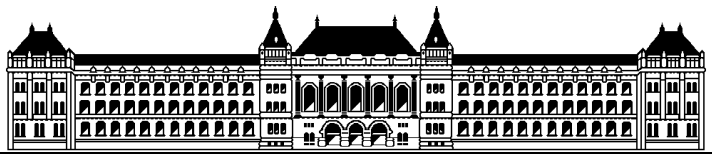


Bevezetés a Network Simulator használatába (mérési segédlet)

Távközlési és Médiainformatikai Tanszék

összeállította: Németh Felicán (nemethf@tmit.bme.hu)

átdolgozta: Császár András (Andras.Csaszar@tmit.bme.hu)

2003. október 20.

1. Bevezetés

A *Network Simulator* egy diszkrét eseményvezérelt, csomagszintű hálózati szimulátor, amelyet számos egyetem, kutatóintézet támogat; forráskódja szabadon hozzáférhető[1]. A program segítségével lehetőség van többek között útvonalválasztó, multicast vagy TCP protokoll vizsgálatára akár vezetéknélküli hálózatok esetén is.

Az *ns*-t c++ és otcl nyelven írták. C++ nyelven a szimulátor magját, a protokollok implementációját, valamint a nagy számításigényű feladatokat valósították meg, ezzel szemben az otcl szkriptnyelv a szimulációs konfiguráció megadásához, és egyszeri, nagy hatékonyságot nem igénylő feladatok megoldására használatos.

[1]Ld. http://www.isi.edu/nsnam

A rendszer objektum hierarchiája jól átgondolt, a rendszerhez új funkcionalitás a c++ környezetben könnyedén illeszthető, erre azonban ebben a bevezető jellegű mérésben nem kerül sor. Így a mérés célja az otcl felület alapjainak megismertetése, ezen keresztül néhány egyszerűbb hálózati szimulációs vizsgálati módszer elsajátításának támogatása.

Az ns program tulajdonképpen felfogható úgy is, mint objektum orientált kiterjesztéssel ellátott tcl interpreter, amelyből elérhető a hálózati szimulációt támogató osztályhierarchia.

A szimuláció első lépesként el kell készíteni egy (o)tcl szkriptet, amely definiálja a hálózati komponenseket, azok összekapcsolódását, valamint az előre megadott események (pl. adatforrás adáskezdésének) időzítését. Az ns programot ezután meg kell hívni szkripttel („ns program.tcl"), ami a kimeneti fájlokban előállítja a naplóbejegyzéseket és a szimulációs eredményeket. Az utófeldolgozás során szimulációs eredményeket kielemezhetjük awk, perl, gnuplot, excel, stb programok segítségével. Az eredmények vizuális elemzéséhez a speciális formátumú naplófájllal meghívhatjuk a nam programot (NAM - Network Animator), amely grafikusan ábrázolva mutatja meg az egyes csomagok útját.

2. Tcl alapok

Noha az idők során fellángolt a vita, hogy a Tcl nyelv C programok szkriptnyelv-kiterjesztéseként elég hatékonyan használható-e[2], az ns rendszerben elérte a célját: egyszerű, világos nyelvi elemekkel gyors fejlesztést biztosít. Ebben a fejezetben a nyelv rövid, teljeségre nem törekvő leírása olvasható, amely mindazonáltal elegendő ismeretet biztosít a mérési feladtatok elvégzéséhez.

Egy Tcl szkript parancsait újsor vagy pontosvessző választja el. A parancsok egy vagy több szóból állnak, ahol az első szó a parancs neve, az esetlegesen azt követő szavak pedig a parancs argumentumai. A szavakat szóközök, illetve tabulátor jelek választják el egymástól.

Egy parancs kiértékelésekor az interpreter először szavakra bontja a parancsot és elvégzi az esetleges helyettesítéseket, majd végrehajtja a parancsot az adott argumentumokkal. `$változónév` a változó értékével helyettesítődik, míg a `[parancs]` alak alkalmazásával a megadott parancs visszatérési értéke kerül behelyettesítésre. A változóknak nincsen típusa, minded érték stringként tárolódik.

