


ΔΙΚΤΥΑ ΕΠΙΚΟΙΝΩΝΙΩΝ

Εργαστηριακή Άσκηση 2

1. Συνθετότερα προβλήματα με το NS2

Στο πρώτο μέρος της Άσκησης 2 θα ορίσουμε στο NS2 μια τοπολογία δικτύου με τέσσερις κόμβους, στην οποία ένας κόμβος λειτουργεί ως δρομολογητής και προωθεί τα δεδομένα που στέλνουν δύο κόμβοι στον τέταρτο κόμβο. Θα βρούμε έναν τρόπο να διακρίνουμε τις ροές δεδομένων των δύο κόμβων αποστολής και θα δείξουμε πώς μπορεί να επιβλέπεται κάθε ουρά, ώστε να βλέπουμε πόσο γεμάτη είναι και πόσα πακέτα απορρίπτονται.

Σημείωση: Είναι σκόπιμο να ανατρέξετε στο φυλλάδιο της προηγούμενης Άσκησης, για να θυμηθείτε τις διαδικασίες με τις οποίες σώζονται και τρέχουν τα αρχεία. Θυμίζουμε ότι πριν προσομοιώσετε ένα αρχείο “*.tcl” πρέπει κάθε φορά να το σώζετε (*Save*). Για να εμφανιστεί ο δρομέας (cursor) στο “Command Prompt”, όταν είναι ανοιχτά το NAM ή το *Xgraph*, πρέπει πρώτα να κλείσετε τα παράθυρά τους. Κλείστε τα **μόνο** πατώντας το “X” δεξιά επάνω στην μπάρα του παραθύρου , **όχι** από το μενού *File*. Στο τέλος του εργαστηρίου κάντε “log off” πριν φύγετε. Μια έκδοση του κώδικα δίδεται στο τέλος κάθε Ενότητας για συμβουλευτικό σκοπό και μόνο, π.χ. σε περίπτωση λάθους κλπ.

1.1 Τοπολογία

Όπως πάντα, το πρώτο βήμα είναι ο ορισμός της τοπολογίας του δικτύου. Θα πρέπει να δημιουργήσετε ένα αρχείο, π.χ. “lab2a.tcl”, με τον τρόπο που περιγράφεται στην *Εργαστηριακή Άσκηση 1*, χρησιμοποιώντας τον κώδικα της *Ενότητας 2.2* εκείνης της άσκησης ως υπόδειγμα. Όπως ειπώθηκε προηγουμένως, αυτός ο κώδικας θα είναι πάντοτε παρόμοιος. Θα πρέπει πάντοτε να δημιουργείτε ένα αντικείμενο προσομοίωσης, να αρχίζετε την προσομοίωση με την ίδια εντολή και, αν θέλετε να τρέχει το NAM και το *Xgraph*, θα πρέπει να ανοίγετε πάντα αρχεία “trace”, να τα αρχικοποιείτε και να ορίζετε μια διαδικασία που να τα κλείνει. Εισάγετε τώρα τις παρακάτω γραμμές στο αρχικό *template* του προηγούμενου κώδικα, ώστε να δημιουργήσετε 4 κόμβους.

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

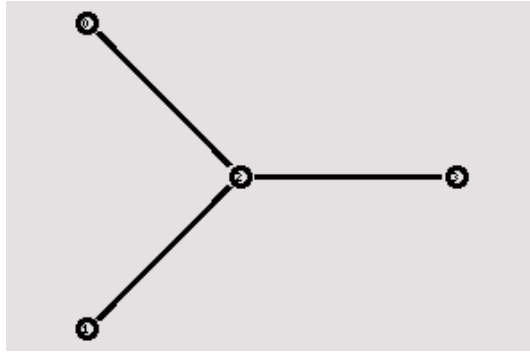
Το παρακάτω τμήμα κώδικα *Tcl* δημιουργεί τρεις αμφίδρομες ζεύξεις μεταξύ των κόμβων.

```
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n3 $n2 2Mb 10ms DropTail
```

Μπορείτε τώρα να σώσετε το αρχείο, με *File* → *Save*, και να τρέξετε το *script*. Μπορεί να φανεί στο NAM, ότι η τοπολογία είναι λίγο άκομψη. Μπορείτε να κάνετε κλικ στο κουμπί “re-layout” για να την κάνετε να φαίνεται καλύτερη, αλλά θα ήταν καλό να υπάρχει περισσότερος έλεγχος στο σχέδιο. Προσθέστε τις επόμενες γραμμές στο *Tcl script*, ξανασώστε και ξαναρχίστε το.

```
$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
```

Προφανώς αντιλαμβάνεστε τι κάνει αυτός ο κώδικας, όταν κοιτάξετε τώρα την τοπολογία στο παράθυρο του NAM. Θα πρέπει να έχει τη μορφή που φαίνεται στην Εικόνα 1.



