats obtenus correspondent bien à ce qui était prévu dans l'énoncé de ce premier TP.

TP 2: Simulation de réseaux ad hoc

I. Introduction:

Dans ce deuxième TP, nous allons étudier un réseau ad hoc ayant 2 nœuds qui communiquent en communication hertzienne sur différentes distances entre eux pour les 3 modes de propagation qu'on a exploiter dans le premier TP et on va observer le taux de réception des messages.

II. Travail à faire:

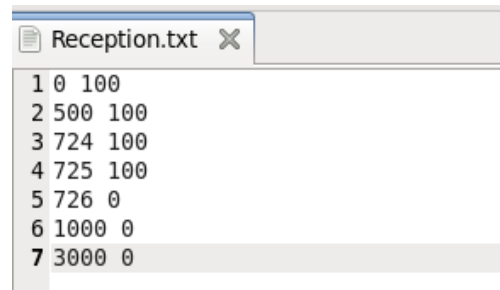
1) Le code TCL fournit dans l'énoncé du TP initialise tous les paramètres dont on a besoin. On modifie juste le mode de propagation et les coordonnées des 2 nœuds dans chaque cas pour l'analyse.

2-3-4-5) On a pris plusieurs distances différentes entre les 2 nœuds pour calculer à chaque fois le taux de réception. En fait, en fonction du mode de propagation utilisé, on a défini chaque fois les coordonnées des nœuds. Commençons par le mode Free Space on a calculé par les commandes grep avec les différentes paramètres ('s', 'r', 'd') qui signifient les trames émises, reçues et dropés pour les différentes distances entre les 2 nœuds. On illustre dans la figure 6 un cas à une distance de 726 mètres entre les nœuds et on obtient les résultats suivants :

```
[esirem@gr13-11l Bureau]$ grep ^s TP2.tr | grep cbr | grep "_0_" | grep "MAC" |  
wc -l  
0  
[esirem@gr13-11l Bureau]$ grep ^r TP2.tr | grep cbr | grep "_1_" | grep "MAC" |  
wc -l  
0  
[esirem@gr13-11l Bureau]$ grep ^D TP2.tr | grep cbr | grep "_1_" | grep "MAC" |  
wc -l  
0  
[esirem@gr13-11l Bureau]$
```

Figure 6 : Calcul des trames émises, reçues et perdues

Ensuite, on a calculé le taux de réception pour les différentes distances inter-nœuds et on a remplacé ces valeurs dans un fichier texte illustré dans la figure 7. La première colonne c'est la distance entre les 2 nœuds et la deuxième colonne c'est le taux de réception qu'on a calculé pour chaque distance.



1	0	100
2	500	100
3	724	100
4	725	100
5	726	0
6	1000	0
7	3000	0

Figure 7 : Distance et taux de réception pour le mode Free Space

Après ça, on a dessiné la courbe pour le mode Free Space en utilisant l'outil xgraph et on remarque que le modèle Free Space possède un taux de réception qui chute directement à 0 après 725 mètres de distance.

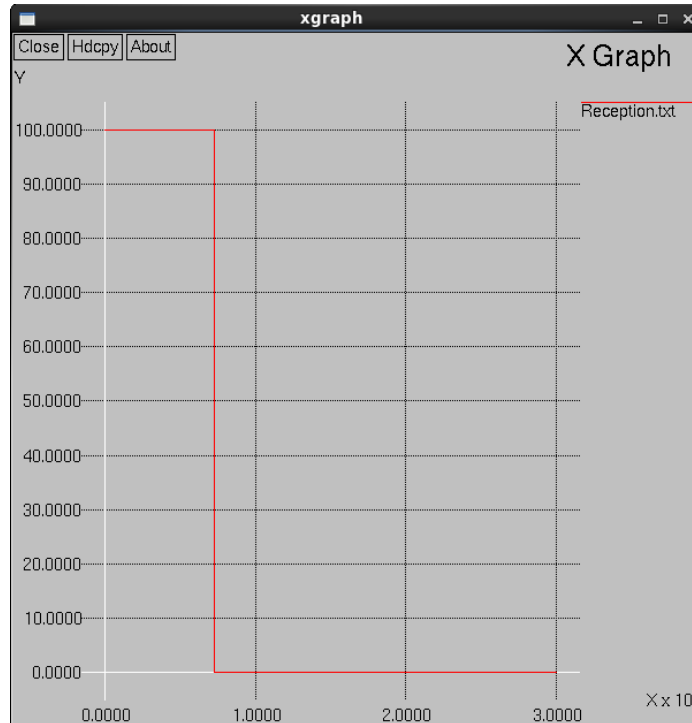


Figure 8 : Graphe représentant le taux de réception par rapport à la distance inter-nœud pour le mode Free Space