A nyelvről részletesebb leírás a tanszék Tcl/Tk mérési segédletében olvasható[3], melynek ismerete szükséges és épp ezért elvárt a mérérs megkezdésekor. A

[2]Ld. „The TCL War" http://www.vanderburg.org/Tcl/war

[3]Ld. http://alpha.ttt.bme.hu/pub/meresek/3x/02/tcltk/

10! értéket kiszámító 1. szkriptben néhány alapvető Tcl nyelvi elem megtalálható.

```
1:  set fakt 1
2:  for {set i 1} {$i <= 10} {incr i} {
3:     set fakt [expr $fakt*$i]
4:  }
5:  puts $fakt
```

1. program. 10 faktoriálist kiszámító tcl szkript

Az első sorban a `set` parancs segítségével a `fakt` változó értékét egyre állítjuk. A második sorban a C nyelvhez hasonló for ciklusban a kapcsos zárójelek megakadályozzák a helyettesítést a parancs szavakra bontásakor, így például a ciklus magjának kiértékelésekor a változók mindig az aktuális értéküket veszik fel. A szögletes zárójelekkel történő parancshelyettesítésre mutat példát a harmadik sor, ahol az `expr` parancs egy numerikus kifejezést számol ki, amelynek eredménye lesz – a `set` értékadás után – a `fakt` változó új értéke. Végezetül az eredmény kiíratását a `puts` parancs végzi el az utolsó sorban.

A faktoriális kiszámításának rekurzív módjából a függvények deklarálására láthatunk példát (ld. a 2. programot).

```
0:  proc faktorialis num {
1:    if {$num > 0} {
2:      return [expr $num*[faktorialis [expr $num-1]]]
3:    } else {
4:      return 1
5:    }
6:  }
7:  puts [faktorialis 10]
```

2. program. Faktoriális-számító tcl parancs

3. Otcl alapok

Ez a fejezet a Tcl objektum orientált kiterjesztésének, az otcl nyelvnek a használatára mutat egy egyszerű példát. Az átlagos *ns* felhasználónak ritkán van szüksége új objektum megírására, mégis mivel a szimulációs szkriptek készítésekor az ns objektumokat otcl-ből lehet elérni, szükséges megérteni a otcl nyelvi elemeit.

```
 0:  Class mom
 1:  mom instproc greet {} {
 2:    $self instvar age_
 3:    puts "$age_ year old mom say:  How are you doing?"
 4:  }

 5:  Class kid -superclass mom
 6:  kid instproc greet {} {
 7:    $self instvar age_
 8:    puts "$age_ year old kid say:  What's up, dude?"
 9:  }

10:  set a [new mom]
11:  $a set age_ 45
12:  set b [new kid]
13:  $b set age_ 15

14:  $a greet
15:  $b greet
```

3. program. Otcl példa

Az ns honlapján található klasszikus példa a 3. programban olvasható.

A példaprogram a 0. sorban definiálja a `mom` osztályt a `Class` kulcsszóval, majd ebből származtatja le az 5. sorban a `kid` osztályt a `-superclass` kulcsszó segítségével. Az osztálydefiníciók után a tagfüggvények (metódusok) definiálása következik az `instproc` kulcsszó felhasználásával. A második és a hetedik sorban a tagfüggvények definiálásakor a `$self` szerepe a C++ nyelvbeli „this” mutatóéval megegyező. Az `instvar` tagfüggvény egyrészt deklarálja az `age_` attribútumot (tagváltozót) (ha az még nem volt deklarálva az osztályban vagy valamelyik szülőosztályban), másrészt az adott blokkon belül lehetőséget teremt csak a változónévvel hivatkozni az attribútumra. Végezetül a `new` paranccsal lehet egy osztályt példányosítani (10.,12. sor), a tagfüggvények a 14., 15. sorban kerülnek meghívásra, amelyek kimenete a következő lenne:

```
45 year old mom say:  How are you doing?
15 year old kid say:  What's up, dude?
```

4. Szimulációs példaprogram

Ez a fejezet egy egyszerű szimulációs szkriptet ismertet sorról sorra, bemutatva egy szimuláció elkészítéséhez szükséges alapvető lépéseket.

