Modelarea retelei folosind simulatorul NS2

1 Prezentare generala

1.1 Obiectiv

Primul obiectiv al activităţii practice este de a prezenta software-ul de simulare de reţea NS-2 şi NAM, care permite simularea unor reţele complexe şi vizualizarea modului în care acestea funcţionează.

Obiectivul de bază este de a simula diverse situaţii întâlnite într-o reţea, cum ar fi congestia, tăierea unui inel de legătură din reţea, sau simularea M/M/1 şi M / M / 1 / 2.

1.2 Limbajul TCL

TCL este un limbaj de scripting puternic, care poate utiliza opţional o abordare de programare orientată pe obiect. Este uşor expandabil de o serie de module.

În cazul nostru, este esenţial de a folosi limbajul TCL pentru lucrul cu obiecte oferite de către NS-2.

1.3 Simulatorul NS-2

Instrumentul NS-2 oferă un set de obiecte TCL special adaptate pentru simularea reţelelor. Cu obiectele propuse de motorul de simulare, putem reprezenta reţele cu legături cu fir sau fără fir, maşini şi routere (Nod), fluxuri TCP şi UDP (de exemplu, pentru a simula un flux CBR), şi selecta politicile şi normele cozilor puse în aplicare în fiecare nod.

În toate exemplele, avem mereu simulat un flux constant UDP, deoarece permite studiul fenomenelor de respingere de pachete fără a impovăra reţeaua.

NS-2 nu arată rezultatele testelor. Acesta poate stoca doar o urmă de simulare, care poate fi exploatată de către alte software-uri, cum ar fi NAM.

1.4 Instrumentul de vizualizare NAM

NAM este un instrument de vizualizare care prezintă două interese principale : să reprezinte topologia unei reţele descrisă cu NS-2, şi de a afişa temporal rezultatele unei execuţii NS-2. De exemplu, este capabil să reprezinte TCP sau UDP, pierderea unei legături între noduri, sau să reprezinte pachetele rejectate de o coadă plină.

Acest software este adesea numit direct din scriptul TCL pentru NS-2, astfel încât să se vizualizeze direct rezultatul simulării.

2 TCL

Pentru a intelege limbajul TCL, necesar proiectării unor simulări de reţea cu NS-2, am realizat o serie de sesiuni de formare : înţelegerea unui script TCL, traducerea unui program C în TCL, şi realizarea unei primi cereri pentru NS–2 şi NAM.

2.1 Un exemplu de script TCL comentat

Pentru a face un contact efectiv limbajul scripting TCL, care permite manipularea obiectelor simulate cu NS–2, să se studieze următorul script :

```
# – Definirea unei metode numite “test”
 proc test {} {
# – a ia valoarea 43
set a 43
# – b ia valoarea 27
set b 27
# – c ia valoarea 27 + 43 = 70
set c [expr $a + $b]
# – d ia valoarea (43 – 27) * 70 = 16 * 70 = 1120
set d [expr [expr $a - $b] * $c]
# – pentru k de la 0 la 9 inclusiv, incrementam variabila k la fiecare bucla
for {set k 0} {$k <10} {incr k} { if {$k <5} {
# – daca k <5 afisam d la puterea k
puts “k <5, pow = [expr pow($d, $k)]”} else {
 # – daca k ≥ 5 afisam d modulo k
puts ”k ≥5, mod = [expr $d % $k]”
}
}
}
 # – Apelarea metodei “test”
test
```

Executarea acest script se face prin simpla utilizare a comenzii :

```
$ ns test.tcl
```

Care este rezultatul obtinut la rularea scriptului ?