Εικόνα 1 - Αναπαράσταση της τοπολογίας τεσσάρων κόμβων στο NAM

Παρατηρήστε ότι τα μέρη του NAM που έχουν σχέση με την αυτόματη σχεδίαση έχουν εξαφανιστεί, επειδή τώρα αναλαμβάνετε εσείς το σχέδιο. Μπορείτε να πειραματιστείτε με αυτό, προς το παρόν όμως αφήστε την τοπολογία όπως είναι.

1.2 Τα γεγονότα (events)

Δημιουργήστε τώρα δύο *UDP agents* με πηγές που παράγουν κίνηση *CBR* και προσαρτήστε τους στους κόμβους “n0” και “n1”. Έπειτα δημιουργήστε έναν *Sink Agent* και προσαρτήστε τον στον κόμβο “n3”.

Δημιουργία ενός *UDP agent* και «προσάρτησή» του στον κόμβο «n0»

```
set udp0 [new Agent/UDP]
```

```
$ns attach-agent $n0 $udp0
```

Δημιουργία μιας πηγής κίνησης *CBR traffic source* και «τοποθέτησή» της στον «udp0»

```
set cbr0 [new Application/Traffic/CBR]
```

Προσδιορισμός της κίνησης δεδομένων που παράγεται στον κόμβο «n0»

```
$cbr0 set packetSize_ 1500
```

```
$cbr0 set interval_ 0.01
```

```
$cbr0 attach-agent $udp0
```

Δημιουργία ενός *UDP agent* και «προσάρτησή» του στον κόμβο «n1»

```
set udp1 [new Agent/UDP]
```

```
$ns attach-agent $n1 $udp1
```

Δημιουργία μιας πηγής κίνησης *CBR* και «τοποθέτησή» της στον «udp1»

```
set cbr1 [new Application/Traffic/CBR]
```

Προσδιορισμός της κίνησης δεδομένων που παράγεται στον κόμβο «n1»

```
$cbr1 set packetSize_ 1500
```

```
$cbr1 set interval_ 0.01
```

```
$cbr1 attach-agent $udp1
```

Δημιουργία δύο *agents sink0* και *sink1* για τη λήψη δεδομένων

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

```
$ns attach-agent $n3 $sink0
```

```
set sink1 [new Agent/LossMonitor]
$ns attach-agent $n3 $sink1
# Σύνδεση των δύο CBR agents με τους sink agents
$ns connect $udp0 $sink0
$ns connect $udp1 $sink1
```

Είναι επιθυμητό να αρχίσει να στέλνει ο πρώτος *CBR agent* όταν $t = 0.5 \text{ sec}$ και να σταματήσει όταν το $t = 3.5 \text{ sec}$, ενώ ο δεύτερος *CBR agent* να αρχίσει όταν $t = 1.0 \text{ sec}$ και να σταματήσει όταν $t = 3.0 \text{ sec}$.

```
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 3.0 "$cbr1 stop"
$ns at 3.5 "$cbr0 stop"
```

Αρχίστε το script πληκτρολογώντας “ns lab2a.tcl” και ύστερα τρέξτε το *animation*.

- Εξηγήστε τις παρατηρήσεις σας.

Είναι προφανές ότι κάποια πακέτα από τις ροές χάνονται, αλλά δεν είναι εύκολο να τα προσδιορίσουμε. Και οι δύο ροές παριστάνονται με μαύρο χρώμα, οπότε ο μόνος τρόπος να διαπιστωθεί τι συμβαίνει στα πακέτα είναι να τα παρακολουθεί κάποιος στο NAM κάνοντας κλικ πάνω τους. Στις επόμενες δύο ενότητες αναφέρεται ο τρόπος διάκρισης των δύο διαφορετικών ροών και ο τρόπος παρακολούθησης του τι πραγματικά συμβαίνει στην ουρά της ζεύξης από “n2” προς “n3”.