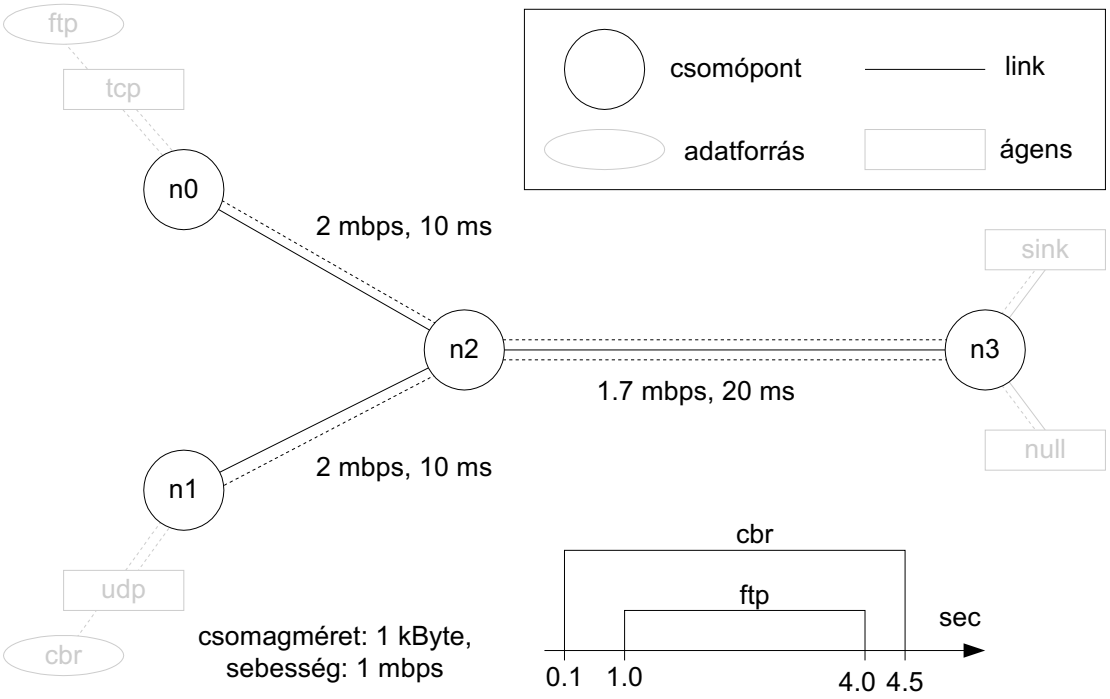
A szimulálandó hálózat, ahogy az az 1. ábrán látható, négy csomópontból áll (`n0, n1, n2, n3`). Az `n0` és `n2` valamint az `n1` és `n2` csomópontok közötti duplex link kapacitása 2 mbps, késleltetése 10 milliszekundum. A harmadik, `n2` és `n3` közötti duplex link kapacitása 1.7 mbps, késleltetése 20 milliszekundum. Mindegyik csomópont 10 csomag kapacitású `DropTail` sort használ, amely FIFO elv szerint szolgálja ki a várakozó csomagokat és a 10 csomag tárolására alkalmas pufferének telítődése esetén az újonnan érkező csomagokat eldobja.

Az `n0` csomóponthoz csatolt TCP ágens az `n3` csomóponthoz csatolt TCP nyelő (`sink`) ágenssel épít ki kapcsolatot. A TCP ágens – alapértelmezés szerint – legfeljebb egy kilóbájtos csomagokat tud generálni. A nyelő feladata az ACK válaszüzenetek küldése.

Az `n1` csomóponthoz csatolt UDP ágens az `n3`-beli `null` ágenssel kommunikál. A TCP nyelővel ellentétben a `null` ágens egyszerűen csak felszabadítja a csomaghoz rendelt memóriát, mivel UDP esetén nincs szükség nyugta küldésére. A `tcp` ágenshez `ftp`, míg az `udp` ágenshez `cbr` forgalomgenerátor van csatlakoztatva, ez utóbbi konstans bitsebességű *(Constant Bit Rate)* adatfolyamot állít elő: egy kilóbájtos csomagokat küld 1 Mpbs sebességgel. A `cbr` $0.1s$ és $4.5s$ között generál forgalmat, az `ftp` pedig $1.0s$ időpontban kezd el adni és $4.0s$ időpillanatban fejezi be az adást.