Dans le cas du Two-Ray Ground on obtient les résultats suivantes (figure 9-10) :

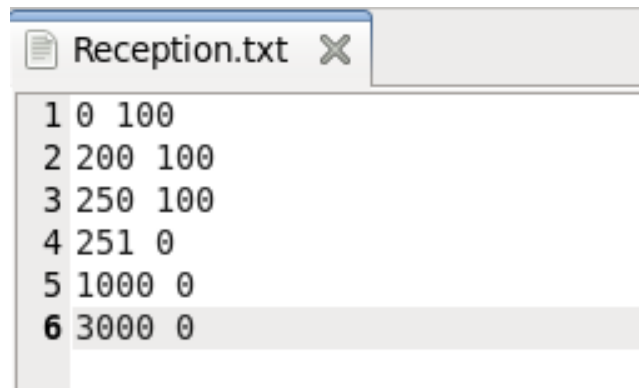


Figure 9 : Distance et taux de réception pour le mode Two-Ray Ground

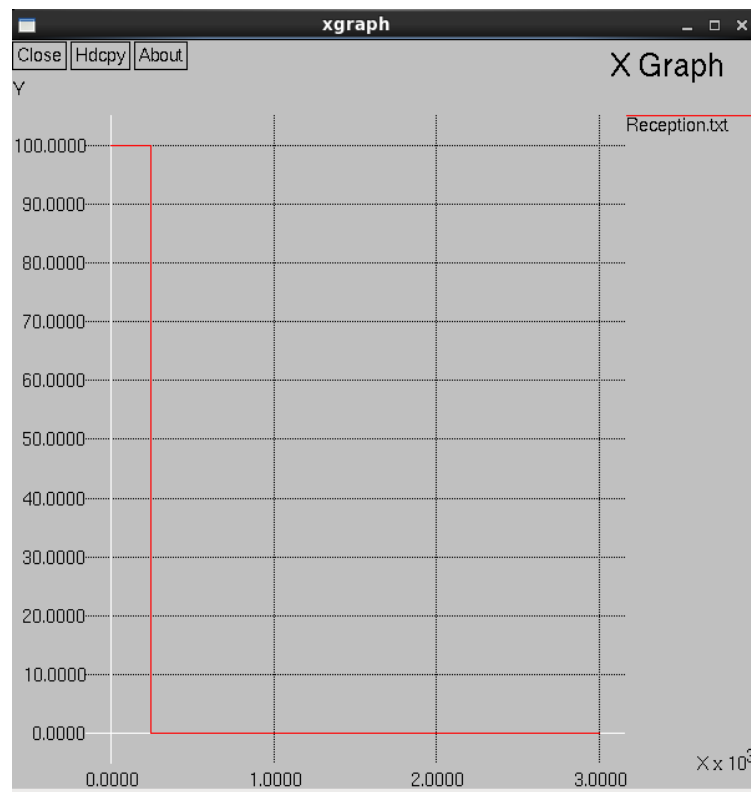
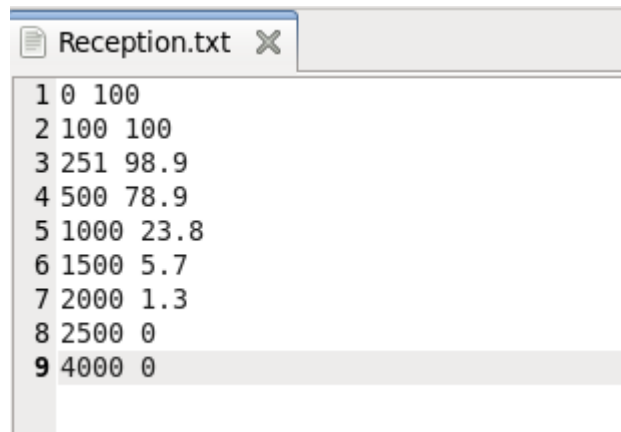


Figure 10 : Graphe représentant le taux de réception par rapport à la distance inter-nœud pour le mode Two-Ray Ground

On remarque que le modèle Two-Ray Ground possède un taux de réception qui chute directement à partir de 251 mètres. Une chute plus rapide que le mode Free Space.

