14 / 1 / 2025

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ ΕΡΓΑΣΤΗΡΙΟ

ΕΡΓΑΣΙΑ
ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ
2024-2025

Ηλίας Ξανθόπουλος 58545

Πίνακας τιμών παραμέτρων σύμφωνα με ΑΕΜ 58545 (X=5, Y=4, Z=5)	
Ταχύτητες (bandwidth) των καναλιών	
Sender1 – Router1	5Mbps
Sender2 – Router1	6Mbps
Sender3 – Router2	60Mbps
Router1 – Router2	2Mbps
Router2 – Receiver1	2Mbps
Router2 – Receiver2	4Mbps
Router2 – Receiver3	6Mbps
Καθυστέρηση των καναλιών	20ms
Ο **Sender1** στέλνει στον:	**Receiver2**
Ο **Sender2** στέλνει στον:	**Receiver2**
Ο **Sender3** στέλνει στον:	**Receiver3**
Διάρκεια προσομοίωσης	**20sec**
Αποστολείς και παραλήπτες ακούν στην θύρα:	**1001**
Οι αποστολείς ξεκινούν να στέλνουν τις χρονικές στιγμές:	
Sender1	**1sec**
Sender2	**7sec**
Sender3	**6sec**
Maximum Segment Size του TCP	**1460**
Αρχικό ssthresh και advertised window:	**65535**
Έκδοση TCP	**Tahoe**
Αλλαγή έκδοσης TCP (άσκηση 1.4)	Από **Tahoe** σε **NewReno**
Άσκηση 2	
Μέγεθος **DropTailQueue** και **RedDropperQueue**	**10**

Πίνακας 1

Σχόλιο

Τα αρχεία *lab545.ned* και *omnetpp.ini* έχουν συμπληρωθεί χωρίς τις αλλαγές που συμβαίνουν στις δύο *ασκήσεις*. Οποιαδήποτε αλλαγή στον κώδικα είναι απαραίτητη για τα ζητούμενα των ασκήσεων, θα παρουσιάζεται στο παρών αρχείο μέσω *εικόνων*.

Διορθώσεις/Αλλαγές εκφώνησης

• Το δίκτυο, σύμφωνα με την εκφώνηση, πρέπει να αποτελείται από 6 EthernetHost. Ωστόσο τα κανάλια στα οποία διαδίδουν οι EthernetHost πρέπει να έχουν μια από τις standard ταχύτητες Ethernet, το οποίο δεν ισχύει, όπως φαίνεται από τα bandwidth του *Πίνακα 1*. Για αυτό στην θέση των EthernetHost, χρησιμοποιούνται StandardHost (βλπ. *lab545.ned*), έτσι ώστε να μπορεί να υπάρχει διάδοση στα καθορισμένα datarate.

• Σύμφωνα με την εκφώνηση, το αρχικό ssthresh και advertised window είναι 65536. Αυτή όμως η ρύθμιση, όπως φαίνεται με την εκκίνηση της προσομοίωσης, είναι invalid, για αυτό και επιλέχθηκε ως αρχικό window το 65535 (βλπ. *omnetpp.ini*).

• Στην εκφώνηση δεν ορίζεται το μέγεθος των δεδομένων που πρέπει να στείλει ο κάθε αποστολέας. Για αυτό και, όπως φαίνεται στην γραμμή 17 του αρχείου *omnetpp.ini*, ρυθμίστηκε οι αποστολείς να έχουν να στείλουν δεδομένα 1GB, ένα μέγεθος τόσο μεγάλο, έτσι ώστε να μην τελειώσει η αποστολή των δεδομένων πριν το τέλος της προσομοίωσης.

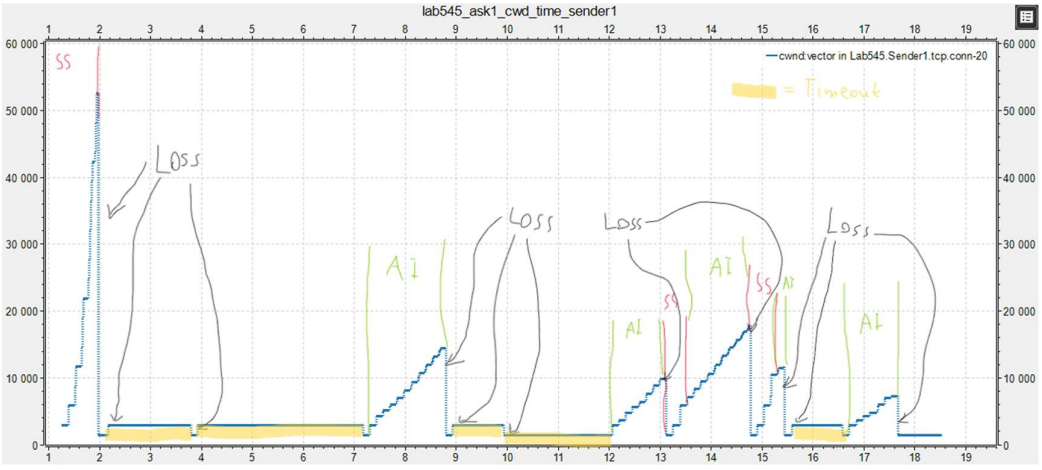
Άσκηση 1

Κάνοντας την αλλαγή της *Εικόνας 1* στον κώδικα του αρχείου *lab545.ned* εισάγουμε ένα ποσοστό σφάλματος πακέτου (Packet Error Rate) στα χαρακτηριστικά του καναλιού που συνδέει τα *Router1* και *Router2* της τάξεως του 4%. Με αυτήν την αλλαγή, κατά την προσομοίωση, κάποια από τα πακέτα που μεταδίδονται μέσω της σύνδεσης Router*1* και *Router2* δεν προωθούνται στις πύλες εξόδου τους και χάνονται.

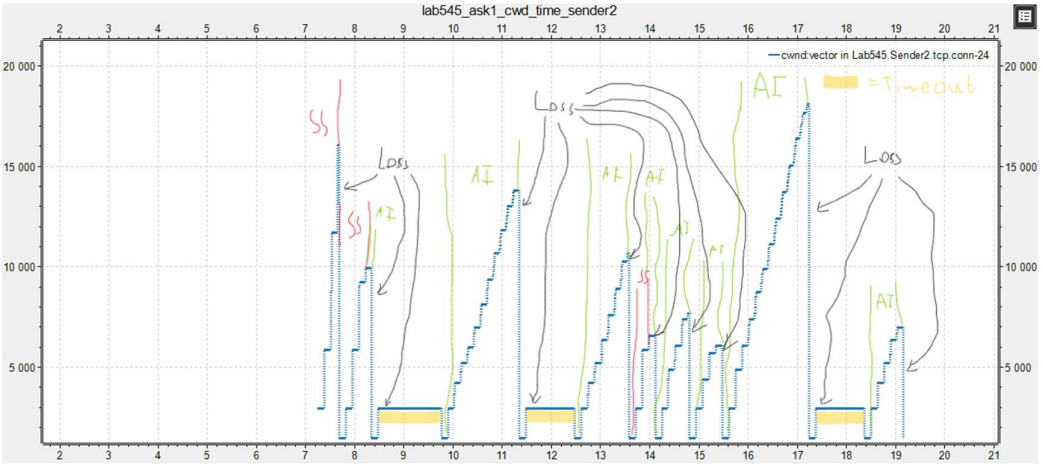
```
57●     connections:
58          // X=5 Y=4 Z=5
59          Sender1.pppg++ <--> {  delay = 20ms; datarate = 5Mbps; } <--> Router1.pppg++;
60          Sender2.pppg++ <--> {  delay = 20ms; datarate = 6Mbps; } <--> Router1.pppg++;
61          Router1.pppg++ <--> {  delay = 20ms; datarate = 2Mbps; per = 4e-2; } <--> Router2.pppg++;
62          Router2.pppg++ <--> {  delay = 20ms; datarate = 60Mbps; } <--> Sender3.pppg++;
63          Router2.pppg++ <--> {  delay = 20ms; datarate = 4Mbps; } <--> Receiver2.pppg++;
64          Router2.pppg++ <--> {  delay = 20ms; datarate = 2Mbps; } <--> Receiver1.pppg++;
65          Router2.pppg++ <--> {  delay = 20ms; datarate = 6Mbps; } <--> Receiver3.pppg++;
66  }
```

Εικόνα 1: Αλλαγή κώδικα "lab545.ned" έτσι ώστε Packet Error Rate 4% (Router1 – Router2)

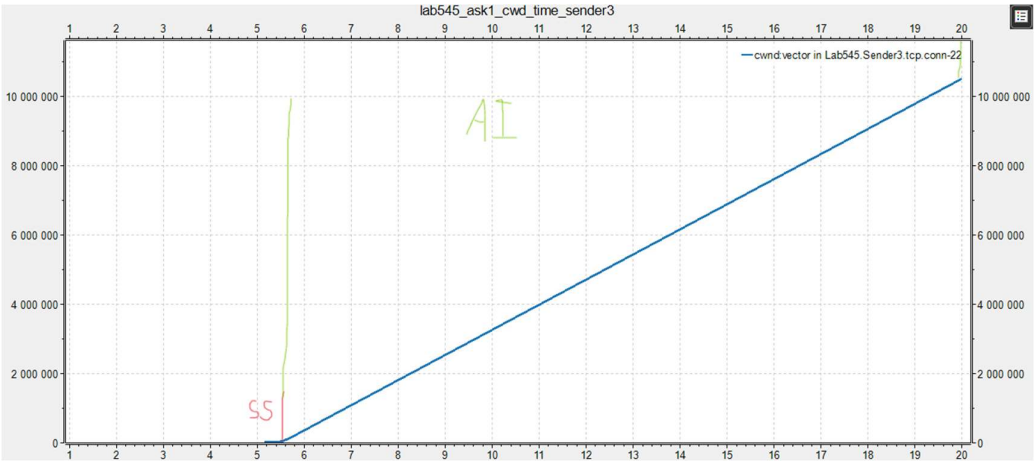
2.a) Τρέχοντας την προσομοίωση **Lab_erg_ask1** με τις παραμέτρους του *πίνακα 1* και την αλλαγή της *εικόνας 1* προκύπτουν οι παρακάτω γραφικές παραστάσεις "congestion window / time":



Εικόνα 2.a: Γραφική παράσταση cwnd / time του Sender1 για το ερώτημα 1.2.a



Εικόνα 2.β: Γραφική παράσταση cwnd / time του Sender2 για το ερώτημα 1.2.a



Εικόνα 2.γ: Γραφική παράσταση cwnd / time του Sender3 για το ερώτημα 1.2.a

Όπως φαίνεται από τον *πίνακα 1* και τις *εικόνες 2*, οι Senders ξεκινούν να στέλνουν σε διαφορετικές χρονικές στιγμές, τόσο μεταξύ τους, όσο και από την έναρξη της προσομοίωσης. Όταν έρθουν οι χρονικές στιγμές του *πίνακα 1*, οι αντίστοιχοι αποστολείς δεν ξεκινούν απευθείας να στέλνουν τα δεδομένα προς αποστολή, αλλά πρώτα επιβεβαιώνουν την σύνδεση με τους αντίστοιχους παραλήπτες μέσω three way handshake. Αυτό ερμηνεύει και την μικρή αρχική καθυστέρηση πριν την έναρξη του πρώτου Slow Start μηχανισμού και στα τρία παραπάνω διαγράμματα.