A fenti szimulációs elrendezést a 4. tcl program valósítja meg. A program tanulságosabb mozzanataihoz fűzött magyarázatok a megfelelő sorszám megjelölésével az alábbiakban olvashatóak.

- `0: set ns [new Simulator]`: példányosítja az ns szimulátor objektumot és az objektumreferenciát az `ns` változóban tárolja le. Ez minden ns szkript kezdő sora, amely többek közt inicializálja a diszkrét idejű ütemezőt. A `Simulator` osztály a metódusain keresztül a következő feladatok ellátására is képes:
  - hálózati objektumok hozhatók létre (csomópontok, linkek, ...),
  - összekapcsolhatók a hálózati objektumok (pl. `attach-agent`),
  - hálózati elemek paraméterei állíthatók be,
  - adatkapcsolat definiálható ágensek között (pl. a `tcp` és a `sink` között),
  - a NAM megjelenítési paraméterei szabályozhatók.



1. ábra. Az egyszerű szimuláció hálózati elrendezése

- `1-2:` `$ns color` *`fid color`*: az *`fid`* folyamazonosító által meghatározott folyam csomagjainak színét állítja be. A `Simulator` objektum ezen metódusának csak a NAM megjelenítésre van hatása, és nem a tényleges szimulációra.

- `4:` `$ns namtrace-all` *`fájl-leíró`*: a szimulációs nyomkövetési adatok NAM bemeneti formátumban történő lementését kapcsolja be ez a metódus. Ehhez hasonló a `trace-all` metódus, de az általános (könnyebben feldolgozható) formátumot használ.

- `5:` `proc finish {}`: a parancsot a szimuláció végén hívja meg az 52. sor `$ns at 5.0 "finish"` utasítása. A naplófájl lezárása után a NAM program meghívásával zárul a szimulációs szkript futása.

- `12-15:` `set n0 [$ns node]`: Ezzel a metódussal van lehetőség egy hálózati csomópont létrehozására.

- `16-18:` `$ns duplex-link` *`csomópont1 csomópont2 kapacitás késleltetés sor-típus`*: két simplex linket hoz létre a megadott sávszélesség és késleltetés értékekkel és összekapcsolja a két megadott

csomópontot. Az NS sajátossága, hogy a csomópont kimeneti sora a link részeként van megvalósítva, így a link létrehozásakor kell megadni a kimeneti sor típusát. Ha a példában `DropTail` helyett az aktív sormenedzsmentet használó `RED` típusú sort szeretnénk használni, egyszerűen azt a típust kell megadni a metódus utolsó paramétereként. Fontos tudni, hogy lehetőség van veszteséges linkek modellezésére is. (A felhasználható sortípusokról és a veszteséges linkek használatáról az NS dokumentációjából lehet többet megtudni.)

- `20-23`: a parancsoknak csak a NAM programmal való megjelenítéskor van szerepe, érdemes megfigyelni a változást a sorok kikommentezése után.

A fenti rész definiálja a hálózati topológiát, így a következő feladat a forgalmi ágensek (TCP, UDP) és az adatforrások (FTP, CBR) beállítása, valamint az ágensek csomópontokhoz és a források ágensekhez való csatlakoztatása.

- `24,27,34,36: set tcp [new Agent/TCP]`: így lehet TCP ágens létrehozni, ehhez hasonlóan lehet bármilyen ágenst vagy forgalomgenerátor létrehozni, mindössze az osztály nevének ismeretére van szükség. Ez megtalálható az NS dokumentációjában, azonban egy másik lehetőség beleolvasni az „ns-2/tcl/libs/ns-default.tcl” fájlba. Ez a fájl tartalmazza a hálózati objektumok konfigurációs paraméterek alapértelmezett értékeit, így megtudhatjuk milyen szimulációs objektumok vannak és melyek az objektumok beállítható paraméterei.