Et finalement pour le mode Shadowing on a les résultats suivants (figure 11-12)



1	0	100
2	100	100
3	251	98.9
4	500	78.9
5	1000	23.8
6	1500	5.7
7	2000	1.3
8	2500	0
9	4000	0

Figure 11 : Distance et taux de réception pour le mode Shadowing

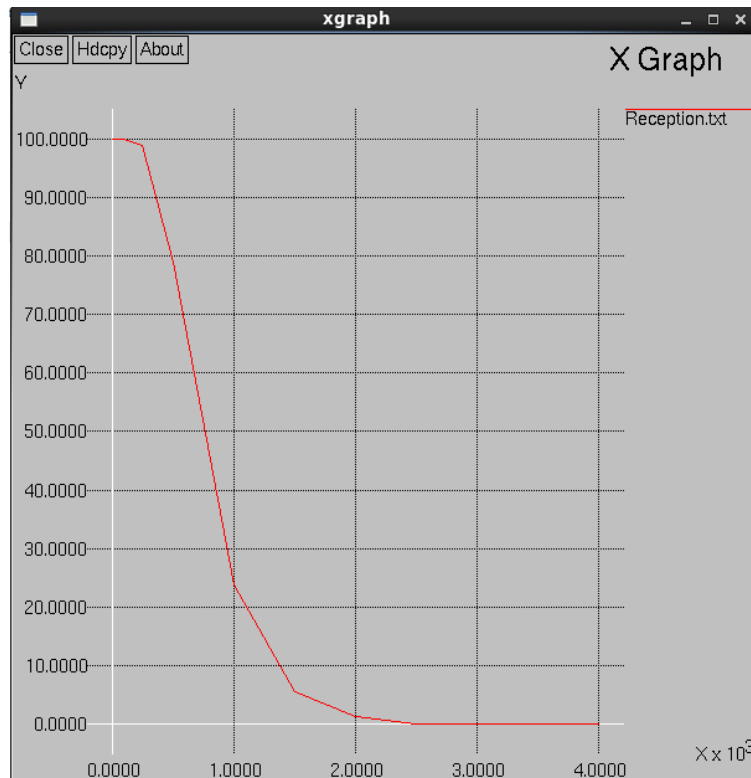


Figure 12 : Graphe représentant le taux de réception par rapport à la distance inter-nœud pour le mode Shadowin

Dans ce cas du modèle Shadowing on remarque qu'on obtient à peu près la courbe obtenue au TP1.

III. Conclusion

En générant nous même nos propres fichiers traces à partir du script Tcl donné, on obtient pratiquement le même résultat que le TP1. Les courbes sont pratiquement similaires.

TP 3: Routage ad hoc

I. Introduction :

Ce TP va nous permettre d'exploiter les différentes phases de la découverte des routes dans les réseaux ad hoc et l'acheminement des paquets. Pour réaliser ce TP, on a utilisé le mode de propagation Two-ray ground qui est déterministe.

II. Protocole DSDV:

1) Pour pouvoir utiliser le protocole DSDV on modifie le code :

```
set val(rp) DSDV
```

2) Modifions les coordonnées des nœuds :

```
set node_(0) [$ns node]
```

```
$node_(0) set X_ 1000.0
```

```
$node_(0) set Y_ 1500.0
```

```
$node_(0) set Z_ 0.0
```

```
set node_(1) [$ns node]
```

```
$node_(1) set X_ 1245.0
```

```
$node_(1) set Y_ 1500.0
```

```
$node_(1) set Z_ 0.0
```

3) Pour que le trafic cbr commence à la 5^{ème} seconde :

```
set val(dataStart) 5.0
```

4) D'après le fichier de traces obtenues on peut rendre compte que les paquets de routage sont présents dès le début du fichier et même avant que le trafic cbr commence ce qui vérifie la faite qu'on utilise le protocole DSDV qui est un protocole de routage proactif donc il calcule les routes vers le nœud en avance.

