Σχολή Θετικών Επιστημών & Τεχνολογίας

Μεταπτυχιακή Εξειδίκευση στα Πληροφοριακά Συστήματα

Διπλωματική Εργασία

«Μοντέλα απωλειών πολυδιάστατης κίνησης και εφαρμογές σε σύγχρονα τηλεπικοινωνιακά δίκτυα»

Κωνσταντίνος Λώλης

Επιβλέπων καθηγητής: Ιωάννης Μοσχολιός

ΠΑΤΡΑ, ΜΑΪΟΣ 2026

ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

«Μοντέλα απωλειών πολυδιάστατης κίνησης και εφαρμογές σε σύγχρονα τηλεπικοινωνιακά δίκτυα»

Κωνσταντίνος Λώλης

Επιτροπή Επίβλεψης Διπλωματικής Εργασίας

Επιβλέπων Καθηγητής:
Ιωάννης Μοσχολιός
Μέλος ΣΕΠ, ΕΑΠ

Συν-Επιβλέπουσα Καθηγήτρια:
Ροδάνθη Τσώνη
Μέλος ΣΕΠ, ΕΑΠ

ΠΑΤΡΑ, ΜΑΪΟΣ 2026

στον γιο μου Σπύρο

και στη γυναίκα μου Κατερίνα

Ευχαριστίες

Ιδιαίτερες ευχαριστίες οφείλω στον επιβλέποντα καθηγητή μου, κ. Ιωάννη Μοσχολιό, για την υπομονή και την προθυμία του να με στηρίζει σε κάθε στάδιο της εκπόνησης της παρούσας εργασίας. Η συμβολή του δεν περιορίστηκε μόνο στην παροχή επιστημονικών οδηγιών, αλλά περιλάμβανε και ουσιαστική ενθάρρυνση, γόνιμη συζήτηση ιδεών και έμπνευση για περαιτέρω εμβάθυνση στο αντικείμενο, γεγονός που κατέστησε τη διαδικασία μάθησης και έρευνας ακόμη πιο εποικοδομητική και ενδιαφέρουσα.

Περίληψη

Η παρούσα εργασία εστιάζει: α) στη μελέτη, ανάλυση και προσομοίωση μοντέλων απωλειών πολυδιάστατης κίνησης, τα οποία αξιοποιούνται για την περιγραφή συστημάτων απωλειών που εξυπηρετούν κλήσεις από διαφορετικές κατηγορίες κίνησης και β) στις εφαρμογές των μοντέλων αυτών σε σύγχρονα τηλεπικοινωνιακά δίκτυα.

Κύρια χαρακτηριστικά των κλήσεων είναι: α) ο τυχαίος τρόπος άφιξής τους (κατανομή *Poisson*) και β) η δυνατότητα προσαρμογής των απαιτήσεων σε εύρος ζώνης κατά την είσοδό τους στο σύστημα. Στόχος της εργασίας είναι ο υπολογισμός βασικών παραμέτρων απόδοσης του συστήματος, με βάση αναλυτικές προσεγγίσεις καθώς και μέσω προσομοιώσεων.

Αρχικά, εξετάζεται το κλασικό μοντέλο απωλειών *Erlang B*, το οποίο αποτελεί το απλούστερο μοντέλο μονοδιάστατης κίνησης και περιγράφει ένα σύστημα με πεπερασμένη χωρητικότητα εξυπηρετητών, χωρίς δυνατότητα αναμονής των κλήσεων. Στη συνέχεια, μελετάται μία επέκταση του μοντέλου αυτού, στην οποία παρατηρούνται απώλειες εξυπηρετητών και η αποκατάστασή τους πραγματοποιείται σταδιακά.

Ιδιαίτερη έμφαση δίνεται στο μοντέλο *Erlang Multirate Loss Model (EMLM)*, το οποίο επιτρέπει την εξυπηρέτηση πολλαπλών κατηγοριών κίνησης με διαφορετικές απαιτήσεις σε πόρους. Στο πλαίσιο αυτό, αναλύονται πολιτικές διαχείρισης όπως η πλήρης διάθεση πόρων *(CS)* και η δέσμευση εύρους ζώνης *(BR)*.

Τέλος, εξετάζονται προηγμένες επεκτάσεις του *EMLM*, όπως τα μοντέλα επανακλήσεων *(SRM, MRM)*, στα οποία οι κλήσεις έχουν τη δυνατότητα να προσπαθήσουν ξανά για εξυπηρέτηση μετά την αρχική τους απόρριψη, συμβάλλοντας έτσι στη βελτίωση της συνολικής απόδοσης και της αξιοποίησης των διαθέσιμων πόρων καθώς και τα μοντέλα κατωφλίων *(STM, MTM, CDTM)*, τα οποία ενσωματώνουν πολιτικές ελέγχου αποδοχής κλήσεων, επιτρέποντας την αποτελεσματικότερη διαχείριση της συμφόρησης του δικτύου και την προτεραιοποίηση διαφορετικών κατηγοριών κίνησης.

Συνολικά, οι επεκτάσεις αυτές συμβάλλουν στη βελτίωση της απόδοσης και της καλύτερης κατανομής των πόρων του συστήματος.

Λέξεις – Κλειδιά : Μοντέλα απωλειών, προσομοίωση, απώλεια ζεύξης, πιθανότητα απώλειας κλήσεων, επανακλήση, κατώφλι

Multidimensional Traffic Loss Models and Their Applications in Modern Telecommunications Networks

Konstantinos Lolis

Abstract

The present thesis focuses on: a) the study, analysis, and simulation of multi-dimensional traffic loss models, which are utilized to describe loss systems serving calls from different traffic classes, and b) the application of these models in modern telecommunication networks.

The main characteristics of the calls under consideration are: a) their random arrival process, which follows a Poisson distribution, and b) the ability to adapt their bandwidth requirements upon entering the system. The objective of this work is the evaluation of key system performance metrics, based on analytical approaches as well as simulation techniques.

Initially, the classical Erlang B loss model is examined, which constitutes the simplest model of single-dimensional traffic and describes a system with finite server capacity, without allowing call queuing. Subsequently, an extension of this model is studied, in which server losses are considered and their restoration occurs gradually.

Particular emphasis is placed on the Erlang Multirate Loss Model (EMLM), which enables the service of multiple traffic classes with different resource requirements. In this context, resource management policies such as Complete Sharing (CS) and Bandwidth Reservation (BR) are analyzed.

Finally, advanced extensions of the EMLM are investigated, including retry models (SRM, MRM), where calls are allowed to attempt re-service after an initial rejection, thereby improving overall system performance and resource utilization. In addition, threshold-based models (STM, MTM, CDTM) are examined, which incorporate call admission control policies, enabling more efficient congestion management and prioritization of different traffic classes.

Overall, these extensions contribute to the improvement of system performance and the more efficient allocation of available resources.

Keywords: Loss models, simulation, link loss, call blocking probability, retry, threshold

Περιεχόμενα

Κατάλογος Εικόνων / Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες & Ακρωνύμια

4G	Fourth generation mobile networks – Τέταρτη γενιά κινητών δικτύων
5G	Fifth generation mobile networks– Πέμπτη γενιά κινητών δικτύων
ADSs	Anomaly based detection systems – Συστήματα ανίχνευσης βασισμένα σε ανωμαλίες
BBU	Baseband Unit – Μονάδες βασικής ζώνης
b.u.	Bandwidth unit – Μονάδα εύρους ζώνης
bps	Bits per second – Bits ανά δευτερόλεπτο
BR	Bandwidth reservation – Πολιτική δέσμευσης εύρους ζώνης
BS	Base station – Σταθμός βάσης
CBP	Call blocking probability – Πιθανότητα απώλειας κλήσης
CC	Call congestion – Συμφόρηση κλήσεων
CDTM	Connection dependent threshold model - Μοντέλο Κατωφλίου Εξαρτώμενο από τις Συνδέσεις
C-RAN	Cloud radio access network – Ραδιοπρόσβαση βασισμένη στο υπολογιστικό νέφος
CS	Complete sharing - Πολιτική πλήρους διάθεσης
eMBB	Enhanced mobile broadband – Ενισχυμένο κινητό ευρυζωνικό δίκτυο
EMLM	Erlang multirate loss model – Σύστημα απωλειών πολυδιάστατης κίνησης
erl	The Erlang unit of traffic load - Μονάδα μέτρησης του φορτίου κίνησης
FIFO	First in – first out – Ο πρώτος εισερχόμενος εξέρχεται και πρώτος
GB	Global balance – Σφαιρική ισορροπία
GoS	Grade of service - Βαθμός εξυπηρέτησης
IoT	Internet of Things - Διαδίκτυο των Πραγμάτων
LB	Local balance – Τοπική ισορροπία
LEO	Low earth orbit – Χαμηλή τροχιά επί της Γης
mMTC	Massive machine type communications – Μαζικές επικοινωνίες τύπου μηχανής
MRM	Multi retry model – Μοντέλο πολλαπλών επανακλήσεων
MSS	Mobile Satellite System – Κινητά δορυφορικά συστήματα
MTM	Multi threshold model- Μοντέλο πολλαπλών κατωφλίων
OFDMA	Orthogonal frequency division multiple access – Πολυπλεξία ορθογώνιας διαίρεσης συχνότητας με πολλαπλή πρόσβαση
PASTA	Poisson arrivals see time averages - Οι αφίξεις Poisson βλέπουν τους χρονικούς μέσους όρους
PON	Passive optical networks – Παθητικά οπτικά δίκτυα
QoS	Quality of service – Ποιότητα εξυπηρέτησης
RRH	Remote radio head – Απομακρυσμένες κεφαλές ραδιοσυχνοτήτων
SFC	Satellite Fixed Cells – Κυψελοειδή δομή σταθερή ως προς τον δορυφόρο
SRM	Single retry model – Μοντέλο μονής επανακλήσης
STM	Single threshold model – Μοντέλο ενός κατωφλίου
TC	Time congestion – Χρονική συμφόρηση
TDM	Time division multiplexing – Πολυπλεξία διαίρεσης χρόνου
TH	Threshold – Κατώφλι
tr	Trunk reservation – Δέσμευση χωρητικότητας
TWDM	Time/Wavelength division multiplexing – Υβριδική πολυπλεξία των

	τεχνικών TDM και WDM
URLLC	Ultra reliable low latency communication – Επικοινωνία υπερυψηλής αξιοπιστίας και χαμηλής καθυστέρησης
WDM	Wavelength division multiplexing – Πολυπλεξία διαίρεσης μήκους κύματος

1. Βασικές Έννοιες

1.1 Εισαγωγή

Το συγκεκριμένο κεφάλαιο είναι εισαγωγικό και θα προσπαθήσει να εξηγήσει κάποιες βασικές έννοιες της *θεωρίας της τηλεπικοινωνιακής κίνησης* (*teletraffic theory*), που είναι χρήσιμες στα επόμενα κεφάλαια. Γενικά, η *θεωρία της τηλεπικοινωνιακής κίνησης* είναι η εφαρμογή της θεωρίας των πιθανοτήτων σε τηλεπικοινωνιακά συστήματα και δίκτυα επικοινωνιών και έχει σαν σκοπό να λύσει προβλήματα στο σχεδιασμό, την απόδοση και τη διαχείριση του συστήματος, όπως π.χ. να μετρήσει απώλειες κλήσεων για να διαπιστώσει τις ανάγκες του συστήματος σε χωρητικότητα κ.τ.λ [1], [2], [24].

Ένα απλό μοντέλο που θα αναλύσουμε παρακάτω είναι το μοντέλο απωλειών *Erlang* και η κλασσική φόρμουλα *Erlang B* που υπολογίζει, όπως θα δούμε, πόσο εύκολα χάνονται κλήσεις σε ένα δίκτυο π.χ. τηλεφωνικό, με συγκεκριμένη χωρητικότητα. Οι κλήσεις έρχονται στο σύστημα ακολουθώντας μια διαδικασία *Poisson* και εξυπηρετούνται με χρόνο εκθετικά κατανεμημένο. Η συγκεκριμένη φόρμουλα υπολογίζει την πιθανότητα απώλειας κλήσεων. Όταν η πιθανότητα αυτή υπερβαίνει αποδεκτά επίπεδα, είναι απαραίτητη η παρέμβαση στο δίκτυο, όπως η αύξηση της χωρητικότητας ή η βελτιστοποίηση του χρόνου εξυπηρέτησης των κλήσεων [1], [2], [5], [24].

Η θεωρία της τηλεπικοινωνιακής κίνησης δεν περιορίζεται μόνο στα δίκτυα επικοινωνιών, αλλά βρίσκει εφαρμογή και σε άλλους τομείς, όπως στη διαχείριση της εναέριας κυκλοφορίας, στη λειτουργία των μονάδων εντατικής θεραπείας στα νοσοκομεία, καθώς και στη διαχείριση της εξυπηρέτησης πελατών σε συστήματα λιανικής πώλησης, όπως τα ταμεία πολυκαταστημάτων.

1.2 Μαρκοβιανές αλυσίδες

Όταν αναφερόμαστε σε Μαρκοβιανή αλυσίδα *(Markov Chain*), ουσιαστικά εννοούμε ένα σύστημα που περιλαμβάνει ένα σύνολο καταστάσεων και μπορούμε να μεταβούμε από τη μία κατάσταση στην άλλη στιγμιαία [10].

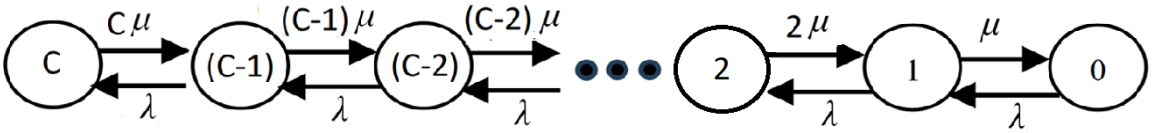
Βασικό κριτήριο είναι ότι δεν μπορούμε να βρισκόμαστε ταυτόχρονα σε πάνω από μία κατάσταση και η μετάβαση σε μία επόμενη κατάσταση δεν εξαρτάται από τις προηγούμενες καταστάσεις αλλά μόνο από την τρέχουσα κατάσταση. Εδώ μετράει μόνο το παρόν και όχι το παρελθόν [28]. Έτσι στο πεδίο των πιθανοτήτων η δεσμευμένη πιθανότητα να μεταβούμε σε επόμενη κατάσταση εξαρτάται μόνο από την τρέχουσα κατάσταση και όχι τις προηγούμενες [10].

Συγκεκριμένα, αν με $X(t_k)$ συμβολίσουμε την κατάσταση ενός συστήματος στο χρόνο t_k με k = 0, 1, 2, 3… και αν $X(t_n)$ είναι η παρούσα κατάσταση στον παρόντα χρόνο t_n , $n \in k$ τότε η επόμενη κατάσταση είναι η $X(t_{n+1})$ τη χρονική στιγμή t_{n+1}και η προηγούμενη είναι $X(t_{n-1})$ τη χρονική στιγμή t_{n-1}. Η πιθανότητα να βρεθεί το σύστημα στην επόμενη κατάσταση $X(t_{n+1})$ εξαρτάται μόνο από την τρέχουσα κατάσταση $X(t_n)$ τη χρονική στιγμή t_n [24], [25]:

$$P_{\mu\varepsilon\tau\acute{\alpha}\beta\alpha\sigma\eta\varsigma\_\kappa\alpha\tau\acute{\alpha}\sigma\tau\alpha\sigma\eta\varsigma} = P(\ X(t_{n+1}) = X_{n+1} |\ X(t_n) = X_n\ ) \qquad (1.1)$$

Στον τομέα των τηλεπικοινωνιακών δικτύων οι καταστάσεις της Μαρκοβιανής αλυσίδας εκφράζουν το πλήθος των κατειλημμένων μονάδων εύρους ζώνης (*bandwidth units, b.u.*) και μπορούν να αναπαρασταθούν μέσω διαγραμμάτων κάνοντας χρήση γράφων.

Στους γράφους αυτούς οι κόμβοι σχηματοποιούν τις καταστάσεις και οι ακμές τις μεταβάσεις. Σε κάθε ακμή αντιστοιχίζεται ένας ρυθμός μετάβασης.



Σχήμα 1: Μαρκοβιανή αλυσίδα

Στο Σχήμα 1 έχει σχεδιαστεί ένας γράφος με τους κόμβους και τις ακμές και αναπαριστά μία Μαρκοβιανή αλυσίδα. Ο κάθε κόμβος συμβολίζει τις κατειλημμένες μονάδες εύρους ζώνης σε ένα σύστημα με χωρητικότητα C *b.u.* και η κάθε ακμή το ρυθμό άφιξης της κλήσης λ ή το ρυθμό εξυπηρέτησης μ της κλήσης.

Στα συστήματα απώλειας κλήσεων που μελετάμε, οι τυχαίες αφίξεις των κλήσεων και οι εξυπηρετήσεις αυτών από το σύστημα σε τυχαίο χρόνο ακολουθούν την εκθετική κατανομή

και έτσι διασφαλίζεται και η ιδιότητα της Μαρκοβιανής αλυσίδας που εξαρτάται από το παρόν και όχι από το παρελθόν [10], [28].

1.3 Εκθετική κατανομή του χρόνου εξυπηρέτησης της κλήσης

Έστω μ ο μέσος ρυθμός εξυπηρέτησης της κλήσης, επειδή η εξυπηρέτηση της κλήσης ολοκληρώνεται σε τυχαίο χρόνο είναι ανεξάρτητη του χρόνου. Ακόμα, κατά το χρονικό διάστημα $(t + \Delta t)$ με $\Delta t \to 0$, η πιθανότητα να ολοκληρωθεί η εξυπηρέτηση είναι μΔt ενώ $1 - \mu\Delta t$ η πιθανότητα να μην ολοκληρωθεί η κλήση στο Δt. Αν θεωρήσουμε ότι ένα χρονικό διάστημα t τεμαχίζεται σε n μικρά κομμάτια τότε $\Delta t = \frac{t}{n}$, άρα $1 - \mu\Delta t = 1 - \mu\frac{t}{n}$ [24].

Έτσι, η πιθανότητα να μην τερματιστεί η κλήση στο διάστημα αυτό είναι [24]:

$$H(t) = \left(1 - \mu\frac{t}{n}\right)^n \tag{1.2}$$

Για $\Delta t \to 0$ και $n \to \infty$ έχουμε:

$$H(t) = \lim_{n\to\infty}\left(1 - \mu\frac{t}{n}\right)^n \tag{1.3}$$

Είναι γνωστό ότι [24][25]:

$$e^x = \lim_{n\to\infty}\left(1 + \frac{x}{n}\right)^n \tag{1.4}$$

Τελικά έχουμε τον τύπο της εκθετικής κατανομής [24]:

$$H(t) = e^{-\mu t} \tag{1.5}$$

Αποδείξαμε ότι ο χρόνος εξυπηρέτησης της κλήσης είναι εκθετικά κατανεμημένος με μέσο ρυθμό εξυπηρέτησης μ. Η υπόθεση της εκθετικής κατανομής του χρόνου εξυπηρέτησης, συμφωνεί απολύτως με τον πραγματικό χρόνο εξυπηρέτησης των τηλεπικοινωνιακών συστημάτων. Επιπλέον, η υπόθεση αυτή έχει ευρέως εφαρμοστεί στη θεωρία τηλεπικοινωνιακής κίνησης επειδή απλουστεύεται η θεωρητική ανάλυση [24].

1.4 Διαδικασία *Poisson*

Πολλές φορές θα συναντήσουμε στα συστήματα απώλειας κλήσεων την αναφορά στην τυχαιότητα της άφιξης των κλήσεων ή στο ότι η κλήση ακολουθεί τη στοχαστική διαδικασία *Poisson*.

Στη διαδικασία *Poisson* ισχύει ότι οι κλήσεις είναι ανεξάρτητες, έτσι το πλήθος των αφίξεων των κλήσεων σε ένα συγκεκριμένο χρονικό διάστημα είναι ανεξάρτητο από τις αφίξεις εκτός αυτού, δηλαδή σε οποιοδήποτε άλλο χρονικό διάστημα διαφορετικό από αυτό [24].

Σε πολύ μικρό χρονικό διάστημα *(t, Δt+t)* με $\Delta t \to 0$ μπορεί να αφιχθεί μόνο μία κλήση με πιθανότητα $\lambda \Delta t$ ή καμία με πιθανότητα $1 - \lambda \Delta t$, ανεξάρτητα από το ιστορικό αφίξεων των κλήσεων στο σύστημα, δηλαδή ανεξάρτητα του χρόνου *t*. Επίσης, η τυχαιότητα των κλήσεων οφείλεται και από τις άπειρες πηγές που τις στέλνουν [24].

Όπως δείξαμε στην ενότητα 1.1.2 έτσι και εδώ αποδεικνύεται με τον ίδιο τρόπο ότι ο μέσος ρυθμός άφιξης λ των κλήσεων ακολουθεί την εκθετική κατανομή $e^{-\lambda t}$ [24].

Με άλλα λόγια, αν υποθέσουμε ότι κατά το χρονικό διάστημα t δεν έχει αφιχθεί κάποια κλήση τότε δεν επηρεάζεται η πιθανότητα να αφιχθεί κάποια κλήση σε επόμενο χρονικό διάστημα t' [24]. Αυτή η πρόταση συνδέεται με τις Μαρκοβιανές αλυσίδες όπως είδαμε στην Ενότητα 1.1.

1.5 Η ιδιότητα *PASTA*

Στα συστήματα απώλειας κλήσεων μελετάμε αφίξεις που ακολουθούν την τυχαιότητα της διαδικασίας *Poisson*. Η ιδιότητα *PASTA (Poisson Arrivals See Time Average)* έχει να κάνει με το ότι η πιθανότητα να έχουμε κλήσεις που εξυπηρετούνται από ένα σύστημα που βρίσκεται σε σταθερή κατάσταση είναι ίση με την πιθανότητα μία νεοεισερχόμενη κλήση να βρει το σύστημα στην κατάσταση αυτή.

Με άλλα λόγια η πιθανότητα το σύστημα να εξυπηρετεί *n* πλήθος κλήσεων τη στιγμή που οι κλήσεις εισέρχονται στο σύστημα, ισούται με την πιθανότητα το σύστημα να εξυπηρετεί *n* κλήσεις σε οποιαδήποτε άλλη χρονική στιγμή [24].

Ουσιαστικά η κλήση που φτάνει βλέπει την τυπική κατάσταση του συστήματος, χωρίς να αναλύεται ειδικά η στιγμή της άφιξης. Έτσι απλουστεύονται οι αναλύσεις και οι υπολογισμοί [24].

1.5 Φορτίο κίνησης α

Ως φορτίο κίνηση α θεωρούμε το άθροισμα των χρόνων εξυπηρέτησης h των κλήσεων κατά τη χρονική περίοδο T [24].

$$a = \frac{\sum h}{T} \tag{1.5}$$

Παρ' όλο που το φορτίο κίνησης α είναι ένας καθαρός αριθμός, η μονάδα μέτρησης είναι το erl προς τιμή του *Δανού μαθηματικού Agner Krarup Erlang (1878–1929)* [11] που θεωρείται, μεταξύ άλλων, πρωτοπόρος της μαθηματικής ανάλυσης των τηλεπικοινωνιών.

Ιδιότητες του Φορτίου Κίνησης [24]:

- $\alpha = ch$ (1.6)

  με c να είναι το πλήθος των κλήσεων που φτάνουν στο σύστημά μας σε δεδομένη χρονική στιγμή επί h που είναι ο μέσος χρόνος εξυπηρέτησής τους.

  $c = \frac{n}{T}$ (1.7)

  δηλαδή το πλήθος των κλήσεων n την περίοδο παρατήρησης του συστήματος *(observation period)* T, παριστάνει ουσιαστικά το μέσο ρυθμό άφιξης των κλήσεων και συμβολίζεται, πολλές φορές, και με το γράμμα λ. Ο χρόνος εξυπηρέτησης συνδέεται με το μέσο ρυθμό εξυπηρέτησης που συμβολίζεται με μ μέσω της σχέσης.

  $\mu = \frac{1}{h}$. (1.8)

  Επίσης, από τα παραπάνω εξάγεται και η σχέση:

  $\alpha T = nh$ (1.9)

  Δηλαδή, το γινόμενο του φορτίου κίνησης α επί το χρόνο παρατήρησης T *(observation period)* είναι ίσο με το γινόμενο των κατειλημμένων γραμμών επί το μέσο χρόνο εξυπηρέτησης της κλήσης.

- Το φορτίο κίνησης είναι ίσο με τον αριθμό των κλήσεων που δημιουργούνται σε χρόνο όσο είναι ο χρόνος εξυπηρέτησής τους *h*. Για παράδειγμα, αν έχουμε 10 κλήσεις σε έναν χρόνο παρατήρησης *(observation time)* 30 λεπτών, τότε στο ερώτημα πόσο φορτίο κίνησης έχουμε όταν μία κλήση κατά μέσο όρο έχει χρονική διάρκεια εξυπηρέτησης 7.5 λεπτά, η απάντηση είναι (7.5 * 10) / 30 = 2.5 *erl*.
- Το φορτίο κίνησης που μεταφέρεται από μία μόνο γραμμή είναι ίσο με την πιθανότητα ότι η γραμμή είναι κατειλημμένη. Η ιδιότητα αυτή δείχνει ότι μία γραμμή δεν μπορεί να μεταφέρει πάνω από 1 *erl*. (Η πιθανότητα είναι μικρότερη ή ίση του 1.). Αν αθροίσουμε τις πιθανότητες αυτών των γραμμών τότε αυτό είναι το φορτίο κίνησης *α*. Να διευκρινίσουμε ότι η τιμή του *α* μπορεί να είναι πάνω από 1 erl, για παράδειγμα αν έχουμε τέσσερις γραμμές τότε μπορούμε να αθροίσουμε τα φορτία κίνησης των τεσσάρων γραμμών και να έχουμε μία τιμή με μέγιστο το 4 *erl*.
- Το φορτίο κίνησης, μιας δέσμης γραμμών είναι ίσο με τη μέση αναμενόμενη τιμή των κατειλημμένων γραμμών της δέσμης αυτής. Η πρόταση αυτή είναι σημαντική γιατί βγαίνει από το «πλάνο» ο χρόνος εξυπηρέτησης *h*. Για παράδειγμα, αν έχουμε 5 γραμμές και το φορτίο κίνησης *α = 5 erl*, η απάντηση στο πόσο χρόνο είναι κατειλημμένες οι γραμμές αυτές είναι:

  «Οι γραμμές αυτές παραμένουν πλήρως κατειλημμένες καθ’ όλη τη διάρκεια της περιόδου παρατήρησης (observation period).»

1.7 Βαθμός Εξυπηρέτησης *(Grade of Service, GoS)*

Ο βαθμό εξυπηρέτησης *GoS* είναι ένας δείκτης της ποιότητας του συστήματος που εξυπηρετεί τους χρήστες και εννοούμε το ποσοστό των κλήσεων που χάνονται σε σχέση με αυτές τις κλήσεις που προσφέρθηκαν [24]. Αυτή η έννοια του *GoS* είναι εξαιρετικά σημαντική στη μελέτη των συστημάτων απώλειας κλήσεων και συναντάται πολλές φορές και με το σύμβολο *B*, όπου εκφράζουμε την πιθανότητα απώλειας κλήσεων ενός συστήματος. Αυτόν τον συμβολισμό (*B*) θα τον διατηρήσουμε και παρακάτω στα πλαίσια αυτής της εργασίας.

Κάποια χαρακτηριστικά του *GoS* είναι ότι επειδή εκφράζει πιθανότητα είναι πάντα μικρότερος ή ίσος της μονάδας και όσο αυξάνεται τόσο χειρότερα αποδίδει το σύστημα.

Παρακάτω θα παρουσιάσουμε τρία κλάσματα που εκφράζουν τον GoS [24]:

1. Συμφόρηση κλήσεων (call congestion)

$$GoS = \frac{\text{συνολικές κλήσεις που χάνονται } (total\ call\ lost)}{\text{συνολικές κλήσεις που προσφέρονται } (total\ calls\ offered)}$$

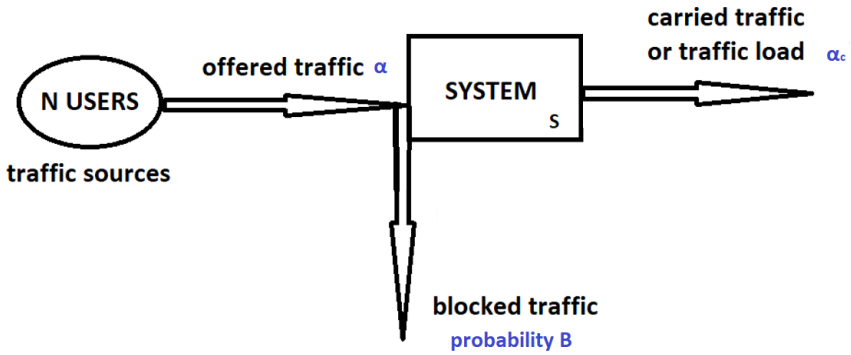
2. Η κίνηση που χάνεται *(traffic congestion)*

$$GoS = \frac{\text{κίνηση που χάνεται } (traffic\ lost)}{\text{κίνηση που προσφέρεται } (traffic\ offered)}$$

3. Η πιθανότητα συμφόρησης στον χρόνο εκφράζει την πιθανότητα όλες οι *b.u.* (π.χ. εξυπηρετητές) να είναι απασχολημένοι *(time congestion probability)*

$$GoS = \frac{\text{συνολικός χρόνος που όλες οι } b.u. \text{ είναι απασχολημένες}}{\text{συνολικό χρόνο παρατήρησης}}$$

Στα συστήματα απωλειών κλήσης που θα μελετήσουμε, θα θεωρήσουμε ότι οι τρεις παραπάνω πιθανότητες θα εκφράζουν το ίδιο μέγεθος που θα το συμβολίζουμε με *B*, όπως αναφέραμε και προηγουμένως .



Σχήμα 2: Σύστημα απώλειας κλήσεων

Σύμφωνα με το Σχήμα 2, αν το πλήθος των χρηστών *N* είναι μεγαλύτερο από το πλήθος των εξυπηρετητών *S (N>>S)* τότε μια κλήση που έρχεται μπορεί να βρει όλους τους εξυπηρετητές κατειλημμένους. Αν συμβεί κάτι τέτοιο τότε η κλήση μπλοκάρεται και χάνεται χωρίς να επηρεαστεί το υπόλοιπο σύστημα.

Γενικά, ισχύει [24]:

Κίνηση που διεκπεραιώνεται = Κίνηση που προσφέρθηκε – Κίνηση που χάθηκε
(carried traffic = offered traffic – blocked traffic)

αν a είναι η κίνηση που προσφέρεται τότε aB είναι η κίνηση που χάνεται, οπότε a_c είναι η κίνηση που εξυπηρετείται [24]:

$$a_c = a - aB = a(1 - B) \tag{1.10}$$

Είναι σημαντικό να υπολογιστεί η πιθανότητα το σύστημα να είναι πλήρως κατειλημμένο όταν έρχεται μια κλήση, γιατί στα μοντέλα που πραγματεύεται αυτή η εργασία, η πιθανότητα απόρριψης είναι πάντα ίδια με την πιθανότητα όλες οι γραμμές του συστήματος να είναι κατειλημμένες [24].

Σε κάποιες περιπτώσεις θεωρούμε ότι το πλήθος των εξυπηρετητών των κλήσεων είναι άπειρο. Αυτό γίνεται γιατί απλουστεύονται κάποιες προσεγγίσεις σε επίπεδο πολυπλοκότητας, παρόλο που μπορεί να λαμβάνουμε προσεγγιστικά αποτελέσματα, συνήθως οριακές τιμές της επίδοσης των συστημάτων, και όχι ακριβή [24].

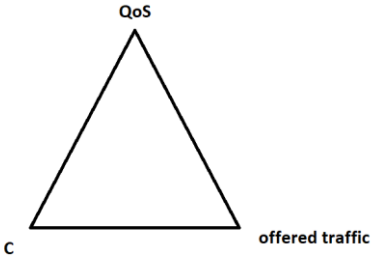
Συμπερασματικά, ξέροντας το GoS ενός συστήματος μπορούμε να δράσουμε καταλλήλως κατανέμοντας σωστά τους πόρους ώστε να καλύψουμε καλύτερα το σύστημά μας, ακόμα, και στη μέγιστη συμφόρησή (*congestion*) του, η οποία μπορεί να προκληθεί από ένα γεγονός όπως για παράδειγμα πολύ μεγάλος αριθμός κλήσεων μετά από έναν σεισμό. Επιπρόσθετα, μπορεί να γίνει σωστότερη διαχείριση των πόρων λαμβάνοντας υπόψη και τα οικονομικά μέσα που διαθέτουμε. Ο στόχος πάντα είναι η ικανοποίηση των απαιτήσεων των χρηστών.

1.8 Το τρίγωνο *C - offered traffic – QoS*

Μια σημαντική έννοια που λαμβάνεται υπόψη για τη σχεδίαση ενός συστήματος είναι η ποιότητα εξυπηρέτησης (*Quality of Service - QoS*), δηλαδή η καλύτερη, χωρίς απώλειες, και ταχύτερη εξυπηρέτηση κλήσεων σε ένα τηλεπικοινωνιακό σύστημα. Ένας από τους δείκτες που μετρούν την QoS είναι ο GoS [24].

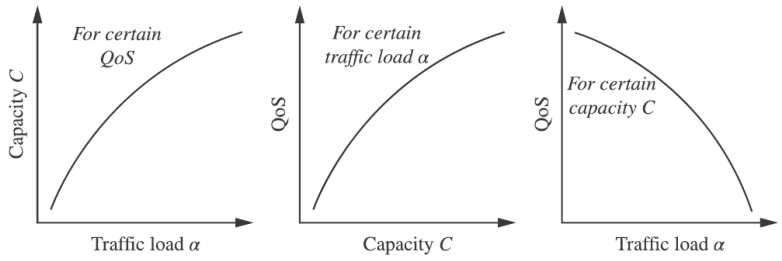
Επομένως, η χωρητικότητα του συστήματος *C* (π.χ. ο αριθμός εξυπηρετητών), η προσφερόμενη κίνηση (offered traffic) (π.χ. το πλήθος των κλήσεων στο χρόνο) και η QoS επηρεάζουν την αποδοτικότητα ενός τηλεπικοινωνιακού συστήματος.

Με βάση το Σχήμα 3 [2], [24] μπορούμε να εξάγουμε κάποια συμπεράσματα είτε μέσω των τριών αυτών παραμέτρων είτε μέσω διαγραμμάτων, Σχήμα 4, ώστε να βοηθήσουμε τη δημιουργία ή βελτίωση των τηλεπικοινωνιακών συστημάτων [2].



Σχήμα 3: Τρίγωνο *QoS, C, offered traffic*

Τα ποσοτικά αποτελέσματα μέσω των αναλυτικών μοντέλων, αποτελούν κρίσιμα στοιχεία για το σχεδιασμό και την αναβάθμιση αυτών των συστημάτων καθώς και ενέργειες που πρέπει να γίνουν σε περιπτώσεις έκτακτων καταστάσεων όπως π.χ. βλάβες κ.α.



Σχήμα 4: Διαγράμματα *QoS, C, α*

Έτσι, τα ερωτήματα που μπορούμε να θέσουμε (π.χ. κατά το σχεδιασμό ενός συστήματος) είναι:

- Αν ξέρουμε τη χωρητικότητα *C* του συστήματος και την εισερχόμενη κίνηση, ποιο είναι το *QoS* που βλέπει ο χρήστης π.χ. ποια η πιθανότητα να μπλοκαριστεί ή να καθυστερήσει;
- Αν ξέρουμε την προσφερόμενη κίνηση και ξέρουμε ποιο είναι το απαιτούμενο *QoS*, πόση πρέπει να είναι η χωρητικότητα *C* του συστήματος για να ανταποκριθεί στο απαιτούμενο *QoS;*
- Αν ξέρουμε τη χωρητικότητα *C* και ξέρουμε το απαιτούμενο *QoS*, πόση είναι η μέγιστη κίνηση που μπορούμε να δεχτούμε, έτσι ώστε να μην επηρεαστεί το *QoS;*

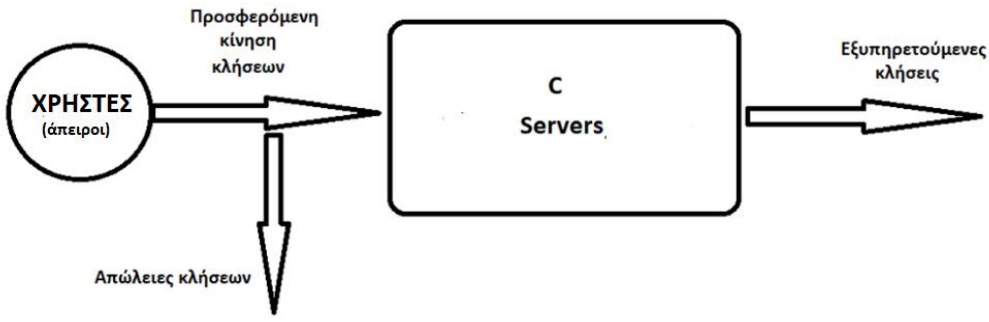
2. Μοντέλο απώλειας κλήσεων *Erlang B*

2.1 Εισαγωγή

Γενικά, ένα σύστημα απώλειας κλήσεων είναι ένα σύστημα που δέχεται κλήσεις με τυχαίο τρόπο, ακολουθώντας τη διαδικασία *Poisson*, με μέσο ρυθμό άφιξης αυτών λ και τις εξυπηρετεί ακολουθώντας εκθετική κατανομή με μέσο ρυθμό εξυπηρέτησης μ. Η κλήση όταν εισέρχεται σε ένα σύστημα με χωρητικότητα *C b.u.*, μπαίνει σε μία διαδικασία απαίτησης μίας μονάδας εύρους ζώνης *b.u.* Ανάλογα με το μοντέλο υλοποίησης η κλήση γίνεται αποδεκτή ή χάνεται. Η απλούστερη εκδοχή μοντέλου αυτής της κατηγορίας, αλλά και η πλέον αντιπροσωπευτική [3], [4], είναι το μοντέλο απωλειών κλήσεων *Erlang B* [5]. Το μοντέλο αυτό χρησιμοποιείται ευρέως για την επίλυση πρακτικών προβλημάτων, όπου ο χρόνος εξυπηρέτησης μπορεί να παρεκκλίνει από την εκθετική κατανομή [4].

2.2 Μοντέλο *Erlang B*

Έστω ότι έχουμε ένα σύστημα, όπως απεικονίζεται στο Σχήμα 5, με χωρητικότητα *C b.u.* και σ' αυτό καταφθάνουν κλήσεις μίας κατηγορίας κίνησης *(K*=1), με τη διαδικασία *Poisson*, με μέσο ρυθμό άφιξης λ, ζητώντας μία μονάδα *b.u.* για να εξυπηρετηθούν. Αν το σύστημα διαθέτει αυτή τη *b.u.* τότε την παραχωρεί με μέσο ρυθμό εξυπηρέτησης μ, με βάση την εκθετική κατανομή, και η κλήση εξυπηρετείται. Αν το σύστημα δεν διαθέτει αυτή τη *b.u.* τότε η κλήση μπλοκάρεται και χάνεται, δηλαδή δεν ακολουθείται καμία πολιτική αναμονής για να προσπαθήσει ξανά.



Σχήμα 5: Το μοντέλο απωλειών κλήσεων *Erlang-B*

Ας υποθέσουμε ότι σε μία χρονική στιγμή t εξυπηρετούνται $X(t)$ εισερχόμενες κλήσεις. Επειδή, η κάθε κλήση δεσμεύει 1 *b.u.* τότε η $X(t)$ εκφράζει, επίσης, και το πλήθος των κατειλημμένων *b.u.* τη χρονική στιγμή t. Άρα $0 \leq X(t) \leq C$. Ας υποθέσουμε, ακόμα, ότι τη χρονική στιγμή $t + \Delta t$ ($\Delta t \to 0$), ο αριθμός των εισερχόμενων κλήσεων είναι n τότε θα ισχύει $X(t + \Delta t) = n$, δηλαδή θα μπορούσε να λεχθεί ότι το σύστημα βρίσκεται στην κατάσταση n.

Επομένως, θα λέμε ότι το σύστημα βρίσκεται στην κατάσταση n στις παρακάτω τρεις περιπτώσεις [24] [26]:

- $X(t) = n$, και καμία κλήση δεν εισέρχεται ούτε εξέρχεται από το σύστημα.
- $X(t) = n-1$, και μόνο μία κλήση εισέρχεται σε χρόνο Δt.
- $X(t) = n+1$, και μόνο μία κλήση που είχε εισέλθει εξέρχεται από σύστημα σε χρόνο Δt.

Έστω ότι $P_n(t + \Delta t)$ είναι η πιθανότητα να βρισκόμαστε στην κατάσταση n τότε αν συμβαίνει η πρώτη περίπτωση, δηλαδή, δεν έχουμε καμία εισερχόμενη ή εξερχόμενη κλήση, η πιθανότητα θα είναι $P_n(t)(1 - \lambda\Delta t - n\mu\Delta t)$.

Να σημειώσουμε εδώ ότι η πιθανότητα μία συγκεκριμένη εισερχόμενη κλήση να αποχωρήσει από το σύστημα σε χρόνο Δt είναι $\mu\Delta t$ και μπορεί να αποχωρήσει είτε η πρώτη, είτε η δεύτερη… είτε η n-οστή εισερχόμενη κλήση και γι' αυτό η πιθανότητα αποχώρησης γίνεται: $n\mu\Delta t$.

Αν, τώρα, συμβαίνει η δεύτερη περίπτωση που έχουμε μία εισερχόμενη κλήση και καμία εξερχόμενη ενώ βρισκόμαστε στην κατάσταση $n-1$ τότε η πιθανότητα να βρισκόμαστε στην n κατάσταση θα είναι : $P_{n-1}(t)\lambda\Delta t$.

Τέλος, για την τρίτη περίπτωση ενώ βρισκόμαστε στην κατάσταση n+1 δηλαδή βρισκόμαστε σε n+1 εισερχόμενες κλήσεις που εξυπηρετούνται και έχουμε μόνο μία εξερχόμενη κλήση τότε η πιθανότητα να πάμε στην n κατάσταση θα είναι : $P_{n+1}(t)(n+1)\mu\Delta t$.

Με βάση τα παραπάνω η πιθανότητα να βρισκόμαστε τη χρονική στιγμή $t+\Delta t$ στην κατάσταση n είναι να συμβαίνει ή η πρώτη ή η δεύτερη ή η τρίτη περίπτωση, τα ενδεχόμενα

αυτά είναι αυμβίβαστα με μηδενική τομή και έτσι προκύπτει το παρακάτω άθροισμα. Δηλαδή, έχουμε [26][24]:

$$P_n(t+\Delta t) = P_n(t)(1-\lambda\Delta t - n\mu\Delta t) + P_{n-1}(t)\lambda\Delta t + P_{n+1}(t)(n+1)\mu\Delta t \qquad (2.1)$$

Από τη σχέση (2.1) έχουμε:

$$P_n(t+\Delta t) = [\lambda P_{n-1}(t) + (n+1)\mu P_{n+1}(t) - (\lambda + n\mu)P_n(t)]\Delta t + P_n(t)$$

$$P_n(t+\Delta t) - P_n(t) = [\lambda P_{n-1}(t) + (n+1)\mu P_{n+1}(t) - (\lambda + n\mu)P_n(t)]\Delta t$$

$$\frac{P_n(t+\Delta t) - P_n\,(t)}{\Delta t} = \lambda P_{n-1}(t) + (n+1)\mu P_{n+1}(t) - (\lambda + n\mu)P_n(t) \qquad (2.2)$$

Επειδή, $\Delta t \to 0$ από τη σχέση (2.2) έχουμε:

$$\lim_{\Delta t\to 0} \frac{P_n(t+\Delta t) - P_n\,(t)}{\Delta t} \equiv \frac{d}{dt}P_n(t) = \lambda P_{n-1}(t) + (n+1)\mu P_{n+1}(t) - (\lambda + n\mu)P_n(t) \qquad (2.3)$$

Όταν το σύστημα δουλεύει κανονικά σε ένα πάρα πολύ μεγάλο χρονικό διάστημα τότε θα μεταβαίνει και σε μία σταθερή κατάσταση (ή στατιστική ισορροπία). Εμείς καλούμαστε να υπολογίσουμε αυτή την πιθανότητα, που ονομάζεται και πιθανότητα μόνιμης κατάστασης. Επομένως, πιθανότητα μόνιμης κατάστασης είναι σταθερή και ανεξάρτητη της μεγάλης τιμής του χρόνου. Δηλαδή, αν $t \to \infty$ τότε $\frac{d}{dt}P_n(t) \to 0$, οπότε από τη σχέση (2.3) προκύπτει [24]:

$$0 = \lambda P_{n-1} + (n+1)\mu P_{n+1} - (\lambda + n\mu)P_n$$

$$(\lambda + n\mu)P_n = \lambda P_{n-1} + (n+1)\mu P_{n+1}, \text{όπου } n = 0,1,2 \dots C \text{ και } P_{-1} = P_C = 0 \qquad (2.4)$$

Από τη σχέση (2.4) έχουμε:

Για $n = 0$: $\lambda P_0 = \mu P_1$

Για $n = 1$: $\lambda P_1 + \mu P_1 = \lambda P_0 + 2\mu P_2$

Για $n = 2$: $\lambda P_2 + 2\mu P_2 = \lambda P_1 + 3\mu P_3$

...

Για $n = n-2$: $\lambda P_{n-2} + (n-2)\mu P_{n-2} = \lambda P_{n-3} + (n-1)\mu P_{n-1}$

Για $n = n - 1$: $\lambda P_{n-1} + (n-1)\mu P_{n-1} = \lambda P_{n-2} + n\mu P_n$

Από τις παραπάνω εξισώσεις καταλήγουμε στον αναδρομικό τύπο που είναι ο εξής [24][26]:

$$P_n = \frac{\alpha}{n} P_{n-1}\,, \qquad για\ n = 1, 2, \dots, C \tag{2.5}$$

$$όπου\ a = \frac{\lambda}{\mu}\ το\ φορτίο\ κίνησης\ σε\ erl. \tag{2.6}$$

Από τη σχέση (2.5):

$$P_n = \frac{\alpha}{n} P_{n-1} = \frac{\alpha}{n}\frac{\alpha}{n-1} P_{n-2} = \frac{\alpha}{n}\frac{\alpha}{n-1}\frac{a}{n-2} P_{n-3} = \cdots = \frac{a^n}{n!} P_0$$

Δηλαδή, έχουμε [24], [26]:

$$P_n = \frac{a^n}{n!} P_0\ , για\ n = 1, 2, \dots, C \quad και\ P_0\ η\ πιθανότητα\ το\ σύστημα\ να\ είναι\ άδειο. \tag{2.7}$$

Γνωρίζουμε ότι το σύστημα θα είναι πάντα σε μία κατάσταση n=1, 2, …C οπότε θα ισχύει:

$$\sum_{n=0}^{C} P_n = 1 \tag{2.8}$$

και από τη σχέση (2.7) και (2.8) προκύπτει:

$$\sum_{n=0}^{C} \frac{a^n}{n!} P_0 = 1 \Rightarrow P_0 \sum_{n=0}^{C} \frac{a^n}{n!} = 1 \Rightarrow\ P_0 = \frac{1}{\sum_{n=0}^{C} \frac{a^n}{n!}} \tag{2.9}$$

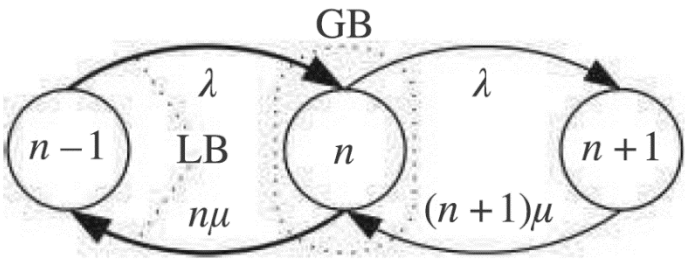
καλύτερα [24][26]:

$$P_0 = \left(\sum_{n=0}^{C} \frac{a^n}{n!}\right)^{-1}\ για\ n = 0, 1, 2, \dots C \tag{2.10}$$

Ο τύπος (2.10) είναι γνωστός ως τύπος *Erlang B* ή κατανομή *Erlang B* ή φόρμουλα *Erlang B* όταν το σύστημα δεν έχει κατειλημμένα b.u. [2].

2.3 Σφαιρική Ισορροπία και Τοπική Ισορροπία

Αξίζει, σ' αυτό το σημείο να αναφέρουμε τις έννοιες της Σφαιρικής Ισορροπίας (Global Balance, *GB*) και της Τοπικής Ισορροπίας (Local Balance, *LB*), όπως παρουσιάζονται και στο Σχήμα 6 [2].



Σχήμα 6: Σφαιρική και Τοπική Ισορροπία

Από τη σχέση (2.4) αν πολλαπλασιάσουμε όλους τους όρους με το *Δt* θα προκύψει ο παρακάτω τύπος [24]:

$$(\lambda\Delta t + n\mu\Delta t)P_n = \lambda\Delta t P_{n-1} + (n+1)\mu\Delta t P_{n+1} \qquad (2.11)$$

Από τον τύπο (2.11) το αριστερό μέλος $(\lambda\Delta t + n\mu\Delta t)P_n$ μας δείχνει την πιθανότητα του αθροίσματος δύο γεγονότων, όταν το σύστημα βρίσκεται στην κατάσταση *n*. Μία κλήση φτάνει στο σύστημα $(\lambda\Delta t)$ και μία άλλη φεύγει από το σύστημα $(n\mu\Delta t)$. Με αυτόν τον τρόπο το σύστημα φεύγει από την κατάσταση *n*.

Από τον τύπο (2.11) το δεξιό μέλος $\lambda\Delta t P_{n-1} + (n+1)\mu\Delta t P_{n+1}$ μας δείχνει την πιθανότητα να πάμε στην κατάσταση n από τις γειτονικές καταστάσεις $n-1$ κατά την άφιξη μιας κλήσης ή $n+1$ κατά την αποχώρηση μιας κλήσης. Αν από τη σχέση βγάλουμε το *Δt* τότε έχουμε την λεγόμενη εξίσωση σφαιρικής ισορροπίας στην κατάσταση *n* στην οποία ισχύει [24]:

Ρυθμός Εξόδου = Ρυθμός Εισόδου

Από τον τύπο (2.5) προκύπτει:

$$nP_n = \alpha P_{n-1} => nP_n = \frac{\lambda}{\mu} P_{n-1} => n\mu P_n = \lambda P_{n-1}$$

Και αν πολλαπλασιάσουμε όλους τους όρους με το Δt θα προκύψει ο παρακάτω τύπος:

$$n\mu \Delta t P_n = \lambda \Delta t P_{n-1} \quad (2.12)$$

Το αριστερό μέλος του τύπου (2.12), $n\mu \Delta t P_n$, εκφράζει την πιθανότητα μετάβασης προς τα κάτω, δηλαδή από την κατάσταση *n* στην κατάσταση $n-1$, λόγω αποχώρησης μιας κλήσης. Αντίστοιχα, το δεξί μέλος, $\lambda \Delta t P_{n-1}$, εκφράζει την πιθανότητα μετάβασης προς τα επάνω, δηλαδή από την κατάσταση $n-1$ στην κατάσταση *n*, λόγω άφιξης μιας κλήσης.

Αν στη σχέση (2.12) απαλειφθεί το *Δt*, προκύπτουν οι αντίστοιχοι ρυθμοί μετάβασης και οδηγούμαστε στη συνθήκη τοπικής ισορροπίας μεταξύ των γειτονικών καταστάσεων *n* και $n-1$, όπου ισχύει [24]:

Ρυθμός προς τα Κάτω = Ρυθμός προς τα Επάνω

Τέλος, η ύπαρξη GB γειτονικών καταστάσεων δεν εγγυάται την ύπαρξη *LB*. Η ύπαρξη *LB* εγγυάται πάντα την ύπαρξη *GB* [24] .

2.4 Η πιθανότητα *CBP* και ο μέσος όρος *U*

Η πιθανότητα απώλειας κλήσης (CBP) αποτελεί μια βασική μετρική αξιολόγησης της ποιότητας ενός τηλεπικοινωνιακού συστήματος. Συγκεκριμένα, εκφράζει την πιθανότητα μια εισερχόμενη κλήση να απορριφθεί, όταν το σύστημα αδυνατεί να την εξυπηρετήσει λόγω πλήρους απασχόλησης των διαθέσιμων πόρων (π.χ. καναλιών). Επομένως, η CBP παρέχει ένα μέτρο της απόδοσης του συστήματος, δείχνοντας πόσο συχνά αναμένεται να συμβεί απόρριψη κλήσεων υπό δεδομένες συνθήκες λειτουργίας και συγκεκριμένα χαρακτηριστικά του συστήματος.

Από τον τύπο (2.7) έχουμε :

$$P_n = \frac{a^n}{n!}P_0\ , \qquad \textit{για } n = 1, 2, \dots, C \ \textit{ και } P_0 \textit{ η πιθανότητα το σύστημα να είναι άδειο.}$$

Αν θεωρήσουμε ότι το σύστημά μας είναι γεμάτο δηλαδή n = C, δηλαδή βρίσκεται στην κατάσταση C , τότε έχουμε:

$$P_C = \frac{a^C}{C!}P_0 \tag{2.13}$$

$\textit{και } P_0 \textit{ η πιθανότητα το σύστημα να είναι άδειο.}$

Με τη βοήθεια του τύπου (2.10) προκύπτει:

$$B \equiv P_C = \frac{\frac{\alpha^C}{C!}}{\sum_{n=0}^{C}\frac{a^n}{n!}} \quad \textit{για } n = 1, 2, \dots, C \ \textit{ και } \alpha = \frac{\lambda}{\mu} \tag{2.14}$$

Η σχέση (2.14) αποτελεί τη φόρμουλα *Erlang B* όταν το σύστημα βρίσκεται στην κατάσταση C , δηλαδή είναι γεμάτο και με βάση αυτή τη σχέση μπορούμε να υπολογίσουμε την πιθανότητα απώλειας κλήσης *(CBP) B* [24].

Η πιθανότητα απώλειας κλήσης *B* μπορεί να υπολογιστεί και με αναδρομικό τρόπο, μέσω του τύπου (2.14), [2]:

$$B \equiv P_C = \frac{\frac{\alpha^C}{C!}}{\sum_{n=0}^{C}\frac{a^n}{n!}} = \frac{\frac{\alpha}{C}\frac{\alpha^{C-1}}{(C-1)!}}{\sum_{n=0}^{C-1}\frac{a^n}{n!}+\frac{\alpha^C}{C!}} = \frac{\frac{\alpha}{C}\frac{\alpha^{C-1}}{(C-1)!}}{\sum_{n=0}^{C-1}\frac{a^n}{n!}+\frac{a}{C}\frac{\alpha^{C-1}}{(C-1)!}}$$

$$= \frac{\frac{\alpha}{C}\frac{\alpha^{C-1}}{(C-1)!}/\sum_{n=0}^{C-1}\frac{a^n}{n!}}{1+\frac{a}{C}\frac{\alpha^{C-1}}{(C-1)!}/\sum_{n=0}^{C-1}\frac{a^n}{n!}} = \frac{\frac{a}{C}P_{C-1}}{1+\frac{a}{C}P_{C-1}} = \frac{aP_{C-1}}{C+aP_{C-1}}\ , \qquad \mu\varepsilon\ C \geq 1.$$

$$B \equiv P_C = \frac{aP_{C-1}}{C+aP_{C-1}} \tag{2.15}$$

Μια ακόμη μετρική αρκετά χρήσιμη, είναι το ποσοστό χρήσης του συστήματος που υπολογίζει το μέσο όρο των κατειλημμένων *b.u.* και συμβολίζεται ως U *(Utilization*) [2]:

$$U = \sum_{n=1}^{C} nP_n = \sum_{n=1}^{C} n\frac{a^n}{n!}P_0 \tag{2.16}$$

Από τη σχέση (2.13) και τη σχέση (2.14) προκύπτει ότι [2]:

$$U = \sum_{n=1}^{C} nP_n = \sum_{n=1}^{C} n\frac{a^n}{n!}P_0 = aP_0\sum_{n=1}^{C}\frac{a^{n-1}}{(n-1)!} = aP_0\left[\sum_{n=0}^{C}\frac{a^n}{n!} - \frac{a^C}{C!}\right]$$

$$= a\left[\sum_{n=0}^{C}\frac{a^n}{n!}\right]^{-1}\left[\sum_{n=0}^{C}\frac{a^n}{n!} - \frac{a^C}{C!}\right] = a(1-B)$$

$$U = a(1-B) \tag{2.17}$$

Για τον υπολογισμό του *U* σε ποσοστιαία μορφή χρησιμοποιούμε τον τύπο:

$$U\% = \frac{U}{C} = \frac{a}{C}(1-B) \tag{2.18}$$

που επιβεβαιώνει τη φράση ότι:

«Το φορτίο κίνησης α ισούται με την αναμενόμενη μέση τιμή U των κατειλημμένων ζεύξεων C» και δείχνει το βαθμό της απόδοσης της ζεύξης [2].

2.5 Επεξήγηση των προγραμμάτων του μοντέλου *Erlang B*

2.5.1 Επεξήγηση του προγράμματος με βάση την εκτέλεση των τύπων

Υπενθυμίζεται ότι το μοντέλο *Erlang B* χρησιμοποιείται για τον υπολογισμό της πιθανότητας απώλειας κλήσεων *B* σε ένα σύστημα με πεπερασμένη χωρητικότητα *C*, δηλαδή την πιθανότητα μια εισερχόμενη κλήση να απορριφθεί λόγω έλλειψης διαθέσιμων πόρων (εξυπηρετητών ή καναλιών). Η πληροφορία αυτή είναι ιδιαίτερα σημαντική για τον σχεδιασμό και τη διάσταση του συστήματος, καθώς επιτρέπει την αξιολόγηση της απόδοσής του και την εκτίμηση της ανάγκης αύξησης της χωρητικότητας, σε περιπτώσεις όπου η πιθανότητα απώλειας κλήσεων είναι υψηλή.

Στα πλαίσια της παρούσας εργασίας όλες οι εφαρμογές υλοποίηθηκαν σε γλώσσα προγραμματισμού *Python*. Έτσι και εδώ αναπτύχθηκε εφαρμογή με γραφικό περιβάλλον, η οποία υλοποιεί το μοντέλο *Erlang B*. Μέσω αυτής, ο χρήστης μπορεί να εισάγει τη

χωρητικότητα του συστήματος C, καθώς και τους μέσους ρυθμούς άφιξης και εξυπηρέτησης κλήσεων, λ και μ αντίστοιχα. Στη συνέχεια, παρέχεται η δυνατότητα επιλογής της επιθυμητής μεθόδου υπολογισμού και εκτέλεσής της μέσω του κουμπιού «Υπολογισμός».

Με βάση τα δεδομένα εισόδου, υπολογίζεται επίσης το προσφερόμενο φορτίο κίνησης α, το οποίο αποτελεί βασική παράμετρο για την αξιολόγηση της απόδοσης του συστήματος και για τον προσδιορισμό της πιθανότητας απώλειας κλήσεων.

Αξίζει να σημειωθεί ότι, στο συγκεκριμένο στάδιο της διαδικασίας της εφαρμογής, δεν λαμβάνεται υπόψη οποιαδήποτε τιμή ενδέχεται να εισαχθεί στο πεδίο της πιθανότητας απώλειας κλήσης B. Η παράμετρος αυτή καθορίζεται από τον χρήστη και χρησιμοποιείται αποκλειστικά για την παραγωγή των αντίστοιχων διαγραμμάτων, όπως θα παρουσιαστεί στη συνέχεια, και όχι για τον άμεσο υπολογισμό των αποτελεσμάτων.

Οι μέθοδοι υπολογισμού μπορούν να ταξινομηθούν σε δύο βασικές κατηγορίες, ανάλογα με τον τρόπο επίλυσης:

- **Αναδρομική μέθοδος**, η οποία βασίζεται σε αναδρομικό τύπο.
- **Μέθοδος παραγοντικών**, η οποία στηρίζεται στον κλασικό τύπο που περιλαμβάνει παραγοντικά.

Η αναδρομική μέθοδος, πέρα από τη θεωρητική της διατύπωση μέσω αναδρομής, μπορεί να υλοποιηθεί και υπολογιστικά με επαναληπτική διαδικασία, για λόγους αποδοτικότητας και αριθμητικής σταθερότητας, όπως θα αναλυθεί στη συνέχεια.

Τα συμπεράσματα για τους τρόπους υπολογισμού συνοψίζονται ως εξής:

• **Αναδρομική διαδικασία**: Η διαδικασία αυτή παρέχει γρήγορα αποτελέσματα, ωστόσο περιορίζεται από τους διαθέσιμους πόρους μνήμης του συστήματος. Με βάση πειραματικές δοκιμές, ως άνω φράγματα επιλέγονται οι τιμές 1000 τόσο για τη χωρητικότητα C όσο και για το φορτίο κίνησης ($\alpha = \lambda/\mu$). Σε περίπτωση που ο χρήστης εισάγει τιμές μεγαλύτερες από τα όρια αυτά, το πρόγραμμα εμφανίζει σχετικό μήνυμα περιορισμού.

• **Επαναληπτική διαδικασία**: Η προσέγγιση αυτή βασίζεται σε επανάληψη αντί της αναδρομής και επιτυγχάνει αντίστοιχα γρήγορους υπολογισμούς, χωρίς να δημιουργούνται προβλήματα υπερχείλισης μνήμης (overflow). Η επιλογή χρήσης τόσο της αναδρομικής όσο και της επαναληπτικής διαδικασίας αιτιολογείται από το γεγονός ότι η αναδρομή υπόκειται σε περιορισμούς της *Python* ως προς το μέγιστο βάθος κλήσεων. Συνεπώς, για μεγάλες τιμές εισόδου, η επαναληπτική διαδικασία αποτελεί πρακτική εναλλακτική, διατηρώντας παράλληλα την εννοιολογική σαφήνεια των δύο προσεγγίσεων. Και οι δύο διαδικασίες οδηγούν ουσιαστικά στα ίδια αποτελέσματα, με παρόμοιους χρόνους εκτέλεσης. Οι αντίστοιχες υλοποιήσεις σε *Python* παρουσιάζονται στο Παράρτημα Β.

• **Αρχικός τύπος (με παραγοντικά)**: Η εφαρμογή του αρχικού τύπου του *Erlang B*, ο οποίος περιλαμβάνει παραγοντικά, παρουσιάζει σημαντικούς περιορισμούς ως προς την κατανάλωση υπολογιστικών πόρων. Ειδικότερα, για μεγάλες τιμές των παραμέτρων, οι υπολογισμοί γίνονται ιδιαίτερα απαιτητικοί, γεγονός που καθιστά τη μέθοδο αυτή λιγότερο πρακτική σε σχέση με τις αναδρομικές και επαναληπτικές προσεγγίσεις. Ωστόσο, υιοθετήθηκε μια εναλλακτική προσέγγιση, η οποία επιτυγχάνει ικανοποιητική ακρίβεια με αποδεκτούς χρόνους εκτέλεσης, και η οποία αναλύεται στο Παράρτημα Β.

Συνεχίζοντας, μετά την επιλογή του τρόπου υπολογισμού και αφού πατήσουμε το κουμπί «Υπολογισμός», το πρόγραμμα δίνει τα εξής αποτελέσματα:

- Το αποτέλεσμα του τύπου, δηλαδή τη πιθανότητα απώλειας κλήσης *B,*
- το χρόνο διεκπεραίωσης του υπολογισμού σε *msec,*
- το *U* ως τιμή αλλά και επί τοις εκατό U%,
- το πλήθος των πράξεων *Ops,*

  (ο τρόπος υπολογισμού των πράξεων αναλύεται στο Παράρτημα Β).

Ταυτόχρονα, μπορεί να αποθηκεύονται αυτόματα όλες οι δοκιμές σε ένα αρχείο *Excel* για την μετέπειτα επεξεργασία. Το αρχείο ονομάζεται *erlang_results.xlsx* και δημιουργείται στον ίδιο φάκελο που βρίσκεται και το πρόγραμμα που τρέχουμε.

Ο χρήστης θα πρέπει να είναι ιδιαίτερα προσεκτικός, καθώς σε περίπτωση που κλείσει το πρόγραμμα και το επανεκκινήσει, τα προηγούμενα δεδομένα δεν διατηρούνται. Για τον

λόγο αυτό, συνιστάται πριν τον τερματισμό της εφαρμογής να αποθηκεύει το αντίστοιχο αρχείο Excel σε προσωπικό φάκελο, ώστε να διασφαλίζεται η διατήρηση των αποτελεσμάτων.

Διαγράμματα

Η εφαρμογή, πέρα από το υπολογιστικό σκέλος, παρέχει στον χρήστη και τη δυνατότητα παραγωγής διαγραμμάτων. Συγκεκριμένα, διατίθενται τρεις (3) επιλογές:

- Με σταθερό α: Παράγεται το 1ο διάγραμμα του υπολογιζόμενου B σε συνάρτηση με το C του χρήστη, διατηρώντας σταθερό το α του χρήστη, για όλες τις τιμές μέχρι C. Ομοίως, παράγεται και το 2ο διάγραμμα για το U. Αν ο χρήστης έχει δώσει κάποια τιμή στο B, αυτή δεν λαμβάνεται υπόψη.
- Με σταθερό C : Παράγεται το 1ο διάγραμμα του υπολογιζόμενου B σε συνάρτηση με το α του χρήστη, διατηρώντας σταθερό το C του χρήστη, για όλες τις τιμές μέχρι α. Ομοίως, παράγεται και το 2ο διάγραμμα για το U. Και σε αυτή την περίπτωση, τυχόν τιμή στο B δεν λαμβάνεται υπόψη.
- Με σταθερό B: Εδώ λαμβάνεται υπόψη το $B_{χρήστη}$ που έχει εισάγει ο χρήστης. Η εφαρμογή αναζητά, για τιμές των α και C από το μηδέν έως τα όρια που έχει ορίσει ο χρήστης, εκείνους τους συνδυασμούς που ικανοποιούν προσεγγιστικά την ισότητα του υπολογιζόμενου B με το επιθυμητό $B_{χρήστη}$, με επιτρεπτή απόκλιση της τάξης του 0.001.

Τα διαγράμματα εμφανίζονται με το κουμπί «Διαγράμματα.»

Για τις παραπάνω οδηγίες που καθοδηγούν τον χρήστη, δημιουργήθηκε ένα ξεχωριστό κουμπί με το όνομα «Διαγράμματα : Οδηγίες»

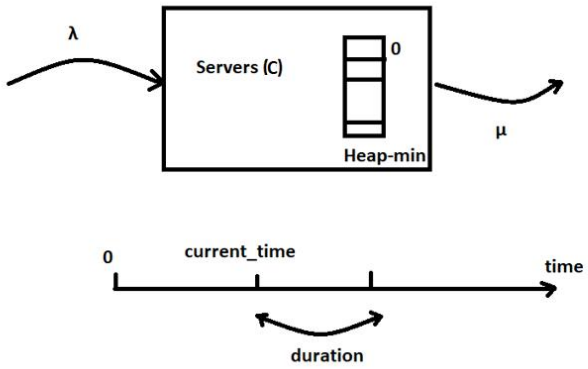
Τέλος, η εφαρμογή παρέχει στον χρήστη τη δυνατότητα διαχείρισης των δεδομένων μέσω δύο κουμπιών:

- «Καθαρισμός Δεδομένων στα Excel αρχεία»: Αφαιρεί όλα τα αποθηκευμένα δεδομένα από τα αρχεία Excel, ώστε να είναι δυνατή η παραγωγή νέων διαγραμμάτων με νέα στοιχεία.