2.2 Antrenare cu TCL : traducerea unui script C

Pentru a vă familiariza mai mult cu limbajul TCL, şi pentru a descoperi uşurinţa de utilizare, rescrieţi următorul program C în TCL :

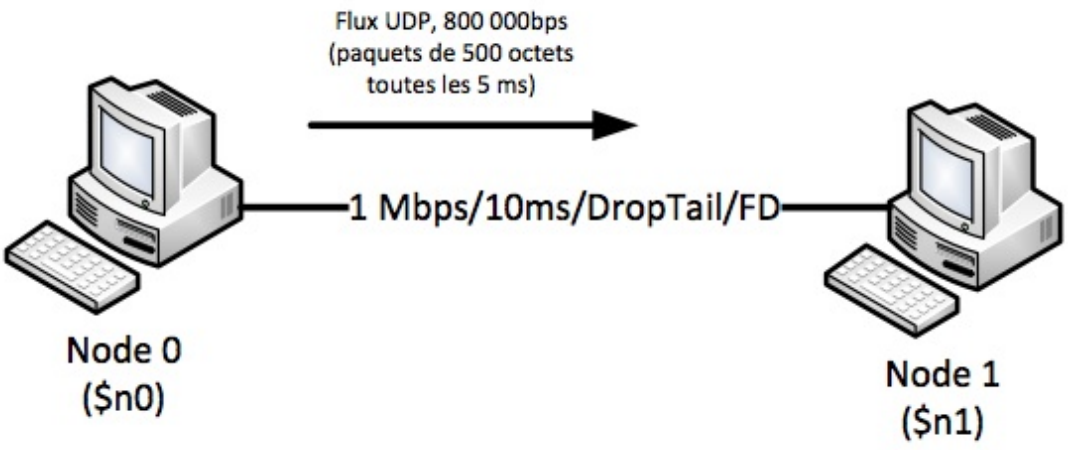
```
#include <stdio.h >
int main( int argc, char *argv[] ) {
int count = 20; /* The number of Fibonacci numbers to print */
int num[] = { 0, 1 }; /* First and second seed values */
int i;
printf( ‘The first %d Fibonacci numbers :\n, count ); for ( i = 1; i <count; i++ ) {
printf( “%d :\t%d \n”, i, num[0] );
num[1] = num[0] + num[1];
i++;
printf( “%d :\t%d \n”, i, num[1] );
num[0] = num[0] + num[1];
}
return 0;
}
```

Pentru declararea unui tabel se foloseste instructiunea :

```
array set num { 00
11
}
```

3 Prima simulare folosind NS–2

Pentru a observa funcţionarea NS–2 şi a simulatorului NAM, simulaţi următoarea reţea simplă :



Pentru aceasta, creaţi două noduri NS-2(node), conectate printr-un full duplex (duplex–link), care suportă un debit de 1Mbps, un timp de acces mediu de 10 ms, şi utilizează algoritmul DropTail pentru gestionarea congestiei.

Apoi creaţi un agent UDP (Agent / UDP), ataşat la nodul n0 (attach–agent), care permite nodului n0 să transmită pachete UDP în întreaga reţea. Creaţi o transmisie de pachete la o rată constantă (Constant Bit Rate–CBR– Application / Trafic / CBR) ataşat la agentul UDP definit mai sus (attach–agent).

Se creează în cele din urmă un agent de vid pentru primirea de pachete UDP sau TCP (Agent / Null). Il ataşăm la nodul n1 (attach–agent) iar apoi conectăm agentul UDP si agentul vid (Connect).

In final, se indică următoarele : urmele de simulare ar trebui să fie logate, se doreste ca NAM să fie lansat la sfârşitul simulării, care va dura 5 secunde şi că CBR va fi lansat incepand cu a 0.5 secundă de simulare si va dura 4.5 secunde.