5) Les 8 premiers lignes correspondent aux paquets de routage.

```

1 s 0.000821201 _0_ RTR --- 0 message 32 [0 0 0 0] ----- [0:255 -1:255 32 0]
2 s 0.001216201 _0_ MAC --- 0 message 90 [0 ffffffff 0 800] ----- [0:255 -1:255 32 0]
3 r 0.001937018 _1_ MAC --- 0 message 32 [0 ffffffff 0 800] ----- [0:255 -1:255 32 0]
4 r 0.001962018 _1_ RTR --- 0 message 32 [0 ffffffff 0 800] ----- [0:255 -1:255 32 0]
5 s 1.114891050 _1_ RTR --- 1 message 32 [0 0 0 0] ----- [1:255 -1:255 32 0]
6 s 1.115206050 _1_ MAC --- 1 message 90 [0 ffffffff 1 800] ----- [1:255 -1:255 32 0]
7 r 1.115926866 _0_ MAC --- 1 message 32 [0 ffffffff 1 800] ----- [1:255 -1:255 32 0]
8 r 1.115951866 _0_ RTR --- 1 message 32 [0 ffffffff 1 800] ----- [1:255 -1:255 32 0]
9 s 5.000000000 _0_ AGT --- 2 cbr 512 [0 0 0 0] ----- [0:0 1:0 32 0] [0] 0 0
10 r 5.000000000 _0_ RTR --- 2 cbr 512 [0 0 0 0] ----- [0:0 1:0 32 0] [0] 0 0
11 s 5.000000000 _0_ RTR --- 2 cbr 532 [0 0 0 0] ----- [0:0 1:0 32 1] [0] 0 0
12 s 5.000315000 _0_ MAC --- 0 ARP 86 [0 ffffffff 0 806] ----- [REQUEST 0/0 0/1]
13 r 5.001003817 _1_ MAC --- 0 ARP 28 [0 ffffffff 0 806] ----- [REQUEST 0/0 0/1]
14 s 5.001098817 _1_ MAC --- 0 ARP 86 [13a 0 1 806] ----- [REPLY 1/1 0/0]
15 r 5.001336724 _0_ MAC --- 0 ARP 28 [13a 0 1 806] ----- [REPLY 1/1 0/0]
16 s 5.001346724 _0_ MAC --- 0 ACK 38 [0 1 0 0]
17 r 5.001651541 _1_ MAC --- 0 ACK 38 [0 1 0 0]
18 s 5.001900724 _0_ MAC --- 2 cbr 590 [13a 1 0 800] ----- [0:0 1:0 32 1] [0] 0 0
19 r 5.002505177 _1_ MAC --- 2 cbr 532 [13a 1 0 800] ----- [0:0 1:0 32 1] [0] 1 0
20 s 5.002515177 _1_ MAC --- 0 ACK 38 [0 0 0 0]

```

Figure 13 : les 20 premières lignes du fichier de traces

6) Les 8 premiers lignes sont dédiées au routage proactif. On se rend compte que les paquets de routage créés à la couche RTR passe vers la couche MAC pour pouvoir les transmettre sur le support physique. Dans la ligne 9 on remarque le début de trafic cbr qui est à 5 secondes. Au niveau MAC, des requêtes ARP sont émises (ligne 12) dans le but d'avoir l'adresse physique du destinataire et on peut voir les réponses à ces requêtes (ligne 14) et les acquittements de ces réponses juste après. Ensuite on a le trafic cbr qui apparait dans les lignes suivantes.

7) Pour calculer le taux de perte moyen des paquets cbr on utilise les commandes appliquées dans la figure 14 pour compter le nombre de paquets émis et reçus et on applique la formule suivante :

Taux = Nombre de paquets émis – Nombre de paquets reçus / Nombre de paquets émis

```
[esirem@gr13-11l Bureau]$ grep ^s TP3.tr | grep cbr | grep "_0_" | grep "AGT" |
wc -l
13184
[esirem@gr13-11l Bureau]$ grep ^r TP3.tr | grep cbr | grep "_1_" | grep "AGT" |
wc -l
13184
```

Figure 14 : Nombre de paquets émis et reçu pour $t=5.0s$

Dans le cas où $t=5.0s$ le taux de perte est égale à 0 car le nombre de paquets émis est équivalent au nombre de paquets reçus. En fait, au niveau de la couche physique si une file d'attente n'a plus de place libre alors que la recherche de l'adresse MAC n'est pas encore finit les paquets qui vont arriver seront perdus.

8) On peut constater sur la figure 15 qu'il y a eu des pertes. Le taux de perte est donc de 2% qui explique que maintenant le protocole DSDV n'a plus le temps pour préparer les routes en avance avant que le trafic démarre et donc on a le remplissage de la capacité de mémoire et pour cela des paquets vont être perdus.

```
[esirem@gr13-11l Bureau]$ grep ^r TP3.tr | grep cbr | grep "_1_" | grep "AGT" |
wc -l
14133
[esirem@gr13-11l Bureau]$ grep ^s TP3.tr | grep cbr | grep "_0_" | grep "AGT" |
wc -l
14405
```