1.3 Σημάδεμα ροών

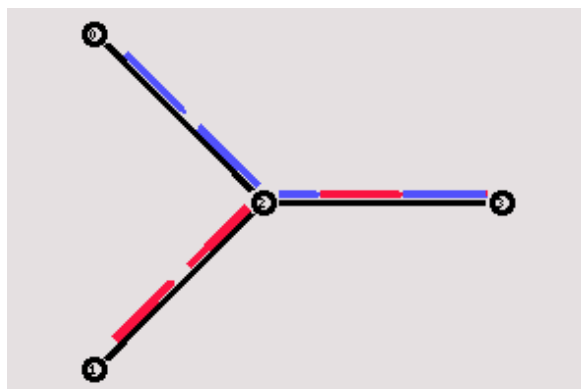
Προσθέστε τις παρακάτω δύο γραμμές στους δυο ορισμούς των *udp agents* αντίστοιχα.

```
$udp0 set class_ 1
$udp1 set class_ 2
```

Προσθέστε τώρα το παρακάτω τμήμα κώδικα στο *Tcl script*, κατά προτίμηση στην αρχή μετά τη δημιουργία του αντικείμενου (*object*) προσομοίωσης.

```
$ns color 1 Blue
$ns color 2 Red
```

Ο κώδικας αυτός σας επιτρέπει να αντιστοιχίσετε διαφορετικά χρώματα σε κάθε ροή πακέτου. Σώστε και τρέξτε το *script* και θα δείτε στο NAM ότι η μια ροή θα είναι μπλε ενώ η άλλη θα είναι κόκκινη, όπως φαίνεται στην *Εικόνα 2*.



Εικόνα 2 – Αναπαράσταση της κίνησης με έγχρωμη ροή πακέτων στο NAM

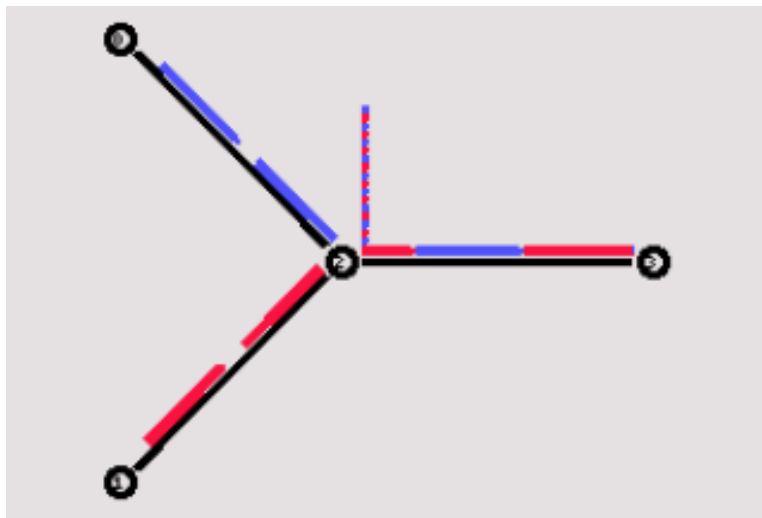
Αν παρατηρήσετε τη ζεύξη από “n2” προς “n3” για λίγο, και θα διαπιστώσετε ότι, μετά από παρέλευση κάποιου χρονικού διαστήματος, η κατανομή μεταξύ μπλε και κόκκινων πακέτων δεν είναι πλέον ίδια. Στην επόμενη ενότητα αναφέρεται ο τρόπος με τον οποίο μπορείτε να δείτε μέσα στην ουρά αναμονής κάθε ζεύξης για να παρατηρήσετε τι συμβαίνει.

1.4 Παρακολούθηση ουράς με το NAM

Πρέπει να προσθέσετε την παρακάτω γραμμή ορισμού της θέσης της ουράς στον κώδικά σας, για να παρακολουθήσετε την ουρά της ζεύξης από “n2” προς “n3”.

```
$ns duplex-link-op $n2 $n3 queuePos 0.5
```

Σώστε και τρέξτε πάλι το αρχείο, αρχίστε το NAM και θα δείτε μια εικόνα παρόμοια με την παρακάτω.