- «Καθαρισμός Όλων»: Καθαρίζει όλα τα πλαίσια εισαγωγής στοιχείων στο γραφικό περιβάλλον, επιτρέποντας την εισαγωγή νέων δεδομένων για υπολογισμούς και διαγράμματα.

2.5.2 Επεξήγηση του προγράμματος της προσομοίωσης

Έστω ότι διαθέτουμε ένα σύστημα, όπως απεικονίζεται στο Σχήμα 7, με συνολική χωρητικότητα C π.χ. C εξυπηρετητές, οι οποίοι αρχικά είναι όλοι μη κατειλημμένοι. Στο σύστημα καταφθάνουν κλήσεις (αφίξεις *Poisson*) με μέση τιμή ρυθμού άφιξης λ. Έστω ότι οι κλήσεις αυτές εξυπηρετούνται από τους εξυπηρετητές με μέσο ρυθμό εξυπηρέτησης μ (με εκθετική κατανομή).



Σχήμα 7: Η λογική της προσομοίωσης στον *Erlang B*

Πριν προχωρήσουμε στην ανάλυση της λογικής της προσομοίωσης, είναι σημαντικό να κατανοήσουμε τις παρακάτω έννοιες:

Εκθετική κατανομή της γλώσσας προγραμματισμού *Python Expovariate*

Από την ενσωματωμένη βιβλιοθήκη της *Python, random* χρησιμοποιούμε τη συνάρτηση *expovariate*. Η *expovariate* είναι μία συνάρτηση που δημιουργεί τυχαίες τιμές, που ακολουθούν την εκθετική κατανομή. Η τιμή εισόδου είναι τιμή κάποιου ρυθμού π.χ. λ, μ και η τιμή εξόδου είναι η χρονική διάρκεια (οι μονάδες μέτρησης καθορίζονται από τον ρυθμό εισόδου).

Για παράδειγμα, από τη συνάρτηση *expovariate* βρίσκουμε τους χρόνο άφιξης και τη διάρκεια εξυπηρέτηση της κλήσης *(arrival_time και duration) ως εξής*:

arrival_time = expovariate(λ) και *duration = expovariate(μ)*

Σπόρος (*Seed*)

Στο πρόγραμμά γίνεται χρήση του *seed*. Ο σπόρος (*seed*) αρχικοποιεί τον αλγόριθμο που παράγει τυχαίους αριθμούς, δηλαδή ο μαθηματικός μηχανισμός πίσω από αυτή τη γεννήτρια τυχαίων αριθμών ξεκινά από την τιμή του *seed*. Αυτό προσφέρει το πλεονέκτημα να παίρνουμε τους ίδιους τυχαίους αριθμούς και να παράγουμε τα ίδια αποτελέσματα, όσες φορές και να τρέξουμε τη γεννήτρια. Μ' αυτό τον τρόπο ελέγχουμε την προσομοίωση, η οποία κάνει εκτελέσεις (π.χ. συγκρίσεις κ.α.) με τα ίδια δεδομένα εισόδου κάθε φορά.

Αν δεν χρησιμοποιηθεί το s*eed*, δημιουργείται διαφορετική ακολουθία τυχαίων αριθμών σε κάθε εκτέλεση από τη γεννήτρια, οδηγώντας σε διαφορετικά αποτελέσματα.

Στο Παράρτημα Β εξηγείται ο μηχανισμός των *seeds*.

Σωρός Ελαχίστων *(min Heap)*

Στο πρόγραμμά μας χρησιμοποιούμε, επίσης, την ενσωματωμένη βιβλιοθήκη *heapq* της *Python* που ακολουθεί τη λογική του σωρού, μέσω λίστας. Στη λίστα τοποθετείται στη θέση 0 το μικρότερο στοιχείο, λογική των ελαχίστων (--*min Heap*--). Στο σωρό μπορούμε να εισάγουμε στοιχεία και να εξάγουμε πάντα το στοιχείο που βρίσκεται στη θέση μηδέν της λίστας. Στη θέση αυτή θα βρίσκεται πάντα η μικρότερη τιμή του σωρού.

heapq.heappop(λίστα): Εξαγωγή του στοιχείου της θέση μηδέν της λίστας (pop)

heapq.heappush(λίστα, τιμή): Εισαγωγή στοιχείου στη λίστα (push)

heapq.heapify(λίστα) : Δημιουργία σωρού

Μεταβατική Περίοδος (Transient period, transit)

Το *transit* αντιπροσωπεύει τον αριθμό των αρχικών κλήσεων που δεν λαμβάνονται υπόψη κατά τον υπολογισμό των αποτελεσμάτων. Δίνεται η δυνατότητα στον χρήστη να καθορίσει την τιμή αυτής της παραμέτρου. Η τιμή του *transit* δεν πρέπει να είναι αρνητική ούτε να υπερβαίνει το συνολικό πλήθος κλήσεων της προσομοίωσης. Η χρήση του *transit* αποσκοπεί στο να δοθεί στο σύστημα ο απαραίτητος χρόνος ώστε να φτάσει σε κατάσταση σταθερής λειτουργίας. Η αρχική αυτή περίοδος, κατά την οποία δεν συλλέγονται δεδομένα, αντιστοιχεί συνήθως στο 1% έως 5% των αρχικών κλήσεων και ονομάζεται μεταβατική περίοδος (*transient period*)**.**

Η κεντρική σκέψη του αλγορίθμου της προσομοίωσης βρίσκεται στη συνάρτηση με το όνομα ***simulation_of_erlang_b***. Η συνάρτηση δέχεται ως ορίσματα τις συνολικές κλήσεις (calls) στο σύστημα, το *C* π.χ. πόσοι εξυπηρετητές υπάρχουν στο σύστημα, τα *λ και μ*, δύο *seeds* για τις εκθετικές κατανομές των *λ και μ* καθώς και το *transit.*

Με βάση τα παραπάνω στοιχεία η υλοποίηση της προσομοίωσης είναι η ακόλουθη:

Όταν μια κλήση φτάνει στο σύστημα, η χρονική στιγμή άφιξής της υπολογίζεται με ακρίβεια χρησιμοποιώντας την εκθετική κατανομή του μέσου ρυθμού άφιξης λ.

Συγκεκριμένα:

- Αρχικοποιούμε τη μεταβλητή *current_time* με μηδέν, δηλαδή πριν φτάσει η πρώτη κλήση: *current_time = 0,*
- Η χρονική απόσταση μέχρι την επόμενη κλήση προκύπτει από την εκθετική κατανομή: *arrival_time = expovariate(λ)*
- Η νέα χρονική στιγμή άφιξης μιας κλήσης είναι το άθροισμα της προηγούμενης *current_time* και της *arrival_time***:** *current_time = current_time + arrival_time*

Με αυτόν τον τρόπο, γνωρίζουμε ακριβώς πότε φτάνει κάθε κλήση στο σύστημα, και η *current_time* ενημερώνεται δυναμικά για να αντιπροσωπεύει τη συνολική χρονική πρόοδο του συστήματος. Αυτή η διαδικασία επαναλαμβάνεται για κάθε νέα κλήση, διατηρώντας το χρονολογικό πλαίσιο της προσομοίωσης. (Σχήμα 7)

Αμέσως μετά την άφιξη κάθε νέας κλήσης, ο αλγόριθμος ελέγχει τον σωρό που κρατάει τις τρέχουσες κλήσεις σε εξέλιξη (με το χρόνο λήξης τους). Η λογική έχει ως εξής:

Γίνεται έλεγχος του πρώτου στοιχείου του σωρού. Το πρώτο στοιχείο του min heap (heap[0]) είναι πάντα η κλήση που τελειώνει νωρίτερα, λόγω ιδιότητας του min heap. Έτσι, τρέχει μία επανάληψη που εξετάζει αν ο σωρός είναι άδειος ή έχει τιμές μικρότερες του χρόνου που υπολογίσαμε για την κλήση που μόλις κατέφθασε.

heap[0] <= current_time

Αν συμβαίνει αυτό σημαίνει ότι υπάρχουν κλήσεις που έχουν τελειώσει την εργασία τους και πρέπει να αποδεσμεύσουν τον εξυπηρετητή που χρησιμοποιούσαν. Ουσιαστικά, διαγράφονται από τον σωρό και ταυτόχρονα αυξάνεται κατά μία μονάδα η μεταβλητή που κρατά τους μη κατειλημμένους εξυπηρετητές. Αν δεν συντρέχουν οι προϋποθέσεις που είπαμε η μεταβλητή αυτή δεν επηρεάζεται.

Ξέροντας, πλέον, το πλήθος των εξυπηρετητών που είναι ελεύθεροι εξετάζουμε αν υπάρχει κάποιος προς διάθεση στη νέα κλήση. Αν υπάρχει, δίνεται στην κλήση, μειώνοντας τη μεταβλητή των ελεύθερων εξυπηρετητών κατά ένα και μετά υπολογίζεται η διάρκεια (*duration*) εξυπηρέτησης με την εκθετική κατανομή του μ .

duration = expovariate(μ)

Η τιμή που προκύπτει από τη διάρκεια (*duration*) συν τον χρόνο που υπολογίστηκε κατά την άφιξη της κλήσης (*current_time*), τοποθετείται στο σωρό.

push Heap (current_time + duration)

Αν δεν βρεθεί ελεύθερος εξυπηρετητής η κλήση απορρίπτεται , δηλαδή αυξάνεται ένας μετρητής απορριφθέντων κλήσεων.

Παρατίθενται ορισμένα επιπλέον στοιχεία που αφορούν τη λειτουργία της εφαρμογής:

Πιθανότητα απόρριψης κλήσης (Blocking Probability – B): Η πιθανότητα απόρριψης κλήσεων αποτελεί το βασικό μέγεθος ενδιαφέροντος της προσομοίωσης. Υπολογίζεται ως το πηλίκο του αριθμού των απορριφθεισών κλήσεων προς το συνολικό πλήθος των ενεργών

κλήσεων, δηλαδή των κλήσεων που λαμβάνονται υπόψη μετά τη μεταβατική περίοδο (total_calls−transit).

Βαθμός αξιοποίησης (Utilization - U): Το πρόγραμμα υπολογίζει επίσης τον βαθμό αξιοποίησης των εξυπηρετητών. Για τον σκοπό αυτό χρησιμοποιείται ένας μετρητής που καταγράφει κάθε φορά τον αριθμό των εξυπηρετητών που είναι δεσμευμένοι τη στιγμή άφιξης μιας κλήσης στο σύστημα. Η τιμή του μετρητή διαιρείται με το πλήθος των ενεργών κλήσεων (total_calls−transit), ώστε να προκύψει ο βαθμός αξιοποίησης του συστήματος.

Χρόνος εκτέλεσης: Επιπλέον, η εφαρμογή εμφανίζει τον χρόνο εκτέλεσης του αλγορίθμου, επιτρέποντας την αξιολόγηση της υπολογιστικής του απόδοσης.

Γραφικό περιβάλλον χρήστη: Η εφαρμογή λειτουργεί μέσω γραφικού περιβάλλοντος χρήστη, στο οποίο ο χρήστης εισάγει τις ακόλουθες παραμέτρους:

- αριθμό κλήσεων (*Calls*),
- χωρητικότητα συστήματος *C*,
- ρυθμούς άφιξης λ και εξυπηρέτησης μ,
- τιμές *seeds* για τις κατανομές λ και μ,
- τιμή *transit*.

Αρχικοποίηση παραμέτρων: Οι παράμετροι transit καθώς και οι τιμές των seeds αρχικοποιούνται με μηδέν.

Αποθήκευση αποτελεσμάτων: Όλα τα αποτελέσματα αποθηκεύονται σε αρχείο Excel. Επιπλέον, σε ορισμένες περιπτώσεις προσομοίωσης συγκεκριμένων μοντέλων, ο χρήστης μπορεί να επιλέξει την εμφάνιση ενδιάμεσων αποτελεσμάτων, όπως οι πιθανότητες κατάληψης των εξυπηρετητών. Τα αποτελέσματα αυτά εμφανίζονται σε ξεχωριστό παράθυρο.

Λειτουργίες καθαρισμού: Η εφαρμογή διαθέτει κουμπί καθαρισμού των πεδίων εισαγωγής, καθώς και κουμπί διαγραφής των αρχείων Excel που δημιουργούνται κατά την εκτέλεση των προσομοιώσεων.

Τέλος, ο χρήστης θα πρέπει να είναι ιδιαίτερα προσεκτικός κατά τη διαχείριση του αρχείου Excel. Εάν το αρχείο δεν αποθηκευτεί σε διαφορετική τοποθεσία, υπάρχει πιθανότητα απώλειας των μετρήσεων, καθώς τα δεδομένα διαγράφονται σε περίπτωση τερματισμού και επανεκκίνησης της εφαρμογής.

2.6 Αποτελέσματα προγραμμάτων

2.6.1 Παράδειγμα με βάση την εκτέλεση των τύπων

Ας υποθέσουμε ότι έχουμε ένα σύστημα χωρητικότητας *C = 2 b.u.*, π.χ. 2 εξυπηρετητές *(servers)*, μέσο ρυθμό άφιξης κλήσεων λ=3 και μέσο ρυθμό εξυπηρέτησης μ = 1**,** δηλαδή το φορτίο κίνησης είναι α = 3 *erl*.

Θα ψάξουμε να βρούμε την πιθανότητα απώλειας κλήσεων *Blocking Probability (B),* δηλαδή με ποια πιθανότητα αυτό το σύστημα θα απορρίπτει κλήσεις. Επίσης, ποιο είναι το ποσοστό χρήσης *U (Utilization)* των εξυπηρετητών και πόσες πράξεις χρειάζονται να εκτελεστούν (*Ops*) για να εξαχθούν τα αποτελέσματα.

Ο έλεγχος θα πραγματοποιηθεί με τον αρχικό τύπο του μοντέλου *Erlang B*, δηλαδή με τα παραγοντικά. Ο χρήστης έχει τη δυνατότητα να χρησιμοποιήσει για τον υπολογισμό του, τόσο την επαναληπτική όσο και την αναδρομική διαδικασία που στηρίζονται στον αναδρομικό τύπο.

$$B = P_3 = \frac{\frac{3^2}{2!}}{\sum_{i=0}^{2}\frac{3^i}{i!}} = \frac{\frac{9}{2}}{\frac{1}{1}+\frac{3}{1}+\frac{9}{2}} = \frac{4.5}{8.5} = 0.52941176$$

U= α(1-B) = 3(1-0.52941176) = 1.411764706 servers*

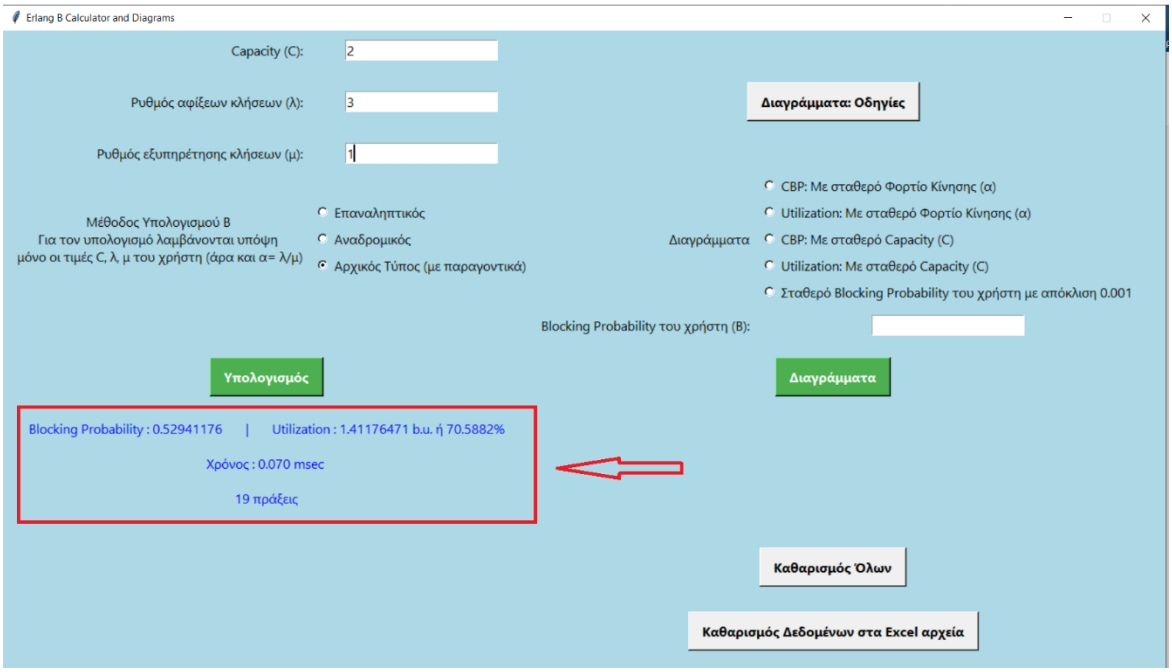
*U% =(U/C)*100 % = (1.411764706/2)*100%= 70.5882%*

Για τις πράξεις έχουμε περίπου στον αριθμητή : 1 διαίρεση και 2 πολλαπλασιασμούς και στον παρονομαστή : 2 προσθέσεις, 8 πολλαπλασιασμούς και 3 διαιρέσεις, αν προσθέσουμε και τη διαίρεση αριθμητή με παρονομαστή τότε οι πράξεις είναι περίπου 17.

Αξίζει να αναφέρουμε ότι στη συγκεκριμένη εφαρμογή έχουμε μειώσει την πολυπλοκότητα των παραγοντικών, όπως περιγράφεται στο Παράρτημα Β. Επομένως, γι' αυτές τις μικρές

τιμές στα *C, λ, μ* και *α,* το πλήθος των πράξεων του τύπου (2.14) είναι περίπου ίσο (με απόκλιση 2 πράξεων) μ' αυτόν της εφαρμογής που υλοποιήσαμε.

Σε μεγαλύτερες τιμές, όμως, των *C* και *α= (λ/μ)* τα αποτελέσματα της εφαρμογής, στο πλήθος των πράξεων, θα είναι μικρότερα από το πλήθος των πράξεων του αρχικού τύπου με τα παραγοντικά (2.14).



Εικόνα 1: Πρόγραμμα για το μοντέλο *Erlang B* με βάση τους τύπους

Για να επαληθεύσουμε την ιδιότητα του μοντέλου *Erlang B*, που λέει ότι [2]:

«Το φορτίο κίνησης α ισούται με την αναμενόμενη μέση τιμή U των κατειλημμένων ζεύξεων C»

Θα υπολογίσουμε τη αναμενόμενη μέση τιμή με βάση τον τύπο (2.16) (Εικόνα 1):

Για *C=2* b.u. *και α= λ/μ = 3/1 = 3*
*U = 0*p(0) + 1*p(1) + 2*p(2)*
όπου *p(1)= 3 / (1+3+4.5) =3/8.5* και *p(2) =4.5/8.5,*
άρα *U = 0 + 1*(3/8.5) +2*(4.5/8.5) = 12/8.5 =1.411764* servers

ή διαφορετικά από τον τύπο (2.17) :
U =α (1-B) = 3(1-0.52941176) = 1.411764* servers

και από τον τύπο (2.18):

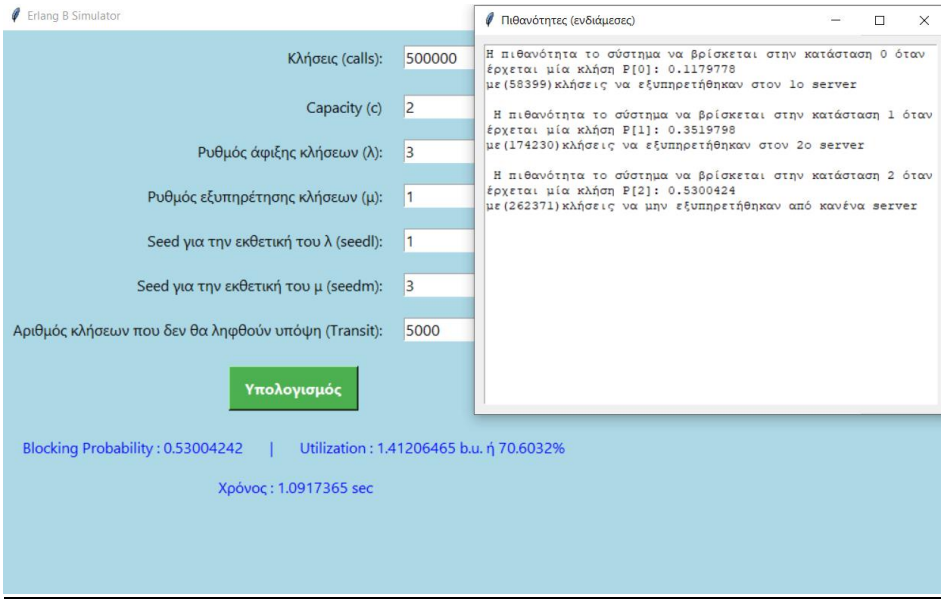
*U% =(U/C)*100 % = (1.411764706/2)*100%= 70.5882%*

Από την εφαρμογή πήραμε το *U* = 1.411764 *servers* και *U% = 70.5882%*

Τα αποτελέσματα είναι ακριβώς ίδια, όπως φαίνονται στην Εικόνα 2.

2.6.2 Παράδειγμα με βάση την προσομοίωση

Η προσομοίωση του Παραδείγματος της Ενότητας 2.6.1, Εικόνα 2, όπου *C*=2 *b.u.* *λ*=3 *και* *μ*=1. Η προσέγγιση με 500000 κλήσεις, *transit* 5000 κλήσεις και *seeds* για *λ* , *μ* τις τιμές 1 και 3 αντίστοιχα, αποδεικνύεται αξιόπιστη.



Εικόνα 2: Προσομοίωση για το μοντέλο *Erlang B*

Αποτελέσματα αναλυτικού μοντέλου *Erlang B* :

C: 9, *λ*: 2, *μ*: 1 δίνει *B*: 0.000019096 και *U*: 1.99961808 *servers*

Η προσομοίωση για το μοντέλο *Erlang B* :

Calls : 8000, *C*: 9, *λ*: 2, *μ*: 1 δίνει *B*: 0.00000000 και *U*: 1.99375 *servers*

Calls : 80000, *C*: 9, *λ*: 2, *μ*: 1 δίνει *B*: 0.00000000 και *U*: 1.9978125 *servers*

Calls : 800000, *C*: 9, *λ*: 2, *μ*: 1 δίνει *B*: 0.00000125 και *U*: 2.00083755 *servers*

Calls : 8000000, C: 9, *λ*: 2, *μ*: 1 δίνει *B*: 0.000161 και *U*: 1.99945075 *servers*

Calls : 80000000, *C*: 9, λ: 2, μ: 1 δίνει *B*: 0.00019122 και *U*: 1.99987319 *servers*

Όπως προκύπτει από την προσομοίωση, όταν το πλήθος των κλήσεων είναι σχετικά μικρό, από μερικές χιλιάδες έως μερικές δεκάδες χιλιάδες, δεν παρατηρείται καμία απόρριψη κλήσης, κυρίως λόγω της μεγάλης χωρητικότητας *C* του συστήματος. Σε αυτή την περίπτωση, η πιθανότητα μπλοκαρίσματος *B,* που υπολογίζει η προσομοίωση, προκύπτει 0 κάτι που είναι αναμενόμενο καθώς όλοι οι εξυπηρετητές είναι διαθέσιμοι για τις εισερχόμενες κλήσεις.

Καθώς αυξάνεται το πλήθος των κλήσεων σε επίπεδα εκατομμυρίων, αρχίζουν να καταγράφονται οι πρώτες απορρίψεις κλήσεων, και η πιθανότητα μπλοκαρίσματος *B* παίρνει σταδιακά θετικές τιμές. Έπρεπε να φτάσουμε τις 80 εκατομμύρια κλήσεις για να βγάλουμε περίπου τα ίδια αποτελέσματα με τον θεωρητικό τύπο του *Erlang B*.

Σημειώνεται ότι στον συγκεκριμένο πειραματισμό δεν εξετάστηκαν οι επιδράσεις των διαφορετικών seeds για τις κατανομές λ και μ, δηλαδή τις αυξομειώσεις των αποτελεσμάτων λόγω τυχαιότητας, αλλά η ανάλυση επικεντρώθηκε αποκλειστικά στο πλήθος των κλήσεων (Calls) και στο σημείο όπου η προσομοίωση αρχίζει να παρέχει αποτελέσματα συγκρίσιμα με το αναλυτικό μοντέλο.

Με άλλα λόγια, η σύγκλιση της προσομοίωσης προς την αναλυτική λύση απαιτεί πολύ μεγάλο αριθμό κλήσεων, ειδικά όταν η χωρητικότητα του συστήματος είναι υψηλή, γεγονός που αναδεικνύει τη σημασία του μεγέθους του δείγματος στον προσομοιωτικό έλεγχο. Ιδιαίτερα στην ανεύρεση πολύ μικρών τιμών *B* θέλει πολλά δείγματα για να έχουμε καλές προσεγγίσεις.

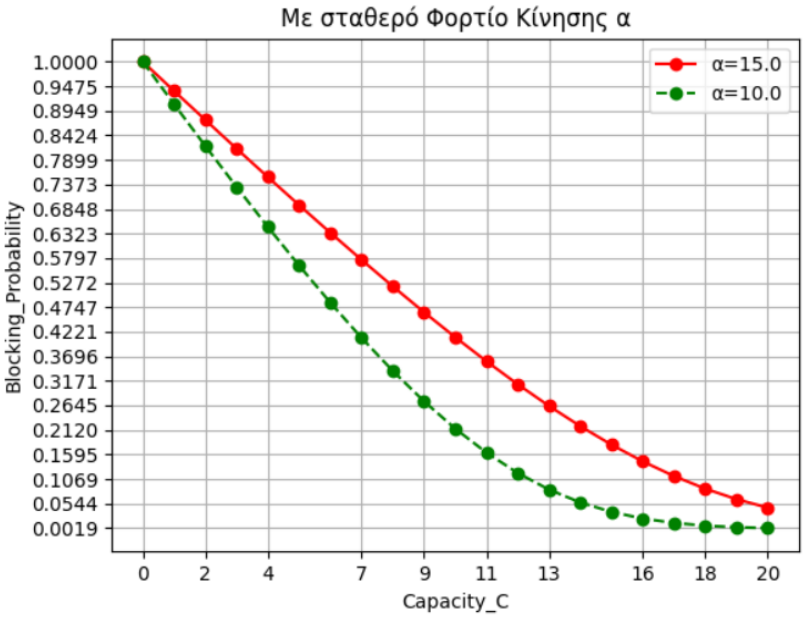
Τέλος, κάτι τέτοιο, είναι εύκολο να διαπιστωθεί όταν γνωρίζουμε τον μαθηματικό τύπο της φόρμουλας, αλλιώς θα πρέπει να πειραματιστούμε πάρα πολλές φορές και με πολλές κλήσεις και, ίσως, να καταλήξουμε σε κάποια συμπεράσματα. Όλα αυτά ενέχουν κάποια αβεβαιότητα και, ταυτόχρονα, κόστος σε πόρους και σε χρόνο.

2.6.3 Γραφικές Παραστάσεις

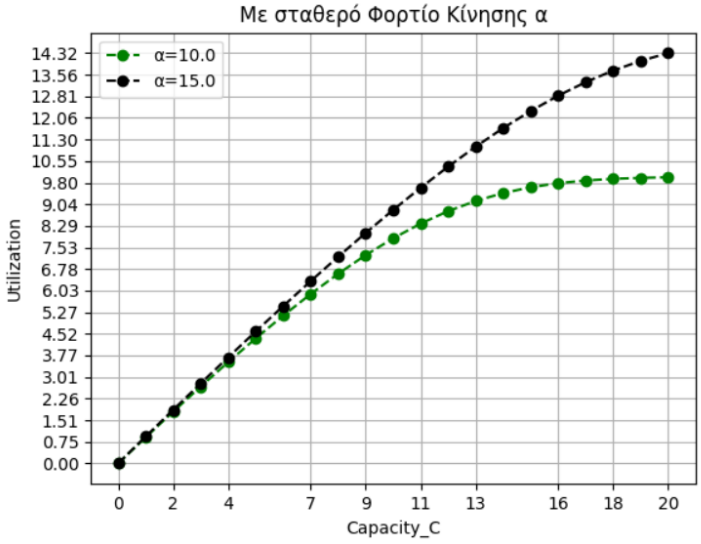
Στην εφαρμογή που υλοποιήσαμε με βάση τους τύπους, δίνεται η δυνατότητα στον χρήστη να εξάγει και διαγράμματα.

Παραδείγματα με διαγράμματα

- Σταθερό $\alpha = 15\ erl$ ή σταθερό $\alpha = 10\ erl$ και C από 0 μέχρι 20 $b.u$



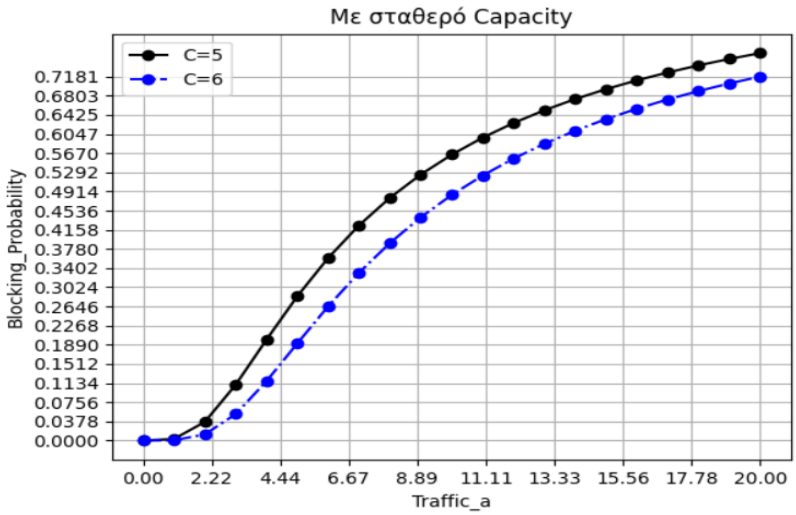
Εικόνα 3: Διάγραμμα *B* με σταθερό φορτίο κίνησης *α*



Εικόνα 4: Διάγραμμα *U* με σταθερό φορτίο κίνησης *α*

Από τα διαγράμματα των Εικόνων 3 και 4 παρατηρούμε ότι για σταθερό φορτίο κίνησης *α* όσο αυξάνεται η χωρητικότητα *C* τόσο μειώνεται η πιθανότητα απώλειας κλήσης *B*, και αυξάνεται το U, κάτι που είναι αναμενόμενο.

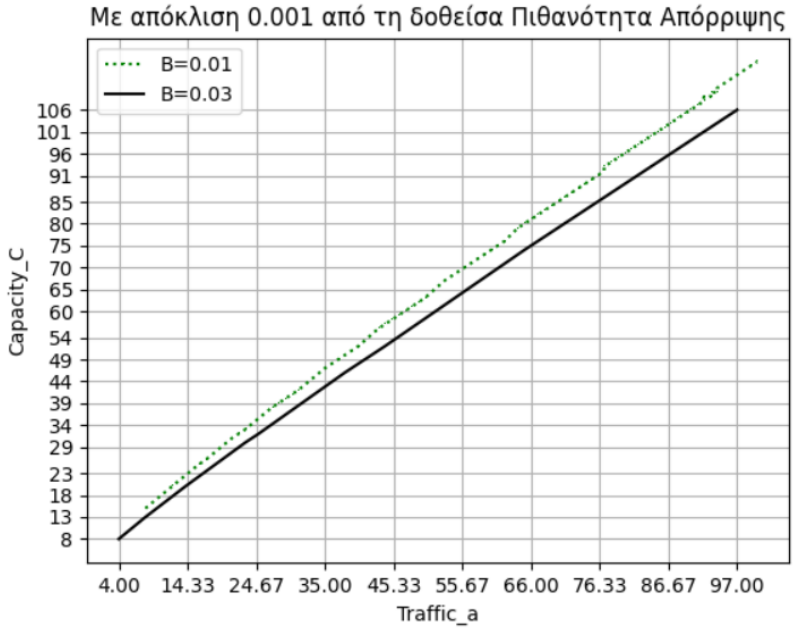
- Για σταθερό *C* = 5 *b.u.* ή *C* = 6 *b.u.* με *α από* 0 *έως* 20 *erl*



Εικόνα 5: Διάγραμμα με σταθερή χωρητικότητα *C*

Από το διάγραμμα της Εικόνας 5 διαπιστώνουμε πολύ λογικά ότι για σταθερή χωρητικότητα C όσο αυξάνεται το φορτίο κίνησης *α* τόσο αυξάνεται και το *B*. Η μεγάλη κίνηση κλήσεων συνοδεύεται και με μεγαλύτερη πιθανότητα απώλειας.

- Για να επιτύχουμε σταθερό B=0.01 *ή* B=0.03 με α από 0 έως 100 *erl* και C από 0 έως 120 *b.u.*



Εικόνα 6: Διάγραμμα με σταθερή χωρητικότητα *C*

Από το διάγραμμα (Εικόνα 6) βλέπουμε ότι για να πετύχουμε μία σταθερή πιθανότητα απώλειας κλήσης B θα πρέπει όσο αυξάνεται το φορτίο κίνησης α να αυξάνουμε καταλλήλως και τη χωρητικότητα C του συστήματος.

3. Μοντέλο *Erlang B* με δυνατότητα προσωρινής απώλειας των εξυπηρετητών

3.1 Εισαγωγή

Στην παρούσα ενότητα μελετάται το μοντέλο απωλειών κλήσεων τύπου *Erlang B*, εμπλουτισμένο με τη δυνατότητα εμφάνισης αστοχίας εξυπηρετητών [12], [13]. Συγκεκριμένα, εξετάζεται η περίπτωση κατά την οποία ένας εξυπηρετητής καθίσταται μη διαθέσιμος λόγω βλάβης (π.χ. καταστροφή εξοπλισμού), όταν βρίσκεται σε κατάσταση εξυπηρέτησης τη χρονική στιγμή της αστοχίας [6]. Στην περίπτωση αυτή, η εξυπηρετούμενη κλήση χάνεται, ενώ ο εξυπηρετητής τίθεται εκτός λειτουργίας για ένα χρονικό διάστημα, μέχρι την αποκατάστασή του.

Σύμφωνα με τις παραδοχές του μοντέλου, η διαδικασία αποκατάστασης των εξυπηρετητών πραγματοποιείται διαδοχικά. Επιτρέποντας την επιδιόρθωση ενός μόνο εξυπηρετητή κάθε φορά. Επομένως, σε περιπτώσεις πολλαπλών ταυτόχρονων αστοχιών. Η αποκατάσταση πραγματοποιείται σειριακά [7], [8].

Η ανάλυση του μοντέλου πραγματοποιείται μέσω τριών προσεγγίσεων:

- της προσέγγισης *Performability* (ο όρος αναλύεται στην παράγραφο 3.2.2) [7], [8],
- της μεθόδου BT, βασισμένης στην προσέγγιση των *Bobbio και Trivedi* [7], [8], [9] και
- της προτεινόμενης μεθόδου που βασίζεται στη μέθοδο *BT* [12].

Πριν από την παρουσίαση των ανωτέρω προσεγγίσεων, εξετάζεται ένα απλό παράδειγμα με χρήση της ακριβούς μεθόδου (*exact method*). Η ανάλυση αυτή βασίζεται στη μοντελοποίηση του συστήματος μέσω Μαρκοβιανής αλυσίδας και στον υπολογισμό των πιθανοτήτων απώλειας μέσω των εξισώσεων *GB* [7], [8].

3.2 Ανάλυση μοντέλου *Erlang B* με δυνατότητα προσωρινής απώλειας των εξυπηρετητών

Έστω ένα σύστημα απωλειών κλήσεων τύπου *Erlang B*, στο οποίο οι αφίξεις των κλήσεων ακολουθούν διαδικασία *Poisson* με μέσο ρυθμό άφιξης λ_{new}, ενώ οι χρόνοι εξυπηρέτησης είναι εκθετικά κατανεμημένοι με μέσο ρυθμό εξυπηρέτησης μ_{new}.

Για τη μοντελοποίηση της αστοχίας και αποκατάσταση των εξυπηρετητών, υιοθετούνται οι ακόλουθες παραδοχές [12], [13]*:*

- Κάθε εξυπηρετητής που βρίσκεται σε κατάσταση λειτουργίας μπορεί να υποστεί αστοχία με μέσο ρυθμό λ_f, ενώ η αποκατάστασή του πραγματοποιείται με μέσο ρυθμό μ_r .
- Οι ελεύθεροι (μη απασχολημένοι) εξυπηρετητές θεωρείται ότι δεν υπόκεινται σε αστοχίες.
- Η διαδικασία επιδιόρθωσης είναι σειριακή, δηλαδή επιτρέπεται η αποκατάσταση ενός μόνο εξυπηρετητή κάθε χρονική στιγμή.
- Σε περίπτωση αστοχίας εξυπηρετητή κατά τη διάρκεια εξυπηρέτησης, η αντίστοιχη κλήση διακόπτεται και θεωρείται χαμένη.

Με βάση τις παραπάνω υποθέσεις, το σύστημα αναλύεται αρχικά μέσω της ακριβούς μεθόδου (exact method) [12], [13], αξιοποιώντας τη μοντελοποίηση με Μαρκοβιανή αλυσίδα και τις εξισώσεις σφαιρικής ισορροπίας. Στη συνέχεια, εξετάζονται και οι τρεις προσεγγίσεις που παρουσιάστηκαν προηγουμένως, με στόχο τη συγκριτική αξιολόγηση της απόδοσης και της ακρίβειάς τους.

3.2.1 Ακριβής ανάλυση (*exact method*) μέσω ενός απλού παραδείγματος

Αρχικά, εξετάζεται ένα απλό παράδειγμα με χρήση της ακριβούς μεθόδου (exact method), με στόχο την αξιολόγηση του βαθμού απόκλισης των τριών προσεγγίσεων σε σχέση με την ακριβή λύση.

Παράδειγμα 3.1

Έστω ένα σύστημα με συνολική χωρητικότητα $C = 3$ *b.u.*, στο οποίο οι αφίξεις των κλήσεων ακολουθούν διαδικασία *Poisson* με μέσο ρυθμό λ_{new}, ενώ οι χρόνοι εξυπηρέτησης είναι εκθετικά κατανεμημένοι με μέσο ρυθμό μ_{new} .

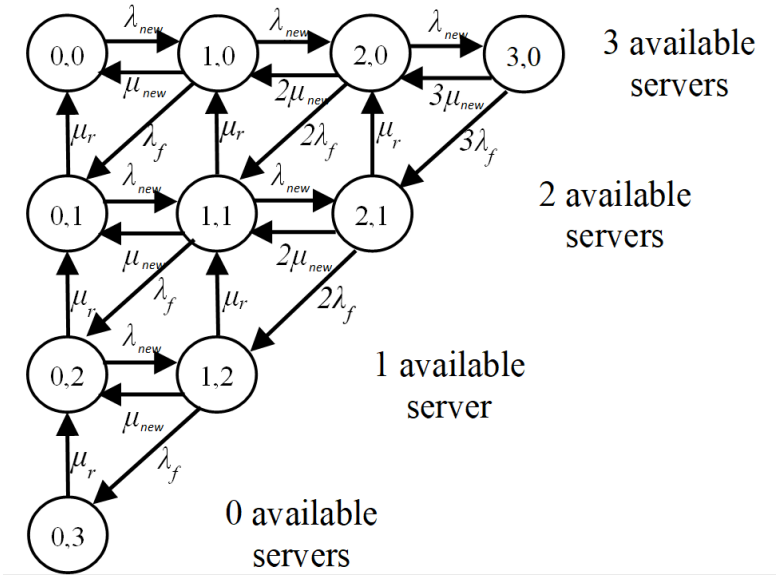
Το σύστημα περιγράφεται από ένα σύνολο καταστάσεων της μορφής (n_b , n_f), όπου n_b θεωρούμε τον αριθμό των απασχολημένων εξυπηρετητών και με n_f τον αριθμό των εξυπηρετητών που βρίσκονται σε κατάσταση αστοχίας. Με βάση τους περιορισμούς του συστήματος, προκύπτει ότι υπάρχουν συνολικά δέκα δυνατές καταστάσεις.

Στον παρακάτω πίνακα παρουσιάζονται αυτές οι καταστάσεις ως εξής:

n_b	n_f	$n_b + n_f \leq C$
0	0	0
0	1	1
0	2	2
0	3	3
1	0	1
1	1	2
1	2	3
2	0	2
2	1	3
3	0	3

Πίνακας 1: Πίνακας Καταστάσεων Ενότητας 3.2.1

Με βάση τα δεδομένα του Πίνακα 1, διαμορφώνεται η αντίστοιχη Μαρκοβιανή αλυσίδα, η οποία απεικονίζεται στο Σχήμα 8 [12], [13]. Η αλυσίδα αυτή περιγράφει τις δυνατές μεταβάσεις μεταξύ των καταστάσεων του συστήματος, λαμβάνοντας υπόψη τόσο τις αφίξεις και τις ολοκληρώσεις εξυπηρέτησης κλήσεων όσο και τα γεγονότα αστοχίας και αποκατάστασης των εξυπηρετητών.



Σχήμα 8: Μαρκοβιανή αλυσίδα παραδείγματος της Ενότητας 3.2.1

Πριν προχωρήσουμε στη διατύπωση των εξισώσεων *GB* του συστήματος, κρίνεται σκόπιμο να επισημανθούν τα ακόλουθα [12], [13]:

- Η δυνατότητα αποκατάστασης ενός μόνο εξυπηρετητή κάθε χρονική στιγμή αποτυπώνεται στο διάγραμμα καταστάσεων μέσω των μεταβάσεων με ρυθμό μ_r. Ενδεικτικά, παρατηρείται η διαδοχική μετάβαση από την κατάσταση (0,3) στην κατάσταση (0,2) και στη συνέχεια στην κατάσταση (0,1), γεγονός που υποδηλώνει τη σειριακή διαδικασία επιδιόρθωσης των εξυπηρετητών.
- Το γεγονός ότι οι κλήσεις που εξυπηρετούνταν από εξυπηρετητές οι οποίοι υφίστανται αστοχία χάνονται οριστικά αποτυπώνεται επίσης στις μεταβάσεις με ρυθμό μ_r. Για παράδειγμα, η μετάβαση από την κατάσταση (0,1) στην κατάσταση (0,0) υποδηλώνει ότι, μετά την αποκατάσταση του εξυπηρετητή, δεν υπάρχει ενεργή κλήση προς εξυπηρέτηση. Αντίθετα, σε περίπτωση που υποθέταμε επαναφορά της κλήσης, η αντίστοιχη μετάβαση θα ήταν από την κατάσταση (0,1) στην κατάσταση (1,0).

Οι εξισώσεις *GB* του συγκεκριμένου παραδείγματος είναι οι εξής:

$$\mu_{new}p(1,0) + \mu_r p(0,1) = \lambda_{new}p(0,0) \quad (3.1)$$

$$\lambda_f p(1,0) + \mu_{new}p(1,1) + \mu_r p(0,2) = (\lambda_{new} + \mu_r)p(0,1) \quad (3.2)$$

$$\lambda_f p(1,1) + \mu_{new}p(1,2) + \mu_r p(0,3) = (\lambda_{new} + \mu_r)p(0,2) \quad (3.3)$$

$$\lambda_f p(1,2) = \mu_r p(0,3) \quad (3.4)$$

$$\lambda_{new} p(0,0) + 2\mu_{new} p(2,0) + \mu_r p(1,1) = (\lambda_{new} + \lambda_f + \mu_{new}) p(1,0) \quad (3.5)$$

$$\lambda_{new} p(0,1) + 2\mu_{new} p(2,1) + 2\lambda_f p(2,0) + \mu_r p(1,2) = (\mu_{new} + \mu_r + \lambda_f + \lambda_{new}) p(1,1) \quad (3.6)$$

$$\lambda_{new} p(0,2) + 2\lambda_f p(2,1) = (\mu_{new} + \mu_r + \lambda_f) p(1,2) \quad (3.7)$$

$$2\lambda_{new} p(1,0) + 3\mu_{new} p(3,0) + \mu_r p(2,1) = (\lambda_{new} + 2\mu_{new} + 2\lambda_f) p(2,0) \quad (3.8)$$

$$\lambda_{new} p(1,1) + 3\lambda_f p(3,0) = (2\lambda_f + 2\mu_{new} + \mu_r) p(2,1) \quad (3.9)$$

$$\lambda_{new} p(2,0) = (3\mu_{new} + 3\lambda_f) p(3,0) \quad (3.10)$$

Αν θεωρήσουμε $\lambda_{new} = \mu_{new} = \lambda_f = \mu_r = 1$ τότε οι εξισώσεις μετασχηματίζονται σε:

$$p(1,0) + p(0,1) = p(0,0)$$

$$p(1,0) + p(1,1) + p(0,2) = 2p(0,1)$$

$$p(1,1) + p(1,2) + p(0,3) = 2p(0,2)$$

$$p(1,2) = p(0,3)$$

$$p(0,0) + 2p(2,0) + p(1,1) = 3p(1,0)$$

$$p(0,1) + 2p(2,1) + 2p(2,0) + p(1,2) = 4p(1,1)$$

$$p(0,2) + 2p(2,1) = 3p(1,2)$$

$$2p(1,0) + 3p(3,0) + p(2,1) = 5p(2,0)$$

$$p(1,1) + 3\lambda_f p(3,0) = 5p(2,1)$$

$$p(2,0) = 6p(3,0)$$

ή καλύτερα στην παρακάτω μορφή:

$$p(1,0) + p(0,1) - p(0,0) = 0 \quad (3.11)$$

$$p(1,0) + p(1,1) + p(0,2) - 2p(0,1) = 0 \quad (3.12)$$

$$p(1,1) + p(1,2) + p(0,3) - 2p(0,2) = 0 \quad (3.13)$$

$$p(1,2) - p(0,3) = 0 \quad (3.14)$$

$$p(0,0) + 2p(2,0) + p(1,1) - 3p(1,0) = 0 \quad (3.15)$$

$$p(0,1) + 2p(2,1) + 2p(2,0) + p(1,2) - 4p(1,1) = 0 \quad (3.16)$$

$$p(0,2) + 2p(2,1) - 3p(1,2) = 0 \quad (3.17)$$

$$2p(1,0) + 3p(3,0) + p(2,1) - 5p(2,0) = 0 \quad (3.18)$$

$$p(1,1) + 3p(3,0) - 5p(2,1) = 0 \quad (3.19)$$

$$p(2,0) - 6p(3,0) = 0 \quad (3.20)$$

Η λύση του συστήματος των 10 εξισώσεων (Σχέσεις 3.11 ως 3.20) δίνει τα παρακάτω αποτελέσματα:

$$p(0,0) = 0.338028 \quad (3.21)$$
$$p(0,1) = 0.169014 \quad (3.22)$$
$$p(0,2) = 0.084507 \quad (3.23)$$
$$p(0,3) = 0.042254 \quad (3.24)$$
$$p(1,0) = 0.169014 \quad (3.25)$$
$$p(1,1) = 0.084507 \quad (3.26)$$
$$p(1,2) = 0.042254 \quad (3.27)$$
$$p(2,0) = 0.042254 \quad (3.28)$$
$$p(2,1) = 0.021127 \quad (3.29)$$
$$p(3,0) = 0.007042 \quad (3.30)$$

Η πιθανότητα απώλειας των νεοεισερχόμενων κλήσεων είναι:

$$B_{EX} = p(3,0) + p(2,1) + p(1,2) + p(0,3) =$$
$$= 0.007042 + 0.021127 + 0.042254 + 0.04225 = 0.112677$$

Δηλαδή. έχουμε:

$$B_{EX} = 0.112677 \quad (3.31)$$

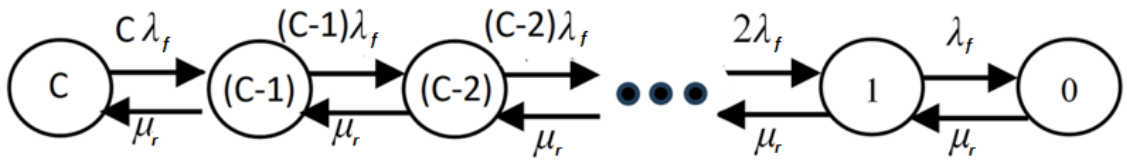
3.2.2 1η προσέγγιση: Μέθοδος *Performability*

Ο όρος *Performability* αποτελεί σύνθετη έννοια που προκύπτει από τους όρους *performance* (απόδοση) και *reliability* (αξιοπιστία). Συνεπώς, αναφέρεται στη συνδυαστική αξιολόγηση τόσο της απόδοσης όσο και της αξιοπιστίας ενός συστήματος.

Στο πλαίσιο της συγκεκριμένης προσέγγισης, η Μαρκοβιανή αλυσίδα που παρουσιάστηκε στην προηγούμενη ενότητα μπορεί να αποσυντεθεί εννοιολογικά σε δύο επιμέρους Μαρκοβιανές αλυσίδες. Η πρώτη περιγράφει τη διαθεσιμότητα των εξυπηρετητών, δηλαδή τις διαδικασίες αστοχίας και αποκατάστασης. Η δεύτερη αναπαριστά ένα κλασικό σύστημα

απωλειών τύπου *Erlang-B,* χωρίς να λαμβάνει υπόψη τις καταρρεύσεις και τις επιδιορθώσεις των εξυπηρετητών [7], [8].

Με τον τρόπο αυτό, η προσέγγιση *Performability* επιτρέπει τον διαχωρισμό και ταυτόχρονα τη συνδυαστική μελέτη των επιδράσεων της απόδοσης και της αξιοπιστίας στη συνολική λειτουργία του συστήματος.



Σχήμα 9: 1η Μαρκοβιανή αλυσίδα για τη διαθεσιμότητα των εξυπηρετητών

Στο Σχήμα 9 απεικονίζονται τόσο οι εξυπηρετητές που βρίσκονται σε κατάσταση λειτουργίας όσο και εκείνοι που παραμένουν ανενεργοί. Κατά συνέπεια, κατά τη μετάβαση του συστήματος από μία κατάσταση του συστήματος σε μία άλλη, ο εξυπηρετητής που ενδέχεται να καταρρεύσει μπορεί είτε να εξυπηρετεί ενεργά αιτήματα είτε να βρίσκεται σε κατάσταση αδράνειας. Ωστόσο, μια τέτοια θεώρηση αποκλίνει από την προσέγγιση που λαμβάνει υπόψη αποκλειστικά τους εξυπηρετητές οι οποίοι βρίσκονται σε χρήση.

Για παράδειγμα, έστω ότι έχουμε ένα σύστημα με χωρητικότητα $C = 3$ *b.u.,* με ρυθμό $3\lambda_f$ ένας από τους τρεις εξυπηρετητές μπορεί να καταρρεύσει και το σύστημα πηγαίνει στην κατάσταση $C = 2$ *b.u.* . Με βάση το Σχήμα 9, ο ρυθμός $3\lambda_f$ εμφανίζεται μόνο στη μετάβαση από την κατάσταση (3,0) στην κατάσταση (2,1), δηλαδή από την κατάσταση που είχαμε τρεις ενεργούς εξυπηρετητές στην κατάσταση με δύο ενεργούς και έναν εκτός λειτουργίας λόγω αστοχίας.

Από την άλλη πλευρά στο Σχήμα 9, η κατάσταση με διαθέσιμη χωρητικότητα $C = 2$ *b.u.* μπορεί να προκύψει από διαφορετικές επιμέρους καταστάσεις του συστήματος, ανάλογα με τον αριθμό των απασχολημένων και των κατεστραμμένων εξυπηρετητών.

Συγκεκριμένα είναι δυνατόν να προκύψει από:

- την κατάσταση (0,1), όπου κανένας εξυπηρετητής δεν βρίσκεται σε χρήση (δηλαδή και οι δύο διαθέσιμοι εξυπηρετητές είναι ανενεργοί) και ένας εξυπηρετητής βρίσκεται σε κατάσταση αστοχίας,
- την κατάσταση (1,1), όπου ένας εξυπηρετητής βρίσκεται σε χρήση, ένας είναι ανενεργός και ένας βρίσκεται σε κατάσταση αστοχίας,
- την κατάσταση (2,1), όπου δύο εξυπηρετητές βρίσκονται σε χρήση και ένας έχει καταρρεύσει.

Από τις εξισώσεις *LB* με βάση το Σχήμα 9 έχουμε:

$$C\lambda_f P_C^{av} = \mu_r P_{C-1}^{av} => P_{C-1}^{av} = \frac{\lambda_f}{\mu_r} C P_C^{av} => P_{C-1}^{av} = \frac{\lambda_f}{\mu_r} \frac{C!}{(C-1)!} P_C^{av}$$

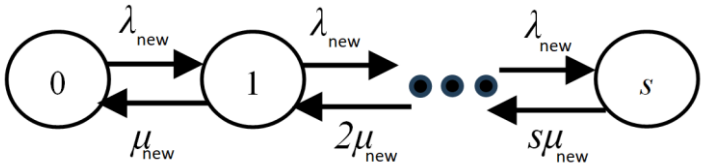
$$(C-1)\lambda_f P_{C-1}^{av} = \mu_r P_{C-2}^{av} => P_{C-2}^{av} = \frac{\lambda_f}{\mu_r}(C-1) P_{C-1}^{av} => P_{C-2}^{av} = (\frac{\lambda_f}{\mu_r})^2 \frac{C!}{(C-2)!} P_C^{av}$$

Με βάση τα παραπάνω καταλήγουμε στον τύπο της πιθανότητας μόνιμης κατάστασης P_i^{av} [7]:

$$P_i^{av} = (\frac{\lambda_f}{\mu_r})^{C-i} \frac{C!}{i!} P_C^{av}, \qquad για\ i = 0, \dots, C \tag{3.32}$$

$$όπου: P_C^{av} = \frac{1}{\sum_{j=0}^{C} (\frac{\lambda_f}{\mu_r})^j \frac{C!}{(C-j)!}} \tag{3.33}$$

Η δεύτερη Μαρκοβιανή αλυσίδα, Σχήμα 10, περιγράφει ένα τυπικό μοντέλο απωλειών *Erlang B* .



Σχήμα 10: 2η Μαρκοβιανή αλυσίδα για ένα τυπικό σύστημα *Erlang B*

Στο Σχήμα 10 [12], [13], κάθε κατάσταση αριθμείται, με τον αριθμό αυτό να εκφράζει το πλήθος των κατειλημμένων μονάδων b.u. ή ισοδύναμα των απασχολημένων εξυπηρετητών. Η μεταβλητή *s* που εμφανίζεται στο σχήμα αντιστοιχεί στον αριθμό αυτό, όπου *s* = 1, 2, 3, …, *C*.

Συνεπώς, εάν το σύστημα διαθέτει *C* λειτουργικούς εξυπηρετητές (με βάση το Σχήμα 9), τότε στο Σχήμα 10 αντιστοιχεί $s = C$. Αντίστοιχα αν οι διαθέσιμοι εξυπηρετητές μειωθούν σε *C*-1, τότε προκύπτει $s = C$-1, και ούτω καθεξής.

Με άλλα λόγια, το μοντέλο που απεικονίζεται στο Σχήμα 9 μπορεί να θεωρηθεί ως ένα σύνολο από *C* επιμέρους απλά μοντέλα απωλειών *Erlang-B*, καθένα από τα οποία αντιστοιχεί σε διαφορετικό επίπεδο διαθέσιμης χωρητικότητας ***(s)*** και λειτουργεί ως ανεξάρτητο υποσύστημα υπό συγκεκριμένες συνθήκες διαθεσιμότητας εξυπηρετητών.

Από τα παραπάνω προκύπτει ότι η πιθανότητα μόνιμης κατάστασης P_r^{per} [2]:

$$P_r^{per} = \frac{\frac{(\lambda_{new}/\mu_{new})^r}{r!}}{\sum_{i=0}^{s}(\frac{\lambda_{new}/\mu_{new}}{i!})^i}, όπου\ r = 0,1,\dots,s \qquad (3.34)$$

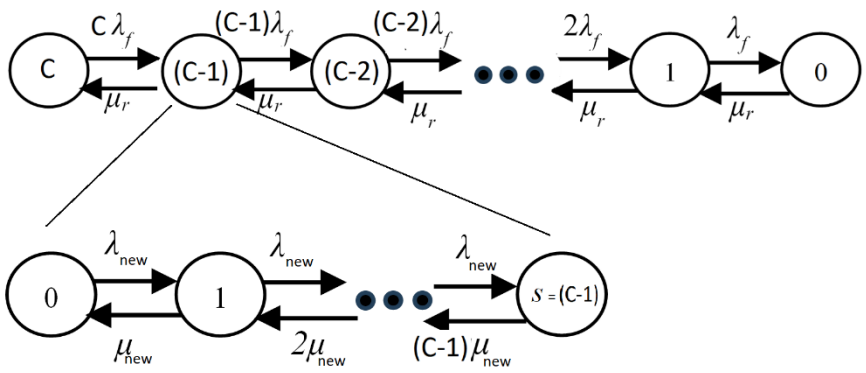
Τα δύο μοντέλα που αναλύσαμε με τις δύο Μαρκοβιανές αλυσίδες αποτελούν τα μοντέλα:

1. *The pure availability model,* δηλαδή το μοντέλο διαθεσιμότητας εξυπηρετητών και
2. *The pure performance model,* που περιγράφει ένα τυπικό μοντέλο απωλειών *ErlangB*

Η σύνθεση αυτών των μοντέλων αποτελεί το μοντέλο: *The pure Performability model* [12], [13].

Σύμφωνα με το μοντέλο που απεικονίζεται στο Σχήμα 11, όταν το σύστημα διαθέτει *C*-1 διαθέσιμους λειτουργικούς εξυπηρετητές τότε το *s* είναι *C*-1. Σε αυτή την περίπτωση, το σύστημα μπορεί να θεωρηθεί ισοδύναμο με ένα σύστημα απωλειών τύπου Erlang-B με μειωμένη χωρητικότητα.

Το σύστημα οδηγείται σε κατάσταση μόνιμης ισορροπίας (*steady state*), και οι πιθανότητες κατάστασης υπολογίζονται με βάση τον τύπο (3.34).



Σχήμα 11: Η σύνθεση των αλυσίδων για τη *Performability* προσέγγιση

Από τις σχέσεις (3.32), (3.33) και (3.34) καταλήγουμε ότι ο τύπος της πιθανότητας απωλειών του μοντέλου *Performability* είναι [7]:

$$B_P = P_0^{av} + \sum_{i=1}^{C} P_i^{av} P_i^{per} \qquad (3.35)$$

Στον τύπο (3.35) να διευκρινίσουμε ότι όπου "*av*" εννοούμε *availability* και "per" *performance*. Επίσης, ο δείκτης μηδέν αναφέρεται στην περίπτωση όπου δεν υπάρχουν διαθέσιμοι εξυπηρετητές, ενώ η μεταβλητή *i* αναφέρεται στον αριθμό των διαθέσιμων εξυπηρετητών που βρίσκονται σε ενεργή χρήση.

Στο σημείο αυτό θα εξετάσουμε το παράδειγμα της ενότητας 3.2.1, υπενθυμίζουμε ότι: C=3 *b.u.* και $\lambda_{new}= \mu_{new}=\lambda_f= \mu_r=1$. Με βάση τους τύπους (3.32) , (3.33), (3.34) και (3.35) έχουμε:

$$P_3^{av} = \frac{1}{\sum_{j=0}^{3}(\frac{\lambda_f}{\mu_r})^j \frac{3!}{(3-j)!}} = \frac{1}{\sum_{j=0}^{3} 1 \frac{3!}{(3-j)!}} = \frac{1}{1+3+6+6} = 0.0625$$

$$P_2^{av} = (\frac{\lambda_f}{\mu_r})^{3-2} \frac{3!}{2!} P_3^{av} = 1 * 3 * 0.0625 = 0.1875$$

$$P_1^{av} = 1 * 6 * 0.0625 = 0.375$$

$$P_0^{av} = 1 * 6 * 0.0625 = 0.375$$

$$P_1^{per} = \frac{(\lambda_{new}/\mu_{new})^1}{\sum_{i=0}^{1}(\frac{\lambda_{new}/\mu_{new}}{i!})^i} = \frac{1}{1+1} = 0.5$$

$$P_2^{per} = \frac{(\lambda_{new}/\mu_{new})^2}{\sum_{i=0}^{2}(\frac{\lambda_{new}/\mu_{new}}{i!})^i} = \frac{1/2}{1+1+1/2} = 0.2$$

$$P_3^{per} = \frac{(\lambda_{new}/\mu_{new})^3}{\sum_{i=0}^{3}(\frac{\lambda_{new}/\mu_{new}}{i!})^i} = \frac{1/6}{1+1+1/2+1/6} = 0.0625$$

Με βάση τον τελικό τύπο (3.35) έχουμε:

$$B_P = P_0^{av} + \sum_{i=1}^{3} P_i^{av} P_i^{per} = P_0^{av} + P_1^{av} P_1^{per} + P_2^{av} P_2^{per} + P_3^{av} P_3^{per}$$
$$= 0.375 + 0.1875 + 0.0375 + 0.003906 = 0.603906$$

$$B_P = 0.603906 \qquad (3.36)$$

Το αποτέλεσμα που μας δίνει η προσέγγιση *Performability*, (3.36) αποκλίνει από αυτό της ακριβής μεθόδου (*exact method*) (2.31) που είναι: $B_{EX} = 0.112677$.

3.2.3 2η προσέγγιση: *BT* Μέθοδος

Η μέθοδος αυτή προτάθηκε από τους *Bobbio* και *Trivedi* [9] και βασίζεται στον διαχωρισμό των μεταβάσεων καταστάσεων σε γρήγορες και αργές. Συγκεκριμένα:

- Οι γρήγορες μεταβάσεις χαρακτηρίζονται από πολύ υψηλούς ρυθμούς, πολλών τάξεων μεγέθους μεγαλύτερους σε σχέση με τις αργές μεταβάσεις. Στο πλαίσιο της μελέτης μας οι ρυθμοί λ_{new} και μ_{new} θεωρούνται γρήγοροι.
- Οι αργές μεταβάσεις αντιστοιχούν σε χαμηλούς ρυθμούς (λ_f και μ_r) και περιλαμβάνουν τις καταρρεύσεις και επαναφορές εξυπηρετητών.

Επίσης, ο χώρος των καταστάσεων διαιρείται σε χώρο γρήγορων καταστάσεων και σε χώρο αργών καταστάσεων. Οι γρήγορες καταστάσεις είναι εκείνες που έχουν τουλάχιστον μία

γρήγορη εξερχόμενη μετάβαση π.χ. με βάση το Σχήμα 8 από την κατάσταση (1,0) έχουμε μία εξερχόμενη μετάβαση λ_{new} στην κατάσταση (2,0). Από την άλλη πλευρά οι αργές καταστάσεις δεν έχουν καμία εξερχόμενη μετάβαση π.χ. από το Σχήμα 8 μόνο η κατάσταση (0,3) είναι η μόνη αργή κατάσταση. Έτσι, στην περίπτωση που μελετάμε, η μόνη αργή κατάσταση θα είναι η πάντα η $(0,C)$.

Κάθε γρήγορο υποσύνολο (*fast recurrent subset*) αποτελεί ένα ανεξάρτητο υποσύστημα, που συνδέεται εσωτερικά με γρήγορες μεταβάσεις (λ_{new}, μ_{new}) και εξωτερικά με άλλα υποσύνολα μόνο μέσω αργών μεταβάσεων (λ_f, μ_r).

Στο παράδειγμά μας με $C = 3$ *b.u.*, το πρώτο γρήγορο υποσύνολο περιλαμβάνει τις καταστάσεις (0,0), (1,0), (2,0), (3,0), οι οποίες συνδέονται μεταξύ τους μέσω των γρήγορων μεταβάσεις με ρυθμούς λ_{new} *και* μ_{new}. Σύμφωνα με την προσέγγιση της *BT* μεθόδου, όλες οι καταστάσεις εντός ενός γρήγορου επαναλαμβανόμενου υποσυνόλου συνδέονται εσωτερικά μόνο με γρήγορες μεταβάσεις.

Κάθε υποσύνολο, όμως, συνδέεται εξωτερικά με άλλα υποσύνολα αποκλειστικά μέσω αργών μεταβάσεων, δηλαδή μεταβάσεων με μικρούς ρυθμούς, όπως οι λ_f και μ_r. Στην περίπτωση αυτή, το πρώτο γρήγορο υποσύνολο συνδέεται με το δεύτερο υποσύνολο, που περιλαμβάνει τις καταστάσεις (0,1), (1,1), (2,1), μέσω των αργών ρυθμών μεταβάσεων.

Με αυτόν τον τρόπο, κάθε γρήγορο υποσύνολο μπορεί να αναλυθεί ως ένα ξεχωριστό υποσύστημα *Erlang-B* με χωρητικότητα s ανάλογη του αριθμού των διαθέσιμων *b.u.* $(s = 1,2,\ldots,C)$, ενώ οι αργές μεταβάσεις επιτρέπουν τη σύνδεση των υποσυστημάτων για την εκτίμηση της συνολικής *performability* του συστήματος (Σχήμα 10).

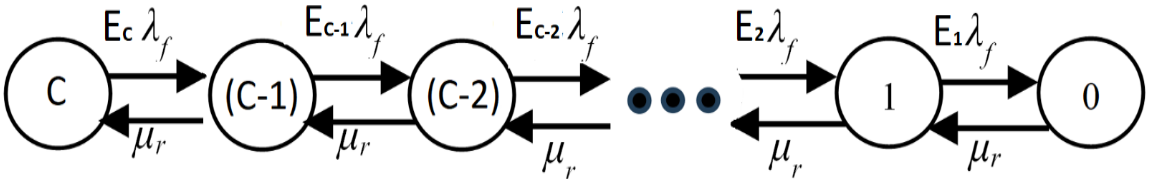
Για κάθε ένα γρήγορο υποσύνολο μπορούμε να χρησιμοποιήσουμε τον τύπο (3.34) και να υπολογίσουμε την πιθανότητα μόνιμης κατάστασης P_r^{per}.

Ακόμα μέσω της φόρμουλας:

$$E_s = \frac{\lambda_{new}}{\mu_{new}} (1 - P_s^{per}) \text{ , για } s=1,2,\ldots,C \qquad (3.37)$$

μπορούμε να βρούμε το πλήθος των απασχολημένων εξυπηρετητών (E_s).

Έχοντας καθορίσει τον τρόπο υπολογισμού του πλήθους E_s των εξυπηρετητών, σε κάθε ένα γρήγορο υποσύνολο από τα συνολικά C, μπορούμε να επανασχεδιάσουμε το Σχήμα 9 με βάση τη διαθεσιμότητα E_sτων εξυπηρετητών, Σχήμα 12.



Σχήμα 12: Τροποποιημένη αλυσίδα του Σχήματος 2.2 με βάση τη μέθοδο *BT*

Η κεντρική ιδέα για να υπολογίσουμε το πλήθος των εξυπηρετητών E_s είναι να ξέρουμε τη διαθεσιμότητα των ενεργών εξυπηρετητών. Στη μέθοδο *Performability* συνυπολογίζονται στην κατάρρευση και οι στάσιμοι εξυπηρετητές ενώ στη μέθοδο *BT* μόνο οι ενεργοί.

Έτσι η μέθοδος *BT* ικανοποιεί καλύτερα την αρχική μας υπόθεση ότι ένας εξυπηρετητής μπορεί να καταρρεύσει με ρυθμό λ_f αρκεί να είναι ενεργός, αφού δεν νοείται κατάρρευση σε ανενεργό εξυπηρετητή, σε αντίθεση με τη μέθοδο *Performability*.