Figure 15 : Nombre de paquets émis et reçu pour $t=0s$

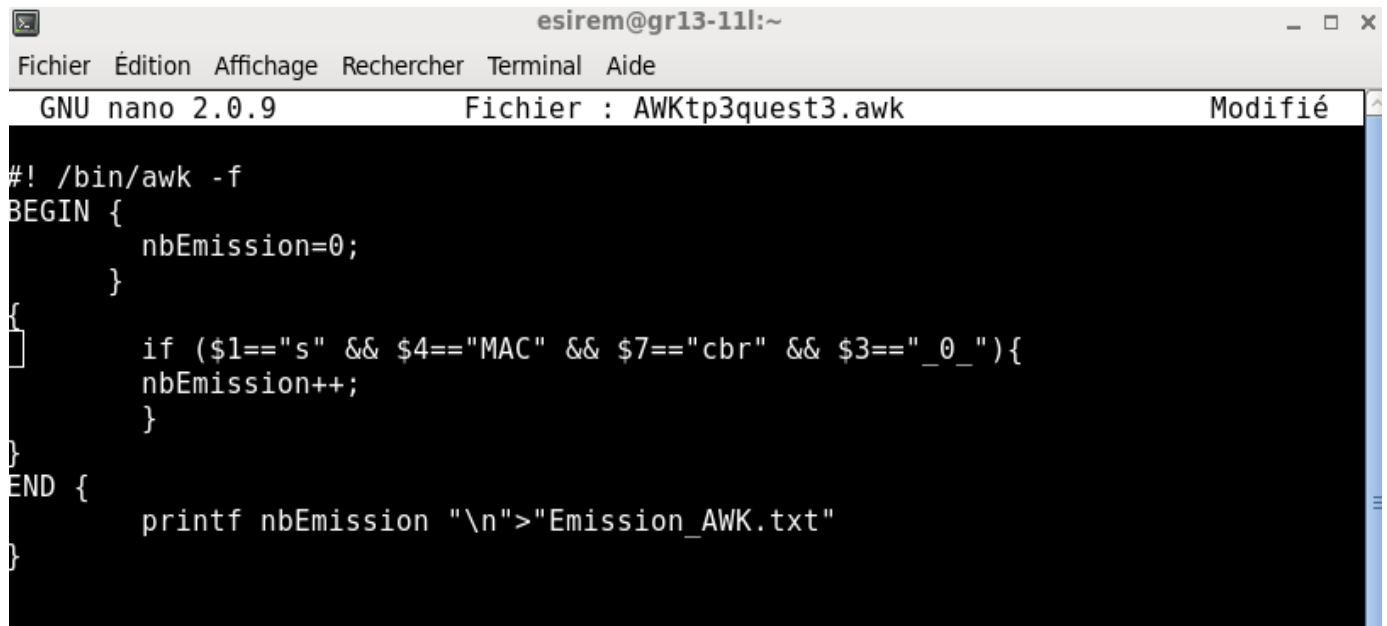
III. Routage multi-sauts:

1) Pour la génération des nœuds espacés de 250 mètres l'un de l'autre, on a créé une boucle for qui est présenté dans la figure 16 pour le cas de 4 nœuds dans le réseau.

```
for {set i 0} {$i < 4} {incr i} {
    set node_($i) [$ns node]
    $node_($i) set X_ [expr $i*250+500]
    $node_($i) set Y_ 500.0
    $node_($i) set Z_ 0.0
}
```

Figure 16 : Génération de 4 nœuds

3) Le script de la figure 17 nous permet de compter le nombre de trames émises et cela est fait en vérifiant que le premier élément de la ligne commence par la lettre 's' qui signifie que c'est un message émis et que le 7^{ème} élément est 'cbr' donc un trafic cbr et que le 3^{ème} élément est 0 donc c'est le nœud 0 qui envoie le message qui est de la couche 'MAC' qu'on peut le savoir grâce au 4^{ème} élément.

A screenshot of a terminal window running the GNU nano 2.0.9 text editor. The window title is 'esirem@gr13-11l:~'. The menu bar includes 'Fichier', 'Édition', 'Affichage', 'Rechercher', 'Terminal', and 'Aide'. The status bar shows 'GNU nano 2.0.9', 'Fichier : AWKtp3quest3.awk', and 'Modifié'. The editor contains an AWK script with the following content:

```
#!/bin/awk -f
BEGIN {
    nbEmission=0;
}
{
    if ($1=="s" && $4=="MAC" && $7=="cbr" && $3=="_0_"){
        nbEmission++;
    }
}
END {
    printf nbEmission "\n">"Emission_AWK.txt"
}
```