Εικόνα 3 – Αναπαράσταση στο NAM της ουράς πακέτων στη ζεύξη

Τώρα, μπορείτε να βλέπετε τα πακέτα στην ουρά, ενώ μετά από λίγο θα παρατηρήσετε πως απορρίπτονται κάποια πακέτα. Θα διαπιστώσετε ότι απορρίπτονται κυρίως μπλε πακέτα, αλλά δεν μπορούμε να περιμένουμε δικαιοσύνη από μια απλή ουρά *DropTail*.

Για να βελτιώσετε την ουρά αναμονής, κάνοντάς την πιο “δίκαιη”, χρησιμοποιήστε μια ουρά *SFQ* (Stochastic Fair Queuing) για τη ζεύξη από “n2” προς “n3”. Για να το κάνετε αυτό, αλλάξτε την γραμμή που ορίζει τη ζεύξη μεταξύ “n2” και “n3” με την παρακάτω γραμμή:

```
$ns duplex-link $n3 $n2 2Mb 10ms SFQ
```

Σημείωση: Θυμηθείτε ότι πρέπει να κλείνεται το NAM μόνο πατώντας το “X” δεξιά επάνω στην μπάρα του παραθύρου  και όχι από το μενού *File*.

1.5 Ερωτήσεις

- Ποιο είναι το ποσοστό των πακέτων που χάνονται από την μπλε και κόκκινη ροή, όταν χρησιμοποιείται η ουρά *DropTail*;
- Παρατηρώντας το *animation*, εκτιμήστε το ποσοστό των πακέτων που χάνονται από την μπλε και από την κόκκινη ροή, όταν χρησιμοποιείται η ουρά *SFQ*;
- Είναι το ποσοστό αυτό αναμενόμενο, αν λάβουμε υπόψη τον ρυθμό μετάδοσης κάθε πηγής και τη χωρητικότητα της ζεύξης; Ποιες είναι οι απώλειες κάθε ροής σε bit/sec;

1.6 Παρακολούθηση ουράς με το Xgraph

Για να μπορέσετε να δείτε τα αποτελέσματα της προσομοίωσης και σε γραφική παράσταση, θα πρέπει να ορίσετε μια διαδικασία καταγραφής της κίνησης σε ένα αρχείο εξόδου *trace*, προσθέτοντας τις παρακάτω γραμμές στο *script*, κάτω από τη εντολή ορισμού της θέσης της “ουράς” που είδαμε στην §1.4.

Διαδικασία (procedure) καταγραφής της κίνησης

```
proc record {} {
    global sink0 sink1 f0 f1
    set ns [Simulator instance]
    # Ορισμός της χρονικής περιόδου που θα ξανακληθεί η διαδικασία.
    set time 0.2
    # Καταγραφή των bytes
    set bw0 [$sink0 set bytes_]
    set bw1 [$sink1 set bytes_]
    # Ορισμός του χρόνου της τρέχουσας καταγραφής
    set now [$ns now]
    # Υπολογισμός του bandwidth και καταγραφή αυτού στο αρχείο.
    puts $f0 "$now [expr $bw0/$time*8/1000000]"
    puts $f1 "$now [expr $bw1/$time*8/1000000]"
    # Κάνει την τιμή bytes_ 0
    $sink0 set bytes_ 0
    $sink1 set bytes_ 0
    # Επαναπρογραμματισμός της διαδικασίας.
    $ns at [expr $now+$time] "record"
}
```

Η διαδικασία της καταγραφής θα πρέπει να κληθεί στην αρχή της προσομοίωσης, βάζοντας τις παρακάτω γραμμές στους καθορισμούς των χρόνων των γεγονότων (events).

Αρχή της καταγραφής δεδομένων

```
$ns at 0.0 "record"
```

Επίσης, πρέπει να εισαχθούν και οι παρακάτω γραμμές για τη δημιουργία του αρχείου του *Xgraph*, μετά τη δημιουργία αρχείου εγγραφής για το NAM.