Με βάση τις εξισώσεις της *LB* και από τον τύπο (3.37) έχουμε:

$$E_C\lambda_f P_C^{BT} = \mu_r P_{C-1}^{BT} => P_{C-1}^{BT} = \frac{\lambda_f}{\mu_r} E_C P_C^{BT}$$

$$E_{C-1}\lambda_f P_{C-1}^{BT} = \mu_r P_{C-2}^{BT} => P_{C-2}^{BT} = \frac{\lambda_f}{\mu_r} E_{C-1} P_{C-1}^{BT} => P_{C-2}^{BT} = (\frac{\lambda_f}{\mu_r})^2 E_c E_{C-1} P_C^{BT}$$

Στηριζόμενοι στα προηγούμενα, η πιθανότητα μόνιμης κατάστασης P_i^{BT} είναι [8]:

$$P_i^{BT} = (\frac{\lambda_f}{\mu_r})^{C-i} P_C^{BT} \prod_{s=i+1}^{C} E_s, \quad για \ i < C \qquad (3.38)$$

$$όπου \ P_C^{BT} = \frac{1}{1 + \sum_{j=0}^{C-1} (\frac{\lambda_f}{\mu_r})^{C-J} \prod_{s=j+1}^{C} E_s} \qquad (3.39)$$

Έτσι από τους τύπους (3.37), (3.38) και (3.39), για την πιθανότητα απωλειών κλήσης του συστήματος, έχουμε [8]:

$$B_{BT} = P_0^{BT} + \sum_{i=1}^{C} P_i^{BT} P_i^{per} \tag{3.40}$$

Παρατηρούμε ότι οι τύποι (3.35) και (3.40) εμφανίζουν ομοιότητες στη μορφή τους.

Συνεχίζουμε, μέσω του παραδείγματος της ενότητας 3.2.1, όπου C=3 *b.u.* και $\lambda_{new} = \mu_{new} = \lambda_f = \mu_r = 1$, να βρούμε την πιθανότητα απωλειών κλήσης του συστήματος B_{BT} μέσω της προσέγγισης *BT*.

Υπενθυμίζουμε ότι από την ενότητα 3.2.2 έχουμε υπολογίσει τα $P_1^{per}, P_2^{per}, P_3^{per}$ και οι τιμές τους είναι 0.5 , 0.2 , 0.0625 αντίστοιχα.

Μέσω του τύπου (3.37) έχουμε:

$$E_1 = \frac{\lambda_{new}}{\mu_{new}} \left(1 - P_1^{per}\right) = 0.5$$

$$E_2 = \frac{\lambda_{new}}{\mu_{new}} \left(1 - P_2^{per}\right) = 1 - 0.2 = 0.8$$

$$E_3 = \frac{\lambda_{new}}{\mu_{new}} \left(1 - P_3^{per}\right) = 1 - 0.0625 = 0.9375$$

Από τις σχέσεις (3.38) και (3.39) έχουμε:

$$P_3^{BT} = \frac{1}{1 + \sum_{j=0}^{3-1} (\frac{\lambda_f}{\mu_r})^{3-J} \prod_{s=j+1}^{3} E_s} = \frac{1}{1 + \sum_{j=0}^{2} \prod_{s=j+1}^{3} E_s}$$

$$= \frac{1}{1 + \prod_{s=1}^{3} E_s + \prod_{s=2}^{3} E_s + \prod_{s=3}^{3} E_s} = \frac{1}{1 + E_1 E_2 E_3 + E_2 E_3 + E_3}$$

$$= \frac{1}{1 + 0.5 * 0.8 * 0.9375 + 0.8 * 0.9375 + 0.9375}$$

$$= \frac{1}{1 + 0.375 + 0.75 + 0.9375} = \frac{1}{3.0625} = 0.326531$$

$$P_0^{BT} = (\frac{\lambda_f}{\mu_r})^{3-0} P_3^{BT} \prod_{s=0+1}^{3} E_s = 0.326531 * 0.5 * 0.8 * 0.9375 = 0.122449$$

$$P_1^{BT} = (\frac{\lambda_f}{\mu_r})^{3-1} P_3^{BT} \prod_{s=1+1}^{3} E_s = 0.326531 * 0.8 * 0.9375 = 0.244898$$

$$P_2^{BT} = (\frac{\lambda_f}{\mu_r})^{3-2} P_3^{BT} \prod_{s=2+1}^{3} E_s = 0.326531 * 0.9375 = 0.306122$$

Τέλος από τον τύπο (3.40) :

$$B_{BT} = P_0^{BT} + \sum_{i=1}^{3} P_i^{BT} P_i^{per} = 0.122449 + P_1^{BT} P_1^{per} + P_2^{BT} P_2^{per} + P_3^{BT} P_3^{per}$$
$$= 0.122449 + 0.244898 * 0.5 + 0.306122 * 0.2 + 0.326531 * 0.0625$$
$$= 0.326531$$

Δηλαδή:

$$B_{BT} = 0.326531 \quad (3.41)$$

Από το αποτέλεσμα (3.41) βλέπουμε ότι έχουμε καλύτερη προσέγγιση σε σχέση με το αποτέλεσμα (3.36) που είναι 0.603906, ωστόσο είμαστε πολύ μακριά από το ακριβές αποτέλεσμα (3.31) που είναι 0.112677.

Η σημαντική απόκλιση της *BT* μεθόδου σε σχέση με την ακριβή μέθοδο του συγκεκριμένου παραδείγματος αποδίδεται στο γεγονός ότι η *BT* μέθοδος υπερεκτιμά τον μέσο αριθμό διαθέσιμων ενεργών εξυπηρετητών. Συγκεκριμένα, η *BT* μέθοδος θεωρεί έναν μέσο αριθμό ενεργών εξυπηρετητών E_s σε ένα σύστημα απωλειών *Erlang-B* με *s* εξυπηρετητές που είναι πάντα διαθέσιμοι. Αντίθετα, όπως φαίνεται στο Σχήμα 8, οι ενεργοί εξυπηρετητές υπόκεινται σε πιθανότητα κατάρρευσης, με αποτέλεσμα ο πραγματικός μέσος αριθμός ενεργών εξυπηρετητών να είναι μικρότερος από αυτόν που υπολογίζει η *BT* μέθοδος [12], [13].

Αυτό έχει ως συνέπεια η BT μέθοδος να υποεκτιμά την πιθανότητα απώλειας κλήσεων σε σχέση με την ακριβή μέθοδο, ιδιαίτερα σε συστήματα με υψηλή πιθανότητα αποτυχίας εξυπηρετητή [12], [13].

Ο μέσος όρος των εξυπηρετητών του Σχήματος 8 με βάση τα αποτελέσματα των (3.25) ως (3.30) είναι:

$$E = 1 * \big(P(1,0) + P(1,1) + P(1,2)\big) + 2 * \big(P(2,0) + P(2,1)\big) + 3 * P(3,0)$$
$$= 0.295775 + 2 * 0.063381 + 3 * 0.007042 = 0.443663$$

Το αποτέλεσμα αυτό είναι μικρότερο από τα E_1 = 0.9375 E_2 = 0.8 και E_3=0.5 που μας δίνει η μέθοδος *BT*.

3.2.4 3η προσέγγιση: Προτεινόμενη μέθοδος βασιζόμενη στην *BT* Μέθοδο

Η τροποποίηση που προτείνεται στη μέθοδο των *Bobbio* και *Trivedi* αφορά τον χρόνο που ένας εξυπηρετητής μπορεί να μείνει ενεργός. Ο χρόνος αυτός ενδέχεται να είναι μικρότερος εξαιτίας μιας πιθανής κατάρρευσής του. Με αυτή την τροποποίηση, η εκτίμηση του μέσου αριθμού ενεργών εξυπηρετητών γίνεται πιο ρεαλιστική, βελτιώνοντας την ακρίβεια της πιθανότητας απώλειας κλήσεων [12].

Έτσι, αν ένας εξυπηρετητής μπορεί να καταρρεύσει με μέσο ρυθμό λ_f τότε ο χρόνος που παραμένει ενεργός μπορεί να είναι [12]:

Από

$$h = \frac{1}{\mu_{new}} \tag{3.42}$$

σε

$$h' = \frac{1}{\mu_{new} + \lambda_f} \tag{3.43}$$

Επομένως, υπολογίζουμε την πιθανότητα απώλειας κλήσης και τον μέσο αριθμό ενεργών εξυπηρετητών, για κάθε σύστημα απωλειών *Erlang B* με *s* εξυπηρετητές, όπου *s* = 1,2…,*C* αρκεί ο μέσος ρυθμός εξυπηρέτησης να είναι [12]:

$$\mu'_{new} = \mu_{new} + \lambda_f \tag{3.44}$$

Συνεχίζοντας το παράδειγμά της ενότητας 3.2.1, όπου *C*=3 *b.u.* και λ_{new}=μ_{new}=λ_f=μ_r=1, θα υπολογίσουμε την πιθανότητα απωλειών κλήσης του συστήματος B^{Prop} μέσω της προτεινόμενης μεθόδου που στηρίζεται στην προσέγγιση της *BT* μεθόδου.

Από τον τύπο (3.44) υπολογίζουμε το μ'_{new} που είναι:

$\mu'_{new} = \mu_{new} + \lambda_f = 1 + 1 = 2$

Από τον τύπο (3.34) αντικαθιστούμε όπου μ_{new} το μ'_{new} και έχουμε:

$$P_1^{per} = \frac{\frac{(\lambda_{new}/\mu'_{new})^1}{1!}}{\sum_{i=0}^{1}(\frac{\lambda_{new}/\mu'_{new}}{i!})^i} = \frac{0.5}{1+0.5} = 0.3333$$

$$P_2^{per} = \frac{\frac{(\lambda_{new}/\mu'_{new})^2}{2!}}{\sum_{i=0}^{2}(\frac{\lambda_{new}/\mu'_{new}}{i!})^i} = \frac{0.5^2/2}{1+0.5+(0.5^2/2)} = 0.076923$$

$$P_3^{per} = \frac{\frac{(\lambda_{new}/\mu'_{new})^3}{3!}}{\sum_{i=0}^{3}(\frac{\lambda_{new}/\mu'_{new}}{i!})^i} = \frac{0.5^3/6}{1+0.5+(0.5^2/2)+0.5^3/6} = 0.012658$$

Μέσω του τύπου (3.37) και των παραπάνω αποτελεσμάτων έχουμε:

$$E_1 = \frac{\lambda_{new}}{\mu'_{new}}\left(1-P_1^{per}\right) = 0.5(1-0.3333) = 0.3333$$

$$E_2 = \frac{\lambda_{new}}{\mu'_{new}}\left(1-P_2^{per}\right) = 0.5(1-0.076923) = 0.461538$$

$$E_3 = \frac{\lambda_{new}}{\mu'_{new}}\left(1-P_3^{per}\right) = 0.5(1-0.012658) = 0.493671$$

Από τις σχέσεις (3.38) και (3.39) έχουμε:

$$P_3^{BT} = \frac{1}{1+\sum_{j=0}^{3-1}(\frac{\lambda_f}{\mu_r})^{3-J}\prod_{s=j+1}^{3}E_s} = \frac{1}{1+\sum_{j=0}^{2}\prod_{s=j+1}^{3}E_s}$$

$$= \frac{1}{1+\prod_{s=1}^{3}E_s+\prod_{s=2}^{3}E_s+\prod_{s=3}^{3}E_s} = \frac{1}{1+E_1E_2E_3+E_2E_3+E_3}$$

$$= \frac{1}{1+0.3333*0.461538*0.493671+0.461538*0.493671+0.493671}$$

$$= \frac{1}{1+0.075942+0.227848+0.493671} = \frac{1}{1.797461} = 0.556340$$

$$P_0^{BT} = (\frac{\lambda_f}{\mu_r})^{3-0}P_3^{BT}\prod_{s=0+1}^{3}E_s = 0.556340*0.3333*0.461538*0.493671 = 0.042249$$

$$P_1^{BT} = (\frac{\lambda_f}{\mu_r})^{3-1} P_3^{BT} \prod_{s=1+1}^{3} E_s = 0.556340 * 0.461538 * 0.493671 = 0.126760$$

$$P_2^{BT} = (\frac{\lambda_f}{\mu_r})^{3-2} P_3^{BT} \prod_{s=2+1}^{3} E_s = 0.556340 * 0.493671 = 0.274649$$

Οπότε από τον τύπο που υπολογίζει την πιθανότητα απώλειας κλήσης:

$$B_{PROP} = P_0^{BT} + \sum_{i=1}^{3} P_i^{BT} P_i^{per}$$
$$= 0.042249 + (0.126760 * 0.3333) + (0.274649 * 0.076923)$$
$$+ (0.556340 * 0.012658)$$
$$= 0.042249 + 0.042249 + 0.021127 + 0.007042 = 0.112667$$

Δηλαδή:

$B_{PROP} = 0.112667$ *(3.45)*

Φαίνεται, τελικά ότι έχουμε το ίδιο αποτέλεσμα με το αποτέλεσμα (3.31) της *exact method* [12], η οποία είναι:

$B_{EX} = 0.112677$

3.3 Επεξήγηση προγραμμάτων της επέκτασης Erlang B

Τα τμήματα κώδικα των αντίστοιχων προγραμμάτων περιλαμβάνονται στο Παράρτημα Γ.

3.3.1 Επεξήγηση του προγράμματος με βάση τους τύπους της επέκτασης Erlang B

Στο συγκεκριμένο πρόγραμμα εφαρμόζονται οι μαθηματικές προσεγγίσεις που αναλύθηκαν στα προηγούμενα κεφάλαια και, όπως στην περίπτωση του απλού μοντέλου *Erlang B*, η λογική των παραγοντικών ακολουθείται σύμφωνα με όσα περιγράφονται στην Ενότητα 2.5.1 και το Παράρτημα Β. Για παράδειγμα, το α^c υπολογίζεται ως e^{clna} .

3.3.2 Επεξήγηση του προγράμματος προσομοίωσης της επέκτασης Erlang B

Στην προσομοίωση του μοντέλου της επέκτασης *Erlang B* με δυνατότητα προσωρινής απώλειας εξυπηρετητών, ακολουθούμε την ίδια φιλοσοφία με εκείνη της προσομοίωσης του απλού μοντέλου *Erlang B*. Ωστόσο, προστίθεται η διαχείριση καταρρεύσεων εξυπηρετητών και ο χρόνος επιδιόρθωσής τους. Συγκεκριμένα, καθώς οι κλήσεις καταφθάνουν στο σύστημα (*Poisson* αφίξεις), υπολογίζονται οι χρόνοι άφιξης με τον τρόπο που περιγράφεται στην προσομοίωση του απλού μοντέλου *Erlang B* και αποθηκεύονται σε σωρό ελαχίστων (min Heap), ώστε να μπορεί το σύστημα να επεξεργάζεται τα συμβάντα με χρονική σειρά εμφάνισης.

Η ιδιαιτερότητα στην προκειμένη περίπτωση είναι ότι στον σωρό εκτός από τους χρόνους αποθηκεύονται και οι καταστάσεις γεγονότων. Αυτό γίνεται, γιατί ο χρόνος του συστήματος επηρεάζεται και από τις καταρρεύσεις των εξυπηρετητών και από το χρόνο που χρειάζεται ο κάθε ένας να επιδιορθωθεί.

Να επισημανθεί ότι σε περίπτωση πολλαπλών καταρρεύσεων εξυπηρετητών, η επιδιόρθωση γίνεται διαδοχικά, ένας εξυπηρετητής τη φορά**,** και όχι ταυτόχρονα. Για την υλοποίηση αυτού του μηχανισμού, χρησιμοποιείται μια ουρά αναμονής FIFO**,** η οποία κρατά τους εξυπηρετητές που περιμένουν να επιδιορθωθούν, διασφαλίζοντας ότι κάθε εξυπηρετητής επανέρχεται στο σύστημα με τη σωστή σειρά.

Επομένως, τα είδη των καταστάσεων είναι τα εξής:

1. ”arrival” - εννοούμε τις εισερχόμενες κλήσεις.
2. “call” - οι κλήσεις που διεκπεραιώνονται χωρίς βλάβη εξυπηρετητή.
3. “server_crash” - έχουμε βλάβη εξυπηρετητή.
4. “repair” - έχουμε επιδιόρθωση εξυπηρετητή.

Ειδικότερα, τρέχει μία επαναληπτική διαδικασία που ελέγχει και επεξεργάζεται κάθε στοιχείο του σωρού που βρίσκεται στην πρώτη θέση, πριν το διαγράψει. Υπενθυμίζουμε, ότι στο απλό μοντέλο Erlang B είχαμε μόνο χρόνους που αφορούσαν τις κλήσεις, μιας και μόνον αυτές επηρέαζαν το σύστημα. Εδώ, αντίθετα, έχουμε και άλλους χρόνους που επηρεάζουν το σύστημα (χρόνους κατάρρευσης και χρόνους επιδιόρθωσης εξυπηρετητών)

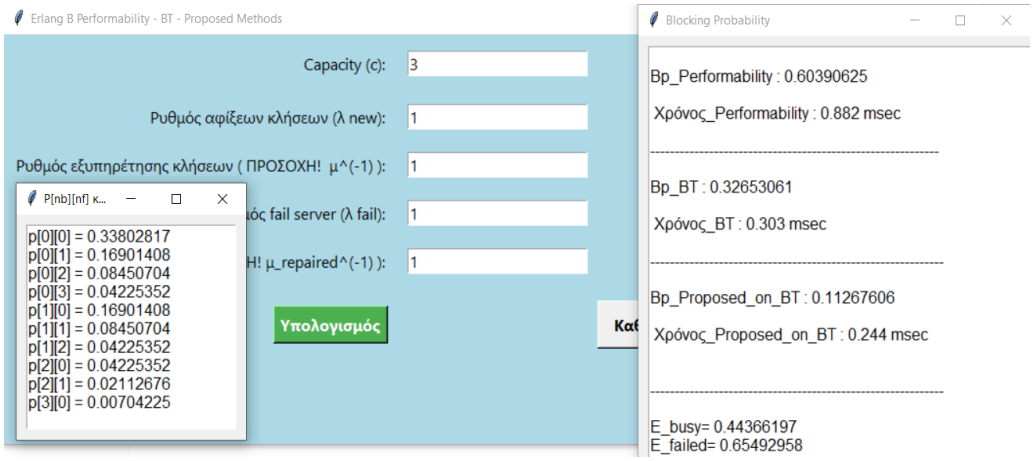
Με βάση την πληροφορία της κατάστασης από το πρώτο στοιχείο του σωρού έχουμε τα εξής:

1. Αν η μεταβλητή της κατάστασης έχει την τιμή “arrival” (δηλαδή ο χρόνος αντιστοιχεί στο χρόνο άφιξης της κλήσης) και υπάρχουν διαθέσιμοι εξυπηρετητές τότε υπολογίζονται οι χρόνοι διεκπεραίωσης της κλήση από το σύστημα και:
   - Αν υπάρχει μέσα στον χρόνο αυτό κατάρρευση εξυπηρετητή, τότε η κλήση χάνεται (δεν γίνεται αποθήκευση στο σωρό) και στον σωρό καταχωρείται ο χρόνος κατάρρευσης μαζί με την κατάσταση “server crash”.
   - Αν διαπιστωθεί με βάση τους χρόνους ότι δεν υφίστανται κατάρρευση εξυπηρετητή, κατά το χρόνο εξυπηρέτησης της κλήσης τότε αυτή αποθηκεύεται στο σωρό με βάση τον χρόνο εξυπηρέτησής της και την κατάσταση “call”.
2. Αν η μεταβλητή της κατάστασης έχει την τιμή “call”, ουσιαστικά αυτό σημαίνει ότι δεν είχαμε κατάρρευση εξυπηρετητή και η κλήση εξυπηρετήθηκε κανονικά. Διαγράφεται από το σύστημα (σωρό) η κλήση και απελευθερώνεται ο εξυπηρετητής που την εξυπηρετούσε.
3. Αν η μεταβλητή της κατάστασης έχει την τιμή “server_crash” (όπου ο χρόνος είναι ο χρόνος κατάρρευσης εξυπηρετητή) τότε υπολογίζεται ο χρόνος που χρειάζεται ο εξυπηρετητής να επιδιορθωθεί. Στην πράξη, αυτό σημαίνει ότι ο συγκεκριμένος εξυπηρετητής είναι προσωρινά δεσμευμένος και το σύστημα λειτουργεί χωρίς αυτόν κατά τη διάρκεια της επιδιόρθωσης. Έτσι, αποθηκεύεται ο χρόνος επιδιόρθωσης μαζί με την κατάσταση “repair_server” στον σωρό.

   Δεδομένου ότι μπορεί να υπάρξουν πολλαπλές καταρρεύσεις εξυπηρετητών και ότι το σύστημα μπορεί να επιδιορθώνει έναν εξυπηρετητή τη φορά, χρησιμοποιείται μία ουρά FIFO που κρατά τις χρονικές διάρκειες επιδιόρθωσης των εξυπηρετητών που έχουν καταρρεύσει και δεν έχουν ακόμη επιδιορθωθεί. Μόνο ο χρόνος του πρώτου εξυπηρετητή στην ουρά λαμβάνεται υπόψη κάθε φορά, και όταν ολοκληρωθεί η επιδιόρθωσή του, η επόμενη διάρκεια παίρνει τη σειρά της από την ουρά.
4. Αν η μεταβλητή της κατάστασης έχει την τιμή “repair_server” τότε επανέρχεται στο σύστημα ο συγκεκριμένος εξυπηρετητής γιατί έχει πλέον επιδιορθωθεί, στην ουσία αυξάνονται κάθε φορά οι μονάδες b.u. του συστήματος.

3.4 Αποτελέσματα προγραμμάτων της επέκτασης Erlang B

Τα αποτελέσματα του παραδείγματος της Ενότητας 3.2.1 (για $C = 3\ b.u.$ και $\lambda_{new} = \mu_{new} = \lambda_f = \mu_r = 1$) που μας δίνουν οι κώδικες των προσεγγίσεων και της προσομοίωσης, παρουσιάζονται παρακάτω στις Εικόνες 7 και 8.



Εικόνα 7: Παράδειγμα 3.1 Μοντέλα *Performability*, *BT* και Προτεινόμενο

Εικόνα 8: Παράδειγμα 3.1 Προσομοίωση

Στον Πίνακα 2 βρίσκονται τα αποτελέσματα για διαφορετικές τιμές στα λ_f, μ_r όταν το $C = 20\ b.u.$ και $\lambda_{new} = 8$, $\mu_{new} = 1$. Για την περίπτωση της προσομοίωσης τρέξαμε το πρόγραμμα για 500000 κλήσεις με Transit 5000 κλήσεις και για τα seeds (λ_{new} , μ_{new}) πήραμε τις διαδοχικές τιμές: (2,4) , (4,5) , (5,6) , (6,7) , (7,8) αντίστοιχά, ενώ για τα seeds (λ_f , μ_r) πήραμε μόνο τις τιμές (2,4) αντίστοιχα.

	Proposed method	**Performability method**	**BT method**	**Προσομοίωση**
λ_f=1.0, μ_r = 0.1	0.975	0.988903	0.9875	0.975662±0.000176
λ_f=0.5, μ_r = 0.1	0.9625	0.977833	0.975	0.963442±0.000401
λ_f=0.1, μ_r = 0.1	0.8625	0.890361	0.875	0.865617±0.002349
λ_f=0.05, μ_r = 0.1	0.7375	0.784193	0.750	0.745211±0.002678
λ_f=0.01, μ_r = 0.1	0.016687	0.170037	0.018255	0.016426±0.001895
λ_f=1.0, μ_r = 1.0	0.75	0.890361	0.875	0.753025±0.000926
λ_f=0.5, μ_r = 1.0	0.625	0.784193	0.750	0.631583±0.006382
λ_f=0.1, μ_r = 1.0	0.007274	0.170037	0.018255	0.007960±0.000837
λ_f=0.05, μ_r = 1.0	0.000342	0.010921	0.00060	0.000354±0.000079

Πίνακας 2: Αποτελέσματα παρ. Ενότητας 3.2.5

Από τον Πίνακα 2 παρατηρούμε ότι:

- Όταν ο ρυθμός επιδιόρθωσης μ_r είναι μικρός ανεξάρτητα του ρυθμού κατάρρευσης λ_f όλες οι προσεγγίσεις και η προσομοίωση δίνουν αποτελέσματα χωρίς μεγάλες αποκλίσεις. Αντίθετα, όταν ο ρυθμός επιδιόρθωσης αυξάνεται τότε τα αποτελέσματα αποκλίνουν.
- Η απόκλιση αυξάνεται ακόμα περισσότερο, όταν κατά την αύξηση του ρυθμού επιδιόρθωσης έχουμε, μαζί, και μείωση του ρυθμού κατάρρευσης. Δηλαδή, όταν δεν συμβαίνουν πολλές καταρρεύσεις και αυτές επιδιορθώνονται γρήγορα.
- Σ' όλα τα παραδείγματα η προσομοίωση δίνει τα ίδια αποτελέσματα με την Προτεινόμενη μέθοδο.

Ως γενικό συμπέρασμα, από τα αποτελέσματα των παραδειγμάτων θα μπορούσαμε να αναφέρουμε ότι σε ασταθή περιβάλλοντα με δυσκολία άμεσης επέμβασης για επιδιόρθωση θα μπορούσαν να χρησιμοποιηθούν όλες οι προσεγγίσεις, με καλά αποτελέσματα. Αν, όμως επιθυμούμε ακρίβεια αποτελεσμάτων τότε ενδείκνυται η προτεινόμενη μέθοδος.

4. Μοντέλο απωλειών πολυδιάστατης κίνησης *EMLM*

4.1 Εισαγωγή

Στο παρόν κεφάλαιο εξετάζεται το μοντέλο απωλειών πολυδιάστατης κίνησης, γνωστό ως *Erlang Multirate Loss Model (EMLM)*. Το μοντέλο αυτό περιγράφει ένα σύστημα εξυπηρέτησης συνολικής χωρητικότητας C, στο οποίο καταφθάνουν κλήσεις διαφορετικών κατηγοριών. Κάθε κατηγορία κλήσεων χαρακτηρίζεται από συγκεκριμένες απαιτήσεις σε μονάδες εύρους ζώνης *b.u.*για την εξυπηρέτησή της.

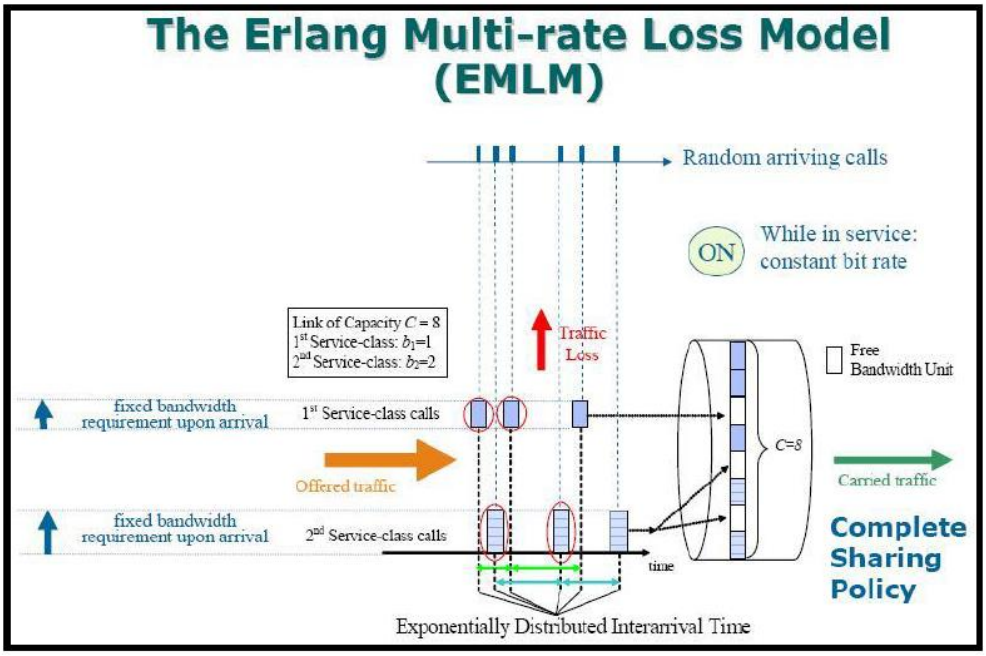
Το σύστημα κατανέμει δυναμικά τους διαθέσιμους πόρους χωρίς να επιβάλλει περιορισμούς στον τρόπο διάθεσής τους μεταξύ των κατηγοριών κλήσεων. Εφόσον κατά τη στιγμή άφιξης μιας κλήσης υπάρχουν διαθέσιμες b.u. επαρκείς για την κάλυψη της απαιτούμενης χωρητικότητας, οι πόροι δεσμεύονται και η κλήση γίνεται αποδεκτή προς εξυπηρέτηση. Αντίθετα, στην περίπτωση που οι διαθέσιμοι πόροι δεν επαρκούν για την ικανοποίηση της απαίτησης, η κλήση απορρίπτεται και θεωρείται μπλοκαρισμένη.

Οποιαδήποτε τηλεπικοινωνιακή κίνηση περιλαμβάνει περισσότερες από μία κατηγορίες, ονομάζεται πολυδιάστατη κίνηση (*multi – dimensional traffic*) [27].

4.2 Το μοντέλο *EMLM* υπό την πολιτική πλήρους διάθεσης CS

Έστω ότι διαθέτουμε μία ζεύξη με συνολική χωρητικότητα *C b.u.,* η οποία μπορεί να εξυπηρετήσει κλήσεις *k* διαφορετικών κατηγοριών, όπου *K* είναι ο συνολικός αριθμός κατηγοριών και $k = 1,2,3,\ldots K$. Θεωρούμε ότι στο σύστημα έρχονται κλήσεις με μέσο ρυθμό άφιξης λ_k *(Poisson αφίξεις).* Κάθε κλήση της κατηγορίας *k* απαιτεί b_k μονάδες εύρους ζώνης για χρονική διάρκεια που ακολουθεί εκθετική κατανομή με μέσο ρυθμό εξυπηρέτησης μ_k.

Αν το σύστημα διαθέτει τις απαιτούμενες μονάδες b.u., αυτές δεσμεύονται και η κλήση γίνεται αποδεκτή προς εξυπηρέτηση. Σε αντίθετη περίπτωση, αν οι διαθέσιμες μονάδες δεν επαρκούν για την κάλυψη της απαίτησης της κλήσης, η κλήση απορρίπτεται και θεωρείται μπλοκαρισμένη.



Εικόνα 9: Μοντέλο *EMLM*

Για την καλύτερη κατανόηση του μοντέλου θα αναλύσουμε την Εικόνα 9 [26], στην οποία παρουσιάζεται ένα σύστημα με $C = 8$ *b.u.* και οι κατηγορίες κίνησης είναι δύο, $K=2$. Η πρώτη κατηγορία ζητά $b_1 = 1$ *b.u.* και η δεύτερη $b_2 = 2$ *b.*u.

Στην Εικόνα 9, απεικονίζεται μία στιγμιαία κατάσταση του μοντέλου *EMLM* υπό την πολιτική πλήρους διάθεσης (Complete Sharing, CS). Συγκεκριμένα, υπάρχουν τρεις διαθέσιμες μονάδες *b.u.*, οι οποίες κατανέμονται ως εξής: 1 *b.u.* σε μία κλήση της κατηγορίας 1 και 2 *b.u.* σε μία κλήση της κατηγορίας 2, $b_1 + b_2 = 3$ *b.u.*.

Οι υπόλοιπες τέσσερεις κλήσεις (δύο της δεύτερης και δύο της πρώτης κατηγορίας) δεν μπορούν να ικανοποιηθούν τη δεδομένη χρονική στιγμή, με αποτέλεσμα να μπλοκάρονται (σημειώνονται στην Εικόνα 9 με την κόκκινη γραμμή).

Επίσης, να τονίσουμε ότι δεν υπάρχει περιορισμός στη διάθεση των μονάδων *b.u.*, δηλαδή οι *b.u.* που παραχωρούνται στις κλήσεις δεν χρειάζεται να είναι συνεχόμενες στη ζεύξη.

Αναλυτικό Μοντέλο

Το αναλυτικό μοντέλο χρησιμοποιείται για τον υπολογισμό των πιθανοτήτων κατάστασης *q(j)* στο μοντέλο *EMLM*. Οι πιθανότητες αυτές υπολογίζονται με τον αναδρομικό τύπο των

Kaufman–Roberts, ο οποίος επιτρέπει να προσδιορίζεται η πιθανότητα το σύστημα να βρίσκεται σε κάθε επιτρεπόμενη κατάσταση δεσμευμένων πόρων. Ο τύπος αυτός είναι [2]:

$$q(j) = \begin{cases} 1 & \alpha\nu\ j = 0 \\ \frac{1}{j}\sum_{k=1}^{K} \alpha_k b_k q(j - b_k) & \alpha\nu\ 1 \le j \le C \\ 0 & \alpha\nu\ j < 0\ \text{ή}\ j > C \end{cases} \quad (4.1)$$

Με φορτίο κίνησης [2]:

$$\alpha_k = \frac{\lambda_k}{\mu_\kappa} \quad (4.2)$$

Ο αναδρομικός τύπος υπολογίζει τις μη κανονικοποιημένες πιθανότητες *q(j),* όπου *j* το πλήθος των κατειλημμένων *b.u. με j=1,2,3…C*

Για να κανονικοποιηθούν οι παραπάνω πιθανότητες χρησιμοποιούμε τη σταθερά κανονικοποίησης *G* με βάση τον τύπο [2]:

$$G = \sum_{j=0}^{C} q(j) \quad (4.3)$$

Οι κανονικοποιημένες πιθανότητες *Q(j)* υπολογίζονται με βάση τον τύπο [2]:

$$Q(j) = \frac{q(j)}{G} \quad (4.4)$$

Για τον υπολογισμό του τύπου της πιθανότητας απώλειας της κλήσης [2]:

$$B_k = \sum_{j=C-b_k+1}^{C} G^{-1} q(j) \quad (4.5)$$

Για τον υπολογισμό του ποσοστού χρήσης [2]:

$$U = \sum_{j=1}^{C} j\frac{q(j)}{G} \quad (4.6)$$

Παράδειγμα 4.1

Έστω ότι έχουμε 2 κατηγορίες κίνησης $K = 2$ σε ένα σύστημα εύρους ζώνης $C = 5$ *b.u.* τα φορτία κίνησης είναι $\alpha_1=\alpha_2=1$ και οι μονάδες εύρους ζώνης της κάθε κατηγορίας είναι : $b_1 = 1$ *b.u.* και $b_2 = 2$ b.u.

Με βάση τον αναδρομικό τύπο του *Kaufman - Roberts* (4.1) έχουμε:

$jq(j) = a_1 b_1 q(j - b_1) + a_2 b_2 q(j - b_2)$ δηλαδή $jq(j) = q(j - 1) + 2q(j - 2)$

Για $0 \le j \le C$ δηλαδή $0 \le j \le 5$ έχουμε:

$q(0) = 1$

$1q(1) = q(0) + 0 = 1$

$2q(2) = q(1) + 2q(0) = 3$ άρα $q(2) = 1.5$

$3q(3) = q(2) + 2q(1) = 3.5$ άρα $q(3) = 1.1667$

$4q(4) = q(3) + 2q(2) = 4.1667$ άρα $q(4) = 1.0417$

$5q(5) = q(4) + 2q(3) = 3.3751$ άρα $q(5) = 0.675$

Η σταθερά κανονικοποίησης : $G = \sum_{j=0}^{5} q(j) = 6.3834$

Οι κανονικοποιημένες πιθανότητες είναι:

$$Q(0) = \frac{q(0)}{G} = \frac{1}{6.3834} = 15.67\%$$

$$Q(1) = \frac{q(1)}{G} = \frac{1}{6.3834} = 15.67\%$$

$$Q(2) = \frac{q(2)}{G} = \frac{1.5}{6.3834} = 23.49\%$$

$$Q(3) = \frac{q(3)}{G} = \frac{1.1667}{6.3834} = 18.28\%$$

$$Q(4) = \frac{q(4)}{G} = \frac{1.0417}{6.3834} = 16.32\%$$

$$Q(5) = \frac{q(5)}{G} = \frac{0.675}{6.3834} = 10.57\%$$

Οι πιθανότητες απώλειας της κάθε κλήσης είναι:

$$B_1 = \sum_{j=C-b_1+1}^{C} G^{-1}q(j) = \sum_{j=5-1+1}^{5} G^{-1}q(j) = \frac{q(5)}{G} = \frac{0.675}{6.3834} = 10.57\%$$

$$B_2 = \sum_{j=C-b_2+1}^{C} G^{-1}q(j) = \sum_{j=5-2+1}^{5} G^{-1}q(j) = \frac{q(4)+q(5)}{G} = \frac{1.0417+0.675}{6.3834}$$
$$= 26.89\%$$

Για τον υπολογισμό του ποσοστού χρήσης *U* (*Utilization*):

$$U = \sum_{j=1}^{C} j\frac{q(j)}{G} = 1\frac{q(1)}{G} + 2\frac{q(2)}{G} + 3\frac{q(3)}{G} + 4\frac{q(4)}{G} + 5\frac{q(5)}{G}$$
$$= 1Q(1) + 2Q(2) + 3Q(3) + 4Q(4) + 5Q(5)$$
$$= 1 * 0.1567 + 2 * 0.2349 + 3 * 0.1828 + 4 * 0.1632 + 5 * 0.1057$$
$$= 2.3562\ b.u.$$

4.3 Το μοντέλο *EMLM* υπό την πολιτική δέσμευσης εύρους ζώνης BR

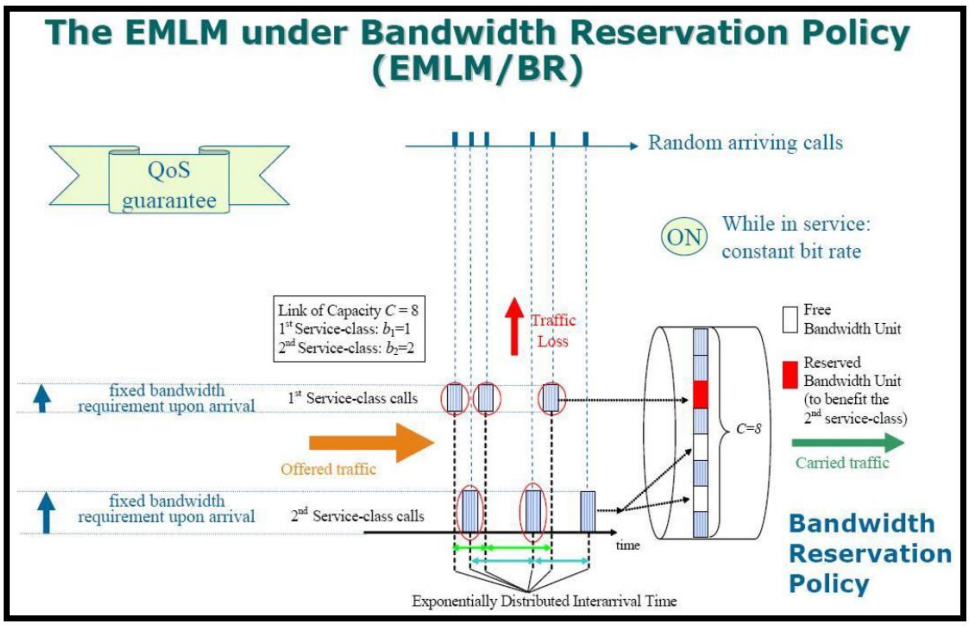
Σε πολλές περιπτώσεις τηλεπικοινωνιακών συστημάτων, υπάρχει η ανάγκη ορισμένες κλήσεις, συγκεκριμένης κατηγορίας, να έχουν προτεραιότητα έναντι άλλων [2] για να διασφαλιστεί η ποιότητά τους . Οι κατηγορίες αυτές είναι, συνήθως, κατηγορίες κλήσεων που χρειάζονται αυξημένο εύρος ζώνης. Για το λόγο αυτό ακολουθείται η πολιτική δέσμευσης εύρους ζώνης της ζεύξης (*Bandwidth Reservation, BR*).

Με βάση την πολιτική *BR* ένα μέρος του εύρους ζώνης δεσμεύεται αποκλειστικά για χρήση από τις κλήσεις άλλων κατηγοριών. Αν θεωρήσουμε τη δέσμευση μονάδων *b.u.* ως tr_k *(trunk reservation)* τότε η κλήση αυτής της κατηγορίας *k* γίνεται αποδεκτή όταν βρει στο σύστημα το πολύ $C\text{-}tr_k$ διαθέσιμα *b.u.*, διαφορετικά μπλοκάρεται. Θα πρέπει να επισημάνουμε ότι σε κλήσεις άλλων κατηγοριών που έχουν διαφορετικό tr_k, εφαρμόζεται η πολιτική *BR* σ' αυτές με βάση το δικό τους tr_k . Τέλος, αν υπάρχουν κλήσεις κατηγοριών που δεν έχουν *BR* τότε σ' αυτές δεν υπάρχει κάποιος περιορισμός και ζητούν *b.u.* αρκεί να είναι μικρότερο ή ίσο με *C*. Δηλαδή ευνοούνται έναντι των πρώτων.

Ουσιαστικά, αυτή η πολιτική εφαρμόζεται για εκείνες τις υπηρεσίες που ζητούν αυξημένες μονάδες *b.u.* π.χ. ροή βίντεο σε πραγματικό χρόνο. Έτσι, επιλέγοντας το είδος της κράτησης, για μεγάλες απαιτήσεις σε *b.u.* έχουμε λιγότερες πιθανότητες απόρριψης των κλήσεων αυτών.

Η εξισορρόπηση του συστήματος κατά την εφαρμογή της πολιτικής BR, δηλαδή η επίτευξη ίσων πιθανοτήτων απώλειας κλήσεων μεταξύ των διαφορετικών κατηγοριών κλήσεων, επιτυγχάνεται όταν το άθροισμα των απαιτούμενων μονάδων εύρους ζώνης κάθε κατηγορίας κλήσης και της αντίστοιχης παραμέτρου δέσμευσης tr_k είναι το ίδιο για όλες τις κατηγορίες κλήσεων. Με άλλα λόγια, η συνθήκη ισορροπίας του συστήματος ικανοποιείται όταν ισχύει η ακόλουθη σχέση [2]:

$$b_1 + tr_1 = b_2 + tr_2 = \cdots = b_k + tr_k \tag{4.7}$$



Εικόνα 10: Μοντέλο *EMLM/BR*

Στην Εικόνα 10 [26] μπορούμε να διακρίνουμε δύο κατηγορίες κίνησης κλήσεων $K = 2$ με χωρητικότητα ζεύξης $C = 8$ *b.u.* Οι απαιτήσεις σε μονάδες εύρους ζώνης ανά κατηγορία κλήσεων είναι $b_1 = 1$ *b.u.* και $b_2 = 2$ *b.*u.

Στην Εικόνα 10 απεικονίζεται μία στιγμιαία κατάσταση του συστήματος υπό την πολιτική *BR*. Τη συγκεκριμένη χρονική στιγμή, υπάρχουν τρεις διαθέσιμες μονάδες *b.u.*, εκ των

οποίων η μία είναι δεσμευμένη (tr_1=1) και σημειώνεται με κόκκινο χρώμα. Η δέσμευση αυτή γίνεται για να ευνοήσει τις κλήσεις της κατηγορίας 2.

Όταν καταφθάνει μία κλήση της δεύτερης κατηγορίας οι διαθέσιμες *b.u.* που απαιτεί επαρκούν για την εξυπηρέτησή της, οπότε η κλήση γίνεται αποδεκτή. Αντίθετα, όταν στη συνέχεια καταφθάνει μία κλήση της κατηγορίας 1 που απαιτεί 1 *b.u.*, αυτή δεν μπορεί να εξυπηρετηθεί παρά τη διαθέσιμη χωρητικότητα, διότι η συγκεκριμένη μονάδα b.u. είναι δεσμευμένη υπέρ των κλήσεων της κατηγορίας 2. Ως αποτέλεσμα, η κλήση της κατηγορίας 1 απορρίπτεται (μπλοκάρεται).

Αναλυτικό Μοντέλο

Όπως και στην πολιτική CS, έτσι και εδώ για τον υπολογισμό των πιθανοτήτων *q(j)* του μοντέλου *EMLM/BR*, με βάση τον αναδρομικό τύπο του *Roberts*, είναι ο εξής [2]:

$$q(j) = \begin{cases} 1 & \alpha\nu\ j = 0 \\ \frac{1}{j}\sum_{k=1}^{K} \alpha_k D_k(j-b_k) q(j-b_k) & \alpha\nu\ 1 \le j \le C \\ 0 & \alpha\nu\ j < 0 \text{ ή } j > C \end{cases} \quad (4.8)$$

Όπου:

$$D_k(j-b_k) = \begin{cases} b_k & \gamma\iota\alpha\ j \le C - tr_k \\ 0 & \gamma\iota\alpha\ j > C - tr_k \end{cases} \quad (4,9)$$

και φορτίο κίνησης α_k, με βάση τον τύπο (4.2).

Για τον υπολογισμό της πιθανότητας απώλειας της κλήσης ο τύπος είναι:

$$B_k = \sum_{j=C-b_k-tr_k+1}^{C} G^{-1} q(j) \quad (4.10)$$

και *G* η σταθερά κανονικοποίησης, με βάση τον τύπο (4.3).

Παράδειγμα 4.2

Ομοίως με το Παράδειγμα 4.1 έχουμε 2 κατηγορίες κίνησης ($K = 2$) σε ένα σύστημα με χωρητικότητα $C = 5$ *b.u.* τα φορτία κίνησης είναι $\alpha_1=\alpha_2=1$ *erl* και οι μονάδες εύρους ζώνης της κάθε κατηγορίας είναι : $b_1 = 1$ *b.u.* και $b_2 = 2$ *b.u.* Επίσης, έχουμε δέσμευση μονάδων *b.u.* μόνο στην πρώτη κατηγορία κλήσεων $tr_1 = 1$ *b.u.*

Με βάση τον αναδρομικό τύπο του *Roberts* (4.8) και (4.9) έχουμε:

$D_1(j - b_1) = b_1, \quad$ όταν $j \le C - tr_1$ δηλαδή όταν $j \le 5 - 1 \Rightarrow j \le 4$

$$jq(j) = \begin{cases} q(j-1) + 2q(j-2), & \text{όταν } j = 1,2,3,4 \\ 2q(j-2), & \text{όταν } j = 5 \end{cases}$$

$q(0) = 1$

$1q(1) = q(0) + 0 = 1$ άρα $q(1) = 1$

$2q(2) = q(1) + 2q(2) = 3$ άρα $q(2) = 1.5$

$3q(3) = q(2) + 2q(1) = 3.5$ άρα $q(3) = 1.667$

$4q(4) = q(3) + 2q(2) = 4.1667$ άρα $q(4) = 1.0417$

$5q(5) = 2q(3) = 2.3333$ άρα $q(5) = 0.4667$

Η σταθερά κανονικοποίησης με βάση τον τύπο (3.3) είναι : $G = 6.1751$

Για τον υπολογισμό της πιθανότητας απώλειας της κλήσης με βάση τον τύπο (4.10) έχουμε:

$$B_1 = \sum_{j=C-b_1-tr_1+1}^{C} G^{-1}q(j) = \sum_{j=4}^{5} G^{-1}q(j) = \frac{q(4)+q(5)}{G} = 24.43\%$$

$$B_2 = \sum_{j=C-b_2+1}^{C} G^{-1}q(j) = \sum_{j=4}^{5} G^{-1}q(j) = \frac{q(4)+q(5)}{G} = 24.43\%$$

Επιτυγχάνοντας εξισορρόπηση του συστήματος, κάτι που ήταν αναμενόμενο με βάση τη σχέση (4.7)

$b_1 + tr_1 = b_2 \rightarrow 2 = 2$.

Παράδειγμα 4.3

Έστω ότι έχουμε 2 κατηγορίες κίνησης ($K = 2$) σε ένα σύστημα χωρητικότητας $C = 4$ *b.u.* τα φορτία κίνησης είναι $\alpha_1 = \alpha_2 = 1$ *erl* και οι μονάδες εύρους ζώνης της κάθε κατηγορίας είναι: $b_1 = 1$ *b.u.* και $b_2 = 1$ *b.u.* Επίσης, έχουμε δέσμευση *b.u.* μόνο στην δεύτερη κατηγορία κλήσεων $tr_2 = 2$ *b.u.*

Από τον αναδρομικό τύπο του *Roberts* (4.8) και (4.9) έχουμε:

$D_2(j - b_2) = b_2, \quad$ όταν $j \leq C - tr_2$ δηλαδή όταν $j \leq 4 - 2 \Rightarrow j \leq 2$

$$jq(j) = \begin{cases} q(j-1) + q(j-1), & \text{όταν } j = 1,2 \\ q(j-1), & \text{όταν } j = 3,4 \end{cases}$$

$q(0) = 1$

$1q(1) = q(0) + q(0) = 2$ άρα $q(1) = 2$

$2q(2) = q(1) + q(1) = 4$ άρα $q(2) = 2$

$3q(3) = q(2) = 2$ άρα $q(3) = 0.6666$

$4q(4) = q(3) = 0.6666$ άρα $q(4) = 0.16665$

Η σταθερά κανονικοποίησης με βάση τον τύπο (4.3) είναι : $G = 5.8333$

Για τον υπολογισμό της πιθανότητας απώλειας της κλήσης με βάση τον τύπο (4.10) είναι:

$$B_1 = \sum_{j=C-b_1+1}^{C} G^{-1}q(j) = \sum_{j=4}^{4} G^{-1}q(j) = \frac{q(4)}{G} = 0.0285687$$

$$B_2 = \sum_{j=C-b_2-tr_2+1}^{C} G^{-1}q(j) = \sum_{j=2}^{4} G^{-1}q(j) = \frac{q(2)+q(3)+q(4)}{G} = 0.4857017$$

Εδώ δεν επιτυγχάνεται εξισορρόπηση μιας και δεν ικανοποιείται η σχέση (4.7) :

$b_1 \neq b_2 + tr_2 \rightarrow 1 \neq 3$

4.4 Επεξήγηση των προγραμμάτων του *EMLM*

Σημειώνεται ότι τα τμήματα κώδικα των αντίστοιχων προγραμμάτων τόσο του αναλυτικού μοντέλου όσο και της προσομοίωσης, περιλαμβάνονται στο Παράρτημα Δ.

4.4.1 Επεξήγηση προγράμματος του αναλυτικού μοντέλου

Κατά την εκτέλεση της προγραμματιστικής εφαρμογής του μοντέλου *EMLM* με βάση τον αναδρομικό τύπο των *Kaufman και Roberts*, εμφανίζεται παραθυρικό περιβάλλον που επιτρέπει στον χρήστη να εισάγει τα απαραίτητα δεδομένα, όπως:

- Το πλήθος των κατηγοριών κίνησης K ,
- Το συνολική χωρητικότητα C,
- Τους μέσους ρυθμούς άφιξης κλήσεων λ_k για κάθε κατηγορία,
- Τους μέσους ρυθμούς εξυπηρέτησης κλήσεων μ_k για κάθε κατηγορία,
- Τις απαιτήσεις σε μονάδες *b.u.* κάθε κατηγορίας.

Από τις τιμές των λ_k και μ_k θα υπολογιστούν τα αντίστοιχα φορτία κίνησης a_k των κατηγοριών, τα οποία χρησιμοποιούνται στη συνέχεια για την εκτίμηση των πιθανοτήτων μπλοκαρίσματος και την ανάλυση της απόδοσης του συστήματος.

Στα αντίστοιχα πεδία εισαγωγής δεδομένων για τις τιμές των K, λ , μ και μονάδων *b.u.*, ο χρήστης μπορεί να εισάγει τις τιμές διαδοχικά διαχωρισμένες είτε με κενό (*Space*), είτε με *Enter,* είτε με κόμμα.

Σε όλες τις φόρμες ισχύουν οι εξής κανόνες: Αν τα δεδομένα είναι μη έγκυρα, όπως π.χ. γράμματα, αρνητικές τιμές κ.α. εμφανίζεται κατάλληλο μήνυμα σφάλματος και η εφαρμογή δεν επιτρέπει την εξαγωγή αποτελεσμάτων. Μόνο με την εισαγωγή έγκυρων τιμών συνεχίζεται η διαδικασία.

Όταν τα δεδομένα περάσουν επιτυχώς όλους τους ελέγχους, εμφανίζεται μήνυμα επιτυχούς επικύρωσης, καθώς και οι υπολογισμένες τιμές των φορτίων κίνησης. Ακολούθως, ξεκινά ο υπολογισμός της φόρμουλας *Kaufman-Roberts*. Η διαδικασία επικύρωσης και υπολογισμού ενεργοποιείται με τη χρήση του κουμπιού:

«Έλεγχος και Υπολογισμός»

Έπειτα, εμφανίζονται οι πιθανότητες *B* κάθε κατηγορίας καθώς και η *U*.

Η *U* υπολογίζεται με 3 διαφορετικούς τρόπους [2] :

1. Με τον τύπο της μέσης αναμενόμενης τιμής (4.6),
2. Με τον τύπο (2.16)
3. Από τη μέση τιμή κλήσεων κάθε κατηγορίας n_k (4.11) [2], αφού πρώτα υπολογίσουμε τις μέσες τιμές των κλήσεων της κατηγορίας *K* στην κατάσταση *j* (Y_{kj}) με βάση τους τύπους (4.12), (4.13), (4.14).

$$U = n_1 * b_1 + n_2 * b_2 + ... + n_k * b_k \qquad (4.11)$$

Σημείωση [2]:

- $\bar{n}_k = \sum_{j=1}^{C} y_k(j) \frac{q(j)}{G}$, με σταθερά κανονικοποίησης *G* από τον τύπο (4.3) (4.12) [2]
- $Y_k(j) = \alpha_k \frac{q(j-b_k)}{q(j)}$, $q(j)>0$, $j = 1,2,\ldots C$ με $Y_k(j) = 0$, αν $j<b_k$ (4.13) [2]
- Επίσης, *το j υπολογίζεται :* $j = Y_k(1) + Y_k(2) + Y_k(3) + \ldots + Y_k(K)$ (4.14) [2]

Η τιμή της *U* εμφανίζεται επιπλέον και ως ποσοστό.

Τελικά, εκτυπώνονται τα εξής :

- Η μέση τιμή κλήσεων κάθε κατηγορίας $\bar{n}_k$,
- το συνολικό άθροισμα του Blocking από *1…C* ,
- η σταθερά *G* ,
- *U και U%,*
- ο χρόνος εκτέλεσης και
- το πλήθος των πράξεων των υπολογισμών του αλγόριθμου που σχετίζονται αμιγώς με τη φόρμουλα των *Kaufman-Roberts*.

Το πλήθος των πράξεων υπολογίστηκε να είναι:

*C*K (συγκρίσεις, προσθέσεις, 2* πολλαπλασιασμούς, 1 διαίρεση), C (προσθέσεις) , C (διαιρέσεις), K*(C-bi+1 ως C) προσθέσεις, 3 πολλαπλασιασμούς για καθάρισμα λιστών και 1 διαίρεση.*

Δηλαδή :

$$5*C*K + C + C + K*(C-(C-bi+1)) + 4 = C*(5*K + 2) + K*(C-(C-bi+1)) + 4 \qquad (4.15)$$

Οι πιθανότητες απώλειας κλήσεων των κατηγοριών *B* εμφανίζονται σε ξεχωριστό παράθυρο αποτελεσμάτων, το οποίο παραμένει ενεργό για όσο επιθυμεί ο χρήστης. Με αυτόν τον τρόπο, είναι δυνατή η εκτέλεση πολλαπλών σεναρίων με διαφορετικά δεδομένα εισόδου και η συγκριτική αξιολόγηση των αποτελεσμάτων μεταξύ τους.

Επιπλέον, η εφαρμογή περιλαμβάνει ένα πεδίο επιλογής (*check box*) με την ένδειξη «Προβολή ενδιάμεσων τιμών». Αν επιλεγεί, εμφανίζονται σε ξεχωριστό παράθυρο οι ενδιάμεσοι υπολογισμοί των μη κανονικοποιημένων πιθανοτήτων *q*, καθώς και οι κανονικοποιημένες πιθανοτήτων *Q* (στο πρόγραμμα συμβολίζονται με *q'*), προσφέροντας στον χρήστη αναλυτική εικόνα της διαδικασίας υπολογισμού.

Η εφαρμογή δίνει μία επιπλέον λειτουργία ανάλυσης χωρητικότητας. Συγκεκριμένα, αφού εκτελεστεί η διαδικασία «Έλεγχος και Υπολογισμός» και εξαχθούν οι τιμές *B* για κάθε κατηγορία, ο χρήστης μπορεί να εισάγει ένα όριο *B*. Το πρόγραμμα συγκρίνει τις τιμές *B* κάθε κατηγορίας με το όριο αυτό και, αν βρεθεί ότι οποιαδήποτε τιμή υπερβαίνει το καθορισμένο όριο, αυξάνει τη χωρητικότητα *C* κατά μία μονάδα και επαναλαμβάνει τον υπολογισμό των *B* για κάθε κατηγορία. Η διαδικασία αυτή επαναλαμβάνεται αυτομάτως μέχρι να ικανοποιηθεί η συνθήκη ότι όλες οι τιμές *B* όλων των κατηγοριών βρίσκονται κάτω από το όριο του χρήστη. Η ενεργοποίηση αυτής της λειτουργίας πραγματοποιείται μέσω του κουμπιού:

«*Εύρεση Επιθυμητού Capacity*»

Με άλλα λόγια, η εφαρμογή υπολογίζει την ελάχιστη χωρητικότητα *C* ώστε τα *B* όλων των κατηγοριών να βρίσκονται κάτω από το όριο που έχει ορίσει ο χρήστης. Σημειώνεται ότι, για να ενεργοποιηθεί η εύρεση του επιθυμητού *C*, θα πρέπει προηγουμένως να έχει εκτελεστεί έστω μία φορά η διαδικασία «Έλεγχος και Υπολογισμός» με τις αρχικές τιμές. Η ενεργοποίηση του κουμπιού από την αρχή χωρίς προηγούμενη εκτέλεση δεν προκαλεί κάποια ενέργεια.

Όλα τα αποτελέσματα καταγράφονται και σε αρχεία *Excel*, επομένως ο χρήστης πρέπει να είναι προσεκτικός με την αποθήκευσή τους, καθώς τα δεδομένα διαγράφονται αν η εφαρμογή κλείσει και επανεκκινηθεί χωρίς αποθήκευση.

Επιπλέον, σε ξεχωριστό αρχείο *Excel* καταγράφονται οι κανονικοποιημένες πιθανότητες που χρησιμοποιήθηκαν για τον υπολογισμό των πιθανοτήτων *B* κάθε κατηγορίας.

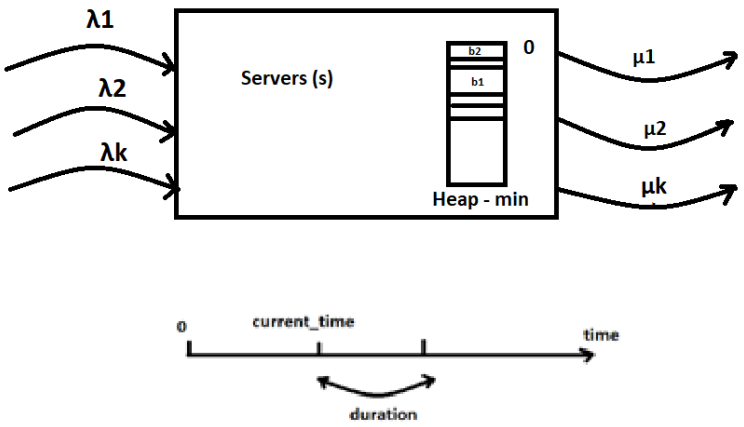
Η εφαρμογή παρέχει, επίσης, δυνατότητες καθαρισμού των δεδομένων:

- Το κουμπί «Καθαρισμός» διαγράφει όλα τα στοιχεία της τρέχουσας συνεδρίας.
- Το κουμπί «Καθαρισμός Δεδομένων στα Excel αρχεία» αφαιρεί τα περιεχόμενα των Excel αρχείων.

Τέλος, είναι πολύ σημαντικό να διευκρινίσουμε ότι σε όλες τις εφαρμογές των αναλυτικών μοντέλων δίνεται η δυνατότητα επιλογής και του κατάλληλου tr της πολιτικής BR. Αν δεν υπάρχει tr τότε η συμπλήρωση του μηδενός είναι υποχρεωτική.

4.4.2 Επεξήγηση προγράμματος της προσομοίωσης

Η προσομοίωση του μοντέλου *EMLM* ακολουθεί την ίδια βασική φιλοσοφία με την προσομοίωση του κλασικού μοντέλου *Erlang B*. Ωστόσο, λόγω της εμπλοκής πολλαπλών κατηγοριών κίνησης *K*, (Σχήμα 13) καθίσταται αναγκαία η παροχή μιας πιο αναλυτικής περιγραφής της λειτουργίας του προγράμματος, με στόχο την αποφυγή ενδεχόμενης σύγχυσης και τη διευκόλυνση της κατανόησης της μεθοδολογίας που εφαρμόζεται.



Σχήμα 13: Προσομοίωση *EMLM*

Έστω ότι έχουμε τα εξής:

- Ένα σύστημα με χωρητικότητα *C*. *π.χ. s* εξυπηρετητές, οι οποίοι αρχικά είναι όλοι ελεύθεροι,

- κλήσεις διαφορετικών κατηγοριών *K,*
- άφιξη κλήσεων (αφίξεις *Poisson*) με μέση τιμή ρυθμού άφιξης ανά κατηγορία λ_1 , λ_2 ... λ_K,
- οι κλήσεις εξυπηρετούνται με μέσο ρυθμό εξυπηρέτησης ανά κατηγορία μ_1 , μ_2... μ_K (με εκθετική κατανομή),
- η κλήση κάθε κατηγορίας δεσμεύει διαφορετικές μονάδες *b.u.* b_1 , b_2 ,...,b_K.

Η κεντρική ιδέα του αλγορίθμου υλοποιείται στη συνάρτηση **simulate_kaufman_roberts**, η οποία αποτελεί τον βασικό πυρήνα της προσομοίωσης. Τα κύρια ορίσματά της περιλαμβάνουν: Το πλήθος των κλήσεων προς προσομοίωση, τη συνολική χωρητικότητα του συστήματος *C*, το πλήθος των κατηγοριών κίνησης *K*, τους πίνακες των μέσων ρυθμών άφιξης λ, των χρόνων εξυπηρέτησης εξυπηρέτησης μ^{-1} καθώς και των απαιτούμενων μονάδων b.u.για κάθε κατηγορία. Επιπλέον, η συνάρτηση δέχεται ως παραμέτρους τις τιμές αρχικοποίησης *seeds* των λ, μ και την παράμετρο *transit*, η οποία καθορίζει το αρχικό διάστημα προσαρμογής της προσομοίωσης.

Στη συνάρτηση αυτή, αρχικά, πραγματοποιείται η αρχικοποίηση του πίνακα - λίστα *arrivals*, στον οποίο αποθηκεύονται οι χρόνοι των πρώτων αφίξεων κάθε κατηγορίας κίνησης. Οι χρόνοι αυτοί παράγονται σύμφωνα με την εκθετική κατανομή, μέσω της συνάρτησης *expovariate*. Η *expovariate* δέχεται, κάθε φορά, ως είσοδο τους ρυθμούς άφιξης ανά κατηγορία που έχει ορίσει ο χρήστης $(\lambda_1, \lambda_2, \ldots, \lambda_k)$ και επιστρέφει χρονικές στιγμές άφιξης. Με τον τρόπο αυτό, μέσω επαναληπτικής διαδικασίας, αρχικοποιείται πλήρως ο πίνακας *arrivals*.

Μετά την αρχικοποίηση του πίνακα – λίστα *arrivals*, αρχίζει να εκτελείται μία επανάληψη για πλήθος ίσο με τον αριθμό των κλήσεων που προσομοιώνονται. Σε κάθε επανάληψη, προσδιορίζεται αρχικά η επόμενη χρονική στιγμή γεγονότος, θέτοντας στη μεταβλητή *current_time* την ελάχιστη τιμή του πίνακα *arrivals*. Παράλληλα, στη μεταβλητή *Ki* καταχωρούμε τη θέση του μικρότερου στοιχείου, που αντιστοιχεί στην κατηγορία κίνησης από την οποία προέρχεται η επόμενη άφιξη.

Πριν συνεχίσει ο αλγόριθμος, εξετάζεται ο σωρός αν έχει κλήσεις που έχουν διεκπεραιωθεί, δηλαδή αν έχουν λήξει χρονικά. Έτσι, αν το στοιχείο που βρίσκεται στη μηδενική θέση (0) του σωρού έχει τιμή μικρότερη της μεταβλητής *current_time* κάνουμε *απώθηση* (*pop*) από

το σωρό και, φυσικά, αποδεσμεύεται οι αντίστοιχες μονάδες *b.u.* (έχει κρατηθεί η τιμή του και εξηγείται ο λόγος παρακάτω). Η διαδικασία αυτή γίνεται επαναληπτικά μέχρι να μην ισχύει η συνθήκη: *Η τιμή της μηδενικής θέσης να είναι μικρότερη του current_time*. Φυσικά, κάθε φορά εξετάζεται αν ο σωρός είναι άδειος.

Αμέσως μετά, ξέροντας τόσο την κατηγορία της κλήσης *Ki* όσο και τη χρονική στιγμή άφιξής της *current_time*, ελέγχεται αν υπάρχουν διαθέσιμες μονάδες *b.u.* για την εξυπηρέτησή της. Αν υπάρχουν, η κλήση γίνεται αποδεκτή, αυξάνοντας κατά μία μονάδα την αντίστοιχη θέση του πίνακα-λίστα *allowed_calls* (στη θέση *Ki*), ενώ τα απαιτούμενα *b.u.* δεσμεύονται.

Ύστερα, υπολογίζεται μέσω εκθετικής κατανομής η χρονική διάρκεια διεκπεραίωσης της κλήσης (*duration*), με το αντίστοιχο μ της κατηγορίας. Η χρονική στιγμή ολοκλήρωσης της εξυπηρέτησης, δηλαδή το άθροισμα *current_time* + *duration* εισάγεται στον σωρό με ώθηση (*push*). Παράλληλα, καταχωρείται μαζί και η τιμή των μονάδων *b.u.* που δεσμεύονται, ώστε κατά την επόμενη απώθηση *(pop)* να είναι δυνατή η ορθή αποδέσμευση των πόρων.

Στην περίπτωση που δεν υπάρχουν διαθέσιμες μονάδες *b.u.* προς παραχώρηση για την εξυπηρέτηση της εισερχόμενης κλήσης, η κλήση μπλοκάρεται. Έτσι, αυξάνεται κατά μία μονάδα η αντίστοιχη θέση του πίνακα-λίστα *blocked_calls*, η οποία αντιστοιχεί στην κατηγορία *Ki* της κλήσης.

Μετά την ολοκλήρωση των παραπάνω βημάτων, υπολογίζεται ο επόμενος χρόνος άφιξης (μέσω εκθετικής κατανομής) της συγκεκριμένης κατηγορίας *Ki*. Η νέα αυτή χρονική στιγμή ενημερώνει τον πίνακα-λίστα *arrivals* στη θέση *Ki*. Στη συνέχεια, η διαδικασία επαναλαμβάνεται, προσδιορίζοντας εκ νέου την ελάχιστη τιμή του πίνακα *arrivals* και την αντίστοιχη κατηγορία, μέχρι να ολοκληρωθεί η προσομοίωση για το σύνολο των κλήσεων.