Figure 17 : Nombre de trames émises

4-5)

```

BEGIN {
    resultat = "results.txt"
    nbEmission = 0;
    #nbReception = `grep "^r" TP3.tr | grep "AGT" | grep cbr | grep "_14_" | wc -l`
    nbReception = 100

    for (i=0;i<nbReception;i++) {
        tabEmis[i] = -1
        tabRecus[i] = -1
    }
}

{
    if($4 == "AGT" && $7 == "cbr") {
        if($1=="s" && $3 == "_0_") {
            nbEmission++
            tabEmis[$6] = $2
        }
        else if($1=="r" && $3=="_14_") {
            tabRecus[$6] = $2
        }
    }
}

END {
    resultatTemps = 0
    var = 0

    for (i=0;i<nbReception;i++) {
        if(tabRecus[i] != -1) {
            resultatTemps += tabRecus[i]-tabEmis[i]
            var++
        }
    }

    resultatTemps = resultatTemps/var
    printf "Temps d'acheminement moyen:" ResultatTemps "\n"
    printf (15 " " nbEmission " " resultatTemps) >> resultat
}

```

Figure 18 : Script AWK calculant le temps d'acheminement moyen des paquets

Dans la partie BEGIN de ce script, on a initialisé plusieurs variables (nombre de paquets émis, reçus), un tableau stockant le temps d'émission et un autre pour le temps de réception et toutes les cases sont initialisées à -1 et on a utilisé la commande grep pour avoir le nombre de paquets de type 'AGT' reçus. Ensuite, un test est réalisé pour filtrer le trafic CBR au niveau de la couche AGT et en fonction de si la trame est émise du premier nœud ou reçue par le dernier, les bons compteurs sont incrémentés et les temps récupérés par le biais du champ. Dans le bloc END, on applique le calcul du temps d'acheminement moyen en ne prenant pas les cases ayant une valeur de -1 et on met les résultats dans le fichier 'results.txt' défini par la variable 'result'

TP 4: Routage ad hoc proactif

I. Introduction:

Ce TP va nous permettre d'exploiter les différentes phases de la découverte des routes dans les réseaux ad hoc et l'acheminement des paquets. Pour appliquer le travail on a utilisé le mode de propagation Two-ray ground qui est déterministe.

II. Protocole DSR :

1-2) Bloc d'instructions qui vérifie si le protocole de routage choisi est DSR. Le cas échéant, il changera la valeur courante de `val(ifq)` en `CMUPriQueue`.

```
set val(rp) DSR

if {$val(rp) == "DSR"} {

    set val(ifq) CMUPriQueue

}
```

4) Le trafic CBR commence à 5 secondes. En fait, après analysant le fichier de trace on peut rendre compte qu'on a un message applicatif à 5s qui est le 1^{er} message ce qui n'était pas le cas pour le protocole DSDV. Aucun mécanisme de routage n'est visible avant cela et c'est suite à ce message applicatif que la construction d'une route avec DSR à lieu. Donc on peut confirmer que le protocole DSR est un des protocoles réactifs.

5) On a un taux de perte qui est nul comme on peut le voir sur la figure 19 car le nombre de paquets émis (13184) est égale au nombre de paquets reçus.

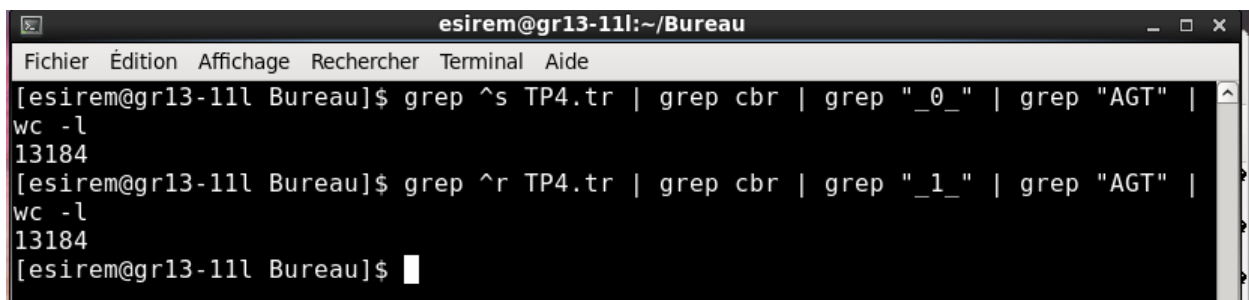
A screenshot of a terminal window titled 'esirem@gr13-11l:~/Bureau'. The terminal shows two commands being executed. The first command is 'grep ^s TP4.tr | grep cbr | grep "_0_" | grep "AGT" | wc -l', which returns '13184'. The second command is 'grep ^r TP4.tr | grep cbr | grep "_1_" | grep "AGT" | wc -l', which also returns '13184'. The terminal prompt is '[esirem@gr13-11l Bureau]\$'.