Στις γραφικές παραστάσεις των *εικόνων 2* έχουν επισημανθεί οι περιοχές **Slow Start**, **Additive Increase** και **Timeout** αλλά και τα σημεία στα οποία υπάρχει απώλεια πακέτων (σημεία **Loss**). Παρατηρούμε λοιπόν ότι στην επικοινωνία του Sender3 με τον παραλήπτη του, δεν υπάρχει καμία απώλεια (*εικόνα 2.γ*), κάτι το οποίο δεν ισχύει στην επικοινωνία των Sender1 (*εικόνα 2.α)* και Sender2 (*εικόνα 2.β*) με τους αντίστοιχους Receivers. Αυτό οφείλεται στο ότι τα πακέτα των δευτέρων διαδίδονται μέσα από το κανάλι *Router1* και *Router2* στο οποίο έχει εισαχθεί packet error rate, ενώ στην διαδρομή των πακέτων του Sender3 δεν έχει ορισθεί τίποτα αντίστοιχο. Επιπλέον, επειδή δεν υπάρχει κανένας περιορισμός ως προς την χωρητικότητα των routers, δεν

υπάρχει σε καμία περίπτωση καταστροφική συμφόρηση, για αυτό και υπάρχει στο plot cwnd / time του Sender3 (*εικόνα 2.γ)* ένα "ατελείωτο" **Additive Increase**.

Οι τρείς παραπάνω γραφικές παραστάσεις επαληθεύουν τους μηχανισμούς του TCP Tahoe· **Slow Start**, **AI**MD congestion avoidance και Fast Retransmit. Το **Slow Start** είναι η εκθετική αύξηση του congestion window μέχρι να ξεπεράσει το ssthresh. Κάθε φορά που υπάρχει απώλεια πακέτου, το ssthresh μειώνεται στο μισό του congestion window (Multiplicative Decrease) με ελάχιστο το 2, για αυτό και στην *εικόνα 2.β* μεταξύ των 2 πρώτων διαδοχικών **Slow Start**, το δεύτερο είναι πιο σύντομο. Ο μηχανισμός **Additive Increase** ενεργοποιείται μόλις ξεπεραστεί το ssthress. Κατά το **Additive Increase** αυξάνεται αθροιστικά το cwnd και κατά συνέπεια το ssthress του επόμενου κύκλου, για αυτό και στον επόμενο κύκλο μετά από κάποια σημεία **Loss**, κατά το **Additive Increase**, βλέπουμε μεγαλύτερη **Slow Start** περιοχή από εκείνη του προηγούμενου κύκλου (*εικόνες 2.α* και *2.β*). Ο μηχανισμός Fast Retransmit, δηλαδή η επαναποστολή ενός χαμένου πακέτου μετά από την παραλαβή από τον αποστολέα τριών duplicate ACKs, φαίνεται από την άμεση μείωση στο 1 του cwnd και την συνέχεια αποστολής πακέτων χωρίς να περάσει διάστημα **Timeout**, μετά από κάποια σημεία **Loss** (*εικόνες 2.α* και *2.β*). Τα σημεία **Loss** στα οποία ενεργοποιείται αυτός ο μηχανισμός είναι αυτά στα οποία υπάρχει μόνο μία απώλεια πακέτου και τα υπόλοιπα πακέτα που δεν χάνονται είναι πάνω από 3, επειδή χρειάζονται 3 duplicate ACKs. Εάν δεν ισχύουν αυτές οι συνθήκες, τότε περνάει ένας χρόνος **Timeout** μέχρι την επανέναρξη αποστολής πακέτων (*εικόνες 2.α* και *2.β*), έτσι ώστε να ξεπεραστεί η "καταστροφική συμφόρηση", η οποία στην πραγματικότητα είναι παροδική, αφού δεν υπάρχει το πρόβλημα της χωρητικότητας των buffer των δρομολογητών.

2.b) Το Throughput προκύπτει από το πηλίκο της ποσότητας των απεσταλμένων δεδομένων προς τον χρόνο. Για τον υπολογισμό του μεγέθους αυτού χρησιμοποιείται το διάνυσμα packetReceived:vector(packetBytes) το οποίο δείχνει πόσα πακέτα έλαβε ο κάθε παραλήπτης από τον κάθε αποστολέα καθόλη την διάρκεια της προσομοίωσης. Οπότε με το γινόμενο του μέτρου αυτού του διανύσματος, αναφερόμενο σε καθένα αποστολέα, επί το μέγεθος σε Bytes του κάθε πακέτου προς τον συνολικό χρόνο αποστολής του αντίστοιχου αποστολέα υπολογίζεται το throughput ανά αποστολέα, ενώ με το γινόμενο του μέτρου του διανύσματος αναφερόμενο σε όλους τους αποστολής επί το μέγεθος σε Bytes του κάθε πακέτου προς τον συνολικό χρόνο της προσομοίωσης υπολογίζεται το throughput της προσομοίωσης.

Σχόλιο: Κανονικά οι υπολογισμοί θα έπρεπε να γίνονται με βάση το packetSent:vector(packetBytes). Ωστόσο, κατά την προβολή του, αυτό το διάνυσμα εμφανίζει μη λογικά αποτελέσματα. Για αυτό και επιλέγεται στην θέση του το packetReceived:vector(packetBytes) με το οποίο στην ουσία υπολογίζεται το application throughput.

Πίνακας μέτρων packetReceived:vector(packetBytes)	
Receiver2 από Sender1	210 πακέτα
Receiver2 από Sender2	202 πακέτα
Receiver3 από Sender3	7222 πακέτα
Σύνολο	**7634 πακέτα**

Πίνακας 2: Αναφερόμενος στο ερώτημα 1.2.b

Λαμβάνοντας υπόψιν τους χρόνους και το maximum segment size του *πίνακα 1* και τις τιμές του *πίνακα 2*, υπολογίζονται τα παρακάτω throughput.

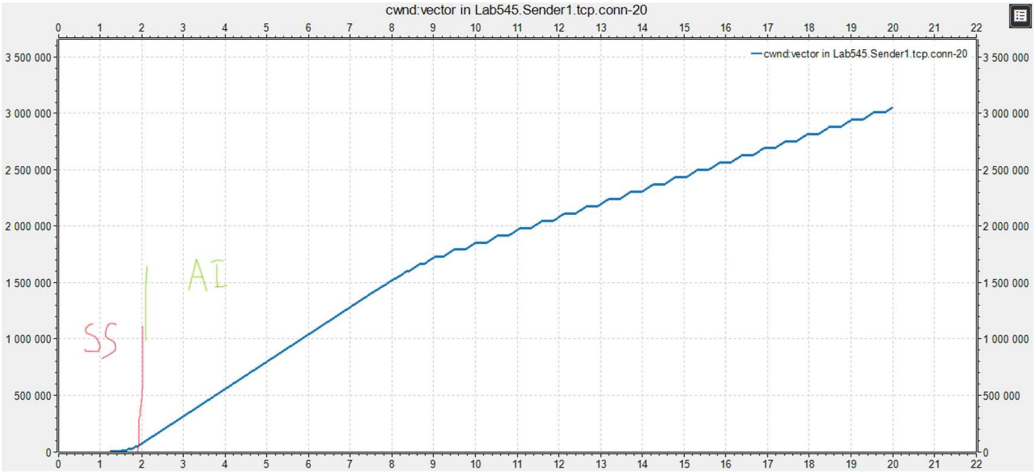
$$Throughput_{Sende} = \frac{210\ packets\ \cdot 1460\ \frac{Bytes}{packet}}{20\,\text{sec} - 1\ sec} = 16.137\ KBps$$

$$Throughput_{Sender2} = \frac{202\ packets\ \cdot 1460\ \frac{Bytes}{packet}}{20\,\text{sec} - 7\ sec} = 22.686\ KBps$$

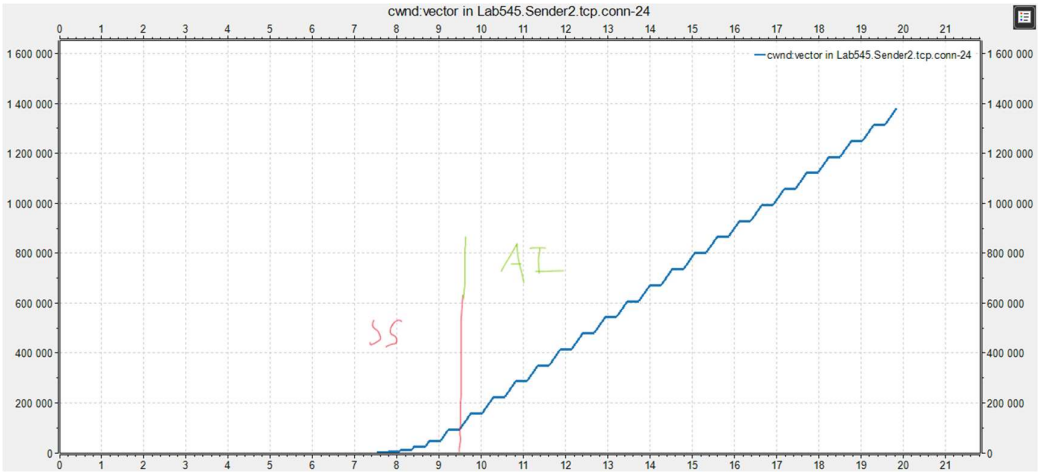
$$Throughput_{Sende} = \frac{7222\ packets\ \cdot 1460\ \frac{Bytes}{packet}}{20\,\text{sec} - 5\ sec} = 702.941\ KBps$$

$$Throughput_{Simulation} = \frac{7634\ packets\ \cdot 1460\ \frac{Bytes}{packet}}{20\ sec} = 557.282\ KBps$$

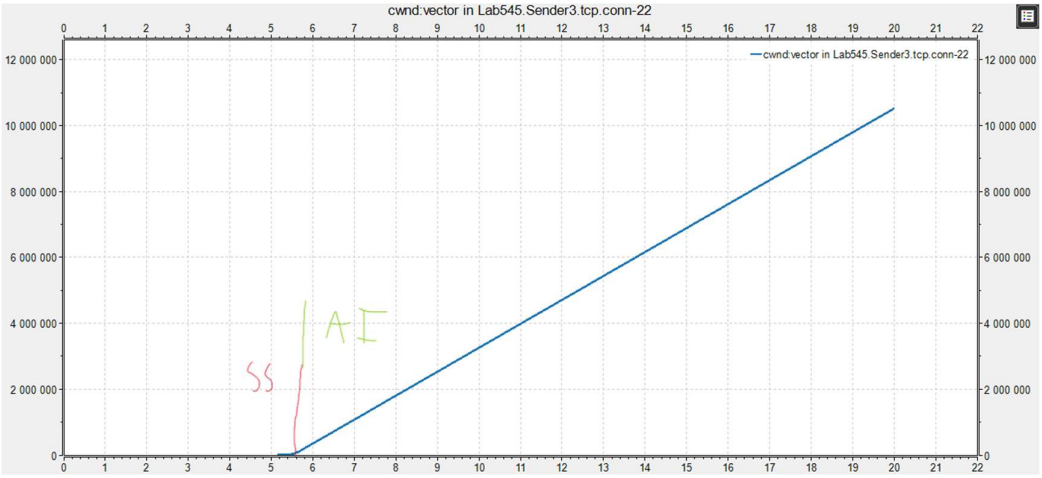
3) Αφαιρώντας το packet error rate, δηλαδή τρέχοντας τον κώδικα *.ned* χωρίς την αλλαγή στην αρχή της *άσκησης 1*, προκύπτουν τα παρακάτω plots cwnd / time:



Εικόνα 3.a: Γραφική παράσταση cwnd / time του Sender1 για το ερώτημα 1.3



Εικόνα 3.β: Γραφική παράσταση cwnd / time του Sender2 για το ερώτημα 1.3



Εικόνα 3.γ: Γραφική παράσταση cwnd / time του Sender3 για το ερώτημα 1.3

Η γραφική παράσταση της *εικόνας 3.γ* είναι η ίδια με εκείνη της εικόνας *2.γ*, δηλαδή δεν υπήρξε καμία αλλαγή στην αποστολή του Sender3, το οποίο είναι λογικό. καθώς η αφαίρεση του packet error rate δεν επηρεάζει με κανέναν τρόπο την δική του λειτουργία. Από την άλλη, με την αφαίρεση του per, δεν υπάρχουν πλέον απώλειες πακέτων στην επικοινωνία των Sender1 και Sender2 με τους αντίστοιχους παραλήπτες, με αποτέλεσμα τα αντίστοιχα μέτρα του packetReceived:vector(packetBytes) (και κατά συνέπεια των αντίστοιχων throughput) να είναι κατά πολύ αυξημένα (*πίνακας 3)*, όπως και οι γραφικές παραστάσεις των *εικόνων 3.α* και *3.β* να είναι πλέον ποιοτικά ίδιες με εκείνη του Sender3 *(εικόνα 3.γ*). Έχουν μια μόνο μια εμφανή διαφορά η οποία παρατηρείται λίγο πιο μετά από το σημείο που ξεκινάει να στέλνει ο Sender2 (περίπου στα 8.5sec). Από εκεί και μετά, μια μικρή διακοπή αποστολής ακολουθεί την κάθε προσθετική αύξηση. Παρακολουθώντας την προσομοίωση σε εκείνα τα διαστήματα γίνεται φανερό ότι οι διακοπές αυτές είναι στην ουσία εναλλαγές μεταξύ αποστολών Sender1 και Sender2 στον ίδιο παραλήπτη Receiver2. Δηλαδή, οι δυο αποστολείς δεν στέλνουν ταυτόχρονα τα δεδομένα τους στον κοινό παραλήπτη, αλλά στέλνουν ο καθένας εναλλάξ, χωρίς να μπερδεύονται τα πακέτα τους. Όμως, αν ξανατρέξουμε την

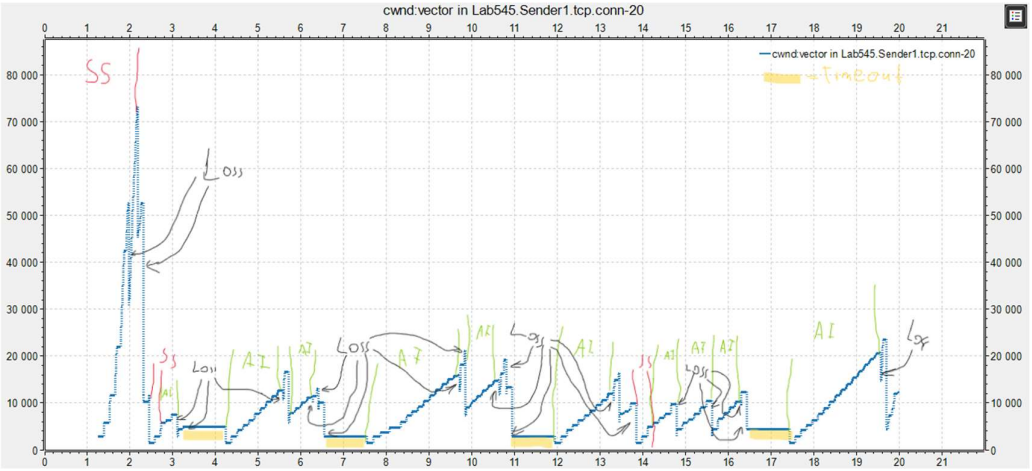
προσομοίωση με το packet error rate, θα παρατηρήσουμε ότι αυτή εναλλαγή μεταξύ των Sender1 και Sender2 δεν υπάρχει και τα πακέτα τους φτάνουν στον Receiver2 ανακατεμένα μεταξύ τους. Αυτή η συμπεριφορά είναι μη αναμενόμενη αλλά συγκρίνοντας τις δύο περιπτώσεις γίνεται η υπόθεση ότι το τεράστιο πλήθος των πακέτων που αποστέλλονται σε κάθε κύκλο της *άσκησης 1.3* καθιστά αναγκαίο το διαχωρισμό τον δύο αποστολών έτσι ώστε να μην υπάρχει αποστολή ACK σε λάθος Sender.

Πίνακας μέτρων packetReceived:vector(packetBytes)	
Receiver2 από Sender1	2099 πακέτα
Receiver2 από Sender2	943 πακέτα
Receiver3 από Sender3	7222 πακέτα
Σύνολο	**10264 πακέτα**

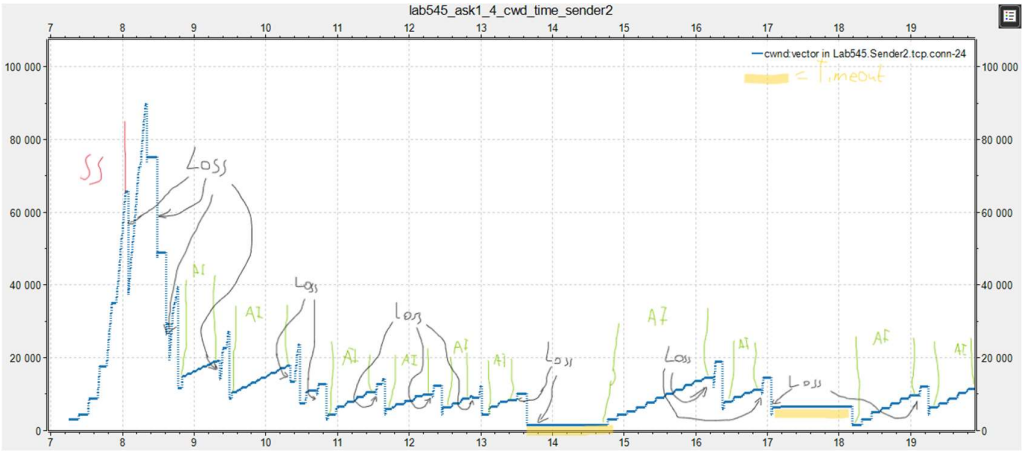
Πίνακας 3: Αναφερόμενος στο ερώτημα 1.3

4.b) Με την αλλαγή έκδοσης TCP από Tahoe σε NewReno, διατηρούνται οι μηχανισμοί που αναφέρονται στο *υποερώτημα 2.α* αλλά προστίθεται και εκείνος του "τροποποιημένου" Fast Recovery. Στην ουσία με την απώλεια ενός πακέτου, αφού γίνει ο μηχανισμός του Fast Retransmit, το congestion window μειώνεται στο μισό του flight size, αντί να επιστρέφει στο 1, όπως γίνεται στο Tahoe. Ο χαρακτηρισμός "τροποποιήμενο" του μηχανισμού Fast Recovery αναφέρεται στην βελτιωμένη έκδοση του η οποία είναι ικανή να ανιχνεύει πολλαπλές απώλειες πακέτων αξιοποιώντας τα μερικά ACKs.

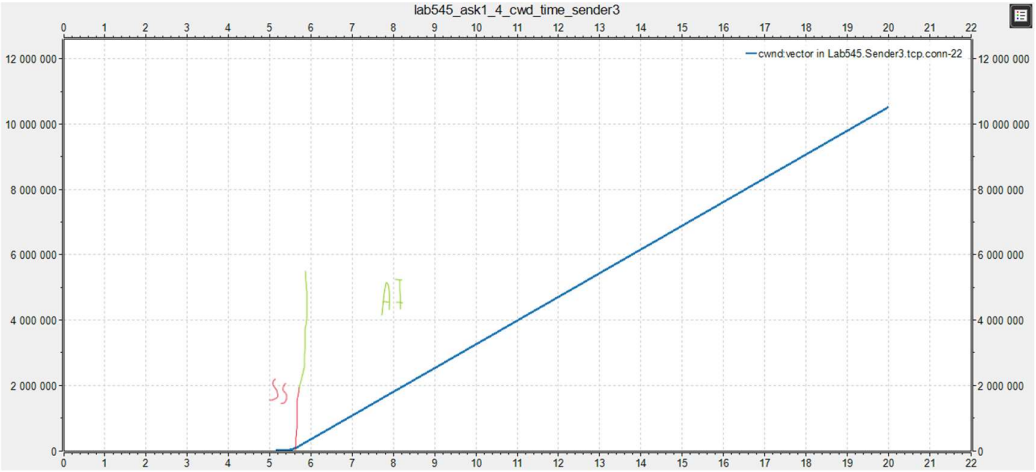
Αλλάζοντας λοιπόν το TCP σε αυτήν την έκδοση (και επαναφέροντας το packet error rate) προκύπτουν οι παρακάτω γραφικές παραστάσεις cwnd / time:



Εικόνα 4.a: Γραφική παράσταση cwnd / time του Sender1 για το ερώτημα 1.3.b



Εικόνα 4.β: Γραφική παράσταση cwnd / time του Sender2 για το ερώτημα 1.3.b



Εικόνα 4.γ: Γραφική παράσταση cwnd / time του Sender3 για το ερώτημα 1.4.b

Η γραφική παράσταση του Sender3 (*εικόνα 4.γ*) δεν υπέστη καμία αλλαγή σε σχέση με εκείνη της προηγούμενης έκδοσης (*εικόνα 2.γ*), καθώς δεν υπάρχουν απώλειες πακέτων για να εφαρμοστεί ο καινούργιος μηχανισμός. Επομένως η αλλαγή δεν επηρεάζει ούτε το cwnd, ούτε την απόδοση. Από την άλλη στις γραφικές παραστάσεις των Sender1 και Sender2 (*εικόνες 4.α* και *4.β*) βλέπουμε λιγότερα **Timeout** σε σχέση με εκείνες των *εικόνων 2.α* και *2.β*, αφού μετά από τα περισσότερα σημεία **Loss** εφαρμόζεται ο μηχανισμός του "τροποποιημένου" Fast Recovery (είναι τα spikes που ακολουθούν τις πτώσεις του cwnd στις δύο γραφικές παραστάσεις), οπότε επανέρχεται γρήγορα η αποστολή δεδομένων χωρίς μεγάλες παύσεις. **Timeout** περιοχές συνεχίζουν να υπάρχουν επειδή σε αυτές τις περιπτώσεις δεν υπάρχουν αρκετά πακέτα που στέλνονται επιτυχημένα ώστε να υπάρχουν τρία duplicate ACKs για την ενεργοποίηση των αντίστοιχων ρουτίνων, αλλά προφανώς είναι λιγότερες και πιο σύντομες. Λιγότερα και πιο σύντομα **Timeout** σε συνδυασμό με μεγαλύτερο μέσο μέγεθος του congestion window (στις *εικόνες 4.α* και *4.β* φαίνεται ξεκάθαρα το cwnd να παίρνει μεγαλύτερες τιμές) σημαίνει μεγαλύτερη απόδοση, δηλαδή αποστολή περισσότερων δεδομένων στο ίδιο χρονικό διάστημα, το οποίο επαληθεύεται και από τα μέτρα του packetReceived:vector(packetBytes) στον *πίνακα 4*.