Όταν ολοκληρωθεί η επεξεργασία όλων των κλήσεων υπολογίζεται η πιθανότητα *B* κάθε κατηγορίας ως εξής:

$$\text{Blocking}[i] = \text{bloc}ked\_calls[i] / (allowed\_calls[i] + blocked\_calls[i]), \ με\ i = 1,2,\ldots K \quad (4.16)$$

Όπου:

- *Blocking[]* είναι ο πίνακας απωλειών κλήσεων ανά κατηγορία,
- *blocked_calls[]* είναι ο πίνακας του πλήθους των μπλοκαρισμένων κλήσεων ανά κατηγορία
- *allowed_calls[i]* είναι ο πίνακας του πλήθους των επιτρεπόμενων κλήσεων

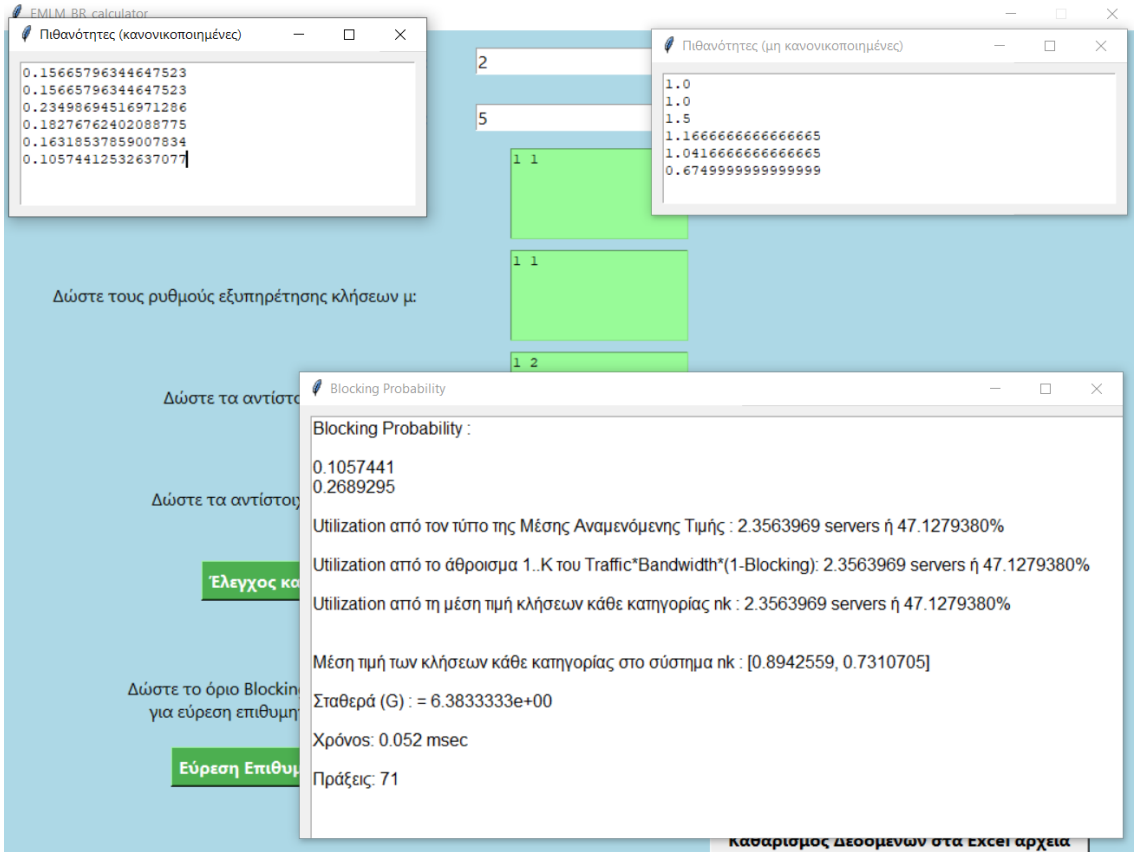
Το άθροισμα των πινάκων μας δίνει το πλήθος των κλήσεων ανά κατηγορία.

Τέλος, είναι πολύ σημαντικό να διευκρινίσουμε ότι σε όλες τις εφαρμογές της προσομοίωσης δίνεται η δυνατότητα επιλογής και του κατάλληλου tr της πολιτικής BR. Αν δεν υπάρχει tr τότε η συμπλήρωση του μηδενός είναι υποχρεωτική.

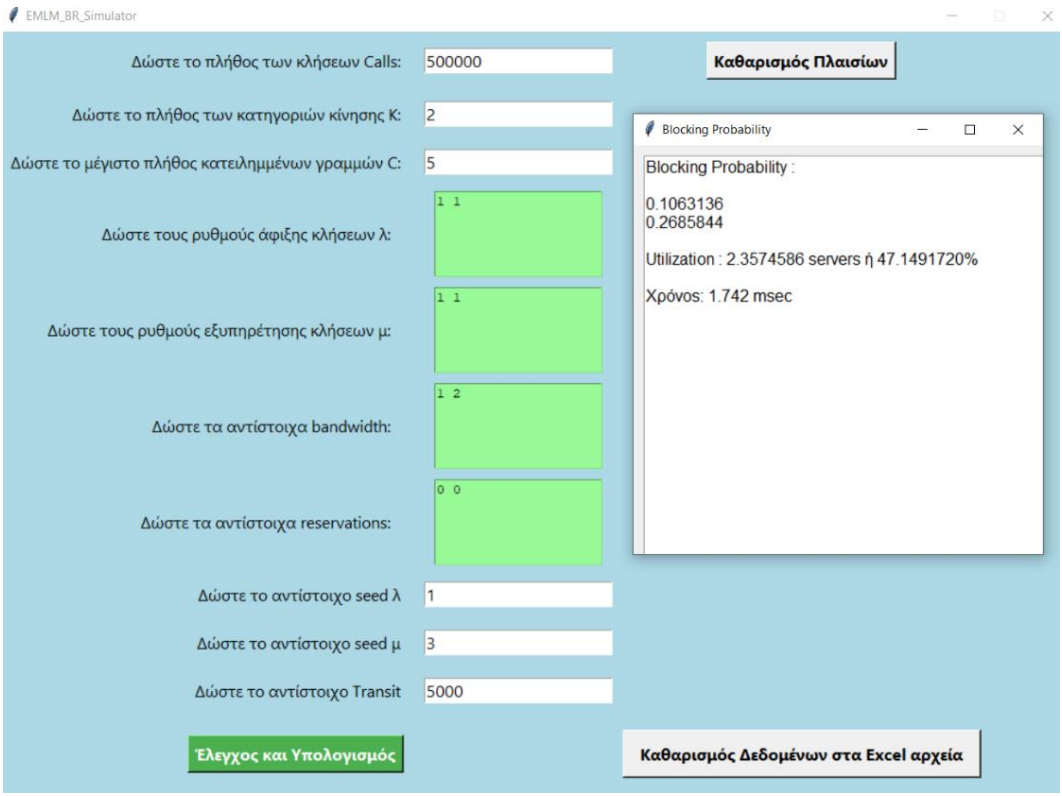
4.5 Αποτελέσματα παραδειγμάτων μέσω των εφαρμογών

Τα αποτελέσματα των Παραδειγμάτων 4.1, 4.2 και 4.3, όπως προκύπτουν από την εκτέλεση των κωδίκων του αναλυτικού μοντέλου και της προσομοίωσης, παρουσιάζονται στις Εικόνες 11 ως 16 . Από τη σύγκριση των αποτελεσμάτων παρατηρείται πολύ μικρή απόκλιση μεταξύ των δύο προσεγγίσεων, γεγονός που καταδεικνύει την υψηλή ακρίβεια του αναλυτικού μοντέλου.

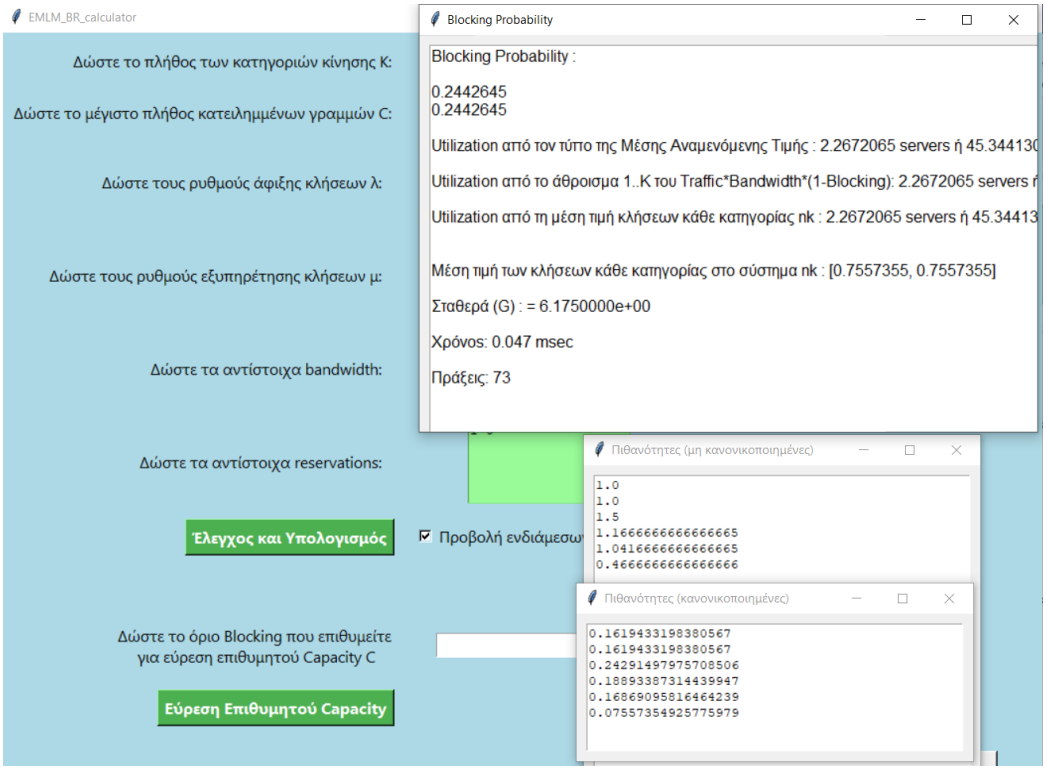
Συνεπώς, μπορεί να εξαχθεί το συμπέρασμα ότι ο αναδρομικός τύπος των *Kaufman–Roberts* προσεγγίζει με μεγάλη αξιοπιστία τη συμπεριφορά του πραγματικού συστήματος, καθώς οι αποκλίσεις που εμφανίζονται είναι αμελητέες.



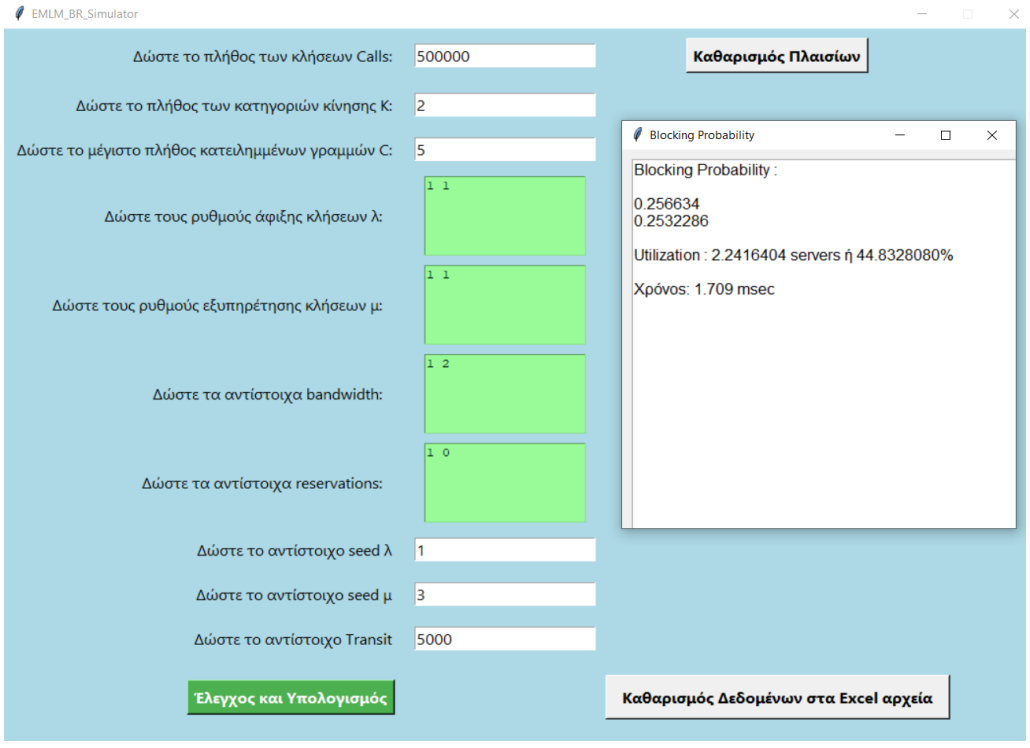
Εικόνα 11: Παράδειγμα 4.1 Αναλυτικό μοντέλο



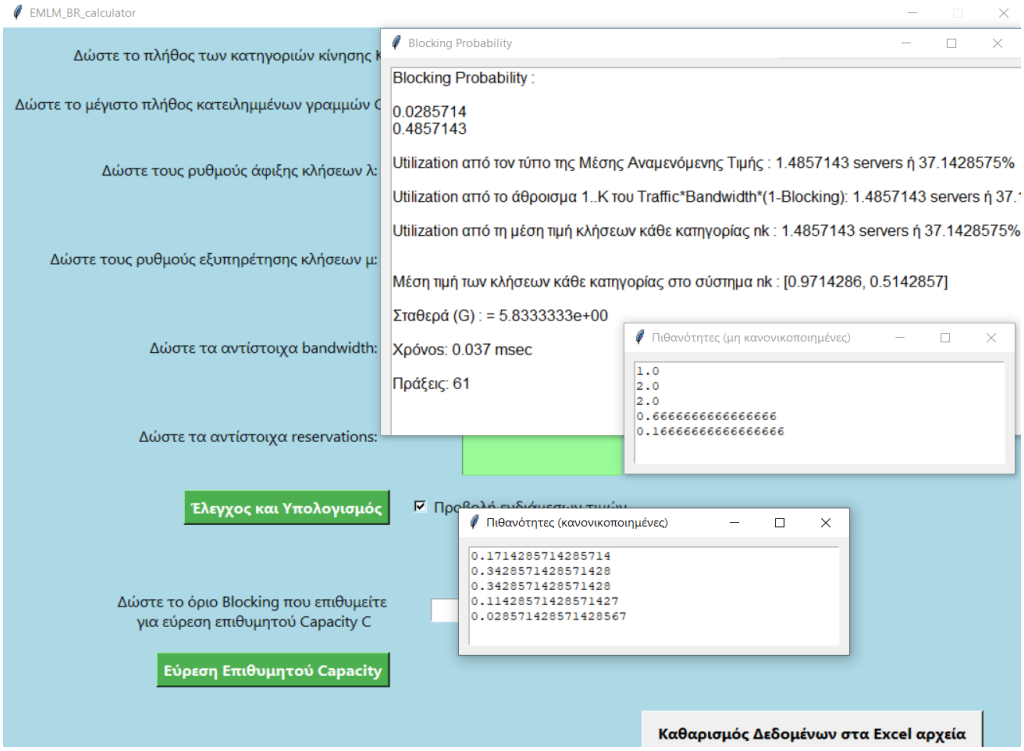
Εικόνα 12: Παράδειγμα 4.1 Προσομοίωση



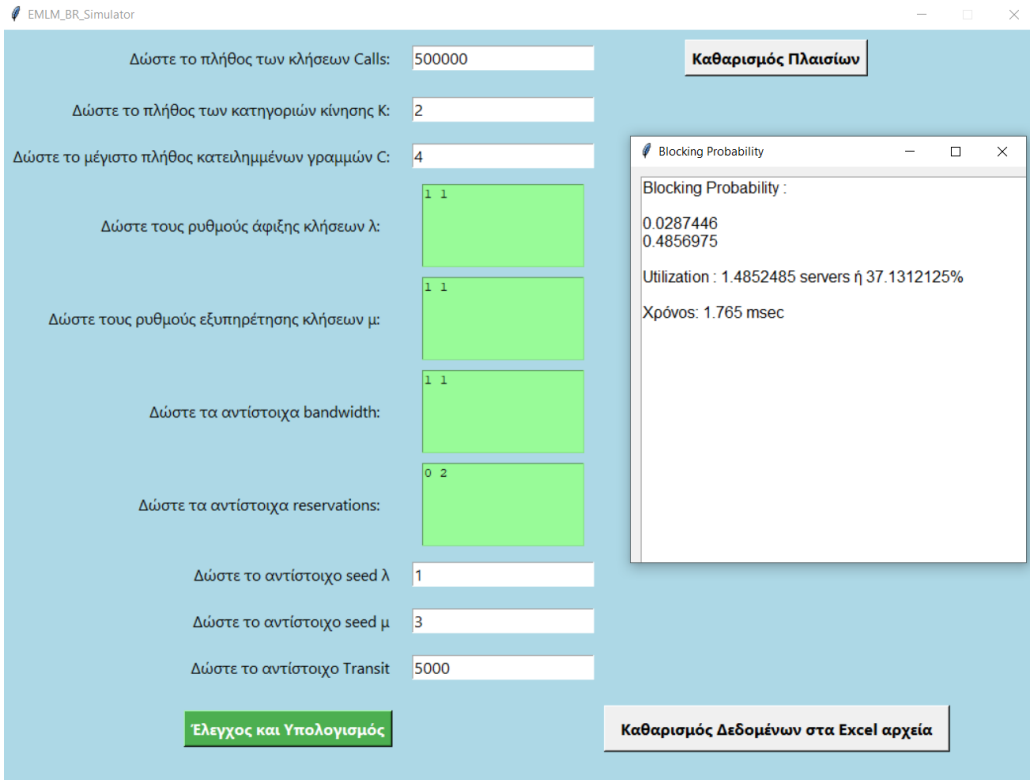
Εικόνα 13: Παράδειγμα 4.2 Αναλυτικό μοντέλο



Εικόνα 14: Παράδειγμα 4.2 Προσομοίωση



Εικόνα 15: Παράδειγμα 4.3 Αναλυτικό μοντέλο



Εικόνα 16: Παράδειγμα 4.3 Προσομοίωση

5. Μοντέλο μίας μόνο επανακλήσης *SRM*

5.1 Εισαγωγή

Στο μοντέλο μίας μόνο επανακλήσης *(Single Retry Model, SRM),* οι κλήσεις κάθε κατηγορίας επιχειρούν αρχικά να εισέλθουν στο σύστημα με μία προκαθορισμένη απαίτηση σε μονάδες εύρους ζώνης *b.u.*. Σε περίπτωση αποτυχίας, παρέχεται μία και μοναδική δυνατότητα επανάληψης της προσπάθειας με μειωμένη απαίτηση σε *b.u.*

Αν και η δεύτερη αυτή προσπάθεια δεν καταστεί επιτυχής, η κλήση απορρίπτεται (μπλοκάρεται). Επισημαίνεται ότι για κάθε κατηγορία κλήσεων επιτρέπεται αποκλειστικά μία επανακλήση, γεγονός που διαφοροποιεί το μοντέλο από μοντέλα πολλαπλών επανακλήσεων.

5.2 Το μοντέλο *SRM* υπό την πολιτική *CS*

Έστω ένα σύστημα με συνολική χωρητικότητα C *b.u.*, το οποίο εξυπηρετεί κλήσεις K διαφορετικών κατηγοριών $(k = 1, 2, 3 \ldots, K)$. Οι κλήσεις της κατηγορίας k καταφθάνουν στο σύστημα με μέσο ρυθμό άφιξης λ_k (αφίξεις *Poisson*) και απαιτούν b_k *b.u.*, ενώ ο χρόνος εξυπηρέτησής τους ακολουθεί εκθετική κατανομή με μέσο ρυθμό μ_k.

Κατά την άφιξη της κλήσης, εάν το σύστημα διαθέτει επαρκείς πόρους, της παραχωρεί τις απαιτούμενες μονάδες b.u.. Σε αντίθετη περίπτωση, παρέχεται η δυνατότητα άμεσης επανάληψης της προσπάθειας με μειωμένη απαίτηση σε b.u., σύμφωνα με τους κανόνες του μοντέλου επανακλήσης.

Βασικό χαρακτηριστικό του μοντέλου αποτελεί η διατήρηση σταθερού του λόγου μεταξύ των απαιτούμενων μονάδων εύρους ζώνης και του μέσου ρυθμού εξυπηρέτησης, δηλαδή [2]:

$$\frac{b_k}{\mu_\kappa} = \frac{b_{kr}}{\mu_{kr}} \tag{5.1}$$

ή ισοδύναμα, σε όρους προσφερόμενης κίνησης [2]:

$$\alpha_{\kappa r} = \frac{b_\kappa}{b_{\kappa r}}\, a_\kappa \tag{5.2}$$

Η παραπάνω συνθήκη εξασφαλίζει ότι η σχέση μεταξύ των δεσμευμένων πόρων και του χρόνου εξυπηρέτησης παραμένει σταθερή, ακόμη και κατά τις επανακλήσεις, διατηρώντας έτσι την αναλογικότητα του φορτίου στο σύστημα.

Αναλυτικό Μοντέλο

Όπως και στο μοντέλο EMLM, ο αντίστοιχος αναδρομικός τύπος του *Roberts* για το μοντέλο SRM είναι ο εξής [2]:

$$q(j) = \begin{cases} 1 & \alpha\nu\ j = 0 \\ \dfrac{1}{j}\displaystyle\sum_{k=1}^{K} \alpha_k b_k q(j - b_k) + \sum_{k=1}^{K} \alpha_{kr} b_{kr} \gamma_k(j) q(j - b_{kr}) & \alpha\nu\ 1 \le j \le C \\ 0 & \alpha\nu\ j < 0 \text{ ή } j > C \end{cases} \tag{5.3}$$

όπου :

$\gamma_k(j) = 1$ *όταν* $j > C - (b_k - b_{kr})$, και j το πλήθος των κατειλημμένων γραμμών.

$$a_{kr} = \frac{\lambda_{kr}}{\mu_{kr}} \tag{5.4}$$

α_κ *με βάση τον τύπο* (4.2).

Ο υπολογισμός της πιθανότητας απώλειας κλήσης κατηγορίας k, B_k με b_k $b.u.$ γίνεται μέσω του τύπου (4.2).

Ο τύπος πιθανότητας απώλειας κλήσης κατηγορίας k, B_{kr} με b_{kr} $b.u.$ είναι ο εξής [2]:

$$B_{kr} = \sum_{j=C-b_{kr}+1}^{C} G^{-1} q(j) \tag{5.5}$$

με σταθερά κανονικοποίησης σύμφωνα με τον τύπο (4.3)

Ο τύπος της δεσμευμένης πιθανότητας απώλειας της κλήσης κατηγορίας k, B^*_{kr} με b_{kr} $b.u.$ όταν $(j > C - b_{kr} \mid j > C - b_k)$ είναι ο εξής [2]:

$$B^*_{kr} = \frac{B_{kr}}{B_k} = \text{Prob}(j > C - b_{kr} \mid j > C - b_k) \tag{5.6}$$

Το U υπολογίζεται μέσω της σχέσης (4.6).

Παράδειγμα 5.1

Έστω ότι έχουμε 2 κατηγορίες κίνησης ($K = 2$) σε ένα σύστημα με χωρητικότητα $C = 5$ *b.u.*, με μέσους ρυθμούς άφιξης και για τις δύο κατηγορίες $\lambda_1 = \lambda_2 = 1$ και μέσους ρυθμούς εξυπηρέτησης $\mu_1 = \mu_2 = 1$ (άρα $\alpha_1 = \alpha_2 = 1$ *erl*) , τα αντίστοιχα *b.u.* $b_1 = 1$ *b.u. και* $b_2 = 3$ *b.u.* και τα ενδιάμεσα $(b_{kr}, \mu_{kr}^{-1}, a_{kr})$ της δεύτερης κατηγορίας είναι:

$(b_{2r},\ \mu_{2r}^{-1}, \alpha_{2r}) = (2, 1.5, 1.5)$

Από τον αναδρομικό τύπο έχουμε:

$$jq(j) = \alpha_1 b_1 q(j - b_1) + \alpha_2 b_2 q(j - b_2) + \alpha_{2r} b_{2r} \gamma_2(j) q(j - b_{2r})$$

όπου : $\gamma_2(j) = 1$ όταν $j > C - (b_2 - b_{2r})$ δηλαδή $j > 5 - (3 - 2)$ άρα $j > 4$,

$$\alpha_{2r} = \frac{b_2}{b_{2r}}\, a_2 = 1.5\ erl.$$

$q(0) = 1$

$1q(1) = 1q(0) + 0 = 1$ άρα $q(1) = 1$

$2q(2) = 1q(1) + 0 = 1$ άρα $q(2) = 0.5$

$3q(3) = 1q(2) + 3q(0) = 0.5 + 3 = 3.5$ άρα $q(3) = 1.16667$

$4q(4) = 1q(3) + 3q(1) = 1.16667 + 3 = 4.16667$ άρα $q(4) = 1.04167$

$5q(5) = 1q(4) + 3q(2) + 3q(3) = 1.04167 + 1.5 + 3.50001 = 6.04168$

$G = q(0) + q(1) + q(2) + q(3) + q(4) + q(5) = 5.91668$

$$B_1 = \frac{q(5)}{G} = 0.20423$$

$$B_2 = \frac{q(3)+q(4)+q(5)}{G} = 0.57745$$

$$B_{2r} = \frac{q(4) + q(5)}{G} = 0.38028$$

$$B_{2r}^{*} = \frac{B_{2r}}{B_2} = 0.65855$$

5.3 Το μοντέλο *SRM* υπό την πολιτική *BR*

Όπως και στο μοντέλο *EMLM/BR*, έτσι και στο μοντέλο *SRM/BR* προβλέπεται η εκ των προτέρων δέσμευση ενός μέρους των *b.u.* από συγκεκριμένες κατηγορίες κλήσεων, με σκοπό την εξυπηρέτηση άλλων κατηγοριών που έχουν αυξημένες απαιτήσεις ή προτεραιότητα. Η δέσμευση αυτή πραγματοποιείται εις βάρος της κατηγορίας από την οποία αφαιρούνται οι πόροι, ενώ κάθε κατηγορία μπορεί να διαθέτει διαφορετικό επίπεδο κράτησης σε b.u. ανάλογα με τις απαιτήσεις του συστήματος.

Αναλυτικό Μοντέλο

Ο υπολογισμός των πιθανοτήτων κατάστασης του συστήματος βασίζεται στον αναδρομικό τύπο του *Roberts*, ο οποίος δίνεται από τη σχέση [2]:

$$q(j) = \begin{cases} 1 & \alpha\nu\ j = 0 \\ \frac{1}{j}\sum_{k=1}^{K} \alpha_k D_k(j-b_k) q(j-b_k) + \sum_{k=1}^{K} \alpha_{kr} D_{kr}(j-b_{kr}) \gamma_k(j) q(j-b_{kr}) & \alpha\nu\ 1 \le j \le C \\ 0 & \alpha\nu\ j < 0 \text{ ή } j > C \end{cases} \quad (5.7)$$

όπου:

$\gamma_k(j) = 1$ *όταν* $j > C - (b_k - b_{kr})$, (5.8)

$D_k(j - b_k)$ από τον τύπο (4.9)

$D_{kr}(j - b_{kr}) = b_{kr}$, *(αν* $j \le C - tr_k$*)* (5.9)

a_k με βάση τον τύπο (4.2) και a_{kr} με βάση τον τύπο (5.4)

Ο υπολογισμός του B_k κατηγορίας *k με* b_k *b.u.* και κράτηση tr_k προκύπτει από τον τύπο (4.10)

Ο τύπος του B_{kr} κατηγορίας *k με* b_{kr} $b.u.$ είναι [2]:

$$B_{kr} = \sum_{j=C-b_{kr}-tr_k+1}^{C} G^{-1} q(j) \quad (5.10)$$

με σταθερά κανονικοποίησης *G,* βάση του τύπου (4.3).

Αντιστοίχως με τον τύπο (5.6) προκύπτει η δεσμευμένη πιθανότητα απώλειας κλήσεων κατηγορίας k, B_{kr}^{*}, λαμβάνοντας υπόψη τις δεσμεύσεις της πολιτικής BR [2].

Παράδειγμα 5.2

Συνεχίζουμε το παράδειγμα 5.1 προσθέτοντας $tr_1 = 2b.u.$ δηλαδή στην κατηγορία 1 έχουμε κράτηση 2 *b.u.*

Από τον αναδρομικό τύπο έχουμε:

$$jq(j) = \alpha_1 b_1 D_1(j - b_1)q(j - b_1) + \alpha_2 b_2 q(j - b_2) + \alpha_{2r} b_{2r} \gamma_2(j) q(j - b_{2r})$$

όπου:

$\gamma_2(j) = 1$, όταν $j > C - (b_2 - b_{2r})$ *δηλαδή* $j > 5 - (3 - 2)$ *άρα* $j > 4$

$$\alpha_{2r} = \frac{b_2}{b_{2r}} a_2 = 1.5\ erl$$

$D_1(j - b_1) = b_1$, όταν $j \leq C - tr_1$ *δηλαδή* $j \leq 3$

$q(0) = 1$

$1q(1) = 1q(0) = 1$ άρα $q(1) = 1$

$2q(2) = 1q(1) = 1$ άρα $q(2) = 0.5$

$3q(3) = q(2) + 3q(0) = 0.5 + 3 = 3.5$ άρα $q(3) = 1.16667$

$4q(4) = 0 + 3q(1) = 3$ άρα $q(4) = 0.75$

$5q(5) = 0 + 3q(2) + 3q(3) = 1.5 + 3.5 = 5$ άρα $q(5) = 1$

$G = q(0) + q(1) + q(2) + q(3) = 5.41666$

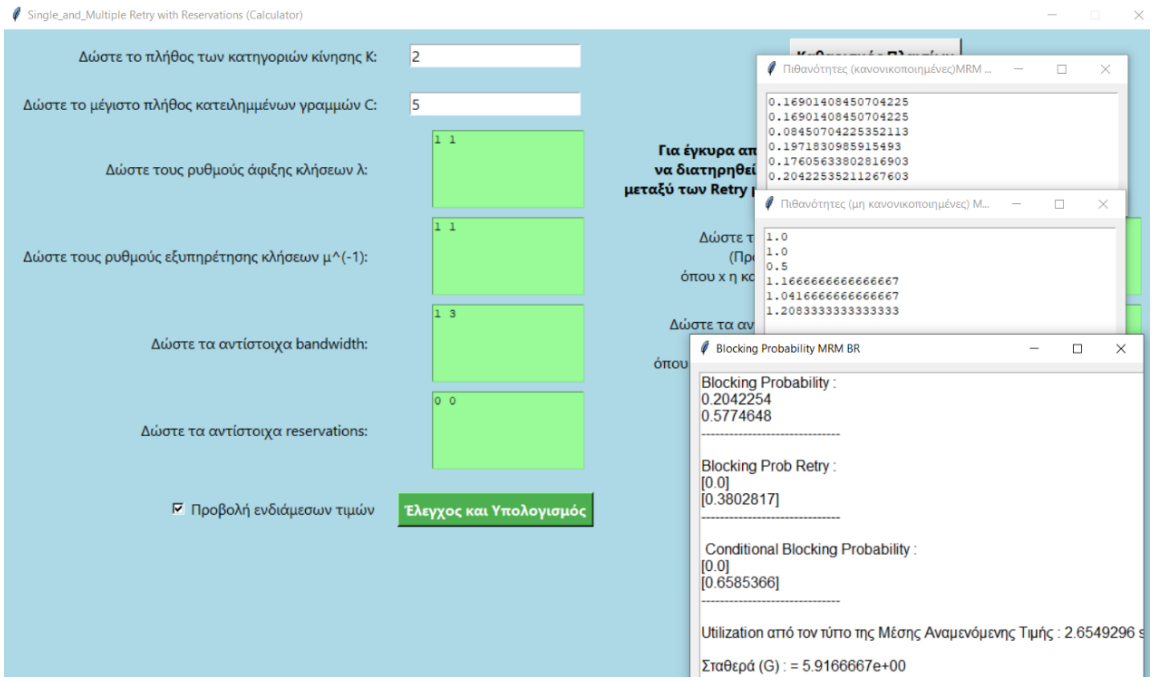
$$B_1 = \sum_{j=C-b_1-tr_1+1}^{C} G^{-1} q(j) = \frac{q(3) + q(4) + q(5)}{G} = 0.53846$$

$$B_2 = \sum_{j=C-b_2+1}^{C} G^{-1} q(j) = \frac{q(3) + q(4) + q(5)}{G} = 0.53846$$

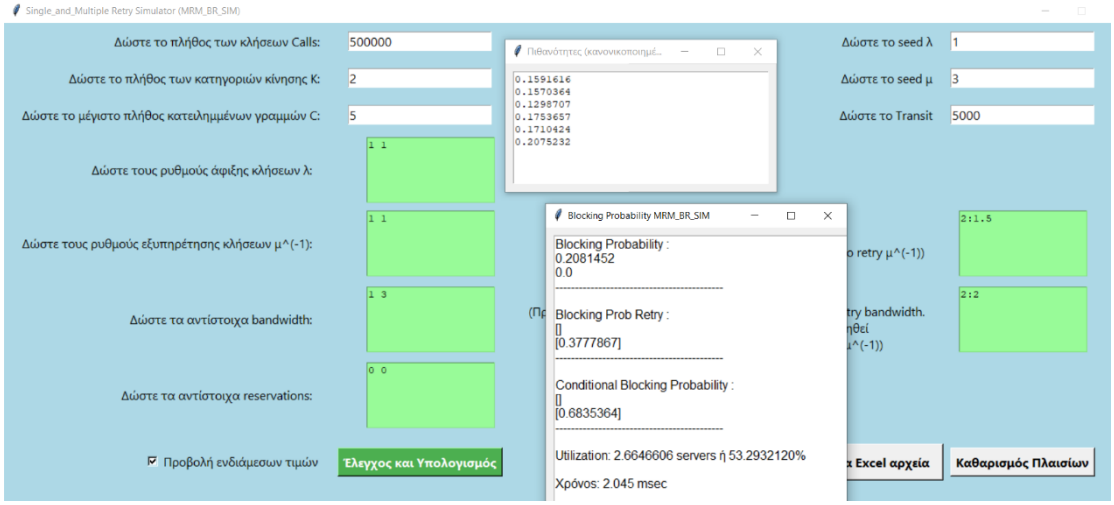
$$B_{2r} = \sum_{j=C-b_{2r}+1}^{C} G^{-1} q(j) = \frac{q(4)+q(5)}{G} = 0.32308$$

5.4 Αποτελέσματα παραδειγμάτων μέσω των εφαρμογών

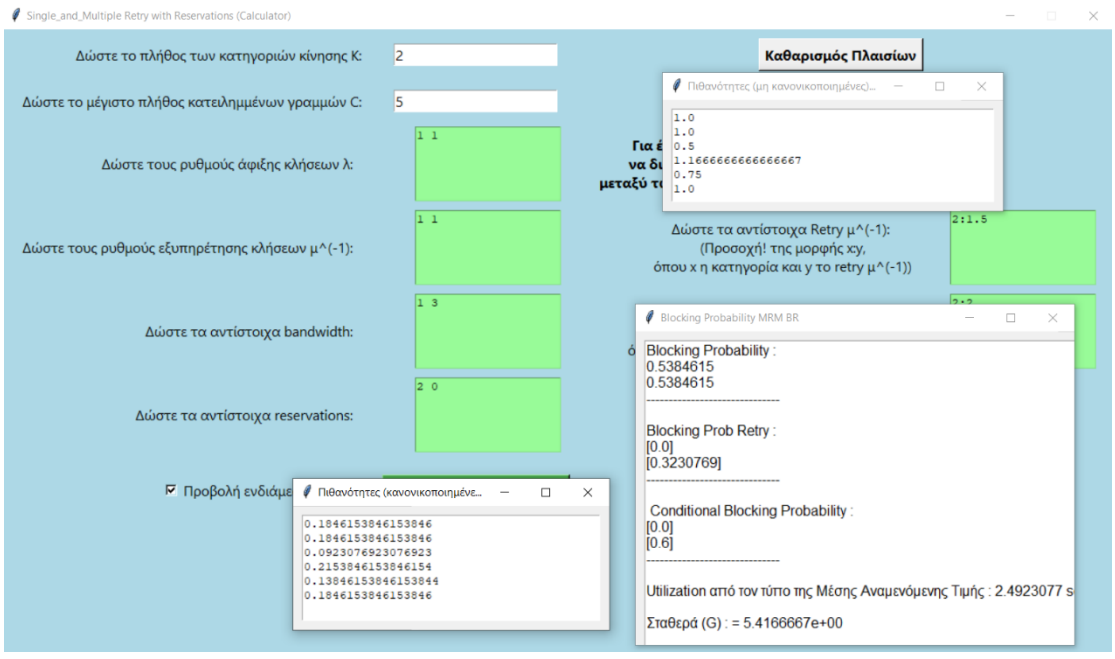
Τα αποτελέσματα των παραδειγμάτων 5.1 και 5.2 , όπως προκύπτουν από την εκτέλεση των κωδικών του αναλυτικού μοντέλου και της προσομοίωσης, παρουσιάζονται στις Εικόνες 17 ως 20. Οι κώδικες αυτοί έχουν συγχωνευθεί με τους αντίστοιχους του μοντέλου *MRM*, και αναλύονται στο Κεφάλαιο 6. Ακόμη, τα τμήματα κώδικα των αντίστοιχων προγραμμάτων, τόσο του αναλυτικού μοντέλου όσο και της προσομοίωσης, που αφορούν τα Κεφάλαια 5 και 6, παρατίθενται στο Παράρτημα Ε.



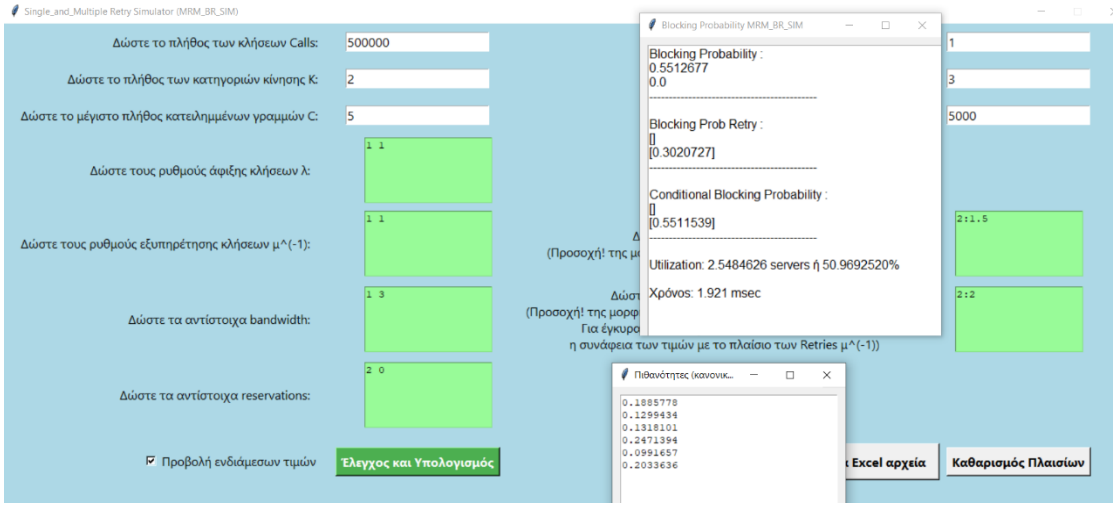
Εικόνα 17: Παράδειγμα 5.1 Αναλυτικό μοντέλο



Εικόνα 18: Παράδειγμα 5.1 Προσομοίωση



Εικόνα 19: Παράδειγμα 5.2 Αναλυτικό μοντέλο



Εικόνα 20: Παράδειγμα 5.2 Προσομοίωση

Παρατηρείται απόκλιση μεταξύ των αποτελεσμάτων που προκύπτουν από το αναλυτικό μοντέλο και εκείνων της προσομοίωσης, γεγονός που θεωρείται αναμενόμενο, καθώς ο αναδρομικός τύπος του *Roberts* αποτελεί προσεγγιστική μέθοδο και ενδέχεται να αποκλίνει από τις πραγματικές τιμές του συστήματος.

Παράδειγμα 5.3

Έστω ότι έχουμε ένα σύστημα με τέσσερις κατηγορίες κλήσεων ($K = 4$) και κλήσεις καταφθάνουν τυχαία (αφίξεις *Poisson*) σε ένα σύστημα με εύρος ζώνης $C = 500$ *b.u.* Τα χαρακτηριστικά των κλήσεων δίνονται με τη μορφή διανυσμάτων και μόνο η κατηγορία 4 έχει δικαίωμα να προσπαθήσει ξανά.:

$\lambda = (8,4,2,1)$, $\mu^{-1} = (12,10,8,6)$, $b = (1,7,14,28)$

$b_{4r} = 14$ και $\mu_{4r}^{-1} = 12$ ώστε $(b_4\mu_4^{-1} = b_{4r}\mu_{4r}^{-1})$

Αναλυτικό μοντέλο					
C	B_1	B_2	B_3	B_{4r}	B^*_{4r}
500	0.0478594	0.299651	0.5088651	0.5088651	0.6913733
1000	0.0004126	0.003062	0.0063392	0.0063392	0.4955091
Προσομοίωση					
C	B_1	B_2	B_3		
500	0.049029±0.00118	0.298579±0.00279	0.506339±0.00411		
	B_{4c_4}	$B^*_{4c_4}$			
	0.507502±0.00301	0.68309548±0.00432			
C	B_1	B_2	B_3		
1000	0.000412±0.00009	0.003023±0.00037	0.006105±0.00123		
	B_{4c_4}	$B^*_{4c_4}$			
	0.005850±0.00084	0.476586±0.04095			

Πίνακας 3: Αποτελέσματα παρ. 5.3 με CS

Με βάση τον Πίνακα 3 έχουμε τα ακόλουθα:

- Τα προγράμματα του αναλυτικού μοντέλου και της προσομοίωσης εκτελέστηκαν για διαφορετικές τιμές της χωρητικότητας *C*, οι οποίες κυμαίνονται σε υψηλά επίπεδα, με αρχική τιμή 500 *b.u.* και τελική τιμή 1000 *b.u.*
- Στην περίπτωση της προσομοίωσης, ο συνολικός αριθμός κλήσεων ανήλθε στις 500000, ενώ η περίοδος εξομάλυνσης (*transient phase*) ορίστηκε στις Transit = 5000 κλήσεις. Επιπλέον, τα *seeds* που χρησιμοποιήθηκαν για τις γεννήτριες τυχαίων αριθμών των (λ, μ) ήταν τα εξής: (2,4), (4,5), (5,6), (6,7), (7,8). Να τονίσουμε ότι οι συγκεκριμένες προδιαγραφές εφαρμόζονται για όλα τα παραδείγματα προσομοίωσης των μοντέλων που παρουσιάζονται στα επόμενα κεφάλαια.
- Όπως μπορούμε να διαπιστώσουμε, η αύξηση της χωρητικότητας του συστήματος *C* έχει ως αποτέλεσμα τη μείωση όλων των πιθανοτήτων απώλειας κλήσεων, γεγονός που συνάδει με τη θεωρητική συμπεριφορά των συστημάτων.
- Επίσης, οι τιμές του αναδρομικού τύπου του Roberts αποκλίνουν από εκείνες της προσομοίωσης.

Για εξισορρόπηση των πιθανοτήτων απόρριψης *B*, θεωρούμε παραμέτρους δέσμευσης εύρους ζώνης ανά κατηγορία με μορφή διανύσματος $tr = (13, 7, 0, 0)$, ώστε να ισχύει:

$b_1 + tr_1 = b_2 + tr_2 = b_3 + tr_3 = b_{4r_4} + tr_4 = 14$

Τα αντίστοιχα αποτελέσματα παρουσιάζονται στον Πίνακα 4.

Αναλυτικό μοντέλο				
C	**B1**	**B2**	**B_3**	**B_{4c_4}**
500	0.3804302	0.3804302	0.3804302	0.3804302
Προσομοίωση				
C	**B1**	**B2**	**B_3**	
500	0.383760±0.00408	0.384306±0.00397	0.382918±0.00449	
	B_{4c_4}			
	0.380165±0.00246			

Πίνακας 4: Αποτελέσματα παρ. 5.3 με BR

Παρατηρείται ότι τα αποτελέσματα του αναλυτικού μοντέλου και της προσομοίωσης συγκλίνουν σε μεγάλο βαθμό, όταν εφαρμόζονται παράμετροι δέσμευσης εύρους ζώνης της μορφής $tr = (13, 7, 0, 0)$, γεγονός που υποδηλώνει βελτιωμένη προσαρμογή του αναλυτικού μοντέλου στη συμπεριφορά του συστήματος υπό τις συγκεκριμένες συνθήκες.

6. Μοντέλο πολλαπλών επανακλήσεων *MRM*

6.1 Εισαγωγή

Στο κεφάλαιο αυτό θα ασχοληθούμε με το Μοντέλο Πολλαπλών Επανακλήσεων *(Multi Retry Model, MRM),* σύμφωνα με το μοντέλο αυτό οι κλήσεις διαφορετικών κατηγοριών επιχειρούν να εισέλθουν στο σύστημα με μία αρχική απαίτηση σε *b.u* ανά κατηγορία. Σε περίπτωση αποτυχίας εξυπηρέτησης, παρέχεται η δυνατότητα άμεσων επανακλήσεων, κατά τις οποίες η απαιτούμενη ποσότητα *b.u*. μειώνεται διαδοχικά ανά κατηγορία.

Ο μέγιστος αριθμός επανακλήσεων καθορίζεται εκ των προτέρων και δύναται να διαφέρει μεταξύ των κατηγοριών κλήσεων, ενώ δεν είναι απαραίτητο όλες οι κατηγορίες να διαθέτουν δυνατότητα επανακλήσεων. Οι απαιτήσεις σε b.u. ακολουθούν φθίνουσα πορεία ως προς τις τιμές τους σε κάθε διαδοχική προσπάθεια.

Σε περίπτωση που η κλήση δεν καταστεί δυνατό να εξυπηρετηθεί μετά την ολοκλήρωση όλων των επιτρεπόμενων επανακλήσεων αυτή απορρίπτεται (μπλοκάρεται) από το σύστημα.

6.2 Το μοντέλο *MRM* υπό την πολιτική *CS*

Έστω ένα σύστημα με χωρητικότητα *C b.u*. το οποίο εξυπηρετεί κλήσεις διαφορετικών κατηγοριών *K*, όπου $k = 1, 2, 3\ldots, K$. Κάθε κλήση κατηγορίας *k* καταφθάνει στο σύστημα σύμφωνα με διαδικασία *Poisson* με μέσο ρυθμό άφιξης λ_k και απαιτεί b_k *b.u*., ενώ ο μέσος χρόνος εξυπηρέτησης μ_k ακολουθεί εκθετική κατανομή.

Κατά την άφιξη μιας κλήσης, εφόσον το σύστημα διαθέτει τις απαιτούμενες b.u., η κλήση εξυπηρετείται άμεσα. Σε αντίθετη περίπτωση, παρέχεται η δυνατότητα άμεσης επανακλήσης, κατά την οποία η απαίτηση σε b.u. μειώνεται προοδευτικά.

Θεωρούμε ότι πλήθος προσπαθειών για την κατηγορία *k* είναι $s = 1, 2, 3,\ldots, S(k)$. Οι απαιτήσεις σε b.u. καθώς και οι αντίστοιχοι ρυθμοί εξυπηρέτησης ανά επανακλήση ικανοποιούν τις σχέσεις [2]:

$$b_k > b_{kr_1} > b_{kr_2} > \ldots. > b_{kr_s} > \ldots. > b_{kr_{s(k)}} \qquad (6.1)$$

$$\mu_k > \mu_{kr_1} > \mu_{kr_2} > \ldots. > \mu_{kr_s} > \ldots. > \mu_{kr_{s(k)}} \quad (6.2)$$

δηλαδή, τόσο οι απαιτήσεις σε b.u. όσο και οι ρυθμοί μ μειώνονται διαδοχικά σε κάθε επανακλήση.

Μια κλήση κατηγορίας k γίνεται αποδεκτή κατά την s-οστή επανακλήση με (b_{kr_s}, μ_{kr_s}), εφόσον ικανοποιείται η ακόλουθη ανισότητα [2]:

$$C - b_{kr_{s-1}} \leq j \leq C - b_{kr_s}, \quad (6.3)$$

όπου j είναι το πλήθος των κατειλημμένων b.u. του συστήματος.

Θεωρούμε επίσης ότι για την αρχική προσπάθεια ισχύουν [2]:

$$b_{kr_0} = b_k \text{ και } \mu_{kr_0} = \mu_k \quad (6.4)$$

Αναλυτικό Μοντέλο

Το αναλυτικό μοντέλο για τον υπολογισμό των πιθανοτήτων κατάστασης $q(j)$ βασίζεται στον αναδρομικό τύπο του *Roberts*, ο οποίος προσαρμόζεται κατάλληλα για την περίπτωση του μοντέλου *MRM* και χρησιμοποιείται για την αποδοτική εκτίμηση της κατανομής πιθανότητας των καταστάσεων του συστήματος και είναι ο εξής [2]:

$$q(j) = \begin{cases} 1 & \text{αν } j = 0 \\ \frac{1}{j} \sum_{k=1}^{K} \alpha_k b_k q(j - b_k) + \sum_{k=1}^{K} \sum_{s=1}^{S(k)} \alpha_{kr_s} b_{kr_s} \gamma_{k_s}(j) q(j - b_{kr_s}) & \text{αν } 1 \leq j \leq C \\ 0 & \text{αν } j < 0 \text{ ή } j > C \end{cases} \quad (6.5)$$

Όπου:

$\gamma_k(j) = 1$ *όταν* $j > C - (b_{kr_{s-1}} - b_{kr_s})$, (6.6)

$$a_{kr_s} = \frac{\lambda_k}{\mu_{kr_s}} \quad (6.7)$$

και α_k με βάση τον τύπο (4.2)

Ο υπολογισμός της πιθανότητας απώλειας κλήσης B_k κατηγορίας k με b_k *b.u.* γίνεται μέσω του τύπου (4.5).

Ο τύπος της πιθανότητας απώλειας κλήσης $B_{kr_{s(k)}}$ κατηγορίας k με $b_{kr_{s(k)}}$ $b.u.$ είναι ο εξής [2]:

$$B_{kr_{s(k)}} = \sum_{j=C-b_{kr_{s(k)}}+1}^{C} G^{-1}q(j) \tag{6.8}$$

με σταθερά κανονικοποίησης G που υπολογίζεται από τον τύπο (4.3).

Ο τύπος της δεσμευμένης πιθανότητας απώλειας κλήσης $B^*_{kr_{s(k)}}$κατηγορίας k με $b_{kr_{s(k)}}$ $b.u.$ όταν ισχύει $(j > C - b_{kr_{s(k)}} \,|j > C - b_k)$ είναι ο εξής [2]:

$$B^*_{kr_{s(k)}} = Prob\left(j > C - b_{kr_{s(k)}} \,\middle|\, j > C - b_k\right) = \frac{B_{kr_{s(k)}}}{B_k} \tag{6.9}$$

Το U υπολογίζεται με βάση τον τύπο (4.6).

Παράδειγμα 6.1

Έστω ότι έχουμε 2 κατηγορίες κίνησης (K = 2) σε ένα σύστημα με χωρητικότητα C = 3 *b.u.*, με μέσους ρυθμούς άφιξης και για τις δύο κατηγορίες $\lambda_1 = \lambda_2 = 1$ και μέσους ρυθμούς εξυπηρέτησης $\mu_1 = \mu_2 = 1$ (άρα $\alpha_1 = \alpha_2 = 1$ *erl*) και τα $b_1 = 1$ *b.u.*, $b_2 = 3$ *b.u.*.
Τα αντίστοιχα ενδιάμεσα (b, μ^{-1}, α) ανά κατηγορία και προσπάθεια είναι:
$(b_{2r_1},\ \mu_{2r_1}^{-1}, \alpha_{2r_1}) = (2, 2, 2)$ $\quad$ $(b_{2r_2},\ \mu_{2r_2}^{-1}, \alpha_{2r_2}) = (1, 4, 4)$

Από τον αναδρομικό τύπο έχουμε:

$$jq(j) = \alpha_1 b_1 q(j - b_1) + \alpha_2 b_2 q(j - b_2) + \alpha_{2r_1} b_{2r_1} \gamma_{2_1}(j) q(j - b_{2r_1}) + \alpha_{2r_2} b_{2r_2} \gamma_{2_2}(j) q(j - b_{2r_2})$$

όπου : $\gamma_{2_1}(j) = 1$ όταν $j >$ C – $(b_{2r_0} - b_{2r_1})$ δηλαδή $j > 3 - (3 - 2)$ άρα j > 2

$\gamma_{2_2}(j) = 1$ όταν $j >$ C – $(b_{2r_1} - b_{2r_2})$ δηλαδή $j > 3 - (2 - 1)$ άρα j > 2

$q(0) = 1$
$1q(1) = 1q(0) + 0$ άρα $q(1) = 1$

$2q(2) = 1q(1) + 0$ άρα $q(2) = 0.5$

$3q(3) = 1q(2) + 3q(0) + 4q(1) + 4q(2) = 0.5 + 3 + 4 + 2 = 9.5$ άρα $q(3) = 3.16666$

$G = 5.66666$

$B_1 = \frac{q(3)}{G} = 0.55882$ και

$B_2 = \frac{q(1)+q(2)+q(3)}{G} = 0.82353$

$$B_{2r_1} = \frac{q(2) + q(3)}{G} = 0.64705$$

$$B_{2r_2} = \frac{q(3)}{G} = 0.55882$$

Παράδειγμα 6.2

Έστω ότι έχουμε 2 κατηγορίες κίνησης ($K = 2$) σε ένα σύστημα με χωρητικότητα $C = 8$ *b.u.*, με μέσους ρυθμούς άφιξης και για τις δύο κατηγορίες $\lambda_1 = \lambda_2 = 1$ και μέσους ρυθμούς εξυπηρέτησης $\mu_1 = \mu_2 = 1$ (άρα $\alpha_1 = \alpha_2 = 1$ *erl*) και τα b_1=3 *b.u.*, $b_2 = 4$ *b.u.*.

Τα αντίστοιχα ενδιάμεσα (b, μ^{-1}, α) ανά κατηγορία και κατώφλι είναι:

$(b_{1r_1}, \mu_{1r_1}^{-1}, \alpha_{1r_1}) = (2, 1.5, 1.5)$ $\quad$ $(b_{1r_2}, \mu_{1r_2}^{-1}, \alpha_{1r_2}) = (1, 3, 3)$

$(b_{2r_1}, \mu_{2r_1}^{-1}, \alpha_{2r_1}) = (3, 4/3, 4/3)$ $\quad$ $(b_{2r_2}, \mu_{2r_2}^{-1}, \alpha_{2r_2}) = (2, 2, 2)$

Από τον αναδρομικό τύπο έχουμε:

$$jq(j) = \alpha_1 b_1 q(j - b_1) + \alpha_2 b_2 q(j - b_2) + \alpha_{1r_1} b_{1r_1} \gamma_{1_1}(j) q(j - b_{1r_1}) +$$
$$\alpha_{1r_2} b_{1r_2} \gamma_{1_2}(j) q(j - b_{1r_2}) + \alpha_{2r_1} b_{2r_1} \gamma_{2_1}(j) q(j - b_{2r_1}) + \alpha_{2r_2} b_{2r_2} \gamma_{2_2}(j) q(j - b_{2r_2})$$

όπου : $\gamma_{1_1}(j) = 1$ όταν $j >$ C - ($b_1 - b_{1r_1}$) δηλαδή $j > 8 – (3 - 2)$ => j > 7

$\gamma_{1_2}(j) = 1$ όταν $j >$ C – (b_{1r_1} - $b_{1r_{21}}$) δηλαδή $j > 8 – (2 - 1)$ => j > 7

$\gamma_{2_1}(j) = 1$ όταν $j >$ C - ($b_2 - b_{2r_1}$) δηλαδή $j > 8 – (4 - 3)$ => j > 7

$\gamma_{2_2}(j) = 1$ όταν $j >$ C - ($b_{2r_1} - b_{2r_2}$) δηλαδή $j > 8 – (3 - 2)$ => j > 7

$q(0) = 1$

$q(1) = 0$

$q(2) = 0$

$3q(3) = 3q(0) + 0 + 0 + 0 + 0 + 0$ άρα $q(3) = 1$

$4q(4) = 3q(1) + 4q(0) + 0 + 0 + 0 + 0 = 4$ άρα $q(4) = 1$

$5q(5) = 3q(2) + 4q(1) = 0$ άρα $q(5) = 0$

$6q(6) = 3q(3) + 4q(2) = 3$ άρα $q(6) = 0.5$

$7q(7) = 3q(4) + 4q(3) = 7$ άρα $q(7) = 1$

$8q(8) = 3q(5) + 4q(4) + 3q(6) + 3q(7) + 4q(5) + 4q(6) = 4 + 1.5 + 3 + 2 = 10.5$ άρα $q(8) = 10.5/8$

$G = 5.8125$

$$B_{1r_2} = \sum_{j=C-b_{1r_2}+1}^{C} G^{-1}q(j) = \frac{q(8)}{G} = 0.225806$$

$$B_{2r_2} = \sum_{j=C-b_{2r_2}+1}^{C} G^{-1}q(j) = \frac{q(7)+q(8)}{G} = 0.3978495$$

6.3 Το μοντέλο *MRM* υπό την πολιτική *BR*

Στο μοντέλο *MRM/BR* προβλέπεται η εκ των προτέρων δέσμευση ενός αριθμού *b.u.* σε μία ή περισσότερες κατηγορίες κλήσεων, με σκοπό την εξυπηρέτηση άλλων κατηγοριών που έχουν αυξημένες απαιτήσεις ή προτεραιότητα. Η δέσμευση αυτή πραγματοποιείται εις βάρος των κατηγοριών στις οποίες ανήκουν οι πόροι, περιορίζοντας τη διαθέσιμη χωρητικότητά τους προς όφελος των υπολοίπων.

Κάθε κατηγορία κλήσεων δύναται να διαθέτει διαφορετικό επίπεδο δέσμευσης πόρων, επιτρέποντας έτσι την ευέλικτη διαχείριση των διαθέσιμων b.u. και την επίτευξη επιθυμητών χαρακτηριστικών απόδοσης του συστήματος.

Αναλυτικό Μοντέλο

Ο αναδρομικός τύπος του *Roberts* προσαρμόζεται ως εξής [2]:

$$q(j) = \begin{cases} 1 & \alpha\nu\ j = 0 \\ \frac{1}{j}\sum_{k=1}^{K} \alpha_k D_k(j-b_k)q(j-b_k) + \sum_{k=1}^{K}\sum_{s=1}^{S(k)} a_{kr_s} D_{kr_s}(j-b_{kr_s})\gamma_{k_s}(j)q(j-b_{kr_s}) & \alpha\nu\ 1 \le j \le C \\ 0 & \alpha\nu\ j < 0 \text{ ή } j > C \end{cases} \quad (6.10)$$

όπου:

$\gamma_k(j)$, από τη σχέση (6.6)

$D_k(j - b_k)$, από τη σχέση (4.9)

$D_{kr_s}(j - b_{kr_s}) = b_{kr_s}$, αν j ≤ C - tr_k (6.11)

a_{kr_s} από τον τύπο (6.7) και α από τον τύπο (4.2)

Ο υπολογισμός της πιθανότητας απώλειας κλήσης κατηγορίας k με b_k *b.u.* γίνεται από τον τύπο (4.10).

Ο τύπος της πιθανότητας απώλειας κλήσης κατηγορίας k με $b_{kr_{s(k)}}$ $b.u.$ είναι ο εξής [2]:

$$B_{kr_{s(k)}} = \sum_{j=C-b_{kr_{s(k)}}-tr_k+1}^{C} G^{-1}q(j) \quad (6.12)$$

με σταθερά κανονικοποίησης G που υπολογίζεται με βάση τον τύπο (4.3).

Ο τύπος της δεσμευμένης πιθανότητας απώλειας κλήσεων κατηγορίας k, $B^*_{kr_{s(k)}}$ με $b_{kr_{s(k)}}$ $b.u.$ είναι ο ίδιος με τον τύπο (6.9), λαμβάνοντας υπόψη τις δεσμεύσεις της πολιτικής BR [2].

Παράδειγμα 6.3

Συνεχίζουμε το παράδειγμα 5.1 προσθέτοντας $tr_1 = 2$ *b.u.* δηλαδή στην κατηγορία 1 έχουμε κράτηση *2 b.u.*

Από τον αναδρομικό τύπο έχουμε:

$$jq(j) = \alpha_1 D_1(j - b_1)q(j - b_1) + \alpha_2 b_2 q(j - b_2) + \alpha_{2r_1} b_{2r_1} \gamma_{2_1}(j) q(j - b_{2r_1}) + \alpha_{2r_2} b_{2r_2} \gamma_{2_2}(j) q(j - b_{2r_2})$$

όπου : $\gamma_{2_1}(j) = 1$ όταν $j >$ C – $(b_{2r_0} - b_{2r_1})$, δηλαδή $j > 3 - (3 - 2)$ άρα $j > 2 \Rightarrow j = 3$

$\gamma_{2_2}(j) = 1$ όταν $j >$ C – $(b_{2r_1} - b_{2r_2})$, δηλαδή $j > 3 - (2 - 1)$ άρα $j > 2 \Rightarrow j = 3$

$D_1(j - b_1) = b_1$ όταν $j \leq C - tr_1$, δηλαδή $j \leq 1$

$q(0) = 1$

$1q(1) = 1q(0) = 1$ άρα $q(1) = 1$

$q(2) = 0$ (εξαιτίας της κράτησης)

$3q(3) = 0 + 3q(0) + 4q(1) + 4q(2) = 3 + 4 + 0 = 7$ άρα $q(3) = 3/7$

$G = q(0) + q(1) + q(2) + q(3) = 4.33333$

$$B_1 = \sum_{j=C-b_1-tr_1+1}^{C} G^{-1}q(j) = \frac{q(1)+q(2)+q(3)}{G} = 0.76923$$

$$B_2 = \sum_{j=C-b_2+1}^{C} G^{-1}q(j) = \frac{q(1)+q(2)+q(3)}{G} = 0.76923$$

$$B_{2r_1} = \sum_{j=C-b_{2r_1}+1}^{C} G^{-1}q(j) = \frac{q(2)+q(3)}{G} = 0.53846$$

$$B_{2r_2} = \sum_{j=C-b_{2r_2}+1}^{C} G^{-1}q(j) = \frac{q(3)}{G} = 0.53846$$

Παράδειγμα 6.4

Συνεχίζοντας το Παράδειγμα 5.3, θεωρούμε τιμή κράτησης $tr_1 = 1$ *b.u.* , δηλαδή στην κατηγορία 1 έχουμε δέσμευση 1 *b.u.* Επιλέγουμε $tr_1 = 1$ *b.u.* ώστε να ικανοποιείται η σχέση: $b_{1r_2} + tr_1 = b_{2r_2}$, με στόχο να επιτύχουμε εξισορρόπηση της απόδοσης του συστήματος μεταξύ των κατηγοριών κλήσεων.

Από τον αναδρομικό τύπο έχουμε:

$$\begin{aligned} jq(j) = {} & \alpha_1 D_1(j-b_1)q(j-b_1) + \\ & \alpha_2 b_2 q(j-b_2) + \\ & \alpha_{1r_1} D_{1r_1}(j-b_{1r_1})\gamma_{1_1}(j)q(j-b_{1r_1}) + \\ & \alpha_{1r_2} D_{1r_2}(j-b_{1r_2})\gamma_{1_2}(j)q(j-b_{1r_2}) + \\ & \alpha_{2r_1} b_{2r_1}\gamma_{2_1}(j)q(j-b_{2r_1}) + \\ & \alpha_{2r_2} b_{2r_2}\gamma_{2_2}(j)q(j-b_{2r_2}) \end{aligned}$$

όπου : $\gamma_{1_1}(j) = 1$, όταν $j >$ C - ($b_1 - b_{1r_1}$), δηλαδή $j > 8 - (3 - 2) \Rightarrow j > 7$

$\gamma_{1_2}(j) = 1$, όταν $j >$ C – (b_{1r_1} - $b_{1r_{21}}$), δηλαδή $j > 8 - (2 - 1) \Rightarrow j > 7$

$\gamma_{2_1}(j) = 1$, όταν $j >$ C - ($b_2 - b_{2r_1}$), δηλαδή $j > 8 - (4 - 3) \Rightarrow j > 7$

$\gamma_{2_2}(j) = 1$, όταν $j >$ C - ($b_{2r_1} - b_{2r_2}$), δηλαδή $j > 8 - (3 - 2) \Rightarrow j > 7$

$D_1(j - b_1) = b_1$, όταν $j \le C - tr_1$, δηλαδή $j \le 7$

$D_{1r_1}(j - b_{1r_1}) = b_{1r_1}$, όταν $j \le C - tr_1$, δηλαδή $j \le 7$

$D_{1r_2}(j - b_{1r_2}) = b_{1r_2}$, όταν $j \le C - tr_1$, δηλαδή $j \le 7$

$q(0) = 1$

$q(1) = 0$

$q(2) = 0$

$3q(3) = 3q(0) + 0 + 0 + 0 + 0 + 0$ άρα $q(3) = 1$

$4q(4) = 3q(1) + 4q(0) + 0 + 0 + 0 + 0 = 4$ άρα $q(4) = 1$

$5q(5) = 3q(2) + 4q(1) = 0$ άρα $q(5) = 0$

$6q(6) = 3q(3) + 4q(2) = 3$ άρα $q(6) = 0.5$

$7q(7) = 3q(4) + 4q(3) = 7$ άρα $q(7) = 1$

$8q(8) = 0 + 4q(4) + 0 + 0 + 4q(5) + 4q(6) = 6$ άρα $q(8) = 6/8$

$G = 5.25$

$$B_{1r_2} = \sum_{j=C-b_{1r_2}-tr_1+1}^{C} G^{-1}q(j) = \frac{q(7)+q(8)}{G} = 0.33333$$

$$B_{2r_2} = \sum_{j=C-b_{2r_2}+1}^{C} G^{-1}q(j) = \frac{q(7)+q(8)}{G} = 0.33333$$

$$B^*_{1r_2} = \frac{B_{1r_2}}{Prob\ (j>C-b_1)} = 0.77777$$

$$B^*_{2r_2} = \frac{B_{2r_2}}{Prob\ (j>C-b_2)} = 0.77777$$

$U = 4.38095$

6.4 Επεξήγηση των προγραμμάτων των *SRM/MRM*

Τα σχετικά τμήματα κώδικα των αντίστοιχων προγραμμάτων παρατίθενται στο Παράρτημα Ε.

6.4.1 Επεξήγηση του προγράμματος του αναλυτικού μοντέλου

Όπως και στο αντίστοιχο πρόγραμμα του μοντέλου *EMLM*, έτσι και στην παρούσα περίπτωση εφαρμόζεται το αναλυτικό μοντέλο μέσω του αναδρομικού τύπου του *Roberts* σε γλώσσα προγραμματισμού *Python*.

Όσον αφορά το γραφικό περιβάλλον της εφαρμογής αξίζει να επισημανθεί η εξής ιδιαιτερότητα: Οι τιμές των b.u καθώς και οι αντίστοιχοι χρόνοι εξυπηρέτησης (μ^{-1}) εισάγονται από τον χρήστη με τη μορφή ζευγών *x:y*, όπου *x* αντιστοιχεί στην κατηγορία κλήσης και *y* στη σχετική τιμή. Στην περίπτωση που δοθούν πολλαπλά ζεύγη με το ίδιο *x* και διαφορετικές τιμές *y*, αυτό υποδηλώνει ότι η συγκεκριμένη κατηγορία διαθέτει διαφορετικές τιμές ανά επανακλήση.

