Dragoș NICULESCU

INFRASTRUCTURI ȘI SERVICII
pentru
REȚELE MOBILE
Îndrumar de laborator

Editura POLITEHNICA PRESS
BUCUREȘTI, 2017

Infrastructuri și servicii pentru rețele mobile

Autor: conf.dr.ing. Dragoș Niculescu
Editura: POLITEHNICA PRESS
Adresa: Splaiul Independenței, nr. 313, sector 6, București
email: edituraupb@yahoo.com telefon: 021.402.90.74
ISBN: 978-606-515-782-8

Universitatea Politehnica din București
Facultatea de Automatică și Calculatoare
Catedra de Calculatoare
Direcția de Masterat "Securitatea Rețelelor Informatice Complexe"

Cuprins

Capitolul 1

Introducere

1.1 Scopul laboratorului

Cursul de "Infrastructuri și servicii pentru rețele mobile" este un curs de inițiere în cercetare. Această inițiere presupune citirea literaturii de specialitate, dezbaterea critică și comparativă a ideilor propuse îon articole de cercetare recente, propunerea și analizarea unor soluții proprii. Acest ultim scop al propriiilor analize este în atenția laboratorului prin aprofundarea a patru competențe importante:;

1. înțelegerea intimă a sistemelor și protocoalelor studiate.

2. implementarea unor topologii de simulare/experimentare care permit măsurarea

3. colectarea și prelucrarea datelor din măsurare/simulare

4. trasarea graficelor si interpretarea

Prima competență este desigur câștigată și prin citirea articolelor sau cărților, dar implementarea sau simularea aduc un nivel superior de înțelegere. Pe parcursul laboratorului vor fi exersate activ ultimele trei competențe: simulare, prelucrare date, trasare si interpretare. Pentru parcurgerea laboratoarelor este necesară o cunoaștere la nivel mediu a protocoalelor din familia 802.11 [1].

1.2 Simularea rețelelor fără fir

În general în cercetare sunt acceptate trei metode de caracterizare a unui fenomen, sau sistem complex. Prima metodă este aceea a analizei teoretice, si este preferabila ori de câte ori sistemul este tractabil matematic. Din nefericire, puține sisteme reale au o comportare care poate fi caracterizată matematic. O a doua metodă este aceea a simulării unui protocol sau sistem pentru a mima comportamentul real cât mai fidel. Simularea este desigur o simplificare, dar poate fi folosită în mod judicios atunci când caracteristicile simulate nu fac simplificări ce produc rezultate eronate. A treia metodă, și cea mai credibilă în cercetarea systems (rețele, sisteme de operare, BD, sisteme distribuite) este aceea a implementării. Este desigur cea mai dificilă, dar cu rezultatele cele mai credibile din punct de vedere al performanței și functionalității.

În acest laborator se folosește simulatorul ns-2 și MAC-ul de 802.11[2], împreună cu două modele de nivel fizic. Explorarea în simulare are câteva avantaje notabile:

- simulatorul ns-2[3] a fost testat cu multe topologii si protocoale, având un grad de corelare ridicat cu rezultatele obținute pe sisteme reale
- permite studentului să exploreze topologii mari într-un interval scurt de timp
- permite rularea a multe experimente cu variații minore care ar fio prohibitive de pregătit în realitate
- permite studentului accesul la toate părțile sistemului pentru monitorizare, contorizare, si statistică, ceea ce poate fi dificil pe sisteme reale
- permite realizarea facilă a unor situații radio idealizate dificil de obtinut în realitate, sau a unor situații greu repetabile
- permite rularea în paralel a experimentelor diverse pentru a grăbi obtinerea graficelor

Un alt simulator opensource care beneficiază în prezent de validare extensivă este ns-3 [4], însă posibilitățile curente de scripting nu sunt la fel de potrivite pentru laboratoarele de două ore.

Toate materialele aferente acestui îndrumar, inclusiv textul, codul, imaginile, solutiile sunt disponibile pe site-ul cursului [5], unde sunt în permanență actualizate. În particular, scripturile disponibile pe site la secțiunea Media Manager/laboratoare/src și referite în îndrumar sunt:

- `infra.tcl` - topologie de bază pentru modul infrastructură cu un AP în mijloc cu mai mulți clienți plasați în cerc sau în linie
- `cw.tcl` - topologie cu populație mare de surse si destinații pentru explorarea mediilor WiFi aglomerate
- `cw-fair.tcl` - aceeași topologie cu populații mari, care permite explorarea echității la concurența între vorbitorii WiFi
- `shadow2.tcl` - topologie cu două noduri care compară două modele de propagare la nivelul radio
- `twoflows.tcl` - topologie cu 4 noduri care explorează terminalele ascunse si expuse
- `multirate.tcl` - folosirea ratelor diverse sub un singur AP
- `string.tcl` - topologie multihop

1.3 Instalarea simulatorului ns-2

Pentru a folosi ns-2 în ubuntu/debian, sunt necesare următoarele pachete:

```
# apt-get install build-essential autoconf automake
  libxmu-dev libxt-dev libx11-dev libxt6 gnuplot-x11
```

În aceste laboratoare se folosește un ns-2.34 modificat pentru a utiliza agentul de rutare NOAH, și pentru a dezactiva ARP. În acest mod experimentele nu sunt afectate de funcționarea unui algoritm de rutare sau de stabilirea tabelelor ARP. În realitate, aceste fenomene necesită eliminarea sistematică datelor de la începutul experimentelor pentru a obține măsurători stabile.

- descărcați de pe pagina cursului arhiva ns-allinone-2.34.tgz [85MiB] — are probabil numele /Downloads/ns-allinone-2.34.tgz
- comutați în modul superuser:

```
$ sudo bash
# cd /
# tar xzvf ~/Downloads/ns-allinone-2.34.tgz
# ln -s /opt/ns/bin/ns /usr/local/bin
# ln -s /opt/ns/bin/nam /usr/local/bin
```

- ca user normal:

```
$ which ns   #dorim /usr/local/bin/ns
$ wget http://www.isi.edu/nsnam/ns/tutorial/\
      examples/example1b.tcl
$ ns example1b.tcl
```

1.4 Surse de inspirație

- Laboratorul 1 conține text bazat pe ghidul NS by example[6]
- Din laboratorul 2, tutorialul de gnuplot este o reproducere identică, în limba româna a ghidului scurt de la adresa: http://www.gnuplotting.org/plotting-data/
- Laboratorul 5 reface o parte dintre experimentele propuse în articolul [7]
- Laboratoarele 6 și 9 sunt bazate pe două laboratoare de la centrul wireless Illinois [8].
- Laboratorul 10 verifică rezultate dintr-un articol clasic din domeniu [9]

Capitolul 2

Laboratorul 1

2.1 Introducere în ns-2

NS-2 este un simulator bazat pe evenimente dezvoltat la UC Berkely scris in C++ si OTcl. NS-2 este în primul rând util pentru simularea rețelelor LAN și WAN, dar și pentru rețele wireless, adhoc, de senzori. Cu toate ca NS este destul de ușor de utilizat pentru avansați, poate părea dificil la prima utilizare, deoarece există puține manuale user-friendly. Chiar dacă există o mulțime de documente scrise de către dezvoltatori, care conțin explicații aprofundate ale simulatorului, acestea sunt scrise pentru utilizatorii NS-2 avansați. Această secțiune oferă o idee de bază a modului în care funcționează simulatorul, cum se pregătește o simulare, unde se găsesc componentele interne, cum se configurează componentele de rețea, etc. În principal se prezintă exemple simple și explicații bazate pe experimente simple, familiare unui student care a luat un curs introductiv de rețele.

2.1.1 Limbajul OTcl

NS-2 este în esență un interpretor de OTcl cu biblioteci de obiecte C++ specializate în simularea rețelelor. Este necesar să se cunoască sumar programarea în OTcl pentru a utiliza NS-2. Această secțiune prezintă un exemplu OTcl care ilustrează ideea de bază a programării în OTcl, exemplu importat din manualul oficial NS-2. Această secțiune, ca și secțiunile următoare presupun că cititorul a instalat NS-2, și este familiarizat cu shell și cu C++.

Primul exemplu este un script general OTcl, care arată modul de declarare o procedură și a o apela, modul de a atribui valori variabilelor, și modul de a implementa o buclă. OTcl este de fapt extensia orientată obiect a Tcl(pronunțat "tickle"), relația dintre Tcl și Otcl este la fel ca și cea dintre C și C++, dar în acest laborator rareori sunt folosite obiectele în OTcl. Pentru a rula acest script rulați "ns ex-tcl.tcl" la promptul shell - comanda "ns" execută

NS-2 (interpretorul OTcl). Veți obține aceleași rezultate dacă tastați "tclsh ex-tcl.tcl", în cazul în care tcl8.0 este instalat pe mașina dumneavoastră.

```
# Writing a procedure called "test"
proc test {} {
    set a 43
    set b 27
    set c [expr $a + $b]
    set d [expr [expr $a - $b] * $c]
    for {set k 0} {$k < 10} {incr k} {
if {$k < 5} {
    puts "k < 5, pow = [expr pow($d, $k)]"
} else {
    puts "k >= 5, mod = [expr $d % $k]"
}
    }
}

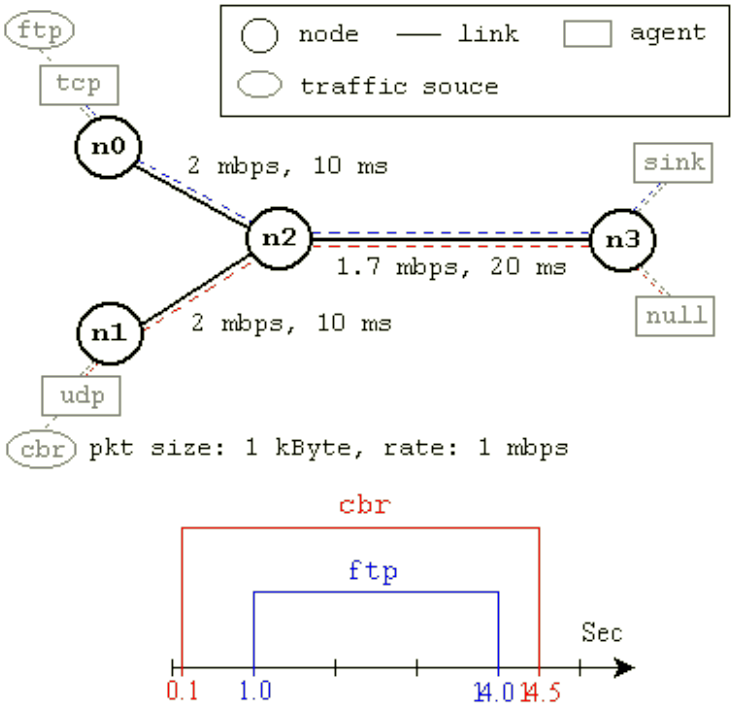
# Calling the "test" procedure created above
test
```

În Tcl, cuvântul cheie `proc` este folosit pentru a defini o procedură, urmat de un nume de procedură și argumentele în acolade. Cuvântul cheie `set` este folosit pentru a atribui o valoare unei variabile. `[Expr ...]` este pentru a calcula valoarea expresiei între paranteze drepte, după cuvântul cheie. Un lucru de remarcat este faptul că pentru a obține valoarea atribuită unei variabile, `$` este utilizat cu numele variabilei (ca în shell). Cuvântul cheie `puts` imprimă șirul de caractere la consolă. Rulând exemplul de mai sus, se obține:

```
k < 5, pow = 1.0
k < 5, pow = 1120.0
k < 5, pow = 1254400.0
k < 5, pow = 1404928000.0
k < 5, pow = 1573519360000.0
k >= 5, mod = 0
k >= 5, mod = 4
k >= 5, mod = 0
k >= 5, mod = 0
k >= 5, mod = 4
```

2.1.2 Primul exemplu de simulare

Această secțiune prezintă un script simplu NS-2 si explica ce face fiecare linie. Exemplul este un script OTcl care creează o configurație de rețea simplă și rulează scenariul de simulare din figura de mai jos. Pentru a rula această simulare, descărcați "ns-simple.tcl" și rulați "ns ns-simple.tcl" în shell.



Această rețea este formată din 4 noduri (n0, n1, n2, n3), așa cum se arată în figura de mai sus. Link-urile duplex între n0 și n2, și n1 și n2 au capacitatea 2Mbps[Megabit/s] și 10ms de întârziere. Link-ul duplex între n2-n3 are capacitatea de 1.7Mbps și 20ms de întârziere. Fiecare nod folosește la ieșire o coadă DropTail, a cărei mărime este de 10 pachete. Un agent "TCP" este atașat la n0, și este stabilită o conexiune la un sink TCP atașat la n3. În mod implicit, dimensiunea maximă a unui pachet pe care un agent de TCP poate genera este 1KByte. Un sink TCP generează și trimite pachete ACK către expeditor (agent TCP) și livrează la aplicație pachetele primite. Un agent UDP, care este atașat la n1 este conectat la un agent sink (LossMonitor) atașat la n3. Un agent LossMonitor doar contorizează pachetele primite. Aplicații de tio FTP și generator de trafic CBR sunt atașate la agenții TCP șiUDP respectivi, iar CBR este configurat pentru a genera pachete de 1000 bytes la rata de 1Mbps. Aplicația CBR este setată să pornească de la 0.1s, și să se oprească la 14,5 sec, iar FTP pornește de la 1.0s și se oprește la 14.0s.

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#ns trace file
set tracefd       [open simple.tr w]
$ns use-newtrace
$ns trace-all $tracefd


#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 5
$ns queue-limit $n2 $n3 5

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link n2-n3 (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5

#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sinkt [new Agent/TCPSink]
$ns attach-agent $n3 $sinkt
$ns connect $tcp $sinkt
```

```
$tcp set fid_ 1

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set usink [new Agent/LossMonitor]
$ns attach-agent $n3 $usink
$ns connect $udp $usink
$udp set fid_ 2

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false

#Define a 'finish' procedure
proc finish {} {
    global ns nf fout
    $ns flush-trace
    #Close the NAM trace file
    close $nf
    #Execute NAM on the trace file
    exec nam out.nam &
    exit 0
}

#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 14.0 "$ftp stop"
$ns at 14.5 "$cbr stop"

#Call the finish procedure after 15 seconds of
#simulation time
$ns at 15.0 "finish"