Πίνακας μέτρων packetReceived:vector(packetBytes)	
Receiver2 από Sender1	425 πακέτα
Receiver2 από Sender2	344 πακέτα
Receiver3 από Sender3	7222 πακέτα
Σύνολο	**7991 πακέτα**

Πίνακας 4: Αναφερόμενος στο ερώτημα 1.4.b

5) Το delay εκφράζει το propagation delay της κάθε σύνδεσης και επομένως η αλλαγή του μεταβάλλει ανάλογα και τον χρόνο που κάνει ένα πακέτο να φτάσει από την αρχή εώς το τέλος του αντίστοιχου καναλιού.

Το datarate (ή αλλιώς bandwidth) εκφράζει την χωρητικότητας μιας τομής του καναλιού σε bits. Η μεταβολή αυτού του μεγέθους επηρεάζει αντιστρόφως ανάλογα το transmission delay.

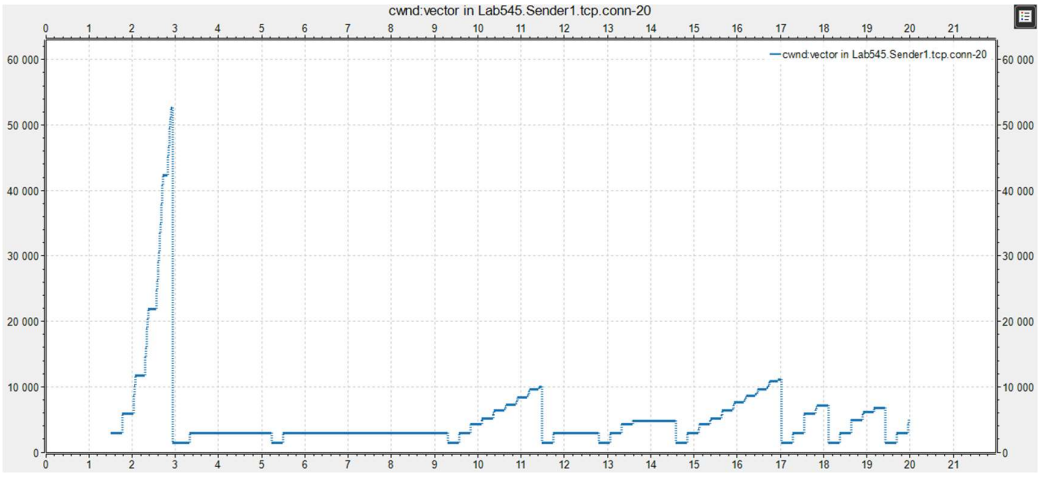
Επομένως τα δύο μεγέθη επηρεάζουν με αντίστροφο τρόπο την συνολική καθυστέρηση κατά την μεταφορά των δεδομένων. Π.χ. η αύξηση του delay και η μείωση του datarate θα αυξήσει την συνολική καθυστέρηση σε όλες τις επικοινωνίες, στον ίδιο χρόνο θα μεταδίδονται λιγότερα πακέτα και άρα θα πέσει η συνολική απόδοση, όπως και θα υπάρχει μια “επιμήκυνση” στον άξονα του χρόνου και μια μετατόπιση προς τα δεξία στα διαγράμματα cwnd / time.

Πράγματι, αν εφαρμόσουμε στον κώδικα του *.ned* αρχείου την αλλαγή της *εικόνας 5* (αύξηση των delay κατά 50% και μείωση των datarate κατά 50%) προκύπτουν οι γραφικές παραστάσεις των *εικόνων 6.α, 6.β* και *6.γ* και ο *πίνακας 5* τα οποία επιβεβαιώνουν τα προαναφερθέντα συμπεράσματα.

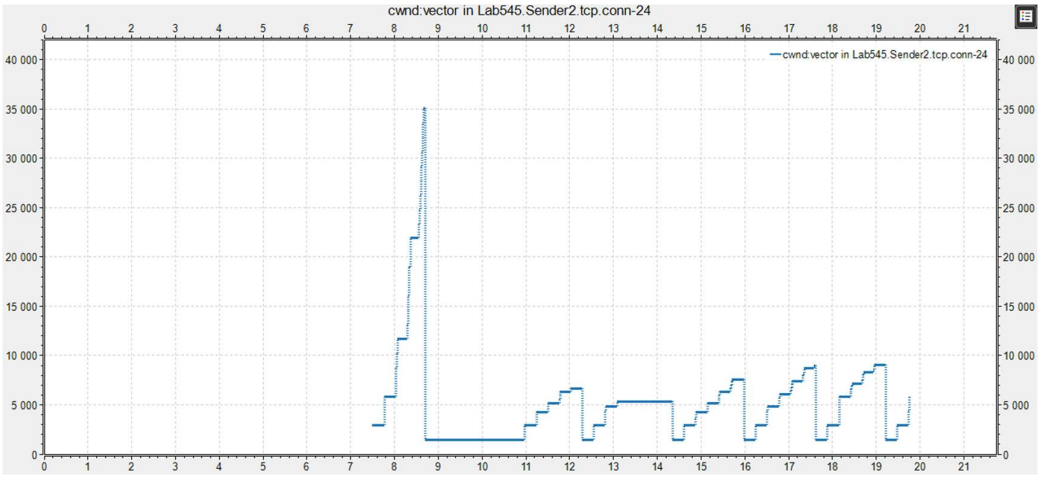
Σχόλιο: Συγκρίνοντας τις *εικόνες 2* και *6* βλέπουμε ότι οι γραφικές παραστάσεις με τις αυξημένες καθυστερήσεις διαφέρουν, όπως έχει οριστεί παραπάνω, από τις αρχικές γραφικές παραστάσεις, αλλά ταυτόχρονα βλέπουμε να έχουν προστεθεί/αφαιρεθεί κάποιες αποστολές δεδομένων. Αυτό οφείλεται στο ότι το per είναι μία πιθανότητα να χαθούν τα πακέτα στο αντίστοιχο κανάλι, οπότε μεταβάλλοντας τα μεγέθη delay και datarate, αλλάζουν και οι χρονικές στιγμές στις οποίες είναι προκαθορισμένο να χαθούν τα πακέτα. Επομένως, οι συγκεκριμένες διαφορές δεν σχετίζονται ποιοτικά με το delay και datarate οπότε αγνοούνται.

```
57●     connections:
58          // X=5 Y=4 Z=5
59          Sender1.pppg++ <--> {  delay = 40ms; datarate = 2.5Mbps; } <--> Router1.pppg++;
60          Sender2.pppg++ <--> {  delay = 40ms; datarate = 3Mbps; } <--> Router1.pppg++;
61          Router1.pppg++ <--> {  delay = 40ms; datarate = 1Mbps; per = 4e-2; } <--> Router2.pppg++;
62          Router2.pppg++ <--> {  delay = 40ms; datarate = 30Mbps; } <--> Sender3.pppg++;
63          Router2.pppg++ <--> {  delay = 40ms; datarate = 2Mbps; } <--> Receiver2.pppg++;
64          Router2.pppg++ <--> {  delay = 40ms; datarate = 1Mbps; } <--> Receiver1.pppg++;
65          Router2.pppg++ <--> {  delay = 40ms; datarate = 3Mbps; } <--> Receiver3.pppg++;
66  }
```

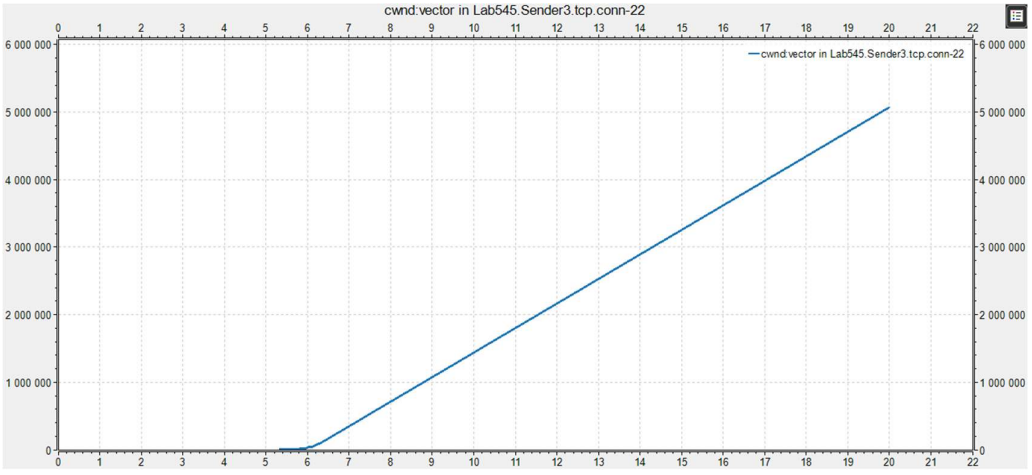
Εικόνα 5: Αλλαγή datarate και delay του κώδικα “lab545.ned”



Εικόνα 6.α: Γραφική παράσταση cwnd / time του Sender1 για το ερώτημα 1.5



Εικόνα 6.β: Γραφική παράσταση cwnd / time του Sender2 για το ερώτημα 1.5



Εικόνα 6.γ: Γραφική παράσταση cwnd / time του Sender3 για το ερώτημα 1.5

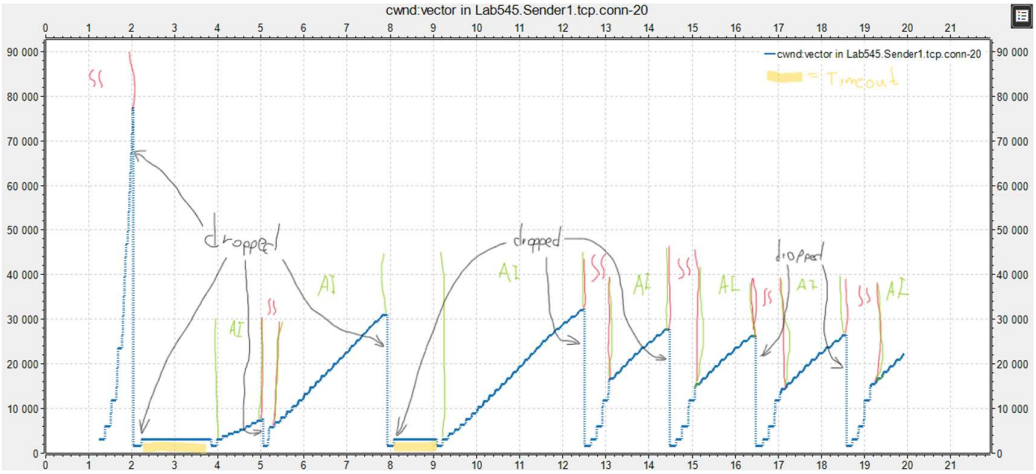
Πίνακας μέτρων packetReceived:vector(packetBytes)	
Receiver2 από Sender1	113 πακέτα
Receiver2 από Sender2	87 πακέτα
Receiver3 από Sender3	3489 πακέτα
Σύνολο	**3689 πακέτα**