Salvaţi fişierul sub numele de 2nodes–udp–cbr.tcl.

```
# Crearea unui simulator
set ns [new Simulator]
# Crearea fisierul de urme
set nf [open out.nam w]
# Indica NS-ului sa logheze urmele in fisierul $nf (out.nam)
$ns namtrace-all $nf
# De cum se termina simularea, aceasta procedura va fi apelata
# pentru lansarea automata a vizualizatorului (NAM)
proc finish{} {
# Fortam scrierea in fisierul care contine urmele
global ns nf
$ns flush-trace
```

```
close $nf
# Lansarea instrumentului de vizualizare nam
exec nam out.nam &
# Iesirea din script-ul TCL
exit 0
}
# Crearea celor doua noduri
set n0 [$ns node]
set n1 [$ns node]
# Crearea unei comunicatii de tip full duplex intre nodurile n0 & n1
#care functioneaza la 1Mbps, 10ms de delai, si foloseste pentru coada
#de asteptare algoritmul DropTail
$ns duplex–link $n0 $n1 1Mb 10ms DropTail
# Crearea unui agent UDP atasat lui n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
# Generarea de trafic CBR. Nodul 0 este generator de pachete la viteza constanta
# Pachete de 500 octets (4000 bits), generate la fiecare 5 ms.
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
# Atasam traficul agentului UDP udp0
$cbr0 attach-agent $udp0
# Crearea unui agent vid, destinat sa primeasca pachetelede la nodul n1
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
# Traficul generat de agentul udp0 este trimis catre null0
$ns connect $udp0 $null0
# Inceputul transmisiei CBR la 0.5s de la inceperea simularii
$ns at 0.5 “$cbr0 start”
# Sfarsitul transmisie CBR la 4.5s
$ns at 4.5 “$cbr0 stop”
# Simularea se opreste dupa 5 secunde, si apeleaza procedura
# TCL “finish” definita anteriort
$ns at 5.0 “finish”
# Pornirea simularii
$ns run
```

Lansarea simulării şi vizualizarea rezultatelor se face folosind comanda :

```
$ ns 2nodes-udp-cbr.tcl
```

Acest lucru dă rezultatul urmator :

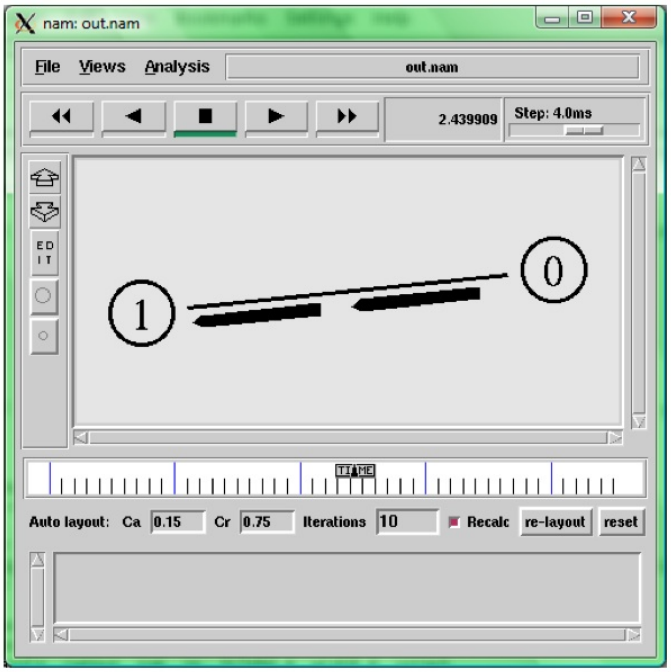
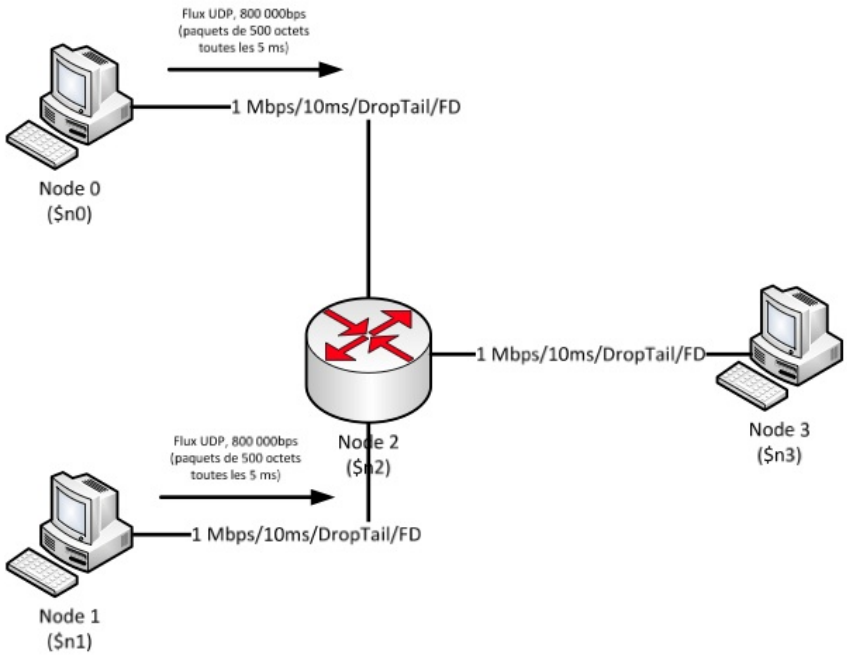