#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"
```

```
#Run the simulation
$ns run
```

2.1.3 Descrierea script-ului

În general, un script ns-2 începe cu o instanță a obiectului simulator:

- `set ns [new Simulator]` generează un obiect de tip simulator și îl atribuie variabilei *ns* (caracterele cursive sunt folosite pentru variabile și valori). Această linie:

  1. inițializează formatul pachetelor (ignorați acest lucru deocamdată)
  2. crează un planificator
  3. selectează formatul adreselor (ignorați acest lucru deocamdată)
  4. Obiectul "simulator" are funcții de membru care:
  5. crează obiecte compuse, cum ar fi nodurile și legăturile (descrise mai târziu)
  6. conectează componentele de rețea create (de ex., `attach-agent`)
  7. setează parametrii componentelor de rețea (mai ales pentru obiectele compuse)
  8. crează conexiuni între agenți (de ex., Face legătura între un TCP și sink)
  9. specifică opțiunile NAM (network animator)

Cele mai multe dintre funcții sunt pentru configurarea simularii și pentru planificare, iar unele dintre ele sunt pentru animatorului NAM. Multe alte setări ale simulatorului pot fi consultate în fișierul `ns-2 / TCL / lib / ns-lib.tcl`.

- `$ns color fid color` stabilește culoarea pachetelor pentru un flux specificat de id-ul fluxului (fid). Acest membru al obiectului "simulator" este pentru afișajul NAM, și nu are nici un efect asupra simulării propriuzise.

- `$ns namtrace-all file-descriptor` Acest membru specifică un fișier trace de simulare în format specific NAM. Acesta oferă numele de fișier pe care informațiile de trace vor fi scrise mai târziu de comandă `$ns flush-trace`. În mod similar, `trace-all` este pentru înregistrarea trace-urilor de simulare într-un format general.

- `proc finish` este rulată la sfârșitul simulării când se interpretează comanda `$ns at 15.0 "finish"`. În această funcție, sunt specificate diverse procesări post-simulare.

- `set n0 [$ns node]` Funcția membru `node` creează un nod nou. Un nod în NS-2 este un obiect compus din adresa si port clasificator. Utilizatorii pot crea un nod prin crearea separată a adresei și a unui port clasificator, și conectarea lor ulterioară. Pentru a vedea modul în care se creează un nod, se pot consulta fișierele: `ns-2 / tcl / libs / ns-lib.tcl` și `ns-2 / tcl / libs / ns-node.tcl`.

- `$ns duplex-link node1 node2 bandwidth delay queue-type` creează **două** legături simplex de bandă și întârziere specificate, pentru a conecta cele două noduri. În NS-2, coada de ieșire a unui nod este implementată ca parte a unei legături, prin urmare, utilizatorii ar trebui să specifice tipul de coadă atunci când crează legături. În script-ul de mai sus, este utilizată coadă de tip DropTail. În cazul în care se dorește o coadă RED, pur și simplu se înlocuiește DropTail cu RED. Implementarea NS-2 a unei legături este prezentată într-o secțiune ulterioară. Ca și nodurile, link-urile sunt obiect compuse, deci se pot crea sub-obiecte ce pot fi conectate. Codul sursă pentru legături poate fi găsit în "ns-2 / tcl / libs / ns-lib.tcl" și "ns-2 / tcl / libs / ns-link.tcl". Se remarcă faptul că se pot insera modulele de eroare într-o legătură pentru a simula o legătură cu pierderi (de fapt, utilizatorii pot crea și insera orice obiect de rețea). Consultați documentația pentru NS pentru detalii.

- `$ns queue-limit node1 node2 number` Această linie stabilește dimensiunea cozii de așteptare a celor două legături simplex care conectează node1 și node2. Aruncați o privire la "ns-2 / tcl / libs / ns-lib.tcl" și "ns-2 / tcl / libs / ns-link.tcl", sau documentația NS-2 pentru mai multe informații.

- `$ns duplex-link-op node1 node2 ...` Urmatoarea linie este folosită pentru afișajul NAM. Pentru a vedea efectele ei, utilizatorii pot comenta și rerula simularea.

Odată configurată rețeaua de bază, trebuie configurați agenții de generare/consum trafic, cum ar fi TCP și UDP, aplicațiile de generare/consum trafic, cum ar fi FTP și CBR. Acestea trebuie atașate nodurilor și agenților într-un mod similar cu rularea aplicațiilor peste anumite protocoale în nodurile reale.

- `set tcp [new Agent/TCP]` Această linie crează un agent TCP. Agenții și sursele de trafic sunt obiecte de bază în cea mai mare parte implementate în C++ și "legate" în OTcl. Prin urmare, nu există funcții specifice ale obiectului simulator care creează aceste instanțe. Pentru a crea agenți sau surse de trafic, trebuie să se cunoască numele clasei (`Agent/TCP`, `Agent/TCPSink`, `Application/FTP`, etc). Aceste informații pot fi găsite rapid în fișierul "ns-2/tcl/libs/ns-default.tcl". Acest fișier conține configurări implicite pentru diverse obiecte. Sursele sunt un bun indicator pentru obiectele disponibile în NS-2 și ce parametrii lor.

- `$ns attach-agent node agent` Funcția `atttach-agent` conectează un obiect agent la un nod.

- `$ns connect agent1 agent2` După ce s-au creat doi agenți care se vor comunicatrebuie stabilită o conexiune logică între ei. Această linie stabilește o conexiunea prin setarea reciprocă a adreselor și a porturilor.

Odată configurația rețelei specificată, următorul lucru este implementarea unui scenariu de simulare. Obiectul Simulator are funcția `at`:

- `$ns at time \string"` Această funcție programează executarea șirului specificat la un moment de simulare. De exemplu, `$ns at 0.1 "$CBR start"` va face scheduler-ul să apeleze o funcție membru de start al obiectului `$CBR` la momentul `0.1s`. În NS-2, o sursă de trafic nu transmite date reale, ci notifică agentul care de la baza o anumită cantitate de date, iar agentul, creează pachete și le trimite în mod fidel implementărilor din kernel, **considerând un CPU cu viteză infinită**.

După ce s-au efectuat toate specificațiile de configurare a rețelei, specificat procedura post-simulare, singurul lucru rămas este rularea propriu-zisă. Acest lucru se face cu funcția `$ns run`.

- La rularea din shell `ns ./ns-simple.tcl`, se execută simularea și se generează "filmul simulării" out.nam, apoi se lansează animatorul `nam`. Rulați slide-ul în animator pentru a accelera filmul, observând transferul pachetelor și comportarea cozii din nodul 2.

2.2 Trasarea unui grafic

ns-2 permite implementarea cu ușurință a procedurilor specializate de generare de loguri. Aceste proceduri sunt apelate periodic în timpul simulării și permit adresarea tuturor datelor specifice nodurilor, fluxurilor, cozilor, și ale celorlalte

entități din rețea. În partea de final a scriptului de mai sus, inserați patch-ul următor. Apelată periodic, această procedură contorizează numărul de octe ți primiți de destinația UDP, și numărul de octeți confirmați la sursa TCP. În acest mod, se poate calcula debitul obținut de cele două fluxuri pe intervale fixe de timp. Cele două valori sunt stocate periodic într-un fișier text `out.tr`.

```
#Open a trace file
set fout [open out.tr w]

#save running byte counters
set tbytes 0
set ubytes 0

proc record {} {
  global tcp usink fout ubytes tbytes

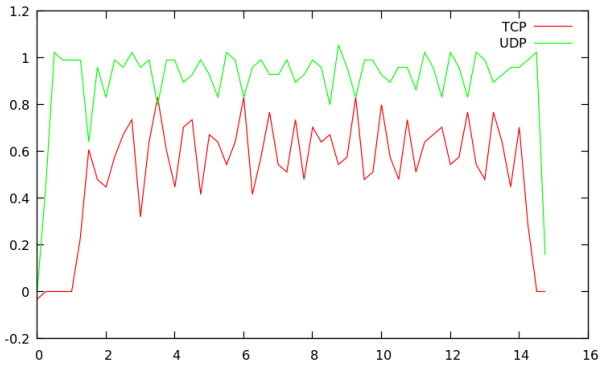
  set ns [Simulator instance]
  #time after which the procedure should be called again
  set time 0.25
  #How many bytes have been received/acked?
  set tbytes1 [expr [$tcp set ack_]*[$tcp set packetSize_]]
  set ubytes1 [expr [$usink set bytes_]]
  set now [$ns now]
  #Calculate bandwidth in MBps and write it to the log
  puts $fout "$now [expr ($tbytes1 - $tbytes)/$time*8/1e6] \
                   [expr ($ubytes1 - $ubytes)/$time*8/1e6]"
  #Reset the bytes_ values on the traffic sinks
  set tbytes $tbytes1
  set ubytes $ubytes1
  #Re-schedule the procedure
  $ns at [expr $now+$time] "record"
}
```

Procedura `finish` trebuie actualizată pentru a include închiderea fișierului de trace, și pentru a plota conținutul său:

```
#Close the trace file
 close $fout
#Call gnuplot to display the results
 exec echo  "plot 'out.tr' using 1:2 t 'TCP' w l, \
         '' using 1:3 t 'UDP' w l" | gnuplot -persist
```

În plus, înainte de a demara simularea, trebuie să armăm procedura record cu `$ns at 0.0 "record"`. După execuția scriptului, se vor lansa automat

atât fereastra animatorului, cât și o fereastră gnuplot care afișează conținutul fișierului trace `out.tr`.



2.3 Exerciții

1. examinați cu editorul de text conținutul fișierelor `out.nam` și `out.tr`
2. Ce reprezintă axele x, y? Explicați comportarea graficelor.
3. Măriți coada de la link-ul bottleneck la 100. Cum explicați noua comportare?
4. Coada aruncă pachete TCP în mod disproporționat. De ce? Folosiți o coadă SFQ pentru a remedia situația.
5. În prezent se măsoară debitul la fiecare 250ms. Experimentați cu diverse rezoluții de măsurare în funcția "record".
6. **Facultativ**: monitorizarea cozii (trace-ului queue)

```
set qfile [$ns monitor-queue $n2 $n3
                [open queue.tr w] 0.1]
[$ns link $n2 $n3] queue-sample-timeout
```

2.4 Documentație suplimentară

- Tutorial oficial ns-2 de Marc Greis[10], NS by Example [6].
- Manual ns-2 [11]
- Tutorial TCL [12]

Capitolul 3

Laboratorul 2

3.1 Introducere în gnuplot

Plotarea[1] datelor rezultate din măsurători sau simulări este probabil cea mai frecventă utilizare a programului gnuplot. Gnuplot are două moduri generale de funcționare - interactiv și script. În modul interactiv, fiecare comandă este scrisa la promptul gnuplot, cu posibilitatea de a folosi la sfârsit comanda `save` pentru a salva comenzuile într-un script. Un script scris manual, sau obtinut cu `save` poate fi pasat ca parametru programului gnuplot pentru a genera grafice.

Datele discrete de intrare corespund funcțiilor date prin puncte. Avem nevoie de un fișier de date de intrare și de câteva comenzi pentru a manipula datele. Vom începe cu plotarea de bază a datelor simple și apoi vom analiza plotarea datelor cu erori.

3.1.1 Date simple

La început, vom analiza un fișier de date. Acesta poate fi un fișier text care conține datele ca coloane. Considerăm fișierul de date numit plotting_data1.dat ce conține (gnuplot-ul ignoră liniile care încep cu diez #):

```
# plotting_data1.dat
# XY
  1 2
  2 3
  3 2
  4 1
```

Poate fi plotat scriind:

[1] Această secțiune este traducerea în limba româna a ghidului [13]

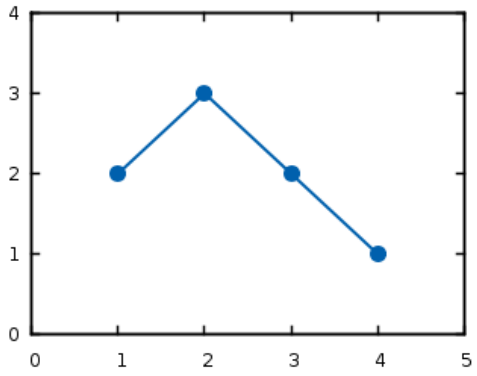


Figura 3.1: Plotul datelor din plotting_data1.dat

```
set style line 1 lc rgb '#0060ad' lt 1 lw 2 pt 7 ps 3.5
# albastru:
plot 'plotting_data1.dat' with linespoints ls 1
```

Aici setăm tipul de punct (pt) și dimensiunea punctului (ps) de utilizat. Pentru stilurile de puncte și linii disponibile, puteți să rulați comanda test la promptul gnuplot. Imaginea rezultată este prezentată în figura 3.1.

Dacă avem colecții de puncte care reprezintă date ne-continue, putem indica aceasta prin introducerea unei linii libere între date (Figura 3.2).

```
# plotting_data2.dat
# XY
  1 2
  2 3

  3 2
  4 1
```

Dacă dorim să folosim altă culoare pentru cel de-al doilea set de date care este totuși în același fișier, putem introduce încă o linie liberă. Apoi trebuie să indexăm blocul de date începând cu comanda 'index'.

```
# plotting_data3.dat
# Primul bloc de date (index 0)
# X Y
  1 2
  2 3
```

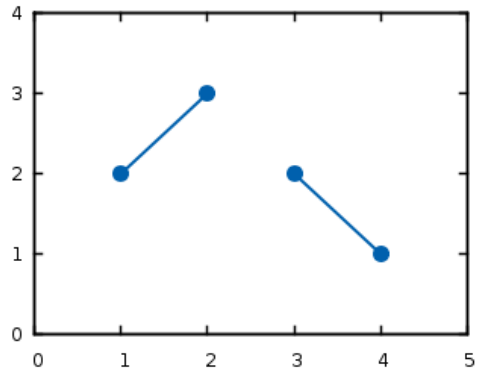


Figura 3.2: Plotarea datelor din plotting_data2.dat ca linie non-continuă

```
# Al doilea bloc de index (index 1)
# X Y
  3 2
  4 1
```

Și plotăm cu comenzile:

```
# albastru:
set line style 1 lc rgb '# 0060ad' lt 1 lw 2 pt 7 ps 3.5
# roșu:
set line style 2 lc rgb '# dd181f' lt 1 lw 2 pt 5 ps 3.5
plot 'plotting-data3.dat' index 0 w lp ls 1 , \
     '' indicele 1 w lp ls 2
```

După cum putem vedea, am adăugat un alt tip de culoare și punct și am reprezentat cele două seturi de date utilizând indexul și separat parcelele cu o virgulă. Pentru a reutiliza ultimul nume de fișier, putem scrie doar " . Rezultatul este prezentat în figura 3.

3.1.2 Datele cu erori

O necesitate frecventă este reprezentarea datelor cu bare de erori pentru a indica de exemplu comportarea funcției într-un punct. În următorul exemplu avem măsurători ale puterii pentru o rezistență dată, stocate în formatul: r, P, Perror care poate semnifica eroarea de măsurare a puterii:

```
# battery.dat
# X Y stddev
```

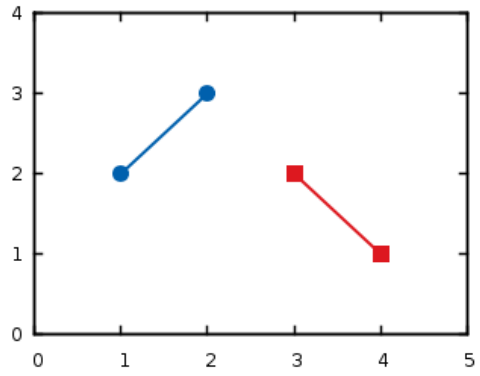


Figura 3.3: Plotarea datelor din plotting_data3.dat cu două stiluri diferite.

