Πανεπιστήμιο Πατρών
Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Ακαδημαϊκό έτος 2012-2013

(ΒΑΘΜΟΣ ΆΣΚΗΣΗΣ: 10)

Εργαστήριο Δικτύων Υπολογιστών
Αναφορά 4ης Άσκησης

Αφροδίτη Αλεβιζοπούλου ΑΜ: 3879

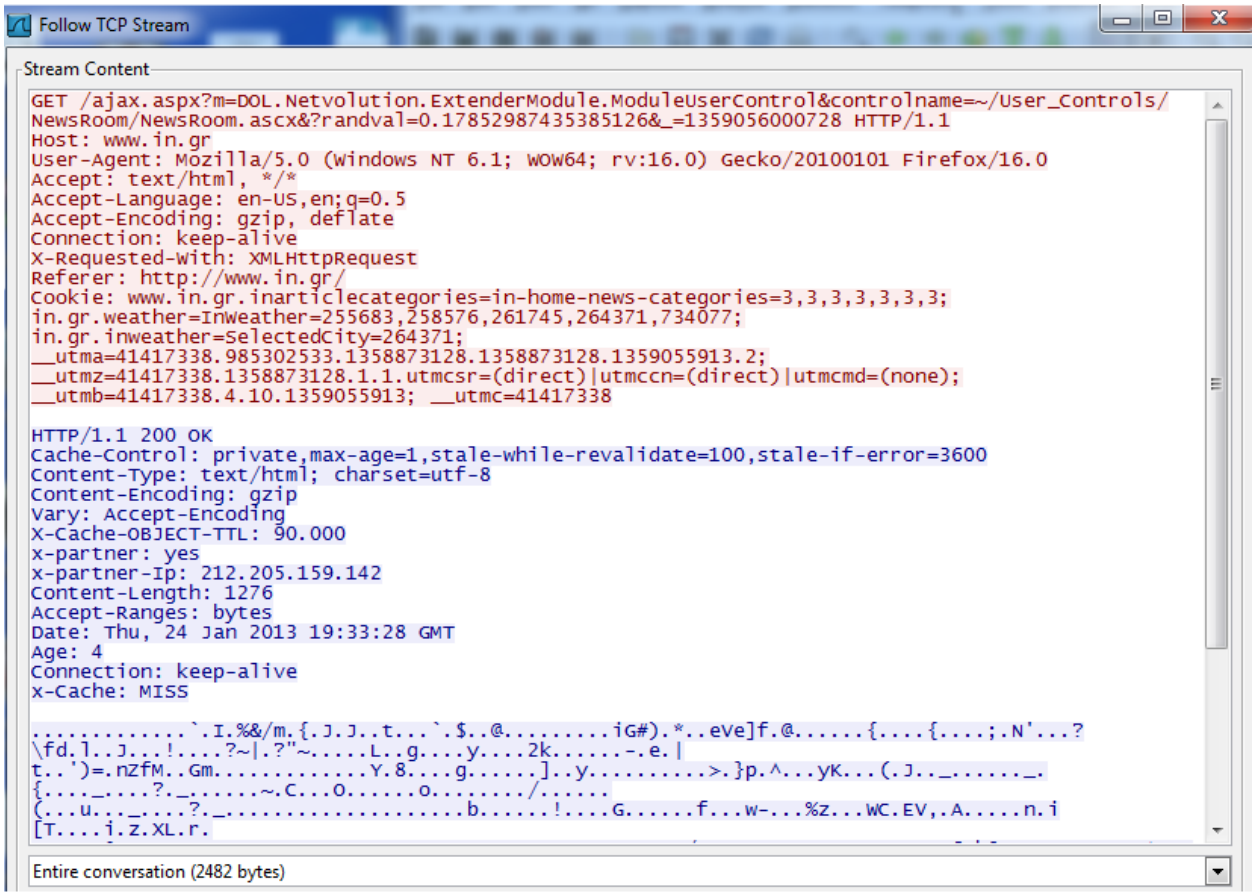
Ομάδα NETLAB 39

Πρώτο Ερώτημα:

Ως browser για το πρώτο ερώτημα της άσκησης χρησιμοποιήσαμε τον firefox (έκδοση 16.0) και με το εργαλείο Wireshark κάναμε capture τα πακέτα όταν ζητήσαμε να «κατέβει» η σελίδα www.in.gr .