Δημιουργία του αρχείου εγγραφής που θα χρησιμοποιηθεί με το *Xgraph*

```
set f0 [open lab2a0.tr w]
```

```
set f1 [open lab2a1.tr w]
```

Τέλος, στη διαδικασία κλεισίματος των αρχείων θα πρέπει να προστεθούν οι εξής εντολές:

Κλείσιμο των αρχείων εξόδου

```
close $f0
```

```
close $f1
```

Μπορείτε να δείτε ταυτόχρονα τις γραφικές παραστάσεις του ρυθμού μετάδοσης των δεδομένων που συνέλεξαν και οι δύο *sink agents* για τις δύο ροές πληκτρολογώντας την εντολή: “`xgraph lab2a0.tr lab2a1.tr`”.

1.7 Ερωτήσεις

- Επαληθεύστε τις απαντήσεις της *Ενότητας 1.5* (για ουρά DropTail και ουρά SFQ), χρησιμοποιώντας τις γραφικές παραστάσεις του *Xgraph*.
- Ποια είναι η μέγιστη και ποια η ελάχιστη τιμή του ρυθμού μεταφοράς των δεδομένων για τις δυο ροές;
- Στις γραφικές παραστάσεις του *Xgraph* υπάρχουν διαστήματα με μηδενικό ρυθμό μετάδοσης και διαστήματα που η μια πηγή από μόνη της φθάνει τα 2 Mbps. Πώς ερμηνεύετε αυτές τις παρατηρήσεις;

Ολοκληρωμένος κώδικας για την Εργαστηριακή Άσκηση 2, Ενότητα 1

```
set ns [new Simulator]

$ns color 1 Blue
$ns color 2 Red

set nf [open lab2a.nam w]
$ns namtrace-all $nf

set f0 [open lab2a0.tr w]
set f1 [open lab2a1.tr w]

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n3 $n2 2Mb 10ms DropTail

$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
# Προσδιορισμός της θέσης της γραμμής που αναπαριστά την ουρά στο NAM
$ns duplex-link-op $n2 $n3 queuePos 0.5

proc record {} {
    global sink0 sink1 f0 f1
    set ns [Simulator instance]
    set time 0.1
    set bw0 [$sink0 set bytes_]
    set bw1 [$sink1 set bytes_]
    set now [$ns now]
    puts $f0 "$now [expr $bw0/$time*8/1000000]"
    puts $f1 "$now [expr $bw1/$time*8/1000000]"
    $sink0 set bytes_ 0
    $sink1 set bytes_ 0
    $ns at [expr $now+$time] "record"
}

proc finish {} {
    global ns nf f0 f1
    $ns flush-trace
    close $nf
    close $f0
    close $f1
    exit 0
}

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
$udp0 set class_ 1

set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 1500
$cbr0 set interval_ 0.01
$cbr0 attach-agent $udp0

set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
$udp1 set class_ 2
```

```
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 1500
$cbr1 set interval_ 0.01
$cbr1 attach-agent $udp1
```

```
set sink0 [new Agent/LossMonitor]
$ns attach-agent $n3 $sink0
```

```
set sink1 [new Agent/LossMonitor]
$ns attach-agent $n3 $sink1
```

```
$ns connect $udp0 $sink0
$ns connect $udp1 $sink1
```

```
$ns at 0.0 "record"
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 3.0 "$cbr1 stop"
$ns at 3.5 "$cbr0 stop"
$ns at 4.0 "finish"
```

```
$ns run
```


2. Δυναμική συμπεριφορά δικτύου

Στο δεύτερο μέρος της Άσκησης 2 θα παρουσιάσουμε ένα παράδειγμα δυναμικού δικτύου, όπου η δρομολόγηση αναπροσαρμόζεται όταν κοπεί κάποια ζεύξη. Κατά την πορεία θα σας δείξουμε πώς μπορείτε να διατηρήσετε ένα μεγαλύτερο αριθμό κόμβων σε ένα *Tcl array* αντί να δώσετε σε κάθε κόμβο το δικό του όνομα.

2.1 Δημιουργία μεγαλύτερης τοπολογίας

Δημιουργήστε ένα *Tcl script* γι' αυτή την άσκηση με όνομα 'lab2b.tcl'. Μπορείτε να μεταφέρετε το αρχικό *template* από την *Εργαστηριακή Άσκηση 1* σ' αυτό το αρχείο. Σημειώνεται πάλι, ότι για επαλήθευση και διόρθωση τυχόν λαθών, ο κώδικας υπάρχει ολοκληρωμένος στο τέλος της άσκησης.