```
50.000000 0.036990 0.007039
47.000000 0.036990 0.007039
44.000000 0.038360 0.007053
41.000000 0.042160 0.007050
38.000000 0.043200 0.007018
35.000000 0.046900 0.007021
32.000000 0.048840 0.006963
29.000000 0.052000 0.006929
26.000000 0.055470 0.006947
23.000000 0.060000 0.006882
20.000000 0.064660 0.006879
17.000000 0.069600 0.006936
14.000000 0.079800 0.007080
11.000000 0.086920 0.007232
8.000000 0.085500 0.007262
5.000000 0.101260 0.008415
2.000000 0.091000 0.011203
0.000000 0.081480 0.011828
```

Vom plota astfel:

```
set xrange [ -2 : 52 ]
set yrange [ 0 : 0.12 ]
set format y '% .0s'
plot 'battery.dat' using 1:2:3 \
     w yerrorbars ls 1 , \
     '' using  1:2 w lines ls 1
```

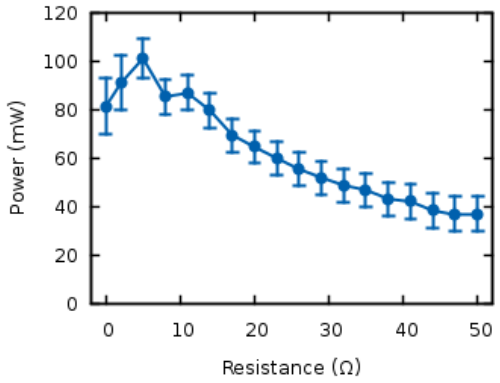


Figura 3.4: Plotarea datelor din battery.dat cu erori pentru axa y (putere).

Valorile puterii sunt stocate în Watt în fișierul de date, dar au valori mai mici decât 1. De aceea dorim să folosim mW ca unitate de măsură. Așadar, am setat opțiunea de format pentru a spune gnuplot-ului să folosească "mantisa as base of current logscale", vezi documentația gnuplot . Apoi, cu indicația using, se specifică gnuplot ce coloane din fișierul de date ar trebui să utilizeze. Deoarece vrem să plotăm eroarea puterii și puterea, avem nevoie de trei coloane - 1,2, și 4. Folosind stilul de plotare yerrorbars nu este posibilă combinarea punctelor cu o linie. Prin urmare, adăugăm o a doua linie la comanda plot pentru a combina punctele cu o linie. Acest lucru ne va da rezultatul din figura 3.4. Putem evita comanda set format din ultimul grafic prin manipularea directă a datelor de intrare:

```
set yrange [ 0:120 ]
plot 'battery.dat' using 1:($2*1000):($4*1000)\
                   w yerrorbars ls 1
```

Atenție, sunt necesare paranteze în jurul expresiilor pe coloană, iar numărul coloanei este indicat cu $column_number .

În ultimul grafic vom adăuga date teoretice și o legendă:

```
# Legendă
set key at  50,112
set xlabel 'Resistance (ohm)'
set ylabel 'Power (mW)'
# Curba teoretică
P(x) = 1.53**2 * x/(5.67 + x)**2 * 1000
plot 'battery.dat' using 1:($2*1000):($4*1000) \
                    title "Power" w yerr ls 2 , \
      P(x) title 'Theory' w lines ls 1
```

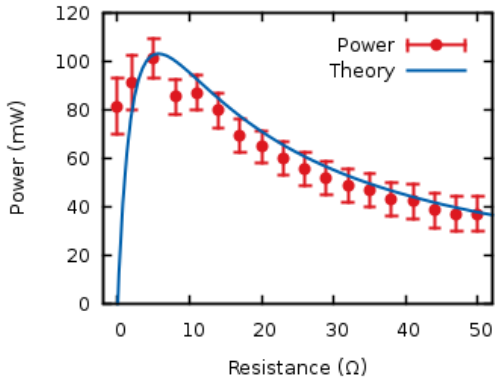


Figura 3.5: Plotarea datelor din battery.dat cu erori y și o curbă teoretică.

În general, legenda este activată cu comanda set key, iar poziția sa poate fi specificată prin set key top left, etc. Puteți, de asemenea, să o setați direct la un punct așa cum am făcut-o aici, pentru a avea suficient spațiu între chei și ticuri. Cuvântul cheie din title în comanda complot specifică textul care va fi afișat în legendă.

Alte comenzi utile gnuplot:

```
set term png        # setează terminalul într-un format
set out 'file.png'  # va plota într-un fisier
set xrange [0:4]    # domeniul funcției
set ytics 0.2       # axa y etichetată la 0.2
test                # afișează puncte/linii cu stilul default
replot              # redesenează
help plot with      # manual la prompt
save 'file.plot'    # creează un script pt plotul curent
```

3.1.3 Exercițiu

Realizați folosind `gnuplot` graficul din laboratorul 1. Indicați semnificațiile axelor și legenda. Salvați imaginea în format `png/svg`, și scriptul pentru restaurarea imaginilor.

```
# se rulează interactiv
plot 'out.tr' using 1:2 t "TCP" with lp, \
     'out.tr' using 1:3 t "UDP" with lines lw 3
set xlabel 'Time[s]'
set ylabel 'Bandwidth[Mbps]'
set grid ytics
set term png
```

```
set out 'lab02bw.png'
replot
save 'lab02bw.plot'
# vom putea reface graficul cu gnuplot ./lab02bw.plot
```

3.2 Introducere în awk

AWK (K vine de la Kernighan) este un interpretor mic, simplu, și rapid, spre deosebire de perl sau python. Nu putem face tot ce facem în perl/python, dar putem face foarte ușor multe taskuri de procesare de text. Are o sintaxă apropiată de C, dar preferă datele organizate pe coloane, ca foarte multe date în rețelistică: trace-uri de simulare, tcpdump, loguri, etc. Un mare avantaj este ca poate fi rulat direct de pe linia de comandă, fără a mai folosi un script separat - de multe ori apare într-un pipeline cu cat, sed, tr.

În cazul cel mai des întâlnit, se specifică *un program care este rulat succesiv pentru fiecare linie de intrare*:

```
cat trace.out | awk '{print $2}'
```

afișează coloana a doua a fiecărei linii. De exemplu, pentru acest fișier trace.out:

```
10 2    0.2
11 3 0.3
12 2 0.2

13 3 0.1
14 4 0.05
```

```
cat trace.out | awk '{print $1+$2, $2 $3, i++;}'
```

produce

```
12 20.2 0
14 30.3 1
14 20.2 2
0  3
16 30.1 4
18 40.05 5
```

Din acest exemplu se observă că:

- caracterul $ trebuie protejat de shell
- separatorul implicit este (tab—spațiu)+

- variabilele sunt inițializate la 0, și își păstrează valoarea de la o linie la alta
- câmpurile inexistente ale unei linii sunt șirul vid
- tipurile sunt slabe - int, float, string, din context
- spațiu este operator de concatenare pe stringuri

```
cat trace.out | awk 'NF==3 { s+=$3; n++}
               /1[2-3]/{print $0} END{print n,s/n}'
```

produce

```
12 2 0.2
13 3 0.1
5 0.17
```

Din acest exemplu se observă că:

- există variabile predefinite NF= number of fields(each line); NR=number of records; $0 = toată linia
- se pot rula mai multe programe per linie, dacă sunt activate de condiții logice/regex. Pentru o linie se execută TOATE programele care se pot activa.
- există secțiunea BEGIN care se rulează o singură dată înainte de input, și END la sfârșit
- sunt disponibile multe funcții de bibliotecă: printf, sqrt, substr, xor - vedeți man awk

3.2.1 Exercițiu

Pentru simularea din laboratorul precedent, plotați dimensiunea în pachete a cozii pe link-ul de ieșire (fișierul queue.tr) pentru a explora relația dintre lungimea cozii la bottleneck (link n2 - n3), RTT-ul perceput de TCP, și fracțiunea de debit obținută în concurență cu UDP.

3.3 ns-2 wireless

Explorăm sectiunea de wireless din tutorialul ns-2 al lui Marc Greiss [10]. Pentru toate simulările din acest laborator, se va folosi formatul nou de trace care este mai usor de analizat `$ns_ use-newtrace`

- Exemple de 2 linii din trace cu newtrace (liniile sunt pliate pentru a încăpea în pagină):

```
r -t 0.016905500 -Hs 1 -Hd -2 -Ni 1 -Nx 0.00\
 -Ny 75.00 -Nz 0.00 -Ne -1.000000 -Nl MAC\
 -Nw --- -Ma 0 -Md 1 -Ms 0 -Mt ACK

d -t 1.804824308 -Hs 2 -Hd 2 -Ni 2 -Nx 75.00\
-Ny 0.00 -Nz 0.00 -Ne -1.000000 -Nl MAC\
-Nw COL -Ma 13a -Md 2 -Ms 0 -Mt cbr -Is 0.0\
-Id 2.1 -It cbr -Il 1590 -If 0 -Ii 144\
-Iv 32 -Pn cbr -Pi 34 -Pf 0 -Po 0
```

- Fiecare linie descrie un eveniment de trimitere, primire, dirijare, sau dropare a unui pachet. Câmpurile cele mai importante dintr-o linie a fișierului trace sunt:

```
s: Send
r: Receive
d: Drop
f: Forward

-t          double         Time (* For Global Setting)
-Ni         int         Node ID
-Nx         double         Node X Coordinate
-Ny         double         Node Y Coordinate
-Nz         double         Node Z Coordinate
-Ne         double         Node Energy Level
-Nl         string        Network trace AGT, RTR, MAC
-Nw         string         Drop Reason
-Hs         int         Hop source node ID
-Hd         int         Hop destination Node ID, -1, -2
-Ma         hexadecimal        Duration
-Ms         hexadecimal        Source Ethernet Address
-Md         hexadecimal        Destination Eth Address
-Mt         hexadecimal        Ethernet Type
-P         string         Packet Type (arp/dsr/imep/tora
-Pn         string         Packet Type (cbr, tcp)
-Ps         sequence number (pentru tcp, coloana 47)

detalii la http://www.isi.edu}/nsnam/ns/doc/node186.html
```

3.3.1 Exercițiu

Modificați `simple-wireless.tcl` din Marc Greis secțiunea IX pentru

- a utiliza noul format de trace (cu `$ns_ use-newtrace`)
- a monitoriza evenimentele de la nivelele 2 și 4 (agent și MAC)
- a avea o coadă de doar 10 pachete în interfața wireless
- identificați în trace cadrele de tip CTS, ACK, ack, tcp.
- la ce moment începe transferul propriuzis (nodurile sunt suficient de aproape pentru a schimba cadre)?

```
cat simple.tr | grep  '^r.*AGT.*ack.*tcp' |
        awk "{print $3, $47}" | head}
```

- Desenați o diagramă cu transferul unui segment TCP între noduri, indicând tipurile cadrelor/pachetelor.
- calculați numărul de cadre de date(tcp)pierdute de fiecare nod la nivelul 2 - MAC

```
cat simple.tr | grep '^d' | grep MAC | grep 'tcp' |
    grep -v 'ack' | wc -l
```

- calculați numărul de cadre de date(tcp)pierdute de fiecare nod la nivelul 3 - IFQ

```
cat simple.tr | grep '^d' | grep IFQ | grep 'tcp' |
     grep -v 'ack' | wc -l
```

- justificați diferențele

Atenție, avem cadre ACK, tcp fără ack , tcp și ack. ACK sunt confirmări 802.11, ack sunt confirmări TCP. Se pierd 11 cadre TCP/date (linii cu tcp, dar fără ack) la nivelul 2, și 83 sunt aruncate din coadă. Dacă destinația nu mai răspunde în aer, 802.11 ajunge la maximum retries, apoi trece la următorul pachet. TCP însă nu trece mai departe după pierdere. Când destinația iese din zona de comunicare, TCP continuă să trimită deoarece are fereastră suficientă, și se pierde din coadă.

- comparați pierderile între pachetele tcp și ack `unack` se generează ca răspuns la un tcp. Dacă tcp sunt pierdute la transmisie (coadă sau aer), se generează mai puține ack
- comparați pierderile între cadrele ACK și RTS `ACK=0`pierderi, RTS=14 pierderi. Dacă nodurile sunt în apropiere, conversația RTS-CTS-Date-ACK se desfășoară cu bine, nu se pierd ACK-uri. La distanță mare, destinația nu răspunde la RTS, deci sunt pierdute, și nu se mai ajunge la ACK.

Capitolul 4

Laboratorul 3

4.1 Capacitatea WiFi

Scopul acestui laborator este de a calcula capacitatea unui canal 802.11 pentru diverse standarde, în condiții optime. Vom face o analiză teoretică folosind temporizările, randomizările, și dimensiunile antetelor din standard, versus estimarea în simulator. Cazurile de interes sunt:

- 802.11b, 802.11a, 802.11g
- cu/fără RTS/CTS
- UDP, TCP
- clienți multipli la un AP downlink, uplink

4.1.1 Capacitatea ideală teoretică

- calculați durata unei tranzacții atomice de transmitere a unui cadru: DIFS, arbitraj, PHY header, MAC header, IP/UDP header, UDP Payload de x octeți, FCS, SIFS, ACK
  - atenție: MAC, IP,UDP,Payload, CRC și corp ACK se transmit prin aer la rata MCS, măsurată în bps
- folosiți duratele temporizărilor specificate în standard, și dimensiunile antetelor PHY din schemele de mai sus
  - 11b: slot=20, SIFS=10, DIFS=50
  - 11g: slot=9, SIFS=10, DIFS=28
- determinați durata în microsecunde a unei tranzacții (separat 11b și 11g)

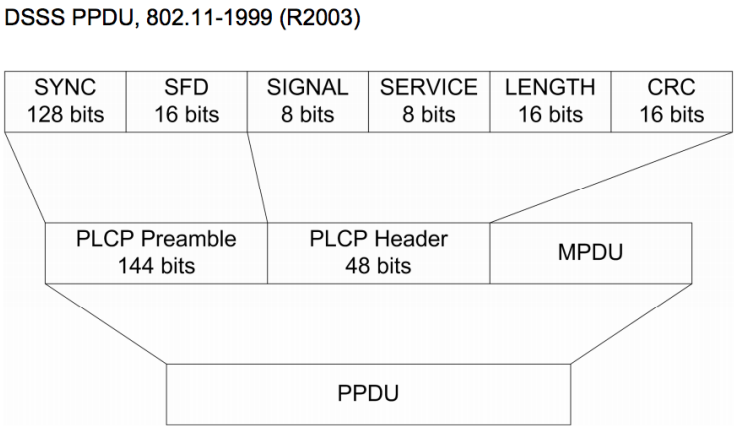


Figura 4.1: 802.11b Antet PHY de 192biți trimis la MCS=1Mbps.

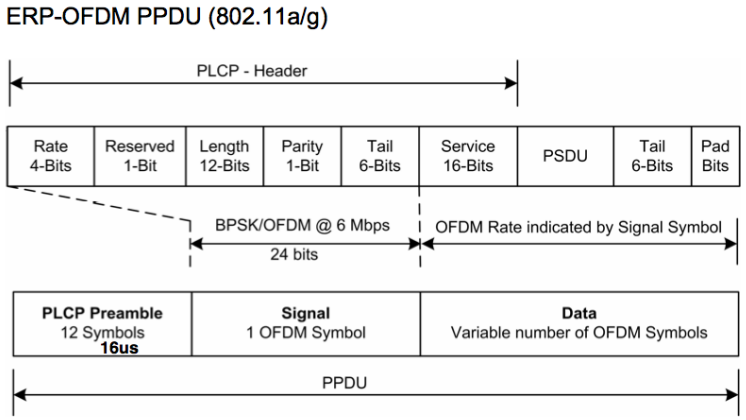


Figura 4.2: 802.11g Antet PHY de 16us + 24biți trimis la MCS=6Mbps

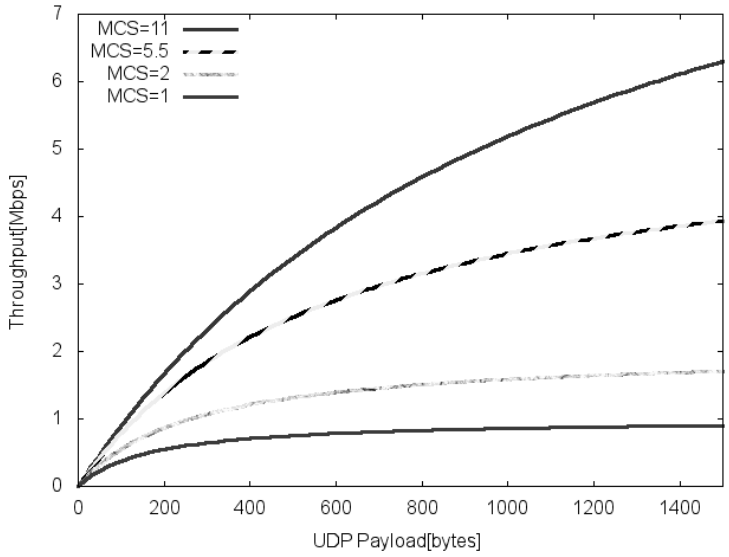


Figura 4.3: 4 curbe pentru 1Mbps, 2Mbps, 5.5Mbps, 11Mbps

- Durata pentru 802.11b [1]

- scrieți funcția Throughput_11b(x) unde x este payload UDP, iar MCS(o constantă) ia valorile din standard 11b=1, 2, 5.5, 11 [2]
- scrieți funcția Throughput_11g(x) unde x este payload UDP, iar MCS ia valorile din standard 11g=6, 12, 36, 54
- deduceți modul în care debitul obținut în Mbps depinde de MCS și de dimensiunea UDP payload
  - `gnuplot> plot MCS=1, Throughput(x) w l t '1Mbps', MCS=2, Throughput(x) w l t '2Mbps', MCS=5.5, Throughput(x) ...`
- realizați grafice pentru 802.11b (x:payload UDP; y:Throughput[Mbps])
- realizați grafice pentru 802.11g (x:payload UDP; y:Throughput[Mbps])

4.1.2 Capacitatea ideală simulare

- Pentru un singur client, se vor repeta curbele de mai sus folosind ns-2. Scriptul `infra.tcl` configurează la (0,0) un AP și n-1 noduri

[1] 50 + 310 + 192 + (24 + 20 + 8 + x + 4)*8/MCS + 10 + 192 + 14*8/MCS

[2] `gnuplot> Throughput_11b(x) = x*8 /(764 + (70 + x)*8/MCS)`

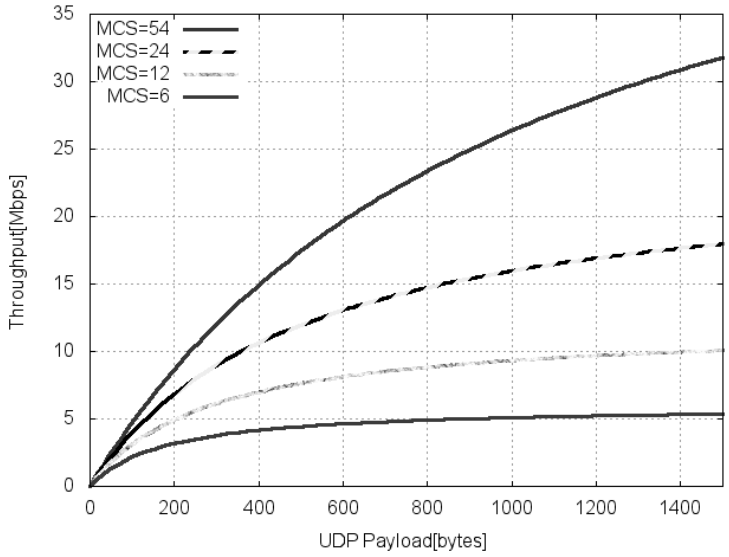


Figura 4.4: 4 curbe pentru 6Mbps, 12Mbps, 24Mbps, 54Mbps

plasate în vecinătatea sa. Traficul este generat de la AP către fiecare nod.

- pentru capacitatea ideală, folosim un AP, un client,trafic de tip UDP
- necesită parametrii `-run_tcp 0 -nn 2 -packetSize 100 -sendingRate 1Mbps`
- parametrul '-sendingRate' corespunde traficului trimis de aplicație în socketul UDP

- în script **trebuie** decomentată numai porțiunea corespunzătoare standardului (11b, 11g,sau 11a)
- în script **trebuie** indicată MCS – se numește `dataRate_`
- pe linia de comandă **trebuie** dați parametrii relevanți pentru dimensiunea pachetului și rata dorită de UDP
- repetați graficele precedente/teoretice folosind simularea. Puncte de evaluare pentru packet size: 20, 50, 100, 500, 1000, 1500
  - De ce? VoIP 20-300; DNS, TCP 500; Ethernet=1500; 802.11 Beacon=380
- repetați experimentele cu RTS/CTS activat. Ce impact are asupra pachetelor mari? Dar a celor mici?

Capitolul 5

Laboratorul 4

5.1 Modelul Two Ray Ground

Acest model este idealizat și este util analizării unor situații în care se dorește explorarea comportării MAC-ului în condiții ideale de propagare. La marginea zonei de propagare se face o tranzitie brusca de la livrare 100% la 0%, așa cum se observă în figura 5.1

Cercurile de comunicare și de CS sunt definite în putere(W), care corespunde unei distanțe(m).