Για παράδειγμα έστω ότι έχουμε τρεις κατηγορίες κίνησης (K = 1, 2, 3) και συνολική χωρητικότητα $C = 12$ *b.u.*. Οι αρχικές απαιτήσεις σε *b.u.* είναι (b = 2, 5, 3) ανά κατηγορία, ενώ οι αντίστοιχοι χρόνοι εξυπηρέτησης είναι (μ^{-1} = 1, 2, 3) χρονικές μονάδες.

Ο χρήστης καλείται να εισάγει τις τιμές των b και μ για κάθε επανακλήση και για κάθε κατηγορία κλήσεων. Οι τιμές αυτές δίνονται με τη μορφή ζευγών *x:y* ως εξής:

- Για τα b έχουμε: (2:4 2:2 3:2 3:1)
- Για τα μ^{-1} έχουμε: (2:3 2:4 3:4 3:5)

Οι παραπάνω τιμές υποδηλώνουν ότι κατά την πρώτη επανακλήση η κατηγορία 2 απαιτεί 4 *b.u.* με χρόνο εξυπηρέτησης 3 χρονικές μονάδες, ενώ η κατηγορία 3 απαιτεί 2 *b.u.* με χρόνο εξυπηρέτησης 4 χρονικές μονάδες. Αντίστοιχα, κατά τη δεύτερη επανακλήση, η κατηγορία 2 απαιτεί 2 b.u. με χρόνο εξυπηρέτησης 4 χρονικές μονάδες, ενώ η κατηγορία 3 απαιτεί 1 b.u. με χρόνο εξυπηρέτησης 5 χρονικές μονάδες.

Σημειώνεται ότι η πρώτη κατηγορία δεν διαθέτει επανακλήσεις και, συνεπώς, συμπεριφέρεται, κατά προσέγγιση, όπως στο μοντέλο *EMLM*.

Το πρόγραμμα πραγματοποιεί αυτόματη ταξινόμηση των δεδομένων εισόδου σε περίπτωση που αυτά δεν έχουν δοθεί με τη σωστή σειρά από τον χρήστη. Συγκεκριμένα, οι τιμές b ταξινομούνται κατά φθίνουσα σειρά, ενώ οι αντίστοιχοι χρόνοι μ^{-1} ταξινομούνται κατά αύξουσα σειρά.

Οι αρχικές τιμές των b και μ, καθώς και τα λοιπά δεδομένα εισόδου, εισάγονται από τον χρήστη στα αντίστοιχα πεδία της εφαρμογής, χωρίς να απαιτείται ιδιαίτερη εξοικείωση, καθώς το περιβάλλον είναι σχεδιασμένο ώστε να διευκολύνει τη διαδικασία εισαγωγής δεδομένων.

6.4.2 Επεξήγηση του προγράμματος της προσομοίωσης

Ακολουθώντας την ίδια λογική, με τη λογική της προσομοίωσης του μοντέλου *EMLM*, οι κλήσεις εισέρχονται στο σύστημα και οι χρόνοι άφιξής τους καταγράφονται. Στη συνέχεια, επαναληπτικά, για κάθε κλήση εξετάζεται το πλήθος των δεσμευμένων κλήσεων με τις ακόλουθες διαδικασίες:

Αρχικά, ελέγχεται ο σωρός για τις κλήσεις που έχουν ολοκληρώσει τον χρόνο εξυπηρέτησής τους, ώστε να αποδεσμευτούν οι αντίστοιχες b.u. Στη συνέχεια, ελέγχεται αν υπάρχουν διαθέσιμες b.u. για την τρέχουσα κλήση. Αν οι απαιτούμενες μονάδες είναι διαθέσιμες, αυτές παραχωρούνται στην κλήση για την εξυπηρέτησή της, με χρόνο εξυπηρέτησης που καθορίζεται από την παράμετρο μ^{-1}. Ο χρόνος εξυπηρέτησης της κλήσης, μαζί με τις δεσμευμένες μονάδες εύρους ζώνης, καταχωρούνται στον σωρό.

Σε περίπτωση που οι απαιτούμενες μονάδες εύρους ζώνης δεν είναι διαθέσιμες, η κλήση, εφόσον ανήκει σε κατηγορία που επιτρέπεται επανακλήση, έχει τη δυνατότητα άμεσης νέας προσπάθειας με μειωμένο αριθμό μονάδων. Το πλήθος των επιτρεπόμενων επανακλήσεων καθορίζεται από τον χρήστη της εφαρμογής. Εάν κατά τη διάρκεια κάποιας επανακλήσης υπάρχουν διαθέσιμες b.u., η κλήση εξυπηρετείται με αυτές, σε χρόνο εξυπηρέτησης αντίστοιχο της συγκεκριμένης προσπάθειας, ο οποίος είναι αυξανόμενος σε σχέση με τους χρόνους των προηγούμενων επανακλήσεων, της ίδιας κατηγορίας. Και σ' αυτήν την περίπτωση, τα στοιχεία καταχωρούνται στο σωρό με βάση τον χρόνο εξυπηρέτησης της συγκεκριμένης επανακλήσης.

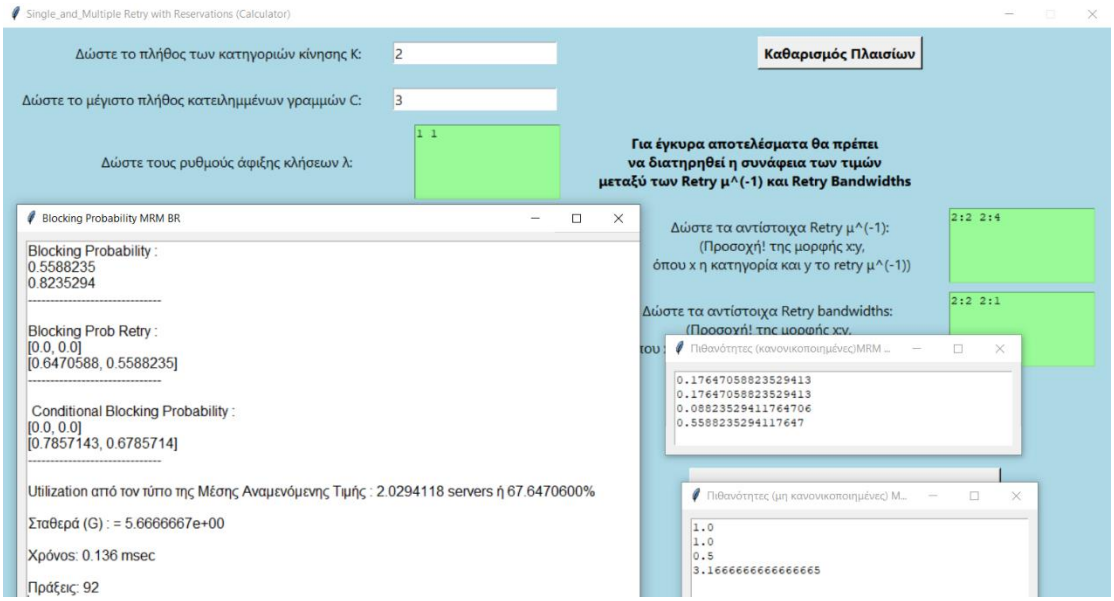
Αν μετά την ολοκλήρωση όλων των επανακλήσεων, η κλήση δεν εξυπηρετηθεί, αυτή απορρίπτεται (μπλοκάρεται). Στο πλαίσιο του μοντέλου *SRM*, προβλέπεται μόνο μία επανακλήση ανά κλήση.

Για κάθε επανακλήση καταγράφεται επίσης μία μετρική που υπολογίζει την πιθανότητα απώλειας κλήσεων ανά κατηγορία, επιτρέποντας την ποσοτικοποίηση της απόδοσης του συστήματος.

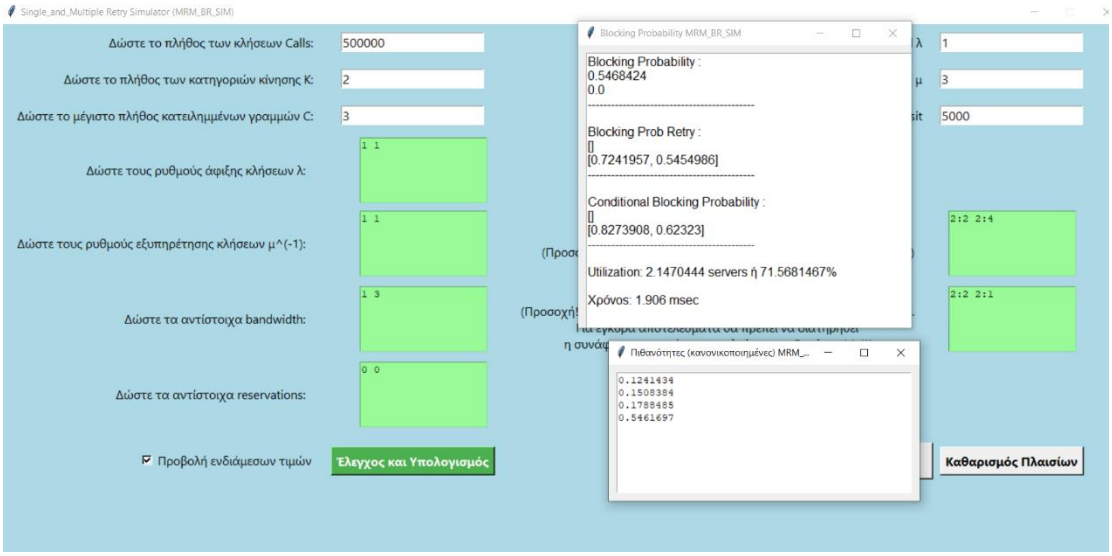
Όσον αφορά το γραφικό περιβάλλον της εφαρμογής, ισχύουν οι ίδιες ιδιαιτερότητες και περιορισμοί που περιγράφονται στην Ενότητα 6.4.1.

6.5 Αποτελέσματα παραδειγμάτων μέσω των εφαρμογών

Τα αποτελέσματα των παραδειγμάτων 6.1, 6.2, 6.3 και 6.4 , όπως προκύπτουν από την εκτέλεση των κωδικών του αναλυτικού μοντέλου και της προσομοίωσης, παρουσιάζονται παρακάτω στις Εικόνες 21 και 22, καθώς και στον Πίνακα 5. Τα δεδομένα αυτά παρέχουν συγκριτική εικόνα της συμπεριφοράς του συστήματος και επιτρέπουν την αξιολόγηση της ακρίβειας της προσομοίωσης σε σχέση με το αναλυτικό μοντέλο.



Εικόνα 21: Παράδειγμα 6.1 Αναλυτικό μοντέλο



Εικόνα 22: Παράδειγμα 6.1 Προσομοίωση

Κατά τον ίδιο τρόπο εξάγονται και τα αποτελέσματα για τα υπόλοιπα παραδείγματα, τα οποία παρουσιάζονται στον Πίνακα 5.

Αναλυτικό μοντέλο	B_1	B_{2r_2}	$B^*_{2r_2}$
Παράδειγμα 6.1	0.5588235	0.5588235	0.6785714
Παράδειγμα 6.3 (με tr_1=2)	0.7692308	0.5384615	0.7

Προσομοίωση	B_1	B_{2r_2}	$B^*_{2r_2}$
Παράδειγμα 6.1	0.546887±0.00139	0.545364±0.00049	0.622533±0.00067
Παράδειγμα 6.3 (με tr_1=2)	0.819518±0.00112	0.514455±0.00027	

Αναλυτικό μοντέλο	B_{1r_1}	B_{2r_2}	$B^*_{1r_2}$	$B^*_{2r_2}$
Παράδειγμα 6.2	0.2258065	0.3978495	0.4666667	0.8222222
Παράδειγμα 6.4 (με tr_1 =1)	0.3333333	0.3333333	0.7777778	0.7777778

Προσομοίωση	B_{1r_2}	B_{2r_2}
Παράδειγμα 6.2	0.256309±0.00163	0.367513±0.00062
	$B^*_{1r_2}$	$B^*_{2r_2}$
	0.611905±0.00191	0.716333±0.00101
	B_{1r_2}	B_{2r_2}
Παράδειγμα 6.4 (με tr_1 =1)	0.359473±0.00166	0.359244±0.00085

Πίνακας 5: Αποτελέσματα παρ. 6.1, 6.2, 6.3, 6.4

Παρατηρείται διαφορά μεταξύ των αποτελεσμάτων που προκύπτουν από το αναλυτικό μοντέλο και εκείνων της προσομοίωσης. Η απόκλιση αυτή είναι αναμενόμενη, δεδομένου ότι ο αναδρομικός τύπος ενσωματώνει προσεγγιστικές παραδοχές και, κατά συνέπεια, αποκλίνει από τις πραγματικές τιμές που υπολογίζονται μέσω της προσομοίωσης.

Παράδειγμα 6.5

Έστω ότι έχουμε ένα σύστημα με τέσσερις κατηγορίες κλήσεων (K=4) και κλήσεις καταφθάνουν τυχαία (αφίξεις *Poisson*) σε ένα σύστημα με εύρος ζώνης C=300 *b.u.* Τα χαρακτηριστικά των κλήσεων δίνονται με τη μορφή διανυσμάτων:

$\lambda = (8,6,4,1)$, $\mu^{-1} = (6,4,3,1)$, $b = (1,4,6,24)$

$b_{4r} = (20,16,12,8)$ $\mu_{4r}^{-1} = (1.2, 1.5, 2, 3)$

Τα αποτελέσματα του Παραδείγματος 6.5, όπως προέκυψαν από την εκτέλεση των αντίστοιχων προγραμμάτων, τόσο του αναλυτικού μοντέλου όσο και της προσομοίωσης, παρουσιάζονται στον Πίνακα 6.

Αναλυτικό μοντέλο					
C	B_1	B_2	B_3	B_{4c_4}	$B^*_{4c_4}$
300	0.0048874	0.0202147	0.0273192	0.0349248	0.3007285
310	0.0032829	0.0136701	0.018486	0.0236855	0.2901117
320	0.0021349	0.0089462	0.0121055	0.0155436	0.2803132
330	0.001344	0.0056661	0.0076718	0.0098709	0.2712437
340	0.0008195	0.0034746	0.0047075	0.0060687	0.2628254
350	0.0004843	0.0020645	0.0027989	0.0036151	0.2549906
360	0.0002777	0.0011898	0.0016141	0.0020886	0.2476803
Προσομοίωση					
C	B_1	B_2	B_3	B_{4c_4}	$B^*_{4c_4}$
300	0.0052354	0.0201166	0.0304357	0.0388428	0.3371124
310	0.0032985	0.013953	0.0195768	0.0257031	0.3146754
320	0.0022725	0.0091974	0.0132016	0.0170586	0.3094077
330	0.0013952	0.0059012	0.0081514	0.0119487	0.3154158
340	0.0009013	0.0041027	0.0053382	0.0068388	0.2676692
350	0.0005034	0.0019585	0.0033988	0.0048794	0.322335
360	0.0002972	0.0012801	0.0018914	0.0023436	0.2663755

Πίνακας 6: Αποτελέσματα παρ. 6.5

- Τα προγράμματα του αναλυτικού μοντέλου και της προσομοίωσης εκτελέστηκαν με διαφορετικές τιμές του *C* του συστήματος. Οι τιμές αυτές ήταν υψηλές, με αρχική τιμή 300 *b.u*. και τελική 360 *b.u*, με αύξηση του *C* ανά 10 *b.u.*.
- Η προσομοίωση εκτελέστηκε με τιμές *seed* λ = 1 και *seed* μ = 3. Δεν χρησιμοποιήθηκαν περισσότερες τιμές για τα *seeds*, καθώς το συγκεκριμένο παράδειγμα επικεντρώνεται στην επίδραση της αύξησης της χωρητικότητας *C* στις απώλειες κλήσεων *B*.
- Όπως μπορούμε να διαπιστώσουμε , η αύξηση των γραμμών του συστήματος *C* οδηγεί σε μείωση όλων των *B*. Παράλληλα, οι τιμές που προκύπτουν από τον αναδρομικού τύπου του *Roberts* εμφανίζουν απόκλιση σε σχέση με εκείνες της προσομοίωσης.

Για την εξισορρόπηση των πιθανοτήτων απώλειας κλήσης, θεωρήθηκαν κρατήσεις με τη μορφή διανύσματος $tr = (7, 4, 2, 0)$, έτσι ώστε να ισχύει:

$$b_1 + tr_1 = b_2 + tr_2 = b_3 + tr_3 = b_{4r_4} + tr_4 = 8$$

Τα αντίστοιχα αποτελέσματα παρουσιάζονται στον παρακάτω πίνακα.

Αναλυτικό μοντέλο				
C	**B1**	**B2**	**B_3**	**B_{4c_4}**
300	0.0237857	0.0237857	0.0237857	0.0237857

Προσομοίωση			
C	**B1**	**B2**	**B_3**
300	0.028439±0.00109	0.028151±0.00067	0.027992±0.00096
	B_{4c_4}		
	0.027822±0.00081		

Πίνακας 7: Αποτελέσματα παρ. 6.5 με *tr* = (7,4,2,0)

7. Μοντέλο μονού κατωφλίου *STM*

7.1 Εισαγωγή

Στο μοντέλο μονού Κατωφλίου **(*Single Threshold Model, STM*)**, εξετάζονται κλήσεις διαφόρων κατηγοριών, οι οποίες κοινώς υπόκεινται σε ένα μόνο κατώφλι. Το κατώφλι αυτό εκφράζεται σε μονάδες εύρους ζώνης (b.u.) και καθορίζει τη μεταβολή της λειτουργίας του συστήματος. Συγκεκριμένα, όταν το πλήθος των κατειλημμένων b.u. υπερβαίνει το κατώφλι, οι κατηγορίες κλήσεων υποχρεώνονται να μειώσουν τις απαιτήσεις τους σε b.u., ώστε να καταστεί δυνατή η εξυπηρέτηση των αιτήσεών τους. Η μείωση αυτή των b.u. πραγματοποιείται ανεξάρτητα για κάθε κατηγορία κλήσεων.

7.2 Το μοντέλο *STM* υπό την πολιτική *CS*

Στο μοντέλο μονού κατωφλίου *STM* θεωρούμε ένα μοναδικό κατώφλι J_0, το οποίο ισχύει για όλες τις κατηγορίες κλήσεων. Συγκεκριμένα, έστω ότι εισέρχεται μια κλήση κατηγορίας k με μέσο ρυθμό άφιξης λ_k, κατά *Poisson*, η οποία απαιτεί b_k *b.u.* και έχει μέσο ρυθμό εξυπηρέτησης μ_k (ακολουθώντας εκθετική κατανομή). Αν το σύστημα διαθέτει συνολική χωρητικότητα C *b.u.* και ο αριθμός των κατειλημμένων γραμμών είναι j, τότε:

- Αν $j \leq J_0$, το σύστημα διαθέτει επαρκές εύρος ζώνης και παραχωρεί τις ζητούμενες b.u. για την εξυπηρέτηση της κλήσης.
- Αν $j > J_0$, οι απαιτήσεις σε *b.u.* μειώνονται σε b_{kc} και ο αντίστοιχος μέσος ρυθμός εξυπηρέτησης προσαρμόζεται σε μ_{kc}, ώστε η κλήση να εξυπηρετηθεί με τροποποιημένους πόρους.
- Εάν όλες οι γραμμές είναι κατειλημμένες ($j = C$), η κλήση απορρίπτεται.

Επιπλέον, η σχέση μεταξύ των νέων μονάδων b.u. και του προσαρμοσμένου χρόνου εξυπηρέτησης πρέπει να διατηρεί την ισορροπία πόρων, δηλαδή [2]:

$$\mu_{kc}^{-1} b_{kc} = \mu_k^{-1} b_k \quad \text{με} \quad b_{kc} < b_k \ , \ \mu_{kc} < \mu_k \tag{7.1}$$

Η παραπάνω σχέση εξασφαλίζει ότι η μείωση του εύρους ζώνης συνοδεύεται από αντίστοιχη προσαρμογή του χρόνου εξυπηρέτησης, ώστε να διατηρείται η αναλογικότητα μεταξύ πόρων και χρόνου.

Έστω ότι εξετάζουμε ένα σύστημα με κατηγορίες κλήσεων K $(k = 1,2,3…K)$, μέσους ρυθμοί άφιξης κλήσεων $\lambda = (\lambda_1, \lambda_2, \lambda_3, …., \lambda_k)$, με μέσους ρυθμούς εξυπηρέτησης $\mu = (\mu_1, \mu_2, \mu_3, …, \mu_k)$ και *b.u.* $b = (b_1, b_2, b_3, …., b_k)$.

Η κλήση κατηγορίας k μπορεί να εξυπηρετηθεί με τις πλήρεις τιμές (b_k, μ_k), αν ο αριθμός των κατειλημμένων γραμμών j βρίσκεται κάτω από το κατώφλι J_0, δηλαδή [2]:

$$j \leq J_0 \,, \tag{7.2}$$

Αν η τιμή των κατειλημμένων γραμμών j βρίσκεται πάνω από την τιμή J_0 και κάτω από το συνολικό όριο του συστήματος C, δηλαδή [2]:

$$J_0 < \mathrm{j} < C \,, \tag{7.3}$$

τότε η κλήση εξυπηρετείται με προσαρμοσμένες τιμές (b_{kc}, μ_{kc}), μειωμένες σε σχέση με τις αρχικές, ώστε να διατηρείται η ισορροπία μεταξύ εύρους ζώνης και χρόνου εξυπηρέτησης.

Στην περίπτωση που η κλήση προσεγγίζει το άνω φράγμα C, πρέπει να διασφαλίζεται ότι η παροχή των b.u. δεν υπερβαίνει τη χωρητικότητα του συστήματος [2]:

$$j + b_{kc} <= C \tag{7.4}$$

Σε περίπτωση υπέρβασης, η κλήση απορρίπτεται.

Αναλυτικό Μοντέλο

Το αναλυτικό μοντέλο για τον υπολογισμό των $q(j)$, όπως και στις προηγούμενες περιπτώσεις στηρίζεται στον αναδρομικό τύπο του *Roberts* ο οποίος, για το μοντέλο *STM*, είναι [2]:

$$q(j) = \begin{cases} 1 & \alpha\nu\ j = 0 \\ \frac{1}{j}\sum_{k=1}^{K} \alpha_k b_k \delta_k(j) q(j-b_k) + \sum_{k=1}^{K} \alpha_{kc} b_{kc} \delta_{kc}(j) q(j-b_{kc}) & \alpha\nu\ 1 \le j \le C \\ 0 & \alpha\nu\ j < 0 \text{ ή } j > C \end{cases} \quad (7.5)$$

όπου:

$\delta_k(j) = 1, \quad \alpha\nu\ 1 \le j \le J_0 + b_k \text{ και } b_{kc} > 0 \text{ ή } \alpha\nu\ 1 \le j \le C \text{ και } b_{kc} = 0$ (7.6)

$\delta_{kc}(j) = 1, \quad \alpha\nu\ j > J_0 + b_{kc} \text{ και } b_{kc} > 0$ (7.7)

$a_{kc} = \frac{\lambda_k}{\mu_{kc}}$ (7.8)

α_k υπολογίζεται με βάση τον τύπο (4.2)

Ο υπολογισμός του B_k κατηγορίας k με b_k $b.u.$ γίνεται μέσω του τύπου (4.5)

Ο τύπος για την πιθανότητα απώλειας κλήσης B_{kc} κατηγορίας k με b_{kc} $b.u.$ είναι [2]:

$$B_{kc} = \sum_{j=C-b_{kc}+1}^{C} G^{-1} q(j) \quad (7.9)$$

και σταθερά κανονικοποίησης G, με βάση τον τύπο (4.3)

Ο τύπος της δεσμευμένης πιθανότητα απώλειας κλήσης κατηγορίας k με b_{kc} $b.u.$ όταν $j > J_0$ είναι [2]:

$$B_{kc}^{*} = Prob(j > C - b_{kc} \mid j > J_0) = \frac{B_{kc}}{Prob(j > J_0)} = \frac{\sum_{j=C-b_{kc}+1}^{C} q(j)}{\sum_{j=J_0+1}^{C} q(j)} \quad (7.10)$$

Η διαφοροποίηση της δεσμευμένης πιθανότητα με την πιθανότητα απώλειας κλήσης έγκειται στο γεγονός ότι, στον παρονομαστή δεν λαμβάνονται υπόψη όλες οι καταστάσεις $q(j)$, αλλά μόνο εκείνες για τις οποίες το j υπερβαίνει το κατώφλι J_0 .

Ο υπολογισμός του U βασίζεται στον τύπο (4.6)

Παράδειγμα 7.1

Έστω ότι έχουμε 2 κατηγορίες κίνησης (K = 2) σε ένα σύστημα με χωρητικότητα C = 5 b.u. , με μέσους ρυθμούς άφιξης και για τις δύο κατηγορίες $\lambda_1 = \lambda_2 = 1$, με μέσους ρυθμούς

εξυπηρέτησης $\mu_1 = \mu_2 = 1$ (άρα $\alpha_1 = \alpha_2 = 1$ *erl*) και $b_1 = 1$ *b.u.*, $b_2 = 3$ *b.u.* και το κατώφλι $J_0 = 1$.

Πάνω από το κατώφλι τα αντίστοιχα *(b, μ^{-1}, α)* στη δεύτερη κατηγορία είναι:

$(b_{2c}, \mu_{2c}^{-1}, \alpha_{2c}) = (2, 1.5, 1.5)$, με $\alpha_{2c} = \frac{b_2}{b_{2c}}\ \alpha_2 = 1.5\ erl$

Από τον αναδρομικό τύπο έχουμε:

$$jq(j) = \alpha_1 b_1 \delta_1(j) q(j - b_1) + \alpha_2 b_2 \delta_2(j) q(j - b_2) + \alpha_{2c} b_{2c} \delta_{2c}(j) q(j - b_{2c})$$

όπου : $\delta_1(j) = 1$ γιατί $b_{1c} = 0$ και $1 \le j \le 5$

$\delta_2(j) = 1$ όταν $j \le J_0 + b_2$ δηλαδή $j \le 4$ και $b_{2c} > 0$ (εννοείται $j \ge 1$)

$\delta_{2c}(j) = 1$ όταν $C \ge j > J_0 + b_{2c}$ δηλαδή $5 \ge j > 3$

Υπενθυμίζουμε ότι ανώτατο όριο είναι το *C*.

$q(0) = 1$

$1q(1) = 1q(0) + 0$ άρα $q(1) = 1$

$2q(2) = 1q(1) + 0$ άρα $q(2) = 0.5$

$3q(3) = 1q(2) + 3q(0) = 3.5$ άρα $q(3) = 1.16666$

$4q(4) = 1q(3) + 3q(1) + 3q(2) = 1.16666 + 3 + 1.5 = 5.66667$ άρα $q(4) = 1.416667$

$5q(5) = 1q(4) + 0 + 3q(3) = 1.416667 + 3.5 = 4.916667$ άρα $q(5) = 0.98333$

$G = q(0) + q(1) + q(2) + q(3) + q(4) + q(5) = 6.066667$

$$B_2 = \frac{q(5)}{G} = 0.162087$$

$$B_{2c} = \frac{q(4) + q(5)}{G} = 0.3956$$

$$B_{2c}^{*} = \frac{q(4) + q(5)}{q(2) + q(3) + q(4) + q(5)} = 0.59016$$

Παράδειγμα 7.2

Έστω ένα σύστημα με δύο κατηγορίες κίνησης ($K = 2$) και χωρητικότητα $C = 11$ b.u., με μέσους ρυθμούς άφιξης $\lambda_1 = 0.99$, $\lambda_2 = 5.99$ και μέσους ρυθμούς εξυπηρέτησης $\mu_1 = 4.36$, $\mu_2 = 1$ (άρα $\alpha_1 = 0.23$ *erl*, $\alpha_2 = 5.99$ *erl*).

Οι αντίστοιχες απαιτήσεις σε *b.u.* b_1*=1 b.u. και* b_2*=4 b.u.*, ενώ το κοινό κατώφλι ορίζεται ως $J_0 = 4$. (Διευκρινίζεται ότι, οι μέσοι χρόνοι εξυπηρέτησης είναι: $\mu_1^{-1} = 0.23$, $\mu_2^{-1} = 1$ χρονικές μονάδες)

Για την κατηγορία 2, όταν το πλήθος των κατειλημμένων γραμμών υπερβαίνει το κατώφλι οι παράμετροι προσαρμόζονται ως εξής:

$(b_{2c},\ \mu_{2c}^{-1}, \alpha_{2c}) = (3,\ 1.33333,\ 7.98666667)$, με $\alpha_{2c} = \frac{b_2}{b_{2c}}\ \alpha_2 = 7.98666667\ erl$

Από τον αναδρομικό τύπο έχουμε:

$jq(j) = \alpha_1 b_1 \delta_1(j) q(j - b_1) + \alpha_2 b_2 \delta_2(j) q(j - b_2) + \alpha_{2c} b_{2c} \delta_{2c}(j) q(j - b_{2c})$

όπου :

$\delta_1(j) = 1$, γιατί $b_{1c} = 0$ και $1 \le j \le 11$

$\delta_2(j) = 1$, όταν $1 \le j \le J_0 + b_2$ δηλαδή $1 \le j \le 8$ και $b_{2C_1} > 0$

$\delta_{2c}(j) = 1$ όταν $C \ge j > J_0 + b_{2c}$ δηλαδή $8 \ge j > 7$

Ανώτατο όριο είναι το *C*.

$q(0) = 1$

$1q(1) = \alpha_1 b_1 q(0) = 0.22706422$ άρα $q(1) = 0.22706422$

$2q(2) = \alpha_1 b_1 q(1) = 0.05155816$ άρα $q(2) = 0.02577908$

$3q(3) = \alpha_1 b_1 q(2) = 0.005853507$ άρα $q(3) = 0.0019511689$

$4q(4) = \alpha_1 b_1 q(3) + \alpha_2 b_2 q(0) = 23.96044304$ άρα $q(4) = 5.99011076$

$5q(5) = \alpha_1 b_1 q(4) + \alpha_2 b_2 q(1) = 6.80059855$ άρα $q(5) = 1.36011971$

$6q(6) = \alpha_1 b_1 q(5) + \alpha_2 b_2 q(2) = 0.92650128$ άρα $q(6) = 0.15441688$

$7q(7) = \alpha_1 b_1 q(6) + \alpha_2 b_2 q(3) = 0.081812555$ άρα $q(7) = 0.116875079$

$8q(8) = \alpha_1 b_1 q(7) + \alpha_2 b_2 q(4) + \alpha_{2c} b_{2c} q(5) = 176.114176$ άρα $q(8) = 22.014272$

$9q(9) = \alpha_1 b_1 q(8) + 0 + \alpha_{2c} b_{2c} q(6) = 8.698481937$ άρα $q(9) = 0.966497993$

$10q(10) = \alpha_1 b_1 q(9) + 0 + \alpha_{2c} b_{2c} q(7) = 0.499489402$ άρα $q(10) = 0.0499489402$

$11q(11) = \alpha_1 b_1 q(10) + 0 + \alpha_{2c} b_{2c} q(8) = 527.4732991$ άρα $q(11) = 47.9521181$

$G = 79.75396$

$$B_1 = \sum_{j=C-b_1+1}^{C} G^{-1}q(j) = \frac{q(11)}{G} = 0.6012505$$

$$B_{2c} = \sum_{j=C-b_{2c}+1}^{C} G^{-1}q(j) = \frac{q(9)+q(10)+q(11)}{G} = 0.613995$$

$$B_{2c}^{*} = \frac{B_{2c}}{Prob(j > J_0)} = \frac{q(9)+q(10)+q(11)}{q(5)+q(6)+q(7)+q(8)+q(9)+q(10)+q(11)} = 0.675344$$

7.3 Το μοντέλο *STM* υπό την πολιτική *BR*

Όπως και στο μοντέλο *SRM/BR*, σε ορισμένες κατηγορίες κλήσεων επιτρέπεται η εκ των προτέρων δέσμευση συγκεκριμένου αριθμού μονάδων εύρους ζώνης (*b.u.*) υπέρ άλλων κατηγοριών. Η κράτηση αυτή πραγματοποιείται εις βάρος της κατηγορίας που την υφίσταται και μπορεί να διαφέρει ανάλογα με την κατηγορία κλήσεων, προκειμένου να διασφαλιστεί η βέλτιστη κατανομή των πόρων του συστήματος.

Αναλυτικό Μοντέλο

Στηριζόμενοι στον αναδρομικό τύπο του *Roberts* έχουμε [2]:

$$q(j) = \begin{cases} 1 & \alpha\nu\ j = 0 \\ \frac{1}{j}\sum_{k=1}^{K} \alpha_k D_k(j-b_k)\delta_k(j)q(j-b_k) + \sum_{k=1}^{K} a_{kc} D_{kc}(j-b_{kc})\delta_{kc}(j)q(j-b_{kc}) & \alpha\nu\ 1 \le j \le C \\ 0 & \alpha\nu\ j < 0 \text{ ή } j > C \end{cases} \quad (7.11)$$

όπου:

a_{kc}, από τον τύπο (7.8)

α_k, από τον τύπο (4.2)

$\delta_k(j)$, από τον τύπο (7.6)

$\delta_{kc}(j)$, από τον τύπο (7.7)

$D_k(j-b_k)$, από τον τύπο (4.9)

$D_{kc}(j-b_{kc}) = b_{kc}$, (αν $j \le C$ - tr_k) (7.12)

Ο υπολογισμός της πιθανότητα απώλειας κλήσης B_k κατηγορίας k με b_k *b.u.* γίνεται μέσω του τύπου (4.10).

Ο τύπος για την πιθανότητα απώλειας κλήσης B_{kc} κατηγορίας k με b_{kc} $b.u.$ είναι ο εξής [2]:

$$B_{kc} = \sum_{j=C-b_{kc}-tr_k+1}^{C} G^{-1} q(j) \tag{7.13}$$

με G , σύμφωνα με τον τύπο (4.3).

Ο τύπος για τη δεσμευμένη πιθανότητα απώλειας κλήσης B_{kc}^{*} κατηγορίας k με b_{kc} $b.u.$ όταν $j > J_0$ είναι ο εξής [2]:

$$B_{kc}^{*} = \frac{\sum_{j=C-b_{kc}-tr_k+1}^{C} q(j)}{\sum_{j=J_0+1}^{C} q(j)} \tag{7.14}$$

Παράδειγμα 7.3

Συνεχίζουμε το παράδειγμα 6.1 προσθέτοντας $tr_1 = 2$ *b.u.* δηλαδή στην κατηγορία 1 έχουμε κράτηση 2 *b.u.* και $\alpha_{2c} = \frac{b_2}{b_{2c}} \alpha_2 = 1.5\ erl$

Από τον αναδρομικό τύπο (7.11) έχουμε:

$$jq(j) = \alpha_1 D_1(j - b_1)\delta_1(j)q(j - b_1) + \alpha_2 b_2 \delta_2(j)q(j - b_2) + \alpha_{2c} b_{2c} \delta_{2c}(j)q(j - b_{2c})$$

όπου:

$\delta_1(j) = 1$,	γιατί $b_{1c} = 0$ και $1 \le j \le 5$	
$\delta_2(j) = 1$,	όταν $1 \le j \le J_0 + b_2$,	δηλαδή $1 \le j \le 4$ και $b_{2c} > 0$
$\delta_{2c}(j) = 1$,	όταν $J_0 + b_{2c} < j \le C$,	δηλαδή $3 < j \le 5$
$D_1(j - b_1) = b_1$,	όταν $j \le C - tr_1$,	δηλαδή $j \le 4$

$q(0) = 1$

$q(1) = 1$

$2q(2) = 1q(1) = 1$ άρα $q(2) = 0.5$

$3q(3) = 1q(2) + 3q(0) = 3.5$ άρα $q(3) = 1.16667$

$4q(4) = 0 + 3q(1) + 3q(2) = 3 + 1.5 = 4.5$ άρα $q(4) = 1.125$

$5q(5) = 0 + 0 + 3q(3) = 3.5$ άρα $q(5) = 0.7$

G = 5.491667

$$B_1 = \sum_{j=C-b_1-tr_1+1}^{C} G^{-1}q(j) = \frac{q(3)+q(4)+q(5)}{G} = 0.54476$$

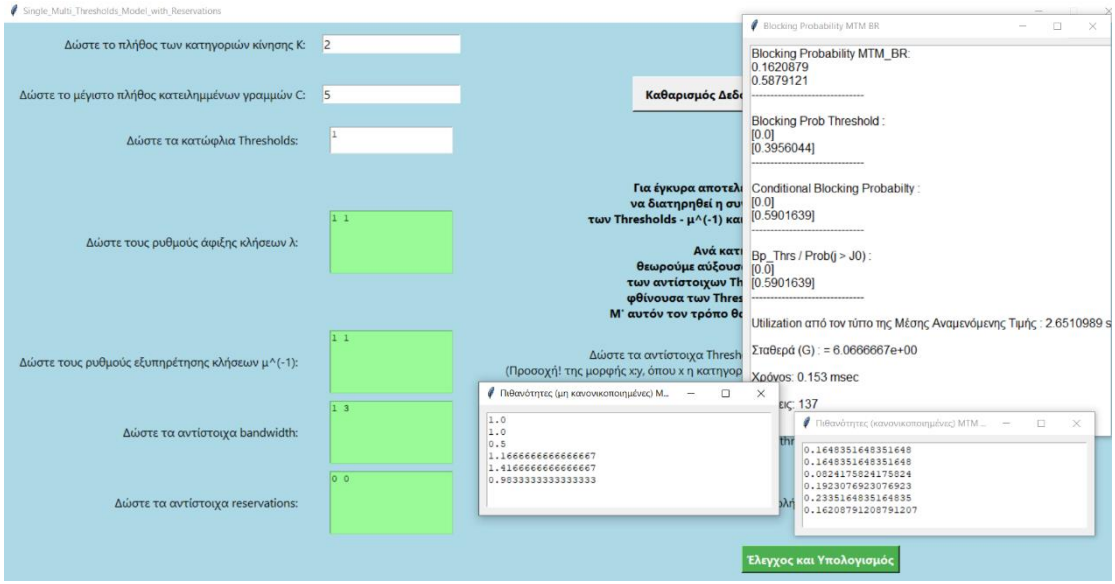
$$B_{2c} = \sum_{j=C-b_{2c}+1}^{C} G^{-1}q(j) = \frac{q(4)+q(5)}{G} = 0.33232$$

Το αποτέλεσμα του υπολογισμού της δεσμευμένης πιθανότητας απώλειας κλήσης είναι το εξής:

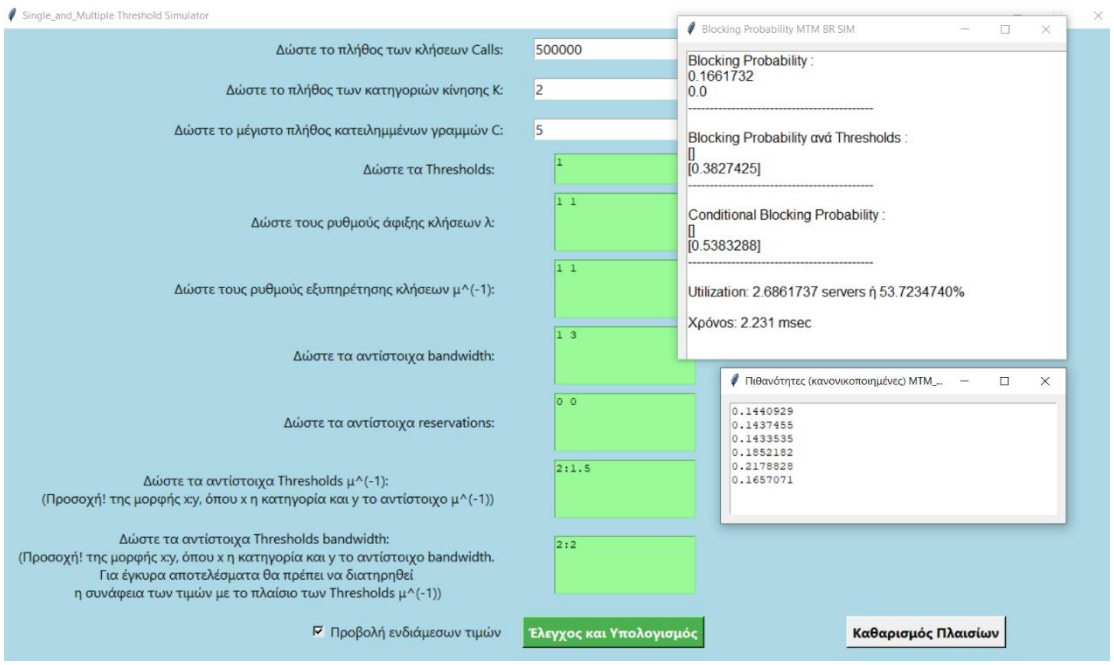
$$B_{2c}^{*} = \frac{B_{2c}}{(q(2)+q(3)+q(4)+q(5))/G} = 0.52267$$

7.4 Αποτελέσματα παραδειγμάτων μέσω των εφαρμογών

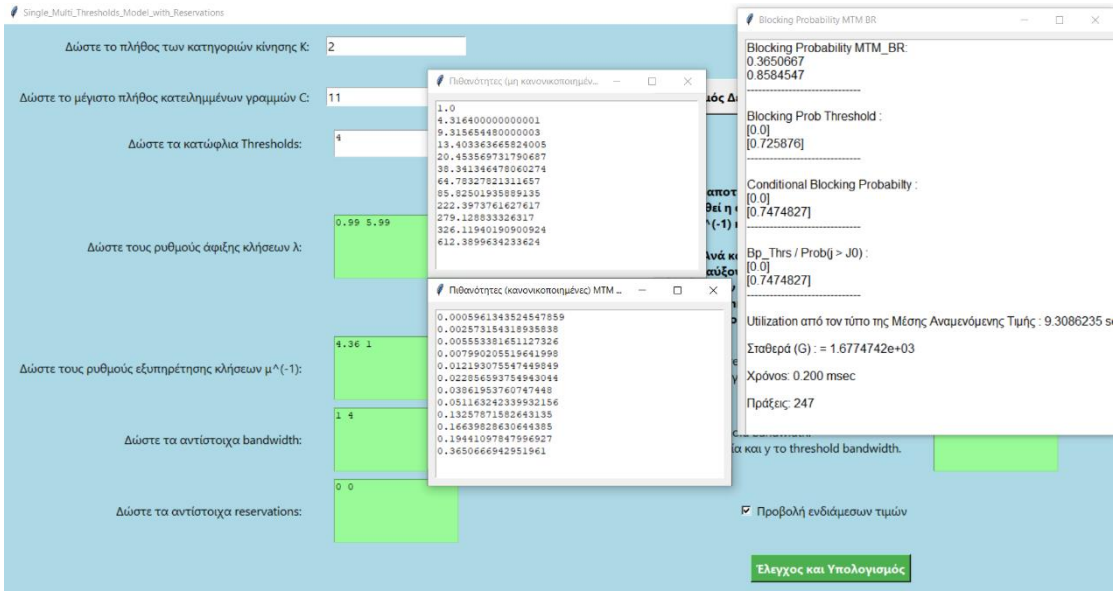
Τα αποτελέσματα των παραδειγμάτων 7.1, 7.2 και 7.3 όπως προκύπτουν από την εκτέλεση των κωδικών του αναλυτικού μοντέλου και της προσομοίωσης παρουσιάζονται στις Εικόνες 23-28. Οι κώδικες αυτοί έχουν συγχωνευτεί με τους αντίστοιχους του μοντέλου *MTM* και αναλύονται στο κεφάλαιο 8. Τα τμήματα κώδικα των αντίστοιχων προγραμμάτων, τόσο για το αναλυτικό μοντέλο όσο και για την προσομοίωση, που αφορούν τα Κεφάλαια 7 και 8, περιλαμβάνονται στο Παράρτημα ΣΤ.



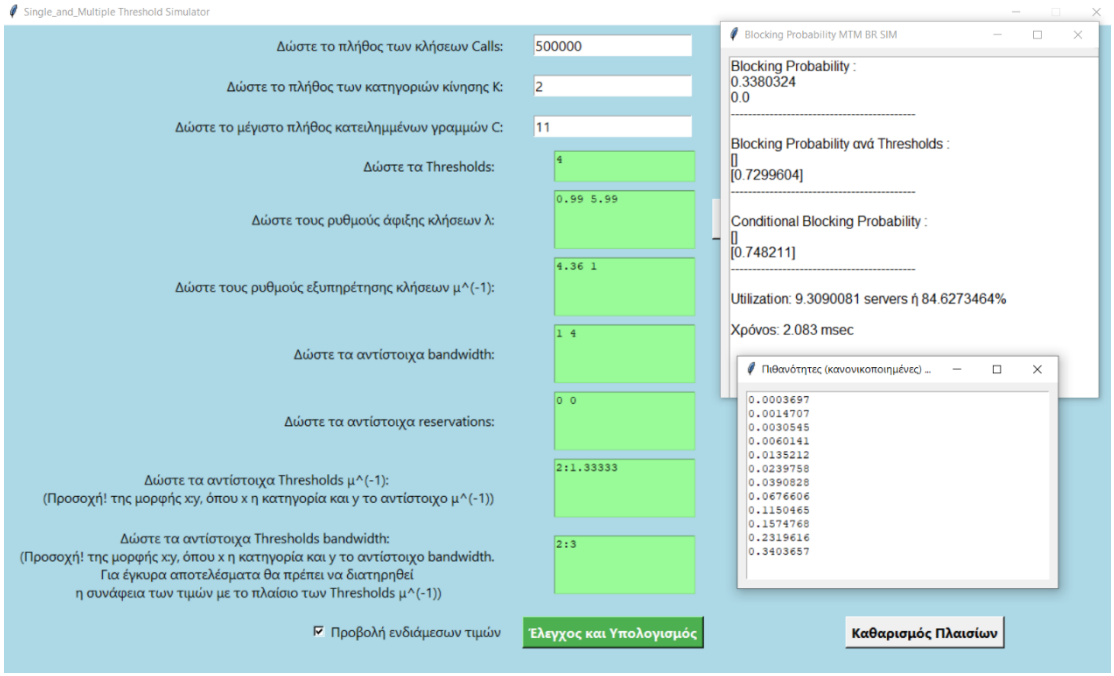
Εικόνα 23: Παράδειγμα 7.1 Αναλυτικό μοντέλο



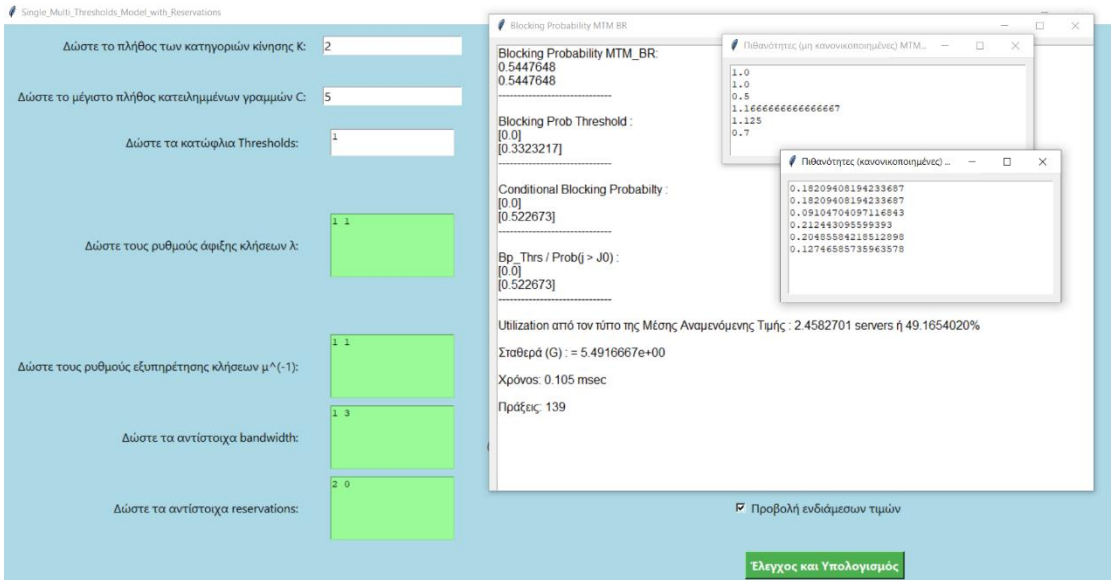
Εικόνα 24: Παράδειγμα 7.1 Προσομοίωση



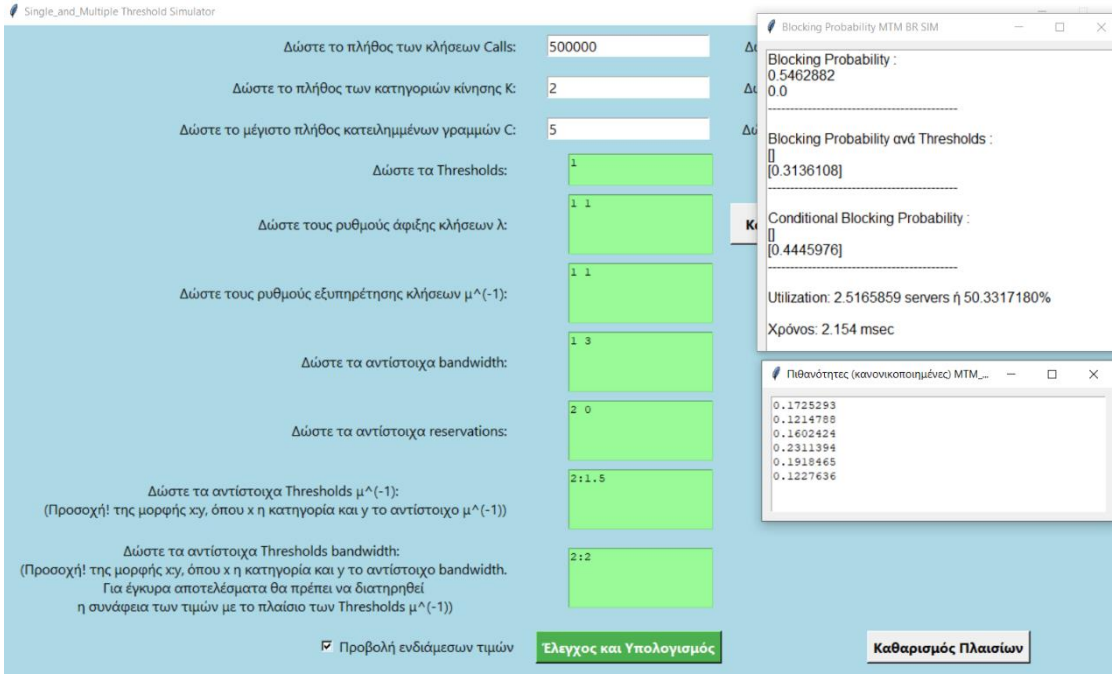
Εικόνα 25: Παράδειγμα 7.2 Αναλυτικό μοντέλο



Εικόνα 26: Παράδειγμα 7.2 Προσομοίωση



Εικόνα 27: Παράδειγμα 7.3 Αναλυτικό μοντέλο



Εικόνα 28: Παράδειγμα 7.3 Προσομοίωση

Παρατηρείται απόκλιση μεταξύ των αποτελεσμάτων του αναλυτικού μοντέλου και εκείνων της προσομοίωσης, γεγονός που είναι αναμενόμενο, δεδομένου ότι ο αναδρομικός τύπος είναι προσεγγιστικός.

Παράδειγμα 7.4

Έστω ότι έχουμε ένα σύστημα με τέσσερις κατηγορίες κλήσεων (K = 4) και κλήσεις καταφθάνουν τυχαία (αφίξεις Poisson) σε ένα σύστημα με εύρος ζώνης C *b.u.* Τα χαρακτηριστικά των κλήσεων δίνονται με τη μορφή διανυσμάτων:

λ = (8,4,2,1), μ^{-1}= (12 ,10,8,6) , b = (1,7,14,28)

$b_{4c} = 14 \quad \mu_{4c}^{-1} = 12$ έτσι ώστε $b_4\ \mu_4^{-1} = b_{4c}\mu_{4c}^{-1}$

Θα μελετηθούν μέσω των εφαρμογών οι περιπτώσεις που το C = 500 *b.u.* και C = 1000 *b.u.* για τα κατώφλια $J_0 = C - 2b_4$.

Τα αποτελέσματα του παραδείγματος 7.4 μετά την εκτέλεση των προγραμμάτων παρουσιάζονται στον Πίνακα 8.

Αναλυτικό μοντέλο					
C = 500	B_1	B_2	B_3	B_{4c}	B^*_{4c}
J_0 = 444	0.0473518	0.2953172	0.5085988	0.5085988	0.5360757

Προσομοίωση			
C = 500	B_1	B_2	B_3
J_0 = 444	0.048349±0.00071	0.294589±0.00088	0.5068503±0.00454
	B_{4c}	B^*_{4c}	
	0.508402±0.00416	0.5288157±0.00345	

Αναλυτικό μοντέλο					
C = 1000	B_1	B_2	B_3	B_{4c}	B^*_{4c}
J_0 = 944	0.000333	0.0024804	0.0054124	0.0054124	0.1579307

Προσομοίωση			
C = 1000	B_1	B_2	B_3
J_0 = 944	0.000256±0.00004	0.001822±0.00029	0.004075±0.00028
	B_{4c}	B^*_{4c}	
	0.004007±0.00039	0.132041±0.00952	

Πίνακας 8: Αποτελέσματα παρ. 7.4

- Τα προγράμματα του αναλυτικού μοντέλου και της προσομοίωσης εκτελέστηκαν για σχετικά υψηλές τιμές του *C*, με κατώφλια κοντά στο *C*.
- Παρατηρείται ότι οι τιμές που προκύπτουν από τον αναδρομικού τύπου του *Roberts* αποκλίνουν από εκείνες της προσομοίωσης.
- Όταν το $C = 500$ *b.u.*, η απόκλιση μεταξύ αναλυτικού μοντέλου και προσομοίωσης παραμένει μικρή.
- Όταν το $C = 1000$ *b.u.*, η απόκλιση αναλυτικού μοντέλου και προσομοίωσης είναι αυξημένη. Στην προσομοίωση, όλες οι κατηγορίες, και ειδικότερα η τέταρτη κατηγορία που επηρεάζεται από το κατώφλι, ικανοποιούν περισσότερες κλήσεις με το αρχικό τους b.u., γεγονός που οφείλεται στην καλύτερη προσαρμογή του συστήματος λόγω του μεγάλου C.

Για την εξισορρόπηση των *B*, θεωρούμε παραμέτρους δέσμευσης εύρους ζώνης ανά κατηγορία με μορφή διανύσματος $tr = (13, 7, 0, 0)$, ώστε να ισχύει:
$b_1 + tr_1 = b_2 + tr_2 = b_3 + tr_3 = b_{4c} + tr_4 = 14$.

Παρακάτω, στον Πίνακα 9, παρουσιάζονται τα αντίστοιχα αποτελέσματα:

Αναλυτικό μοντέλο					
C = 500	B_1	B_2	B_3	B_{4c}	B_{4c}^*
J_0 = 444	0.3782553	0.3782553	0.3782553	0.3782553	0.4044867

Προσομοίωση			
C = 500	B_1	B_2	B_3
J_0 = 444	0.380804±0.00239	0.380152±0.00311	0.378946±0.00328
	B_{4c}		
	0.382146±0.00654		

Αναλυτικό μοντέλο					
C = 1000 b.u.,	B_1	B_2	B_3	B_{4c}	B_{4c}^*
J_0 = 944	0.0041036	0.0041036	0.0041036	0.0041036	0.124351

Προσομοίωση			
C = 1000	B_1	B_2	B_3
J_0 = 944	0.00327±0.00033	0.003302±0.00037	0.00314±0.0006
	B_{4c}		
	0.003096±0.00039		

Πίνακας 9: Αποτελέσματα παρ. 7.4 με *tr* = (13, 7, 0, 0)

8. Μοντέλο πολλαπλών κατωφλίων *MTM*

8.1 Εισαγωγή

Στο κεφάλαιο αυτό εξετάζεται το μοντέλο πολλαπλών κατωφλίων ***(Multi Threshold Model, MTM),*** που πραγματεύεται κλήσεις διαφόρων κατηγοριών, με την ιδιαίτερη χαρακτηριστική ιδιότητα ότι όλες οι κατηγορίες διαθέτουν κοινό πλήθος από κατώφλια. Το *MTM* συνιστά επέκταση του μοντέλου *STM,* στο οποίο προβλέπεται μόνο ένα κατώφλι.

8.2 Το μοντέλο *MTM* υπό την πολιτική *CS*

Στο συγκεκριμένο μοντέλο διαθέτουμε ένα σύνολο S από κατώφλια που είναι κοινό για όλες τις κλήσεις όλων των κατηγοριών, έστω $J_0, J_1, \dots, J_s$. Όταν εισέρχεται μία κλήση της κατηγορίας k με μέσο ρυθμό άφιξης λ_k, σύμφωνα με τη διαδικασία *Poisson*, απαιτεί b_k *b.u.*, με μέσο ρυθμό εξυπηρέτησης μ_k (ο οποίος ακολουθεί την εκθετική κατανομή).

Αν το σύστημα διαθέτει συνολική χωρητικότητα C *b.u.* και ο αριθμός των κατειλημμένων γραμμών είναι j, τότε, όταν j βρίσκεται κάτω **και** από το πρώτο κατώφλι, το σύστημα διαθέτει τις απαιτούμενες μονάδες εύρους ζώνης *b.u.* και τις παραχωρεί για την εξυπηρέτηση της κλήσης.

Στην περίπτωση που ο αριθμός των κατειλημμένων γραμμών j υπερβαίνει το πρώτο κατώφλι αλλά παραμένει κάτω από το δεύτερο κατώφλι (συμπεριλαμβανομένου και του δεύτερου κατωφλίου), οι διαθέσιμες b.u. και ο μέσος ρυθμός εξυπηρέτησης μ μειώνονται αναλόγως, προκειμένου να παραχωρηθούν στην κλήση για την εξυπηρέτησή της.

Η ίδια λογική, δηλαδή η αξιολόγηση των διαστημάτων των κατωφλίων με βάση το πλήθος των κατειλημμένων γραμμών j, εφαρμόζεται διαδοχικά για όλα τα κατώφλια μέχρι το ανώτατο όριο, που αντιστοιχεί στη συνολική χωρητικότητα C του συστήματος. Τέλος, αν όλες οι γραμμές είναι κατειλημμένες, δηλαδή $j = C$ τότε η κλήση απορρίπτεται.

Συγκεκριμένα, έστω ότι διαθέτουμε κατηγορίες κλήσεων K (k = 1,2,3…K), με μέσους ρυθμούς άφιξης κλήσεων $\lambda = (\lambda_1,\lambda_2,\lambda_3,\dots\lambda_k)$, μέσους ρυθμούς εξυπηρέτησης $\mu = (\mu_1,\mu_2,\mu_3,\dots\mu_k)$ και *b.u.* $b = (b_1,b_2,b_3,\dots b_k)$. Κάθε κλήση μπορεί να εξυπηρετηθεί με τις

παραμέτρους *(b_k, μ_k)*, όταν ο αριθμός των κατειλημμένων γραμμών βρίσκεται κάτω από το πρώτο κατώφλι, δηλαδή όταν ισχύει :

$j \leq J_0$, όπου j είναι το πλήθος των κατειλημμένων γραμμών (8.1)

με J_i όπου *(i=0,1,2,3…S-1)* είναι τα S κατώφλια με μέγιστο δυνατό κατώφλι το

$J_{s-1} = C - b_{kc_s}$, όπου $J_{-1} = 0$ και άνω φράγμα $J_s = C$ (8.2)

Ακόμα, (b_{kc_i}, μ_{kc_i}) είναι οι τιμές των *b.u.* και μέσων ρυθμών εξυπηρέτησης μ στα διαστήματα των κατωφλίων.

Φυσικά, αξίζει να σημειώσουμε ότι ισχύει η μείωση των τιμών αυτών ανά διάστημα κατωφλίων, δηλαδή [2]:

$b_k > b_{kc_1} > b_{kc_2} > \ldots > b_{kc_s}$ και $\mu_k > \mu_{kc_1} > \mu_{kc_2} > \ldots > \mu_{kc_s}$ (8.3)

(ως b_k και μ_k μπορούν να νοηθούν τα b_{kc_0} και μ_{kc_0} αντίστοιχα.)

Για τα επόμενα διαστήματα κατωφλιών, δηλαδή πάνω από την τιμή J_0 ισχύει [2]:

$J_{i-1} < j \leq J_i$ (8.4)

με αντίστοιχες τιμές (b_{kc_i}, μ_{kc_i}) και όπως αναφέραμε ως άνω φράγμα το C.

Θα πρέπει να είμαστε ιδιαίτερα προσεκτικοί όταν βρισκόμαστε στην περίπτωση με το άνω φράγμα *C* , διότι η *b.u.* που πρόκειται να δοθεί π.χ. b_{kc_s}, μαζί με τις ήδη κατειλημμένες γραμμές j δεν θα πρέπει να το ξεπερνούν :

$j + b_{kc_s} \leq C$ (8.5)

Αναλυτικό Μοντέλο

Ο υπολογισμός των *q(j),* όπως και στις προηγούμενες περιπτώσεις στηρίζεται στον αναδρομικό τύπο του *Roberts* ο οποίος, για το μοντέλο *MTM*, είναι [2]:

$$q(j) = \begin{cases} 1 & \alpha\nu\ j = 0 \\ \frac{1}{j}\sum_{k=1}^{K} \alpha_k b_k \delta_\kappa(j) q(j - b_k) + \sum_{k=1}^{K}\sum_{s=1}^{S} \alpha_{kc_s} b_{kc_s} \delta_{kc_s}(j) q(j - b_{kc_s}) & \alpha\nu\ 1 \le j \le C \\ 0 & \alpha\nu\ j < 0\ \eta\ \ j > C \end{cases} \quad (8.6)$$

με:

$\delta_k(j) = 1$, $\alpha\nu\ 1 \le j \le J_0 + b_k$ *και* $b_{kc_s} > 0$ *ή* $\alpha\nu\ \ 1 \le j \le C$ *και* $b_{kc_s} = 0$ (8.7)

$\delta_{kc_s}(j) = 1$, $\alpha\nu\ J_s + b_{kc_s} \ge j > J_{s-1} + b_{kc_s}$ *και* $b_{kc_s} > 0$ (8.8)

$$a_{kc_s} = \frac{\lambda_k}{\mu_{kc_s}} \quad (8.9)$$

α_k βάση του τύπου (4.2)

Ο υπολογισμός του B_k κατηγορίας k με b_k *b.u.* γίνεται με βάση τον τύπο (4.5)

Ο τύπος του B_{kc_s} κατηγορίας k με $b_{kc_{s(k)}}$ $b.u.$ είναι [2]:

$$B_{kc_s} = \sum_{j=C-b_{kc_s}+1}^{C} G^{-1} q(j) \quad (8.10)$$

με σταθερά κανονικοποίησης G από τον τύπο (4.3)

Ο τύπος της δεσμευμένης πιθανότητας απώλειας κλήσης $B^*_{kc_s}$ κατηγορίας k *με* b_{kc_s} $b.u.$ όταν $j > J_{S-1}$ είναι [2]:

$$B^*_{kc_s} = \frac{\sum_{j=C-b_{kc_s}+1}^{C} q(j)}{\sum_{j=J_{S-1}+1}^{C} q(j)} \quad (8.11)$$

Η δεσμευμένη αυτή πιθανότητα διαφοροποιείται από την πιθανότητα απώλειας κλήσης στο ότι, στον παρονομαστή, δεν λαμβάνονται υπόψη όλες οι καταστάσεις q, αλλά μόνο εκείνες για τις οποίες ο αριθμός των κατειλημμένων γραμμών j υπερβαίνει το κατώφλι J_{S-1}.

Το U υπολογίζεται με βάση τον τύπο (4.6).

Παράδειγμα 8.1

Έστω ότι έχουμε 2 κατηγορίες κίνησης ($K = 2$) σε ένα σύστημα με χωρητικότητα $C = 8 b.u.$, με μέσους ρυθμούς άφιξης και για τις δύο κατηγορίες $\lambda_1 = \lambda_2 = 1$, με μέσους ρυθμούς εξυπηρέτησης $\mu_1 = \mu_2 = 1$ (άρα $\alpha_1 = \alpha_2 = 1\ erl$) και $b_1 = 1\ b.u.$, $b_2 = 3\ b.u.$
Τα κατώφλια είναι τα εξής: $J_0 = 3$, $J_1 = 4$.

Τα ενδιάμεσα (b, μ^{-1}, α) ανά κατηγορία και κατώφλι είναι τα εξής:

$(b_{2C_1}, \mu_{2C_1}^{-1}, \alpha_{2C_1}) = (2, 1.5, 1.5)$ $\quad$ $(b_{2C_2}, \mu_{2C_2}^{-1}, \alpha_{2C_2}) = (1, 3, 3)$

Από τον αναδρομικό τύπο έχουμε:

$$jq(j) = \alpha_1 b_1 \delta_1(j) q(j - b_1) + \alpha_2 b_2 \delta_2(j) q(j - b_2) + \alpha_{2c_1} b_{2c_1} \delta_{2c_1}(j) q(j - b_{2c_1}) + \alpha_{2c_2} b_{2c_2} \delta_{2c_2}(j) q(j - b_{2c_2})$$

όπου :

$\delta_1(j) = 1$, $\quad$ γιατί $b_{1C_1} = 0$ και $1 \le j \le 8$

$\delta_2(j) = 1$, $\quad$ όταν $j \le J_0 + b_2$, $\quad$ δηλαδή $j \le 6$ και $b_{2C_1} > 0$ (εννοείται $j \ge 1$)

$\delta_{2C_1}(j) = 1$, $\quad$ όταν $J_1 + b_{2C_1} \ge j > J_0 + b_{2C_1}$, $\quad$ δηλαδή $6 \ge j > 5$ άρα $j=6$

$\delta_{2C_2}(j) = 1$, $\quad$ όταν $8 \ge j > J_1 + b_{2C_2}$, $\quad$ δηλαδή $8 \ge j > 5$ άρα το j θα είναι 6 ή 7 ή 8

Υπενθυμίζουμε ότι ανώτατο όριο είναι το C.

$q(0) = 1$

$1q(1) = 1q(0) + 0 + 0 + 0$ άρα $q(1) = 1$

$2q(2) = 1q(1) + 0 + 0 + 0$ άρα $q(2) = 0.5$

$3q(3) = 1q(2) + 3q(0) = 3.5$ άρα $q(3) = 1.16666$

$4q(4) = 1q(3) + 3q(1) = 4.16666$ άρα $q(4) = 1.041666$

$5q(5) = 1q(4) + 3q(2) = 2.5416666$ άρα $q(5) = 0.508333$

$6q(6) = 1q(5) + 3q(3) + 3q(4) + 3q(5) = 8.65831$ άρα $q(6) = 1.44305$

$7q(7) = 1q(6) + 0 + 0 + 3q(6) = 5.7722$ άρα $q(7) = 0.824603$

$8q(8) = 1q(7) + 0 + 0 + 3q(7) = 3.298412$ άρα $q(8) = 0.4123015$

$G = 7.8966$

$B_1 = \frac{q(8)}{G} = 0.05221$ και $B_2 = \frac{q(6)+q(7)+q(8)}{G} = 0.33938$

$$B_{2c_1} = \frac{q(7) + q(8)}{G} = 0.156637$$

$$B_{2c_2} = \frac{q(8)}{G} = 0.05221$$

Παράδειγμα 8.2

Έστω ότι έχουμε 2 κατηγορίες κίνησης ($K = 2$) σε ένα σύστημα με χωρητικότητα $C = 8 b.u.$, με μέσους ρυθμούς άφιξης και για τις δύο κατηγορίες $\lambda_1 = \lambda_2 = 1$, με μέσους ρυθμούς εξυπηρέτησης $\mu_1 = \mu_2 = 1$ (άρα $\alpha_1 = \alpha_2 = 1$ *erl*) και $b_1 = 3$ *b.u. και* $b_2 = 4$ *b.u.*

Τα κατώφλια είναι τα εξής: $J_0 = 3$ $J_1 = 4$.