Το trace αρχείο με τις ρυθμίσεις network.http.keep-alive:true και network.http.version:1.1 είναι το **trace1.pcapng**.

Παραθέτουμε μια εικόνα του τι προκύπτει κάνοντας **follow tcp stream** στα πακέτα που ανταλλάχθηκαν με τον server (στο trace αρχείο είναι η IP 212.205.159.143) :



Με κόκκινο χρώμα φαίνεται το request μας στον server, ύστερα το όνομα του host (www.in.gr) και τέλος τα headers.

Με μπλε χρώμα φαίνεται το response του server (HTTP/1.1 200 OK), μετά τα headers του response και τέλος το περιεχόμενο της απάντησης που είναι τύπου text/html, όπως φαίνεται από το header.

Τόσο από την δική μας πλευρά όσο και από την πλευρά του server, παρατηρούμε ότι υπάρχει το header Connection:keep-alive, δηλαδή όταν στήνεται το tcp connection έχουμε ένα σετ από timers που ασχολούνται με την διαδικασία keepalive.

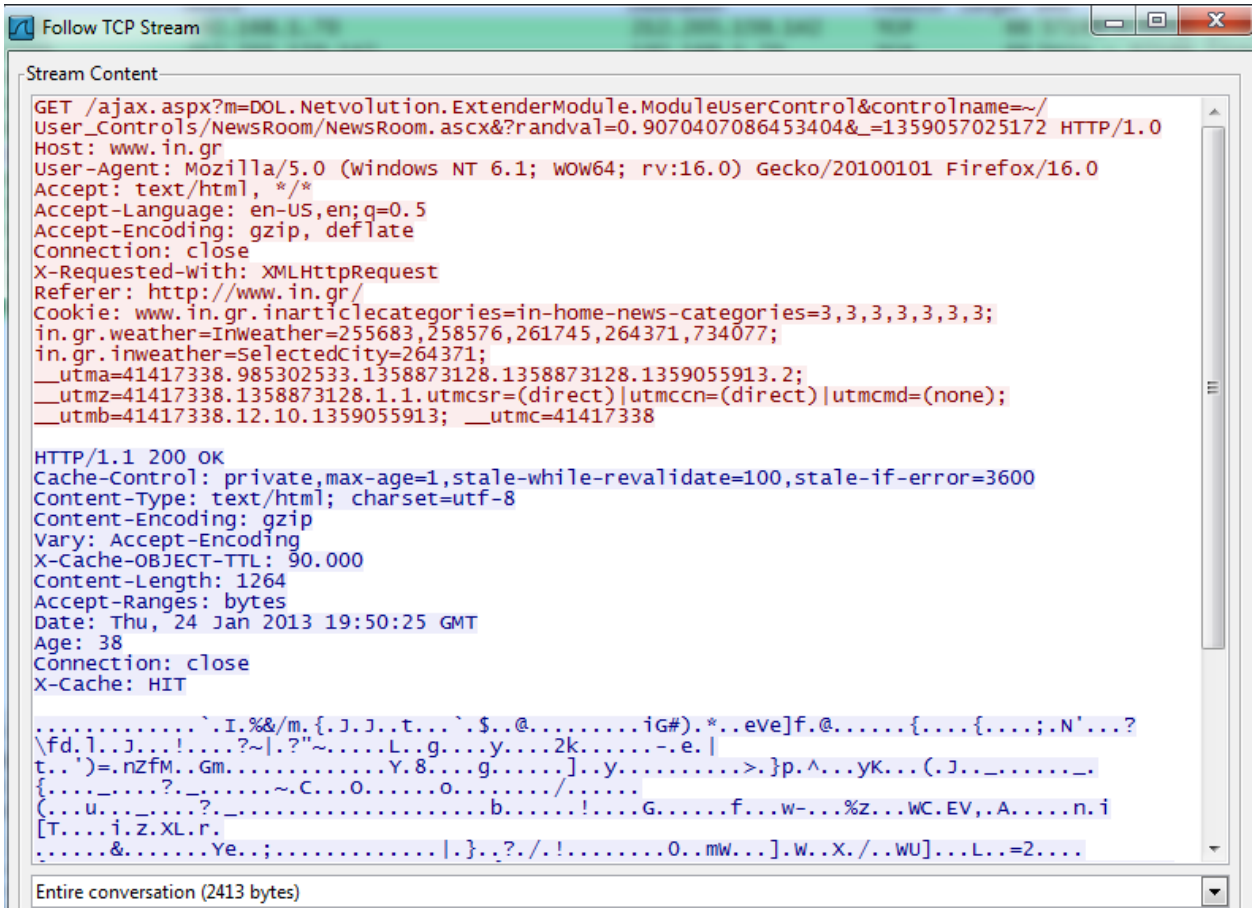
Όταν ο keepalive timer φτάσει 0, στέλνουμε στον απομακρυσμένο host ένα “keepalive packet” που δεν περιέχει δεδομένα,αλλά μόνο το flag ACK και αυτός απαντά με ένα πακέτο που, επίσης, δεν περιέχει δεδομένα αλλά μόνο το flag ACK.