FIGURE 1 – Rezultatul simularii

4 Simulări folosind NS-2

4.1 Simularea congestiei într-o reţea

Simularea congestiei într-o reţea, folosind schema de mai jos :

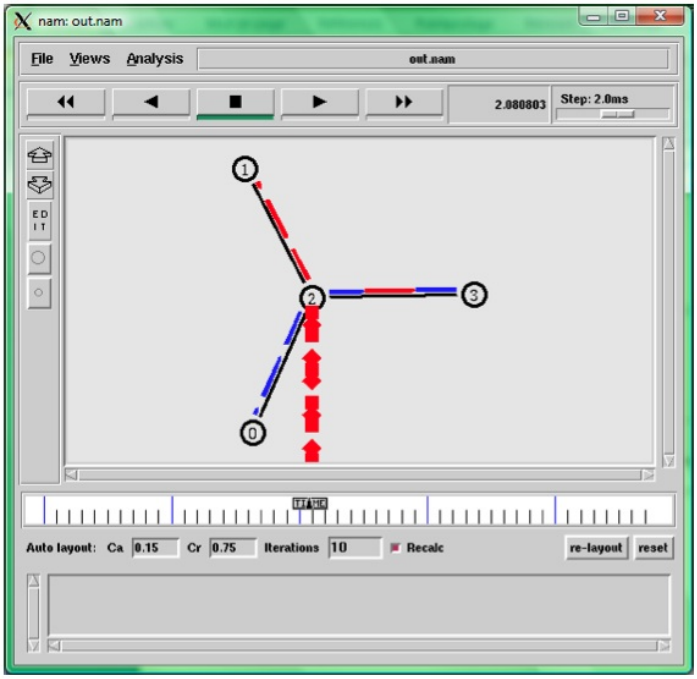


Fie două noduri care transmit CBR de 800 000 bps catre o maşină care asculta cele doua CBR-uri, trecand printr-un nod (router sau switch) la care toate cele trei noduri sunt conectate. Congestionarea va avea loc între router / switch şi nodul destinaţie, deoarece link-ul este limitat la 1 Mbps, în timp ce suma CBR emisa până la destinaţie, este de 1 600 Kbps, deci mai mare.

Ambele noduri începe transmiterea datelor lor în CBR, în acelaşi timp. Observati comporta-

mentul nodului de comutare la acel moment.

Toate cozile sunt link-uri FIFO. Dupa rularea simularii se obţine urmatorul rezulat :



Traficul transmis de cele doua noduri se va colora diferit folosind comenzile :'

```
# Definirea claselor
$udp0 set class_ 1
$udp1 set class_ 2
# Colorarea claselor : albastru pentru udp0 (clasa 1) pour udp1 (clasa 2)
$ns color 1 Blue
$ns color 2 Red
```

Într-adevăr, se observa o saturaţie intre nodurile 2 şi 3. Aceasta are loc din momentul în care cantitatea de pachete ce urmează să fie transmise este prea mare, iar coada de asteptare a link-ului este saturată. Cum politica folosita de coada este FIFO, primele pachete acceptate sunt transmise.

