
Rapport du projet du Réseaux mobiles : Simulation d'un réseau SANET via NS2

Filière : Sécurité des Systèmes d'Information

Préparé par : Ikram BOURHIM

Encadré par : Pr. Amine BERQIA

Année universitaire : 2021-2022

Table de matières

Table des figures	2
Introduction générale	3
1 Présentation du projet	4
1.1 Introduction aux réseaux Ad-Hoc	5
1.2 Les différentes phases basiques des algorithmes d'évitement de congestion	6
1.2.1 Slow Start :	6
1.2.2 Congestion Avoidance :	6
1.2.3 Fast Retransmit :	6
1.2.4 Fast Recovery	7
1.3 Les différentes variantes TCP : TCP Reno et TCP vegas :	7
1.3.1 Tcp reno :	7
1.3.2 TCP Vegas :	7
1.4 TCP RENO comparé à TCP VEGAS :	8
2 Présentation des outils et configuration :	10
2.1 Introduction	10
2.2 Langage TCL :	11
2.3 Network simulator NS2 :	11
2.4 L'installation de NS2 et nam dans un environnement Linux	11
3 Réalisation :	13
3.1 Simulation avec ns2 :	13
3.1.1 Architecture SANET adoptée :	13
3.1.2 Script NS Sanet :	13
3.2 L'exécution :	18
4 Conclusion générale :	20
5 WEBOGRAPHIE	21
6 Bibliographie :	22

Table des figures

1.1	Le modèle des réseaux mobiles sans infrastructure	5
1.2	Topologies les plus utilisées pour les réseaux ad hoc statiques : (a) Hexagon, (b) Carré et (c) triangle	6
1.3	Comportement de TCP Reno	7
1.4	Prévention de la congestion à TCP Vegas	8
2.1	Installation de ns2 sur Ubuntu	11
2.2	Installation de nam sur Ubuntu	12
3.1	Excécution du script SANETVegas.tcl	18
3.2	Visualisation du résultat pour TCPVegas sur nam	18
3.3	Visualisation du résultat pour TCP Reno sur nam	18
3.4	Résultat pour TCP Reno	19
3.5	Résultat pour TCP Vegas	19

Introduction générale

Un réseau sans fil permet aux appareils de rester connectés au réseau sans câbles encombrants. Les points d'accès amplifient les signaux Wi-Fi, de sorte qu'un appareil peut être toujours connecté au réseau bien qu'il soit éloigné d'un routeur.

Le Transmission Control Protocol (TCP) a été conçu à l'origine pour fonctionner sur des réseaux câblés. Cependant, de nos jours, le trafic sur les réseaux sans fil a augmenté à un point tel qu'il faut prendre en compte les caractéristiques spécifiques de ces réseaux lors de la mise en place d'une implémentation TCP particulière sur eux. Afin de comprendre les principales différences entre *TCP Reno* et *TCP Vegas* dans un environnement **SANET**, nous avons simulé plusieurs types de réseaux sans fils ad-hoc afin d'évaluer les performances et le taux de perte dû à la perte de paquets causée par la congestion et comment les deux des algorithmes gèrent la congestion dans le réseau. En utilisant le logiciel de simulation **NS2** (Network Simulator 2) qui est le plus utilisé vu qu'il supporte partiellement les protocoles TCP, de routage et multicast sur réseau filaire ou sans fil, et dont les simulations se programment en **TCL/TK**, nous avons évalué par la simulation les principes de fonctionnement. L'objectif de ce rapport est de présenter notre démarche, les scripts que nous avons mis en place pour évaluer les différents algorithmes dans diverses situations, et les analyses que nous avons tirées de ces expériences. Dans un premier temps, ce rapport contient une brève description des outils utilisés, des réseaux ad hoc, du mode opératoire suivi, ainsi que des variantes du protocoles de routage étudié TCP à savoir *TCP Reno* et *TCP Vegas*. Dans un second temps, il présente les expérimentations liées au fonctionnement statique pour un positionnement aléatoire des nœuds. Enfin, il présente les expérimentations liées au fonctionnement avec des nœuds mobiles.

Chapitre 1

Présentation du projet

Dans ce chapitre nous allons détailler les concepts des réseaux mobiles et l'architecture qui nous interesse qui est l'Ad-hoc, les variantes du protocole TCP étudiées et nous allons mettre la lumière sur toutes les étapes de l'installation et la configuration des outils nécessaires à la simulation.

1.1 Introduction aux réseaux Ad-Hoc

Les réseaux mobiles ad hoc est un réseau local sans fil, sans infrastructure et avec un contrôle réparti. Il peut être défini comme étant un ensemble des nœuds, pouvant être mobiles, interconnectés et communicants entre eux sans l'aide de support fixes et d'une administration centralisée. Le déploiement d'un réseau ad hoc est simple et ne nécessite aucun prérequis ; il suffit de disposer des terminaux dans une zone pour créer un réseau ad hoc. Chaque nœud dans le réseau se comporte comme un routeur et transmet les paquets pour d'autres nœuds. Le chemin emprunté d'un nœud source à un nœud destination est déterminé par un protocole de routage.