Τα αντίστοιχα ενδιάμεσα *(b, μ^{-1}, α)* ανά κατηγορία και κατώφλι είναι τα εξής:

$(b_{1C_1},\ \mu_{1C_1}^{-1},\ \alpha_{1C_1}) = (2, 1.5, 1.5)$ $\quad (b_{1C_2},\ \mu_{1C_2}^{-1},\ \alpha_{1C_2}) = (1, 3, 3)$

$(b_{2C_1},\ \mu_{2C_1}^{-1}, \alpha_{2C_1}) = (3, 4/3, 4/3)$ $\quad (b_{2C_2},\ \mu_{2C_2}^{-1}, \alpha_{2C_2}) = (2, 2, 2)$

Από τον αναδρομικό τύπο έχουμε:

$$jq(j) = \alpha_1 b_1 \delta_1(j) q(j - b_1) + \alpha_2 b_2 \delta_2(j) q(j - b_2) + \alpha_{1C_1} b_{1C_1} \delta_{1C_1}(j) q(j - b_{1C_1}) +$$
$$\alpha_{1C_2} b_{1C_2} \delta_{1C_2}(j) q(j - b_{1C_2}) + \alpha_{2C_1} b_{2C_1} \delta_{2C_1}(j) q(j - b_{2C_1}) + \alpha_{2C_2} b_{2C_2} \delta_{2C_2}(j) q(j - b_{2C_2})$$

με :

$\delta_1(j) = 1$, $\quad$ όταν $j \le J_0 + b_1$, $\quad$ δηλαδή $j \le 6$ και $b_{1C_1} > 0$

$\delta_2(j) = 1$, $\quad$ όταν $j \le J_0 + b_2$, $\quad$ δηλαδή $j \le 7$ και $b_{2C_1} > 0$

$\delta_{1C_1}(j) = 1$, όταν $J_1 + b_{1C_1} \ge j > J_0 + b_{1C_1}$, $\quad$ δηλαδή $6 \ge j > 5$

$\delta_{1C_2}(j) = 1$, όταν $8 \ge j > J_1 + b_{1C_2}$, $\quad$ δηλαδή $8 \ge j > 5$

$\delta_{2C_1}(j) = 1$, όταν $J_1 + b_{2C_1} \ge j > J_0 + b_{2C_1}$, $\quad$ δηλαδή $7 \ge j > 6$

$\delta_{2C_2}(j) = 1$, όταν $8 \ge j > J_1 + b_{2C_2}$, $\quad$ δηλαδή $8 \ge j > 6$

Όπως είπαμε και παραπάνω, το ανώτατο όριο είναι το *C*.

$q(0) = 1$

$q(1) = 0$

$q(2) = 0$

$3q(3) = 3q[0] + 0 + 0 + 0 + 0 + 0$ άρα $q(3) = 1$

$4q(4) = 3q(1) + 4q(0) + 0 + 0 + 0 + 0$ άρα $q(4) = 1$

$5q(5) = 3q(2) + 4q(1) + 3q(3) + 3q(4) + 0 + 0 = 6$ άρα $q(5) = 1.2$

$6q(6) = 0 + 4q(2) + 0 + 3q(5) + 0 + 0$ άρα $q(6) = 0.6$

$7q(7) = 0 + 4q(3) + 0 + 3q(6) + 4q(4) + 4q(5) = 4 + 1.8 + 4 + 4.8$ άρα $q(7) = 14.6/7$

$8q(8) = 0 + 0 + 0 + 3q(7) + 0 + 4q(6) = 43.8/7 + 2.4$ άρα $q(8) = 60.6/56$

$G = 7.967857$

$$B_{1c_2} = \sum_{j=C-b_{1c_2}+1}^{C} G^{-1}q(j) = \frac{q(8)}{G} = 0.1358135$$

$$B_{2c_2} = \sum_{j=C-b_{2c_2}+1}^{C} G^{-1}q(j) = \frac{q(7)+q(8)}{G} = 0.39757956$$

8.3 Το μοντέλο *MTM* υπό την πολιτική *BR*

Όπως και στο μοντέλο *MRM/BR*, στο παρόν σύστημα υπάρχουν ορισμένες μονάδες εύρους ζώνης *b.u.* που δεσμεύονται εκ των προτέρων για ορισμένες κατηγορίες κλήσεων. Η δέσμευση αυτή πραγματοποιείται εις βάρος των αντίστοιχων κατηγοριών, προκειμένου να διασφαλιστεί η δυνατότητα εξυπηρέτησης άλλων κατηγοριών κλήσεων με μεγαλύτερη ανάγκη. Επιπλέον, κάθε κατηγορία κλήσεων μπορεί να διαθέτει το δικό της επίπεδο κράτησης σε *b.u.*

Αναλυτικό Μοντέλο

Όπως αναδρομικός τύπος είναι [2]:

$$q(j) = \begin{cases} 1 & \alpha\nu\ j = 0 \\ \frac{1}{j}\sum_{k=1}^{K} \alpha_k D_k(j-b_k)\delta_k(j)q(j-b_k) + \sum_{k=1}^{K}\sum_{s=1}^{S} a_{kc_s}D_{kc_s}(j-b_{kc_s})\delta_{kc_s}(j)q(j-b_{kc_s}) & \alpha\nu\ 1 \le j \le C \\ 0 & \alpha\nu\ j < 0 \text{ ή } j > C \end{cases} \quad (8.12)$$

Όπου:

$\delta_k(j)$	από τον τύπο (8.7)
$\delta_{kc_s}(j)$	από τον τύπο (8.8)
$D_k(j-b_k)$	από τον τύπο (4.9)

$$D_{kc_s}(j - b_{kc_s}) = b_{kc_s}\ ,\ (\alpha\nu\ j \leq C - tr_k) \tag{8.13}$$

a_{kc_s} από τον τύπο (8.9)

a_k από τον τύπο (4.2)

Ο υπολογισμός του B_k κατηγορίας k με b_k *b.u.* γίνεται σύμφωνα με τον τύπο (4.5)

Ο τύπος του B_{kc_s} κατηγορίας k με $b_{kc_{s(k)}}$ $b.u.$ είναι [2]:

$$B_{kc_s} = \sum_{j=C-b_{kc_s}-tr_k+1}^{C} G^{-1}q(j) \tag{8.14}$$

με G που υπολογίζεται σύμφωνα με τον τύπο (4.3)

Ο τύπος του $B^*_{kc_s}$ κατηγορίας k με $b_{kc_{s(k)}}$ $b.u.$ όταν $j > J_{s-1}$ είναι [2]:

$$B^*_{kc_s} = \frac{\sum_{j=C-b_{kc_s}-tr_k+1}^{C} q(j)}{\sum_{j=J_{s-1}+1}^{C} q(j)} \tag{8.15}$$

Παράδειγμα 8.3

Συνεχίζουμε το παράδειγμα 8.2 προσθέτοντας $tr_1 = 1$ *b.u.* δηλαδή δεσμεύοντας 1 *b.u.* στην κατηγορία 1 επιτυγχάνεται εξισορρόπηση του συστήματος, δηλαδή $b_{1c_2} + tr_1 = b_{2c_2}$

Από τον αναδρομικό τύπο έχουμε:

$$jq(j) = \alpha_1 b_1 \delta_1(j) q(j - b_1) + \alpha_2 b_2 \delta_2(j) q(j - b_2) + \alpha_{1C_1} b_{1C_1} \delta_{1C_1}(j) q(j - b_{1C_1}) +$$
$$\alpha_{1C_2} b_{1C_2} \delta_{1C_2}(j) q(j - b_{1C_2}) + \alpha_{2C_1} b_{2C_1} \delta_{2C_1}(j) q(j - b_{2C_1}) + \alpha_{2C_2} b_{2C_2} \delta_{2C_2}(j) q(j - b_{2C_2})$$

Με:

$\delta_1(j) = 1$	όταν $j \leq J_0 + b_1$,	δηλαδή $j \leq 6$ και $b_{1C_1} > 0$
$\delta_2(j) = 1$	όταν $j \leq J_0 + b_2$,	δηλαδή $j \leq 7$ και $b_{2C_1} > 0$
$\delta_{1C_1}(j) = 1$	όταν $J_1 + b_{1C_1} \geq j > J_0 + b_{1C_1}$,	δηλαδή $6 \geq j > 5$
$\delta_{1C_2}(j) = 1$	όταν $8 \geq j > J_1 + b_{1C_2}$,	δηλαδή $8 \geq j > 5$
$\delta_{2C_1}(j) = 1$	όταν $J_1 + b_{2C_1} \geq j > J_0 + b_{2C_1}$,	δηλαδή $7 \geq j > 6$
$\delta_{2C_2}(j) = 1$	όταν $8 \geq j > J_1 + b_{2C_2}$,	δηλαδή $8 \geq j > 6$
$D_1(j - b_1) = b_1$	όταν $j \leq C - tr_1$,	δηλαδή $j \leq 7$

$D_{1c_1}(j - b_{1c_1}) = b_{1c_1}$ όταν $j \leq C - tr_1$, δηλαδή $j \leq 7$

$D_{1c_2}(j - b_{1c_2}) = b_{1c_2}$ όταν $j \leq C - tr_1$, δηλαδή $j \leq 7$

$q(0) = 1$

$q(1) = 0$

$q(2) = 0$

$3q(3) = 3q[0] + 0 + 0 + 0 + 0 + 0$ άρα $q(3) = 1$

$4q(4) = 3q(1) + 4q(0) + 0 + 0 + 0 + 0$ άρα $q(4) = 1$

$5q(5) = 3q(2) + 4q(1) + 0 + 0 + 0 + 0 = 6$ άρα $q(5) = 0$

$6q(6) = 3q(3) + 4q(2) + 3q(4) + 3q(5) + 0 + 0 = 3 + 3$ άρα $q(6) = 1$

$7q(7) = 0 + 4q(3) + 0 + 3q(6) + 4q(4) + 4q(5) = 11$ άρα $q(7) = 11/7$

$8q(8) = 0 + 0 + 0 + 0 + 0 + 4q(6) = 4$ άρα $q(8) = 0.5$

$G = 6.071428$

$B_{1c_2} = \sum_{j=C-b_{1c_2}-tr_1+1}^{C} G^{-1} q(j) = \frac{q(7)+q(8)}{G} = 0.3411765 = B_{2c_2}$

$B^*_{1c_2} = B^*_{2c_2} = \frac{B_{1c_2}}{(q(5)+q(6)+q(7)+q(8))/G} = 0.6744186$

$U = 4.611764$

8.4 Επεξήγηση των προγραμμάτων των *STM/MTM*

Τα τμήματα κώδικα των αντίστοιχων προγραμμάτων, τόσο για την υλοποίηση του αναλυτικού μοντέλου όσο και για την προσομοίωση, βρίσκονται στο Παράρτημα ΣΤ.

8.4.1 Επεξήγηση του προγράμματος του αναλυτικού μοντέλου

Όπως και στα αντίστοιχα προγράμματα των μοντέλων *EMLM* και *SRM/MRM*, εφαρμόζεται το αναλυτικό μοντέλο, δηλαδή ο αναδρομικός τύπος του *Roberts*, υλοποιημένος σε γλώσσα *Python*.

Όσον αφορά το γραφικό περιβάλλον της εφαρμογής ισχύουν περίπου οι ίδιες παρατηρήσεις με αυτές των μοντέλων *SRM/MRM*. Συγκεκριμένα, οι τιμές των *b.u.* και οι αντίστοιχοι χρόνοι εξυπηρέτησης *(μ^{-1})* εισάγονται από τον χρήστη με τη μορφή *x:y*. Όπου *x* αντιστοιχεί στην κατηγορία κλήσης και *y* στην αντίστοιχη τιμή. Αν υπάρχουν πολλαπλά *x:y* για την ίδια

κατηγορία x με διαφορετικές τιμές y, αυτό υποδηλώνει ότι η κατηγορία x διαθέτει διαφορετικές τιμές y για τα αντίστοιχα διαστήματα των κατωφλίων.

Για παράδειγμα θεωρούμε ένα σύστημα με τρεις κατηγορίες κίνησης τις 1, 2 και 3, και δύο κατώφλια $J_0 = 3$ και $J_1 = 4$, σε ένα σύστημα με συνολική χωρητικότητα $C = 12$ b.u.. Οι αρχικές απαιτήσεις σε *b.u.*είναι 2, 5 και 3 για τις κατηγορίες 1, 2 και 3, αντίστοιχα, ενώ οι χρόνοι εξυπηρέτησης μ^{-1} σε χρονικές μονάδες 1, 2 και 3 αντίστοιχα.

Ο χρήστης πρέπει να καθορίσει τις αντίστοιχες τιμές b και μ στα διαστήματα των κατωφλίων ως εξής:

- *Για τα b* έχουμε : (2:4 2:2 3:2 3:1)
- Για τα μ^{-1} έχουμε : (2:3 2:4 3:4 3:5)

Αυτό σημαίνει ότι στο διάστημα κατωφλίων (3,4] η δεύτερη κατηγορία απαιτεί 4 *b.u.* σε χρόνο εξυπηρέτησης 3 χρονικές μονάδες, ενώ η τρίτη κατηγορία απαιτεί 2 *b.u.* σε 4 χρονικές μονάδες. Στο διάστημα (4,12] η δεύτερη κατηγορία απαιτεί 2 *b.u.* σε χρόνο εξυπηρέτησης 4 χρονικές μονάδες και η τρίτη κατηγορία 1 b.u. σε 5 χρονικές μονάδες εξυπηρέτησης.

Σημειώνεται ότι η πρώτη κατηγορία συμπεριφέρεται, περίπου, όπως στο μοντέλο *EMLM*, καθώς τα κατώφλια δεν επηρεάζουν τη συμπεριφορά της.

Το πρόγραμμα πραγματοποιεί ταξινόμηση των δεδομένων που εισάγει ο χρήστης σε περίπτωση που αυτά δεν έχουν δοθεί με τη σωστή σειρά. Συγκεκριμένα, η ταξινόμηση γίνεται κατά φθίνουσα σειρά ως προς τις τιμές των b και κατά αύξουσα σειρά ως προς τις τιμές των μ^{-1}.

Οι αρχικές τιμές των b και μ^{-1}, καθώς και τα υπόλοιπα δεδομένα που παρέχει ο χρήστης, εισάγονται στα αντίστοιχα πεδία της εφαρμογής, με αποτέλεσμα ο χρήστης να μην αντιμετωπίζει δυσκολία κατά την εισαγωγή τους.

(Διευκρίνιση: Για την αποφυγή σύγχυσης, στο διάστημα $[0, J_0]$ εφαρμόζονται οι αρχικές τιμές των b και μ^{-1}, ενώ στα διαστήματα των κατωφλίων εξετάζεται κάθε φορά ο αριθμός των κατειλημμένων κλήσεων j κατά τη δεδομένη χρονική στιγμή.)

8.4.2 Επεξήγηση του προγράμματος της προσομοίωσης

Η φιλοσοφία της προσομοίωσης στις περιπτώσεις των μοντέλων *STM* και *MTM*, είναι ουσιαστικά ανάλογη με εκείνη των μοντέλων *SRM/MRM*. Συγκεκριμένα, κατά την άφιξη των κλήσεων στο σύστημα, καταγράφονται οι χρόνοι άφιξής τους και επαναληπτικά, για το σύνολο των κλήσεων, εφαρμόζονται οι εξής διαδικασίες:

Αρχικά, εξετάζεται ο σωρός των κλήσεων για να εντοπιστούν οι κλήσεις που έχουν ολοκληρώσει την εξυπηρέτησή τους, ώστε να αποδεσμευτούν οι *b.u.* που ήταν δεσμευμένες. Σε αντίθεση με τα μοντέλα *EMLM, SRM* και *MRM*, στα μοντέλα STM/MTM υπάρχουν κατώφλια, τα οποία αποτελούν τιμές b.u. που καθορίζουν τη συμπεριφορά των κλήσεων. Τα κατώφλια μπορεί να είναι ένα, όπως στην περίπτωση του *STM* μοντέλου ή πολλαπλά, όπως στην περίπτωση του *MTM* μοντέλου. Χαρακτηριστικό των μοντέλων *STM/MTM* είναι ότι τα κατώφλια είναι κοινά για όλες τις κλήσεις όλων των κατηγοριών.

Κάθε κατηγορία κλήσεων συμπεριφέρεται ανάλογα με τα διαστήματα των κατωφλίων, λαμβάνοντας υπόψη τις τιμές b και μ^{-1} που παρέχει ο χρήστης. Αν δεν έχουν οριστεί τιμές για κάποιο διάστημα κατωφλίων, η κατηγορία αντιμετωπίζεται, κατά προσέγγιση, όπως στο μοντέλο *EMLM*, καθώς τα κατώφλια δεν επηρεάζουν τη συμπεριφορά της.

Οπότε, ανάλογα με την απαίτηση της κλήσης, συγκεκριμένης κατηγορίας, σε *b.u.*:

- Αν βρισκόμαστε κάτω από το πρώτο κατώφλι (κλειστό διάστημα), η κλήση εξυπηρετείται με τις αρχικές τιμές b.u. και χρόνο εξυπηρέτησης μ^{-1} και αποθηκεύεται στον σωρό με βάση αυτόν τον χρόνο.
- Αν η τιμή j των κατειλημμένων γραμμών βρίσκεται μεταξύ του πρώτου και του δεύτερου κατωφλίου (συμπεριλαμβανομένου και του δεύτερου), η κλήση ικανοποιείται με μειωμένες μονάδες b.u. και αυξημένο χρόνο εξυπηρέτησης, ανάλογα με την κατηγορία, και αποθηκεύεται στον σωρό με βάση τον νέο χρόνο.

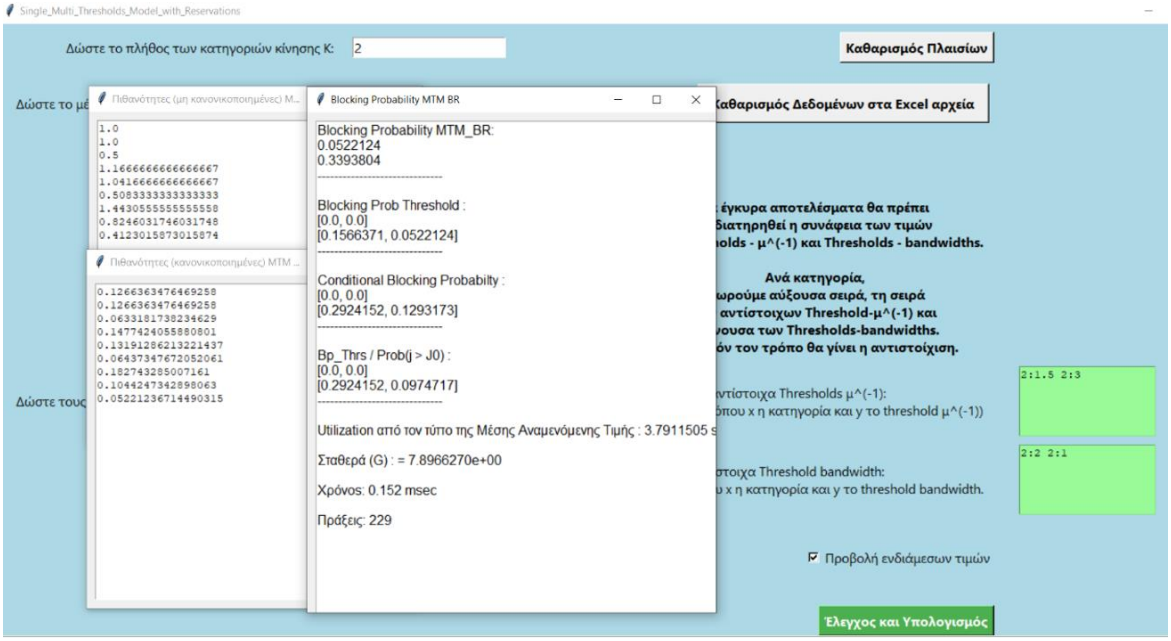
Η ίδια λογική εφαρμόζεται διαδοχικά για όλα τα κατώφλια έως το ανώτατο όριο, που είναι η χωρητικότητα C του συστήματος. Σε περίπτωση που όλα τα διαστήματα των κατωφλίων έχουν εξαντληθεί και η κλήση δεν εξυπηρετήθηκε, τότε απορρίπτεται και δεν καταχωρείται στον σωρό. Σημειώνεται ότι στο μοντέλο *STM* υπάρχει μόνο ένα κατώφλι, με άνω όριο το *C*.

Σε κάθε διάστημα κατωφλίων διατηρείται και μία μετρική που υπολογίζει την πιθανότητα απώλειας κλήσεων για την αντίστοιχη κατηγορία.

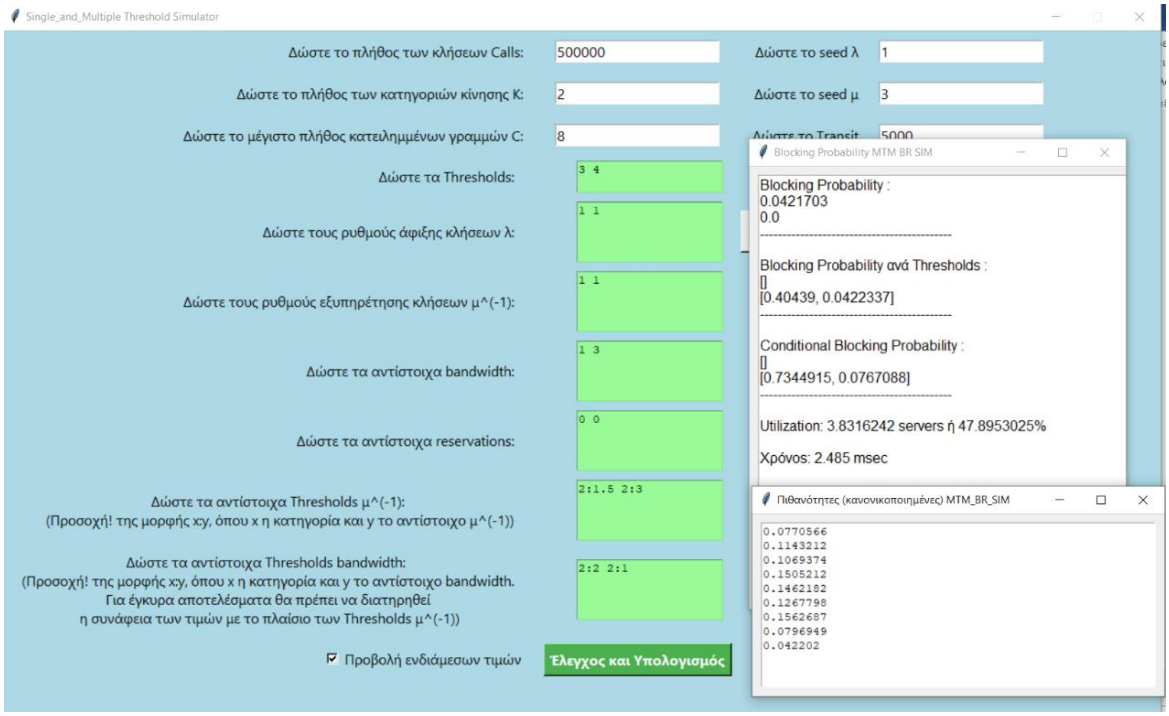
Για το γραφικό περιβάλλον της εφαρμογής, ισχύουν οι ίδιες ιδιαιτερότητες με εκείνες που περιγράφονται στην Ενότητα 8.4.1.

8.5 Αποτελέσματα παραδειγμάτων μέσω των εφαρμογών

Τα αποτελέσματα των Παραδειγμάτων 8.1, 8.2 και 8.3 , όπως προκύπτουν από την εκτέλεση των κωδίκων του αναλυτικού μοντέλου και της προσομοίωσης, παρουσιάζονται στις *Εικόνες 29 και 30*.



Εικόνα 29: Παράδειγμα 8.1 Αναλυτικό μοντέλο



Εικόνα 30: Παράδειγμα 8.1 Προσομοίωση

Με την ίδια διαδικασία προκύπτουν και τα αποτελέσματα για τα υπόλοιπα παραδείγματα, τα οποία παρουσιάζονται στον Πίνακα 10:

- Παρατηρείται απόκλιση μεταξύ των αποτελεσμάτων του αναλυτικού μοντέλου με τα αποτελέσματα της προσομοίωσης, γεγονός που είναι αναμενόμενο, δεδομένου ότι ο αναδρομικός τύπος αποτελεί προσέγγιση και ενδέχεται να αποκλίνει από τις πραγματικές τιμές.
- Στο παράδειγμα 8.2 και στα δύο προγράμματα προκύπτουν αποτελέσματα τα οποία ταυτίζονται με τα αντίστοιχα αποτελέσματα των προγραμμάτων του μοντέλου *CDTM* στο Παράδειγμα 9.2

Αναλυτικό πρόγραμμα	$B_{1_{c2}}$	$B_{2_{c2}}$	$B^*_{1_{c2}}$	$B^*_{2_{c2}}$
Παράδειγμα 8.2	0.1635388	0.3994636	0.297561	0.7268291
Παράδειγμα 8.3 (με tr_1 =1)	0.3411763	0.3411763	0.6744185	0.6744185

Προσομοιωτής Παράδειγμα 8.1	B_1	$B_{2_{c2}}$
	0.042072±0.00031	0.041931±0.0003
	$B^*_{2_{c2}}$	
	0.076092±0.00052	

Προσομοιωτής Παράδειγμα 8.2	$B_{1_{c2}}$	$B_{2_{c2}}$
	0.157718±0.00148	0.357433±0.00098
	$B^*_{1_{c2}}$	$B^*_{2_{c2}}$
	0.211306±0.00147	0.478791±0.0012

Προσομοιωτής Παράδειγμα 8.3 (με tr_1 =1)	$B_{1_{c2}}$	$B_{2_{c2}}$
	0.32594±0.00116	0.326055±0.00063
	$B^*_{1_{c2}}$	$B^*_{2_{c2}}$
	0.472673±0.00159	0.47255864±0.00081

Πίνακας 10: Αποτελέσματα παρ. 8.2, 8.3

Παράδειγμα 8.4

Έστω ότι έχουμε ένα σύστημα με τέσσερις κατηγορίες κλήσεων (K=4) και κλήσεις καταφθάνουν τυχαία (με τη διαδικασία *Poisson*), με χωρητικότητα C=300 *b.u.* Τα χαρακτηριστικά των κλήσεων δίνονται με τη μορφή διανυσμάτων:

λ = (8,6,4,1), μ^{-1}= (6 ,4,3,1) , b = (1,4,6,24)

$b_{4c} = (20\,, 16\,, 12, 8)$ $\quad \mu_{4c}^{-1} = (1.2\,, 1.5\,, 2,\ 3)$

Στους παρακάτω Πίνακες 11 και 12 παρουσιάζονται τα αποτελέσματα των προγραμμάτων που εκτελέστηκαν:

Αναλυτικό Μοντέλο					
(J_1, J_2, J_3, J_4)	B_1	B_2	B_3	B_{4c_4}	$B^*_{4c_4}$
(276,280,284,288)	0.0048874	0.0202147	0.0273192	0.0349248	0.6752276
(266,270,274,278)	0.0036735	0.0156799	0.0249284	0.0352362	0.3063621
(256,260,264,268)	0.0032342	0.0139457	0.0220201	0.0308863	0.1600228
(246,250,254,258)	0.0029306	0.0126516	0.0199793	0.028038	0.0979773
(236,240,244,248)	0.0027194	0.0117486	0.0185534	0.0260364	0.0662693
(206,210,214,218)	0.0024221	0.0104695	0.0165336	0.023202	0.0316385
(166,170,174,178)	0.0023545	0.0101775	0.0160724	0.0225548	0.0231724

Πίνακας 11: Αποτελέσματα παρ.8.4 με το αναλυτικό μοντέλο

Προσομοίωση					
(J_1, J_2, J_3, J_4)	B_1	B_2	B_3	B_{4c_4}	$B^*_{4c_4}$
(276,280,284,288)	0.0052354	0.0201166	0.0304357	0.0388428	0.3371124
(266,270,274,278)	0.0038786	0.0150027	0.0236765	0.031543	0.1557284
(256,260,264,268)	0.0026465	0.0112968	0.0184534	0.0265868	0.0899402
(246,250,254,258)	0.0024979	0.0102151	0.0158995	0.0216306	0.052607
(236,240,244,248)	0.0023348	0.0093959	0.0143825	0.0222069	0.0415558
(206,210,214,218)	0.0022006	0.0092551	0.0149874	0.0212848	0.0249977
(166,170,174,178)	0.0021623	0.0096455	0.0150546	0.0224374	0.0226304

Πίνακας 12: Αποτελέσματα παρ.8.4 με την προσομοίωση

- Τα προγράμματα εκτελέστηκαν για σχετικά μεγάλες τιμές κατωφλίων. Διαπιστώνεται ότι, καθώς οι τιμές των κατωφλίων μειώνονται σταδιακά, το σύστημα τείνει να εξομαλύνεται, γεγονός που οδηγεί σε χαμηλότερες τιμές απωλειών κλήσεων.
- Παρατηρείται ότι οι τιμές που προκύπτουν από τον αναδρομικό τύπο του *Roberts* αποκλίνουν από εκείνες που υπολογίζονται μέσω της προσομοίωσης. Ωστόσο, διαπιστώθηκε ότι για υψηλές τιμές κατωφλίων, στις κατηγορίες όπου δεν υπάρχουν κατώφλια, επιτυγχάνεται καλή προσέγγιση μεταξύ των αποτελεσμάτων των δύο προγραμμάτων. Αντίθετα, για την κατηγορία που επηρεάζεται άμεσα από τα κατώφλια, η προσέγγιση των αποτελεσμάτων είναι καλύτερη όταν τα κατώφλια λαμβάνουν μικρές τιμές.
- Το φαινόμενο αυτό ενδέχεται να εξηγείται από το γεγονός ότι, όταν τα κατώφλια λαμβάνουν τιμές κοντά στο ανώτατο φράγμα C, το σύστημα περιορίζεται τόσο από τις τιμές των κατωφλίων όσο και από το ίδιο το άνω φράγμα. Ως, αποτέλεσμα, να

αδυνατεί να εξυπηρετήσει τις εισερχόμενες κλήσεις, λόγω του μικρού διαστήματος μεταξύ του τελευταίου κατωφλίου και του *C,* και να οδηγείται σε αυξημένη απόρριψη τους. Η συμπεριφορά αυτή, μπορεί να αποτυπωθεί με μεγαλύτερη ακρίβεια από την προσομοίωση, σε αντίθεση με τον αναδρομικό τύπο, ο οποίος αποτελεί προσεγγιστική αναλυτική μέθοδο.

- Η επιλογή χαμηλότερων τιμών στα κατώφλια περιορίζει τη δυνατότητα της 4ης κατηγορίας να χρησιμοποιεί τις αρχικές απαιτήσεις της σε *b.u.,* αφήνοντας έτσι χώρο για τις υπόλοιπες κατηγορίες. Η επίδραση αυτή αποτυπώνεται και στη συνολική μείωση των πιθανοτήτων απώλειας κλήσεων για όλες τις κατηγορίες.
- Επιπλέον, οι τιμές του πρώτου συνόλου κατωφλίων οδηγούν σε αποτελέσματα τα οποία ταυτίζονται με εκείνα που προκύπτουν από το μοντέλο *MRM.*

Για την εξισορρόπηση των απωλειών θεωρούμε ως τιμές κράτησης tr = (7, 4, 2, 0), έτσι ώστε να ισχύει:

$b_1 + tr_1 = b_2 + tr_2 = b_3 + tr_3 = b_{4c_4} + tr_4 = 8$ b.u.

Στον *Πίνακα 13* αποτυπώνονται τα αποτελέσματα.

Αναλυτικό Μοντέλο				
(J_1 , J_2 , J_3 , J_4)	**B1**	**B2**	**B_3**	**B_{4c_4}**
(276,280,284,288)	0.0237857	0.0237857	0.0237857	0.0237857
Προσομοίωση				
(J_1 , J_2 , J_3 , J_4)	**B1**	**B2**	**B_3**	**B_{4c_4}**
(276,280,284,288)	0.0280949 ±0.00113	0.028674 ±0.00093	0.028189 ±0.00086	0.028189 ±0.00078

Πίνακας 13: Αποτελέσματα παρ. 8.4 με *tr* = (7, 4, 2, 0)

9. Μοντέλο *CDTM*

9.1 Εισαγωγή

Στο συγκεκριμένο κεφάλαιο εξετάζεται το μοντέλο με κατώφλι εξαρτώμενο από τις συνδέσεις ***(Connection Dependent Threshold Model, CDTM),*** το οποίο χειρίζεται κλήσεις διαφόρων κατηγοριών, με την ιδιαίτερη χαρακτηριστική ιδιότητα ότι κάθε κατηγορία διαθέτει το δικό της σύνολο κατωφλίων.

9.2 Το μοντέλο *CDTM* υπό την πολιτική *CS*

Ας θεωρήσουμε ένα σύνολο κατηγοριών κλήσεων K $(k = 1,2,3,\ldots.K)$, όπου κάθε κλήση εισέρχεται στο σύστημα τυχαία (με διαδικασία *Poisson*) με μέσο ρυθμό άφιξης λ_k. Έστω ότι κάθε κλήση απαιτεί έναν αριθμό b_k b.u., με μέσο ρυθμό εξυπηρέτησης μ_k (που ακολουθεί την εκθετική κατανομή). Σύμφωνα με τη λογική του συγκεκριμένου μοντέλου, μέχρι **και** το πρώτο κατώφλι η κλήση μπορεί να εξυπηρετηθεί με το b_k και μέσο ρυθμό εξυπηρέτησης μ_k. Πέρα από το πρώτο κατώφλι τόσο το b_k.όσο και το μ_k μειώνονται ανάλογα.

Η λογική λειτουργίας του μοντέλου βασίζεται στο ότι αν μία κλήση δεν μπορεί να ικανοποιηθεί με τις αρχικές τιμές (b_k, μ_k), τότε οι τιμές των $(b\ ,\ \mu)$ μειώνονται προκειμένου να καταστεί δυνατή η εξυπηρέτηση την κλήσης. Η διαδικασία αυτή επαναλαμβάνεται και με τα επόμενα κατώφλια έως το ανώτατο φράγμα C. Σε περίπτωση που η κλήση δεν μπορεί να εξυπηρετηθεί ακόμη και μετά την εφαρμογή του μηχανισμού φιλτραρίσματος των κατωφλίων, αυτή απορρίπτεται.

Συγκεκριμένα, για το σύνολο κατηγοριών κλήσεων K, οι μέσοι ρυθμοί άφιξης κλήσεων $\lambda = (\lambda_1,\lambda_2,\lambda_3,\ldots.\lambda_k)$, με μέσους ρυθμούς εξυπηρέτησης $\mu = (\mu_1,\mu_2,\mu_3,\ldots\mu_k)$ και μονάδες εύρους ζώνης $b = (b_1,b_2,b_3,\ldots.b_k)$ αντίστοιχα. Κάθε κλήση μπορεί να εξυπηρετηθεί με παραμέτρους $(b_k,\ \mu_k)$ όταν το σύστημα βρίσκεται κάτω από το πρώτο κατώφλι της αντίστοιχης κατηγορίας, δηλαδή όταν ισχύει [2]:

$j \leq J_{k_0}$, όπου j είναι το πλήθος των κατειλημμένων γραμμών (9.1)

Τα J_{k_i}, με i=0,1,2,3…s-1, αντιστοιχούν στα s κατώφλια ανά κατηγορία k.

Για το τελευταίο κατώφλι ισχύει [2]:

$$J_{k_z} = C \tag{9.2}$$

Επιπλέον, τα ζεύγη (b_{kc_i}, μ_{kc_i}) αντιστοιχούν στις τιμές των *b.u.* και των μέσων ρυθμών εξυπηρέτησης μ στα αντίστοιχα διαστήματα των κατωφλίων. Επίσης, ισχύουν οι σχέσεις [2]:

$$b_k > b_{kc_1} > b_{kc_2} > \ldots. > b_{kc_s} \quad \text{και} \quad \mu_k > \mu_{kc_1} > \mu_{kc_2} > \ldots. > \mu_{kc_s} \tag{9.3}$$

(Οι παράμετροι b_k και μ_k μπορούν να ερμηνευθούν ως τα b_{kc0} και μ_{kc0}, αντίστοιχα.)

Η σχέση (9.3) μας δείχνει τη μείωση των τιμών των b.u. και μ στα διαστήματα των κατωφλίων.

Για τα επόμενα διαστήματα κατωφλίων, δηλαδή πάνω από την τιμή J_{k0} ισχύει [2]:

$$J_{k_i} < j \leq J_{k_{i+1}} \tag{9.4}$$

με αντίστοιχες τιμές $(b_{kc_{i+1}}, \mu_{kc_{i+1}})$ και ανώτατο φράγμα το C.

Απαιτείται ιδιαίτερη προσοχή στην περίπτωση κατά την οποία το σύστημα βρίσκεται κοντά στο ανώτατο φράγμα *C*, διότι το αποτέλεσμα του αθροίσματος της τιμής του *b.u.* που πρόκειται να αποδοθεί π.χ. b_{kc_s}, μαζί με τις ήδη κατειλημμένες γραμμές *j* δεν πρέπει να υπερβαίνει το ανώτατο όριο του συστήματος C, έτσι έχουμε [2]:

$$j + b_{kc_s} \leq C \tag{9.5}$$

Αναλυτικό Μοντέλο [2]

Το αναλυτικό μοντέλο για τον υπολογισμό των *q(j)*, όπως και στις προηγούμενες περιπτώσεις στηρίζεται στον αναδρομικό τύπο του Roberts ο οποίος, για το μοντέλο *CDTM*, είναι [2]:

$$q(j) = \begin{cases} 1 & \alpha\nu\ j = 0 \\ \frac{1}{j} \sum_{k=1}^{K} \alpha_k b_k \delta_\kappa(j) q(j - b_k) + \sum_{k=1}^{K} \sum_{s=1}^{S(k)} \alpha_{kc_s} b_{kc_s} \delta_{kc_s}(j) q(j - b_{kc_s}) & \alpha\nu\ 1 \leq j \leq C \\ 0 & \alpha\nu\ j < 0 \text{ ή } j > C \end{cases} \tag{9.6}$$

$\delta_k(j) = 1$, αν $1 \le j \le J_{k_0} + b_k$ και $b_{kc_s} > 0$ ή αν $1 \le j \le C$ και $b_{kc_s} = 0$ (9.7)

$\delta_{kc_s}(j) = 1$, αν $J_{k_s} + b_{kc_s} \ge j > J_{k_{s-1}} + b_{kc_s}$ και $b_{kc_s} > 0$ (9.8)

και a_{kc_s} , a_k από τους τύπους (8.9) και (4.2) αντίστοιχα.

Ο τύπος της πιθανότητας B_k κατηγορίας k με b_k *b.u.* είναι ο (4.5)

Ο τύπος της πιθανότητας $B_{kc_{s(k)}}$ κατηγορίας k με $b_{kc_{s(k)}}$ $b.u.$ είναι [2]:

$$B_{kc_{s(k)}} = \sum_{j=C-b_{kc_{s(k)}}+1}^{C} G^{-1} q(j) \quad (9.9)$$

και σταθερά κανονικοποίησης G με βάση τον τύπο (4.3)

Ο τύπος της δεσμευμένης πιθανότητας απώλειας κλήσης $B^*_{kc_{s(k)}}$ κατηγορίας k με $b_{kc_{s(k)}}$ $b.u.$ όταν $j >$ $J_{k_{s(k)-1}}$ είναι [2]:

$$B^*_{kc_{s(k)}} = \frac{\sum_{j=C-b_{kc_{s(k)}}+1}^{C} q(j)}{\sum_{j=J_{k_{s(k)-1}}+1}^{C} q(j)} \quad (9.10)$$

Η δεσμευμένη αυτή πιθανότητα διαφέρει από την πιθανότητα απώλειας κλήσης στο γεγονός ότι στον παρονομαστή δεν λαμβάνονται υπόψη όλες οι πιθανότητες κατάστασης q, αλλά εκείνες για τις οποίες η τιμή του j υπερβαίνει το κατώφλι $J_{k_{s(k)-1}}$.

Ο τύπος του U βασίζεται στη σχέση (4.6)

Παράδειγμα 9.1

Έστω ότι έχουμε 2 κατηγορίες κίνησης (K = 2) σε ένα σύστημα με χωρητικότητα C = 8 *b.u.*, με μέσους ρυθμούς άφιξης και για τις δύο κατηγορίες $\lambda_1 = \lambda_2 = 1$, μέσους ρυθμούς εξυπηρέτησης $\mu_1 = \mu_2 = 1$ (άρα $\alpha_1 = \alpha_2 = 1$ *erl*) και $b_1 = 3$ *b.u.*, $b_2 = 4$ *b.u.* Τα κατώφλια $J_{1_0} = 2$, $J_{1_1} = 3$ και $J_{2_0} = 3, J_{2_1} = 4$.

Τα αντίστοιχα ενδιάμεσα b, μ^{-1}, α ανά κατηγορία και κατώφλι είναι:

$(b_{1C_1},\ \mu_{1C_1}^{-1},\ \alpha_{1C_1}) = (2, 1.5, 1.5)$ $(b_{1C_2},\ \mu_{1C_2}^{-1},\ \alpha_{1C_2}) = (1, 3, 3)$

$(b_{2C_1},\ \mu_{2C_1}^{-1}, \alpha_{2C_1}) = (3, 4/3, 4/3)$ $(b_{2C_2},\ \mu_{2C_2}^{-1}, \alpha_{2C_2}) = (2, 2, 2)$

Από τον αναδρομικό τύπο (9.6) έχουμε:

$$jq(j) = \alpha_1 b_1 \delta_1(j) q(j-b_1) + \alpha_2 b_2 \delta_2(j) q(j-b_2) + \alpha_{1C_1} b_{1C_1} \delta_{1C_1}(j) q(j-b_{1C_1}) + \alpha_{1C_2} b_{1C_2} \delta_{1C_2}(j) q(j-b_{1C_2}) + \alpha_{2C_1} b_{2C_1} \delta_{2C_1}(j) q(j-b_{2C_1}) + \alpha_{2C_2} b_{2C_2} \delta_{2C_2}(j) q(j-b_{2C_2})$$

με : $\delta_1(j) = 1$ όταν $j \leq J_{1_0} + b_1$ δηλαδή $j \leq 5$ και $b_{1C_1} > 0$

$\delta_2(j) = 1$ όταν $j \leq J_{2_0} + b_2$ δηλαδή $j \leq 7$ και $b_{2C_1} > 0$

$\delta_{1C_1}(j) = 1$ όταν $J_{1_1} + b_{1C_1} \geq j > J_{1_0} + b_{1C_1}$ δηλαδή $5 \geq j > 4$

$\delta_{1C_2}(j) = 1$ όταν $J_{1_2} + b_{1C_2} \geq j > J_{1_1} + b_{1C_2}$ δηλαδή $8 \geq j > 4$

$\delta_{2C_1}(j) = 1$ όταν $J_{2_1} + b_{2C_1} \geq j > J_{2_0} + b_{2C_1}$ δηλαδή $7 \geq j > 6$

$\delta_{2C_2}(j) = 1$ όταν $J_{2_2} + b_{2C_2} \geq j > J_{2_1} + b_{2C_2}$ δηλαδή $8 \geq j > 6$

Το J_{1_2} και το J_{2_2} δεν έχουν οριστεί αλλά, όπως αναφέρθηκε και στο θεωρητικό μέρος, ως ανώτατο φράγμα θεωρείται η χωρητικότητα C. Συνεπώς, προκειμένου να διασφαλιστεί ότι η τιμή δεν υπερβαίνει το όριο αυτό, εφαρμόζονται οι ακόλουθες πράξεις:

$J_{1_2} + b_{1C_2} = 8 \Rightarrow J_{1_2} = 8 - 1 = 7$ και $J_{2_2} + b_{2C_2} = 8 \Rightarrow J_{2_2} = 8 - 2 = 6$

$q(0) = 1$

$q(1) = 0$

$q(2) = 0$

$3\,q(3) = 3\,q(0) + 0 + 0 + 0 + 0 + 0$ άρα $q(3) = 1$

$4q(4) = 3q(1) + 4q(0) + 0 + 0 + 0 + 0$ άρα $q(4) = 1$

$5q(5) = 3q(2) + 4q(1) + 3q(3) + 3q(4) + 0 + 0 = 6$ άρα $q(5) = 1.2$

$6q(6) = 0 + 4q(2) + 0 + 3q(5) + 0 + 0$ άρα $q(6) = 0.6$

$7q(7) = 0 + 4q(3) + 0 + 3q(6) + 4q(4) + 4q(5) = 4 + 1.8 + 4 + 4.8$ άρα $q(7) = 14.6/7$

$8q(8) = 0 + 0 + 0 + 3q(7) + 0 + 4q(6) = 43.8/7 + 2.4$ άρα $q(8) = 60.6/56$

$G = 7.967857$

$$B_{1c_2} = \sum_{j=C-b_{1c_2}+1}^{C} G^{-1} q(j) = \frac{q(8)}{G} = 0.1358135$$

$$B_{2c_2} = \sum_{j=C-b_{2c_2}+1}^{C} G^{-1} q(j) = \frac{q(7)+q(8)}{G} = 0.39757956$$

$$B^*_{1c_2} = \frac{B_{1c_2}}{(q(4)+q(5)+q(6)+q(7)+q(8))/G} = 0.1813285$$

$$B^*_{2c_2} = \frac{B_{2c_2}}{(q(5)+q(6)+q(7)+q(8))/G} = 0.6376707$$

U = 5.0022411

Παράδειγμα 9.2

Το συγκεκριμένο παράδειγμα είναι παρόμοιο με το προηγούμενο, με τη διαφορά ότι τα κατώφλια είναι κοινά και για τις δύο κατηγορίες, δηλαδή $J_0 = 3$ και $J_1 = 4$. Όπως θα φανεί από τα αποτελέσματα, το παράδειγμα αυτό εκτελείται υπό τη λογική ενός συστήματος *MTM*.

Ο αναδρομικός τύπος παραμένει ο ίδιος με αυτόν του Παραδείγματος 9.1 και οι παράμετροι δ τροποποιούνται ως εξής:

$\delta_1(j)$ = 1 όταν $j \le J_0 + b_1$ δηλαδή $j \le 6$ και $b_{1C_1} > 0$

$\delta_2(j)$ = 1 όταν $j \le J_0 + b_2$ δηλαδή $j \le 7$ και $b_{2C_1} > 0$

$\delta_{1C_1}(j) = 1$ όταν $J_1 + b_{1C_1} \ge j > J_0 + b_{1C_1}$ δηλαδή $6 \ge j > 5$

$\delta_{1C_2}(j) = 1$ όταν $J_2 + b_{1C_2} \ge j > J_1 + b_{1C_2}$ δηλαδή $8 \ge j > 5$

$\delta_{2C_1}(j) = 1$ όταν $J_1 + b_{2C_1} \ge j > J_0 + b_{2C_1}$ δηλαδή $7 \ge j > 6$

$\delta_{2C_2}(j) = 1$ όταν $J_2 + b_{2C_2} \ge j > J_1 + b_{2C_2}$ δηλαδή $8 \ge j > 6$

Άρα έχουμε:

$q(0) = 1$

$q(1) = 0$

$q(2) = 0$

$3q(3) = 3q(0) + 0 + 0 + 0 + 0 + 0$ άρα $q(3) = 1$

$4q(4) = 3q(1) + 4q(0) + 0 + 0 + 0 + 0$ άρα $q(4) = 1$

$5q(5) = 3q(2) + 4q(1) + 0 + 0 + 0 + 0$ άρα $q(5) = 0$

$6q(6) = 3q(3) + 4q(2) + 3q(4) + 3q(5) + 0 + 0 = 6$ άρα $q(6) = 1$

$7q(7) = 0 + 4q(3) + 0 + 3q(6) + 4q(4) + 4q(5) = 11$ άρα $q(7) = 11/7$

$8q(8) = 0 + 0 + 0 + 3q(7) + 0 + 4q(6) = 33/7 + 4$ άρα $q(8) = 61/56$

G = 6.66071

$$B_{1c_2} = \sum_{j=C-b_{1c_2}+1}^{C} G^{-1}q(j) = \frac{q(8)}{G} = 0.1635388$$

$$B_{2c_2} = \sum_{j=C-b_{2c_2}+1}^{C} G^{-1}q(j) = \frac{q(7)+q(8)}{G} = 0.3994638$$

$$B_{1c_2}^{*} = \frac{B_{1c_2}}{(q(5)+q(6)+q(7)+q(8))/G} = 0.297561$$

$$B_{2c_2}^{*} = \frac{B_{2c_2}}{(q(5)+q(6)+q(7)+q(8))/G} = 0.726829$$

U = 4.911528

Παράδειγμα 9.3

Το συγκεκριμένο παράδειγμα είναι παρόμοιο με το Παράδειγμα 9.1, με τη διαφορά ότι τα κατώφλια αλλάζουν ως εξής: $J_{1_0} = 5,\ J_{1_1} = 6$ και $J_{2_0} = 4, J_{2_1} = 5$. Όπως θα φανεί από τα αποτελέσματα, το παράδειγμα αυτό εκτελείται υπό τη λογική ενός συστήματος *MRM*.

Ο αναδρομικός τύπος παραμένει ο ίδιος με αυτόν του Παραδείγματος 9.1 και οι παράμετροι δ τροποποιούνται ως εξής:

$\delta_1(j)$ = 1 όταν $j \leq J_{1_0} + b_1$ δηλαδή $j \leq 8$ και $b_{1C_1} > 0$

$\delta_2(j)$ = 1 όταν $j \leq J_{2_0} + b_2$ δηλαδή $j \leq 8$ και $b_{2C_1} > 0$

$\delta_{1C_1}(j) = 1$ όταν $J_{1_1} + b_{1C_1} \geq j > J_{1_0} + b_{1C_1}$ δηλαδή $8 \geq j > 7$

$\delta_{1C_2}(j) = 1$ όταν $J_{1_2} + b_{1C_2} \geq j > J_{1_1} + b_{1C_2}$ δηλαδή $8 \geq j > 7$

$\delta_{2C_1}(j) = 1$ όταν $J_{2_1} + b_{2C_1} \geq j > J_{2_0} + b_{2C_1}$ δηλαδή $8 \geq j > 7$

$\delta_{2C_2}(j) = 1$ όταν $J_{2_2} + b_{2C_2} \geq j > J_{2_1} + b_{2C_2}$ δηλαδή $8 \geq j > 7$

Το J_{1_2} και το J_{2_2} δεν έχουν οριστεί αλλά, όπως αναφέρθηκε και στο θεωρητικό μέρος, ως ανώτατο φράγμα θεωρείται η χωρητικότητα C. Συνεπώς, προκειμένου να διασφαλιστεί ότι η τιμή δεν υπερβαίνει το όριο αυτό, εφαρμόζονται οι ακόλουθες πράξεις:

$J_{1_2} + b_{1C_2} = 8 \Rightarrow J_{1_2} = 8 - 1 = 7$ και $J_{2_2} + b_{2C_2} = 8 \Rightarrow J_{2_2} = 8 - 2 = 6$

$q(0)$ = 1

$q(1)$ =0

$q(2) = 0$

$3q(3) = 3q(0) + 0 + 0 + 0 + 0 + 0$ άρα $q(3) = 1$

$4q(4) = 3q(1) + 4q(0) + 0 + 0 + 0 + 0$ άρα $q(4) = 1$

$5q(5) = 3q(2) + 4q(1) = 0$ άρα $q(5) = 0$

$6q(6) = 3q(3) + 4q(2) = 3$ άρα $q(6) = 0.5$

$7q(7) = 3q(4) + 4q(3) = 7$ άρα $q(7) = 1$

$8q(8) = 3q(5) + 4q(4) + 3q(6) + 3q(7) + 4q(5) + 4q(6) = 4 + 1.5 + 3 + 2$ άρα $q(8) = 10.5/8$

$G = 5.8125$

$$B_{1c_2} = \sum_{j=C-b_{1c_2}+1}^{C} G^{-1}q(j) = \frac{q(8)}{G} = 0.225806$$

$$B_{2c_2} = \sum_{j=C-b_{2c_2}+1}^{C} G^{-1}q(j) = \frac{q(7)+q(8)}{G} = 0.3978495$$

Δεσμευμένες πιθανότητες απώλειας κλήσης είναι οι εξής:

$B^*_{1c_2} = \frac{B_{1c_2}}{Prob(j>C-b_1)} = 0.466667$, όπου $J_{1_0} = C - b_1 = 8 - 3 = 5$

$B^*_{2c_2} = \frac{B_{2c_2}}{Prob(j>C-b_2)} = 0.822222$, όπου $J_{2_0} = C - b_2 = 8 - 4 = 4$

$U = 4.73118$

9.3 Το μοντέλο *CDTM* υπό την πολιτική *BR*

Όπως παρατηρείται και στα μοντέλα *MRM/BR* και *MTM/BR*, στο *CDTM* προβλέπεται η ύπαρξη ορισμένων b.u. που δεσμεύονται εκ των προτέρων σε συγκεκριμένες κατηγορίες. Η δέσμευση αυτή πραγματοποιείται εις βάρος των αντίστοιχων κατηγοριών, προκειμένου να διασφαλιστεί η δυνατότητα εξυπηρέτησης άλλων κατηγοριών κλήσεων που έχουν μεγαλύτερη ανάγκη. Επιπλέον, κάθε κατηγορία κλήσεων μπορεί να διαθέτει το δικό της επίπεδο κράτησης.

Αναλυτικό Μοντέλο

Ο αναδρομικός τύπος είναι [2]:

$$q(j)=\begin{cases}1 & \alpha\nu\ j=0\\ \frac{1}{j}\sum_{k=1}^{K}\alpha_k D_k(j-b_k)\delta_k(j)q(j-b_k)+\sum_{k=1}^{K}\sum_{s=1}^{S(k)}a_{kc_s}D_{kc_s}(j-b_{kc_s})\delta_{kc_s}(j)q(j-b_{kc_s}) & \alpha\nu\ 1\le j\le C\\ 0 & \alpha\nu\ j<0\ \text{ή}\ j>C\end{cases} \quad (9.11)$$

όπου:

$\delta_k(j)$, με βάση τον τύπο (9.7)

$\delta_{kc_s}(j)$, με βάση τον τύπο (9.8)

$D_k(j-b_k)$, με βάση τον τύπο (4.9)

$D_{kc_s}(j-b_{kc_s}) = b_{kc_s}$, *(αν* $j \le C$ - tr_k*)* (9.12)

$a_{kc_s}=\frac{\lambda_k}{\mu_{kc_s}}$, με βάση τον τύπο (8.9)

a_k , με βάση τον τύπο (4.2)

Ο τύπος της πιθανότητας απώλειας κλήσης B_k κατηγορίας k με b_k *b.u.* είναι ο (4.10)

Ο τύπος της πιθανότητας απώλειας κλήσης $B_{kc_{s(k)}}$ κατηγορίας k με $b_{kc_{s(k)}}$ $b.u.$ είναι [2]:

$$B_{kc_{s(k)}}=\sum_{j=C-b_{kc_{s(k)}}-tr_k+1}^{C} G^{-1}q(j) \quad (9.13)$$

με σταθερά κανονικοποίησης G από τον τύπο (4.3)

Ο τύπος της δεσμευμένης πιθανότητας απώλειας κλήσης $B^*_{kc_{s(k)}}$ κατηγορίας k με $b_{kc_{s(k)}}$ $b.u.$ όταν $j >$ $J_{k_{s(k)-1}}$ είναι [2]:

$$B^*_{kc_{s(k)}}=\frac{\sum_{j=C-b_{kc_{s(k)}}-tr_k+1}^{C} q(j)}{\sum_{j=J_{k_{s(k)-1}}+1}^{C} q(j)} \quad (9,14)$$

Παράδειγμα 9.4

Έστω ότι έχουμε 2 κατηγορίες κίνησης ($K = 2$) σε ένα σύστημα με χωρητικότητα $C = 8$ *b.u.* με $\lambda_1 = \lambda_2 = 1$, $\mu_1 = \mu_2 = 1$ (άρα $\alpha_1 = \alpha_2 = 1$ *erl*) και $b_1 = 3$ *b.u.*, $b_2 = 4$ *b.u.* Τα κατώφλια $J_{1_0} = 5$ $J_{1_1} = 6$ και $J_{2_0} = 4$ $J_{2_1} = 5$.

Τα ενδιάμεσα b, μ^{-1}, α ανά κατηγορία και κατώφλι είναι τα εξής:

$(b_{1C_1},\ \mu^{-1}_{1C_1},\ \alpha_{1C_1}) = (2, 1.5, 1.5)$ $\quad$ $(b_{1C_2},\ \mu^{-1}_{1C_2},\ \alpha_{1C_2}) = (1, 3, 3)$

$(b_{2C_1},\ \mu_{2C_1}^{-1}, \alpha_{2C_1})$ = (3, 4/3, 4/3) $(b_{2C_2},\ \mu_{2C_2}^{-1}, \alpha_{2C_2})$ = (2, 2, 2)

(Το συγκεκριμένο παράδειγμα είναι ανάλογο με το παράδειγμα 9.3 με τη διαφορά ότι εισάγεται η κράτηση tr_1=1 *b.u.*, ώστε να ισχύει $b_{1C_2} + tr_1 = b_{2C_2}$, προκειμένου να επιτευχθεί εξισορρόπηση του συστήματος.)

Ο αναδρομικός τύπος είναι:

$$jq(j) = \alpha_1 D_1(j-b_1)\delta_1(j)q(j-b_1) +$$
$$\alpha_2 b_2 \delta_2(j)q(j-b_2) +$$
$$\alpha_{1C_1} D_{1C_1}(j-b_{1C_1})\delta_{1C_1}(j)q(j-b_{1C_1}) +$$
$$\alpha_{1C_2} D_{1C_2}(j-b_{1C_2})\delta_{1C_2}(j)q(j-b_{1C_2}) +$$
$$\alpha_{2C_1} b_{2C_1}\delta_{2C_1}(j)q(j-b_{2C_1}) +$$
$$\alpha_{2C_2} b_{2C_2}\delta_{2C_2}(j)q(j-b_{2C_2})$$

με : $\delta_1(j)$ = 1 όταν $j \le J_{1_0} + b_1$ δηλαδή $j \le 8$ και $b_{1C_1} > 0$

$\delta_2(j)$ = 1 όταν $j \le J_{2_0} + b_2$ δηλαδή $j \le 8$και $b_{2C_1} > 0$

$\delta_{1C_1}(j) = 1$ όταν $J_{1_1} + b_{1C_1} \ge j > J_{1_0} + b_{1C_1}$ δηλαδή $8 \ge j > 7$

$\delta_{1C_2}(j) = 1$ όταν $J_{1_2} + b_{1C_2} \ge j > J_{1_1} + b_{1C_2}$ δηλαδή $8 \ge j > 7$

$\delta_{2C_1}(j) = 1$ όταν $J_{2_1} + b_{2C_1} \ge j > J_{2_0} + b_{2C_1}$ δηλαδή $8 \ge j > 7$

$\delta_{2C_2}(j) = 1$ όταν $J_{2_2} + b_{2C_2} \ge j > J_{2_1} + b_{2C_2}$ δηλαδή $8 \ge j > 7$

$D_1(j-b_1) = b_1$ όταν $j \le C - tr_1 = 7$ και

$D_{1C_1}(j-b_{1C_1}) = b_{1C_1}$ όταν $j \le C - tr_1 = 7$

$D_{1C_2}(j-b_{1C_2}) = b_{1C_2}$ όταν $j \le C - tr_1 = 7$

$q(0) = 1$

$q(1) = 0$

$q(2) = 0$

$3q(3) = 3q[0] + 0 + 0 + 0 + 0 + 0$ άρα $q(3) = 1$

$4q(4) = 3q(1) + 4q(0) + 0 + 0 + 0 + 0$ άρα $q(4) = 1$

$5q(5) = 3q(2) + 4q(1) = 0$ άρα $q(5) = 0$

$6q(6) = 3q(3) + 4q(2) = 3$ άρα $q(6) = 0.5$

$7q(7) = 3q(4) + 4q(3) = 7$ άρα $q(7) = 1$

$8q(8) = 0 + 4q(4) + 0 + 0 + 4q(5) + 4q(6) = 6$ άρα $q(8) = 0.75$

$G = 5.25$

$$B_{1c_2} = \sum_{j=C-b_{1c_2}-tr_1+1}^{C} G^{-1}q(j) = \frac{q(7)+q(8)}{G} = 0.33333$$

$$B_{2c_2} = \sum_{j=C-b_{2c_2}-tr_1+1}^{C} G^{-1}q(j) = \frac{q(7)+q(8)}{G} = 0.33333$$

Οι δεσμευμένες πιθανότητες απώλειας κλήσης είναι οι εξής:

$B^*_{1c_2} = \frac{B_{1c_2}}{Prob(j>C-b_1-tr_1)} = \frac{B_{1c_2}}{Prob(j>J_{1_0})} = 0.77777$, όπου $J_{1_0} = C - b_1 - tr_1 = 8 - 3 - 1 = 4$

$B^*_{2c_2} = \frac{B_{2c_2}}{Prob(j>C-b_2)} = \frac{B_{2c_2}}{Prob(j>J_{2_0})} = 0.77777$, όπου $J_{2_0} = C - b_2 = 8 - 4 = 4$

(Σημείωση : Εδώ έτυχε το $q(5)=0$)

και $U = 4.38095$

9.4 Επεξήγηση των προγραμμάτων του *CDTM*

Τα τμήματα κώδικα των αντίστοιχων προγραμμάτων, τόσο του αναλυτικού μοντέλου όσο και της προσομοίωσης, παρουσιάζονται στο *Παράρτημα Ζ*.

9.4.1 Επεξήγηση του προγράμματος του αναλυτικού μοντέλου

Όπως και στα αντίστοιχα προγράμματα των μοντέλων *EMLM, SRM/MRM* και *STM/MTM* εφαρμόζεται το αναλυτικό μοντέλο, δηλαδή ο αναδρομικός τύπος του *Roberts*, υλοποιημένος σε γλώσσα *Python*.

Όσον αφορά το γραφικό περιβάλλον της εφαρμογής αξίζει να σημειωθεί η εξής ιδιαιτερότητα:

Τα κατώφλια εισάγονται από τον χρήστη με τη μορφή *x:y,* όπου *x* αντιστοιχεί στην κατηγορία κλήσης και *y* στην αντίστοιχη τιμή του κατωφλίου από το σύνολο κατωφλίων της κατηγορίας.

Για όλα τα υπόλοιπα στοιχεία που ο χρήστης καλείται να εισάγει ισχύουν οι ίδιες προϋποθέσεις με εκείνες που εφαρμόζονται στα προγράμματα των μοντέλων *STM/MTM*.

9.4.2 Επεξήγηση του προγράμματος της προσομοίωσης

Η φιλοσοφία της προσομοίωσης στο μοντέλο *CDTM* ακολουθεί την ίδια λογική με αυτή του μοντέλου *MTM*, με τη διαφορά ότι κάθε κατηγορία κλήσεων μπορεί να διαθέτει το δικό της σύνολο κατωφλίων. Υπενθυμίζεται ότι στο *MTM* μοντέλο υπήρχε ένα ενιαίο σύνολο κατωφλίων για όλες τις κατηγορίες.

Συνεπώς, κατά ανάλογο τρόπο, όταν οι κλήσεις εισέρχονται στο σύστημα καταγράφονται οι χρόνοι άφιξής τους και, επαναληπτικά για το σύνολο των κλήσεων, εκτελούνται οι ακόλουθες διαδικασίες:

Αρχικά, εξετάζεται από το σωρό το σύνολο των κλήσεων για να προσδιοριστεί ποιες έχουν ολοκληρωθεί χρονικά, προκειμένου να αποδεσμευτούν τα *b.u.* που είχαν δεσμευθεί. Σε αντίθεση με τα μοντέλα *STM* και *MTM*, στο *CDTM* υπάρχουν διαφορετικά σύνολα από κατώφλια που καθορίζουν τη συμπεριφορά των κλήσεων ανάλογα με την κατηγορία στην οποία ανήκουν. Τα σύνολα αυτά μπορεί να περιλαμβάνουν είτε ένα μόνο κατώφλι είτε πολλαπλά, ανάλογα με τις παραμέτρους που ορίζει ο χρήστης.

Κάθε κατηγορία κλήσεων συμπεριφέρεται ανάλογα στα διαστήματα του συνόλου κατωφλίων που της αντιστοιχεί και, φυσικά, σύμφωνα με τις τιμές των *b.u.* και μ^{-1} που έχουν καθοριστεί από τον χρήστη. Σε περίπτωση που δεν έχουν οριστεί τιμές για τα αντίστοιχα διαστήματα, η κατηγορία αντιμετωπίζεται ως όμοια με το *EMLM* μοντέλο, δεδομένου ότι τα κατώφλια δεν επηρεάζουν τη συμπεριφορά της.

Σύμφωνα με τη λογική του μοντέλου, ανάλογα με τις απαιτήσεις της κλήσης, συγκεκριμένης κατηγορίας, σε *b.u.*, εφαρμόζεται η ακόλουθη διαδικασία:

Εάν βρίσκεται κάτω από το πρώτο κατώφλι του συνόλου κατωφλίων της αντίστοιχης κατηγορίας (κλειστό διάστημα), τότε αποδίδεται το αρχικό b.u. της κατηγορίας, με αρχικό χρόνο εξυπηρέτησης που καθορίζεται από το *μ.* Η κλήση καταχωρείται στο σωρό με βάση τον χρόνο αυτό.

Εάν η κλήση υπερβαίνει το πρώτο κατώφλι αλλά παραμένει κάτω από το δεύτερο (συμπεριλαμβανομένης και της τιμής αυτού), η εξυπηρέτηση πραγματοποιείται με μειωμένο b.u. και αντίστοιχα μεγαλύτερο χρόνο εξυπηρέτησης, ανάλογα με την κατηγορία της κλήσης. Η κλήση αποθηκεύεται στο σωρό με βάση τον νέο χρόνο εξυπηρέτησης.

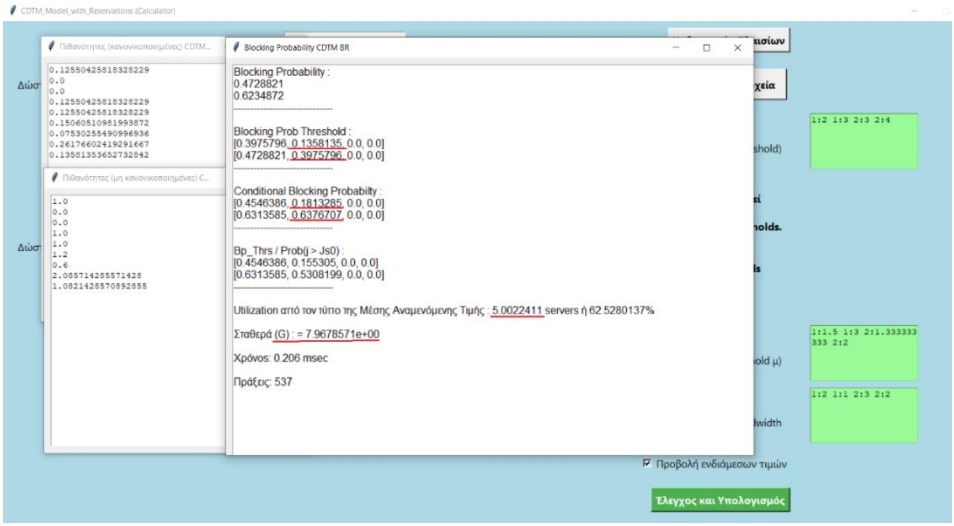
Η ίδια λογική επαναλαμβάνεται διαδοχικά για όλα τα διαστήματα του συνόλου κατωφλίων, μέχρι να φτάσει κανείς στο ανώτατο όριο χωρητικότητας C του συστήματος. Σε περίπτωση που εξαντληθούν όλα τα διαστήματα και η κλήση δεν μπορεί να εξυπηρετηθεί, αυτή θεωρείται χαμένη και δεν καταχωρείται κανένα χαρακτηριστικό της στον σωρό.