Înainte de saturaţie, un pachet din două transmis de la nodul 2 la nodul 3, provine din fluxul UDP al nodului 0 şi un pachet din două provine de la fluxul nodului 1. Din momentul coada este plină, 37,5% din pachte sunt respinse. Având în vedere ordinea sosirii pachetelor, întotdeauna pachetele de la nodul 1 ($ n1) sunt cele care sunt respinse, adica 75% din pachetele primite.

Folosind FIFO, la nodul 3 ajung toate pachetele provenite de la nodul 0 si un sfert din pachetele de la nodul 1, în timp ce trei sferturi dintre pachetele transmise de la nodul 1 la nodul destinaţie 3 sunt pierdute.

Pentru a îmbunătăţi echitatea în pierderi schimbati politica cozii. Inlocuiti disciplina FIFO cu un SFQ (Stochastic Corectitudinea), care se presupune a fi un algoritm de distribuţie echitabilă. Modificati liniile corespunzatoare din programul scris anterior. Explicati ce se intampla in aceasta situatie. Dar daca se utilizeaza algoritmul RED (Random Early Detection) ca politica a cozii de asteptare ?

4.2 Gestionarea cozilor de asteptare

Anterior am discutat despre cum am putea modula disciplina de coada intr-o reţea de patru stele noduri. Cu toate acestea, dacă doriţi să exploatati în continuare cozile situate pe diferitele link-uri de reţea, este posibil sa se utilizeze comenzi specifice în NS-2 .

Astfel, in NS–2 se poate modula dimensiunea unei cozi cu funcţia queue–limit şi se poate modula poziţia de afişare a conţinutului unui fişier folosind comanda duplex–link–op care acţionează asupra variabilei queuePos.

De asemenea, este posibil de simula pierderea şi recuperarea unui link prin intermediul comenzilor down şi up. Pentru a explora aceste posibilităţi, se porneste de la modelul din secţiunea precedentă, cu cozile CBQ şi pierdere detectata la nodul 2.

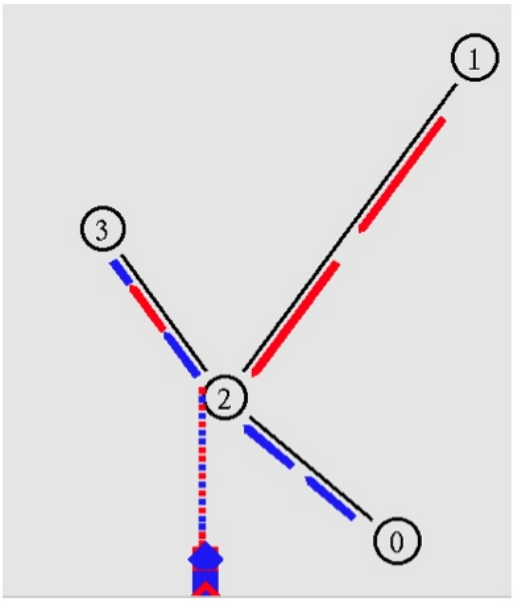
Chiar înainte de liniile :