Ο απομακρυσμένος Host δεν χρειάζεται να υποστηρίζει keepalive connections για να γίνει αυτό.

Όταν δεχόμαστε απάντηση “keepalive”, τότε θεωρούμε οτι το connection είναι ακομα “up” και το TCP μας επιτρέπει ουσιαστικά να έχουμε πολλαπλά requests και responses με ένα TCP connection.

Το trace αρχείο με τις ρυθμίσεις network.http.keep-alive:false και network.http.version:1.0 είναι το **trace2.pcapng**.

Παραθέτουμε μια εικόνα του τι προκύπτει κάνοντας **follow tcp stream** στα πακέτα που ανταλλάχθηκαν με τον server (στο trace αρχείο είναι η IP 212.205.159.142) :



Εδώ παρατηρούμε πως το header Connection είναι Close, τόσο στο request όσο και στο response.

Χωρίς το keepalive connection, για κάθε request γίνεται ένα TCP Connection. Δηλαδή, μετά την αποστολή της απάντησης από τον server, η σύνδεση κλείνει.

Δεύτερο Ερώτημα:

Ο κώδικας για τη $2^{η}$ άσκηση βρίσκεται στο αρχείο **static_udp.tcl** (στατική δρομολόγηση) και **dynamic_udp.tcl** (δυναμική δρομολόγηση). Παρακάτω, παραθέτουμε τον κώδικα μαζί με επαρκή σχολιασμό και επεξηγήσεις και καταγράφουμε τι παρατηρήσαμε, αναλύοντας τα αποτελέσματα που περιέχονται στα δύο παραγόμενα trace αρχεία του πειράματος.

 # Δημιουργούμε ένα νέο αντικείμενο τύπου Simulator

```
set ns [new Simulator]
```

 # Δημιουργία αρχείων για αποθήκευση nam και trace δεδομένων

```
set nf [open out.nam w]
$ns namtrace-all $nf
set tf [open out.tr w]
$ns trace-all $tf
```

 # Ορίζουμε μια διαδικασία με όνομα 'finish', η οποία κλείνει τα αρχεία καταγραφής

```
proc finish {} {
        global ns nf
        $ns flush-trace
# Close the trace file
        close $nf
        exit 0
}
```