Figure 19 : Nombre de paquets émises et reçues

6) Quand on change le trafic cbr à 5.4 Mbit/s on aura les valeurs présentées dans la figure 20. On peut constater qu'on a un taux de perte égale à 40% à peu près et donc le débit dans ce cas sature les buffers avant que les routes soient calculées.

```
[esirem@gr13-11l Bureau]$ grep ^s TP4.tr | grep cbr | grep "_0_" | grep "AGT" | wc -l
71192
[esirem@gr13-11l Bureau]$ grep ^r TP4.tr | grep cbr | grep "_1_" | grep "AGT" | wc -l
42571
[esirem@gr13-11l Bureau]$
```

Figure 20 : Nombre de paquets émis et reçus pour 5.4Mbit/s

7) Le déplacement du nœud est fait avec la commande suivante :

```
$ns at 6.0 "$node_(1) setdest 1260 1500 36.11"
```

8) On retrouve dans le fichier de trace, l'information montrant cette mobilité qui est grisé dans la figure 21. 36.11 c'est la conversion de km/h en m/s.

```
10448 s 5.999857853 1 MAC --- 0 ACK 38 [0 0 0 0]
10449 r 5.999872853 1 RTR --- 1244 cbr 532 [13a 1 0 800] ----- [0:0 1:0 32 1] [1232] 1 0
10450 r 5.999872853 1 AGT --- 1244 cbr 512 [13a 1 0 800] ----- [0:0 1:0 32 1] [1232] 1 0
10451 M 6.00000 1 (1245.00, 1500.00, 0.00), (1260.00, 1500.00), 36.11
10452 r 6.000162670 0 MAC --- 0 ACK 38 [0 0 0 0]
10453 s 6.000352670 0 MAC --- 1246 cbr 590 [13a 1 0 800] ----- [0:0 1:0 32 1] [1234] 0 0
10454 s 6.000485926 0 AGT --- 1331 cbr 512 [0 0 0 0] ----- [0:0 1:0 32 0] [1319] 0 0
10455 r 6.000485926 0 RTR --- 1331 cbr 512 [0 0 0 0] ----- [0:0 1:0 32 0] [1319] 0 0
```

Figure 21 : Fichier de traces après déplacement d'un nœud

9-10) Le nœud 1 va continuer à recevoir des paquets à une petite distance jusqu'à qu'il soit hors de portée. À partir d'une certaine distance et pour un temps > 6.0, il ne recevra plus les paquets car il sera hors portée. Étant dans la zone de couverture il reçoit toujours des paquets.

III. Routage multi-sauts:

3) Script awk qui comptera le nombre de trames émises (niveau de log « MAC »)

```

esirem@gr13-11l:~
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
GNU nano 2.0.9      Fichier : AWKtp3quest3.awk      Modifié

#!/bin/awk -f
BEGIN {
    nbEmission=0;
}
{
    if ($1=="s" && $4=="MAC" && $7=="cbr" && $3=="_0_"){
        nbEmission++;
    }
}
END {
    printf nbEmission "\n">"Emission_AWK.txt"
}

```

Figure 22 : Nombre de trames émises

4-5) Script Awk adapté pour calculer également le temps d'acheminement moyen des paquets données (niveau de log « AGT »)

```

BEGIN {
    resultat = "results.txt"
    nbEmission = 0;
    #nbReception = `grep "^r" TP3.tr | grep "AGT" | grep cbr | grep "_14_" | wc -l`
    nbReception = 100

    for (i=0;i<nbReception;i++) {
        tabEmis[i] = -1
        tabRecus[i] = -1
    }
}

{
    if($4 == "AGT" && $7 == "cbr") {
        if($1=="s" && $3 == "_0_") {
            nbEmission++
            tabEmis[$6] = $2
        }
        else if($1=="r" && $3=="_14_") {
            tabRecus[$6] = $2
        }
    }
}

END {
    resultatTemps = 0
    var = 0

    for (i=0;i<nbReception;i++) {
        if(tabRecus[i] != -1) {
            resultatTemps += tabRecus[i]-tabEmis[i]
            var++
        }
    }

    resultatTemps = resultatTemps/var
    printf "Temps d'acheminement moyen:" ResultatTemps "\n"
    printf (15 " " nbEmission " " resultatTemps) >> resultat
}

```

Figure 23 : Script AWK calculant le temps d'acheminement moyen des paquets

TP 5: Routage AODV et trafic FTP

I. Introduction:

Dans ce TP, on va étudier le protocole AODV en implémentant un réseau composé de 16 nœuds placé sous forme de matrice et chaque nœud sera placé d'une distance de 240 mètres horizontalement et verticalement. de l'autre nœud.