```
set opt(prop) Propagation/TwoRayGround;
Phy/WirelessPhy set CSThresh_ 1.55924e-11   ; #550m - exact
Phy/WirelessPhy set RXThresh_ 3.65262e-10   ; #250m - exact
Phy/WirelessPhy set CPThresh_ 10.0          ; #captura, în dB
```

În acest exemplu la 251m nu se primește nici un cadru, deși la 250m se primesc toate. Două noduri aflate sub 550m își cedează reciproc accesul când este cazul, adică nu-și sunt terminale ascunse unul altuia. Captura este atunci când într-o coliziune se poate recupera cadrul mai puternic, dacă acesta e cu 10dB mai puternic.

5.2 Modelul Shadowing

Modelul shadowing este un model probabilistic, adică nu mai există o limită clară la care se face tranzitia de la livrare 0% la livrare 100%, așa cum se observă în figura 5.2

```
set prop Propagation/Shadowing;
Phy/WirelessPhy set CSThresh_ 1.55924e-11    ; #550m
Phy/WirelessPhy set RXThresh_ 3.65262e-10    ; #50m
Phy/WirelessPhy set CPThresh_ 10.0           ; #captura
```

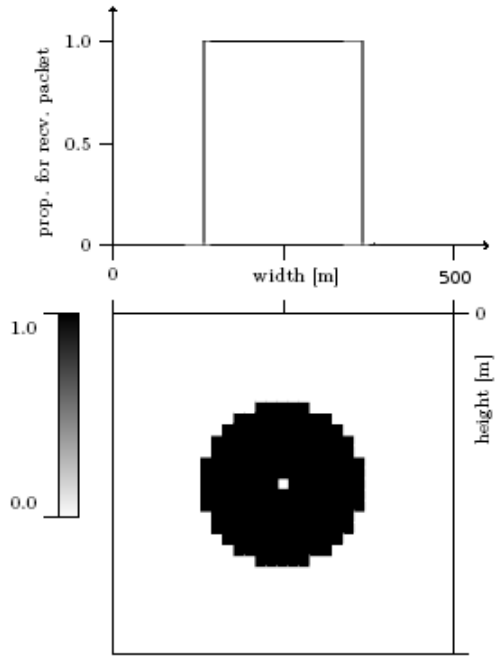


Figura 5.1: Modelul de propagage 'two ray ground'.

```
# parametri specifici modelului
$prop set pathlossExp_ 2.7
$prop set std_db_ 2.2
$prop set dist0_ 1.0
```

5.2.1 Setup simulare

Se utilizează scriptul `shadow2.tcl` care folosește 802.11b, MCS=2Mbps cu modelul shadowing și definește următoarea topologie: `S1--dist--D1---10000m---S2--dist--D2`. Scriptul acceptă parametrii -dist pentru a seta distanța sursă-destinație, și -tries pentru a seta numărul maxim de încercări. Cele două perechi nu se influențează reciproc din cauza distanței mari, iar S1 → D1 folosește UDP, în timp ce S2 → D2 folosește TCP. Practic se rulează două experimente pentru aceeași distanță de comunicare, si acelasi regim de retransmisii. Scriptul afișează debitul în bps obținut de cele 2 fluxuri, dar generează și fișierul shadow2.tr care poate fi analizat pentru a contoriza pachetele în diverse ipostaze.

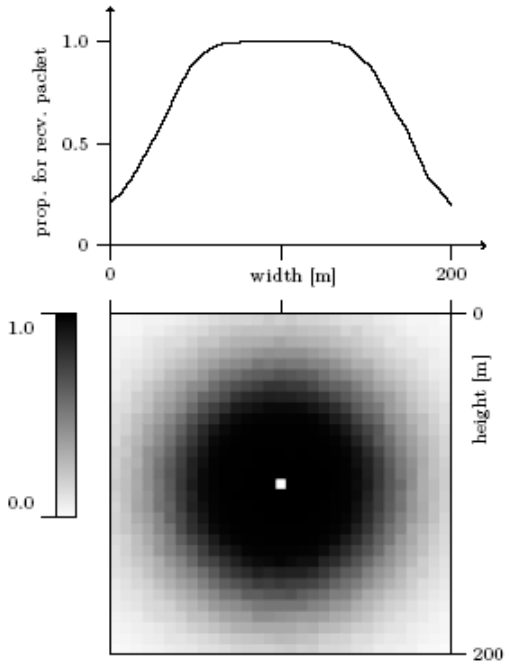


Figura 5.2: Modelul de propagare 'shadowing'.

5.2.2 Plan de organizare a datelor intermediare

1. citiți toate cerințele din secțiunea **Taskuri** de mai jos înainte de a decide un plan de măsurători. Strategia optimă este de a extrage mai mulți parametri în urma unei simulări pentru a putea răspunde mai multor cerințe. Exemple: apariții unice ale unui nr de secvență cbr la emisie/recepție MAC, total cadre cu cbr emise/primite la MAC, thruput UDP, thruput TCP, etc.

2. decideți structura fișierului text care va conține datele din care se pot produce graficele cerute.

- ce avem pe axele x, y?
- ce reprezintă fiecare coloană?
- ce parametri se pot obține ca o funcție de coloane? Exemplu $probabilitate_livrare = primite/trimise$
- ce grafice au axe care nu corespund direct unoar coloane din fișierul generat? cum se pot obține acele axe?

- prelucrați ieșirile simulatorului (stdout sau fișierul de trace) pentru a obține un fișier **text** de date intermediare cu semnificații clare pentru linii și coloane. Exemplu: fiecare linie e pentru o distanță, fiecare coloană e un counter de cadre sau un debit în Mbps
- plotați interactiv datele intermediare. Unele grafice sunt o simplă dependență a două coloane (de exemplu Thruput(distanță. Pentru combinații de coloane, gnuplot acceptă sintaxa `> plot "date" using 1:($3*100.0/$5) with ...` adică axa y este raportul coloanelor 3 și 5 în procente, iar axa x este coloana 1. Nu vă preocupați de partea estetică a graficelor (etichete, legendă, culori, etc).

5.2.3 Exerciții

1. analiză: dacă probabilitatea de livrare la nivel fizic este p, cât este la MAC, după r încercări? $PDR(p, r) = 1 - (1 - p)^r$ de ce?
2. plotați packet delivery ratio la nivel fizic (tries=1) și la nivel MAC (tries=4,10) pentru distanțele 50m-250m. Comparați cu modelul de propagare care prezintă spațial probabilitatea de recepție.
   - se analizează fișierul trace doar pentru traficul cbr
   - tries=1 înseamnă că fiecare cadru este încercat o singură dată
   - pentru tries¿1, trebuie numărate cadrele unice emise și primite la nivelul MAC. Atenție, această statistică nu poate fi obținută la UDP
   - în trace, la coloana 47 se află câmpul cbr sequence number, unic per pachet original emis de udp/cbr.
   - exemplu de calcul al numărului de cadre unice emise

     ```
     unqsent=$(cat shadow2.tr  | grep MAC | grep '^s' |
        grep cbr | awk '$3 <= 25.0 {print $47}' |
        uniq -c| wc -l)
     ```

   - comparați în câteva puncte cu formula din analiză
   - se poate spune că reîncercările cresc distanța de comunicare?
3. plotați capacitatea obținută de UDP pentru tries=1, 4, 10 la distanțe 50-250m. Justificați relația cu graficul de la punctul precedent.
   - Opțional: plotați numărul de cadre unice emise în aer pentru tries=1,4, 10
   - Opțional: plotați interdeparture times pentru cadrele emise de MAC(toate, inclusiv cele care nu vor fi primite)

- Opțional: plotați interarrival times pentru cadrele primite de MAC(implicit doar cele primite cu succes)

4. plotați capacitatea obținută de UDP pentru tries=1, 4, 10 la distanțe 50-250m. Justificați relația cu graficul de la punctul precedent.
   - Opțional: plotați numărul de cadre unice emise în aer pentru tries=1,4, 10
   - Opțional: plotați interdeparture times pentru cadrele emise de MAC(toate, inclusiv cele care nu vor fi primite)
   - Opțional: plotați interarrival times pentru cadrele primite de MAC(implicit doar cele primite cu succes)
5. plotați capacitatea obținută de TCP pentru tries=1, 4, 10 la distanțe 50-250m. Justificați relația cu graficele precedente.
   - de ce la tries=1 (nu se reîncearcă), TCP nu atinge capacitatea maximă chiar când canalul este perfect, de exemplu la 10m?
   - Analizați pachetele pierdute în trace, și justificați folosind timpii de emisie ai pachetelor.
   - ctivați RTS/CTS pentru acest caz(tries=1, loss=0). De ce nu se îmbunătățește situația TCP-ului?
6. plotați capacitatea obținută de TCP pentru tries=4, 10 la distanțe 50-250. Explicați diferențele față de comportarea UDP în același setup
7. dorim exprimarea capacității UDP și TCP ca funcție de rata de livrare la nivel fizic
   - axa x: PDR la nivel fizic se poate obține cu UDP cu tries=1 scalat la capacitatea maximă (1.7Mbps), sau ca raport între numărul de cadre recepționate si emise la nivel MAC.
   - axa y: debit în Mbps obținut de TCP cu tries=1, 4, 10.
   - același grafic pentru UDP
   - tries=1 este best case pentru UDP și worst case pentru TCP?
   - tries=1 de ce TCP nu obține debit maxim, chiar în condiții optime (livrare 100%)?
   - UDP când crește numărul de încercări?

Capitolul 6

Laboratorul 5

6.1 Capacitatea unui grup de mobile sub un Access Point

Se pornește de la același script `infra.tcl` folosit în lucrările 3 și 4. Varianta nemodificată trimite trafic downlink către `nn-1` clienți. Vom evalua capacitatea în condiții noi: **client unic vs. multipli, uplink vs downlink.** Topologia este una specifică modului infrastructură, cu un AP și mai mulți clienți asociați, dar cadrele de beacon, autentificare, și asociere sunt omise pentru simplitate.

- realizați grafice doar pentru 11b, packetSize 1460, MCS=11Mbps
- se variază doar numărul de clienți, și se plotează debitul cumulativ pentru toate fluxurile
- pe baza scriptului original, creați câte un script pentru fiecare experiment mai jos.
- comentați și explicați fiecare curbă din fiecare grafic

6.1.1 Topologiile de simulat

Trebuie modificat scriptul pentru a realiza următoarele topologii:

1. *downlink* - sursa: AP, destinații: 1..30 clienți (scriptul original)
2. *uplink* - surse: 1..30 clienți, destinație: AP
3. *mixed* - surse: 1..15, destinații: 16..30
4. *1up* - sursă: client 1, destinații: clienți 2..30
5. *1down* - surse: clienți 2..30, destinație: client 1

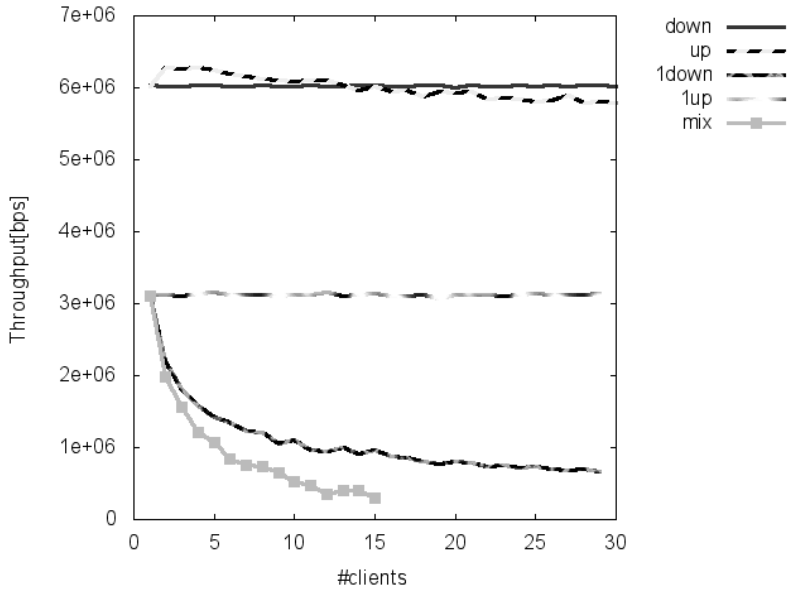


Figura 6.1: 5 topologii pentru UDP

6.1.2 Exerciții

1. Plotați pe același grafic cele 5 curbe pentru UDP, 802.11b
2. plotați pe același grafic cele 5 curbe pentru TCP -
3. se repetă experimentele cu RTS/CTS activat -
4. (acest caz e mai simplu după ce interpretați și comentați rezultatele precedente) Cum e posibil ca *mixed* să obțină rezultate bune și la populații mari? Care este optimul pe care-l poate obține?

Comentați și explicați fiecare rezultat, comparați cu rezultatele de la laboratorul 3, "Client unic".

- cum explicați diferențele *uplink* - *downlink*? [1]
- cum explicați diferenta *uplink* - *1up*? [2]

[1] pentru nn=2 e același lucru. Pentru mai mulți emițători, crește probabilitatea de coliziune

[2] la uplink 'vorbitorii' sunt simetrici ca rol. La 1up tot traficul trece prin AP, deci capacitatea se înjumătățește. AP este deasemenea 'vorbitor', dar primește parte egală cu ceilalți, deși trebuie să transporte pentru alții

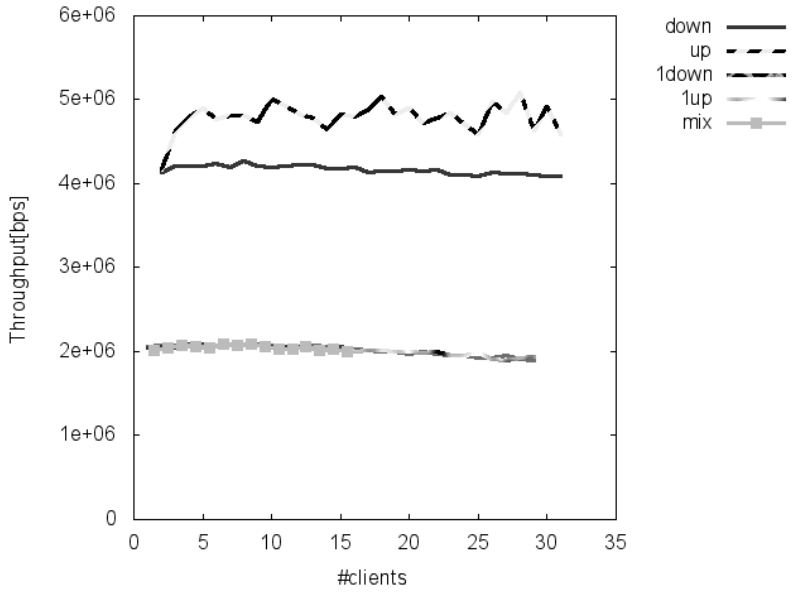


Figura 6.2: 5 topologii pentru TCP

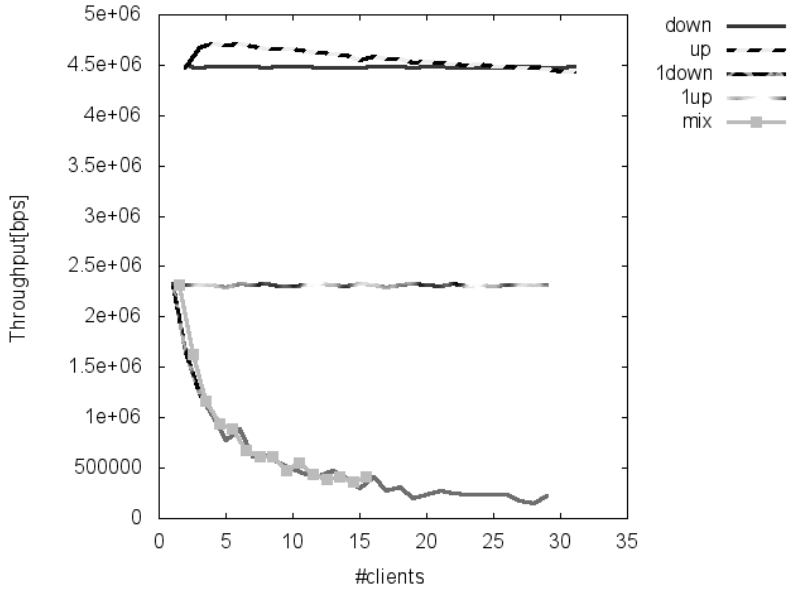


Figura 6.3: 5 topologii pentru UDP, cu RTS

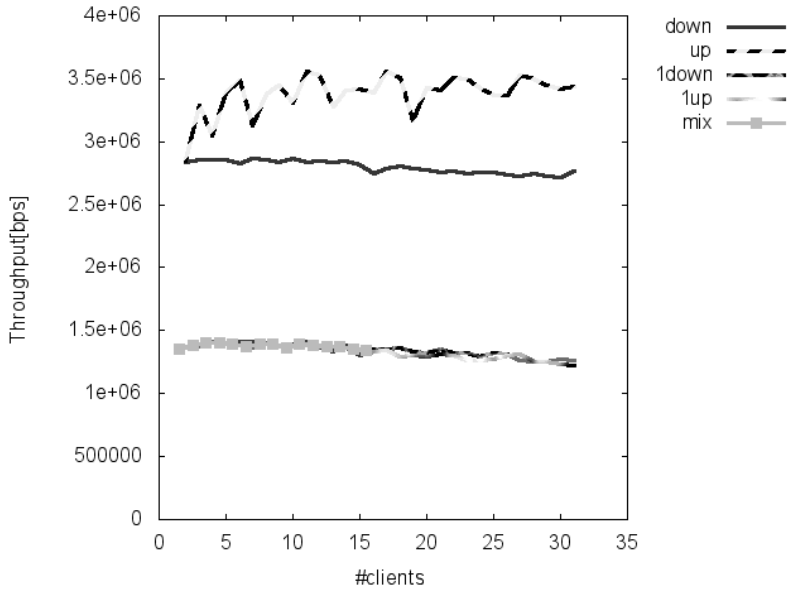


Figura 6.4: 5 topologii pentru TCP, RTS activat

- cum explicați comportarea în cazul *mixed*? Comparați cu *1down*? [3]
- cum explicați diferențele TCP - UDP? [4]
- cum explicați diferențele RTS - simplu?
- explicați diferențele *up* vs *down* [5]
- explicați diferențele *1down* vs *mix* [6]
- care este valoarea până la care coboară cazul UDP*/mixed*, și cum poate fi îmbunătățită? HINT: de ce la TCP *mixed* este similar cu *1up*? [7]
- modificați scriptul pentru a rezolva punctul 4

[3] deasemenea crește numărul de vorbitori. Curba este diferită, deoarece nn stații semnifică nn/2 vorbitori

[4] TCP are ack-uri la nivelul 4; pe uplink TCP nu este agresiv, nu tolerează dropuri în coada AP-ului, deci reduce fereastra încât migrează către un optim global: 50% din aer pentru clienți și 50% pentru AP

[5] coliziunile sunt mai puțin costisitoare - 1 RTS

[6] am plotat mix după numărul de 'vorbitori'; coliziunile sunt mai puțin costisitoare

[7] UDP trimite prea mult, dacă l-am forța să trimită mai puțin de la clienți, atunci AP-ul ar avea loc mai mult.

Capitolul 7

Laboratoarele 6 și 7

7.1 802.11 Contention Window

Scopul acestui exerciţiu este de a investiga impactul dimensiunii ferestrei de arbitrare (contention window) asupra performanţei protocolului IEEE 802.11. MAC-ul IEEE 802.11 prevede ca toate nodurile să aleagă un timp de așteptare aleator cuprins între zero şi CW (fereastra de arbitraj), şi aşteaptă numărul ales de sloturi înainte de a încerca să acceseze canalul. Iniţial, CW este setat la CWMin (minim fereastra susţinute de mărime). Cu toate acestea, atunci când există o coliziune, dimensiunea fereastrei este dublată, până la o valoare maximă: CWMax. Aceasta tehnică de randomizare şi scalare a ferestrei este folosită pentru a reduce coliziunile. În studiul nostru vom lua în considerare o variantă de 802.11 pt cazul în care mărimea ferestrei este fixă, şi anume CWMin = CWmax = CW. Deși nu se scalează fereastra, se folosește randomizarea.

Vom avea nevoie de o topologie în care pentru a studia efectul dimensiunii fereastrei. Pentru aceasta, folosim o reţea de un hop atunci când toate nodurile sunt plasate într-un grup compact. În particular, vom considera o arie de 150x10m comună pentru toate nodurile. Fiecare sursă participă la o conversaţie, o destinaţie poate participa la mai multe.

7.1.1 Instrucţiuni

- Descărcați scriptul tcl `cw.tcl`. În script, veţi găsi următoarea linie care stabilește valorile CWMin şi CWMax la valoarea dorită:

```
$val(mac) set CWMin_ 31
```

iar rata CBR rezultă din intervalul de generare

```
$cbr_($i) set interval_ 0.05
```

Script-ul accepta patru argumente în linia de comandă:

- `ns` numărul de surse
- `nr` numărul de destinații
- `cwmin` pentru a indica limita minima a fereastrei de arbitraj/contenție
- `cwmax` pentru a indica limita maxima a fereastrei de arbitraj/contenție

De exemplu, dacă ar fi să rulaţi un experiment cu 4 noduri în zona de 150m x 10m de reţea, cu fereastra de 63, vi se va cere să rulaţi următoarea comandă:

```
ns cwsim.tcl -ns 2 -nr 2 -cwmin 63 -cwmax 63
```

Pentru 802.11b standard, cu fereastra adaptiva:

```
ns cwsim.tcl -ns 2 -nr 2 -cwmin 15 -cwmax 1023
```

7.1.2 Exerciții

- Rulaţi scriptul pentru ns = 4, 6, 7, 20, 40 şi pentru CWMin = CWMax să ia următorul set de valori: 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047, 4095. Obţineţi şi plotați ca funcții de dimensiunea CW:
- total pachete livrate la destinație (fig. 7.1)
- rata CBR livrată la destinație [Mbps] (fig. 7.1)
- probabilitatea de livrare (PDR) la nivel agent (fig. 7.2)
- probabilitatea de livrare (PDR) la nivel MAC (fig. 7.2)
- numărul de pachete de date emise de MAC
- numărul de pachete de date emise de agent
- numărul de coliziuni pe secundă [pps]
- numărul de retransmisiuni per cadru
- Pentru a calcula PDR, va trebui să calculați numărul de pachete trimise şi numărul de pachete primite. În acest exemplu, pentru a obţine numărul de pachete trimise, puteţi utiliza următoarea comandă:

```
grep AGT cwsim.tr | grep _^s |grep cbr | wc -l
```

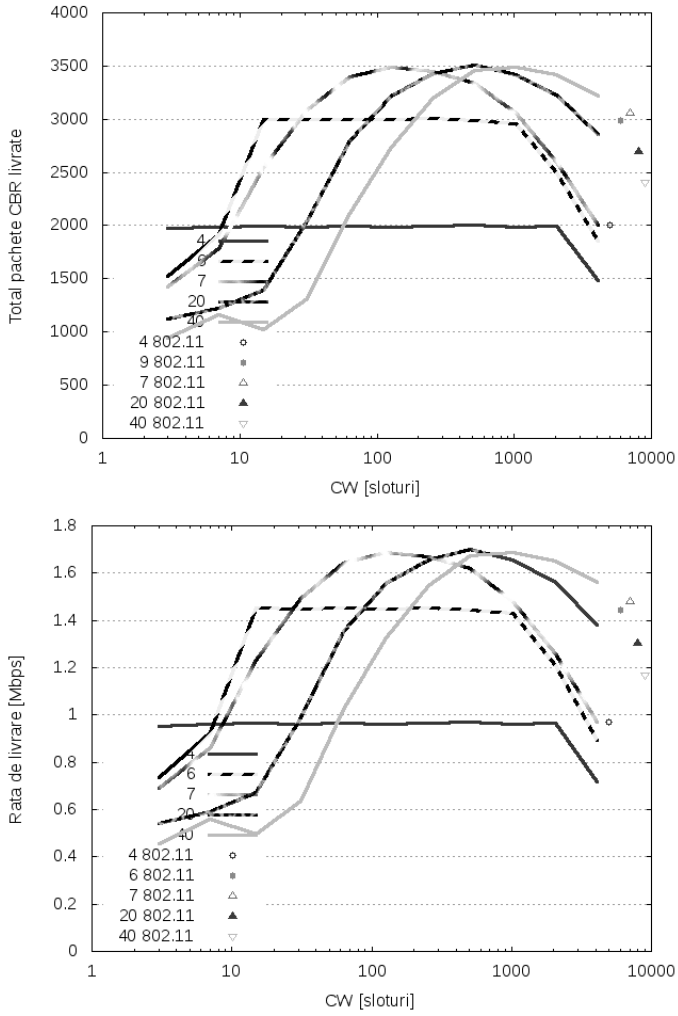


Figura 7.1: sus: pachete livrate; jus: rata livrată

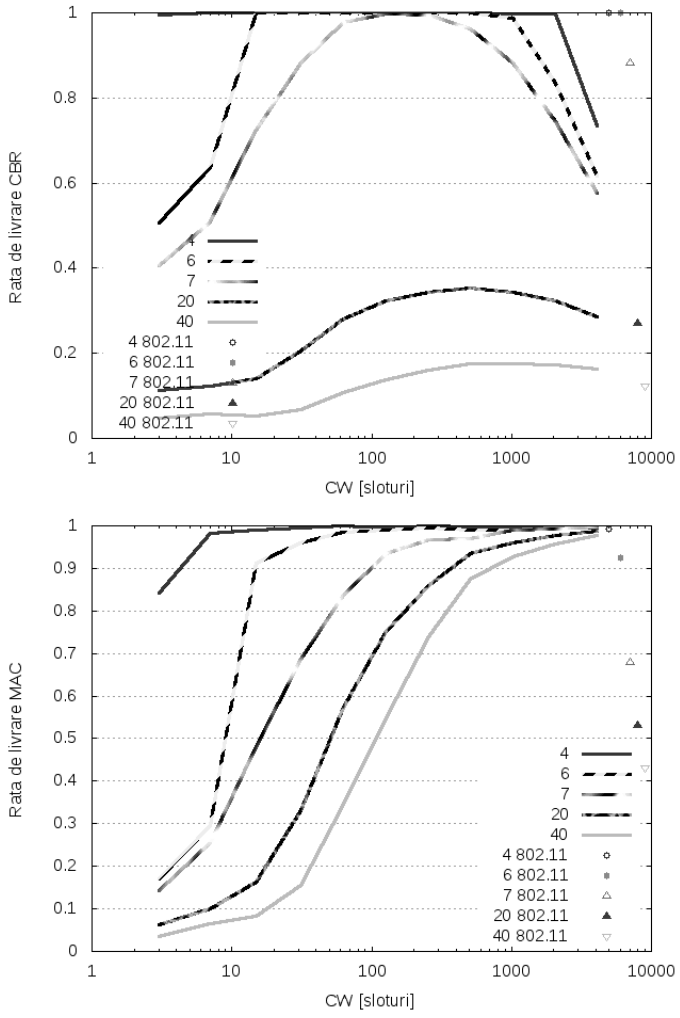


Figura 7.2: sus: PDR la agentul UDP; jos: PDR la MAC

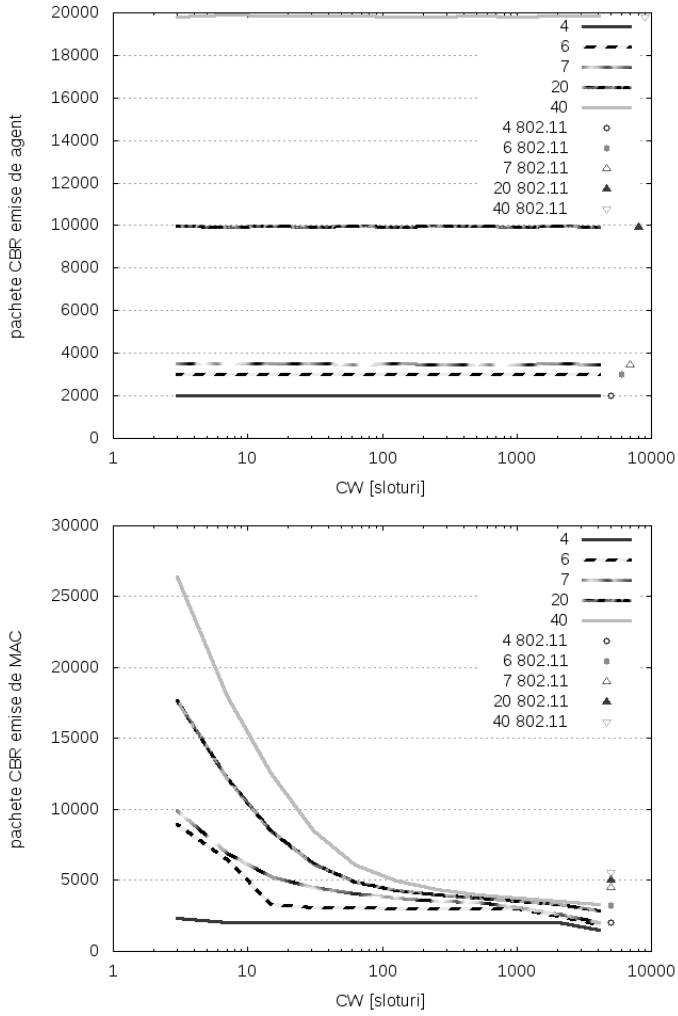


Figura 7.3: sus: trimise la agentul UDP; jos: trimise la MAC

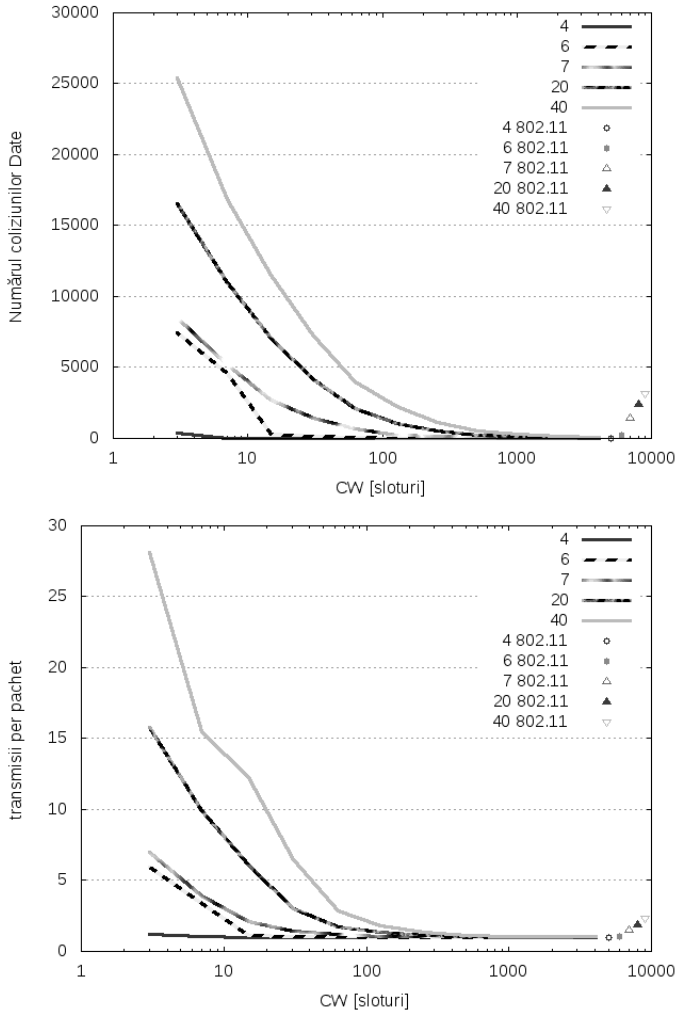


Figura 7.4: sus: coliziuni; jos: reîncercări

- Numărul raportat de mai sus este numărul de pachete trimise de către toate nodurile. Pentru a obţine numărul de pachete primite, aveţi posibilitatea să utilizaţi o comandă similară:

```
grep AGT cwsim.tr | grep ^r |grep cbr | wc -l
```

- Numărul raportat este numărul de pachete primite (ar trebui să fie cel mult egal cu numărul de pachete trimise). Raportul de Pachete [primite] / [pachetele trimise] este PDR.

```
grep COL cwsim.tr | grep ^d |grep cbr | wc -l
```

  Cu comanda de mai sus obțineți numărul global de pachete care au intrat în coliziune.

```
grep MAC cwsim.tr | grep ^s |grep cbr | wc -l
```

  Cu comanda de mai sus obțineți numărul de pachete de date emise de MAC.

- Pentru a înţelege mai multe despre formatul fișierului trace, consultați pagina din manualul ns-2 http://www.isi.edu/nsnam/ns/doc/node186.html.

7.1.3 Analiză

Această temă vă permite să corelați mărimea CW şi dimensiunea reţelei / densitatea. Tendinţa poate sau nu să fie clară din cauza unor factori cum ar fi interferenţe cu creşterea densităţii etc. Încercaţi să răspundeţi la următoarele întrebări cu privire la graficele obţinute mai sus:

- care este capacitatea rețelei în cadre/secundă? Dar în bps? Pentru a valida acest rezultat examinați în script rata folosită la nivelul fizic, parametrii CBR, și dimensiunea pachetelor. [1]

- ce tendinţe ați observat? Există o dimensiune optimă CW pentru fiecare populaţie de reţea? Ce relație există între dimensiunea optimă și populaţia reţelei? [2]

[1] Se rulează cu nr = ns = 1, CW=15-1023, și cu o rată suficient de mare... rezultă 3616 pachete de 1460 primite în 25 secunde, adică 1.69Mbps pentru standardul 802.11b/2Mbps

[2] CW nu ar putea funcționa cu un CW fix, CW optim depinde de populație, trafic

- puteți prezice ce se va întâmpla dacă încercaţi să rulaţi acest script pentru ns = nr = 50? Explicaţi.
- comparati performantele obtinute cu ce ar obtine 802.11 standard.
- rata de livrare la nivel UDP nu atinge mereu 100%. Cum este totuși posibilă ocuparea capacității maxime a aerului?
- numărul de cadre emise de MAC este mai mare decât numărul de pachete generate de agentul UDP. De ce?
- probabilitatea de livrare la MAC este șansa unui pachet de a supraviețui în aer. Când aceasta este 1, suntem în situația ideală
- de ce rata livrării la agentul UDP se comportă radical diferit pentru populația 4 (`ns=nr=4`) față de celelalte populații?
- care este numărul de încercări per pachet obținut de standardul 802.11? Comentați.
- Repetați experimentele activând RTS/CTS
- ce s-a întâmplat cu coliziunile? Comentați.
- de ce pachete CBR emise de MAC arată similar cu cele de capacitate? [3]
- de ce numărul de rtransmisii date este 0?
- repetați experimentele pentru pachete de 212 octeți (fără RTS). Comentați.
- verificați rezultatul cu cel de capacitate din laboratorul 4
- de ce numărul de cadre emise în aer este mai mare decât la pachete mari?
- cât durează un cadru de date?

[3] odată capturat aerul, pachetul reușește mereu

Capitolul 8

Laboratorul 8

8.1 Echitate(Fairness)

În calculatoare și comunicații sunt folosite diverse metrici ale echității pentru a determina împărțirea "echitabilă" a resurselor. Am folosit ghilimelele deoarece există mai multe modele conceptuale de echitate, cu alte cuvinte, dreptatea nu este numai una! Problema echității se modulează diferit în funcție de următoarele aspecte:

- utilizatorii cer părți egale, diferite, sau maximul disponibil
- utilizatorii pot utiliza efectiv părți egale sau diferite (e.g: wireless aproape/departe de BS)
- dependența între resursa primită și QoE nu e liniară (e.g: MPEG frames)
- eficiența globală nu este sacrificată prea mult pentru o echitate dorită (e.g: comunism)
- echitatea se obține doar pe termen lung (e.g: WiFi)

8.2 Jain's Fairness Index

În cazul în care toate cererile sunt egale, iar x_i este cantitatea obținută de participantul i:

$Jain(x_i) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2}$ $Jain(x_i) \in [0, 1]$

Proprietăți:

- x_i egale => $Jain(x_i) = 1$ echitate perfectă
- $x_1..x_k$ egale și $x_{k+1}..x_n = 0$ => $Jain(x_i) = \frac{k}{n}$

- x_1 ia totul, iar ceilalți nimic => $Jain(x_i) = \frac{1}{n}$ - inechitatea cea mai gravă
- independentă de populație
- independentă de dimensiunea x_i
- continuă în [0..1]

8.3 Echitatea ϵ

- capturează inechitatea cea mai gravă
- $\varepsilon(x) = \frac{\min_i x_i}{\max_i x_i}$
- 0 = starving, 1 = echitate perfectă

Descărcați scriptul `cw-fair.tcl` care folosește parametrii: -rlen -cwmin -cwmax. În script este definită durata simulării simtime. Topologia folosită este cu un AP în mijloc, și rlen-1 stații dispuse circular la distanță egală care transmit UDP către AP.

- pentru 10 noduri, estimați rulând manual echitatea cu ferestre CW fixe de 7 vs 4095. Comentați.
- comparați echitatea pe termen scurt (1s), mediu(5s), lung (50s)
- pentru ferestre fixe CW = 7, 31, 511 realizați trei grafice care indică echitatea în funcție de numărul de clienți 2..20. Fiecare grafic conține 3 curbe pentru duratele pe care se face medierea(1s, 5s, 50s)
- cum explicați tendințele de creștere/scădere a echității cu: numărul de clienți, dimensiunea ferestrei, scara de timp considerată? [1]
- Pentru 802.11 standard, calculați echitatea pe termen scurt (5s) și pe termen lung (50s). Realizați grafice care să ilustreze variația echității cu numărul de clienți.
- cum este echitatea față de cazurile cu fereastră fixă? De ce? [2]

[1] Echitatea este determinată (și) de coliziuni. Populație: mai mulți vorbitori înseamnă mai multe șanse de coliziune; Fereastra prea mică duce la mai multe coliziuni; Timp: pe termen lung, toți participanții au relativ aceleași șanse de a obține mediul, sau de a intra în coliziune.

[2] în 802.11, pe termen scurt "Rich get richer, poor get poorer"

- cum se poate îmbunătăți echitatea pentru configurația dată? (modificați, rulați, plotați) [3]
- Utilizarea RTS/CTS duce la creșterea echității? De ce?
- Ce se schimbă atunci când în loc de N fluxuri upstream, avem de exemplu 2 fluxuri downstream și N-2 upstream? [4]

8.4 Max-min Fairness (facultativ)

Pentru fluxurile care au cereri diferite, echitatea Max-min alocă gradual pentru fiecare conexiune până este satisfăcută cea mai mică, apoi se continuă cu cele rămase.

- Modificați rata pachetelor cerute de unele stații, și determinați dacă accesul la aer 802.11 este echitabil în sens Max-min.

[3] Hint: renunțăm la capacitate pt echitate...

[4] WiFi produce o oarecare echitate între vorbitori, deci cele 2 fluxuri downstream pentru care AP este vorbitor vor primi împreună cât un flux upstream

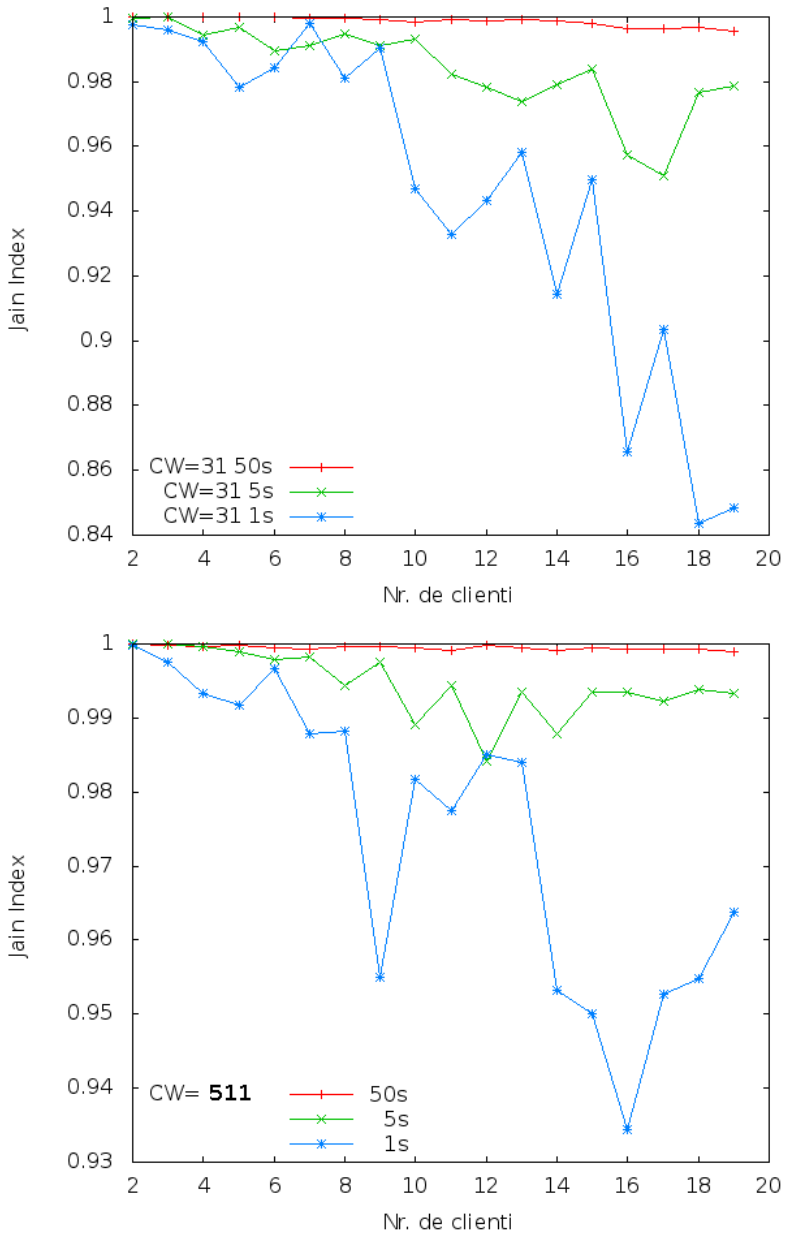


Figura 8.1: Echitatea Jain pentru CW = 31 (stânga), 511(dreapta)

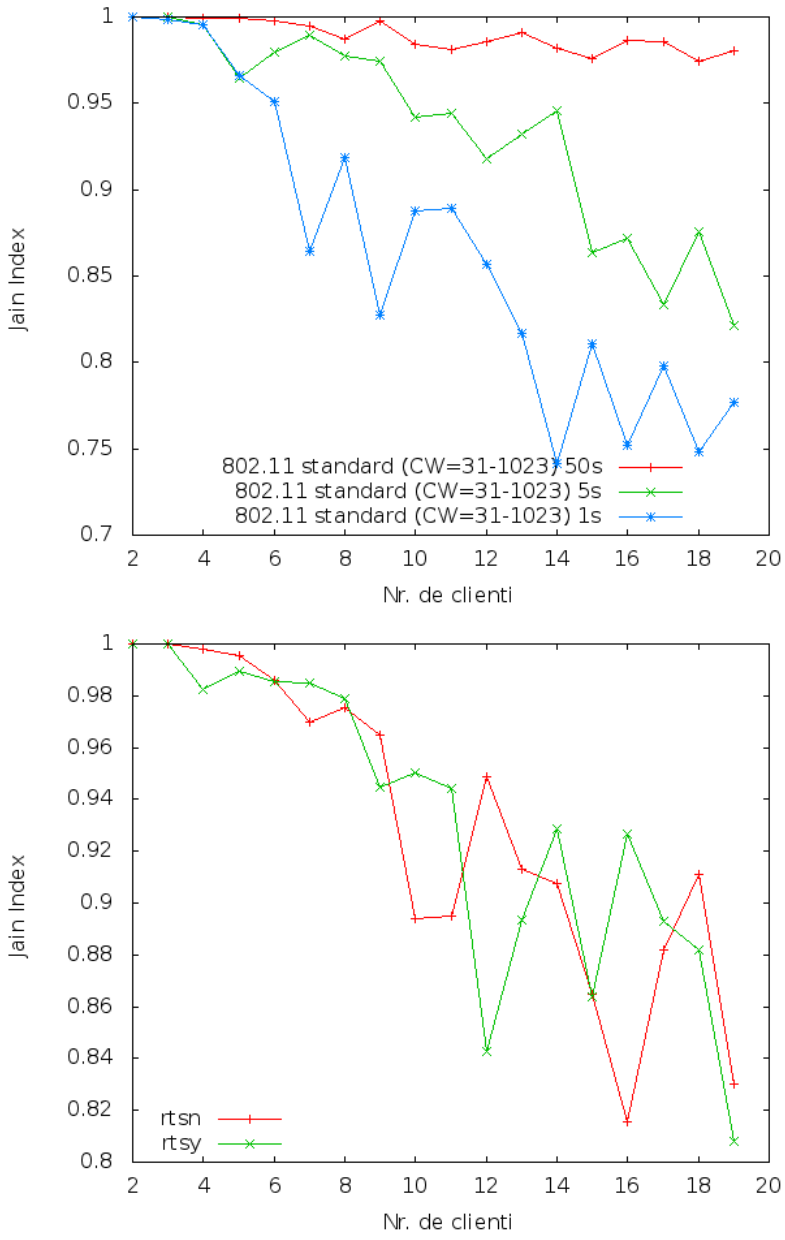


Figura 8.2: stânga: Echitatea Jain pentru 802.11(stânga) ; cu RTS(dreapta)

Capitolul 9

Laboratorul 9

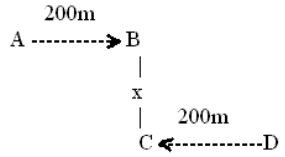
9.1 Carrier Sense fizic și virtual

Scopul acestui laborator este de a înțelege conceptul de Carrier-Sensing în 802.11 și de a compara mecanismele fizic și virtual de a testa prezența purtătoarei.

CS la nivel fizic este folosit atunci când un nod care vrea să transmită mai întâi evaluează starea canalului. Dacă energia detectată este peste un anumit prag, numit 'Carrier sense threshold', canalul este declarat ocupat, iar nodul trebuie să aștepte. Altfel, canalul este considerat liber, și nodul poate începe transmisia.

În contrast, CS virtual folosește indicațiile NAV (Network Allocation Vector) pentru a estima cât timp este mediul ocupat. În principiu, o stație recepționează toate cadrele, inclusiv cele care nu îi sunt adresate explicit. Fiecare cadru conține în câmpul 'Duration' o estimare a duratei conversației din care face parte. Un mod explicit de rezervare a mediului se face prin intermefdiul pachetelor RTS/CTS. Emițătorul trimite RTS, iar receptorul răspunde cu CTS. În pașii 3 și 4 se transmit datele și confirmarea ACK în modul obișnuit. Diferența este acum că nodurile vecine au ocazia să estimeze durata conversației în 4 pași prin citirea câmpului 'Durata' fie din RTS, fie din CTS. În ns2 se folosește următorul **pseudocod** pentru recepția cadrelor de la distanța d în condițiile în care mai poate apărea un cadru de la distanța d1:

```
recepție(d)
{
   if(alt_cadru(d1) && (d1 < CS)){
       if(P(d) - P(d1) >= 10dB)
           captură()
       else
```




```
            coliziune()
    } else {
        recepție OK...
    }
}
```

9.2 Instrucțiuni ns2

Pentru experimente se consideră topologia din figura 9.2. Nodul A transmite pachete cu flux constant CBR către nodul B, iar D transmite către C. Descărcați scriptul `twoflows.tcl` care primește următorii parametri:

```
ns twoflows.tcl -RTSthresh {RTS_Threshold}
                -CSthresh {carrier-sense threshold}
                -dist {x}
                -sendingRate {rate}
```

Cadrele cu lungime mai mare decât RTSthresh vor folosi RTS/CTS. Prin setarea acestui parametru la o valoare mare (3000) se dezactivează RTS/CTS. Setând acest parametru la 1 garantează folosirea RTS/CTS pentru toate cadrele. În scriptul furnizat se folosesc parametrii CSThresh_ și RXThresh_ pentru nivelele de putere în wați pentru CS și recepție respectiv. Setările de bază în ns2 corespund la 250m pentru recepție, și 550m pentru CS. În acest laborator nu vom schimba nivelul pragului de recepție, ci doar pe cel de CS, folosind parametrul din linia de comandă. Astfel, CSThresh=250m corespunde dezactivării CS fizic. Pentru a obține nivelele de putere corespunzând diverselor distanțe, se poate folosi utilitarul

```
/opt/ns/ns-2.34/indep-utils/propagation/threshold
    -m TwoRayGround [distanță_metri]
```

cu modelul TwoRayGround și păstrând toți ceilalți parametri la valorile default. Unele dintre aceste valori sunt deja calculate în scriptul twoflows.tcl (sunt comentate). Nu folosiți -fr 2.4e9, întrucât TwoRayGround trece la Friis sub 235m (vezi /opt/ns/ns-2.34/mobile/tworayground.cc). Pentru fiecare valoare CSThresh, rulați simularea dezactivând RTS/CTS pentru valori x=100m, 200m, 300m, si pentru fiecare valoare CSThresh corespunzătoare distanțelor 250m, 300m, 400m, 550m. Atenție, scriptul cere valoarea CSThresh în watți, nu în dBm, și nici în metri. $1mW = 10^{-6}W$ $dBm = 10\log_{10} mW$ $mW = 10^{\frac{dBm}{10}}$

În utilitarul POSIX `bc`, aceste formule se pot exprima astfel:

```
$ dBm=24
$ mW=3.5*10^-7 # atenţie: bc nu acceptă formatul
               # IEEE 3.5e-7 acceptat de tcl/C++
$ echo "e($dBm/10*l(10))" | bc -l
               # adică 24dBm = 1258.92mW (1.25892W)
$ echo "10*l($mW)/l(10)" | bc -l
               # adică 3.5e-7mW = -64.5dBm
```

Repetați toate experimentele și pentru cazul în care RTS/CTS este activat pentru toate cadrele. Pentru fiecare situație, calulați debitul și numărul de pachete trimise/primite la nivel UDP. Folosiți indicațiile din laboratorul precedent pentru a sintetiza aceste statistici din fișierul trace rezultat în urma simulării. Din comenzile folosite în laboratorul trecut:

- numărul de pachete trimise la nivel UDP de către toate nodurile.

```
cat fișier.tr | grep AGT | grep ^s |grep cbr | wc -l
```

- Pentru a obţine numărul de pachete primite, aveţi posibilitatea să utilizaţi o comandă similară:

```
cat fișier.tr | grep AGT | grep ^r |grep cbr | wc -l
```

- numărul de pachete trimise în mediu de către toate nodurile.

```
cat fișier.tr | grep MAC | grep ^s |grep cbr | wc -l
```

- numărul global de pachete de date (CBR) care au intrat în coliziune.

```
cat fișier.tr | grep COL | grep ^d |grep cbr | wc -l
```

- Deasemenea, calculați numărul de coliziuni pentru date și pentru non-date. De exemplu, pentru a calcula coliziunile cadrelor care nu conțin date (CBR), folosiți

```
cat fișier.tr | grep COL | grep ^d |grep -v cbr | wc -l
```

Cumulați toate rezultatele fie în tabele, fie în grafuri cu bare. Încercați să explicați **fiecare** număr obținut.

9.2.1 Analiză

IEEE 802.11 se bazează în principal pe două mecanisme pentru a combate interferența: CS la nivel fizic și la nivel virtual. În ce măsură sunt capabile aceste mecanisme de a rezolva conflictele dintre stații aflate în diverse configurații geometrice? În ns-2, un cadru este recepționat corect atunci când nivelul interferenței de la alte cadre este sub 10dB. Folosind utilitarul `/opt/ns/ns-2.34/indep-utils/propagation/threshold`, puteți calcula puterea recepționată la o anumită distanță. De exemplu, două cadre primite de la 200 metri și 200*sqrt(2) metri vor fi distruse, întrucât puterile sunt de 8.9e-10W respectiv 2.2e-10W. Pe de altă parte, dintre două cadre primite de la 200m și 360m, cel de la 200m va fi primit corect prin efectul de captură. La 355.7m avem exact 10dBm fata de 200m. Puteți folosi utilitare online pentru a lucra cu dBm. Teorema lui Pitagora

```
echo "sqrt(200^2 + 300^2)" | bc -l
```

Fiecare experiment este caracterizat de distanța x dintre nodurile B și D, și de distanța la care CS fizic este activ. Justificați valorile obținute pentru: debit, numărul de cadre trimise de MAC, numărul de coliziuni date, numărul de coliziuni non-date.

- Care este capacitatea maximă a canalului în pps (pachete/sec), bps (biți/sec)? [1]
- Ce rată de transmisie folosiți pentru agentul UDP emitent? [2]
- În ce măsură schimbul RTS/CTS afectează capacitatea sistemului? [3]

[1] Se rulează cu separare mare și CS la 250m, adică -CSthresh 3.65262e-10 -dist 10000, și rezultă o capacitate de 1.72Mbps pentru fiecare pereche în izolare. 720 pachete livrate pentru toata simularea de 5 duce la o capacitate de 144 pps, sau 1000/144 7ms / pachet.

[2] 1.72Mbps

[3] RTS/CTS reduce capacitatea la 1.57Mbps datorită overhead-ului

- Explicați *fiecare* valoare obținută.
- x=100m CS=250m(inactiv). Ce debit total ați obținut? De ce nu este maxim? [4] De ce nu este 0 [5]?
- x=100m CS=550m. Ce debit total ați obținut? De ce nu este maxim? De ce este exact ....? [6]
- x=200m CS=250m(inactiv). Ce diferență cantitativă este față de cazurile precedente? Dar calitativă?
- x=300m CS=400m. De ce există coliziuni?
- Cu RTS: de ce exista cazuri in care mai avem coliziuni de date?
- Atenție: sunt 3 geometrii (x=100, 200, 300) și 4 cercuri de CS (250/off, 300, 400, 500) - deci un total de 12 situații diferite. **Justificați fiecare valoare obținută**.
- Când este benefic CS virtual, și când CS fizic?
- În unele situatii se foloseste atât CS fizic, cât și virtual. Puteți garanta evitarea completă a coliziunilor atunci când ambele mecanisme sunt activate?
- Găsiți un exemplu de inechitate (CSThresh=?, x=?, debit=?)
- Refaceti xperimentele pentru cadre de dimensiune mica (32 de octeti). Ce se observa?
- Modificați topologia inițială pentru a avea A, B, C coliniare (D rămâne la fel). Rulați aceleași experimente și explicați coliziunile la RTS și la date.

9.3 Rezultate

După rulare cu seed-ul de randomizare default, se obțin rezultatele detaliate în tabelele de mai jos care folosesc următoarele coloane:

CS - CS în metri

CS - puterea la marginea CS

[4] avem coliziuni, este terminal ascuns, cadrele sunt distruse în mod sistematic

[5] datorită coliziunilor repetate, CW crește suficient de mult... la o valoare suficient de mare, cealaltă conversație va reuși să strecoare un pachet

[6] emițătorii sunt în CS, împart mediul în mod egal, fiecare obține 50%

bps - debit

x - distanta in metri

UDP - trimis, primit de agenti

MAC - trimise de mac

COL - coliziuni la date

CN - coliziuni non-date (ACK, RTS)

Rulare x=100m se obține AD=412m, BD=223m, P(200m)=8.91e-10W=-60dBm P(223m)=5.76e-10W=-62dBm.

```
CS     CS          x   bps     UDP  UDP MAC  COL CN
250m 3.65262e-10 100 243904  1709 103  995 892 0
300m 1.76149e-10 100 267584  1713 113 1023 910 0
400m 5.57346e-11 100 265216  1709 112 1002 890 0
500m 1.55924e-11 100 1783104 1734 753  753   0 0
```

Comentarii x = 100m
(250m) coliziuni BD=223m
(300m) coliziuni BD=223m
(400m) coliziuni BD=223m
(500m) A,D in CS

Rulare x = 200m distanțele relevante sunt: AD=447m BD=282m.

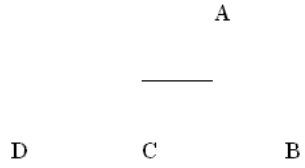
```
CS    CS          x   bps     UDP  UDP MAC  COL CN
250m 3.65262e-10 200 1752320 1727 740 1203 463 0
300m 1.76149e-10 200 291264  1722 123 1014 891 0
400m 5.57346e-11 200 298368  1715 126 1005 879 0
500m 1.55924e-11 200 1757056 1743 742  742   0 0
```

Comentarii x = 200m
(250m) coliziuni doar de ACK (mic, scapa mai des). Cum demonstrăm? Pentru a găsi prima coliziune: timpul t, pachetul pi

```
sh$ read t pi <<< $(cat twoflows-1channel.tr | grep COL | \
        head -n1 | awk '{print $3, $47}')
```

Printăm evenimenlele care duc la coliziune:

```
sh$ cat twoflows-1channel.tr | grep MAC | awk -v t=$t -v pi=$pi
        '$47==pi{p=1}  (p==1)&& ($3 <= t) {print $0}'
```



Se observă ACK de la cealaltă conversație chiar înainte de COLiziune. Pentru a vedea inechitatea rulați de mai multe ori, comparați throughput la cele două fluxuri.
(300m) coliziuni BD<300m, similar cu x=100m
(400m) coliziuni BD<400m, similar cu x=100m
(500m) A,D in CS

Rulare x = 300m AD=500m BD=360m P(360)=8.49e-11W=-71dBm

CS	CS	x	bps	UDP	UDP MAC	COL	CN
250m	3.65262e-10	300	3495168	1703	1476 1476	0	0
300m	1.76149e-10	300	3490432	1736	1474 1474	0	0
400m	5.57346e-11	300	1749952	1707	739 1179	440	0
500m	1.55924e-11	300	1754688	1713	741 741	0	0

Comentarii x = 300m
(250m) captura BD > 355.7
(300m) captura BD > 355.7
(400m) coliziuni ACK 300 < 355.7, capturi date 360 > 355.7
(500m) A, D in CS → terminal expus. cercul de CS de 500m nu permite conversațiile simultane A→B și C→D care ar funcționa cu un CS mai mic, de 250m sau 300m.

9.4 Problemă teoretică

Să considerăm topologia din figura 9.4:

Distanțele AB, BC, AC, and CD sunt toate de 200m. Avem un obstacol între A și C care nu permite propagarea radio (greu de realizat în ns2, deoarece AD=346m). Care vor fi comportările celor două forme de CS în acest caz? Discutați fiecare schemă și estimați care se va comporta mai bine. Reminder1: ACK se transmite după SIFS de la primire cadru. Reminder2: Un cadru primit cu putere peste CSThresh, dar sub RXThresh va distruge recepția curentă, fără captură.

Capitolul 10

Laboratorul 10

10.1 MCS multiple

Se dă scriptul `multirate.tcl` care definește nodurile WiFi 0,1,2,3,4 astfel:

```
      3

4     0     2

      1
```

Nodul 0 funcționează ca AP și emite la 11Mbps, iar celelalte noduri sunt clienți, cu legături uplink de 1Mbps. Sunt definite patru fluxuri UDP 1-¿3, 2-¿4, 4-¿0, 0-¿4, toate cu pachete de 1460 de octeți. Distanțele de la clienți la AP sunt de 20m, RTS este dezactivat, RxThresh=250m, iar acești parametri nu vor fi modificați. Scriptul **necesită** următorii parametri pe linia de comandă:

- rate0 indică rata oferită pentru fluxul 1 → 3
- rate1 indică rata oferită pentru fluxul 2 → 4
- rate2 indică rata oferită pentru fluxul 4 → 0
- rate3 indică rata oferită pentru fluxul 0 → 4

La sfârșitul simulării de 100 secunde se afișează throughputul în bps pentru fiecare flux, și numărul total de pachete transferat pe durata întregii simulări. Scriptul cere ca parametrii rate să fie diferiți de zero, dar pentru a dezactiva un flux, trebuie furnizată o rată neglijabilă, de exemplu 0.01bps. Unitatea de măsură nu se separă cu spațiu: 11Mbps, 360Kbps, etc.

10.2 Exerciții

1. Să se calculeze capacitatea maximă a aerului în pps si bps pentru fiecare emițător luat în izolare. Cât durează transmiterea unui cadru de la stații? Dar de la AP?

2. Dacă doar fluxul 0 ($1 \rightarrow 3$) este activ, ce throughput se obține end to end?

3. Care este rata coliziunilor (cps)? Cât durează o coliziune?

4. Cum se justifică valoarea obținută? Care este compoziția cadrelor din aer?

5. Dacă doar fluxurile 0 ($1 \rightarrow 3$) și 1 ($2 \rightarrow 4$) sunt active, ce throughput se obține end to end?

6. Cum se justifică valoarea obținută?

7. Cum puteți obține o valoare mai mare?

10.3 Rezolvare

1. Se dezactivează toate fluxurile în afară de unul pentru a studia un transmițător în izolare:

```
ns ./multirate.tcl -rate0 0.01bps -rate1 0.01bps
                   -rate2 0.01bps -rate3 11Mbps
```

   Se obțin 511pps, sau 5.97Mbps pentru AP. Un cadru durează $\frac{1000}{511} = 1.95ms$