Σε κάθε διάστημα του συνόλου των κατωφλίων ανά κατηγορία κλήσεων διατηρείται μία αντίστοιχη μετρική που υπολογίζει την πιθανότητα απώλειας κλήσεων της κατηγορίας.

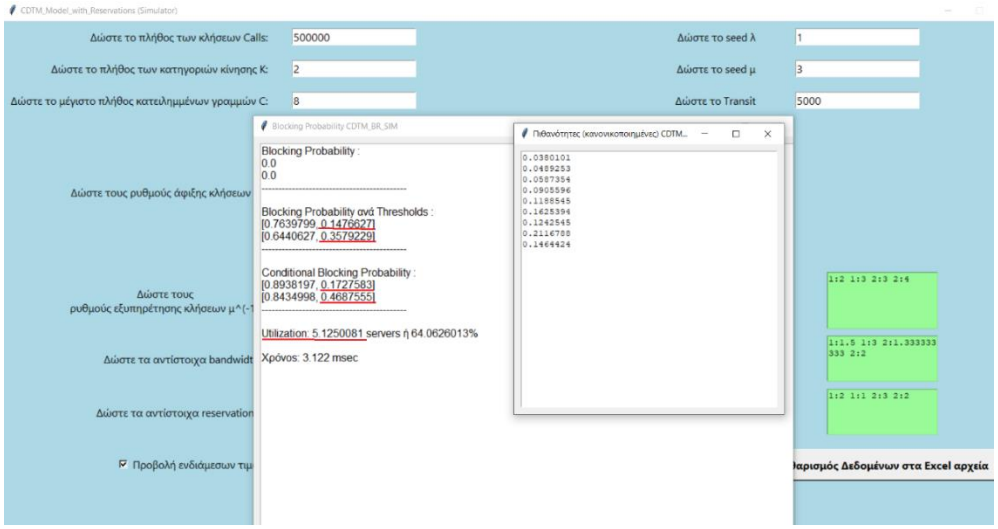
Όσον αφορά το γραφικό περιβάλλον της εφαρμογής ισχύουν οι ίδιες προδιαγραφές και απαιτήσεις με εκείνες που περιγράφονται στην Ενότητα 9.4.1.

9.5 Αποτελέσματα παραδειγμάτων μέσω των εφαρμογών

Τα αποτελέσματα των Παραδειγμάτων 9.1, 9.2, 9.3 και 9.4, όπως προκύπτουν από την εκτέλεση των κωδικών του αναλυτικού μοντέλου και της προσομοίωσης, παρουσιάζονται στις Εικόνες 31 και 32.



Εικόνα 31: Παράδειγμα 9.1 Αναλυτικό μοντέλο



Εικόνα 32: Παράδειγμα 9.1 Προσομοίωση

Με ανάλογη διαδικασία προκύπτουν τα αποτελέσματα και για τα υπόλοιπα παραδείγματα, τα οποία παρουσιάζονται στον Πίνακα 14.

Αναλυτικό μοντέλο	$B_{1_{c2}}$	$B_{2_{c2}}$	$B^*_{1_{c2}}$	$B^*_{2_{c2}}$	U
Παράδειγμα 9.1	0.1358135	0.3975796	0.1813285	0.6376707	5.0022411
Παράδειγμα 9.2	0.1635389	0.3994638	0.297561	0.7268293	4.9115282
Παράδειγμα 9.3	0.2258065	0.3978495	0.5675676 ή για MRM ($B_{1_{c2}}$/Prob(j>J_{1_0})) 0.4666667	0.8222222 ή για MRM ($B_{2_{c2}}$/Prob(j>J_{2_0})) 0.8222222	4.7311828
Παράδειγμα 9.4 (με tr_1 =1)	0.3333333	0.3333333	1.0 ή για MRM ($B_{1_{c2}}$/Prob(j>J_{1_0})) 0.7777778	0.7777778 ή για MRM ($B_{2_{c2}}$/Prob(j>J_{2_0})) 0.7777778	4.3809524

Προσομοίωση	$B_{1_{c2}}$	$B_{2_{c2}}$
Παράδειγμα 9.1	0.146416±0.00117	0.358165±0.00164
	$B^*_{1_{c2}}$	$B^*_{2_{c2}}$
	0.171444 ±0.00115	0.4685495±0.0015
Παράδειγμα 9.2	$B_{1_{c2}}$	$B_{2_{c2}}$
	0.157852±0.00115	0.357242±0.0012
	$B^*_{1_{c2}}$	$B^*_{2_{c2}}$
	0.211482±0.00102	0.478606±0.0012
Παράδειγμα 9.3	$B_{1_{c2}}$	$B_{2_{c2}}$
	0.256309±0.00161	0.367513±0.00062
	$B^*_{1_{c2}}$	$B^*_{2_{c2}}$
	0.611906±0.0019	0.716315±0.001
Παράδειγμα 9.4 (με tr_1 =1)	$B_{1_{c2}}$	$B_{2_{c2}}$
	0.4457874±	0.321181±0.00121
	$B^*_{1_{c2}}$	$B^*_{2_{c2}}$
	1	0.721906±0.00383

Πίνακας 14: Αποτελέσματα παρ. 9.1, 9.2, 9.3, 9.4

Παρατηρούμε τα εξής:

- Τα αποτελέσματα της εφαρμογής του αναλυτικού μοντέλου διαφέρουν αισθητά από τα αποτελέσματα της προσομοίωσης, κάτι που είναι αναμενόμενο, δεδομένου ότι ο αναδρομικός τύπος αποκλίνει από τις πραγματικές τιμές. Η απόκλιση γίνεται πιο εμφανής λόγω του ότι οι τιμές των δεδομένων εισόδου είναι σχετικά μικρές.
- Στο Παράδειγμα 9.2, που εκτελείται ως *MTM* μοντέλο, διαπιστώνεται ότι και τα δύο προγράμματα παράγουν τα ίδια αποτελέσματα με τα αντίστοιχα αποτελέσματα των προγραμμάτων του *MTM* μοντέλου.
- Στο Παράδειγμα 9.3, που εκτελείται ως *MRM* μοντέλο, τα αποτελέσματα συμφωνούν με τα αποτελέσματα των αντίστοιχων προγραμμάτων του *MRM* μοντέλου.
- Το παράδοξο παρατηρείται στο Παράδειγμα 9.4, το οποίο είναι ουσιαστικά ίδιο με το Παράδειγμα 9.3, με τη μόνη διαφορά ότι έχει προστεθεί $tr_1 = 1$ *b.u.* στην κατηγορία 1. Στο συγκεκριμένο παράδειγμα, το πρόγραμμα του αναλυτικού

μοντέλου παρέχει αποτελέσματα συμβατά με το αντίστοιχο πρόγραμμα του *MRM* μοντέλου, ενώ η προσομοίωση εμφανίζει απόκλιση.

Λόγω αυτού του παραδόξου, προχωρήσαμε σε ανάλυση των καταστάσεων μετάβασης, όπου διαπιστώθηκε ότι το πλήθος τους περιορίζεται σε μόλις 20 καταστάσεις, οι οποίες είναι οι εξής:

n_1	n_{1c_1}	n_{1c_2}	n_2	n_{2c_1}	n_{2c_2}	j
0	0	0	0	0	0	0
0	0	0	0	0	1	2
0	0	0	0	0	2	4
0	0	0	0	1	0	3
0	0	0	0	1	1	5
0	0	0	0	2	0	6
0	0	0	0	2	1	8
0	0	0	1	0	0	4
0	0	0	1	0	1	6
0	0	0	1	0	2	8
0	0	0	1	1	0	7
0	0	0	2	0	0	8
1	0	0	0	0	0	3
1	0	0	0	0	1	5
1	0	0	0	0	2	7
1	0	0	0	1	0	6
1	0	0	0	1	1	8
1	0	0	1	0	0	7
2	0	0	0	0	0	6
2	0	0	0	0	1	8

Πίνακας 15: Πίνακας μετάβασης καταστάσεων παρ. 9.4

όπου *n* είναι οι καταστάσεις των κατηγοριών και *j* οι κατειλημμένες γραμμές

Με βάση τον Πίνακα 15 έχουμε:

- Αν *j*=5 και αφιχθεί μία κλήση της 1ης κατηγορίας , τότε δεδομένου ότι $J_{1_0} = 5$ η κλήση θα εξυπηρετηθεί με $b_1 = 3\ b.u.$ άρα :

  $$j + b_1 + tr_1 \leq C$$

  $5 + 3 + 1 \leq 8$
  $9 \leq 8$ αδύνατο

- Αν *j*=6 και αφιχθεί μία κλήση της 1ης κατηγορίας, τότε δεδομένου ότι $J_{1_1} = 6$, η κλήση θα εξυπηρετηθεί με $b_{1c_1} = 2\ b.u.$ άρα:

  $$j + b_{1c_1} + tr_1 \leq C$$

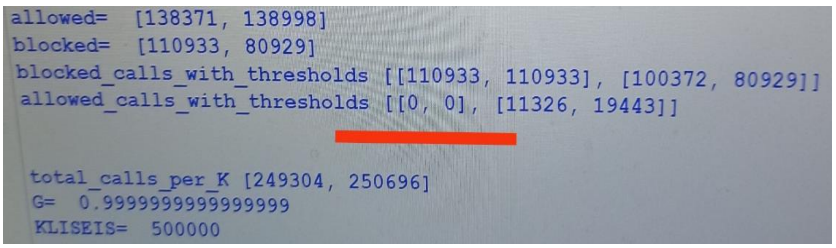
$6 + 2 + 1 \leq 8$
$9 \leq 8$ <u>αδύνατο</u>

- Αν j=7 και αφιχθεί μία κλήση της 1ης κατηγορίας, τότε δεδομένου ότι $J_{1_1} = 6$, η κλήση θα εξυπηρετηθεί με $b_{1c_2} = 1\ b.u.$ άρα :

$j + b_{1c_2} + tr_1 \leq C$
$7 + 1 + 1 \leq 8$
$9 \leq 8$ <u>αδύνατο</u>

Από τις μεταβάσεις παρατηρούμε ότι τα n_{1c_1} , n_{1c_2} λαμβάνουν μηδενικές τιμές, σε αντίθεση με τα n_{2c_1}, n_{2c_2}. Το γεγονός αυτό έχει ως αποτέλεσμα να μην παρατηρούνται τα ίδια ενδιάμεσα *B* στις δύο κατηγορίες, όπως θα αναμέναμε, δεδομένου ότι το συγκεκριμένο παράδειγμα θα έπρεπε να αποδίδει αποτελέσματα περίπου αντίστοιχα με το μοντέλο *MRM/BR*. Τα παραπάνω επιβεβαιώνονται και από τον προσομοιωτή *CDTM/BR*.

Λόγω αυτού του παράδοξου, προσαρμόστηκε η προσομοίωση ώστε να εξάγονται, μέσω της γραμμής εντολών της *Python*, τα πλήθη των κλήσεων ανά κατηγορία και ανά κατώφλι. Στην παρακάτω στιγμιαία απεικόνιση (Εικόνα 33) φαίνεται καθαρά ότι δεν υπάρχουν κλήσεις που γίνονται δεκτές στις ενδιάμεσες καταστάσεις της κατηγορίας 1, γεγονός που συμφωνεί με τα δεδομένα του πίνακα μεταβάσεων (Πίνακας 15).



Εικόνα 33: Αποτέλεσμα γραμμής εντολών Python παρ.9.4

Η παρατηρούμενη συμπεριφορά μπορεί να ερμηνευθεί ως εξής: Τα κατώφλια της κατηγορίας 1 εμφανίζονται υψηλά και πολύ κοντά μεταξύ τους, καθώς και κοντά στο ανώτατο φράγμα C. Επιπλέον, η κράτηση της κατηγορίας 1 λαμβάνει τιμή 1 (tr_1 = 1 *b.u.*). Οι παράγοντες αυτοί συνεπικουρούν στην απόρριψη σχεδόν όλων των κλήσεων που βρίσκονται στα ενδιάμεσα διαστήματα των κατωφλίων της κατηγορίας 1.

Μετά από πειραματικές δοκιμές, παρατηρήθηκε ότι η μείωση όλων των κατωφλίων και των δύο κατηγοριών κατά μία μονάδα οδηγεί σε αποτελέσματα που είναι ιδιαίτερα κοντά σε εκείνα που προκύπτουν από το αντίστοιχο πρόγραμμα του μοντέλου *MRM/BR*.

Παράδειγμα 9.5

Έστω ότι έχουμε τέσσερις κατηγορίες κλήσεων (*K*=4) και κλήσεις καταφθάνουν τυχαία *(*αφίξεις κατά *Poisson)* σε ένα σύστημα με χωρητικότητα *C*=300 *b.u.* Τα χαρακτηριστικά των κλήσεων δίνονται με τη μορφή διανυσμάτων:

λ = (8,6,4,1), μ^{-1}= (6 ,4,3,1) , b = (1,4,6,24)

$b_{3c} = (5\ ,4\ ,3,2)$ $\mu_{3c}^{-1} = (3.6\ ,4.5\ ,6,\ 9)$

$b_{4c} = (20\ ,16\ ,12,8)$ $\mu_{4c}^{-1} = (1.2\ ,1.5\ ,2,\ 3)$

και τα κατώφλια $J_3 = (276,\ 280,\ 284,\ 288)$ και $J_4 = (266,\ 270,\ 274,278)$.

Τα αποτελέσματα του Παραδείγματος 9.5 μετά την εκτέλεση των προγραμμάτων του αναλυτικού μοντέλου και της προσομοίωσης παρουσιάζονται στον Πίνακα 16

Η σκοπιμότητα του παραδείγματος έγκειται στο να εκτελεστούν τα προγράμματα με σχετικά υψηλές τιμές παραμέτρων, ώστε να αξιολογηθεί ότι, όταν το σύστημα λειτουργεί υπό αυξημένο φορτίο, δηλαδή υπό συνθήκες σημαντικής πίεσης, ο αναδρομικός τύπος του *Roberts* προσεγγίζει ικανοποιητικά τα αποτελέσματα της προσομοίωσης.

Κρίθηκε ότι δεν ήταν απαραίτητη η εκτέλεση πολλαπλών προσομοιώσεων για διαφορετικές τιμές των τυχαίων seeds προκειμένου να εκτιμηθούν οι αποκλίσεις των αποτελεσμάτων, όπως γίνεται στα προηγούμενα παραδείγματα με ακρίβεια τάξης 10^{-4}, διότι κάτι τέτοιο δεν θα τροποποιούσε ουσιαστικά τα συμπεράσματα της μελέτης. Για το λόγο αυτό, χρησιμοποιήθηκαν ενδεικτικές τιμές, όπως seed_λ = 1 και seed_μ = 3.

Αναλυτικό μοντέλο				
(λ_1 , λ_2 , λ_3 , λ_4)	**B1**	**B2**	B_{3c_4}	B_{4c_4}
(8,6,4,1)	0.003854	0.017234	0.0081231	0.0393905
(9, 7, 5, 2)	0.0297131	0.1192913	0.0605214	0.234777
(11, 9, 7, 4)	0.1030486	0.3596587	0.200170	0.5956014
(15, 13, 11, 7)	0.2104996	0.6189526	0.3834532	0.8576601
(19, 17, 15, 11)	0.2908102	0.7533751	0.5036015	0.9407204
(25, 23, 21, 17)	0.379089	0.8555473	0.6194542	0.979792
(30, 28, 26, 22)	0.4350303	0.9009385	0.6845247	0.990531

Προσομοίωση				
(λ_1 , λ_2 , λ_3 , λ_4)	**B1**	**B2**	B_{3c_4}	B_{4c_4}
(8,6,4,1)	0.0023924	0.0096967	0.0046854	0.0238589
(9, 7, 5, 2)	0.0254502	0.0986096	0.0512572	0.2059822
(11, 9, 7, 4)	0.1027035	0.3501911	0.1967637	0.5982847
(15, 13, 11, 7)	0.2138315	0.6182837	0.3839157	0.857628
(19, 17, 15, 11)	0.2938025	0.753033	0.5027404	0.9404448
(25, 23, 21, 17)	0.3830295	0.8551569	0.6160243	0.9801918
(30, 28, 26, 22)	0.439376	0.8999099	0.6823854	0.9903228

Πίνακας 16: Αποτελέσματα παρ. 9.5.

Για την εξισορρόπηση των πιθανοτήτων απώλειας θεωρούμε τις τιμές *tr* = (7, 4, 6, 0), οι οποίες αντιστοιχούν στις κρατήσεις των αντίστοιχων κατηγοριών κλήσεων. Οι τιμές αυτές επιλέγονται έτσι ώστε να ισχύει η ισότητα $b_1 + tr_1 = b_2 + tr_2 = b_{3c_4} + tr_3 = b_{4c_4} + tr_4 = 8$. Τα αποτελέσματα που προκύπτουν παρουσιάζονται στον *Πίνακα 17*:

Αναλυτικό μοντέλο				
(λ_1 , λ_2 , λ_3 , λ_4)	**B1**	**B2**	B_{3c_4}	B_{4c_4}
(8,6,4,1)	0.0193383	0.0193383	0.0193383	0.0193383

Προσομοίωση		
(λ_1 , λ_2 , λ_3 , λ_4)	**B1**	**B2**
(8,6,4,1)	0.01572±0.0007	0.015454±0.00041
	B_{3c_4}	B_{4c_4}
	0.015629±0.0008	0.015797±0.00066

Πίνακας 17: Αποτελέσματα παρ. 9.5 με *tr* = (7, 4, 6, 0).

10. Εφαρμογές σε σύγχρονα τηλεπικοινωνιακά δίκτυα

Τα μοντέλα απωλειών πολυδιάστατης κίνησης μπορούν και βρίσκουν εφαρμογή σε πολλά σύγχρονα τηλεπικοινωνιακά συστήματα όπως:

- Σε Κινητά Δορυφορικά Συστήματα Χαμηλής Τροχιάς *(Low Earth Orbit / Mobile Satellite System – **LEO / MSS)***, δηλαδή σε δίκτυα δορυφόρων που κινούνται σε χαμηλό ύψος γύρω από τη Γη και παρέχουν υπηρεσίες επικοινωνίας σε χρήστες που βρίσκονται οπουδήποτε - είτε στη στεριά, είτε στη θάλασσα, είτε στον αέρα.

  Σε τέτοια συστήματα με «κυψελοειδή δομή σταθερή ως προς τον δορυφόρο», *(Satellite Fixed Cells - SFC)*, τα οποία εξυπηρετούν νέες κλήσεις ή πραγματοποιούν μεταγωγή κλήσεων *(handover calls)* διαφορετικών κατηγοριών υπηρεσιών, προτείνονται μοντέλα που ακολουθούν πολιτική αποδοχής κλήσεων με κατώφλι για την αξιολόγηση της απόδοσης [14], [15].

- Σε προηγμένες τεχνολογίες οπτικών ινών, όπως τα Παθητικά Οπτικά Δίκτυα *(Passive Optical Networks – **PON**)*, συνδυάζονται δύο μέθοδοι πολυπλεξίας, ώστε να επιτυγχάνονται πολύ υψηλότερες ταχύτητες και μεγαλύτερη χωρητικότητα σε σύγκριση με τα παραδοσιακά δίκτυα.

  Η υβριδική πολυπλεξία βασίζεται στις τεχνικές *Time Division Multiplexing* και *Wavelength Division Multiplexing (TDM – WDM ή TWDM)*. Συγκεκριμένα, με τη *WDM* χρησιμοποιούνται πολλαπλά μήκη κύματος (δηλαδή διαφορετικά «χρώματα» φωτός) στην ίδια οπτική ίνα, όπου κάθε μήκος κύματος λειτουργεί ως ανεξάρτητο κανάλι επικοινωνίας. Αντίστοιχα, με την *TDM*, μέσα σε κάθε μήκος κύματος, πολλοί χρήστες μοιράζονται τη διαθέσιμη χωρητικότητα χρησιμοποιώντας διαφορετικές χρονικές θυρίδες *(time slots)*.

  Επομένως, για την αξιολόγηση της απόδοσης και της ποιότητας του συστήματος, απαιτείται η χρήση του μοντέλου *EMLM* και των επεκτάσεών του, για τον υπολογισμό των αντίστοιχων παραμέτρων [16].

- Με την εισαγωγή της τεχνολογίας ***5G***, η βιομηχανία των τηλεπικοινωνιών εισέρχεται σε μια φάση σημαντικών αλλαγών. Η ραγδαία αύξηση του φορτίου κίνησης, σε συνδυασμό με την εμφάνιση νέων απαιτήσεων δικτύου, οδήγησε στην ανάγκη ανάπτυξης μιας αρχιτεκτονικής ικανής να υποστηρίζει υψηλή αποδοτικότητα, βελτιωμένους ρυθμούς μετάδοσης και χαμηλή καθυστέρηση.

  Η αρχιτεκτονική *Cloud Radio Access Network (**C-RAN**)* μπορεί να ανταποκριθεί σε αυτές τις προκλήσεις, καθώς βασίζεται στον λειτουργικό διαχωρισμό του Σταθμού Βάσης *(Base Station – BS)*. Συγκεκριμένα, οι Μονάδες Βασικής Ζώνης *(Baseband Unit – BBU)*, οι οποίες είναι υπεύθυνες για λειτουργίες όπως η επεξεργασία σήματος και ο έλεγχος των πρωτοκόλλων δικτύου, αποσυνδέονται από τις Απομακρυσμένες Κεφαλές Ραδιοσυχνοτήτων *(Remote Radio Head – RRH)*, οι οποίες αναλαμβάνουν τη μετάδοση και λήψη σημάτων από τους χρήστες, καθώς και τη μετατροπή των ψηφιακών σημάτων σε ραδιοσυχνότητες.

  Στο παραδοσιακό μοντέλο, οι παραπάνω μονάδες συνυπήρχαν στον ίδιο σταθμό βάσης: οι *BBU* βρίσκονταν στη βάση του σταθμού, ενώ οι *RRH* τοποθετούνταν κοντά στην κεραία, στην κορυφή του πύργου.

  Στην προτεινόμενη αρχιτεκτονική, όλες οι μονάδες *BBU* συγκεντρώνονται σε ένα κεντρικό σημείο, καθιστώντας τους σταθμούς βάσης ελαφρύτερους, μικρότερους και πιο οικονομικούς, ενώ παράλληλα επιτρέπεται η σχεδόν άμεση επικοινωνία μεταξύ των *BBU*.

  Η επιτυχής υλοποίηση της αρχιτεκτονικής αυτής μπορεί να επιφέρει σημαντικά οφέλη στην υποστήριξη νέων απαιτήσεων και υπηρεσιών δικτύου, καθώς και να προσφέρει αυξημένη ευελιξία.

  Η σύγκριση των πιθανοτήτων απόρριψης κλήσεων *(CBP)* μπορεί να οδηγήσει σε χρήσιμα συμπεράσματα σχετικά με την αποδοτικότητα των μοντέλων που βασίζονται στην αρχιτεκτονική *C-RAN* και, κατ' επέκταση, στην αξιολόγηση της

συνολικής απόδοσης του δικτύου. Η ανάλυση αυτού του τύπου δικτύων πραγματοποιείται μέσω των Μοντέλων Επανάκλησης *(Retry Models)* [17].

- Στις μαζικές επικοινωνίες τύπου μηχανής *(massive Machine-Type Communications – **mMTC**)*, ένας πολύ μεγάλος αριθμός συσκευών — από χιλιάδες έως και εκατομμύρια — συνδέεται ταυτόχρονα στο δίκτυο. Οι συσκευές αυτές περιλαμβάνουν αισθητήρες (π.χ. θερμοκρασίας, υγρασίας), έξυπνους μετρητές (ηλεκτρικής ενέργειας, νερού), συσκευές του Διαδικτύου των Πραγμάτων *(Internet of Things – IoT)*, καθώς και συστήματα παρακολούθησης *(tracking)*, μεταξύ άλλων.

  Τα *mMTC* αποτελούν ένα από τα βασικά σενάρια εφαρμογής των κυψελωτών δικτύων πέμπτης γενιάς (*5G*), καθώς και των επόμενων γενεών. Στο πλαίσιο αυτό, ανακύπτει η σημαντική τεχνική πρόκληση της υποστήριξης ενός εξαιρετικά μεγάλου αριθμού συσκευών *MTC* σε κυψελωτά δίκτυα. Ζητήματα όπως η αποδοτική εκτίμηση καναλιού, η ανίχνευση των ενεργών χρηστών και η αξιόπιστη ανίχνευση των μεταδιδόμενων δεδομένων σε ένα τέτοιο περιβάλλον παραμένουν ανοικτά ερευνητικά προβλήματα.

  Κατά συνέπεια, απαιτείται η ανάπτυξη κατάλληλων τεχνικών για την αντιμετώπιση των προκλήσεων που σχετίζονται με την εκτίμηση καναλιού, την ανίχνευση δραστηριότητας και την ανίχνευση δεδομένων, με στόχο τη βελτίωση της απόδοσης των συστημάτων και την ανάδειξη νέων κατευθύνσεων έρευνας [18].

- Στην Επικοινωνία Υπερυψηλής Αξιοπιστίας και Χαμηλής Καθυστέρησης *(Ultra-Reliable Low Latency Communication – **URLLC**)*, η οποία αποτελεί ένα από τα βασικά σενάρια του *5G*, περιλαμβάνονται εφαρμογές όπου τα δεδομένα πρέπει να μεταδίδονται με εξαιρετικά μικρή καθυστέρηση *(latency)* και σχεδόν απόλυτη αξιοπιστία, δηλαδή με εξαιρετικά χαμηλή πιθανότητα αποτυχίας.

  Το *URLLC* επιτρέπει την υποστήριξη κρίσιμων εφαρμογών πραγματικού χρόνου, με σχεδόν μηδενικά σφάλματα, όπως για παράδειγμα η τηλεχειρουργική. Επομένως, δίνεται ιδιαίτερη έμφαση στην ανάπτυξη κατάλληλων στατιστικών μεθοδολογιών

για τον σχεδιασμό και την αξιολόγηση συστημάτων που απαιτούν εξαιρετικά υψηλά επίπεδα αξιοπιστίας [19].

- Στο ενισχυμένο κινητό ευρυζωνικό δίκτυο *(enhanced Mobile Broadband – **eMBB**)*, το οποίο αποτελεί βασικό σενάριο του *5G*, δίνεται έμφαση στην παροχή πολύ υψηλών ταχυτήτων μετάδοσης δεδομένων και μεγάλης χωρητικότητας.

  Το *5G* είναι σε θέση να υποστηρίξει ένα ευρύ φάσμα υπηρεσιών και απαιτήσεων χρηστών. Οι απαιτήσεις αυτές ικανοποιούνται μέσω λειτουργιών όπως ο χρονοπρογραμματισμός *(scheduling)*, η προσαρμογή σύνδεσης *(link adaptation)* και ο έλεγχος ισχύος *(power control)* [20].

  Τα μοντέλα που μελετήσαμε συνδέονται στενά με τις λειτουργίες αυτές, συμβάλλοντας καθοριστικά στην αποδοτική αξιοποίηση των διαθέσιμων πόρων του συστήματος.

- Σε **δίκτυα μικροκυψελών πέμπτης γενιάς** *(5G)*, η συνεργατική μετάδοση *(cooperative transmission)* αποτελεί μια αποτελεσματική προσέγγιση για τις ασύρματες επικοινωνίες με χρήση κινητών μέσων *(vehicles)*, με στόχο τη βελτίωση της χωρητικότητας μετάδοσης και της αξιοπιστίας.

  Με βάση τις αποστάσεις μεταξύ των κινητών μέσων και των συνεργαζόμενων σταθμών βάσης μικροκυψελών *(Base Stations)*, έχουν εξαχθεί η πιθανότητα συνεργασίας *(cooperative probability)* και η πιθανότητα κάλυψης *(coverage probability)* για συνεργατικά δίκτυα μικροκυψελών *5G*, όπου οι σταθμοί βάσης ακολουθούν κατανομές *Poisson*.

  Για την εξισορρόπηση της χωρητικότητας επικοινωνίας των κινητών μέσων και του ρυθμού μεταπομπής *(handoff rate)*, δηλαδή του ρυθμού με τον οποίο ένα κινητό μέσο αλλάζει σταθμό βάσης κατά την κίνησή του στο δίκτυο, μπορεί να επιτευχθεί ένας βέλτιστος βαθμός επιβάρυνσης μέσω της κατάλληλης ρύθμισης του κατωφλίου

συνεργασίας *(cooperative threshold)* στα συνεργατικά δίκτυα μικροκυψελών *5G* [21].

- Στα συστήματα ανίχνευσης βασισμένα σε ανωμαλίες *(Anomaly-based Detection Systems – **ADSs**)*, επιχειρείται η εκμάθηση των χαρακτηριστικών συμπεριφορών και γεγονότων ενός συστήματος ή/και των χρηστών του κατά τη διάρκεια μιας χρονικής περιόδου, με σκοπό τη δημιουργία ενός προφίλ φυσιολογικής λειτουργίας.

  Ένας σημαντικός παράγοντας για τα *ADSs* είναι η ικανότητά τους να διακρίνουν μεταξύ φυσιολογικών και μη φυσιολογικών συμπεριφορών σε μια δεδομένη χρονική περίοδο. Ωστόσο, η διαδικασία αυτή καθίσταται ολοένα και πιο δύσκολη λόγω της δυναμικής φύσης των δικτυακών περιβαλλόντων, τα οποία μεταβάλλονται συνεχώς.

  Η χρήση ενός σταθερού κατωφλίου *(threshold)* για τη διάκριση μεταξύ φυσιολογικής και μη φυσιολογικής συμπεριφοράς σε ένα τέτοιο περιβάλλον ενέχει κινδύνους, καθώς δεν διασφαλίζεται ότι το κατώφλι αυτό θα αντιστοιχεί πάντοτε σε φυσιολογικές συνθήκες.

  Βασιζόμενοι στην υπόθεση της στατιστικής ανάλυσης των *ADSs* — σύμφωνα με την οποία τα φυσιολογικά δεδομένα εμφανίζονται σε περιοχές υψηλής πιθανότητας, ενώ τα κακόβουλα δεδομένα σε περιοχές χαμηλής πιθανότητας ενός στοχαστικού μοντέλου — μπορούν να οριστούν δύο προσαρμοστικά κατώφλια για τη διάκριση μεταξύ φυσιολογικών και μη φυσιολογικών συμπεριφορών.

  Οι συμπεριφορές που εντοπίζονται μεταξύ των δύο αυτών κατωφλίων χαρακτηρίζονται ως ύποπτες και αξιολογούνται περαιτέρω με τη βοήθεια ενός συστήματος διαχείρισης εμπιστοσύνης [22].

- Στην Πολυπλεξία Ορθογώνιας Διαίρεσης Συχνότητας με Πολλαπλή Πρόσβαση *(Orthogonal Frequency Division Multiple Access – **OFDMA**)*, το διαθέσιμο φάσμα συχνοτήτων χωρίζεται σε πολλές μικρές υποσυχνότητες, οι οποίες κατανέμονται

δυναμικά σε διαφορετικούς χρήστες. Με τον τρόπο αυτό, πολλοί χρήστες μπορούν να επικοινωνούν ταυτόχρονα χωρίς αμοιβαίες παρεμβολές. Πρόκειται για μια τεχνολογία που χρησιμοποιείται ευρέως σε δίκτυα όπως το *4G* και το *5G*, επιτρέποντας την αποδοτική κατανομή των διαθέσιμων πόρων του ασύρματου φάσματος.

Στα δίκτυα αυτά έχουν αναπτυχθεί αναλυτικές μέθοδοι για την αξιολόγηση της Ποιότητας Υπηρεσίας *(Quality of Service – QoS)* που αντιλαμβάνονται οι χρήστες στην κατερχόμενη ζεύξη *(downlink)*, ιδίως σε περιβάλλοντα που εξυπηρετούν τόσο ροές συνεχούς μετάδοσης *(streaming)* όσο και ελαστική *(elastic)* κίνηση

.

Η πιθανότητα αποκλεισμού *(blocking probability)* για χρήστες *streaming* μπορεί να υπολογιστεί με τη χρήση του αλγορίθμου *Kaufman–Roberts* [23].

Βιβλιογραφία

1. M. Stasiak, M. Glabowski, A.Wisniewski, and P. Zwierzykowski, Modeling and Dimensioning of Mobile Networks, John Wiley, 2011.
2. I. Moscholios and M. Logothetis, Efficient Multirate Teletraffic Loss Models Beyond Erlang, Wiley & IEEE Press, April 2019.
3. F. A. Cruz-Pérez, S. L. Castellanos-López and G. Hernández-Valdez, "Queueing Systems With Fractional Number of Servers: Analysis and Practical Implementation of the Erlang-B Traffic Model," IEEE Access, vol. 12, pp. 166268-166280, 2024.
4. H. Yang, J. Fu, J. Wu and M. Zukerman, "A study of a loss system with priorities", Heliyon, vol. 10, issue 16, 2024.
5. H. Akimaru and K. Kawashima, Teletraffic – Theory and Applications, 2nd ed., Springer, Berlin, 1999.
6. K. Trivedi and A. Bobbio, Reliability and Availability Engineering: Modeling, Analysis, and Application, Cambridge Univ. Press, 2017.
7. Y. Ma, J. Han and K. Trivedi, "Composite performance and availability analysis of communications networks. A comparison of exact and approximate approaches," Proc. IEEE Globecom, San Francisco, CA, USA, Nov. 2000.
8. Yue Ma, J. Han and K. Trivedi, "Composite performance and availability analysis of wireless communication networks," IEEE Trans. Veh. Tech., vol. 50, no. 5, pp. 1216-1223, Sept. 2001.
9. A. Bobbio and K. Trivedi, "An Aggregation Technique for the Transient Analysis of Stiff Markov Chains," IEEE Trans. Computers, vol. C-35, no. 9, pp. 803-814, Sept. 1986.
10. Stewart, W. J. (2009). *Probability, Markov Chains, Queues, and Simulation*. Princeton: Princeton University Press.
11. H. Halstrom E. Brockmeyer and A. Jensen, *"The Life and Works of A.K.Erlang"*. Transactions of the Danish Academy of Technical Sciences, No.2, 1948.
12. K. Lolis, M. Vlasakis, I. D. Moscholios, I. Keramidi, D. Uzunidis, G. Bouloukakis, N. Tselikas and M. Logothetis, "On Blocking Probabilities in an Erlang Loss System with Server Failures and a Single Repair Facility", accepted for publication in IEEE/IET CSNDSP, Edinburgh, U.K., 15-17 July 2026.
13. K. Lolis, M. Vlasakis, I. D. Moscholios, I. Keramidi, D. Uzunidis, G. Bouloukakis, N. Tselikas and M. Logothetis, "Performance evaluation of an Erlang loss system with server failures and a single repair facility", accepted for publication in the Panhellenic Conf. Electronics and Telecommunications (PACET), University of Patras, Patras, Greece, 23-24 April 2026.
14. I. D. Moscholios, V. G. Vassilakis, N. C. Sagias and M. D. Logothetis, "On Channel Sharing Policies in LEO Mobile Satellite Systems," in *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, no. 4, pp. 1628-1640, Aug. 2018, doi: 10.1109/TAES.2018.2798318.
15. Z. Wang, P. T. Mathiopoulos and R. Schober, "Channeling Partitioning Policies for Multi-Class Traffic in LEO-MSS," in *IEEE Transactions on Aerospace and Electronic Systems*, vol. 45, no. 4, pp. 1320-1334, Oct. 2009, doi: 10.1109/TAES.2009.5310301.
16. J. S. Vardakas, V. G. Vassilakis and M. D. Logothetis, "Blocking Analysis in Hybrid TDM-WDM PONs Supporting Elastic Traffic," *2008 Fourth Advanced International Conference on Telecommunications*, Athens, Greece, 2008, pp. 371-376, doi: 10.1109/AICT.2008.76.]
17. E. Mpantola, J. Vardakas and M. Louta, "Performance evaluation of Cloud Radio Access Networks by jointly considering communicational and computational network resources," *2021 6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, Preveza, Greece, 2021 , pp. 1-5, doi: 10.1109/SEEDA-CECNSM53056.2021.9566269.
18. R. B. Di Renna, C. Bockelmann, R. C. de Lamare and A. Dekorsy, "Detection Techniques for Massive Machine-Type Communications: Challenges and Solutions," in *IEEE Access*, vol. 8, pp. 180928-180954, 2020, doi: 10.1109/ACCESS.2020.3027523.
19. P. Popovski *et al*., "Wireless Access in Ultra-Reliable Low-Latency Communication (URLLC)," in *IEEE Transactions on Communications*, vol. 67, no. 8, pp. 5783-5801, Aug. 2019, doi: 10.1109/TCOMM.2019.2914652.

20. M. I. Saglam and M. Kartal, "5G Enhanced Mobile Broadband Downlink Scheduler," *2019 11th International Conference on Electrical and Electronics Engineering (ELECO)*, Bursa, Turkey, 2019, pp. 687-692, doi: 10.23919/ELECO47770.2019.8990378.
21. X. Ge, H. Cheng, G. Mao, Y. Yang and S. Tu, "Vehicular Communications for 5G Cooperative Small-Cell Networks," in *IEEE Transactions on Vehicular Technology*, vol. 65, no. 10, pp. 7882-7894, Oct. 2016, doi: 10.1109/TVT.2016.2539285.
22. Y. Chae, N. Katenka and L. DiPippo, "An Adaptive Threshold Method for Anomaly-based Intrusion Detection Systems," *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, Cambridge, MA, USA, 2019, pp. 1-4, doi: 10.1109/NCA.2019.8935045.
23. M. K. Karray, "Analytical evaluation of QoS in the downlink of OFDMA wireless cellular networks serving streaming and elastic traffic," in *IEEE Transactions on Wireless Communications*, vol. 9, no. 5, pp. 1799-1807, May 2010, doi: 10.1109/TWC.2010.05.091501.
24. Λογοθέτης, Μ., (2018). Θεωρία Τηλεπικοινωνιακής Κινήσεως και Εφαρμογές (3η έκδ.). Αθήνα: Κλειδάριθμος.
25. Μπερτσεκάς, Δ.Π. & Τσιτσικλής, Γ.Ν. (2013). Εισαγωγή στις Πιθανότητες με Στοιχεία Στατιστικής (2η έκδ.) Θεσσαλονίκη: Εκδόσεις Τζιόλα.
26. Μπουκουβάλας, Α., Μοσχολιός, Ι. Προχωρημένα Θέματα Δικτύων. Πανεπιστήμιο Πελοποννήσου: Φεβρουάριος 2013.
27. Παναγούλιας, Π. (2016). Υπολογισμός απωλειών κλήσεων πολυδιάστατης ψευδοτυχαίας κίνησης στη διασύνδεση X2 δικτύου τεχνολογίας LTE (Μεταπτυχιακή εργασία). Ελληνικό Ανοικτό Πανεπιστήμιο, Πάτρα.
28. Γουρνάρη, Ν. (2017). Η πιθανοτική κατανομή του κλιμακογράμματος με χρήση τεχνικής Monte Carlo. (Διπλωματική Εργασία). Εθνικό Μετσόβιο Πολυτεχνείο, Αθήνα.

Παράρτημα Α: Η γλώσσα προγραμματισμού *Python*

Η *Python* είναι μια σύγχρονη γλώσσα προγραμματισμού που δημιουργήθηκε από τον *Ολλανδό Guido van Rossum* το 1989 και το όνομά της είναι εμπνευσμένο από τους *Monty Python (ή The Pythons),* μια βρετανική κωμική ομάδα με ιδιότυπο χιούμορ που το χαρακτήριζε ο σουρεαλισμός και η πολιτική σάτιρα.

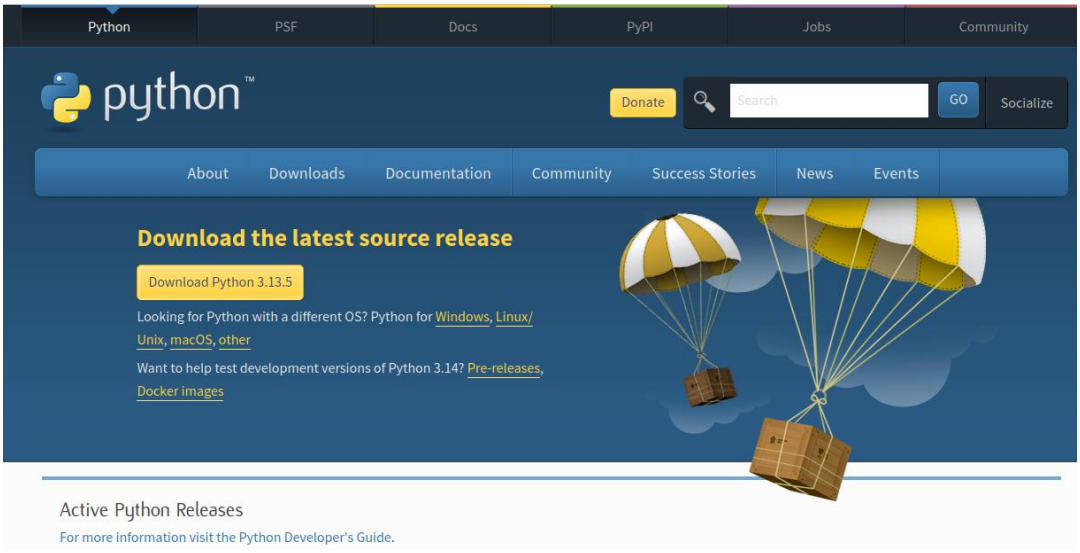
Είναι διαθέσιμη σε 2 εκδόσεις:

• *Python 2.x* (πρώτη έκδοση Οκτώβρης 2000) με τελευταίες εκδόσεις 2.6 & 2.7
• *Python 3.x* (πρώτη έκδοση Δεκέμβριος 2008)

ΠΡΟΣΟΧΗ! Οι *Python 2.x* και *Python 3.x* είναι μη συμβατές.

Βασικές οδηγίες εγκατάστασης της Python σε Windows

Πρώτα πρέπει να επισκεφθούμε την επίσημη ιστοσελίδα η οποία είναι (Εικόνα 34):
https://www.python.org/downloads/



Εικόνα 34: *Python Website*

Έπειτα, κάνουμε κλικ στο **"Download Python 3.x.x"** την πιο πρόσφατη έκδοση. Το αρχείο που θα κατέβει θα έχει περίπου την ονομασία **python-3.x.x.exe** .

Μετά, εγκαθιστούμε την *Python* **(για Windows 8, 10 ή 11)** κάνοντας διπλό κλικ στο αρχείο **.exe** που κατεβάσαμε.

Προσοχή! Για Windows 7, πρέπει να εγκατασταθεί παλαιότερη έκδοση.

Πριν την εγκατάσταση, καλό θα ήταν να τσεκάρουμε κάτω από το κουμπί **"Install Now"**, το κουτάκι **"Add python.exe to PATH"** και μετά να πατήσουμε το **"Install Now"**.

Η επιλογή **"Add python.exe to PATH "** εξασφαλίζει ότι μπορούμε να εκτελέσουμε την Python από οποιοδήποτε φάκελο μέσω **Command Prompt (cmd)** χωρίς να χρειάζεται να προσδιορίσουμε την πλήρη διαδρομή του αρχείου Python κάθε φορά.

Ουσιαστικά, το PATH είναι μια λίστα φακέλων στο σύστημα του υπολογιστή που περιέχει προγράμματα και εκτελέσιμα αρχεία. Όταν πληκτρολογούμε κάτι που σχετίζεται με *Python* στο **Command Prompt (cmd)**, το λειτουργικό σύστημα αναζητά τη λέξη αυτή στους φακέλους που είναι καταχωρημένοι στο PATH και αν τη βρει, τότε εκτελεί το πρόγραμμα.

Κατά τη διάρκεια της εγκατάστασης περιμένουμε λίγο και, στο τέλος, πατάμε το κουμπί **Close**.

Για να ελέγξουμε αν όλα πήγαν καλά κατά την εγκατάστση, ακολουθούμε τα εξής:

1. Ανοίγουμε το **Command Prompt (cmd)** . Ένας εύκολος τρόπος είναι να πληκτρολογήσουμε **cmd** στην **Αναζήτηση** των Windows.
2. Πληκτρολογούμε την εντολή: **python –version** ή απλά **python**

Αν όλα πήγαν καλά, θα δούμε την έκδοση της Python ή θα μπούμε στο περιβάλλον της.

Επίσης, μπορούμε να δοκιμάσουμε ένα μικρό script ως εξής:

1. Ανοίγουμε το Notepad.
2. Γράφουμε την εντολή **print("Hello World!")**
3. Αποθηκεύουμε **File → Save As…**
   με το όνομα **hello.py** (Προσοχή, επιλέγουμε Save as Type: All Files και Encoding:

UTF-8).

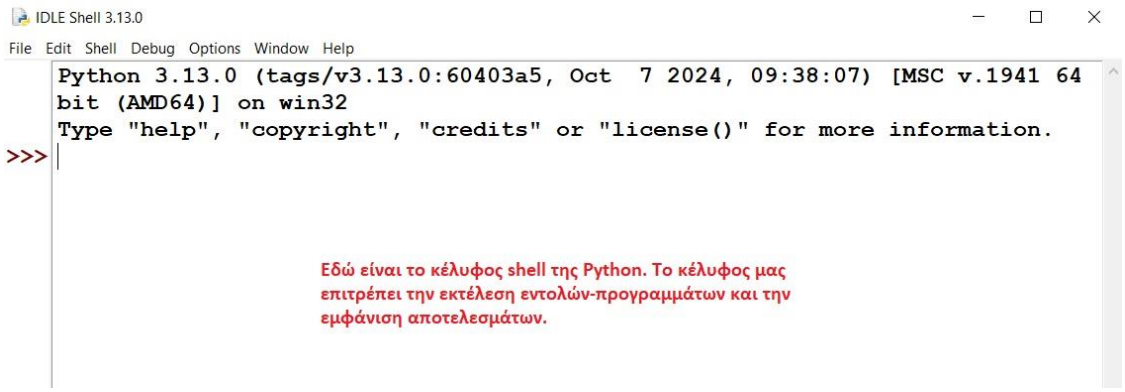
4. Το αποθηκεύουμε στην Επιφάνεια Εργασίας ή σε κάποιο φάκελό μας.
5. Ανοίγουμε το **Command Prompt (cmd)**.
6. Πηγαίνουμε στον φάκελο που το σώσαμε π.χ. στην Επιφάνεια Εργασίας με την εντολή **cd Desktop**.
7. Τρέχουμε το script με την εντολή: **python hello.py**
8. Αν όλα πήγαν καλά, θα δούμε στην οθόνη το αποτέλεσμα που είναι: Hello World!. Αυτό σημαίνει ότι η Python λειτουργεί σωστά.

Λίγα λόγια για το IDLE

Το **IDLE** είναι ένα γραφικό πρόγραμμα - περιβάλλον με editor και interpreter, έρχεται μαζί με την εγκατάσταση της Python και δεν χρειάζεται ξεχωριστή εγκατάσταση.

Για να ανοίξουμε το IDLE στα Windows :

1. Πατάμε Start (Έναρξη) ή το πλήκτρο Windows.
2. Πληκτρολογούμε IDLE.
3. Εμφανίζεται κάτι σαν IDLE (Python 3.x 64-bit).
4. Κάνουμε κλικ και το ανοίγουμε (Εικόνα 35).



Εικόνα 35: Το κέλυφος (*shell*) της *Python*

Πως μπορούμε να τρέξουμε ένα πρόγραμμα μέσα από το IDLE

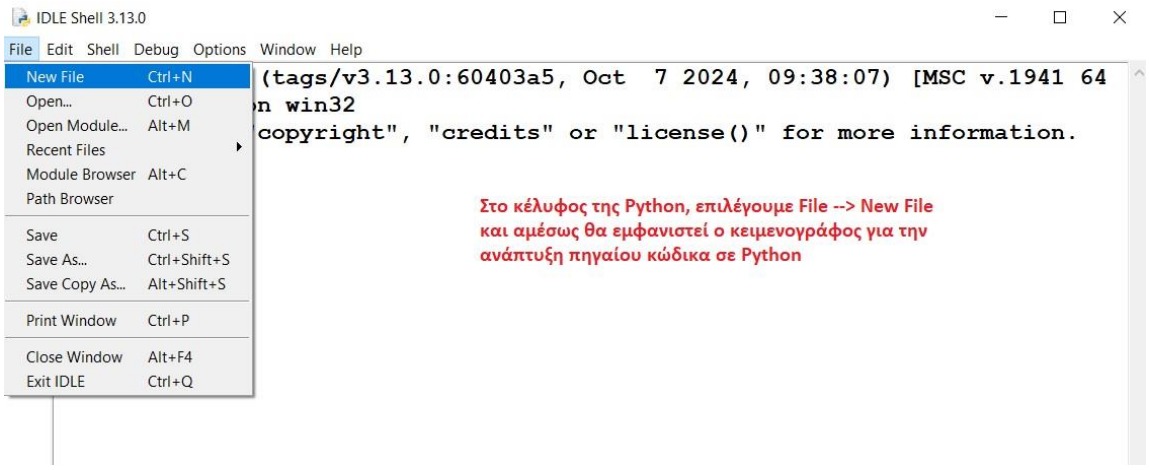
1. Ανοίγουμε το IDLE.
2. Από το μενού πατάμε File → New File (Εικόνα 36)

3. Γράφουμε τον κώδικά μας, για παράδειγμα

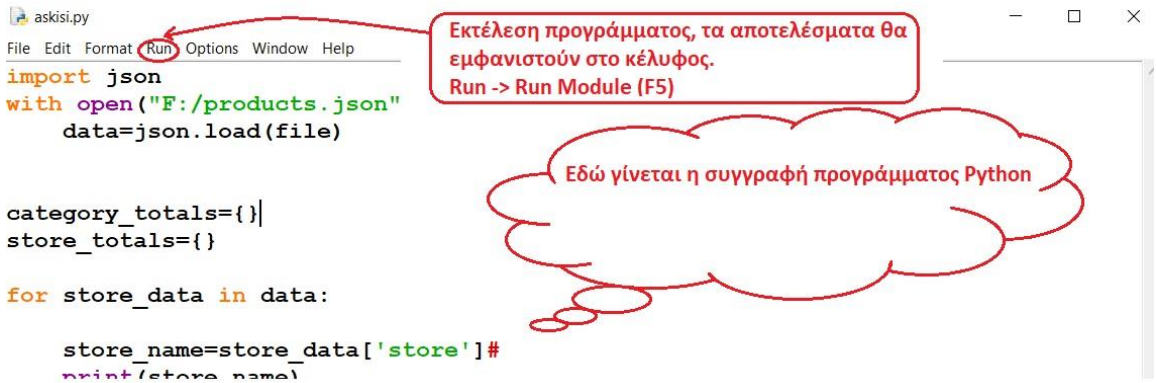
   name = input (“Πώς σε λένε; ”)

   print(f”Χαίρω πολύ, {name}!”)
4. File → Save As… για την αποθήκευση με το όνομα π.χ. my_program.py
5. Τέλος το τρέχουμε με Run → Run Module (ή F5) (Εικόνα 37).

Αν **δεν** έχουμε κάνει Αποθήκευση και προσπαθήσουμε να τρέξουμε με **Run** το πρόγραμμα, μας εμφανίζεται ένα μήνυμα που μας υποχρεώνει να προβούμε σε αποθήκευση, προτού το τρέξουμε.



Εικόνα 36: Κειμενογράφος της *Python*



Εικόνα 37: *Run* στην *Python*

Αν για κάποιο λόγο το IDLE δεν εμφανίζεται στο μενού Έναρξη, μπορούμε να το εκκινήσουμε από το **Command Prompt** ως εξής:

1. Ανοίγουμε το Command Prompt
2. πληκτρολογούμε την εντολή **python -m idlelib.idle**

Έτσι θα ανοίξει το IDLE απευθείας, αν η Python έχει εγκατασταθεί σωστά.

Όλες οι εντολές και εκφράσεις εισάγονται στο κέλυφος της Python (**Python shell**) και εκτελούνται από την Python, όπως φαίνεται από το προτρεπτικό (prompt) >>> πριν από κάθε εντολή/έκφραση.

Όλες οι εντολές που παρουσιάζονται και εκτελούνται μέσω του κελύφους μπορούν, κατά πλειοψηφία, να χρησιμοποιηθούν και εντός οποιουδήποτε προγράμματος Python.

Προαιρετικά, μπορούμε να χρησιμοποιήσουμε, αντί του προτεινόμενου IDLE, διαφορετικά visual περιβάλλοντα όπως το VS Code που μπορεί να το κατεβάσουμε από την ιστοσελίδα: https://code.visualstudio.com/ , αρκεί κατά την εγκατάσταση να μεριμνήσουμε να εγκατασταθούν και τα Python Extensions. Επίσης, υπάρχει και το Thonny από την ιστοσελίδα : https://thonny.org/ το οποίο είναι ένα πολύ καλό και εύχρηστο περιβάλλον, αλλά απευθύνεται περισσότερο σε αρχάριους προγραμματιστές.

Κάποιες χρήσιμες βιβλιοθήκες της Python

math module**:** Είναι μια **ενσωματωμένη** βιβλιοθήκη στην Python που περιέχει μαθηματικές συναρτήσεις και σταθερές όπως π.χ. sqrt (τετραγωνική ρίζα) , log (για το ln), log10 (για το log), sin – cos – tan (για τριγωνομετρικές συναρτήσεις), **π** (για το **π**).

Το συγκεκριμένο module ΔΕΝ απαιτεί επιπλέον εγκατάσταση.

time module : είναι **ενσωματωμένο** module και χρησιμοποιείται για να δουλέψουμε με τον χρόνο π.χ. για μετρητή χρόνου, διαχείριση ημερομηνιών, καθυστέρηση προγράμματος π.χ. με τη time.sleep(2) καθυστερεί το πρόγραμμα για 2 δευτερόλεπτα.

numpy library : Είναι μία εξωτερική βιβλιοθήκη για την Python που εξειδικεύεται στις μεγάλες αριθμητικές δομές και στους προχωρημένους μαθηματικούς υπολογισμούς. Επίσης, είναι πολύ γρήγορη και αποδοτική για υπολογισμούς με μεγάλα δεδομένα, περιλαμβάνει στατιστικά εργαλεία και παρέχει τη δυνατότητα εργασίας με arrays.

Χρειάζεται εγκατάσταση:

1. Ανοίγουμε το Command Prompt (cmd).
2. Πληκτρολογούμε την εντολή pip install numpy

tkinter : Είναι μια **ενσωματωμένη** βιβλιοθήκη της Python για τη δημιουργία γραφικών (**GUI**) με παράθυρα, κουμπιά, ετικέτες, κείμενα, πεδία εισαγωγής κ.α. Είναι σχετικά απλή και παρέχει αρκετές δυνατότητες για να κατασκευάσει κανείς εφαρμογές με γραφικό περιβάλλον, χωρίς να χρειάζεται εξωτερικές βιβλιοθήκες ή περίπλοκες εγκαταστάσεις. Χρησιμοποιείται περισσότερο για μικρές εφαρμογές στις οποίες θέλουμε ευχρηστία και γρήγορη ανάπτυξη, χωρίς πολλές απαιτήσεις γραφικών.

Ένα σημαντικό χαρακτηριστικό του tkinter είναι το **messagebox**, το οποίο είναι μια ειδική βιβλιοθήκη που επιτρέπει την εμφάνιση διαλόγων (pop-ups) με μηνύματα, για να ενημερώνουμε ή να επιβεβαιώνουμε ενέργειες από τον χρήστη. Είναι χρήσιμο για ειδοποιήσεις, σφάλματα ή επιβεβαιώσεις.

Άλλη βιβλιοθήκη του tkinter είναι η **font** που επιτρέπει την αλλαγή της γραμματοσειράς σε όλα τα γραφικά στοιχεία της εφαρμογής, όπως σε ετικέτες, κουμπιά και κείμενα. Μπορούμε, έτσι, να καθορίσουμε παραμέτρους όπως π.χ. μέγεθος, στυλ, όνομα γραμματοσειράς.

Η κλάση **Toplevel** της **Tkinter** χρησιμοποιείται για να δημιουργήσει δευτερεύον παράθυρο (window) πάνω από το κύριο παράθυρο της εφαρμογής. Μπορεί να χρησιμοποιηθεί ως παράθυρο ειδοποίησης, διαλόγου κλπ. Σε συνδυασμό με την **Label** χρησιμοποιείται για να εμφανίσει στατικό κείμενο ή εικόνες σε ένα παράθυρο

Pandas : Είναι δημοφιλής και, ταυτόχρονα, ισχυρή βιβλιοθήκη της Python για ανάλυση δεδομένων. Παρέχει εργαλεία για την εργασία με δεδομένα σε μορφή πίνακα (γραμμές και στήλες), όπως τα Dataframes και Series. Είναι απαραίτητο εργαλείο για επιστήμες που ασχολούνται με την ανάλυση δεδομένων και τη μηχανική μάθηση.

Η εγκατάσταση γίνεται με την εντολή : pip install pandas

Επίσης, το Pandas μπορεί να διαβάσει και να γράψει δεδομένα σε διάφορους τύπους αρχείων όπως CSV , Excel, SQL και JSON.

Έτσι, αν χρειαζόμαστε ευκολία στη διαχείριση δεδομένων (η οργάνωση γίνεται σε Dataframes για ευκολία επεξεργασίας), πολλές συναρτήσεις (για ανάλυση, φιλτράρισμα, συγχώνευση και καθαρισμό δεδομένων), αξιοπιστία, ταχύτητα και ευρεία υποστήριξη για αρχεία τότε το Pandas είναι μονόδρομος.

matplotlib : Είναι μια γνωστή βιβλιοθήκη της Python για οπτικοποίηση δεδομένων. Χρειάζεται, κυρίως, για τη δημιουργία διαγραμμάτων και γραφικών. Μπορούμε, έτσι, να δημιουργήσουμε γραφήματα 2D και 3D. Υποστηρίζει, σχεδόν, κάθε τύπο γραφήματος και συνδυάζεται με άλλες βιβλιοθήκες, όπως το Pandas και το Numpy δίνοντας, με αυτό τον τρόπο, αποτελέσματα σε πιο ισχυρές αναλύσεις.

Ένα πολύ δημοφιλές κομμάτι της βιβλιοθήκης είναι το **pyplot module**, το οποίο προσφέρει μια εύχρηστη διεπαφή για τη δημιουργία γραφημάτων, παρόμοια με εκείνη του λογισμικού MATLAB.

Η εγκατάσταση γίνεται με την εντολή : pip install matplotlib

openpyxl : Το συγκεκριμένο module της Python είναι ουσιαστικά μια βιβλιοθήκη που χρησιμοποιείται για την ανάγνωση, εγγραφή και τροποποίηση αρχείων Excel με κατάληξη .xlsx (Excel 2007 και μετά). Είναι χρήσιμη γιατί χειρίζεται φύλλα εργασίας Excel χωρίς να χρησιμοποιήσει κανείς, κάποιο εμπορικό πρόγραμμα, όπως π.χ. το Microsoft Excel.

Η εγκατάσταση γίνεται με την εντολή : pip install openpyxl

Πολλές φορές το pandas χρειάζεται το openpyxl για να δουλέψει με αρχεία .xlsx (Excel 2010 και μετά). Αν δεν είναι εγκατεστημένο και προσπαθήσουμε να αποθηκεύσουμε ένα Dataframe σε Excel, θα λάβουμε ένα σφάλμα το οποίο θα μας αναφέρει ότι λείπει το συγκεκριμένο module.

Αξίζει να αναφέρουμε ότι το pandas συνεργάζεται στο παρασκήνιο με το openpyxl και δεν χρειάζεται να προσθέσουμε επιπλέον γραμμές κώδικα στο πρόγραμμα που υλοποιούμε.

Εγκατάσταση της Python σε Ubuntu Linux

1. **Αναβάθμιση Πακέτων και Αποθετηρίων**

   Πριν ξεκινήσουμε, είναι καλό να κάνουμε αναβάθμιση των αποθετηρίων και των πακέτων του συστήματός μας. Αυτό διασφαλίζει ότι έχουμε τις πιο πρόσφατες εκδόσεις των πακέτων.

   Ανοίγουμε ένα **τερματικό με ctrl + alt + t** και πληκτρολογούμε:

   sudo apt update

   sudo apt upgrade

   Αυτό θα ανανεώσει τα αποθετήρια και θα εγκαταστήσει τις τελευταίες ενημερώσεις για τα πακέτα του συστήματός μας.

2. **Εγκατάσταση της Python 3.x**

   Η Python 3 είναι η προτιμώμενη έκδοση που χρησιμοποιείται πλέον, καθώς η Python 2 δεν υποστηρίζεται πια (τέλος ζωής το 2020). Αν θέλουμε να εγκαταστήσουμε την τελευταία έκδοση της **Python 3** στην Ubuntu, εκτελούμε την εξής εντολή:

   sudo apt install python3

   Αυτό θα εγκαταστήσει την Python 3 στην τελευταία διαθέσιμη έκδοση από τα αποθετήρια της Ubuntu.

3. **Εγκατάσταση του pip για Python 3**

   Το **pip** είναι το εργαλείο που χρησιμοποιείται για την εγκατάσταση βιβλιοθηκών και πακέτων Python. Για να εγκαταστήσουμε το **pip** για Python3, χρησιμοποιούμε την παρακάτω εντολή:

   sudo apt install python3-pip

Με αυτό το βήμα, θα έχουμε το **pip3**, το οποίο χρησιμοποιείται για την εγκατάσταση βιβλιοθηκών για την Python 3.

4. **Επαλήθευση της Εγκατάστασης**

Για να επιβεβαιώσουμε ότι η Python 3 και το pip έχουν εγκατασταθεί σωστά, μπορούμε να εκτελέσουμε τις εξής εντολές:

- **Έλεγχος για την Python 3**:

  python3 –version

  Αυτό θα εμφανίσει την έκδοση της Python 3 που έχουμε εγκαταστήσει (π.χ., Python 3.x.x).

- **Έλεγχος για το pip**:

  pip3 –version

  Αυτό θα μας δείξει την έκδοση του **pip** για Python 3

5. **Εγκατάσταση Βιβλιοθηκών με pip**

Με το **pip3**, μπορείς να εγκαταστήσεις διάφορες βιβλιοθήκες Python. Για παράδειγμα, αν θέλεις να εγκαταστήσεις το **pandas**, μπορείς να χρησιμοποιήσεις την παρακάτω εντολή:

pip3 install pandas

ή καλύτερα με χρήση του apt **sudo apt install python3-pandas**

Για επιβεβαίωση πληκτρολογούμε:

python3 -c "import pandas; print(pandas.__version__)"

ή καλύτερα **pip3 show pandas**

Για το **IDLE** (το προεπιλεγμένο περιβάλλον ανάπτυξης της Python), συνήθως δεν χρειάζεται να κάνουμε τίποτα επιπλέον, καθώς είναι προεγκατεστημένο με την Python σε πολλές διανομές Linux, όπως η Ubuntu.

- **Έλεγχος, αν το IDLE είναι ήδη εγκατεστημένο**

  Για να δούμε αν έχουμε ήδη το **IDLE** εγκατεστημένο, ανοίγουμε το **τερματικό** και πληκτρολογούμε την εντολή:

  python3 -m idlelib.idle ή **idle**

  Αν το IDLE είναι εγκατεστημένο, θα ανοίξει το περιβάλλον του IDLE. Αν δεν είναι, θα λάβουμε μήνυμα σφάλματος, και θα πρέπει να το εγκαταστήσουμε.

- **Εγκατάσταση του IDLE**

  Εάν το **IDLE** δεν είναι ήδη εγκατεστημένο, μπορούμε να το εγκαταστήσουμε με την παρακάτω εντολή:

  sudo apt install python3-idle

  Αυτό θα εγκαταστήσει το IDLE για την Python 3.

- **Ανοίγοντας το IDLE**

  Αφού εγκατασταθεί, μπορούμε να το ελέγξουμε με την εντολή:

  python3 -m idlelib.idle ή idle

  ή από το μενού εφαρμογών του Ubuntu αν έχουμε γραφικό περιβάλλον.

Εναλλακτικές Προτάσεις (Για Προχωρημένους Χρήστες)

Το **IDLE** είναι ένα πολύ απλό και βασικό περιβάλλον ανάπτυξης για μικρά έργα, αλλά αν θέλουμε κάτι πιο ισχυρό ή επαγγελματικό, μπορούμε να εγκαταστήσουμε άλλα IDEs, όπως:

- **VS Code**: Πολύ δημοφιλές και ισχυρό, με πολλές δυνατότητες και επεκτάσεις.

- **PyCharm**: Ιδανικό για πιο προχωρημένα έργα, με πολλά εργαλεία για ανάπτυξη σε Python.
- **Jupyter Notebooks**: Ιδανικό για ανάλυση δεδομένων και Python scripting.

Παράρτημα Β: Τμήματα κώδικα του απλού μοντέλου *Erlang B*

Η προσέγγιση των παραγοντικών

Συγκεκριμένα, η προσέγγιση των παραγοντικών γίνεται με τη λογική εκθετικών και λογαρίθμων και από τον τύπο (2.14) έχουμε τα εξής:

$$B \equiv P_C = \frac{\frac{\alpha^C}{C!}}{\sum_{n=0}^{C} \frac{a^n}{n!}} = \frac{\alpha^C e^{(-\sum_{i=1}^{C} \ln(i))}}{\sum_{n=0}^{C} \alpha^n e^{(-\sum_{i=1}^{n} \ln(i))}} \quad (2.19)$$

Υπενθυμίζουμε κάποιες ιδιότητες:

- $e^{(lna^k)} = e^{kln(\alpha)}$ (2.20)
- $s! = e^{ln(s!)} = e^{\sum_{i=1}^{s} ln(i)}$ (2.21)
- $e^{slna} = a^s$ (2.22)

Έτσι, έχουμε $\frac{e^{Clna} e^{(-\sum_{i=1}^{C} \ln(i))}}{\sum_{n=0}^{C} e^{nlna} e^{(-\sum_{i=1}^{n} \ln(i))}} = \frac{e^{(Clna - \sum_{i=1}^{C} \ln(i))}}{\sum_{n=0}^{C} e^{(nlna - \sum_{i=1}^{n} \ln(i))}}$ (2.23)

Οπότε υπολογίζουμε το $Clna - \sum_{i=1}^{C} ln(i)$ για τον αριθμητή και $nlna - \sum_{i=1}^{n} ln(i)$ για τον παρονομαστή για n από 0 έως *C*.

Για να μειώσουμε την πολυπλοκότητα, αποθηκεύουμε τα $\sum_{i=1}^{n} ln(i)$, για κάθε *n από* 0…*C* ώστε να μην τα υπολογίσουμε ξανά.

Η ιδέα είναι να φτιάξουμε έναν πίνακα – λίστα *log_fact[] :*

log_fact[n] = $\sum_{i=1}^{n} ln(i)$, *με log_fact[0]=0* (2.24)

Για τον αριθμητή έχουμε :

log_num = C*log(α) – log_fact[C] (2.25)

Για τον παρονομαστή έχουμε :

Για κάθε n από 0 έως C: n*log(α) – log_fact[n] (2.26)

Παρ' όλα αυτά το πρόγραμμα εμφάνισε σχετικό μήνυμα περιορισμού, γιατί η συνάρτηση *exp()* της *Python* δεν μπορεί να ανταποκριθεί σε υπολογισμούς με τεράστιους αριθμούς. Το παραπάνω πρόβλημα αντιμετωπίστηκε ως εξής:

Στον παρονομαστή αντί για exp() για κάθε όρο ακολουθήσαμε τη λογική *log-sum-exp*.