 # Δημιουργούμε 7 κόμβους με ονόματα από n0 μέχρι n6

```
for {set i 0} {$i < 7} {incr i}
{
  set n($i) [$ns node]
}
```

 # Συνδέουμε με link όλους τους κόμβους, ώστε να δημιουργηθεί ένας δακτύλιος

```
for {set i 0} {$i < 7} {incr i}
{
  $ns duplex-link $n($i) $n([expr ($i+1)%7]) 1.5Mb 10ms DropTail
}
```

Ο χαρακτήρας % είναι το υπόλοιπο (modulo). Δηλαδή η παράσταση ($i+1)%7 θα επιστρέφει την τιμή i+1 για όλες τις τιμές του I εκτός από την τιμή 6 (τότε θα επιστρέφει μηδέν). Άρα, οι κόμβοι θα συνδέονται διαδοχικά, εκτός από τον τελευταίο, ο οποίος θα συνδεθεί με τον πρώτο.

Στο επόμενο βήμα θέλουμε να στείλουμε δεδομένα από τον κόμβο n0 στον κόμβο n3. Στον ns-2, τα δεδομένα στέλνονται πάντα από έναν «πράκτορα» (agent) σε έναν άλλο. Επομένως, στη συνέχεια θα δημιουργήσουμε ένα αντικείμενο τύπου agent, το οποίο στέλνει δεδομένα από τον κόμβο n0 και ένα αντικείμενο τύπου agent το οποίο δέχεται τα δεδομένα στον κόμβο n3. Παραθέτουμε τον απαιτούμενο κώδικα:

#Δημιουργία ενός UDP agent και προσάρτησή του στον κόμβο n0

```
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
```

 # Δημιουργία μιας πηγής παραγωγής πακέτων CBR (Constant Bit Rate) και τοποθέτησή της στον udp0

```
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
```

Το πρωτόκολλο εφαρμογής CBR αποτελεί μια πηγή σταθερού δικτυακού φόρτου (δημιουργεί με σταθερό ρυθμό UDP πακέτα). Το μέγεθος του κάθε πακέτου θα είναι 500 bytes (packetSize) και θα αποστέλλονται κάθε 0.005 δευτερόλεπτα (interval).

 # Δημιουργία agent τύπου Null («απαγωγός» πακέτων) και προσάρτησή του στον κόμβο n3

```
set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0
```

 # Σύνδεση των δύο agents μεταξύ τους

```
$ns connect $udp0 $null0
```

 # Ενημέρωση του CBR agent για την έναρξη και τον τερματισμό αποστολής δεδομένων

```
$ns at 0.1 "$cbr0 start"
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
$ns at 4.5 "$cbr0 stop"
```

Δηλαδή, ο κόμβος n0 θα αρχίσει να στέλνει δεδομένα στον κόμβο n3 από τη χρονική στιγμή **0.1 sec** μέχρι τη στιγμή **4.5 sec** και το link ανάμεσα στον κόμβο n1 και n2 θα πάψει να λειτουργεί το χρονικό διάστημα ανάμεσα στο 1^{ο} και στο 2^{ο} δευτερόλεπτο**.**

 # Τερματισμός προσομοίωσης στα 5.0 sec

$ns at 5.0 "finish"

 # Εκκίνηση προσομοίωσης

$ns run

Σώζουμε τον κώδικα στα αρχείο **static_udp.tcl** και εκτελούμε την προσομοίωση μέσω της εντολής:

ns static_udp.tcl

Με την εκτέλεση της εντολής αυτής, δημιουργούνται και τα αρχεία **out.nam** και **out.tr**

Το παραγόμενο αρχείο ίχνους **out.tr** που δημιουργήθηκε, περιέχει πληροφορίες της μορφής:

```
+ 0.1 0 1 cbr 500 ------- 0 0.0 3.0 0 0
- 0.1 0 1 cbr 500 ------- 0 0.0 3.0 0 0
+ 0.105 0 1 cbr 500 ------- 0 0.0 3.0 1 1
- 0.105 0 1 cbr 500 ------- 0 0.0 3.0 1 1
+ 0.11 0 1 cbr 500 ------- 0 0.0 3.0 2 2
- 0.11 0 1 cbr 500 ------- 0 0.0 3.0 2 2
r 0.112667 0 1 cbr 500 ------- 0 0.0 3.0 0 0
```