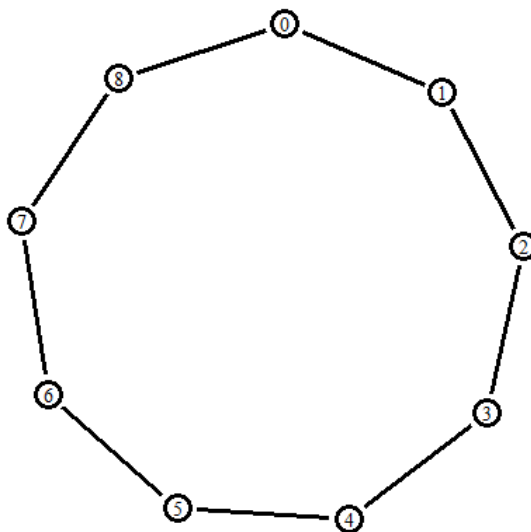
Όπως πάντα, πρέπει πρώτα να δημιουργηθεί η τοπολογία, αν και αυτή τη φορά ακολουθείται άλλη προσέγγιση που θα τη βρείτε πιο βολική, όταν χρειάζεται να δημιουργείτε μεγαλύτερες τοπολογίες. Ο ακόλουθος κώδικας δημιουργεί εννέα κόμβους και τους αποθηκεύει στο *array* "n".

```
for {set i 0} {$i < 9} {incr i} {  
    set n($i) [$ns node]  
}
```

Θα πρέπει να έχετε ήδη συναντήσει βρόχους "for" σε άλλες γλώσσες προγραμματισμού και σίγουρα καταλαβαίνετε αμέσως τη δομή. Σημειώστε ότι τα arrays, όπως ακριβώς και οι άλλες μεταβλητές στην *Tcl*, δεν πρέπει να διευκρινιστούν από την αρχή. Στη συνέχεια συνδέουμε τους κόμβους για να δημιουργηθεί κυκλική τοπολογία. Ο κώδικας που ακολουθεί ίσως σας φαίνεται αρχικά λίγο πολύπλοκος.

```
for {set i 0} {$i < 9} {incr i} {  
    $ns duplex-link $n($i) $n([expr ($i+1)%9]) 2Mb 10ms Droptail  
}
```

Ο βρόχος "for" συνδέει κάθε κόμβο με τον επόμενο του στο *array*, με εξαίρεση τον τελευταίο κόμβο ο οποίος συνδέεται με τον πρώτο κόμβο. Για να γίνει αυτό χρησιμοποιείται ο τελεστής "%" (modulo). Όταν τρέξετε το *script*, η τοπολογία θα φαίνεται αρχικά λίγο περίεργη στο NAM, αλλά αφού κάνετε κλικ στο κουμπί "re-layout" θα φαίνεται όπως στην Εικόνα 4.



Εικόνα 4 – Αναπαράσταση της κυκλικής τοπολογίας στο NAM

2.2 Διακοπή ζεύξης

Το επόμενο βήμα είναι να σταλούν δεδομένα από τον κόμβο “n0” στον κόμβο “n3”.

Δημιουργία ενός UDP agent και «προσκόλλησή» του στον κόμβο «n0»

```
set udp0 [new Agent/UDP]
```

```
$ns attach-agent $n(0) $udp0
```

Δημιουργία μιας πηγής κίνησης CBR traffic source και «τοποθέτησή» της στον «udp0»

```
set cbr0 [new Application/Traffic/CBR]
```

Προσδιορισμός της κίνησης δεδομένων που παράγεται στον κόμβο n0

```
$cbr0 set packetSize_ 1500
```

```
$cbr0 set interval_ 0.01
```

```
$cbr0 attach-agent $udp0
```

Δημιουργία του agent sink0 για τη λήψη δεδομένων και «προσάρτησή» του στον κόμβο «n3»

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

```
$ns attach-agent $n(3) $sink0
```