```
ns ./multirate.tcl -rate0 0.01bps -rate1 0.01bps
                  -rate2 11Mbps -rate3 0.01bps
```

   Se obțin 77pps, sau 902Kbps pentru statii. Un cadru durează $\frac{1000}{77} = 12.98ms$. E mai greu de folosit fluxurile $1 \rightarrow 3$ și $2 \rightarrow 4$ deoarece un pachet trece de două ori prin aer pentru aceste cazuri. Pentru acest setup, cele două treceri sunt la MCS diferite.

2. Se activează doar fluxul 0 ($1 \rightarrow 3$):

```
ns ./multirate.tcl -rate0 902Kbps -rate1 0.01bps
                   -rate2 0.1bps -rate3 0.1bps
```

Se obțin 755Kbps, sau 64pps. În aer se află cadre scurte, la 11Mbps, și cadre lungi, emise la 1Mbps, cel puțin câte 64 din fiecare, deoarece 64pps sunt livrate la destinație. Este posibil să mai fie și cadre în coliziune.

3. `cat ./multirate.tr | grep cbr | grep COL | wc -l`

   Rezultă 4cps coliziuni pe secundă. Deoarece toate dispozitive sunt în CS, nu avem terminale ascunse, și coliziunile sunt datorate ferestrei de contenție. Avem 2 transmițători: AP și stația 1, ce emit cadre de lungimi diferite. O coliziune durează până la sfârșitul cadrului cel mai lung (802.11 nu are CSMA/CD), adică 12.98ms. 4*12.98ms = 52ms = 5.2% din timp.

4. Trebuie să aflăm câte cadre din fiecare tip se află în aer într-o secundă. Fiecare emițător câștigă aceleași oportunități de transmisie pe termen lung[1], x cadre pe secundă, dar fiecare îl ține ocupat în funcție de MCS folosit: (1Mbps, 11Mbps) $R_1 = 77pps, R_2 = 511pps$. În plus mai avem și C coliziuni pe secundă. $\frac{x}{R_1} + \frac{x}{R_2} + \frac{C}{R_1} = 1$ $x = (1 - \frac{C}{R_1})\frac{R_1 R_2}{R_1 + R_2} = 63pps$ adică 63*1480*8 = 751Kbps, consistent cu ce am obținut în simulare.

   - Stația 1 ocupă 64*12.98 = 823ms = 82% din timp.
   - AP ocupă 64*1.95=123ms = 12% din timp
   - coliziuni, restul de 5% din timp
   - Coada de pachete este probabil mereu goală în AP, și în creștere la stație, deoarece se generează mai mult decât se poate emite.

5. Activăm fluxurile 0 și 1, oferind maximul precedent:

```
ns ./multirate.tcl -rate0 755Kbps -rate1 755Kbps
                   -rate2 0.1bps -rate3 0.1bps
cat ./multirate.tr  | grep cbr | grep COL | wc -l
```

Se obțin câte 199Kbps pentru fiecare flux, adică 17pps. Coliziuni 8.2cps.

[1] De ce? Explicați folosind algoritmul de acces la mediu 802.11, sau prin rulare: -rate0 0.1bps -rate1 0.1bps -rate2 11Mbps -rate3 11Mbps

6. Dacă se transmite un maximum de la fiecare sursă (1 și 2), AP-ul va avea de retransmis pentru ambele, dar nu câștigă mediul decât în 1/3 din arbitrări, deși cadrele lui sunt mai scurte. $\frac{x}{R_1} + \frac{x}{R_1} + \frac{x}{R_2} + \frac{C}{R_1} = 1$ Ce este diferit față de cazul precedent este că toți cei trei emițători sunt saturați, AP-ul însă nu dirijează decât $\frac{x}{2}$ pentru fiecare flux, iar restul este pierdut. $x = (1 - \frac{C}{R_1})\frac{R_1 R_2}{R_1 + 2R_2} = 32pps$ Dar debitul obținut pentru fiecare flux este doar $x/2 = 16$pps, adică 190Kbps.

   - Stația 1 ocupă 32*12.98ms = 42% din timp
   - Stația 2 ocupă 32*12.98ms = 42% din timp
   - AP ocupă 32*1.95ms = 6% din timp
   - Coliziuni 8.2*12.98ms = 10% din timp
   - Adică 42% din timp este irosit pentru a transmite prin aer cadre care vor fi aruncate la AP!

7. Ar trebui ca fiecare cadru emis de o sursă să fie dirijat de către AP, si nu aruncat, deci împărțirea prin contenție nu se dorește echitabilă, ci AP-ul ar trebui să obțină dublu față de fiecare stație. x este acum debitul per flux: $\frac{x}{R_1} + \frac{x}{R_1} + \frac{2x}{R_2} + \frac{C}{R_1} = 1$ Din păcte C și x sunt interdependente, și o proporție de coliziuni este inevitabilă chiar dacă AP-ul este nesaturat. Cu 0 coliziuni am obține $x = \frac{0.5 R_1 R_2}{R_1 + R_2} = 33.5pps = 396Kbps$ Dacă rulăm, obținem 214Kbps și 7.9cps, ceea ce este departe de formulă, dar totuși crește debitul și scad coliziunile, deci este clar că trebuie trimis mai puțin de la surse. Un alt indiciu este că nu se pot emite mai mult de 755Kbps pentru un flux, deci 755Kbps/2 = 377Kbps de la fiecare stație 0 și 1, deoarece mediul este comun.

   ```
   ns ./multirate.tcl -rate0 377Kbps -rate1 377Kbps
                      -rate2 0.1bps -rate3 0.1bps
   ```

   Se obțin câte 337Kbps pentru fiecare flux, adică 29pps. Coliziuni C = 5.9cps. Folosind acest ultim C, obținem: $x = (1 - \frac{C}{R_1})\frac{0.5 R_1 R_2}{R_1 + R_2} = 31pps = 365Kbps$ Dacă oferim doar 365K, se obțin 362K, si doar 4cps (similar cu 2., 3.), probabil optimul pentru această configurație.

Capitolul 11

Laboratorul 11

11.1 Multihop, autointerferență

Se folosește un singur canal pentru toate hopurile din rețea. O topologie simplă este string: un șir de noduri care poate transporta trafic de la un capăt la altul. În general o rețea adhoc poate avea forma unui graf, dar două noduri care comunică vor folosi un șir de alte noduri pentru a transporta traficul. O particularitate a acestei topologii este aceea că pachetele aceluiași flux concurează pentru accesul la mediu. De fapt, un șir este o colecție de terminale expuse și ascunse.

11.2 Experiment 1

Se dă topologia A - B - C, nodurile fiind plasate la interval de 20m. CSThresh=550m, RXThresh=250m. Scriptul `string_bidir.tcl` primește parametrii -sendingRate0 [R0] -sendingRate1 [R1] care descriu ratele în bps oferite fluxurilor UDP0 A →B →C și UDP1 C →B →A (-sendingRate0 nu poate fi 0, dar se pot folosi valori foarte mici 0.01Kbps). Scriptul afișează debitul obținut, și produce fișierul trace ".tr" în formatul cunoscut.

1. să se determine capacitatea mediului în bps, și în pps.
2. atunci când fluxurile au cereri egale, să se examineze performanța obținută de fiecare flux în funcție de debitul oferit. Care sunt regiunile importante ale graficului?
3. Cum se justifică logic/numeric valoarea maximă și valoarea de saturație?
4. Ce se schimbă la 2. și 3. când avem o topologie de 4 noduri spațiate la 20m: A - B - C - D? -sendingRate0 se transmite de la A, iar -sendingRate1 se transmite de la D, -nn 4 rulează cu 4 noduri.

5. Ce valoare obține TCP în cele două situații?
6. de ce este mai mică decât ..., și mai mare decât...?

11.3 Experiment 2

Folosindu-se același script, se evaluează capacitatea pentru un șir de 20 noduri spațiate fie la 20m, fie la 200m (variabila hopDistance în script).

- trasați un grafic care arată capacitatea unui șir de noduri în funcție de lungimea lui (1-19 hopuri)
- ce cantitate de trafic injectați în rețea pentru a obține graficul?
- repetați graficul pentru UDP bidirecțional.
- rulați manual pentru diverse valori de trafic injectat
- repetați graficul pentru TCP. Opțiunea -run_tcp 1 generează trafic TCP unidirecțional.
- Explicați asemănările și diferențele pentru cele 2 tipuri de transfer (UDP/TCP)
- repetați graficul pentru TCP bidirecțional.
- comparați
- TCP vs UDP
- 20m vs 200m
- unidirecțional vs. bidirecțional

11.4 Experiment 3

TCP, 10 noduri, distanța 20m:

- Cum se comportă fereastra de congestie TCP? (cwnd_ în fișierul trace), RTT(srtt_ și rttvar_)
- care este valoarea BDP (bandwidth-delay product) pentru legătura obținută?
- unde se aruncă pachete din cozile ruterelor și unde se pierd în coliziuni?
- Ce dimensiune are coada unei interfețe wireless?
- Se poate obține throughput mai bun mărind coada?

Capitolul 12

Exemplu de colocviu

12.1 Overhead impus de SSID-uri multiple

Mulți producători oferă posibilitatea de a avea mai multe SSID-uri pe un AP cu un singur card fizic, operând desigur pe același canal. Se vor transmite beacon-uri și se vor accepta autentificări/asocieri pe amblele SSID ca și cum ar fi separate, deși fizic este un singur card. Se consideră situația în care operăm doar în 802.11b

- beaconul de 802.11b/g se transmite la 1Mbps, și este de minim 150 octeți
- beaconul folosește broadcast (fără SIFS+ACK), dar folosiți un script de unicast pentru aproximare

1. (10%) care este durata unui beacon?
2. (5%) câte beacon-uri pe secundă s-ar putea transmite fără contenție?
3. (5%) ce debit în bps folosește un SSID?
4. (10%) ce debit de TCP aș putea obține de la AP-ul meu 11b, 11Mbps în absența oricăror beacon-uri?
5. (10%) eu și vecinul operăm pe același canal, și fiecare avem câte 3 SSID-uri (intern, prieteni, DMZ). Se mai văd încă alte 4 SSID-uri unice în bloc, **toate foarte aproape și pe același canal**. Ce throughput de TCP pot spera să obțin în condiții optime: dacă sunt aproape de AP-ul meu, nu mai e nici un alt client în aer?
6. (40%) plot manual/gnuplot TCP Throughput(nr. SSID=1..20)
7. (20%) interpretare. Care este relația matematică între capacitatea disponibilă și numărul de rețele create? Puteți folosi funcția `fit` din gnuplot, sau fit manual.

12.2 Rezolvare

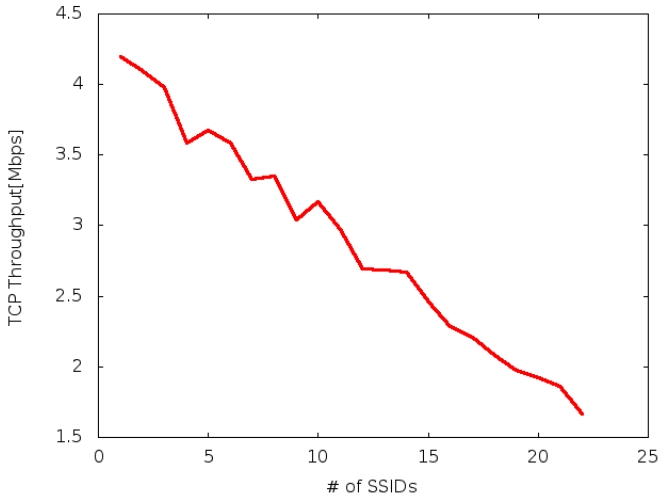
1. se folosește infra.tcl în care se modifică 11b, dataRate 1Mbps
2. ```
   ns ./infra.tcl -run_tcp 0 -nn 2 -packetSize 150
                  -sendingRate 1Mbps
   ```
3. rezultă 4000 packete în 10s, adică 2.5ms/beacon
4. 400pps
5. 150*8biți*10beacons/sec = 0.012Mbps

   ```
   ns ./infra.tcl -run_tcp 0 -nn 2 -packetSize 150
                  -sendingRate 0.012Mbps
   ```

   răspunde 10pps, se poate verifica scalarea cu număr mai mare de emițători
6. cu 11b/11Mbps obținem 4.15Mbps:

   ```
   ns ./infra.tcl -run_tcp 1 -nn 2 -packetSize 1460
   ```
7. un SSID emite 10 beacons/second, deci 25ms/second, adică 2.5% din capacitate. În cazul ideal, pierd liniar cu numărul de SSID vizibile, câte 2.5%. Pentru 3 eu+ 3 vecinul+ 4 altele = 10 = 25% pierderi, deci 3.1Mbps în cel mai bun caz.
8. se modifică infra.tcl pentru a crea această topologie. Vezi deasemenea multirate.tcl (lab 11) pentru a seta noduri cu MCS diferit
   - (idee: 10%)
   - nodurile 3..(nn-1) → 2 emulăm beacon-uri la 1Mbps, UDP de 150bytes, 10pps (0.012Mbps)
   - nodurile 1 → 0 la 11Mbps, TCP cu 1460
   - (descriere modificări: 10%) - rezultat rulare diff

     ```
     > # setarea MCS diferențiat pentru noduri:
     >         if { $i <= 1} {
     >             Mac/802_11 set dataRate_        11Mb
     >           # nodurile 0 și 1
     >         } else {
     >             Mac/802_11 set dataRate_        1Mb
     >           # nodurile 2..(nn-1)
     ```
     ```




```
>          }
>          $ns_ node-config -macType $val(mac)
...
> # planificarea a 1 flux TCP 0-1,
> # nn-2 fluxuri UDP/150/10pps=12Kbps
>
> for {set i 3} {$i < [ expr $val(nn) - 1]} {incr i} {
>      attach-cbr-traffic $i 2 150 0.012Mbps \
>         $val(start0) $val(stop0) $i
>      # emulare beaconuri
313a314
> attach-tcp-traffic 1 0 1460 $val(start0) \
>      $val(stop0) 1001
```

- (cum rulez: 10%)Se rulează

```
x=4; while [ $x -le 23 ]; do
  echo -n "$(($x-3)) ";
  ns ./infra1.tcl -nn $x | grep 'Throughput 1001';
  x=$(($x+1)); done | tee b
```

9. x=4 înseamnă 0 →1 TCP, 3 →2 beaconuri, rezultă 4.15Mbps
10. (grafic cu pixul: 10%)

```
gnuplot> plot 'b' using 1:4 w l
```

Bibliografie

[1] Matthew S Gast. 802.11 Wireless Networks: The Definitive Guide, Second Edition. O'Reilly Media, Inc., 2005.

[2] IEEE 802.11 WIRELESS LOCAL AREA NETWORKS. http://www.ieee802.org/11/.

[3] The Network Simulator - ns-2. URL http://www.isi.edu/nsnam/ns/.

[4] The ns-3 network simulator. URL http://www.nsnam.org/index.html.

[5] Dragoș Niculescu. Infrastructuri și Servicii pentru Rețele Mobile, pagina web a cursului. URL http://ocw.cs.pub.ro/isrm.

[6] J. Chung și M. Claypool. NS by example. URL http://nile.wpi.edu/NS/.

[7] Sunwoong Choi, Kihong Park, and Chong-kwon Kim. On the performance characteristics of wlans: revisited. In ACM SIGMETRICS Performance Evaluation Review, volume 33, pages 97–108. ACM, 2005.

[8] Ns-2 labs @ illinois wireless center. URL www.crhc.illinois.edu/wireless/assignments/simulations/slabc1.html.

[9] Martin Heusse, Franck Rousseau, Gilles Berger-Sabbatel, and Andrzej Duda. Performance anomaly of 802.11 b. In INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies, volume 2, pages 836–843. IEEE, 2003.

[10] Marc Greiss. ns-2 tutorial. URL http://www.isi.edu/nsnam/ns/tutorial/.

[11] ns-2 manual. URL http://www.isi.edu/nsnam/ns/doc/node1.html.

[12] Scott Turner. Tcl cheatsheet. URL http://www.pkturner.org/ programming/ Tcl_cheat_sheet.html.

[13] Hagen Wierstorf. Ghid gnuplot. URL http://www.gnuplotting.org/plotting-data/.