Les réseaux sans fil sans infrastructure :

Ces réseaux se constituent des unités mobiles communiquant entre eux sans l'aide d'une infrastructure fixe. Appelés communément ad hoc, Ils ne nécessitent aucune structure physique pour être déployés et sont opérationnels instantanément. Dans ce type de réseaux, tous les hôtes doivent coopérer pour gérer les communications entre eux (routage, contrôle de l'accès aux médias, ...).

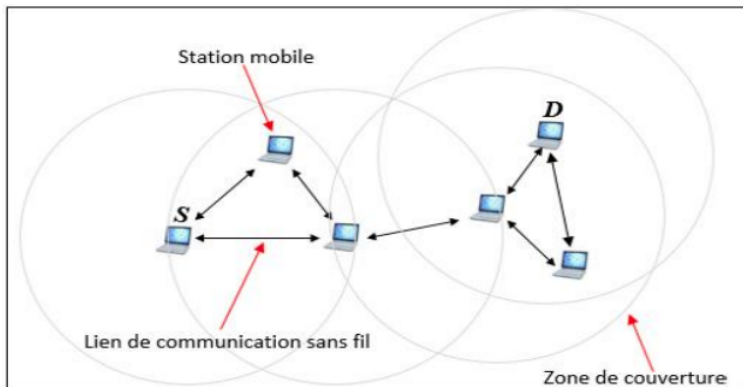


FIGURE 1.1 – Le modèle des réseaux mobiles sans infrastructure

Les réseaux ad hoc sans fil sont divisés en deux types : réseaux ad hoc mobiles (MANETs) et réseaux ad hoc statiques (SANETs). D'un côté, les nœuds dans MANETs sont capables de se déplacer dans une région donnée, ce qui pose des difficultés supplémentaires à surmonter pour le bon fonctionnement du réseau. Alors que dans un SANET les Nœuds sans fil sont fixés en position et ne se déplacent pas.

Le SANET est un mode Point à Point, ne nécessitant pas de points d'accès. Il permet de connecter les stations quand aucun point d'accès n'est disponible. L'absence d'infrastructure oblige les UM à jouer le rôle de routeurs

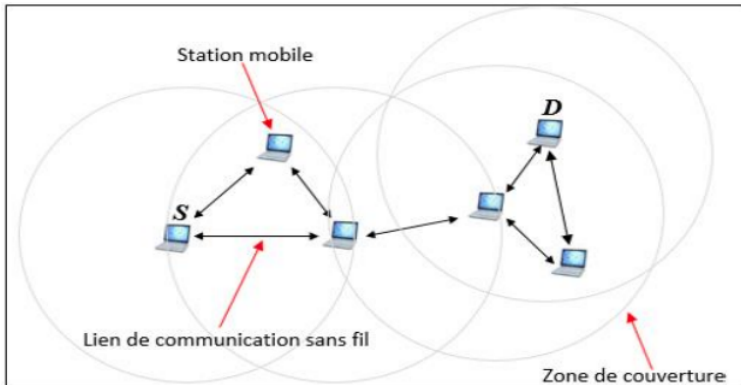


FIGURE 1.2 – Topologies les plus utilisées pour les réseaux ad hoc statiques : (a) Hexagon, (b) Carré et (c) triangle

1.2 Les différentes phases basiques des algorithmes d'évitement de congestion

Comme définis dans la RFC5681 ces phases visent à gérer la congestion dans TCP.

1.2.1 Slow Start :

Défini par le RFC 5681 et son principe c'est d'augmenter le nombre de paquets envoyés après le démarrage d'une connexion. Le but est donc de trouver la bande passante disponible, et utiliser toutes les ressources à disposition. On va alors utiliser une fenêtre d'émission (swnd), qui représente un nombre de paquets à envoyer sans attendre d'acquittement. Cette fenêtre grandira au fur et à mesure que nous recevons des acquittements pour les paquets envoyés (chaque ACK fera augmenter cette fenêtre de 1, autrement dit, après chaque RTT la taille de la fenêtre doublera, tant qu'il n'y a pas de congestion).

1.2.2 Congestion Avoidance :

Au-delà d'une certaine limite de valeur de cwnd (slow start threshold, ssthresh), TCP passe en mode d'évitement de congestion. À partir de là, la valeur de cwnd augmente de façon linéaire et donc bien plus lentement qu'en slow start : cwnd s'incrémente de un MSS (= un paquet) à chaque RTT. Dans ce mode de fonctionnement, l'algorithme détecte aussi rapidement que possible la perte d'un paquet : si nous recevons trois fois le ACK du même paquet, on n'attend pas la fin d'un timeout pour réagir. En réaction à cette perte, on fait descendre la valeur de ssthresh ainsi que cwnd (on repasse éventuellement en mode de Slow Start). On utilise la technique de Fast Retransmit pour renvoyer rapidement les paquets perdus.