Σύνδεση του CBR agent με τον Sink agent

```
$ns connect $udp0 $sink0
```

Αρχή γεννήτριας κίνησης

```
$ns at 0.5 "$cbr0 start"
```

Τέλος γεννήτριας κίνησης

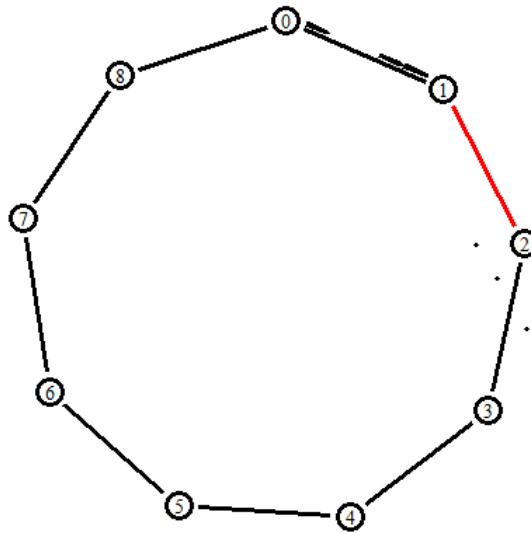
```
$ns at 3.5 "$cbr0 stop"
```

Ο παραπάνω κώδικας θα πρέπει να σας είναι τώρα γνώριμος. Η μόνη διαφορά ως προς τις τελευταίες ενότητες είναι ότι τώρα χρησιμοποιούνται στοιχεία του *array* των κόμβων. Αν αρχίσετε το *script*, θα δείτε ότι η κίνηση ακολουθεί τη συντομότερη διαδρομή από τον κόμβο “0” στον κόμβο “3” μέσω των κόμβων “1” και “2”, όπως θα αναμενόταν. Προσθέτουμε τώρα ένα ενδιαφέρον χαρακτηριστικό. Κάνουμε τη ζεύξη μεταξύ των κόμβων “1” και “2” (που χρησιμοποιείται για τη μετάδοση) να διακοπεί για *1 sec*.

```
$ns rtmodel-at 1.0 down $n(1) $n(2)
```

```
$ns rtmodel-at 2.0 up $n(1) $n(2)
```

Μπορείτε τώρα να αρχίσετε το *script* πάλι και θα δείτε ότι μεταξύ *1.0* και *2.0 sec* η ζεύξη θα διακοπεί και όλα τα δεδομένα που στέλνονται από τον κόμβο “0” χάνονται (Εικόνα 5).

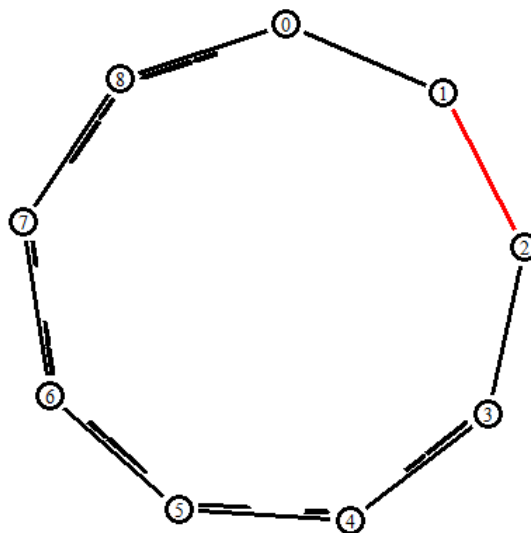


Εικόνα 5 – Αναπαράσταση της διακοπής ζεύξης μεταξύ 1 και 2

Θα δείτε τώρα πώς να χρησιμοποιείτε τη δυναμική δρομολόγηση για την επίλυση αυτού του προβλήματος. Προσθέστε την επόμενη γραμμή στην αρχή του *Tcl script*, μετά τη δημιουργία του *object* προσομοίωσης.

```
$ns rtproto DV
```

Αρχίστε πάλι την προσομοίωση και θα δείτε ότι, αρχικά, ένας μεγάλος αριθμός μικρών πακέτων κυκλοφορούν στο δίκτυο. Αν ελαττώσετε αρκετά την ταχύτητα του NAM για να κάνετε κλικ πάνω στα πακέτα, θα δείτε ότι υπάρχουν πακέτα “rtproto DV” (routing protocol distance vector) τα οποία χρησιμοποιούνται για την ανταλλαγή πληροφορίας δρομολόγησης μεταξύ των κόμβων. Όταν η ζεύξη διακοπεί στο *1.0 sec*, η δρομολόγηση θα ενημερωθεί και η κίνηση θα αναδρομολογηθεί μέσω των κόμβων “8”, “7”, “6”, “5” και “4” (Εικόνα 6).