Πίνακας 5: Αναφερόμενος στο ερώτημα 1.5

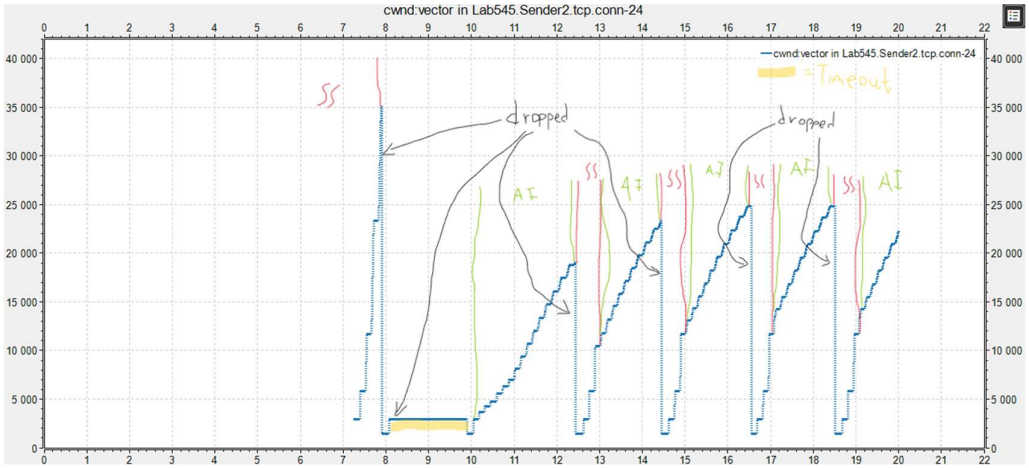
$$Throughput_{Simulatio} = \frac{3689\ packets\ \cdot 1460\ \frac{Bytes}{packet}}{20\ sec} = 269.297\ KBps$$

Άσκηση 2

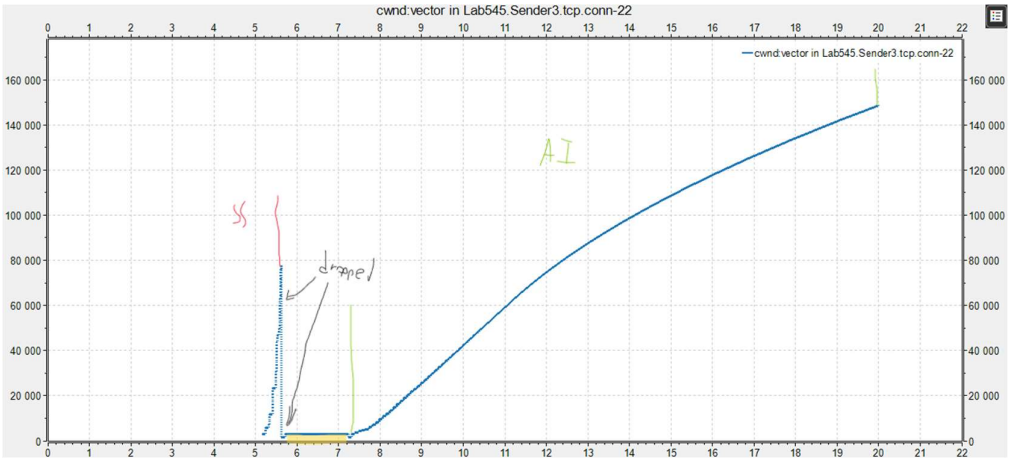
1.a) Τρέχοντας την προσομοίωση **Lab_erg_ask2_dropTail** με τις παραμέτρους του *πίνακα 1* προκύπτουν οι παρακάτω γραφικές παραστάσεις "congestion window / time" και πίνακας μέτρων packetReceived:vector(packetBytes):



Εικόνα 7.α: Γραφική παράσταση cwnd / time του Sender1 για το ερώτημα 2.1.a



Εικόνα 7.β: Γραφική παράσταση cwnd / time του Sender2 για το ερώτημα 2.1.a



Εικόνα 7.γ: Γραφική παράσταση cwnd / time του Sender3 για το ερώτημα 2.1.a

Πίνακας μέτρων packetReceived:vector(packetBytes)	
Receiver2 από Sender1	995 πακέτα
Receiver2 από Sender2	532 πακέτα
Receiver3 από Sender3	5369 πακέτα
Σύνολο	**6896 πακέτα**

Πίνακας 6: Αναφερόμενος στο ερώτημα 2.1.a

$$Throughput_{Simulatio} = \frac{6896\ packets \cdot 1460\ \frac{Bytes}{packet}}{20\ sec} = 503.408\ KBps$$

Σε σχέση με το ερώτημα 1.2 παρατηρούμε ότι στην περίπτωση των Sender1 και Sender2 υπάρχουν λιγότερα **Timeout** (*εικόνες 7.α* και *7.β*) και επομένως επιτυγχάνεται μεγαλύτερη απόδοση στις συγκεκριμένες επικοινωνίες (όπως φαίνεται και από τον *πίνακα 6* σε σύγκριση με τον *πίνακα 2*). Από την άλλη στην περίπτωση του Sender3 η απόδοση πέφτει, καθώς από κανένα σημείο **Loss**, τώρα υπάρχουν δύο (*εικόνες 7.γ*), όπως εμφανίζεται και **Timeout** περιοχή. Αυτό έχει ως αποτέλεσμα όχι μόνο να είναι μειωμένο το throughput του Sender3 σε σχέση με το ερώτημα 1.2, αλλά να πέφτει και η συνολική απόδοση τους συστήματος αφού το throughput της προσομοίωσης είναι μικρότερο.

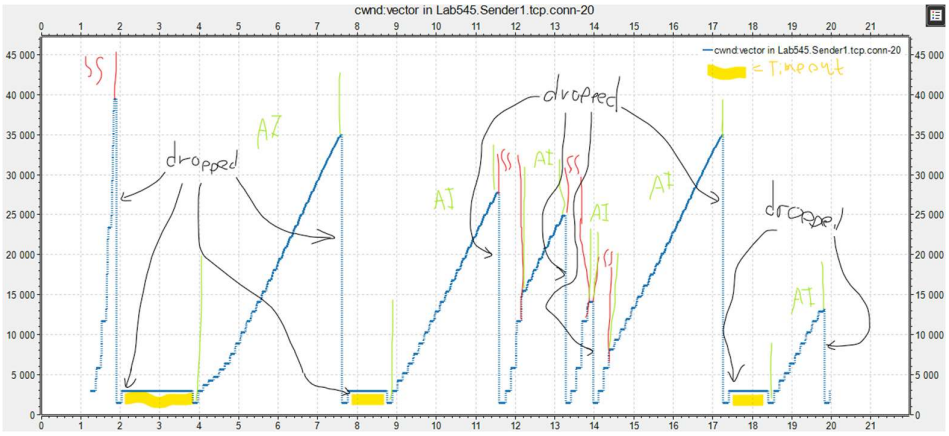
1.b) <u>Σχόλιο</u>: Λόγω το ΑΕΜ, τα X,Y και Z είναι τέτοια ώστε η DropTailQueue να έχει μέγεθος 10, το οποίο δεν υπόκειται σε καμία από τις δύο περιπτώσεις του ερωτήματος 2.1.b). Για αυτόν τον λόγο, σε συνδυασμό με το προσωπικό ενδιαφέρον, επιλέγεται να υποδιπλασιαστεί το μέγεθος της ουράς. Η ίδια επιλογή θα γίνει και στο ερώτημα 2.2.c.

Η αλλαγή του μεγέθους του DropTailQueue από 10 σε 5 (*εικόνα 8*) οδηγεί στα παρακάτω αποτελέσματα της προσομοίωσης.

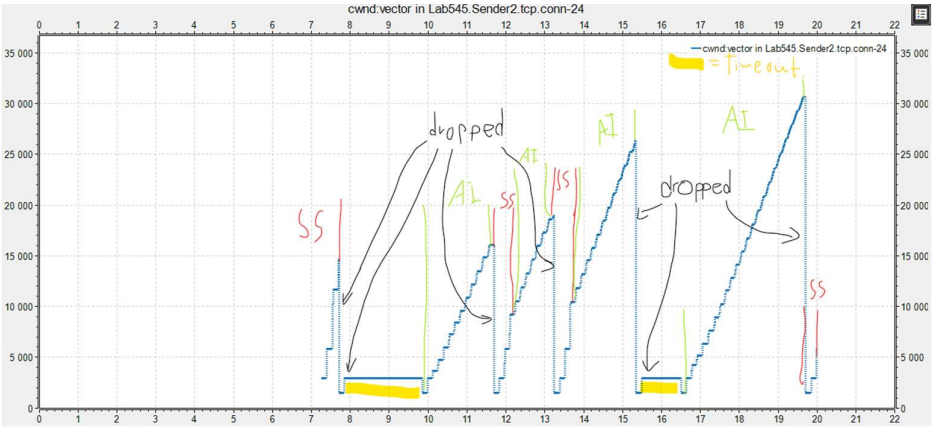
```
[Config Lab_erg_ask2_dropTail]
network = Lab545

## X=5 Y=4 Z=5
**.Router*.ppp[*].queue.typename = "DropTailQueue" # in routers
**.Router*.ppp[*].queue.packetCapacity = 5
```

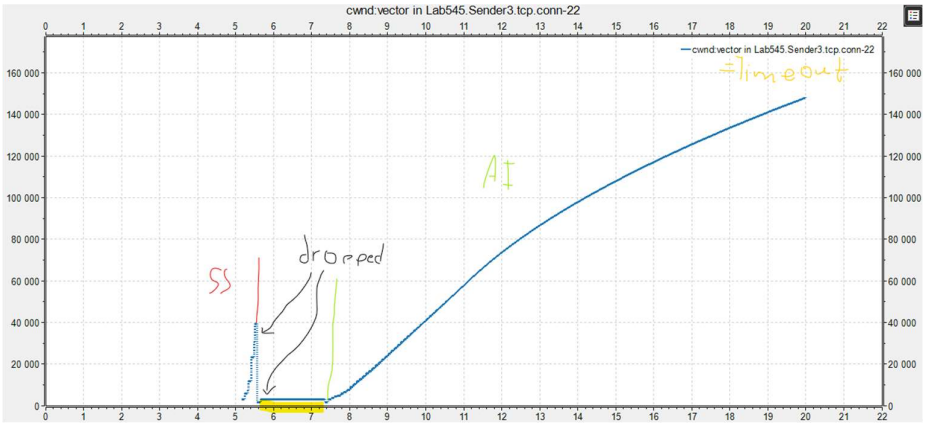
Εικόνα 8: Αλλαγή μεγέθους του DropTailQueue στο "omnetpp.ini"



Εικόνα 9.α: Γραφική παράσταση cwnd / time του Sender1 για το ερώτημα 2.1.b



Εικόνα 9.β: Γραφική παράσταση cwnd / time του Sender2 για το ερώτημα 2.1.b



Εικόνα 9.γ: Γραφική παράσταση cwnd / time του Sender3 για το ερώτημα 2.1.b

Πίνακας μέτρων packetReceived:vector(packetBytes)	
Receiver2 από Sender1	933 πακέτα
Receiver2 από Sender2	507 πακέτα
Receiver3 από Sender3	5305 πακέτα
Σύνολο	**6745 πακέτα**