Κάθε γραμμή του αρχείου αυτού αντιστοιχεί σε ένα γεγονός που συνέβη κατά τη διάρκεια της προσομοίωσης:

- Ο πρώτος χαρακτήρας κάθε γραμμής υποδηλώνει το ***είδος του γεγονότος*** που συνέβη. Ο χαρακτήρας "+" σημαίνει είσοδο του πακέτου σε ουρά αναμονής, ο χαρακτήρας "-" σημαίνει αποχώρηση από ουρά αναμονής, ο χαρακτήρας "r" σημαίνει επιτυχημένη λήψη πακέτου και ο χαρακτήρας "d" σημαίνει απόρριψη πακέτου.
- Η δεύτερη λέξη της κάθε γραμμής είναι η ***χρονική στιγμή*** κατά την οποία συνέβη το γεγονός που καταγράφεται.
- Οι επόμενες δύο λέξεις περιγράφουν ***μεταξύ ποιων κόμβων*** βρίσκεται το πακέτο.
- Η πέμπτη λέξη είναι το ***είδος του πακέτου***.
- Η έκτη λέξη περιγράφει το ***μέγεθος του πακέτου***.
- Οι παύλες αντιστοιχούν σε πεδία που δεν χρησιμοποιούνται στο παράδειγμα. Ο πρώτος αριθμός μετά τις παύλες είναι το flow ID της ροής στην οποία ανήκει το πακέτο. Η τιμή του πεδίου αυτού

καθορίζεται από εντολές της μορφής $tcpX set fid_ Y. Το flow ID ακολουθείται από την διεύθυνση αποστολέα και προορισμού (IP.port), από τον αύξοντα αριθμό (sequence number) του πακέτου και τέλος από έναν μοναδικό αριθμό (unique number) του πακέτου.

Στο σημείο αυτό, παρατηρώντας το trace αρχείο out.tr, βλέπουμε στη **γραμμή 1** ότι ο κόμβος n0 αρχίζει να στέλνει δεδομένα στον κόμβο n3 ακολουθώντας τη συντομότερη διαδρομή, μέσω των κόμβων n1 και n2. Αυτό συμβαίνει μέχρι τη **γραμμή 1592**, όπου παρατηρείται και η τελευταία (μέχρι εκείνο το σημείο) επιτυχής λήψη πακέτου από τον κόμβο n3:

Γραμμή 1592: r 0.998 2 3 cbr 500 ------- 0 0.0 3.0 172 172

Στη **γραμμή 1593** μπαίνουμε στο 1^{ο} δευτερόλεπτο, οπότε η ζεύξη μεταξύ των κόμβων n1 και n2 διακόπτεται (Συγκεκριμένα, κόβεται το link του κόμβου n2 προς τον κόμβο n1). Στη γραμμή αυτή, αλλά και στην επόμενή της βλέπουμε το μήνυμα:

Γραμμή 1593: v 1 link-down 2 1

Γραμμή 1594: v 1 link-down 2 1

Στις **γραμμές 1595, 1596 και 1597** βλέπουμε τα μηνύματα απόρριψης πακέτων από τον κόμβο n1 προς τον κόμβο n2:

Γραμμή 1595: d 1 1 2 cbr 500 ------- 0 0.0 3.0 175 175

Γραμμή 1596: d 1 1 2 cbr 500 ------- 0 0.0 3.0 176 176

Γραμμή 1597: d 1 1 2 cbr 500 ------- 0 0.0 3.0 177 177

Στις **γραμμές 1598 και 1599** βλέπουμε ότι κόβεται το link του κόμβου n1 προς τον κόμβο n2:

Γραμμή 1598: v 1 link-down 1 2

Γραμμή 1599: v 1 link-down 1 2

Από τη **γραμμή 1600** έως και **τη γραμμή 2203** όλα τα δεδομένα που στέλνονται από τον κόμβο n0 χάνονται, καθώς δεν μπορούν να μεταδωθούν πέρα απ'τον κόμβο n1.