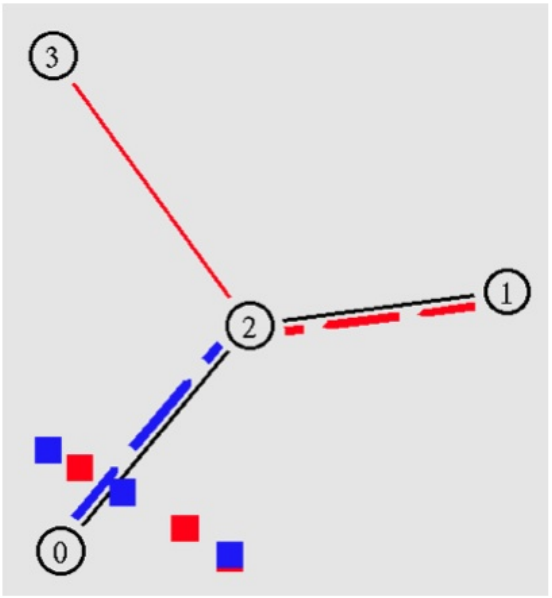
```
# inceperea simularii
$ns run
```

Adăugati următoarele rânduri, utilizate pentru a simula o coada care poate conţine doar 10 pachete, care trebuie poziţionata vizual la 28 de grade (0,5 radiani). Se va tăia legătura dintre nodurile 2 şi 3 pentru 0.5 secunde, incepand din a doua secunda de simulare :

```
$ns queue-limit $n2 $n3 10
$ns duplex-link-op $n2 $n3 queuePos 0.5
$ns rtmodel-at 2.0 down $n2 $n3
$ns rtmodel-at 2.5 up $n2 $n3
```

Se lansează simularea. Diagramele de mai jos arată, comportamentul cu o coadă de 10 KB, şi comportamentul atunci când legătura dintre nodurile 2 şi 3 dispare.





Observati vizual cate pachete poate contine coada. Care este comportamentul disciplinei SFQ ?

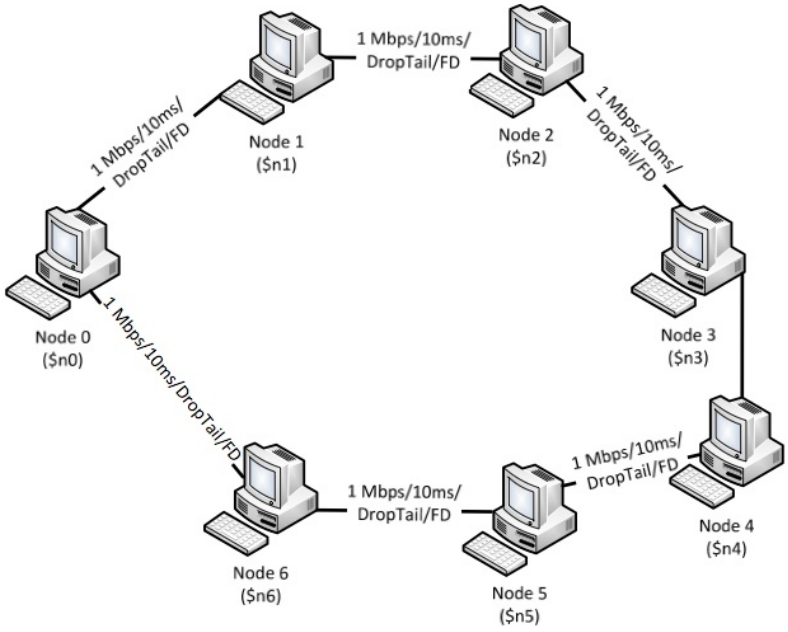
După ruperea legăturii dintre nodurile 2 şi 3, coada este golita instantaneu, deoarece destinatarul (nodul 3) nu mai poate fi contactat de router / switch (nodul 2), şi, prin urmare, nu este necesar ca pachetele sa fie stocate ; ele nu vor fi transmise.

Prin urmare, nodul 2 continuă să primească pachete de la nodurile 0 şi 1, dar le arunca (drop), deoarece nodul destinatar nu este cunoscut.

Odată ce link-ul este restaurat, coada se va umple din nou, pe mesura ce se incarca link-ul, şi după aproximativ o jumătate de secundă, comportamentul sistemului este acelaşi ca şi în cazul precedent : saturaţia cozii de asteptare şi de pierderea pachetelor primite.