II. Travail à faire

1) Commençons par la création des nœuds ayant une forme d'une grille 4x4

```
Set comp 0
for {set i 0} {$i < 4} {incr i} {
  for {set j 0} {$j < 4} {incr j} {
    set node_($comp) [$ns node]
    $node_($comp) set X_ [expr $j * 240]
    $node_($comp) set Y_ [expr $i * 240]
    $node_($comp) set Z_ 0.0
  }
  incr comp
}
```

2-3) Le trafic ftp va être démarré à la 10^{ème} seconde de la simulation. On observe dans le fichier de trace suivant que le premier message de routage intervient suite au premier message applicatif et donc on peut constater que le protocole AODV est un protocole réactif et donc le dernier ne fait pas le calcul de route en avance.

```
s 10.000000000 _0_ AGT --- 0 tcp 40 [0 0 0 0] ----- [0:0 15:0 32 0] [0 0] 0 0
r 10.000000000 _0_ RTR --- 0 tcp 40 [0 0 0 0] ----- [0:0 15:0 32 0] [0 0] 0 0
s 10.000000000 _0_ RTR --- 0 AODV 48 [0 0 0 0] ----- [0:255 -1:255 30 0] [0x2 1 1 [15 0] [0 4]] (REQUEST)
s 10.000115000 _0_ MAC --- 0 AODV 106 [0 ffffffff 0 800] ----- [0:255 -1:255 30 0] [0x2 1 1 [15 0] [0 4]] (REQUEST)
r 10.000963800 _4_ MAC --- 0 AODV 48 [0 ffffffff 0 800] ----- [0:255 -1:255 30 0] [0x2 1 1 [15 0] [0 4]] (REQUEST)
r 10.000963800 _1_ MAC --- 0 AODV 48 [0 ffffffff 0 800] ----- [0:255 -1:255 30 0] [0x2 1 1 [15 0] [0 4]] (REQUEST)
r 10.000988800 _4_ RTR --- 0 AODV 48 [0 ffffffff 0 800] ----- [0:255 -1:255 30 0] [0x2 1 1 [15 0] [0 4]] (REQUEST)
```

Figure 24 : Nœud origine de la requête de découverte de route

4) D'après la 3^{ème} ligne de figure 24 on remarque c'est le nœud 0 qui est à l'origine de la requête de découverte de route et c'est à la 10^{ème} seconde ce qui correspond bien au cas car c'est la source du trafic qui cherche à atteindre la destination du flux FTP.

5) Le nœud 15 est le nœud à l'origine de la réponse à la découverte de route. La ligne de la figure 25 montre la réponse de routage AODV.

```
s 10.020125441 _15_ RTR --- 0 AODV 44 [0 0 0 0] ----- [15:255 0:255 30 14] [0x4 1 [15 4] 10.000000] (REPLY)
```

Figure 25 : Noeud origine de la reponse de decouverte de route

6) Pour trouver les nœuds qui interviennent dans les opérations de routage on filtre les lignes constituant AODV et REPLY et on fait ça avec la commande grep.

```
grep AODV tp4.tr | grep REPLY | cut -g ' -f 3 | uniq | less
```

On obtient le résultat suivant :

15

14

6

5

4

0

On constate que la réponse à la demande de route est débutée du nœud 15 puis a traversé par les nœuds 14,6,5,4 et arriver au nœud 0 à la fin.

7) Les réponses aux messages ARP sont sujettes à l'envoi d'acquittement, contrairement aux messages de requêtes ARP. Cela s'explique car les requêtes ARP sont envoyées en broadcast, et les messages de broadcast ne sont pas acquittés. Cela générerait une grande quantité de messages si le nombre de nœuds était important, et dégraderait fortement le reste des communications. En revanche les réponses elles sont envoyées en unicast, type de message qui peut être acquitté sans causer de désagrément pour le reste du réseau.

III. Conclusion

Grâce à ce tp, on a pu voir le fonctionnement du routage AODV avec un trafic FTP. Bien que le temps ne nous a pas laissé la possibilité de calculer le taux de perte moyen du trfaic ftp.