1.2.3 Fast Retransmit :

c'est une amélioration de TCP qui réduit le temps qu'un expéditeur attend avant de retransmettre un segment perdu. Un expéditeur TCP utilise normalement une simple minuterie pour reconnaître les segments perdus. Si un accusé de réception n'est pas reçu pour un segment particulier dans un délai spécifié (une fonction du délai d'aller-retour estimé), l'expéditeur supposera que le segment a été perdu dans le réseau et retransmettra le segment.

1.2.4 Fast Recovery

Plutôt que repasser en mode Slow Start lors d'une duplication de ACK (et après un passage par le mode Fast Retransmit), nous renvoyons le segment perdu et on attend un ACK pour toute la fenêtre transmise précédemment avant de retourner en mode Congestion Avoidance. Si on atteint le timeout, on repart en mode Slow Start. Grâce à cette technique nous évitons de baisser le débit d'une façon trop brutale.

1.3 Les différentes variantes TCP : TCP Reno et TCP vegas :

Depuis 2002, plus de 10 variantes de TCP proposées

1.3.1 Tcp reno :

Tcp reno est la variante la plus populaire de tcp et il y inclut le mécanisme *Fast Recovery*. Le nouvel algorithme permet d'éviter que le canal de communication ne soit vide après un *Fast Retransmit* évitant ainsi le besoin de démarrer le Slow-start pour le remplir de nouveau après la perte d'un seul paquet. La façon d'estimer la perte de paquets : TCP Reno peut différencier entre les deux cas suivants : (i) perte de paquet aperçue par le RTO (le réseau subit une congestion sévère) et (ii) perte de paquet aperçue par des acquittements dupliqués (la congestion dans le réseau n'est pas sévère). Dans le premier cas, TCP Reno diminue son débit de transmission à une valeur minimale et entre dans la phase slow-start. Dans le second cas, TCP Reno diminue son débit de transmission de moitié sans repasser par la phase slow rate.

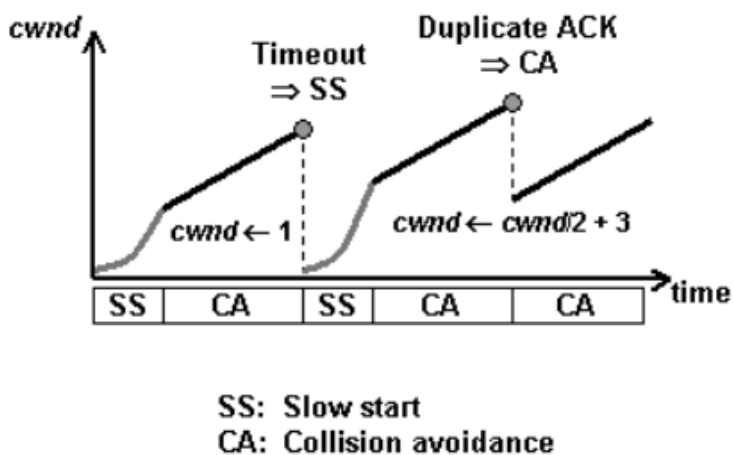


FIGURE 1.3 – Comportement de TCP Reno

1.3.2 TCP Vegas :

C'est une extension de TCP Reno. Dans TCP Vegas, l'émetteur enregistre les temps auxquels un segment est émis ainsi que les temps auxquels on reçoit l'accusé de réception correspondant. Quand un acquittement arrive à l'émetteur, TCP Vegas calcule le RTT

en se basant sur l'horloge du système. Il utilise ensuite cette valeur pour décider s'il doit retransmettre le dernier segment émis ou non. Par exemple, lorsque le RTT estimé est plus grand que la valeur du RTO, il retransmet le paquet suivant directement sans avoir à attendre les 3 acquittements dupliqués. Autrement dit, TCP Vegas traite la réception d'un acquittement comme un indice pour voir s'il est nécessaire de déclencher le RTO pour les paquets suivants ou non. Ainsi, TCP Vegas détecte la perte de paquets de manière plus rapide que les autres variantes. TCP Vegas utilise des augmentations additives dans la fenêtre de congestion. L'objectif mis en évidence dans l'algorithme de Vegas, est d'essayer d'obtenir un taux de transmission plus élevé avec moins de retransmissions.