5 Topologie dinamica in inel

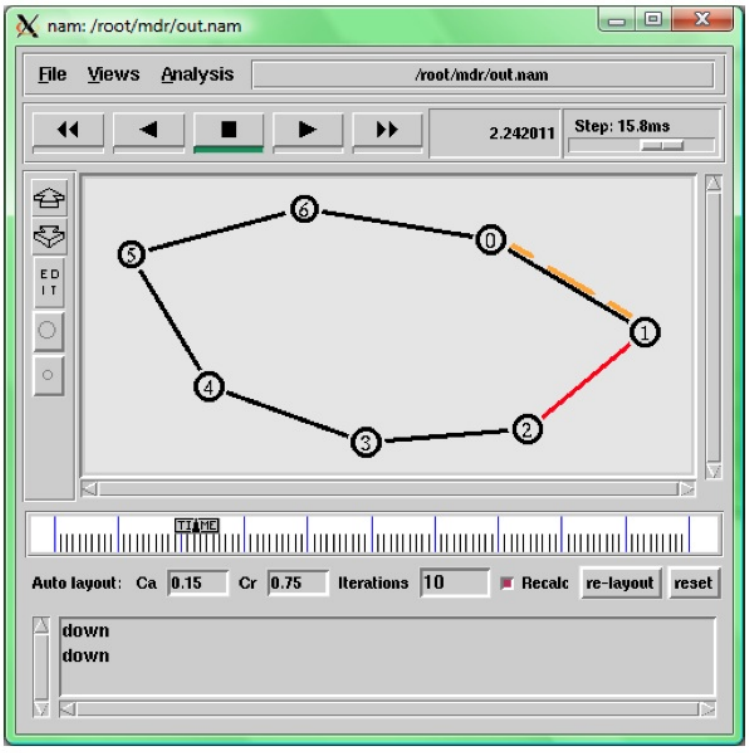
Pentru a studia protocoalelor de rutare puse în aplicare în NS-2, să se studieze cazul unei reţele inel care corespunde următoarei diagrame :



Toate link-urile sunt de lungime egală, şi au caracteristici similare (timpul de acces, viteza de transmitere, disciplina cozii).

Scopul este de a emite CBR de la nodul 0 la nodul destinaţie 3, începand de la a doua secunda pana la a 10 secunda de simulare, tăind legătura dintre nodurile 1 şi 2 timp de 2 secunde, după 2 secunde de simulare .

Rezultatul pe care il obtinem la început este :



In locului unui agent vid se foloseste un agent sink0 care monitorizeaza pierderile pachetelor.

```
set sink0 [new Agent/LossMonitor]
```

Este de remarcat faptul că, în mod normal, pachetele trimise de nodul de la 0 la nodul destinaţie 3 sunt dirijate prin intermediul link-urilor 0–1, 1–2 şi 2–3. După ruperea legăturii între nodurile 1 şi 2, ruta nu este recalculată, determinând pierderea tutoror pachetelor transmise in cele 2 secunde. Cate pachete se vor pierde in acest caz ?

In mod implicit, NS–2 foloseşte un algoritm de rutare folosind proximitatea pentru a determina calea cea mai scurtă, dar cu tabele de rutare statica. În cazul ruperii unei legături, drumul spre nodul 3 nu este recalculat.

Pentru a depăşi acest fenomen, este posibil să se precizeze algoritmul de rutare care urmează să fie utilizat în NS-2. De exemplu, puteţi alege algoritmul de rutare dinamică Distance Vector (DV) pentru a determina calea cea mai scurtă dintre link-urile disponibile.

Pentru a specifica algoritmul de rutare, pur şi simplu adăugaţi următoarea linie imediat după crearea simulatorului :

```
$ns rtproto DV
```

Prin urmare, atunci când link-ul 1–2 dispare, pachetele trec prin noul drum cel mai scurt : 0–6, 6–5, 5–4, 4–3 după cum se arată în diagrama de mai jos.

Pachetele pierdute sunt acelea care au fost deja transmise prin link-ul 0–1, sau erau deja in coada de aşteptare pentru a fi trimise pe link-ul de 1–2 in momentul ruperii link-ului. Acest număr este în mod substanţial neglijabile în comparaţie cu numarul de pachete anterior pierdut.

Ce se intamplă cu pachetele dupa ce link-ul este restaurat ? Care este drumul pe care se transmit pachetele ? De ce diferă de cel anterior (momentul in care nu mai exista link intre nodurile 1 si 2) ?