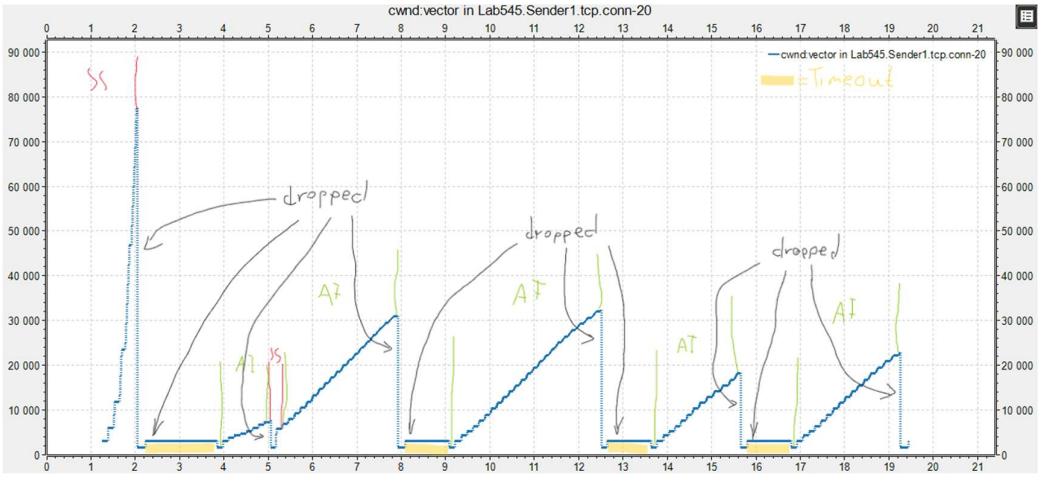
Πίνακας 7: Αναφερόμενος στο ερώτημα 2.1.b

$$Throughput_{Simulation} = \frac{6745\ packets\ \cdot 1460\ \frac{Bytes}{packet}}{20\ sec} = 492.385\ KBps$$

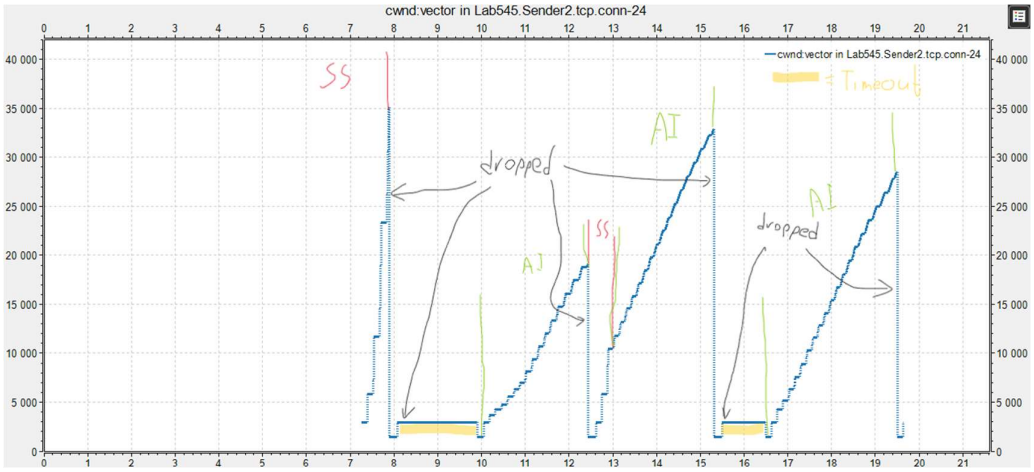
Σε σχέση με το ερώτημα 1.2 παρατηρούμε ακριβώς τα ίδια που παρατηρήσαμε και στο ερώτημα 2.1.a. Εξάγονται όμοια συμπεράσματα μόνο που σε αυτή την περίπτωση όλα τα throughput (αποστολέων και προσομοίωσης) είναι μικρότερα (*πίνακας 7*) σε σύγκριση με το ερώτημα 2.1.a λόγω του μικρότερου μεγέθους της ουράς.

1.c) Και τα δύο προηγούμενα ερωτήματα (2.1.a και 2.1.b) αποτελούν παραδείγματα της αδικίας της ουράς DropTail στον διαμοιρασμό των πόρων μεταξύ πολλαπλών χρηστών. Συγκεκριμένα, η DropTail οδηγεί σε πολλές περιπτώσεις στην μονοπωλιακή απασχόληση της ουράς μόνο από κάποιες ροές, το οποίο φαίνεται ξεκάθαρα από τις γραφικές παραστάσεις των *εικόνων 7* και *9* στις οποίες ο Sender3 διεκδικεί πλήρη χρήση της ουράς εις βάρος των Sender1 και Sender3

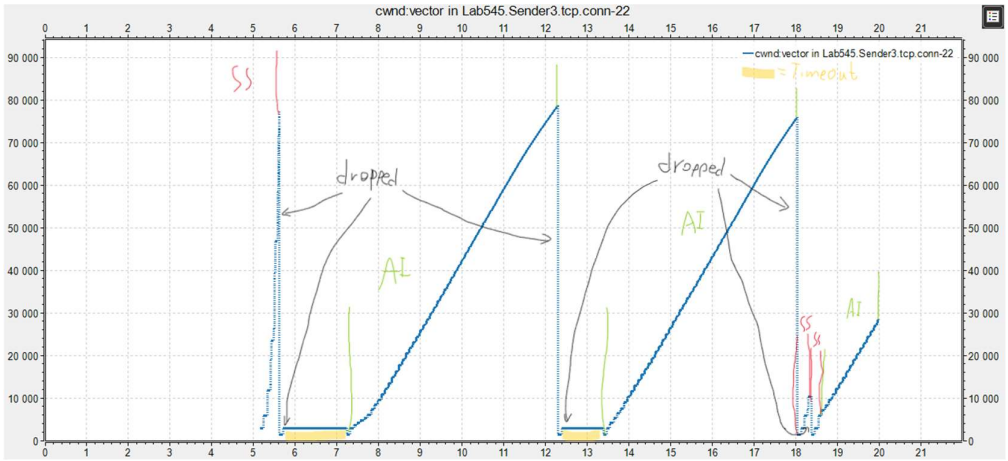
2.a) Τρέχοντας την προσομοίωση **Lab_erg_ask2_red** με τις παραμέτρους του *πίνακα 1* προκύπτουν οι παρακάτω γραφικές παραστάσεις “congestion window / time”:



Εικόνα 10.a: Γραφική παράσταση cwnd / time του Sender1 για το ερώτημα 2.2.a



Εικόνα 10.β: Γραφική παράσταση cwnd / time του Sender2 για το ερώτημα 2.2.a



Εικόνα 10.γ: Γραφική παράσταση cwnd / time του Sender3 για το ερώτημα 2.2.a

Πίνακας τιμών packetReceived:vector(packetBytes)	
Receiver2 από Sender1	726 πακέτα
Receiver2 από Sender2	533 πακέτα
Receiver3 από Sender3	3080 πακέτα
Σύνολο	**4339 πακέτα**

Πίνακας 8: Αναφερόμενος στο ερώτημα 2.2.a

$$Throughput_{Simulation} = \frac{4339\ packets\ \cdot 1460\ \frac{Bytes}{packet}}{20\ sec} = 316.747\ KBps$$

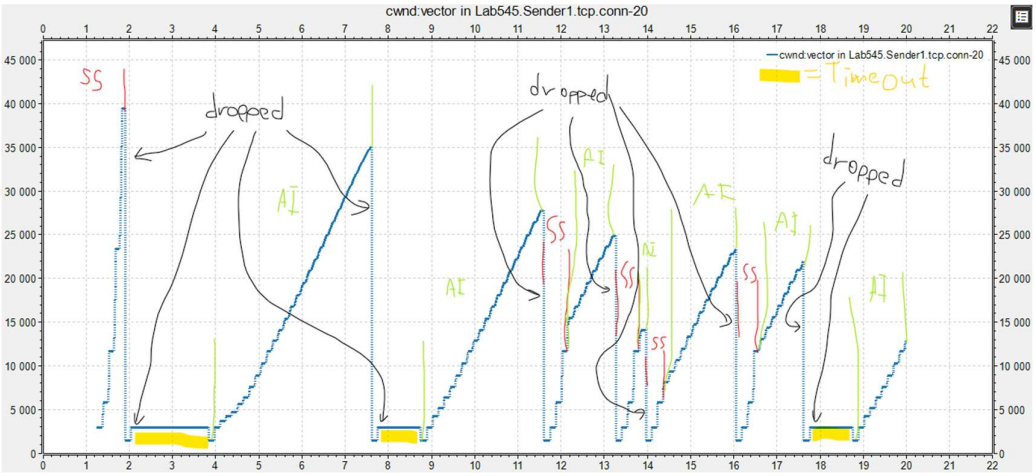
2.b) Σε σύγκριση με την DropTail, η RED είναι λιγότερο αποδοτική, όπως φαίνεται από του *Πίνακες 6* και *8* και τα μικρότερο throughput προσομοίωσης της RED. Αυτό το αποτέλεσμα είναι αναμενόμενο, διότι η RED ρίχνει από μόνη της τυχαία πακέτα και περισσότερα **Dropped** πακέτα σημαίνει χαμηλότερη απόδοση. Ωστόσο, η μείωση στην απόδοση που προκαλεί το drop τυχαίων πακέτων αντισταθμίζεται από την υψηλότερη δικαιοσύνη στην οποία οδηγεί η τυχαιότητα. Επιβεβαίωση αυτής αποτελεί η απουσία της μονοπωλιακής χρήσης της ουράς του Sender3 της εφαρμογής DropTail από την συγκεκριμένη περίπτωση.

2.b) Η αλλαγή του μεγέθους του RedDropperQueue από 10 σε 5, όπως και όλων των παραμέτρων που εξαρτούνται από αυτό (*εικόνα 11*), οδηγεί στα παρακάτω αποτελέσματα της προσομοίωσης.

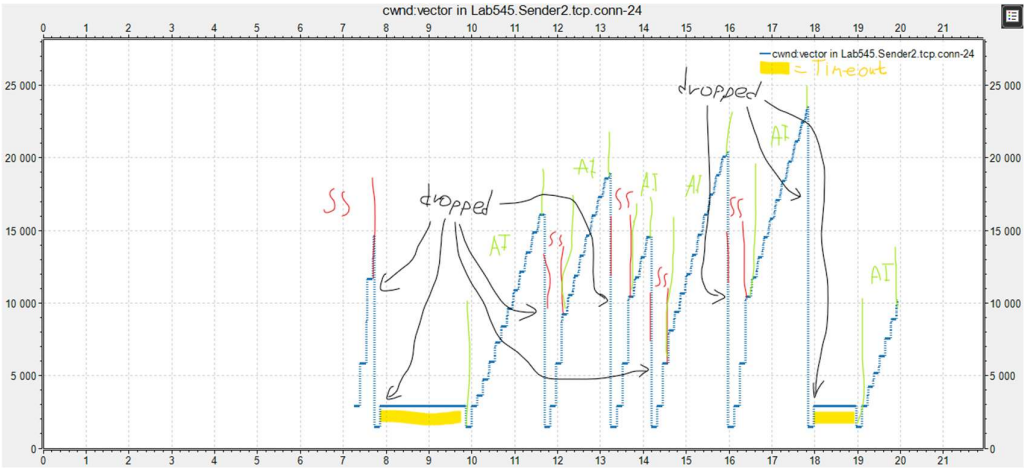
```
[Config Lab_erg_ask2_red]
network = Lab545

## X=5 Y=4 Z=5
**.Router*.ppp[*].queue.typename = "RedDropperQueue"
**.Router*.ppp[*].queue.red.packetCapacity = 5
**.Router*.ppp[*].queue.red.maxp = 0.1
**.Router*.ppp[*].queue.red.maxth = 2.5
**.Router*.ppp[*].queue.red.minth = 0.833
**.Router*.ppp[*].queue.red.wq = 0.002
```