Εικόνα 6 – Αναδρομολόγηση των πακέτων της ζεύξης 0-3

2.3 Ερωτήσεις

- Περιγράψτε με απλά λόγια τη διαδικασία που έλαβε χώρα στο παραπάνω *animation*. Από τι εξαρτάται η δυναμική συμπεριφορά του δικτύου;
- Γιατί ο κόμβος 0 συνεχίζει να στέλνει πακέτα στον κόμβο 1 για κάποια *msecs* ενώ έχει πέσει η σύνδεση μεταξύ των κόμβων 1 και 2;

- Για ποιο λόγο σταματάει και αλλάζει τη δρομολόγηση προς τον κόμβο 8;

2.4 Προσομοίωση στο Xgraph

Για να δείτε τα αποτελέσματα και σε μορφή γραφικής παράστασης με το *Xgraph*, προσθέστε την παρακάτω διαδικασία στο *script* και κάνετε τις απαραίτητες αλλαγές στον υπόλοιπο κώδικα.

Διαδικασία (procedure) καταγραφής της κίνησης

```
proc record {} {
    global sink0 f0
    set ns [Simulator instance]
    # Ορισμός της χρονικής περιόδου που θα ξανακληθεί η διαδικασία
    set time 0.2
    # Καταγραφή των bytes
    set bw0 [$sink0 set bytes_]
    # Ορισμός του χρόνου της τρέχουσας καταγραφής
    set now [$ns now]
    # Υπολογισμός του bandwidth και καταγραφή αυτού στο αρχείο
    puts $f0 "$now [expr $bw0/$time*8/1000000]"
    # Κάνει την τιμή bytes_ 0
    $sink0 set bytes_ 0
    # Επαναπρογραμματισμός της διαδικασίας
    $ns at [expr $now+$time] "record"
}
```

2.5 Ερωτήσεις

- Συγκρίνετε το *animation* με τη γραφική παράσταση. Είναι κατά τη γνώμη σας σωστό το γράφημα;
- Προσπαθήστε να το βελτιώσετε αλλάζοντας μία από τις μεταβλητές της διαδικασίας “record”. Να σχεδιάσετε, με το *Xgraph*, δύο βελτιωμένα γραφήματα για δύο διαφορετικές τιμές της μεταβλητής που θα αλλάξετε;
- Εξηγήστε το σχήμα της γραφικής παράστασης σε συνάρτηση με τα γεγονότα και τις παραμέτρους του δικτύου.

Ολοκληρωμένος κώδικας για την Εργαστηριακή Άσκηση 2, Ενότητα 2

```
set ns [new Simulator]

$ns rtproto DV
set nf [open lab2b.nam w]
$ns namtrace-all $nf
set f0 [open lab2b.tr w]

proc record {} {
    global sink0 f0
    set ns [Simulator instance]
    set time 0.2
    set bw0 [$sink0 set bytes_]
    set now [$ns now]
    puts $f0 "$now [expr $bw0/$time*8/1000000]"
    $sink0 set bytes_ 0
    $ns at [expr $now+$time] "record"
}

proc finish {} {
    global ns nf f0
    $ns flush-trace
    close $nf
    close $f0
    exit 0
}

for {set i 0} {$i < 9} {incr i} {
    set n($i) [$ns node]
}
for {set i 0} {$i < 9} {incr i} {
    $ns duplex-link $n($i) $n([expr ($i+1)%9]) 2Mb 10ms DropTail
}
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize_ 1500
$cbr0 set interval_ 0.01
$cbr0 attach-agent $udp0

set sink0 [new Agent/LossMonitor]
$ns attach-agent $n(3) $sink0
$ns connect $udp0 $sink0

$ns at 0.0 "record"
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
$ns at 3.5 "$cbr0 stop"
$ns at 4.0 "finish"

$ns run
```