Στη **γραμμή 2202** μπαίνουμε στο 2^{ο} δευτερόλεπτο.

Στις **γραμμές 2204, 2205, 2206 και 2207** επανέρχεται η σύνδεση μεταξύ των κόμβων n1 και n2:

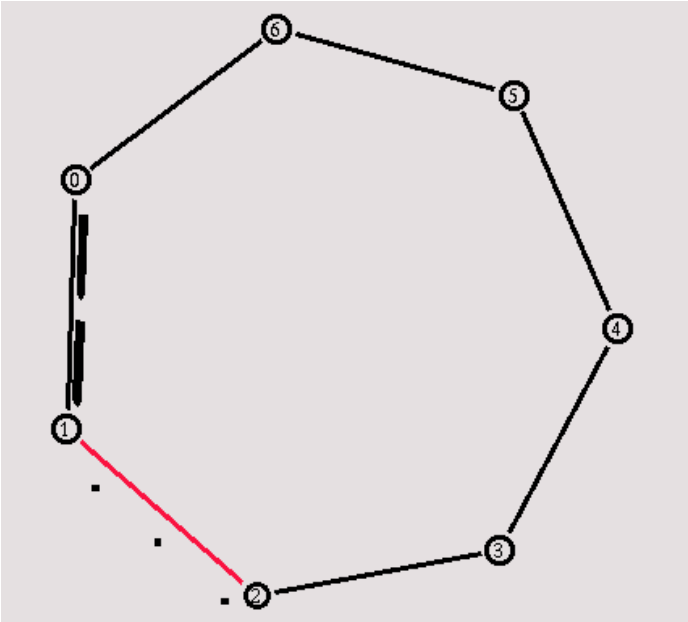
Γραμμή 2204: v 2 link-up 2 1

Γραμμή 2205: v 2 link-up 2 1

Γραμμή 2206: v 2 link-up 1 2

Γραμμή 2207: v 2 link-up 1 2

Από τη **γραμμή 2208** και μετά παρατηρούμε ότι ξεκινάει ξανά η μετάδοση πακέτων από τον κόμβο n0 προς τον κόμβο n3 (μέσω των κόμβων n1 και n2). Η μετάδοση συνεχίζεται μέχρι τέλους, δηλ. μέχρι τη **γραμμή 6728**.



Εικόνα 1 – Αναπαράσταση της διακοπής ζεύξης μεταξύ n1 και n2

Στη συνέχεια, εκτελούμε την παρακάτω εντολή, θέλοντας αρχικά να κρατήσουμε μόνο τις γραμμές του αρχείου out.tr που αναφέρονται σε επιτυχημένη λήψη πακέτου (δηλ. περιέχουν τη λέξη "r") και έπειτα, απ'αυτές τις γραμμές να μας τυπώσει το πλήθος των γραμμών όπου η επιτυχημένη λήψη πακέτου πραγματοποιήθηκε μεταξύ των κόμβων n2 και n3:

grep -w 'r' out.tr | grep -c '2 3'

Το αποτέλεσμα που παίρνουμε με την εκτέλεση της παραπάνω εντολής είναι η τιμή **692**, πράγμα που σημαίνει πως ο κόμβος n3 έλαβε επιτυχώς 692 πακέτα, μέσω στατικής δρομολόγησης.

Για να κάνουμε χρήση δυναμικής δρομολόγησης προσθέτουμε την εντολή:

$ns rtproto DV

ακριβώς μετά την εντολή **set ns [new Simulator]**.