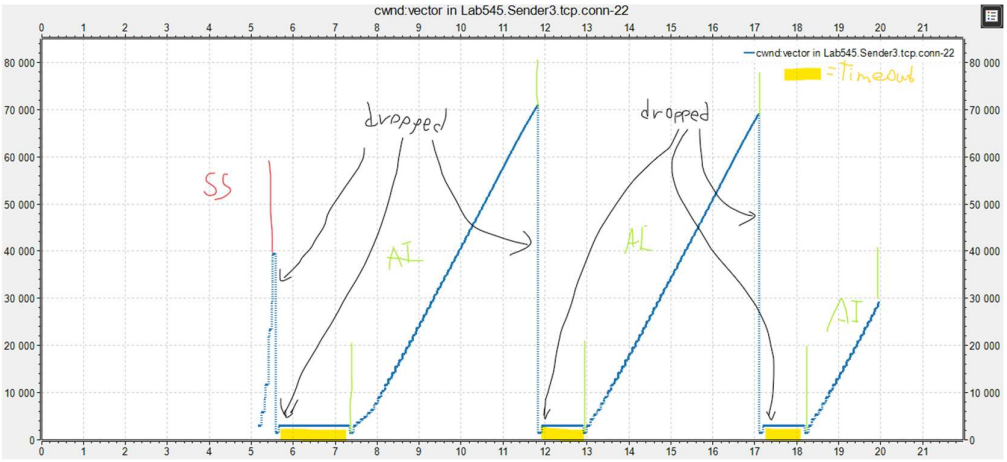
Εικόνα 11: Αλλαγή μεγέθους του RedDropperQueue στο "omnetpp.ini"



Εικόνα 12.α: Γραφική παράσταση cwnd / time του Sender1 για το ερώτημα 2.2.c



Εικόνα 12.β: Γραφική παράσταση cwnd / time του Sender2 για το ερώτημα 2.2.c



Εικόνα 12.γ: Γραφική παράσταση cwnd / time του Sender3 για το ερώτημα 2.2.c

Πίνακας μέτρων packetReceived:vector(packetBytes)	
Receiver2 από Sender1	856 πακέτα
Receiver2 από Sender2	401 πακέτα
Receiver3 από Sender3	2571 πακέτα
Σύνολο	**3828 πακέτα**

Πίνακας 9: Αναφερόμενος στο ερώτημα 2.2.c

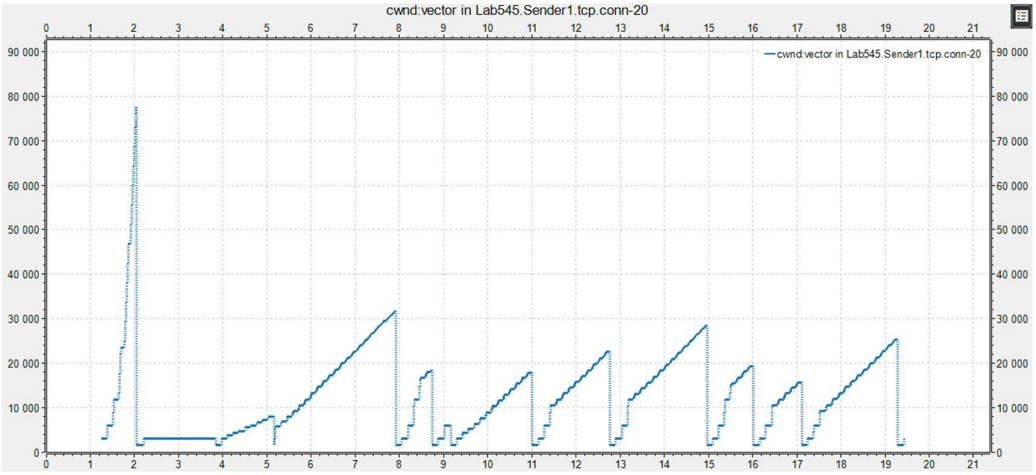
$$Throughput_{Simulation} = \frac{3828\ packets\ \cdot 1460\ \frac{Bytes}{packet}}{20\ sec} = 279.444\ KBps$$

Τα συμπεράσματα που εξάγονται για την συγκεκριμένη ουρά, συγκρίνοντας την με εκείνη του ερωτήματος 2.1.b, είναι όμοια με εκείνα που εξάχθηκαν στο 2.2.b, μόνο που επειδή τα μεγέθη των DropTailQueue και RedDropperQueue είναι μικρότερα, οι αντίστοιχες αποδόσεις τους είναι μειωμένες, με εκείνη της ουράς RED να έχει υποστεί μια σημαντική ελάττωση, η οποία όμως συνοδεύεται από μεγαλύτερη δικαιοσύνη. Αυτό επιβεβαιώνεται υπολογίζοντας τους Fairness Indices των RedDropperQueue μεγέθους 5 και 10 μέσω του τύπου $f(x_1, x_2, x_3) = \frac{(x_1+x_2+x_3)^2}{3\cdot({x_1}^2+{x_2}^2+{x_3}^2)}$, όπου x_i τα throughput των Senders. Αυτοί είναι ίσοι με $f(x_1, x_2, x_3) = 0.631$ και $f(x_1, x_2, x_3) = 0.596$ αντίστοιχα, που σημαίνει ότι μικρότερο μέγεθος ουράς, οδηγεί σε μεγαλύτερη δικαιοσύνη.

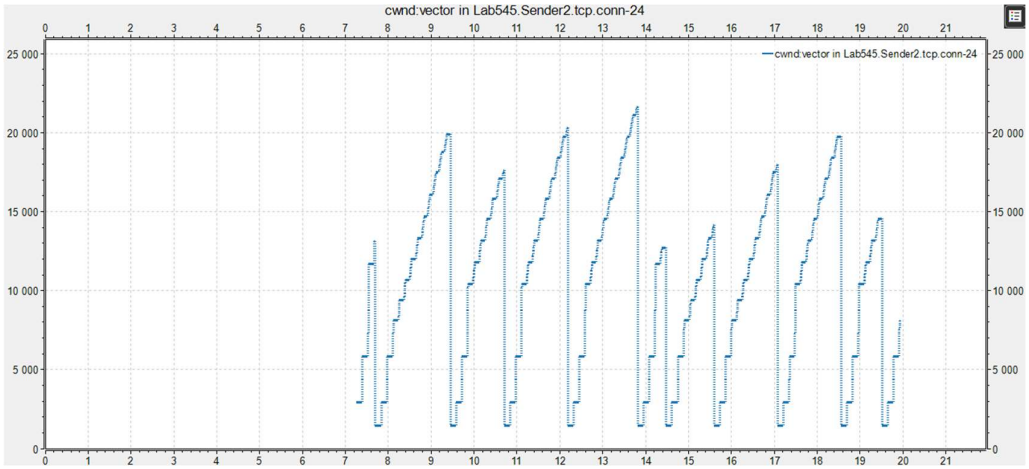
2.d) Για RedDropperQueue μεγέθους 10 οι τιμές των παραμέτρων της RED που επιτυγχάνουν μεγαλύτερη δικαιοσύνη είναι αυτές της *εικόνας 13*.

```
41 [Config Lab_erg_ask2_red]
42 network = Lab545
43
44 ## X=5 Y=4 Z=5
45 **.Router*.ppp[*].queue.typename = "RedDropperQueue"
46 **.Router*.ppp[*].queue.red.packetCapacity = 5
47 **.Router*.ppp[*].queue.red.maxp = 0.1
48 **.Router*.ppp[*].queue.red.maxth = 2.5
49 **.Router*.ppp[*].queue.red.minth = 0.833
50 **.Router*.ppp[*].queue.red.wq = 0.002
```

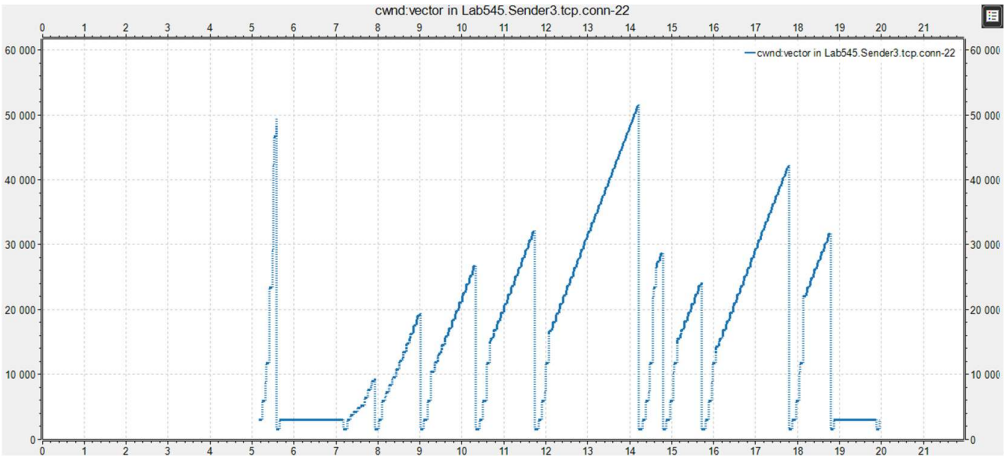
Εικόνα 13: Παράμετροι του RedDropperQueue για καλύτερη δικαιοσύνη στο "omnetpp.ini"



Εικόνα 14.α: Γραφική παράσταση cwnd / time του Sender1 για το ερώτημα 2.2.d



Εικόνα 14.β: Γραφική παράσταση cwnd / time του Sender2 για το ερώτημα 2.2.d



Εικόνα 14.γ: Γραφική παράσταση cwnd / time του Sender3 για το ερώτημα 2.2.d

Πίνακας μέτρων packetReceived:vector(packetBytes)	
Receiver2 από Sender1	888 πακέτα
Receiver2 από Sender2	547 πακέτα
Receiver3 από Sender3	1729 πακέτα
Σύνολο	**3164 πακέτα**

Πίνακας 10: Αναφερόμενος στο ερώτημα 2.2.d

Με βάση τον *πίνακα 10* και τους τύπους που αξιοποιήθηκαν για τον υπολογισμό της δικαιοσύνης στο ερώτημα 2.2.c προκύπτει ότι ο δείκτης δικαιοσύνης είναι ίσος με $f(x_1, x_2, x_3) = 0.805$, σαφώς πολύ μεγαλύτερος από εκείνον που προέκυψε από τις αρχικές παραμέτρους ($f(x_1, x_2, x_3) = 0.596$). Υπάρχουν διάφορες τιμές παραμέτρων οι οποίες αυξάνουν τον δείκτη δικαιοσύνης. Μάλιστα αρκεί τουλάχιστον να αλλάξει μόνο μία από αυτές για την επίτευξη καλύτερης δικαιοσύνης από την αρχική. Ωστόσο. οι τιμές της *εικόνας 13* έφεραν το καλύτερο δυνατό αποτέλεσμα από αυτές που εξετάστηκαν, για αυτό και επιλέχθηκαν αυτές.

3) Μεγαλύτερο μέγεθος buffer δεν σημαίνει πάντα καλύτερη απόδοση. Αυτό επιβεβαιώνεται εξετάζοντας τις περιπτώσεις DropTailQueue πρώτα μεγέθους 67 και μετά οποιουδήποτε μεγαλύτερου μεγέθους, έστω 80. Και στις δύο περιπτώσεις η απόδοση είναι η ίδια, αφού παράγεται ο ίδιος *πίνακας 11*.