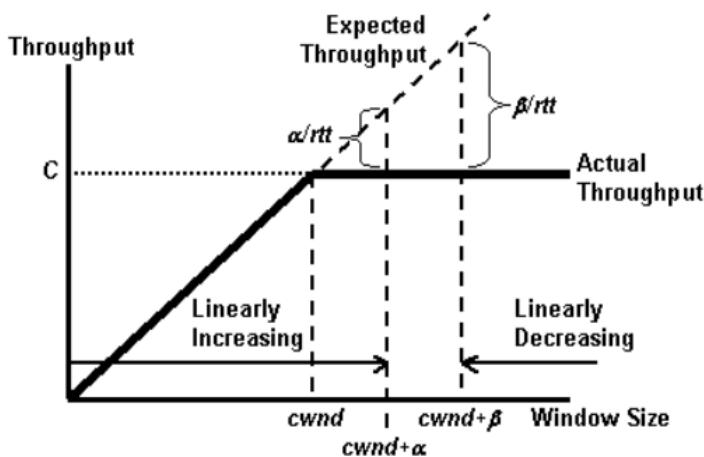


FIGURE 1.4 – Prévention de la congestion à TCP Vegas

1.4 TCP RENO comparé à TCP VEGAS :

Dans **TCP Reno** la congestion n'est pas détectée tant qu'il n'y a pas de perte, alors que **TCP Vegas** propose un algorithme de prévision de congestion du réseau, il détecte la congestion avant que les paquets soient perdus. **TCP Vegas** contrôle la taille de la fenêtre en observant le RTT des paquets que l'hôte expéditeur a envoyés.

Si le *RTT* observée devient grand, **TCP Vegas** reconnaît que le réseau commence à s'encombrer.

Si le *RTT* diminue, **TCP Vegas** comprend que le réseau est libéré de la congestion et augmente la taille de la fenêtre.

Tcp Vegas possède le meilleur comportement énergétique par bit reçu par rapport aux autres variantes de TCP. Cependant, lorsque les taux d'erreur sont importants dans le réseau (10% et 15%), TCP Vegas possède les performances les plus mauvaises parmi toutes les variantes de TCP. Ceci est dû au fait que **TCP Vegas** vérifie l'évolution du timeout après chaque acquittement reçu et décide de retransmettre des paquets qu'il va considérer comme perdus. Ceci permet un dimensionnement adéquat de la valeur des timeout en comparaison avec l'utilisation pur et simple de la valeur RTO dans les autres variantes de TCP.

TCP Reno ne permettait pas d'évaluation précise des tailles des files sur le chemin de transmission car il se base sur des calculs de *RTT* cadencés par une horloge peu fine, fournissant des impulsions par multiples de 500 ms. Dans le cas de Vegas, l'algorithme enregistre les temps auxquels un segment est émis et pour lequel l'accusé de réception correspondant est reçu.

Le fonctionnement de **TCP Vegas** donne une méthode de perception plus rapide des paquets

perdus. Il ne donne pas lieu comme Reno à plusieurs réductions de taille de fenêtre pour des pertes occasionnées dans une même fenêtre de transmission. Dans Vegas, à l'inverse de Reno, l'algorithme ne donne pas lieu à une diminution de taille de fenêtre pour une perte remarquée après cette diminution, mais pour un paquet envoyé avant cette diminution de taille de fenêtre. En effet, il n'est pas sûr que pour la dernière taille de fenêtre obtenue après réduction, il y ait encore congestion.

Conclusion :

On a pu introduire, à travers ce chapitre, les différents concepts des réseaux mobile et la nécessité de simulation pour la comparaison des performances des différents algorithmes. Dans le chapitre suivant on va décrire les étapes de la réalisation du projet.

Chapitre 2

Présentation des outils et configuration :

2.1 Introduction

Ce chapitre est dédié a la présentation des diffèrent outils utilisés pour réaliser la simulation à savoir NS2, Nam et le langage TCL ainsi que la configuration de ces outils sur notre environnement Linux.

2.2 Langage TCL :

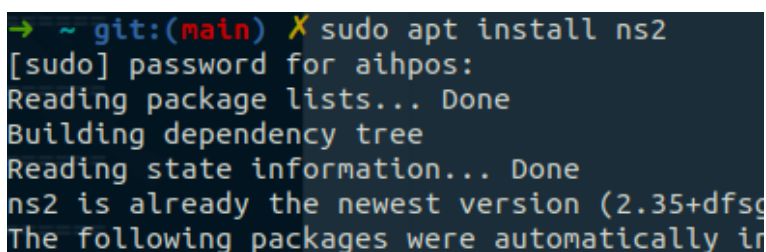
Tcl est un langage de programmation multi-paradigm à usage général. Il s'agit d'un langage de script qui vise à permettre aux applications de communiquer entre elles. D'autre part, Tk est une boîte à outils de widget multi-plateforme utilisée pour construire des GUI dans de nombreuses langues. La principale différence entre TCL et des langages comme le C est que TCL est un langage interprété et non un langage compilé. Les programmes écrits en TCL sont en fait des fichiers texte constitués de commandes TCL qui sont traitées par un interpréteur TCL au moment de l'exécution. Un avantage que cela offre est qu'un programme TCL peut générer lui-même des fichiers de commandes TCL qui peuvent être évaluées à un autre moment. Cela peut être utile, par exemple, lors de la création d'une interface graphique avec un bouton qui accomplit différentes actions selon le moment.