Αποθηκεύουμε τον κώδικα στο αρχείο **dynamic_udp.tcl** και εκτελούμε την προσομοίωση μέσω της εντολής:

ns dynamic_udp.tcl

Με την εκτέλεση της εντολής αυτής, δημιουργούνται και τα αρχεία **out.nam** και **out.tr**

Στο σημείο αυτό, παρατηρώντας το trace αρχείο out.tr, βλέπουμε στη **γραμμή 1** ότι ένας μεγάλος αριθμός μικρών πακέτων τύπου "rtproto DV" κυκλοφορεί στο δίκτυο. Τα πακέτα αυτά χρησιμοποιούνται για να ανταλλάσσονται πληροφορίες δρομολόγησης μεταξύ των κόμβων. Παράλληλα, στη **γραμμή 169** (0.1 sec) ο κόμβος n0 αρχίζει να στέλνει "cbr" πακέτα δεδομένων στον κόμβο n3 ακολουθώντας τη συντομότερη διαδρομή, μέσω των κόμβων n1 και n2. Η αποστολή "cbr" πακέτων συμβαίνει μέχρι και τη **γραμμή 1784**, όπου παρατηρείται και η τελευταία (μέχρι εκείνο το σημείο) επιτυχής λήψη πακέτου από τον κόμβο n3:

Γραμμή 1784: r 0.998 2 3 cbr 500 ------- 0 0.0 3.0 172 236

Στη **γραμμή 1785** μπαίνουμε στο 1^{ο} δευτερόλεπτο, οπότε η ζεύξη μεταξύ των κόμβων n1 και n2 διακόπτεται. (Συγκεκριμένα, κόβεται το link του κόμβου n2 προς τον κόμβο n1). Στη γραμμή αυτή, αλλά και στην επόμενή της βλέπουμε το μήνυμα:

Γραμμή 1785: v 1 link-down 2 1

Γραμμή 1786: v 1 link-down 2 1

Στις **γραμμές 1787, 1788 και 1789** βλέπουμε τα μηνύματα απόρριψης πακέτων από τον κόμβο n1 προς τον κόμβο n2:

Γραμμή 1787: d 1 1 2 cbr 500 ------- 0 0.0 3.0 175 239

Γραμμή 1788: d 1 1 2 cbr 500 ------- 0 0.0 3.0 176 240

Γραμμή 1789: d 1 1 2 cbr 500 ------- 0 0.0 3.0 177 241

Στις **γραμμές 1790 και 1791** βλέπουμε ότι κόβεται το link του κόμβου n1 προς τον κόμβο n2:

Γραμμή 1790: v 1 link-down 1 2

Γραμμή 1791: v 1 link-down 1 2

Από τη **γραμμή 1792** έως και **τη γραμμή 4173** συνεχίζεται κανονικά η μετάδοση "rtproto DV" πακέτων μεταξύ των κόμβων του δικτύου. Επίσης, παρόλο που η ζεύξη μεταξύ των κόμβων n1 και n2 έχει διακοπεί, η δρομολόγηση ενημερώνεται (μέσω των "rtproto DV" πακέτων), αναπροσαρμόζεται και

ανακαλύπτει αυτόματα και χρησιμοποιεί εναλλακτική διαδρομή για τα πακέτα, μέσω των κόμβων “n6”, “n5” και “n4”. Έτσι, συνεχίζεται κανονικά η μετάδοση πακέτων από τον κόμβο “n1” στον κόμβο “n3”.

Για παράδειγμα, στη γραμμή 1932 βλέπουμε μια επιτυχή μετάδοση πακέτου από τον κόμβο “n4” στον κόμβο “n3”:

Γραμμή 1932: r 1.065667 4 3 cbr 500 ------- 0 0.0 3.0 183 253

Στη **γραμμή 4174** μπαίνουμε στο 2^{ο} δευτερόλεπτο.