- `26,28,35,37: $ns attach-agent` *csomópont ágens*: A `Simulator` ezen metódusa egy ágens egy csomóponthoz való csatlakoztatására szolgál. Az `attach-agent` csak meghívja a megadott csomópont `attach` metódusát, amely ezután magához csatolja a megadott ágenst. Tehát például a 26. sorral azonos hatású a „`$n0 attach $tcp`” parancs.

- `29,38: $ns connect` *ágens1 ágens2*: a két egymással kommunikálni kívánó ágens létrehozása után ki kell építeni köztük a logikai kapcsolatot, erre szolgál ez a parancs, amely az ágensekben beállítja a másik ágens hálózati- és portcímét.

A hálózati konfiguráció megadása után a szimulációs forgatókönyvet, azaz az előre megadott események időzítését kell definiálni. A `Simulator` osztálynak számos, az ütemezéssel kapcsolatos metódusa van, azonban a leggyakrabban használt metódusa a következő:

- `46-52: $ns at` *időpont parancs*: A `Simulator` objektum az ütemezője segítségével a megadott szimulációs időben végrehajtja az adott

parancsot. Így például a 46. sor $0.1s$-re ütemezi a „`$cbr start`” parancs végrehajtását, azaz a $0.1s$ időpontban meghívja a `$cbr` adatforrás objektum `start` metódusát, amely ennek hatására elkezdi a forgalomgenerálást.

A hálózati elrendezés, az időzítések és a szimulációt lezáró eljárás megadása után már csak el kell indítani a szimulációt az 55. sor `$ns run` parancsával.

```
 0:  set ns [new Simulator]

 1:  $ns color 1 Blue
 2:  $ns color 2 Red

 3:  set nf [open out.nam w]
 4:  $ns namtrace-all $nf

 5:  proc finish {} {
 6:      global ns nf
 7:      $ns flush-trace
 8:      close $nf
 9:      exec nam out.nam &
10:      exit 0
11:  }

12:  set n0 [$ns node]
13:  set n1 [$ns node]
14:  set n2 [$ns node]
15:  set n3 [$ns node]

16:  $ns duplex-link $n0 $n2 2Mb 10ms DropTail
17:  $ns duplex-link $n1 $n2 2Mb 10ms DropTail
18:  $ns duplex-link $n2 $n3 1.7Mb 20ms DropTail
19:  $ns queue-limit $n2 $n3 10

20:  $ns duplex-link-op $n0 $n2 orient right-down
21:  $ns duplex-link-op $n1 $n2 orient right-up
22:  $ns duplex-link-op $n2 $n3 orient right
23:  $ns duplex-link-op $n2 $n3 queuePos 0.5

24:  set tcp [new Agent/TCP]
25:  $tcp set class_ 2
26:  $ns attach-agent $n0 $tcp
27:  set sink [new Agent/TCPSink]
28:  $ns attach-agent $n3 $sink
29:  $ns connect $tcp $sink
30:  $tcp set fid_ 1
31:  set ftp [new Application/FTP]
32:  $ftp attach-agent $tcp
33:  $ftp set type_ FTP

34:  set udp [new Agent/UDP]
35:  $ns attach-agent $n1 $udp
36:  set null [new Agent/Null]
37:  $ns attach-agent $n3 $null
38:  $ns connect $udp $null
39:  $udp set fid_ 2

40:  set cbr [new Application/Traffic/CBR]
41:  $cbr attach-agent $udp
42:  $cbr set type_ CBR
43:  $cbr set packet_size_ 1000
44:  $cbr set rate_ 1mb
45:  $cbr set random_ false

46:  $ns at 0.1 "$cbr start"
47:  $ns at 1.0 "$ftp start"
48:  $ns at 4.0 "$ftp stop"
49:  $ns at 4.5 "$cbr stop"

50:  $ns at 4.5 "$ns detach-agent $n0 $tcp ;
51:              $ns detach-agent $n3 $sink"

52:  $ns at 5.0 "finish"

53:  puts "CBR packet size = [$cbr set packet_size_]"
54:  puts "CBR interval = [$cbr set interval_]"

55:  $ns run
```