2.3 Network simulator NS2 :

L'outil utilisé pour de nombreuses simulations de réseau au niveau de la recherche est ns, pour simulateur de réseau et développé à l'origine à l'Institut des sciences de l'information. Pour créer une simulation ns-2, nous devons faire ce qui suit :

- définir la topologie du réseau, y compris tous les nœuds, les liens et les règles de mise en file d'attente des routeurs
- créer des connexions TCP (ou UDP), appelées Agents, et les attacher aux nœuds
- créer certaines applications – généralement FTP pour le transfert en bloc ou telnet pour la génération aléatoire intermittente de paquets – et les attacher aux agents -commencer la simulation

2.4 L'installation de NS2 et nam dans un environnement Linux



```
→ ~ git:(main) X sudo apt install ns2
[sudo] password for aihpos:
Reading package lists... Done
Building dependency tree
Reading state information... Done
ns2 is already the newest version (2.35+dfsg
The following packages were automatically in
```

FIGURE 2.1 – Installation de ns2 sur Ubuntu

```
➔ ~ git:(main) ✗ sudo apt-get install nam
[sudo] password for alhpos:
Reading package lists... Done
Building dependency tree
Reading state information... Done
nam is already the newest version (1.15-5build1).
The following packages were automatically installed and are no longer required:
  libstdc++2.0-0 linux-image-5.8.0-43-generic linux-modules-5.8.0-43-generic
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 69 not upgraded.
➔ ~ git:(main) ✗ nam
A nam instance already exists. Use nam <trace file> to view an animation
➔ ~ git:(main) ✗ nam
Cannot connect to existing nam instance. Starting a new one...
]
```

FIGURE 2.2 – Installation de nam sur Ubuntu

Chapitre 3

Réalisation :

3.1 Simulation avec ns2 :

3.1.1 Architecture SANET adoptée :

Réalisation d'un architecture SANET composée de :

- 2 Sources (0 - 1).
- 3 Gateways.
- 2 Destinations (5 — 6).
- 6 Cellules de portee max 250.
- Applications : FTP1 et FTP2 entre les sources et destinations

3.1.2 Script NS Sanet :

A partir d'un code modèle appelé wireless-mitf.tcl fourni dans le package ns all in one, nous avons pu réaliser le code suivant :

SANETReno.tcl :

```
1 #Copyright (c) 1997 Regents of the University of California.
2 # All rights reserved.
3 # Redistribution and use in source and binary forms, with or without
4 # modification, are permitted provided that the following conditions are met:
5 # 1. Redistributions of source code must retain the above copyright
6 # notice, this list of conditions and the following disclaimer.
7 # 2. Redistributions in binary form must reproduce the above copyright
8 # notice, this list of conditions and the following disclaimer in the
9 # documentation and/or other materials provided with the distribution.
10 # 3. All advertising materials mentioning features or use of this software
11 # must display the following acknowledgement:
12 # This product includes software developed by the Computer Systems
13 # Engineering Group at Lawrence Berkeley Laboratory.
14 # 4. Neither the name of the University nor of the Laboratory may be used
15 # to endorse or promote products derived from this software without
16 # specific prior written permission.
17 #
```

```

18 # THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ‘‘AS IS’’ AND
19 # ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
    WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
    DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY
    DIRECT INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (
    INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
    LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
    ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (
    INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
    SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE
20 # Rapport de Projet Réseau Mobile | Année universitaire 2021/2022
21
22 # $Header: /cvsroot/nsnam/ns-2/tcl/ex/wireless-mitf.tcl,v 1.2 2000/08/30
    00:10:45 haoboy Exp $
23 # Simple demo script for the new APIs to support multi-interface for
24 # wireless node.
25 # Define options
26 # Please note:
27 # 1. you can still specify "channelType" in node-config right now:
28 # set val(chan) Channel/WirelessChannel
29 # $ns_ node-config ...
30 # -channelType $val(chan)
31 # ...
32 # But we recommend you to use node-config in the way shown in this script
33 # for your future simulations.
34 #
35 # 2. Because the ad-hoc routing agents do not support multiple interfaces
36 # currently, this script can't generate anything interesting if you config
37 # the interfaces of node 1 and 2 on different channels
38 # —Xuan Chen, USC/ISI, July 21, 2000
39
40 set val(chan) Channel/WirelessChannel ; #Channel Type
41 set val(prop) Propagation/TwoRayGround ; # radio-propagation model
42 set val(netif) Phy/WirelessPhy ; # network interface type
43 set val(mac) Mac/802_11 ; # MAC type
44 set val(ifq) Queue/DropTail/PriQueue ; # interface queue type
45 set val(ll) LL ; # link layer type
46 set val(ant) Antenna/OmniAntenna ; # antenna model
47 set val(ifqlen) 500 ; # max packet in ifq
48 set val(nm) 7 ; # number of mobilenodes
49 set val(rp) DSDV ; # routing protocol
50 set val(x) 1000
51 set val(y) 1000
52
53 #initialisation des variables globale
54 set ns_ [new Simulator]
55 set tracefd [open SANET.tr w]
56 $ns_ trace-all $tracefd
57
58
59 set namtrace [open SANET.nam w]
60 $ns_ namtrace-all-wireless $namtrace $val(x) $val(y)
61
62
63 set topo [new Topography]
64
65
66 $topo load_flatgrid $val(x) $val(y)