Στις **γραμμές 4176, 4177, 4178 και 4179** επανέρχεται η σύνδεση μεταξύ των κόμβων n1 και n2:

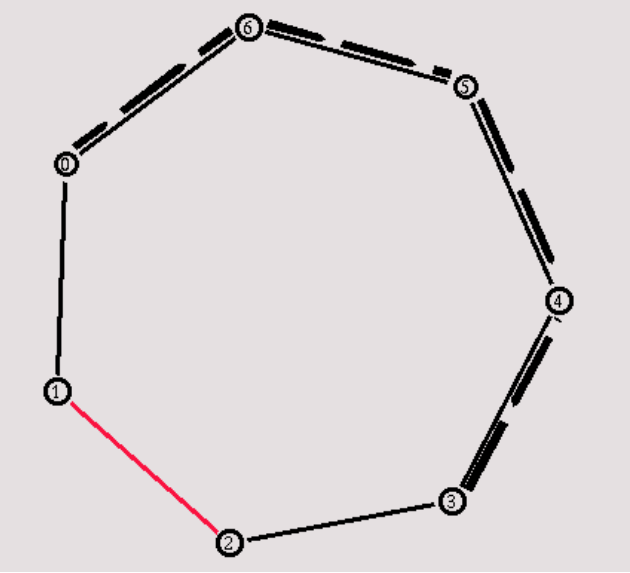
Γραμμή 4176: v 2 link-up 2 1

Γραμμή 4177: v 2 link-up 2 1

Γραμμή 4178: v 2 link-up 1 2

Γραμμή 4179: v 2 link-up 1 2

Από τη **γραμμή 4180** και μετά συνεχίζεται η μετάδοση “rtproto DV” πακέτων μεταξύ των κόμβων του δικτύου. Επίσης, εφόσον επανήλθε η σύνδεση μεταξύ των κόμβων “n1” και “n2”, η δρομολόγηση ενημερώνεται ξανά και έτσι, όλες οι μεταδόσεις πακέτων από τον κόμβο “n0” προς τον κόμβο “n3” γίνονται ξανά μέσω των κόμβων n1 και n2 (=συντομότερη διαδρομή). Η μετάδοση συνεχίζεται μέχρι τέλους, δηλ. μέχρι τη **γραμμή 8891**.



Εικόνα 2 - Αναδρομολόγηση των πακέτων της ζεύξης n0-n3

Στη συνέχεια, εκτελούμε την παρακάτω εντολή, θέλοντας αρχικά να κρατήσουμε μόνο τις γραμμές του αρχείου out.tr που αναφέρονται σε επιτυχημένη λήψη πακέτου (δηλ. περιέχουν τη λέξη “r”). Έπειτα, απ’αυτές τις γραμμές κρατάμε μόνο τις γραμμές όπου η επιτυχημένη λήψη πακέτου πραγματοποιήθηκε είτε μεταξύ των κόμβων n2 και n3, είτε μεταξύ των κόμβων n4 και n3. Και τέλος, απ’αυτές τις γραμμές ζητάμε να μας τυπώσει το πλήθος των γραμμών που περιέχουν τη λέξη “cbr” (δηλαδή, θέλουμε να δούμε πόσα cbr πακέτα έφτασαν στον κόμβο n3. Αυτό είναι απαραίτητο γιατί, όπως αναφέραμε και παραπάνω, πλέον κυκλοφορούν όχι μόνο “cbr” πακέτα στο δίκτυό μας, αλλά και πακέτα “rtproto DV”).

grep -w 'r' out.tr | grep '[2 4] 3' | grep -wc 'cbr'

Το αποτέλεσμα που παίρνουμε με την εκτέλεση της παραπάνω εντολής είναι η τιμή **933**, πράγμα που σημαίνει πως ο κόμβος n3 έλαβε επιτυχώς 993 πακέτα, μέσω δυναμικής δρομολόγησης.

Παρατηρούμε πως η δυναμική δρομολόγηση επιλύει το πρόβλημα που δημιουργείται στη στατική δρομολόγηση, όταν υπάρχει διακοπή της ζεύξης μεταξύ των κόμβων n1 και n2, όπου τα πακέτα προς τον κόμβο n3 χάνονται. Έτσι, ο κόμβος n3 έλαβε 241 παραπάνω πακέτα σε σχέση με πριν.