4. program. Egyszerű szimuációs szkript

5. Eredmények feldolgozása

A szimulációs eredmények a NAM megjelenítő programmal történő vizuális elemzésén túl a rendszer lehetőséget nyújt naplófájlt készíteni minden cso-

mag útjának minden eseményéről. A `Simulator trace-all` metódusa[4] által generált naplófájl minden sora egy-egy csomaggal kapcsolatos eseményre vonatkozik, a sor 12 szóközzel elválasztott mezőből áll, amelyek a következők:

1. az esemény típusa, lehetséges értékei:
   - „r” (receive): a célcsomóponthoz elért a csomag
   - „d” (drop): egy sor eldobta a csomagot
   - „+” (enqueue): várakozási sorba került a csomag
   - „-” (dequeue): várakozási sorból távozott a csomag
2. az esemény bekövetkezésének ideje (másodpercben)
3. a kezdő csomópont azonosítója
4. a vég csomópont azonosítója. Ez és az előző érték együttesen meghatározza azt a linket, amelyen az esemény történt. (Nem összekeverendő a 9., 10. mezők értékeivel)
5. A csomag típusa
6. A csomag mérete bájtokban
7. Jelzőbitek. (jelenleg csak az ECN bit használatos)
8. Folyamazonosító IPv6 esetén, amit tcl-ből be lehet állítani (4 program, 30. és 39. sor); még ha a szimulátor nem is használja ezt az azonosítót, az eredmények kiértékelésénél vagy nyomkövetésnél fel lehet használni. (Például a folyamazonosítót használja fel a NAM a csomagok színének meghatározásakor.)
9. *csomópont.port* alakú forráscím
10. *csomópont.port* alakú célcím
11. A hálózati réteg protokolljának sorozatszáma (sequence number); jóllehet az UDP megvalósítások nem használnak sorozatszámokat, az NS – a nyomkövetést elősegítendő – nyilvántartja az UDP csomagok sorozatszámát is.
12. A csomag egyedi azonosítója

[4]Az eddigi példák a `namtrace-all` metódust használták, amely azonban a `trace-all` metódustól eltérő kimeneti fájt állít elő.

Hosszabb szimuláció futtatása esetén, amikor például pusztán csak néhány aggregált jellemző értéke érdekes, feleslegesen helypazarló és lassú megoldás minden csomagról minden információt letárolni. Ezért – többek között – lehetőség van egyetlen várakozási sor megfigyelésére is, a részletek az NS dokumentáció „Trace and Monitoring Support" fejezetében olvashatóak.

Az NS terminológiában a csomagonkénti megfigyelést a „trace" kifejezéssel, míg az aggregált állapotfigyelést, számlálást a „monitoring" kifejezéssel illetik. Az utóbbi megoldást célszerű a legtöbb szimulációban alkalmazni, hiszen a sok ezer vagy millió csomag egyenkénti megfigyelése és eseményeik kiírása nagyon lelassítja a szimulációt, miközben óriási, akár kezelhetetlen kimeneti trace fájlt eredményez. Példaként kipróbálhatja egy sok csomagot átvivő szimulációt nam tracing bekapcsolásával illetve e nélkül.

Hatékonysági okokból tehát célszerű monitoring-ot alkalmazni. Ilyen objektumokról az NS dokumetáció fent említett fejezetében talál információt. Itt most csak példát mutatunk arra, hogy az előző példát hogyan kellene kiegészíteni, hogy egy sávszélesség (precízebben adatsebesség) kihasználtság ábrát kapjunk a 2-3-as linkre, például 2 másodperces mérési intervallummal. A kimeneti „2-3.bw" fájlt például *xgraph* vagy *gnuplot* programmal is megtekinthetjük.

```
...
set bwSampleInterval 2.0
...
[$ns link $n2 $n3] attach-monitors [new SnoopQueue/In]
     [new SnoopQueue/Out] [new SnoopQueue/Drop] [new QueueMonitor]
...
proc SampleBW {link} {
  global ns bwSampleInterval
  set qm [$link set qMonitor_]
  puts "[$ns now] [expr [$qm set bdepartures_] / $bwSampleInterval]"
  $qm set bdepartures_ 0
  $ns after $bwSampleInterval "SampleBW $link"
}
...
$ns at 0.0 "SampleBW [$ns link $n2 $n3]"
```

5. program. Monitoring kiegészítés