```

```

67
68
69 create-god $val(nn)
70
71 set chan_1_ [new $val(chan)]
72 set chan_2_ [new $val(chan)]
73 set chan_3_ [new $val(chan)]
74 set chan_4_ [new $val(chan)]
75 set chan_5_ [new $val(chan)]
76 set chan_6_ [new $val(chan)]
77 set chan_7_ [new $val(chan)]
78
79 $ns_ node-config -adhocRouting $val(rp) \
80   -llType $val(ll) \
81   -macType $val(mac) \
82   -ifqType $val(ifq) \
83   -ifqLen $val(ifqlen) \
84   -antType $val(ant) \
85   -propType $val(prop) \
86   -phyType $val(netif) \
87   -topoInstance $topo \
88   -agentTrace ON \
89   -routerTrace ON \
90   -macTrace ON \
91   -movementTrace OFF \
92   -channel $chan_1_
93
94 set node_(0) [$ns_ node]
95 set node_(1) [$ns_ node]
96 set node_(2) [$ns_ node]
97 set node_(3) [$ns_ node]
98 set node_(4) [$ns_ node]
99 set node_(5) [$ns_ node]
100 set node_(6) [$ns_ node]
101
102 $node_(6) label 'serveur1'
103 $node_(5) label 'serveur2'
104
105 $node_(0) random-motion 0
106 $node_(1) random-motion 0
107 $node_(2) random-motion 0
108 $node_(3) random-motion 0
109 $node_(4) random-motion 0
110 $node_(5) random-motion 0
111 $node_(6) random-motion 0
112
113 for {set i 0} {$i < $val(nn)} {incr i} {
114     $ns_ initial_node_pos $node_($i) 20
115 }
116
117 $node_(0) set X_ 200.0
118 $node_(0) set Y_ 400.0
119 $node_(0) set Z_ 0.0
120
121 $node_(1) set X_ 350.0
122 $node_(1) set Y_ 400.0
123 $node_(1) set Z_ 0.0
124

```



```

125 $node_(2) set X_ 500.0
126 $node_(2) set Y_ 400.0
127 $node_(2) set Z_ 0.0
128
129 $node_(3) set X_ 650.0
130 $node_(3) set Y_ 600.0
131 $node_(3) set Z_ 0.0
132
133 $node_(4) set X_ 650.0
134 $node_(4) set Y_ 200.0
135 $node_(4) set Z_ 0.0
136
137 $node_(5) set X_ 50.0
138 $node_(5) set Y_ 600.0
139 $node_(5) set Z_ 0.0
140
141 $node_(6) set X_ 50.0
142 $node_(6) set Y_ 200.0
143 $node_(6) set Z_ 0.0
144
145 $ns_ at 0.0 "$node_(0) setdest 200.0 500.0 0.0"
146 $ns_ at 0.0 "$node_(1) setdest 400.0 500.0 0.0"
147 $ns_ at 0.0 "$node_(2) setdest 600.0 500.0 0.0"
148 $ns_ at 0.0 "$node_(3) setdest 700.0 600.0 0.0"
149 $ns_ at 0.0 "$node_(4) setdest 700.0 400.0 0.0"
150 $ns_ at 0.0 "$node_(5) setdest 10.0 600.0 0.0"
151 $ns_ at 0.0 "$node_(6) setdest 10.0 400.0 0.0"
152
153 set tcp1 [new Agent/TCP/Reno]
154 $tcp1 set class_ 2
155
156 set sink1 [new Agent/TCPSink]
157 $ns_ attach-agent $node_(0) $tcp1
158 $ns_ attach-agent $node_(5) $sink1
159 $ns_ connect $tcp1 $sink1
160 set ftp1 [new Application/FTP]
161 $ftp1 attach-agent $tcp1
162 $ns_ at 1.0 "$ftp1 start"
163 set tcp2 [new Agent/TCP/Reno]
164 $tcp2 set class_ 1
165
166 set sink2 [new Agent/TCPSink]
167 $ns_ attach-agent $node_(1) $tcp2
168 $ns_ attach-agent $node_(6) $sink2
169 $ns_ connect $tcp2 $sink2
170 set ftp2 [new Application/FTP]
171 $ftp2 attach-agent $tcp2
172 $ns_ at 1.0 "$ftp2 start"
173
174 for {set i 0} {$i < $val(nn)} {incr i} {
175     $ns_ at 30.0 "$node_($i) reset";
176 }
177
178 $ns_ at 100.0 "stop"
179 $ns_ at 100.01 "puts \"NS EXITING...\" ; $ns_ halt"
180
181 proc stop {} {
182     global ns_ tracefd

```

```

183 $ns_ flush-trace
184 close $tracefd
185 }
186 puts "Starting Simulation..."
187 $ns_ run

```

SANETVegas.tcl :