Πίνακας μέτρων packetReceived:vector(packetBytes)	
Receiver2 από Sender1	2099 πακέτα
Receiver2 από Sender2	943 πακέτα
Receiver3 από Sender3	7222 πακέτα
Σύνολο	**10264 πακέτα**

Πίνακας 11: Αναφερόμενος στο ερώτημα 2.3 (Droptail)

Παρατηρείται η ίδια συμπεριφορά και για RedDropperQueue μεγεθών 31 και 40, όπως φαίνεται και από τον παραγόμενο *πίνακα 12*.

Πίνακας μέτρων packetReceived:vector(packetBytes)	
Receiver2 από Sender1	1034 πακέτα
Receiver2 από Sender2	437 πακέτα
Receiver3 από Sender3	4014 πακέτα
Σύνολο	**5485 πακέτα**

Πίνακας 12: Αναφερόμενος στο ερώτημα 2.3 (RED)

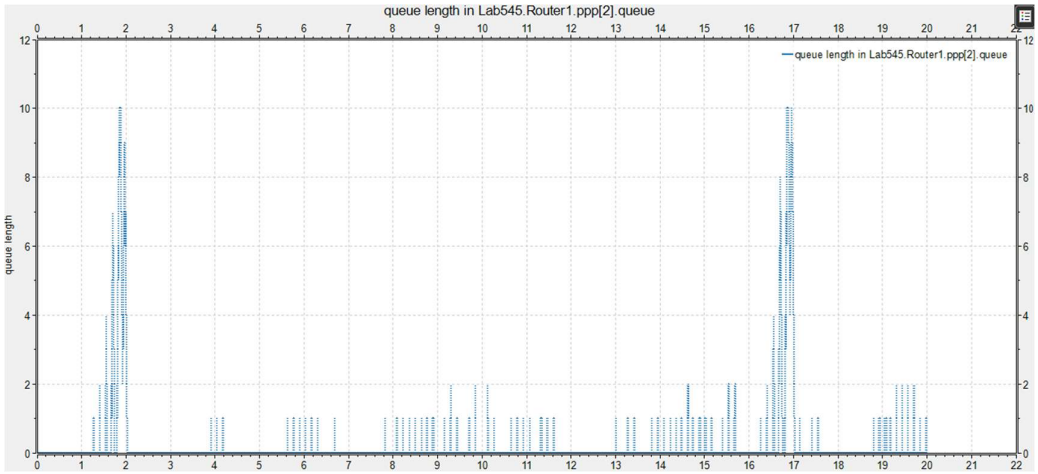
Στην περίπτωση της ουράς DropTail για την αύξηση την δικαιοσύνης, είτε αυξάνουμε το μέγεθος της ουράς, είτε βελτιώνουμε το δίκτυο (μείωση delay και αύξηση datarate). Κατά την αύξηση του μεγέθους της ουράς, η δικαιοσύνη παύει να αυξάνεται από την τιμή 67 και μετά, καθώς τα delay και datarate του δικτύου αποτελούν bottleneck, ενώ κατά την βελτίωση του δικτύου, το μέγεθος της ουράς εμποδίζει την αύξηση του δείκτη δικαιοσύνης από ένα σημείο και μετά, καθιστώντας αυτό ως bottleneck.

Στην περίπτωση της ουράς RED, η δικαιοσύνη αυξάνεται, είτε αυξάνοντας το delay και μειώνοντας το datarate των συνδέσεων του δικτύου, είτε ρυθμίζοντας κατάλληλα τις παραμέτρους της ουράς. Πιο συγκεκριμένα ως προς τις παραμέτρους, η αύξηση του δείκτη δικαιοσύνης επιτυγχάνεται:

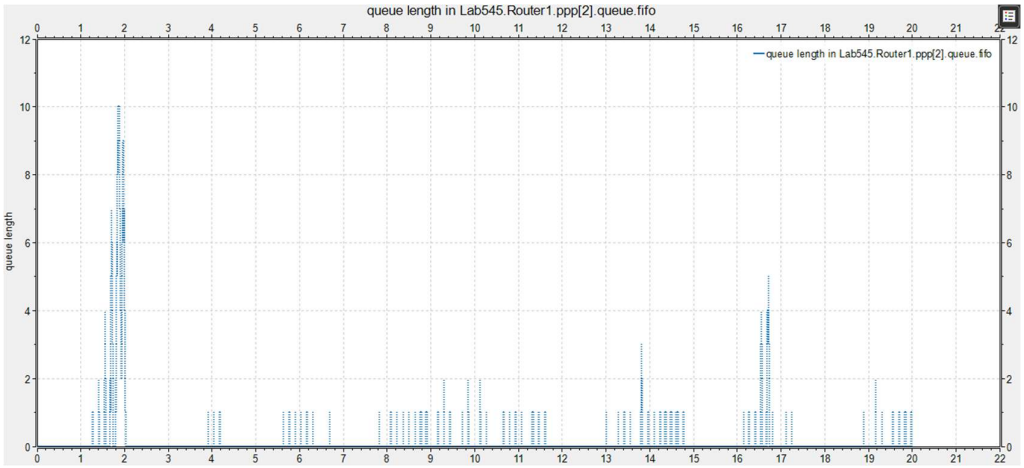
- Μειώνοντας το μέγεθος της ουράς
- Αυξάνοντας το w_q της ουράς
- Αυξάνοντας το max_{th} της ουράς
- Μειώνοντας το min_{th} της ουράς
- Μειώνοντας το max_p της ουράς

Στην αύξηση της δικαιοσύνης από τις ρυθμίσεις του δικτύου bottleneck αποτελούν τα χαρακτηριστικά τις ουράς, ενώ στην σε εκείνη από τις παραμέτρους της ουράς το δίκτυο.

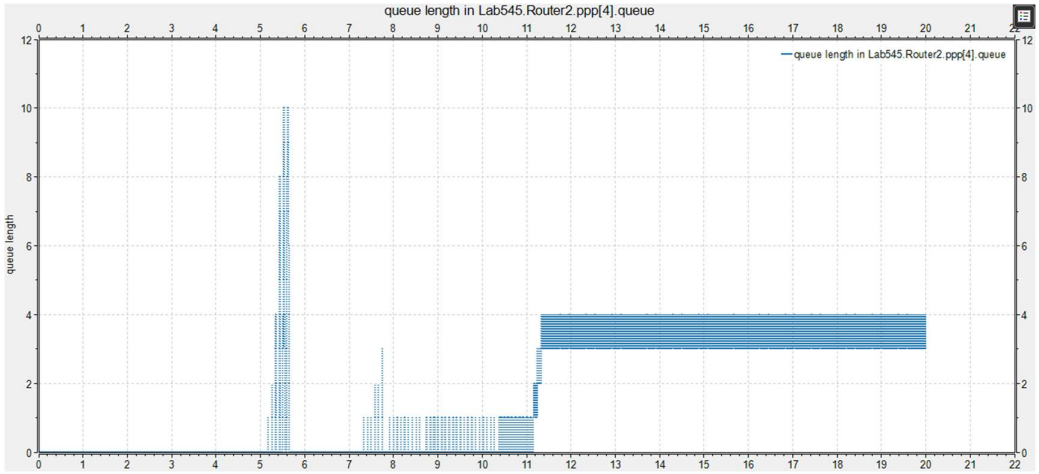
4) Για καθένα από τους δύο τύπους ουράς, 2 είναι τα queueLength:vector των results που παρουσιάζουν ενδιαφέρον· τα Lab545.Router1.ppp[2].queue (*εικόνα 15.α*) και Lab545.Router1.ppp[2].queue.fifo (*εικόνα 15.β*) , τα οποία αντιστοιχούν στην ουρά του Router1 για την προώθηση των πακέτων προς το Router2 (για DropTail και RED αντίστοιχα), και τα Lab545.Router2.ppp[4].queue (*εικόνα 15.γ*) και Lab545.Router2.ppp[4].queue.fifo (*εικόνα 15.δ*), τα οποία αντιστοιχούν στην ουρά του Router2 για την προώθηση των πακέτων προς το Receiver3. Οι γραφικές παραστάσεις των παραπάνω vectors σε συνάρτηση με τον χρόνο φαίνονται παρακάτω.



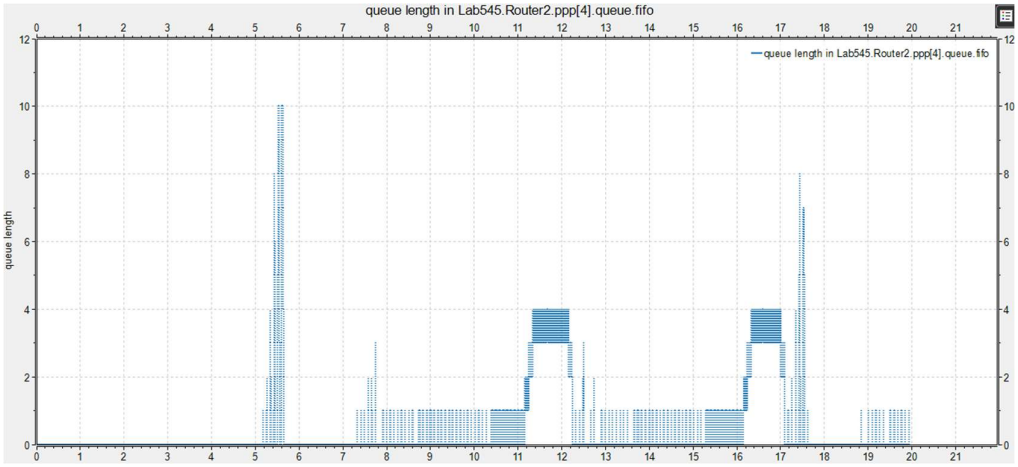
Εικόνα 15.α: Γραφική παράσταση Lab545.Router1.ppp[2].queue / time (DropTail) για το ερώτημα 2.4



Εικόνα 15.β: Γραφική παράσταση Lab545.Router1.ppp[2].queue.fifo / time (RED) για το ερώτημα 2.4



Εικόνα 15.γ: Γραφική παράσταση Lab545.Router2.ppp[4].queue / time (DropTail) για το ερώτημα 2.4



Εικόνα 15.δ: Γραφική παράσταση Lab545.Router2.ppp[4].queue.fifo / time (RED) για το ερώτημα 2.4

Συγκρίνοντας την *εικόνα 15.α* με την *15.β* βλέπουμε ότι το μέγεθος της ουράς RED είναι κατά μέσο όρο μικρότερη από εκείνη της DropTail, το οποίο είναι λογικό, καθώς, όπως έχει προαναφερθεί, η RED κάνει drop τυχαία πακέτα για την επίτευξη μεγαλύτερης δικαιοσύνης. Η διαφορά των μέσων όρων των μεγεθών των δύο τύπων ουρών επιβεβαιώνεται από τα αριθμητικά αποτελέσματα των προσομοιώσεων (Droptail: 2.016506 pk, RED: 1.276899 pk). Συγκρίνοντας την *εικόνα 15.γ* με την *15.δ* βλέπουμε τον προβληματικό διαμοιρασμό πόρων της DropTail, αφού στην πρώτη περίπτωση από ένα σημείο και μετά τα πακέτα του Sender3 φαίνονται να έχουν σταθερή και αποκλειστική θέση στην ουρά. Αυτό το πρόβλημα όμως στην δεύτερη περίπτωση δεν υπάρχει λόγω του τυχαίου packet dropping της RED, το οποίο βέβαια οδηγεί σε μικρότερο μέσο μέγεθος ουράς (Droptail: 3.075337 pk, RED: 1.618712 pk).