Il suffit de changer Reno par Vegas dans le script :

```

1
2 set tcp1 [new Agent/TCP/Vegas]
3 $tcp1 set class_ 2
4
5 set sink1 [new Agent/TCPSink]
6 $ns_ attach-agent $node_(0) $tcp1
7 $ns_ attach-agent $node_(5) $sink1
8 $ns_ connect $tcp1 $sink1
9 set ftp1 [new Application/FTP]
10 $ftp1 attach-agent $tcp1
11 $ns_ at 1.0 "$ftp1 start"
12 set tcp2 [new Agent/TCP/Vegas]
13 $tcp2 set class_ 1
14
15 set sink2 [new Agent/TCPSink]
16 $ns_ attach-agent $node_(1) $tcp2
17 $ns_ attach-agent $node_(6) $sink2
18 $ns_ connect $tcp2 $sink2
19 set ftp2 [new Application/FTP]
20 $ftp2 attach-agent $tcp2
21 $ns_ at 1.0 "$ftp2 start"
22
23 for {set i 0} {$i < $val(nn)} {incr i} {
24     $ns_ at 30.0 "$node_($i) reset";
25 }
26
27 $ns_ at 100.0 "stop"
28 $ns_ at 100.01 "puts \"NS EXITING...\" ; $ns_ halt"
29
30 proc stop {} {
31     global ns_ tracefd
32     $ns_ flush-trace
33     close $tracefd
34 }
35 puts "Starting Simulation..."
36 $ns_ run

```

La simulation réseau est définie dans un fichier Tcl, nommée SANET.tcl; pour exécuter la simulation, on exécute ensuite la commande

```
1 ns SANET.TCL
```

Le résultat de l'exécution de la simulation ns-2 sera de créer certains fichiers, et peut-être d'imprimer certains résultats. Les fichiers les plus courants créés sont le fichier de trace ns-2 – peut-être SANET.tr – qui contient un enregistrement pour chaque arrivée de paquet, départ et événement de file d'attente, et un fichier SANET.nam pour l'animateur réseau, nam, qui permet l'affichage visuel des paquets en mouvement.

3.2 L'exécution :

```

→ Desktop git:(main) x ns SANETVegas.tcl
num_nodes is set 7
INITIALIZE THE LIST xListHead
Starting Simulation...
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5, distCST_ = 550.0
SORTING LISTS ...DONE!
NS EXITING...

```

FIGURE 3.1 – Exécution du script SANETVegas.tcl

```

→ Desktop git:(main) x awk -f stats.awk SANETVegas.tr
Paquets envoyés: 12681
Paquets reçus: 12672
Taux de pertes= 100,07 %
→ Desktop git:(main) x

```

FIGURE 3.2 – Visualisation du résultat pour TCPVegas sur nam

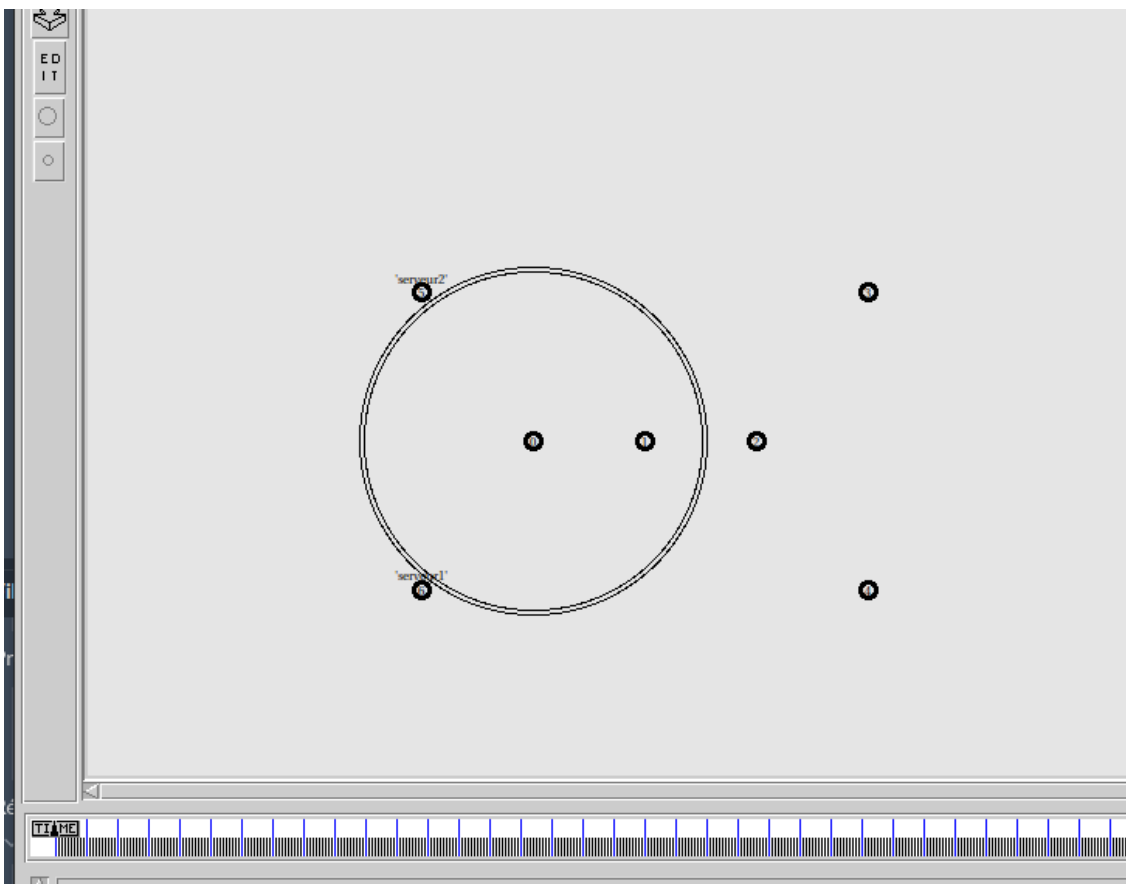


FIGURE 3.3 – Visualisation du résultat pour TCP Reno sur nam

Statistiques pour TCP Reno : Le script Awk stats.awk pour calculer le taux de pertes :

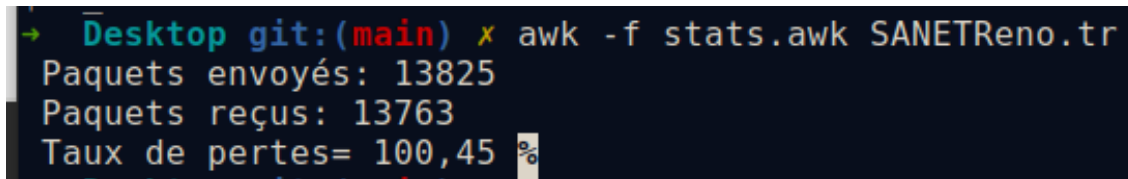
```

1 BEGIN {sent=0;
2 received=0;
3 }
4 {
5     if($1=="s" && $4=="AGT")
6     {sent++;}
7     else if($1=="r" && $4=="AGT")
8     {received++;}
9     }
10 }
11 END{
12
13     printf " Paquets envoy s: %d", sent;
14     printf " \n Paquets re us: %d" , received;
15     printf " \n Taux de pertes= %.2f " , (sent/received)*100;
16 }

```

On peut excécuter la commande suivante pour le fichier trace dans le cas de TCP Reno pour avoir le taux de pertes :

```
1 awk -f stats.awk SANETReno.tr
```



```

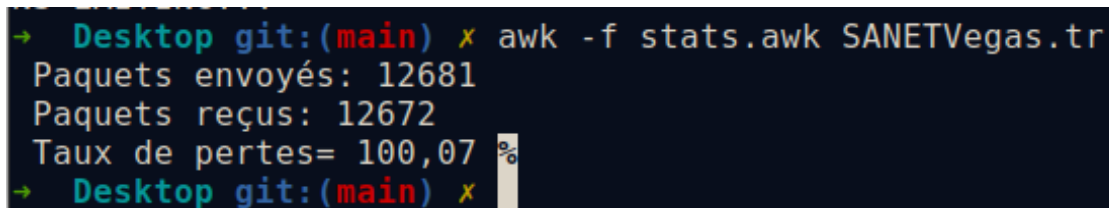
→ Desktop git:(main) x awk -f stats.awk SANETReno.tr
Paquets envoyés: 13825
Paquets reçus: 13763
Taux de pertes= 100,45 %

```

FIGURE 3.4 – Résultat pour TCP Reno

Pour TCP Vegas :

```
1 awk -f stats.awk SANETVegas.tr
```



```

→ Desktop git:(main) x awk -f stats.awk SANETVegas.tr
Paquets envoyés: 12681
Paquets reçus: 12672
Taux de pertes= 100,07 %
→ Desktop git:(main) x

```

FIGURE 3.5 – Résultat pour TCP Vegas

Chapitre 4

Conclusion générale :

Dans ce rapport nous avons étudié l'architecture AD HOC en particulier celle du SANET a l'aide d'un simulateur ns2 et obtenir des résultats sur le taux de perte de paquets entre TCP Reno et Vegas.

Chapitre 5

WEBOGRAPHIE

<https://www.yumpu.com/en/document/read/36268174/a-performance-study-of-tcp-tahoe>
http://www.senouci.net/download/Publications/Others/Senouci_DNAC2004.pdf
TCP Congestion Control : https://en.wikipedia.org/wiki/TCP_congestion_control
<https://technobytz.com/how-to-install-network-simulator-ns2-nam-in-ubuntu-14-04.html>
<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.7433&rep=rep1&type=pdf>
TCP Congestion Control in RFC : <https://datatracker.ietf.org/doc/html/rfc5681>

Chapitre 6

Bibliographie :

Cours des réseaux mobiles : <https://moodle-ensias.um5.ac.ma/course/view.php?id=10128>