Δηλαδή:

$\ln(\sum_k e^{x_k}) = m + \ln(\sum_k e^{(x_k - m)})$, όπου $m = max_k\ (x_k)$ (2.27)

Στην ουσία προσθέτουμε και αφαιρούμε το μέγιστο m των x_k (στο πρόγραμμά μας το x_k είναι ο πίνακας-λίστα *log_terms*). Με αυτό το τέχνασμα κρατάμε τους εκθέτες κοντά στο 0, αποφεύγοντας τεράστια *exp()*. Αξίζει να αναφέρουμε ότι εδώ δανειστήκαμε ένα *ln*, το οποίο εξουδετερώνεται από ένα *exp()* στον τελικό υπολογισμό.

Ο τελικός υπολογισμός γίνεται :

exp (log_num – log_den) (2.28)

(Σημείωση: Η Python χρησιμοποιεί τη συνάρτηση log για τον φυσικό λογάριθμο ln και log10 για τον λογάριθμο με βάση το 10).

Στον αρχικό τύπο του μοντέλου *Erlang B*, ενώ καταφέραμε να αντιμετωπίσουμε τεράστια *exp()*, εντούτοις η πολυπλοκότητα ήταν της τάξης $O(C^2)$.

Ο παρονομαστής έχει $O(C^2)$: Για κάθε *n από 0 έως C*, υπολογίζει *nln(a),* $\sum_{i=1}^{n} ln(i)$, βρίσκει το *max* και υπολογίζει τα *exp()*.

Άρα για κάθε *n από 0…C* έχουμε :

- έναν (1) πολλαπλασιασμό *nlna* → $O(1)$ και
- ένα (1) άθροισμα $\sum_{i=1}^{n} \ln(i)$ για κάθε *i από 1 έως n* → $O(n)$

Επομένως:

$$\sum_{n=0}^{C} O(n) = O(\sum_{n=0}^{C} n) = O\left(\frac{C(C+1)}{2}\right) = O(C^2) \quad (2.29)$$

Ο αριθμητής έχει $O(C)$ γιατί:

- έναν (1) πολλαπλασιασμό *Cln(α)* → *O(1),*
- ένα (1) άθροισμα λογαρίθμων για C! : $\sum_{i=1}^{C} \ln(i)$ → O(C),
- ένα (1) *max* → *O(C)* και
- ένα (1) sum(exp()) → O(C)

Έτσι το κλάσμα βγάζει πολυπλοκότητα $O(C^2)$**.** Όπως γράψαμε και προηγουμένως, για να μειώσουμε την πολυπλοκότητα σε O(C), αποθηκεύσαμε τα $\sum_{i=1}^{n} ln(i)$ για κάθε *n=0 έως C* ώστε να μην τα υπολογίζουμε ξανά.

Η πολυπλοκότητα γίνεται O(C) γιατί:

- *O(C)* για τους υπολογισμούς που έγιναν από πριν και αποθηκεύτηκαν τα αποτελέσματα για να τα χρησιμοποιήσουμε γρηγορότερα αργότερα. *(precompute)*
- *O(C)* για (επανάληψη) loop στον παρονομαστή

Με τον παραπάνω τρόπο καταφέραμε ακόμα και ο αρχικός τύπος με παραγοντικά της φόρμουλας *Erlang B*, να δίνει αποτελέσματα σε λογικά χρονικά πλαίσια. Εντούτοις, δεν καταφέρνει να φτάσει τους χρόνους του Επαναληπτικού και Αναδρομικού τρόπου.

<u>Ο υπολογισμός του πλήθος των πράξεων (*Ops)*</u>

- Για τον **Αρχικό τύπο (με τα παραγοντικά)** μετά την τροποποίηση έχουμε τα εξής:
  1. *Precompute log_fact* για όλα τα *n*=1…*C*
     1*log* + *1* πρόσθεση ανά *n* άρα περίπου 2*C* πράξεις
  2. *Loop* για *log_terms για n*=0…*C*
     1log(a) + 1 πολλαπλασιασμό +1 αφαίρεση άρα περίπου 3*C* πράξεις
  3. *Max : O(C)* συγκρίσεις άρα περίπου *C* πράξεις
  4. *Loop για exp (για log_terms) : O(C)*
     1 αφαίρεση + 1 *exp* άρα περίπου 2*C* πράξεις

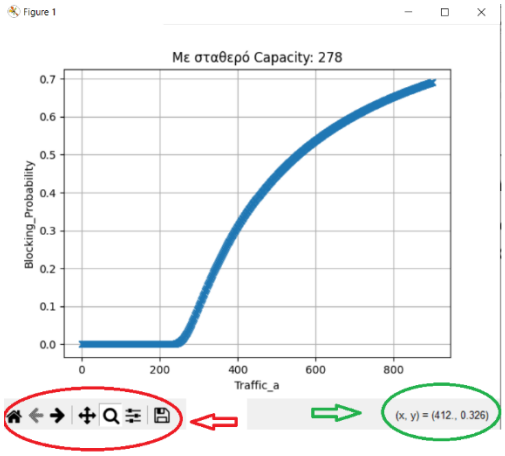
  Τελικά, 2 -3 πράξεις για *log-sum-exp* και τελικό *exp – log*
  Άρα *2C + 3C + 2C + C + 3 = 8C + 3* πράξεις περίπου.

- Για τον **Αναδρομικό** και τον **Επαναληπτικό τύπο,** *C* φορές για το βάθος της αναδρομής ή *C* φορές για την επανάληψη έχουμε:
  1. 2 πράξεις για πολλαπλασιασμό
  2. 1 πράξη για πρόσθεση
  3. 1 πράξη για διαίρεση

  Άρα, *4C* πράξεις

Διαγράμματα

Στα σχήματα των διαγραμμάτων υπάρχουν, κάτω αριστερά, όπως φαίνεται στην Εικόνα 38 με τον κόκκινο κύκλο, εικονίδια για τον έλεγχο του διαγράμματος π.χ. εστίαση, αποθήκευση κ.α. Ενώ, αν θέλουμε να δούμε τις συντεταγμένες ενός σημείου του διαγράμματος πηγαίνουμε πάνω του, με το δείκτη του ποντικιού και κάτω δεξιά μας τις εμφανίζει επακριβώς (πράσινος κύκλος στην εικόνα).



Εικόνα 38: Απλό Διάγραμμα *Erlang B*

Seeds

Για την καλύτερη κατανόηση του *seed* θα δώσουμε το εξής παράδειγμα:

Έστω ότι έχουμε ένα δισκάκι με τραγούδια και σ' αυτό γράφουμε 100 τραγούδια με τυχαίο τρόπο, το *seed(x)* λέει ότι το δισκάκι θα παίζει πάντα από το σημείο *x*. Έτσι, για το συγκεκριμένο *seed* θα έχουμε πάντα την ίδια σειρά εκτέλεσης των τραγουδιών. Η τυχαιότητα έγκειται μόνο στην εγγραφή του δίσκου.

Στο πρόγραμμά μας θέλουμε ξεχωριστό *seed* τόσο για την *expovariate* του λ όσο και του μ. Εδώ η *python,* ενώ έχει ενσωματωμένη την global γεννήτρια *random.seed(),* δεν μας δίνει τη δυνατότητα να έχουμε ξεχωριστά *seeds*, αλλά ένα γενικό *(global) seed*. Αυτό το πρόβλημα λύνεται με τη συνάρτηση *Random της random*.

Η *random.Random* είναι μια κλάση *(class) της Python* που επιτρέπει τη δημιουργία ξεχωριστών ανεξάρτητων γεννητριών τυχαίων αριθμών. Παίρνει ως όρισμα ένα *seed* που καθορίζει την αρχική κατάσταση της γεννήτριας. Έτσι, η *random.Random* υλοποιεί μία δικιά της ανεξάρτητη γεννήτρια τυχαίων αριθμών, και έχει μέσα όλες τις μεθόδους του *random module*, συμπεριλαμβανομένου και του *expovariate().*

Για παράδειγμα στον κώδικα που αναπτύξαμε η random.Random παίρνει ως όρισμα ένα νούμερο *seedl*. Δηλαδή φτιάχνει μία ανεξάρτητη γεννήτρια - αντικείμενο τυχαίων αριθμών που ξεκινά από το *seedl,* μ' αυτό το αντικείμενο καλούμε την *expovariate.* Έτσι, αν δημιουργήσουμε ξανά το ίδιο αντικείμενο με το ίδιο *seed*, θα πάρουμε την ίδια σειρά αριθμών.

seed_L = random.Random(seedl)

arrival_time = seed_L.expovariate(λ)

Αν θέλαμε ένα global seed θα χρησιμοποιούσαμε τις παρακάτω εντολές:

random.seed(x)

arrival_time = random.expovariate(λ)

Η *python* μας διατυπώνει ότι δεν χρειάζεται να μας απασχολεί το εύρος τιμών των *seeds* στη *random* μιας και το φροντίζει αυτή. Άρα μπορούμε να βάλουμε όποια τιμή επιθυμούμε ως όρισμα.

Τμήμα κώδικα που υλοποιεί το μοντέλο *Erlang B* με τους τρεις τρόπους

```
#---------Υπολογισμός του Erlang B -------------------------------------
#-----------------------------------------------------------------------
# Ο αρχικός τύπος του Erlang B που περιλαμβάνει τα παραγοντικά
# τα παραγοντικά αντιμετωπίζονται μέσω log - sum - exp
# (στην python log ειναι το ln)

def erlang_b_arxiki(s,a):
 try:
     # αρχικοποιούμε τον πίνακα-λίστα log_fact με 0.0 --> s+1 θέσεις
     # δεν ξεχνάμε τη μηδενική θέση - κατάσταση άρα συνολικά 0,1,2,...s
     # δηλαδή s+1 θέσεις
     log_fact = [0.0]*(s+1)

     # καταχώρηση των αθροισμάτων στις θέσεις του πίνακα-λίστα
     for i in range(1,s+1):
      log_fact[i]=log_fact[i-1]+log(i)


     # υπολογισμός του log --> του αριθμητή
     log_num = s * log(a) - log_fact[s]

     # υπολογισμός του log --> του παρανομαστή
     # εδώ παίρνουμε έτοιμα τα αθροίσματα
     # που είχαμε υπολογίσει προηγουμένως
     log_terms=[]
     for k in range (0,s+1):
         log_terms.append(k*log(a) - log_fact[k])


     # βρίσκουμε το max από τον log_terms
     megisto= max(log_terms)

     # προσθέτοντας το m και αφαιρώντας ταυτόχρονα,
     # βάζοντας το μέσα στο log-sum-exp
     # διατηρούμε του εκθέτες κοντά στο μηδέν
     # αποφεύγοντας τα τεράστια exp
     # εδώ δανειζόμαστε, για το τέχνασμα υπολογισμού
     # του exp, σκόπιμα ένα log
     log_den=megisto + log(sum(exp(x-megisto) for x in log_terms))

     # στη return εξουδετερώνεται με exp το log
     # που δανειστήκαμε σκόπιμα πριν.
     return exp(log_num - log_den)
 except OverflowError:
    return float('nan')

#-------------------------------------------------------------

# Η αναδρομική συνάρτηση της φόρμουλας Erlang B
def erlang_b(s,a):
 if s == 0:
   return 1.0
 else:
   E_prev = erlang_b(s-1,a)
   return (a * E_prev) / (s + a * E_prev)
#-------------------------------------------------------------

# Η επαναληπτική συνάρτηση της φόρμουλας Erlang B
def erlang_b1(s,a):
 E=1.0
 for k in range(1, s + 1):
   E = (a * E) / (k + a * E)
 return E

#-------------------------------------------------------------
```

Για το κομμάτι του κώδικα που σχετίζεται με τα διαγράμματα

```
#-------------------------------------------------------------------------------------
# Η συνάρτηση calculate1() καλείται για να αποθηκεύσει κάποιους υπολογισμούς, με βάση
# τα δεδομένα του χρήστη, αποθηκεύει όλα τα αποτελέσματα σε Excel αρχεία, μέσω
# Dataframes και έπειτα αντλεί αυτά τα δεδομένα από τα Excel αρχεία για να
# εξάγει τα διαγράμματα.
# Εδώ κάνουμε χρήση μόνο του επαναληπτικού τύπου της φόρμουλας ErlangB
def calculate1():

 try:
        # αρχικοποίηση τιμών
        result=-1
        ops=-1
        c=-1
        l=-1
        m=-1

        # Έλεγχος των δεδομένων που έχει δώσει ο χρήστης
        # δεν επιτρέπονται αρνητικές τιμές στα c,λ,μ
        # δεν επιτρέπεται τιμή πάνω απο 1 ή κάτω από 0
        # στο blocking probability (blocking)

        c = int(entry_c.get() or "0")
        if(s<0):
         result_label.config(text="Το s πρέπει να είναι θετικό")
         return

        l = float(entry_l.get() or "0")
        if (l<0):
         result_label.config(text="Το λ πρέπει να είναι θετικό")
         return

        m = float(entry_m.get() or "0")
        if (m<0):
         result_label.config(text="Το μ πρέπει να είναι θετικό")
         return

        blocking = float(entry_b.get() or "0")
        if (blocking < 0 or blocking >1):
         result_label.config(text="Το Blocking Probability πρέπει να είναι θετικό και <=1")
         return

       # Υπολογισμός του Φορτίου Κίνησης
        a=l/m

        method = method_var1.get()
#-------------------------------------------------------------------------------------------------
    #==== Με σταθερό traffic α ====

       # Υπολογισμός για όλες τις τιμές του s1 (από 0 έως c του χρήστη)
       # της φόρμουλας με τον επαναληπτικό τύπο
        if method == "stable_traffic" :
         for s1 in range(0,n+1):
          result= erlang_b1(s1,a)
          if result>=0 :
             u=a*(1-result) #υπολογισμός του Utilization

             # δημιουργία του Dataframe στον πίνακα-λίστα results1
             results1.append({"Capacity_C":s1,"λ":l,"μ":m,"Traffic_a":a,\
                          "Blocking_Probability":result,"Utilization":u,\
                             "Unchanging_Variable":method})

        # αποθήκευση μέσω Dataframe σε Excel
         try:
           df1=pd.DataFrame(results1)
           df1.to_excel("erlang_results_stable_traffic.xlsx",index=False)
           print("Τα δεδομένα αποθηκεύτηκαν στο 'erlang_results_stable_traffic.xlsx'")
         except Exception as e:
            messagebox.showerror("Σφάλμα", f"Πρόβλημα κατά την αποθήκευση!\n\
Παρακαλώ κλείστε το αρχείο,\ναν είναι ανοιχτό!\n{str(e)}")
            sys.exit()
```

```
        # Διάβασμα του Excel και μετατροπή σε dataframes
        # αυτό γινεται για να πάρουμε τα διαγράμματα
        # Εδώ συνεργάζονται οι βιβλιοθήκες pandas και matplotlib
         df1 =pd.read_excel("erlang_results_stable_traffic.xlsx")

         # interactive mode για αποφυγή μπλοκαρίσματος λόγω των 2 διαγραμμάτων
         # που προσπαθούν να ανοίξουν ταυτόχρονα
         plt.ion()

    # διάγραμμα για το blocking
         plt.figure()
         plt.plot(df1["Capacity_C"],df1["Blocking_Probability"],marker='o')
         plt.xlabel("Capacity_C")
         plt.ylabel("Blocking_Probability")
         plt.title(f"Με σταθερό Φορτίο Κίνησης α: {a}")
         plt.grid(True)

         # Περισσότερα σημεία στον άξονα X
         x_min1 = df1["Capacity_C"].min()
         x_max1 = df1["Capacity_C"].max()
         x_ticks1 = np.linspace(x_min1, x_max1, num=10)  # 10 σημεία στον άξονα X
         plt.xticks(x_ticks1)

         # Περισσότερα σημεία στον άξονα Y
         y_min1 = df1["Blocking_Probability"].min()
         y_max1 = df1["Blocking_Probability"].max()
         y_ticks1 = np.linspace(y_min1, y_max1, num=20)  # 10 σημεία στον άξονα Y
         plt.yticks(y_ticks1)

         plt.show()


    # διάγραμμα για το utilization
         plt.figure()
         plt.plot(df1["Capacity_C"],df1["Utilization"],marker='o')
         plt.xlabel("Capacity_C")
         plt.ylabel("Utilization")
         plt.title(f"Με σταθερό Φορτίο Κίνησης α: {a}")
         plt.grid(True)

         # Περισσότερα σημεία στον άξονα X
         x_min2 = df1["Capacity_C"].min()
         x_max2 = df1["Capacity_C"].max()
         x_ticks2 = np.linspace(x_min2, x_max2, num=10)  # 10 σημεία στον άξονα X
         plt.xticks(x_ticks2)

         # Περισσότερα σημεία στον άξονα Y
         y_min2 = df1["Utilization"].min()
         y_max2 = df1["Utilization"].max()
         y_ticks2 = np.linspace(y_min2, y_max2, num=20)  # 10 σημεία στον άξονα Y
         plt.yticks(y_ticks2)

         plt.show()

         # απενεργοποίηση του interactive mode
         plt.ioff()
#-------------------------------------------------------------------------------------------------
```

```
    #==== Με σταθερό capacity C ====

        elif method =="stable_capacity":
         # Υπολογισμός για όλες τις τιμές του α (από 0 έως α του χρήστη)
         # της φόρμουλας με τον επαναληπτικό τύπο
          for a1 in np.arange(0,a+1,1):
           result= erlang_b1(c,a1)
           if result>=0 :
              u=a1*(1-result) # υπολογισμός του Utilization
              # δημιουργία του Dataframe στον πίνακα-λίστα results2
              results2.append({"Capacity_C":c,"Traffic_a":a1,\
                          "Blocking_Probability":result,"Utilization":u,\
                              "Unchanging_Variable":method})

          # αποθήκευση μέσω Dataframe σε Excel
          try:
            df2=pd.DataFrame(results2)
            df2.to_excel("erlang_results_stable_capacity.xlsx",index=False)
            print("Τα δεδομένα αποθηκεύτηκαν στο 'erlang_results_stable_capacity.xlsx'")
          except Exception as e:
              messagebox.showerror("Σφάλμα", f"Πρόβλημα κατά την αποθήκευση!\n\
Παρακαλώ κλείστε το αρχείο,\ναν είναι ανοιχτό!\n{str(e)}")
              sys.exit()

        # Διάβασμα του Excel και μετατροπή σε dataframes
        # αυτό γινεται για να πάρουμε τα διαγράμματα
        # Εδώ συνεργάζονται οι βιβλιοθήκες pandas και matplotlib
          df2 =pd.read_excel("erlang_results_stable_capacity.xlsx")


         # interactive mode για αποφυγή μπλοκαρίσματος λόγω των 2 διαγραμμάτων
         # που προσπαθούν να ανοίξουν ταυτόχρονα
          plt.ion()

        # διάγραμμα για το blocking
          plt.figure()
          plt.plot(df2["Traffic_a"],df2["Blocking_Probability"],marker='x')
          plt.xlabel("Traffic_a")
          plt.ylabel("Blocking_Probability")
          plt.title(f"Με σταθερό Capacity: {c}")
          plt.grid(True)

          # Περισσότερα σημεία στον άξονα X
          x_min1 = df2["Traffic_a"].min()
          x_max1 = df2["Traffic_a"].max()
          x_ticks1 = np.linspace(x_min1, x_max1, num=10)  # 10 σημεία στον άξονα X
          plt.xticks(x_ticks1)

          # Περισσότερα σημεία στον άξονα Y
          y_min1 = df2["Blocking_Probability"].min()
          y_max1 = df2["Blocking_Probability"].max()
          y_ticks1 = np.linspace(y_min1, y_max1, num=20)  # 10 σημεία στον άξονα Y
          plt.yticks(y_ticks1)

          plt.show()

        # διάγραμμα για το utilization
          plt.figure()
          plt.plot(df2["Traffic_a"],df2["Utilization"],marker='x')
          plt.xlabel("Traffic_a")
          plt.ylabel("Utilization")
          plt.title(f"Με σταθερό Capacity: {c}")
          plt.grid(True)

          # Περισσότερα σημεία στον άξονα X
          x_min2 = df2["Traffic_a"].min()
          x_max2 = df2["Traffic_a"].max()
          x_ticks2 = np.linspace(x_min2, x_max2, num=10)  # 10 σημεία στον άξονα X
          plt.xticks(x_ticks2)
```

```
            # Περισσότερα σημεία στον άξονα Y
            y_min2 = df2["Utilization"].min()
            y_max2 = df2["Utilization"].max()
            y_ticks2 = np.linspace(y_min2, y_max2, num=20)  # 10 σημεία στον άξονα Y
            plt.yticks(y_ticks2)

            plt.show()
          # απενεργοποίηση του interactive mode
            plt.ioff()
#-------------------------------------------------------------------------------------------------
    #=== Σταθερή τιμή του blocking που θέλει να υπάρχει ο χρήστης (με απόκλιση 0.001) ===

    # δηλαδή βρίσκουμε ποια ζεύγη c , a από 0 μέχρι τις τιμές
    # που δίνει ο χρήστης (για τα c , a)
    # ικανοποιούν την ισότητα το υπολογιζόμενο blocking με αυτό
    # που έχει δοθεί από τον χρήστη με απόκλιση 0.001
        elif method == "max_blocking":

            for s1 in range(0,c+1):
             for a1 in np.arange(0,a+1,1):
                 result=erlang_b1(s1,a1)
                 if abs(result - blocking)<=0.001 :
                   u=a1*(1-result) #υπολογισμός του Utilization
                   # δημιουργία του Dataframe στον πίνακα-λίστα results2
                   results3.append({"Capacity_C":s1,"Traffic_a":a1,\
                             "Blocking_Probability":result,"Utilization":u,\
                                    "Unchanging_Variable":method})

            # Dataframe για το Excel
            # επειδή εδώ υπάρχει περίπτωση να μην ικανοποιηθεί η ανισότητα
            # abs(result - blocking)<=0.001 γίνεται
            # έλεγχος αν το df3 dataframe είναι κενό
            # αν είναι, τότε το δημιουργούμε με μηδενικές τιμές για να
            # μην κρασάρει το pandas
             df3=pd.DataFrame(results3)
             if (df3.empty):
               results3.append({"Capacity_C":0,"Traffic_a":0,\
                             "Blocking_Probability":0,"Utilization":0,\
                               "Unchanging_Variable":0})

             try:
               df3=pd.DataFrame(results3)
               df3.to_excel("erlang_results_stable_Blocking.xlsx",index=False)
               print("Τα δεδομένα αποθηκεύτηκαν στο 'erlang_results_stable_Blocking.xlsx'")
             except Exception as e:
               messagebox.showerror("Σφάλμα", f"Πρόβλημα κατά την αποθήκευση!\n\
Παρακαλώ κλείστε το αρχείο,\ναν είναι ανοιχτό!\n{str(e)}")
               sys.exit()

            # Από το Excel σε dataframe
            # χρειάζεται για την παραγωγή διαγραμμάτων
             df3 =pd.read_excel("erlang_results_stable_Blocking.xlsx")


           # διάγραμμα
             plt.figure()
             plt.plot(df3["Traffic_a"],df3["Capacity_C"],marker='+')
             plt.xlabel("Traffic_a")
             plt.ylabel("Capacity_C")
             plt.title(f"Με απόκλιση 0.001 από τη δοθείσα Πιθανότητα Απόρριψης: {blocking}")
             plt.grid(True)
```

```
        # Περισσότερα σημεία στον άξονα X
        x_min1 = df3["Traffic_a"].min()
        x_max1 = df3["Traffic_a"].max()
        x_ticks1 = np.linspace(x_min1, x_max1, num=10)  # 10 σημεία στον άξονα X
        plt.xticks(x_ticks1)

        # Περισσότερα σημεία στον άξονα Y
        y_min1 = df3["Capacity_s"].min()
        y_max1 = df3["Capacity_s"].max()
        y_ticks1 = np.linspace(y_min1, y_max1, num=20)  # 10 σημεία στον άξονα Y
        plt.yticks(y_ticks1)

        plt.show()


      else :
        result_label.config(text="Error_Method")

except ValueError:
      messagebox.showerror("Σφάλμα", "Παρακαλώ δώστε έγκυρους αριθμούς!")
```

Τμήμα του κώδικα της προσομοίωσης για τη μέθοδο *Erlang B*

```
def simulation_of_erlang_b(total_calls, c, l, m, seedl, seedm, Transit):

    #=== Μέτρηση χρόνου ===
    start = time.perf_counter()

    q = [0]*(c+1) # θα μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[i] κατάσταση
                  # Προσοχή, εχουμε και την 0 κατάσταση άρα από 0 , 1, 2,...c
                  # συνολικά c+1
    i=0

    # Ορίζουμε ξεχωριστούς Random generators για λ και μ
    seed_L = random.Random(seedl)
    seed_M = random.Random(seedm)

    servers = c

    blocked_calls = 0

    # λίστα αποθήκευσης των χρόνων των κλήσεων
    times = []

    # δημιουργία σωρού αξιοποιώντας τη λίστα times
    # ο σωρός θα τοποθετεί τον μικρότερο χρόνο - τιμή
    # στη θέση 0 της λίστας times
    heapq.heapify(times)

    current_time = 0
    count=0
    tr = Transit      # Transit είναι η μεταβλητή που θα μεταφερθεί από το χρήστη και δηλώνει
                      # τον αριθμό των αρχικών  κλήσεων που δεν θα ληφθούν υπόψη.
                      # Αυτόν τον αριθμό τον μεταφέρουμε σε μία μεταβλητή tr που δουλεύει
                      # σαν 'αντίθετος χρονοδιακόπτης'
                      # δηλαδή μέχρι να μηδενιστεί εκτελείται το πρόγραμμα χωρίς να αποθηκεύεται
                      # κάτι, μετά το μηδενισμό του αρχίζει η μέτρηση και η εξαγωγή
                      # των αποτελεσμάτων.

    for _ in range(total_calls): # η κάτω παύλα στη for δηλώνει οτι
                                 # απλά χρησιμοποιούμε την επανάληψη
                                 # (loop) για total_calls φορές


      #================= Άφιξη νέας κλήσης (Poisson)=====================================

       # Παίρνουμε τον χρόνο άφιξης της κλήσης, από την εκθετική κατανομή
       # του μέσου ρυθμού αφίξης των κλήσεων λ με seed που έδωσε ο χρήστης
          arrival_time = seed_L.expovariate(l)


       # προσθέτουμε το arrival_time στο προηγούμενο current_time
       # για να βρούμε την ακριβή τιμή του χρόνου
       # θυμίζουμε ότι ο χρόνος ξεκινά με 0 (current_time = 0)
          current_time += arrival_time

        #-------------------------------------------------------------------------------
        # ελέγχει αν ειναι άδεια η λίστα times και μετά αν το στοιχείο στη θέση 0
        # ειναι μικρότερο ή ίσο του current_time πράγμα που σημαίνει ότι έχει τελειώσει.
        # Έτσι, κάνει εξαγωγή του στοιχείου που βρίσκεται στη μηδενική θέση της λίστας
```

```
        # Αυτό επαναλαμβάνεται για όλον το σωρό, εξάγοντας όλες τις κλήσεις που έχουν
        # λήξει χρονικά, έτσι αποδεσμεύονται servers, γι' αυτό αυξάνεται η τιμή των
        # διαθέσιμων servers κατά 1 σε κάθε επανάληψη
          while times and times[0] <= current_time:

            heapq.heappop(times)

            # επειδή έφυγε το στοιχείο 0 της λίστας
            # αυξάνουμε τους διαθέσιμους servers κατά 1
            servers += 1
        #-------------------------------------------------------------------------------

        # Έλεγχος διαθεσιμότητας servers
          if servers > 0:
            if tr <= 0:# Transit: όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
                       # τότε αρχίζουμε να παιρνουμε αποτελέσματα

              # Πόσους servers συναντά η κλήση και είναι κατειλημμένοι
              # αυτό το πλήθος το βάζουμε σε ένα άθροισμα
              count+=(c-servers)

              i=c-servers
              q[i]+=1

            #παραχωρούμε έναν server για διάρκεια duration
            servers -= 1

            # χρονική διάρκεια διεκπεραίωσης της κλήσης (duration) με βάση την
            # εκθετική κατανομή του μέσου ρυθμού εξυπηρέτησης μ
            duration = seed_M.expovariate(m)


            # τοποθετούμε στη λίστα times το current_time μαζί με τη διάρκεια duration
            # η λίστα είναι σωρός που στη θέση 0 τοποθετεί τη μικρότερη τιμή
            heapq.heappush(times, current_time + duration)

          else:
            if tr <=0 :# Transit : όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
                       # τότε αρχίζουμε να παιρνουμε αποτελέσματα

             # εδώ μπλοκάρεται η κλήση γιατί δεν υπάρχουν διαθέσιμοι servers
             blocked_calls += 1
             # Εδώ έχουμε όλους τους servers κατειλημμένους άρα το count αυξάνεται κατά c
             count+= c
             q[c]+=1

          tr-=1 # μείωση σε κάθε επανάληψη του "χρονοδιακόπτη" κατά 1

    #=== Τέλος χρόνου ===
    end = time.perf_counter()


    xronos =(end - start)    # χρόνος διεκπεραίωσης του αλγόριθμου


    U=count/(total_calls - Transit) # το συνόλικό πλήθος των calls
                                    # βγαίνει αν αφαιρέσουμε το transit

    return blocked_calls / (total_calls-Transit) , U ,q , xronos

#-----------------------------------------------------------------------------------------------
```

Παράρτημα Γ: Τμήματα κώδικα *Erlang B* με δυνατότητα προσωρινής απώλειας των εξυπηρετητών.

Κομμάτι του κώδικα που υλοποιεί τις τρεις προσεγγίσεις για το σύστημα *Erlang B* με δυνατότητα προσωρινής απώλειας των εξυπηρετητών.

```
############################ 1η μέθοδος Pure Performability Method ######################################

        #=== Μέτρηση χρόνου ===
        start = time.perf_counter()

        counter=0
        for i in range(1,c+1):
         counter+= erlang_b_performance(i,a_performance) * erlang_b_availability(c,i,a_availability)
        result = erlang_b_availability(c,0,a_availability) + counter

        #=== Τέλος χρόνου ===
        end = time.perf_counter()

############################ 2η μέθοδος Bobbio and Trivedi Method ######################################

        #=== Μέτρηση χρόνου ===
        start2 = time.perf_counter()

        counter=0
        for i in range(1,c+1):
         counter+= erlang_b_performance(i,a_performance) * erlang_b_BT(c,i,a_availability,a_performance)
        result2 = erlang_b_BT(c,0,a_availability,a_performance) + counter

        #=== Τέλος χρόνου ===
        end2 = time.perf_counter()
######################### 3η μέθοδος Proposed based on Bobbio and Trivedi Method ##########################

        #=== Μέτρηση χρόνου ===
        start3 = time.perf_counter()

        counter=0
        for i in range(1,c+1):
         counter+= erlang_b_performance(i,a_proposed) * erlang_b_BT(c,i,a_availability,a_proposed)
        result3 = erlang_b_BT(c,0,a_availability,a_proposed) + counter

        #=== Τέλος χρόνου ===
        end3 = time.perf_counter()
```

Κομμάτι του κώδικα της προσομοίωσης για το μοντέλο απωλειών κλήσεων *Erlang B* με κατάρρευση εξυπηρετητή.

```
def simulation_of_erlang_b_Prop_on_BT(total_calls, s, l, m, l_f, m_r,\
                          seedl, seedm, seedl_f, seedm_r, Transit):


    #=== Μέτρηση χρόνου ===
    start = time.perf_counter()

    q = [[0]*(s+1) for _ in range(s+1)] # θα μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[i][j] κατάσταση
                                       # Προσοχή, εχουμε και την 0 κατάσταση άρα από 0 , 1, 2,...s συνολικά s+1
                                       # τόσο στη διάσταση i (για busy servers) όσο και στη j για crashed servers

    p = [[0]*(s+1) for _ in range(s+1)] # ομοίως για τις αντίστοιχες πιθανότητες



    # Ορίζουμε ξεχωριστούς Random generators για λ, μ, λ_f, μ_r
    seed_L = random.Random(seedl)
    seed_M = random.Random(seedm)

    seed_L_f = random.Random(seedl_f+1) #επειδή την expovariate της λf την ενεργοποιούμε ταυτόχρονα
    seed_M_r = random.Random(seedm_r)   #με την expovariate της μ αυξάνουμε το seed της λf  κατά ένα
                                        #για να μην προκύπτουν ίδιοι χρόνοι

    # Μετρικές Servers
    servers = s # διαθέσιμοι servers αρχικοποιούνται με το πλήθος των servers (capacity)
    failed_servers = 0
    busy_servers=0

    # Μετρικές κλήσεων
    blocked_calls = 0
    lost_calls = 0
    success_calls=0


    # λίστα αποθήκευσης των χρόνων των κλήσεων
    times = []

    # δημιουργία σωρού αξιοποιώντας τη λίστα times
    # ο σωρός θα τοποθετεί τον μικρότερο χρόνο - τιμή
    # στη θέση 0 της λίστας times
    heapq.heapify(times)

    count=0
    count2=0

    # Μετρά τον παρόντα χρόνο
    current_time = 0

    tr = Transit     # Transit είναι η μεταβλητή που θα μεταφερθεί από το χρήστη και δηλώνει
                     # τον αριθμό των αρχικών  κλήσεων που δεν θα ληφθούν υπόψη.
                     # Αυτόν τον αριθμό τον μεταφέρουμε σε μία μεταβλητή tr που δουλεύει
                     # σαν 'αντίθετος χρονοδιακόπτης'
                     # δηλαδή μέχρι να μηδενιστεί εκτελείται το πρόγραμμα χωρίς να αποθηκεύεται
                     # κάτι, μετά το μηδενισμό του αρχίζει η μέτρηση και η εξαγωγή
                     # των αποτελεσμάτων.

    # Σημαφόρος που ελέγχει κάθε φορά την επιδιόρθωση ενός μόνο server (simple repair facility)
    # Αρχικοποιείται να είναι κατεβασμένος --> False
    repair_busy = False

    # Αρχικοποίηση ουράς, εδώ θα εισέρχονται οι servers που πρέπει να επιδιορθωθούν
    repair_queue = deque()

    # Έλεγχος για σωστές τιμές του χρήστη
    #-------------------------------------------------------
    # Αν λf είναι 0 τότε θα πρέπει και το μr να είναι μηδέν
    if l_f==0 and m_r!=0:
      messagebox.showerror("Σφάλμα", "λf=0 και μr!=0")
      sys.exit()
    # Αν λf δεν είναι 0 τότε το μr θα πρέπει να έχει τιμή
    if l_f!=0 and m_r==0:
      messagebox.showerror("Σφάλμα", "λf!=0 και μr=0")
      sys.exit()


    arrivals=0 # μετρά το πλήθος των εισερχόμενων κλήσεων

   #================= Άφιξη πρώτης κλήσης (Poisson)=======================================
    # Παίρνουμε τον χρόνο άφιξης της πρώτης κλήσης, από την εκθετική κατανομή του μέσου
    # ρυθμού αφίξης των κλήσεων λ με seed που έδωσε ο χρήστης και την τοποθετούμε στο σωρό.
    # Η εγγραφή στον σωρό έχει δύο χαρακτηριστικά:
    # 1.Το χρόνο άφιξης της κλήσης
    # 2.Ο τύπος-είδος της εγγραφής, εδώ είναι τύπου άφιξης - "arrival" κλήσης
    arrival_time = seed_L.expovariate(l)
    heapq.heappush(times, (arrival_time, "arrival"))
```

```
# Μετά την τοποθέτηση της αρχικής κλήσης στο σωρό του συστήματος ξεκινά
# μια επανάληψη μέχρι να ελεγχθούν όλες οι υπόλοιπες κλήσεις
while arrivals < total_calls:

        # Κάθε φορά θα παίρνουμε το πρώτο στοιχείο του σωρού times (--> min Heap),
        # δηλαδή το στοιχείο με τη μικρότερη τιμή στη θέση times[0][0].
        # Στην ουσία μ' αυτόν τον τρόπο επεξεργαζόμαστε εγγραφές, πρωτού τις διαγράψουμε (pop)
        # από τον σωρό, που έχουν λήξει χρονικά.
        current_time = times[0][0]

        # Στη μεταβλητή event καταχωρούμε τον τύπο-είδος της εγγραφής που έχει λήξει χρονικά.
        # Οι τύποι των εγγραφών είναι:
        # arrival --> για τις κλήσεις που μπαίνουν στο σύτημα
        # call --> οι κλήσεις που διεκπεραιώνονται χωρίς βλάβη του server
        # fail_server --> Βλάβη  server
        # repair --> επιδιόρθωση server
        event = times[0][1]


        ##########################################################################################
        # Ο τύπος της εγγραφής του σωρού times που έληξε και την επεξεργαζόμαστε είναι arrival ##

        # arrival σημαίνει ότι ήρθε η ώρα να επεξεργαστούμε τη συγκεκριμένη κλήση
        if event == "arrival":
          arrivals += 1 # μετρική κλήσεων που δείχνει επιπλέον και ποια κλήση είναι

          # εδώ αυξάνεται και το στοιχείο του πίνακα-λίστα q με βάση το τι ακριβώς βρίσκει
          # η κλήση που επεξεργαζόμαστε σε servers δηλαδή πόσοι είναι busy πόσοι είναι fail
          # οι γραμμές του πίνακα είναι οι busy και οι στήλες είναι οι busy
          q[busy_servers][failed_servers] += 1

          # Αν η κλήση που επεξεργαζόμαστε βρίσκει το σύστημα με το αθροισμα των απασχολημένων
          # και χαλασμένων servers να είναι μικρότερο του πλήθους των servers, σημαίνει ότι
          # υπάρχουν διαθέσιμοι servers και γι' αυτό παραχωρείται ένας εξ'αυτών στην κλήση μας.
          if busy_servers + failed_servers < s:
              if tr <= 0: # Transit: όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
                      # τότε αρχίζουμε να παιρνουμε αποτελέσματα

                  # Αθροισμα για τον υπολογισμό του Utilization
                  # Πόσους servers συναντά η κλήση και είναι κατειλημένοι
                  # αυτό το πλήθος το βάζουμε σε ένα άθροισμα
                  count+= busy_servers


              # Αφού παραχωρούμε server στην κλήση μας τότε
              # η μετρική των διαθέσιμών servers μειώνεται
              # και η μετρική των απασχολημένων servers αυξάνεται.
              servers -=1
              busy_servers +=1

              # Εδώ υπολογίζεται η χρονική διάρκεια διεκπεραίωσης της κλήσης (duration)
              # με βάση την εκθετική κατανομή του μέσου ρυθμού εξυπηρέτησης μ
              duration = seed_M.expovariate(m)

              # Εδώ υπολογίζεται ο χρόνος αποτυχίας του server που έχει δοθεί στην κλήση
              # άρα θα πρέπει ο ρυθμός lf πάνω στον οποίο υπολογίζεται εκθετικά ο χρόνος αυτός
              # να είναι θετικός αριθμός (δηλαδή να υπάρχει) και, επίσης, η μετρική των
              # απασχολημένων servers busy_servers να δείχνει ότι υπάρχει απασχολημένος
              # server (όχι idle)
              if l_f > 0 : # and busy_servers > 0 :
                  crash_time = seed_L_f.expovariate(l_f)

                  # αφού υπολογιστεί αυτός ο χρόνος θα πρέπει να ελεγχθεί αν βρίσκεται εντός
                  # του χρόνου εξυπηρέτησης της κλήσης,
                  # current_time + crach_time <= current_time + duration  ή καλύτερα
                  # crach_time <= duration
                  # αν συμβαίνει κάτι τέτοιο τότε δημιουργείται εγγραφή στο σωρό με χρόνο
                  # τον παρόντα χρόνο συν το χρόνο κατάρρευσης και τύπο εγγραφής server_crash.
                  # Η κλήση χάνεται και έτσι αυξάνεται η μετρική των χαμένων κλήσεων --> lost_calls
                  if crash_time <= duration:
                    heapq.heappush(times, (current_time + crash_time, "server_crash"))
                    lost_calls+=1

                  # διαφορετικά αν το crash έγινε εκτός χρονικού πλαισίου της εξυπηρέτησης κλήσης
                  # η κλήση συνεχίζει κανονικά , καταχωρώντας την στο σωρό με χρόνο
                  # τον παρόντα χρόνο συν τον χρόνο διεκπεραίωσης (με βάση την εκθετική του ρυθμού μ)
                  else:
                    heapq.heappush(times, (current_time + duration, "call"))


              # αν δεν υπάρχει lf δηλαδή έχουμε κλασικό απλό Erlang B
              else:
                heapq.heappush(times, (current_time + duration, "call"))

        #-----------------------------------------------------------------
         else:
            if tr <=0 :# Transit : όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
                  # τότε αρχίζουμε να παιρνουμε αποτελέσματα

                  # Αθροισμα για τον υπολογισμό του Utilization
                  # Πόσους servers συναντά η κλήση και είναι κατειλημένοι
                  # αυτό το πλήθος το βάζουμε σε ένα άθροισμα
                  count+= busy_servers
```

```
                # εδώ μπλοκάρεται η κλήση γιατί δεν υπάρχουν διαθέσιμοι servers
                blocked_calls += 1


            #---------------------- Επόμενη άφιξη ---------------------------------#
            heapq.heappush(times, (current_time + seed_L.expovariate(l), "arrival"))

        ##########################################################################################
        ### Ο τύπος της εγγραφής του σωρού times που έληξε και την επεξεργαζόμαστε είναι call ###

        # call σημαίνει ότι η κλήση διεκπεραιώθηκε οπότε αυξάνεται
        # η μετρική των ελεύθερων servers --> servers
        # και μειώνεται η μετρική των απασχολημένων servers --> busy_servers
        elif event == "call":
            servers+=1
            busy_servers-=1

            success_calls+=1 # μια μετρική για τις επιτυχημένες κλήσεις

        ##########################################################################################
        # Ο τύπος της εγγραφής του σωρού times που έληξε και την επεξεργαζόμαστε είναι server_crash #

        # server_crash σημαίνει ότι έχουμε server που κατέρρευσε
        # Επειδή αναφερόμαστε μόνο σε server που είναι απασχολημένος (όχι idle)
        # μειώνεται η μετρική των απασχολημένων servers --> busy_servers
        # αυξάνεται η μετρική των fail servers --> failed_server
        elif event == "server_crash" :
            busy_servers -=1
            failed_servers +=1

            # υπολογίζεται η διάρκεια επιδιόρθωσης μέσω της εκθετικής του ρυθμού μr
            duration_fix = seed_M_r.expovariate(m_r)

            # Αν ο σημαφόρος είναι κάτω (False) σημαίνει ότι δεν υπάρχει servers που έχει μπει
            # σε mode επιδιόρθωσης, οπότε ξεκινά ο υπο κατάρρευση server επιδιόρθωση
            # δηλαδή μπαίνει στο σωρό με χρόνο τον τωρινό-παρόντα χρόνο συν το χρόνο επιδιόρθωσης
            if repair_busy == False:
                repair_busy = True  # ο σημαφόρος γίνεται True (σηκώνεται)
                heapq.heappush(times, (current_time + duration_fix, "repair_server"))

            # εδώ ο σημαφόρος είναι ανεβασμένος πράγμα που σημαίνει ότι δεν μπορεί να δεχτεί
            # άλλον server για επιδιόρθωση λόγω του single repair facility
            # και τοποθετεί τον server που έχει καταρρεύσει σε μια ουρά αναμονής (FIFO)
            # το χαρακτηριστικό που αποθηκεύουμε στην ουρά αναμονής είναι ο χρόνος επιδιόρθωσης
            # με βάση το μr
            else:
                repair_queue.append(duration_fix)

        ##########################################################################################
        # Ο τύπος της εγγραφής του σωρού times που έληξε και την επεξεργαζόμαστε είναι repair_server #

        # repair_server σημαίνει ότι έχουμε ολοκληρώσει τη διαδικασία επισκευής ενός server που
        # είχε καταρρεύσει.
        # Αμέσως αυξάνεται η μετρική των διαθέσιμων servers  --> servers
        # και μειώνεται η μετρική για τους fail servers --> failed_servers
        elif event == "repair_server" :
            failed_servers -=1
            servers+=1

            # Αν υπάρχουν στην ουρά (repair_queue) των crashed servers,
            # servers που περιμένουν επιδιόρθωση τότε ένας από αυτούς
            # (ο παλαιότερος χρονικά - FIFO) μπαίνει στον σωρό times με
            # εγγραφή που έχει χρόνο τον χρόνο επιδιόρθωσης και τύπο repair_server.


            # Ο χρόνος εξάγεται από τον παρόντα χρόνο και το χρόνο επιδιόρθωσης στηριζόμενοι
            # στον εκθετικό χρόνο του ρυθμο επισκευής μr (που είχε αποθηκευτεί στην ουρά)
            if repair_queue:
                next_fix  = repair_queue.popleft()
                heapq.heappush(times, (current_time + next_fix, "repair_server"))

            else:
                repair_busy = False # Αν δεν υπάρχουν στην ουρά servers για επιδιόρθωση
                                    # τότε ο σημαφόρος πέφτει.


        #-------------------------------------------------------------------------------------------
        # Εξαγωγή-Διαγραφή της εγγραφής του σωρού που έληξε χρονικά
        heapq.heappop(times)


        tr-=1 # μείωση σε κάθε επανάληψη του "χρονοδιακόπτη" κατά 1

#=== Τέλος χρόνου ===
end = time.perf_counter()
xronos =(end - start)   # χρόνος διεκπεραίωσης του αλγόριθμου
```

Παράρτημα Δ: Τμήματα κώδικα του μοντέλου *EMLM*

EMLM : Τμήμα του κώδικα του αναλυτικού μοντέλου

```
# Η συνάρτηση της προσεγγιστικής φόρμουλας Roberts
def calculate_EMLM_BR(c,k,a,b,t_reservation):
     global C
     global K
     global traffic
     global bandwidth
     global tk

     C = c
     K = k
     traffic = a
     bandwidth = b
     tk = t_reservation

     #-------------------------------------------------------------------------------------------------
     # Υπολογισμός των μη κανονικοποιημένων πιθανοτήτων

     #=== Μέτρηση χρόνου ===
     start = time.perf_counter()

     q = [0.0]*(c+1)  # θα μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[i] κατάσταση
                      # Προσοχή, εχουμε και την 0 κατάσταση άρα από 0 , 1, 2,...c
                      # συνολικά c+1
     q[0]= 1.0

     total=q[0]

     for j in range (1,c+1):
        for i in range (1,k+1):
             if (j-b[i-1] < 0.0):
               q[j]+=0.0

             elif (j>C-tk[i-1]):
               q[j]+=0.0

             else:
               q[j] += (a[i-1]*b[i-1] * q[j-b[i-1]]) / j

        total+=q[j]


    # Υπολογισμός των κανονικοποιημένων πιθανοτήτων
    # και του Utilization από τον τύπο της μέσης
    # αναμενόμενης τιμής

     qk = [0.0]*(c+1)
     qk[0]= 1.0 / total

     u=0
     for j in range(1,c+1):
        qk[j] = q[j]/total
        u+=j*qk[j]

    # Υπολογισμός του Blocking b_p για κάθε κατηγορία
     deiktis = [0]*(k)
     b_p = [0.0]*(k)
     metritis=0
     for i in range(1,k+1):
          s=0.0
          if (c-bandwidth[i-1]-tk[i-1]+1 >= 0.0):
            for j in range (c-bandwidth[i-1]-tk[i-1]+1,c+1):
              s += qk[j]
```

η «Εύρεση Επιθυμητού *Capacity*» στον ίδιο κώδικα

```
#-------------------------------------------------------------------------------------------------------

# Η συνάρτηση αυτή με τα δεδομένα στοιχεία που έχει δώσει ο χρήστης
# υπολογίζει τα blocking probability των κατηγοριών και τα συγκρίνει
# με ένα όριο που έχει δώσει ο χρήστης (E). Αν βρίσκεται έστω και ένα
# blocking πάνω από το όριο (E), επαναυπολογίζει τη φόρμουλα αλλά αυξημένο
# το Capacity κατά μία μονάδα. Αυτή η επανάληψη ολοκληρώνεται όταν
# όλα τα Blocking είναι κάτω του ορίου E.
# Σκοπός είναι να κρατήσουμε το Capacity εκείνης της στιγμής

def looping_capacity(c,k,a,b,e,t_reservation):

  # δεν θέλουμε να επηρεαστεί το global Capacity C
  capaCity = c

  # υπολογίζουμε το Blocking κάθε κατηγορίας
  # για να εξετάσουμε αν ο χρήστης έχει δώσει στο προγραμμα
  # τα δεδομένα προς έλεγχο (π.χ. K,C,a κτλ)
  B_p,U,U1,Time,Ops,G,p,pk = calculate_EMLM_BR(capaCity,k,a,b,t_reservation)

  if not B_p:
    messagebox.showerror ("Σφάλμα","Υπολογίστε πρώτα το Blocking με το κουμπί\n\
Έλεγχος και Υπολογισμός")
    return

  else:
   # αφού περάσαμε τους ελέγχους αρχικοποιούμε το capacity σε 1 και τρέχουμε τη
   # συνάρτηση για να παράξουμε το B_p κάθε κατηγορίας

   capaCity=1   # ξεκινάμε με capacity 1

   # υπολογίζουμε τα blocking των κατηγοριών
   B_p,U,U1,Time,Ops,G,p,pk = calculate_EMLM_BR(capaCity,k,a,b,t_reservation)

   B_p_rounded = [round(x, 7) for x in B_p] # κάνουμε round γιατί χρειάζεται παρακάτω

   flag=1
   # σε ένα loop γίνονται οι απαραίτητοι έλεγχοι
   # και κάθε φορά αυξάνεται το ζητούμενο capacity
   while True:

    if (e==1.0):
       if any(xx==1.0 for xx in B_p_rounded): # εξετάζεται αν υπάρχει έστω και μια τιμή 1.0
         flag=0                               # στα blocking των κατηγοριών, οπότε στην ουσία
                                              # διακόπτουμε εκείνη τη στιγμή
                                              # Εδώ χρησιμοποιούμε  round γιατί
                                              # υπάρχει πρόβλημα στην σύγκριση ισότητας
                                              # float αριθμών. (Εδώ δεν είναι απαραίτητο
                                              # γιατί όλες οι πιθανές τιμές έχουν
                                              # άνω μέγιστη τιμή την 1.0)



    elif e > 0.0:
      max_value = max(B_p)    # κάθε φορά βρίσκουμε τη μέγιστη τιμή από όλες τις
      if max_value < e :      # κατηγορίες blocking και αυτή θα πρέπει να είναι μικρότερη
         flag=0               # από το όριο που έχει δώσει ο χρήστης, αν είναι διακόπτουμε



    elif e==0.0:                              # εξετάζουμε το όριο 0.0 (του χρήστη) για σύγκριση (==) με τι
       if all(xx==0.0 for xx in B_p_rounded):  # τιμές blocking των κατηγοριών και αν όλες οι τιμές είναι ίσ
          flag=0                              # δηλαδή είναι 0.0 διακόπτουμε.
                                              # Εδώ χρησιμοποιούμε το round γιατί υπάρχει πρόβλημα
                                              # στη σύγκριση ισότητας float αριθμών
```

```
    else:
      messagebox.showerror("Σφάλμα", "Το επιθυμητό Όριο Blocking πρέπει να είναι:\n\
Όριο >= 0.0 και Όριο <= 1.0")
      return

    if flag == 1:
      capaCity+=1
      B_p,U,U1,Time,Ops,G,p,pk = calculate_EMLM_BR(capaCity,k,a,b,t_reservation)
      B_p_rounded = [round(x, 7) for x in B_p]
    else:
      break

  return capaCity

# Η συνάρτηση που τρέχει το looping_capacity και εκτυπώνει το Capacity
def compute_capacity():
    global C

    C1=0
    # έλεγχος των δεδομένων του ορίου
    try:

      # στο E καταχωρείται η τιμή του χρήστη για το όριο
      E = float(entry_loop.get())

      if (E < 0.0 or E > 1.0):
        messagebox.showerror("Σφάλμα", "Το επιθυμητό Blocking πρέπει να είναι:\n\
Blocking >= 0.0 και Blocking <= 1.0")
        return

    except ValueError:
       messagebox.showerror("Σφάλμα","Το επιθυμητό Blocking πρέπει να είναι:\n\
Blocking >= 0.0 και Blocking <= 1.0\nΣΗΜΕΙΩΣΗ: Ο δεκαδικός πρέπει να έχει τελεία!")
       return

    if C != 0 :
        # κρατάμε το global Capacity C γιατί θα επηρεαστεί η τιμή του
        CapacitY = C

        C1 = looping_capacity(C,K,traffic,bandwidth,E,tk)
        Capacity_label.config(text=f"Capacity : {C1}")

        # επιστρέφομε την τιμή που είχε το global Capacity C
        # έτσι ο χρήστης έχει τη δυνατότητα για νέους υπολογισμούς
        # με νέο όριο , μιας και το global Capacity C παιρνει την τιμή
        # που είχε πριν από το looping (την "καθαρή" του τιμή)
        C =  CapacitY
    else :
        messagebox.showerror ("Σφάλμα","Ελέγξτε πρώτα τα δεδομένα σας με το κουμπί\n\
Ελεγχος και Υπολογισμός")
        return
```

EMLM : Τμήμα του κώδικα της προσομοίωσης

```
#=============================================================================
#=============== Η συνάρτηση simulate_EMLM_BR ================================

def simulate_EMLM_BR(calls,c,k,l,m,b,r,seed_l,seed_m,transiT):

    #=== Μέτρηση χρόνου διεκπεραίωσης ===
    start = time.perf_counter()

    q = [0]*(c+1) # θα μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[i] κατάσταση
                  # Προσοχή! , έχουμε και την 0 κατάσταση άρα από 0 , 1, 2,...c
                  # συνολικά c+1
    ii=0

    p = [0]*(c+1)# θα μετράμε τις αντίστοιχες πιθανότητες των q καταστάσεων
                  # Προσοχή! , έχουμε και την 0 κατάσταση άρα από 0 , 1, 2,...c
                  # συνολικά c+1


    # Ορίζουμε ξεχωριστούς Random generators για λ και μ
    seed_L = random.Random(seed_l)
    seed_M = random.Random(seed_m)
    #-------------------------------------------------

    # αρχικοποίηση μεταβλητών
    servers = c
    current_time = 0
    duration = 0.0

    # αρχικοποίηση πινάκων-λιστών
    blocked_calls = [0]*k
    allowed_calls = [0]*k
    arrival_time = [0.0]*k
    Blocking = [0.0]*k
```

```
    # λίστα αποθήκευσης των χρόνων των κλήσεων
    times = []

    # δημιουργία σωρού αξιοποιώντας τη λίστα times
    # ο σωρός θα τοποθετεί τον μικρότερο χρόνο - τιμή
    # στη θέση 0 της λίστας times
    heapq.heapify(times)


    tr = transiT     # transiT είναι η μεταβλητή που θα μεταφερθεί από το χρήστη και δηλώνει
                     # τον αριθμό των αρχικών  κλήσεων που δεν θα ληφθούν υπόψη.
                     # Αυτόν τον αριθμό τον μεταφέρουμε σε μία μεταβλητή tr που δουλεύει
                     # σαν 'αντίθετος χρονοδιακόπτης' ,
                     # δηλαδή μέχρι να μηδενιστεί εκτελείται το πρόγραμμα χωρίς να αποθηκεύεται
                     # κάτι, μετά το μηδενισμό του αρχίζει η μέτρηση και η εξαγωγή
                     # των αποτελεσμάτων.

  #-------------------------------------------------------------------------------------------------------


    # Αρχικοί χρόνοι αφίξεων για κάθε κατηγορία
    arrivals = [seed_L.expovariate(λ) for λ in l]


    for _ in range(calls):       # η κάτω παύλα στη for δηλώνει οτι
                                 # απλά χρησιμοποιούμε την επανάληψη
                                 # (loop) για calls φορές


      #========================== Άφιξη νέας κλήσης (Poisson) ======================================
          current_time = min(arrivals)        # βρίσκουμε κάθε φορά τη μικρότερη χρονικά άφιξη
                                              # αυτή η τιμή θα είναι κάθε φορά το current_time
```

```
      Ki = arrivals.index(current_time)    # βρίσκουμε τη θέση της μικρότερης τιμής των αφίξεων
                                           # που είναι και η κατηγορία της κλήσης


    #-------------------------------------------------------------------------------------
    # ελέγχει αν ειναι άδεια η λίστα times και μετά αν το στοιχείο στη θέση 0
    # ειναι μικρότερο ή ίσο του current_time πράγμα που σημαίνει ότι έχει τελειώσει.
    # Έτσι, κάνει εξαγωγή του στοιχείου που βρίσκεται στη μηδενική θέση της λίστας

    # Αυτό επαναλαμβάνεται για όλον το σωρό, εξάγοντας όλες τις κλήσεις που έχουν
    # λήξει χρονικά, έτσι αποδεσμεύονται servers, γι' αυτό αυξάνεται η τιμή των
    # διαθέσιμων servers κατά b[Ki] σε κάθε επανάληψη
      while times and times[0][0] <= current_time:

        timE,bandwidtH = heapq.heappop(times)

        # επειδή έφυγε το στοιχείο 0 της λίστας
        # αυξάνουμε τους διαθέσιμους servers κατά b[Ki]
        # τα b[Ki] έχουν αποθηκευτεί ως δεύτερη στήλη
        # μέσω του σωρού (min-heap), στο times[0][1]
        # δηλαδή πάνω πάνω θα βρίσκεται ο μικρότερος χρόνος timE με το αντίστοιχο bandwidtH

        servers += bandwidtH

    #-------------------------------------------------------------------------------------
      # Έλεγχος διαθεσιμότητας servers
      # επιπλέον, μαζί με το reservation της κατηγορίας
      if (servers > 0 and servers >= b[Ki]+r[Ki]):

        if tr <= 0:# Transit: όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
                   # τότε αρχίζουμε να παιρνουμε αποτελέσματα

          allowed_calls[Ki]+=1
```

```
        # μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[i] κατάσταση
        ii=c-servers
        q[ii]+=1

      # παραχωρούμε b[i] servers για διάρκεια duration
      servers -= b[Ki]

      # χρονική διάρκεια διεκπεραίωσης της κλήσης (duration) με βάση την
      # εκθετική κατανομή του μέσου ρυθμού εξυπηρέτησης μ
      duration = seed_M.expovariate(m[Ki])


      # τοποθετούμε στη λίστα-πίνακα times το current_time μαζί με τη διάρκεια duration
      # η λίστα είναι σωρός που στη θέση 0 τοποθετεί τη μικρότερη τιμή (min-heap)
      # στη δεύτερη στήλη καταχωρούμε το αντίστοιχο bandwidth που έχουμε παραχωρήσει,
      # αυτό το χρειαζόμαστε κατά την εξαγωγή pop να ξέρουμε τι bandwidth
      # θα πρέπει να επιστρέψουμε.
      heapq.heappush(times, ( current_time + duration , b[Ki])  )

    elif (servers == 0 or servers < b[Ki]+r[Ki]):
      if tr <=0 :# Transit : όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
                 # τότε αρχίζουμε να παιρνουμε αποτελέσματα

        # εδώ μπλοκάρεται η κλήση γιατί δεν υπάρχουν διαθέσιμοι servers
        blocked_calls[Ki] += 1

        # μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[i] κατάσταση
        # το c είναι το όρισμα της συνάρτησης για το capacity.
        ii=c-servers
        q[ii]+=1

    else:
     messagebox.showerror("Σφάλμα", "Σφάλμα με το πλήθος των servers")
     sys.exit()
```

```
            # εξάγουμε εκθετικά τον επόμενο χρόνο άφιξης της Ki κατηγορίας
            arrivals[Ki] += seed_L.expovariate(l[Ki])

            tr-=1 # μείωση σε κάθε επανάληψη του "χρονοδιακόπτη" κατά 1

    #=== Τέλος χρόνου διεκπεραίωσης ===
    end = time.perf_counter()


    xronos =(end - start)   # χρόνος διεκπεραίωσης του αλγόριθμου


    for i in range(k):
     if (allowed_calls[i] + blocked_calls[i]) > 0 :
       Blocking[i] = blocked_calls[i] / (allowed_calls[i] + blocked_calls[i])
     else:
         messagebox.showerror("Σφάλμα", f"Υπάρχει κάποιο σφάλμα με την καταχώρηση των κλήσεων\n\
της κατηγορίας {i} (allowed_calls{i} <===> blocked_calls{i})")
         sys.exit()


    # εδώ εξάγουμε τις αντίστοιχες πιθανότητες των καταστάσεων q
    U=0
    if calls-transiT >0 :
     for i in range(0,c+1):
       p[i] = q[i] / (calls - transiT)
       U+= i*p[i]                        # μέσος όρος για υπολογισμό του
                                          # utilization
    else:
       messagebox.showerror("Σφάλμα", f"Δώσατε Transit μεγαλύτερο ή ίσο με τον αριθμό των κλ;ησεων\n\
Transit= {transiT} και Calls= {calls}")
       sys.exit()
```

Παράρτημα Ε: Τμήματα κώδικα του μοντέλου *SRM/MRM*

SRM/MRM : Τμήμα του κώδικα του αναλυτικού μοντέλου

```
#-------------------------------------------------------------------------------------------------
############################## Υλοποίηση του αναδρομικού τύπου MRM_BR ##############################


for j in range (1,c+1):
  for i in range (1,k+1):
   if(j <= c-res[i-1]):   # για reservation

    ##########################################################################################
    # Το πρώτο άθροισμα του αναδρομικού τύπου
      if( (j-b[i-1] >= 0.0)):
           q[j] += (a[i-1]*b[i-1] * q[j-b[i-1]]) / j


    ##########################################################################################
    # Το δεύτερο άθροισμα του αναδρομικού τύπου
      for z in range (1,len(mr[i-1])):
        if flag[i-1]==1 and j-br[i-1][z]>=0 and br[i-1][z]>0 and mr[i-1][z]>0 and\
           j > c -(br[i-1][z-1] - br[i-1][z]):

             metritis1+=1
             q[j]+= (l[i-1]/mr[i-1][z])* br[i-1][z] * q[j-br[i-1][z]] / j


  G += q[j]

##########################################################################################
```

SRM/MRM : Τμήμα του κώδικα της προσομοίωσης

```
    # λίστα αποθήκευσης των χρόνων των κλήσεων
     times = []

    # δημιουργία σωρού αξιοποιώντας τη λίστα times
    # ο σωρός θα τοποθετεί τον μικρότερο χρόνο - τιμή
    # στη θέση 0 της λίστας times
     heapq.heapify(times)


     tr = transiT      # transiT είναι η μεταβλητή που θα μεταφερθεί από το χρήστη και δηλώνει
                       # τον αριθμό των αρχικών  κλήσεων που δεν θα ληφθούν υπόψη.
                       # Αυτόν τον αριθμό τον μεταφέρουμε σε μία μεταβλητή tr που δουλεύει
                       # σαν 'αντίθετος χρονοδιακόπτης' ,
                       # δηλαδή μέχρι να μηδενιστεί εκτελείται το πρόγραμμα χωρίς να αποθηκεύεται
                       # κάτι, μετά το μηδενισμό του αρχίζει η μέτρηση και η εξαγωγή
                       # των αποτελεσμάτων.

#-------------------------------------------------------------------------------------------------------------
     total_calls_per_K = [0]*k          # καταχώρηση των συνολικών κλήσεων ανά κατηγορία.


    # Αρχικοί χρόνοι αφίξεων για κάθε κατηγορία
     arrivals = [seed_L.expovariate(λ) for λ in l]

     for _ in range(calls):        # η κάτω παύλα στη for δηλώνει οτι
                                   # απλά χρησιμοποιούμε την επανάληψη
                                   # (loop) για calls φορές


      #============================ Άφιξη νέας κλήσης (Poisson) ====================================
          current_time = min(arrivals)         # βρίσκουμε κάθε φορά τη μικρότερη χρονικά άφιξη
                                               # αυτή η τιμή θα είναι κάθε φορά το current_time

          Ki = arrivals.index(current_time)    # βρίσκουμε τη θέση της μικρότερης τιμής των αφίξεων
                                               # που είναι και η κατηγορία της κλήσης

          total_calls_per_K[Ki]+=1             # καταχώρηση των συνολικών κλήσεων ανά κατηγορία

        #-------------------------------------------------------------------------------------
        # ελέγχει αν ειναι άδεια η λίστα times και μετά αν το στοιχείο στη θέση 0
        # ειναι μικρότερο ή ίσο του current_time πράγμα που σημαίνει ότι έχει τελειώσει.
        # Έτσι, κάνει εξαγωγή του στοιχείου που βρίσκεται στη μηδενική θέση της λίστας

        # Αυτό επαναλαμβάνεται για όλον το σωρό, εξάγοντας όλες τις κλήσεις που έχουν
        # λήξει χρονικά, έτσι αποδεσμεύονται servers, γι' αυτό αυξάνεται η τιμή των
        # διαθέσιμων servers κατά b[Ki] σε κάθε επανάληψη
          while times and times[0][0] <= current_time:

            timE,bandwidtH = heapq.heappop(times)

            # επειδή έφυγε το στοιχείο 0 της λίστας
            # αυξάνουμε τους διαθέσιμους servers κατά b[Ki]
            # τα b[Ki] έχουν αποθηκευτεί ως δεύτερη στήλη
            # μέσω του σωρού (min-heap), στο times[0][1]
            # δηλαδή πάνω πάνω θα βρίσκεται ο μικρότερος χρόνος timE με το αντίστοιχο bandwidtH

            # Ελεγχος κατά την αποδέσμευση bandwidth για να μην ξεπεράσουμε το C
            if( servers + bandwidtH <= c) :
             servers += bandwidtH
            else:
             messagebox.showerror("Σφάλμα", "Σφάλμα με την αποδέσμευση bandwidth --> Ξεπερνά το C= ",c)
             sys.exit()
```

```
#-------------------------------------------------------------------------------------
######### Έλεγχος διαθεσιμότητας servers μαζί με το reservation της κατηγορίας ##########
        #-------------------------------------------------------------------#


  j=c-servers # κατειλημμένες γραμμές


  flag1 = False  # Σημαφόρος που αρχικοποιείται να είναι κατεβασμένος


  # Εδώ ελέγχουμε αν έχουμε διαθέσιμο πλήθος servers για το
  # bandwidth που ζητάμε, συνυπολογίζοντας το αντίστοιχο
  # reservation.
  if servers >0 and servers<=c and servers >= b[Ki] + reserve[Ki]:
         metritis1+=1

         if tr <= 0:             # Transit: όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
                                 # τότε αρχίζουμε να παιρνουμε αποτελέσματα

            allowed_calls[Ki]+=1     # αποδεκτή κλήση

            # μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[i] κατάσταση
            q[j]+=1

         #παραχωρούμε b[i] server για διάρκεια duration
         servers -= b[Ki]

         # χρονική διάρκεια διεκπεραίωσης της κλήσης (duration) με βάση την
         # εκθετική κατανομή του μέσου ρυθμού εξυπηρέτησης μ
         duration = seed_M.expovariate(m[Ki])

         # τοποθετούμε στη λίστα-πίνακα times το current_time μαζί με τη διάρκεια duration
         # η λίστα είναι σωρός που στη θέση 0 τοποθετεί τη μικρότερη τιμή (min-heap)
         # στη δεύτερη στήλη καταχωρούμε το αντίστοιχο bandwidth
         # αυτό το χρειαζόμαστε κατά την εξαγωγή pop να ξέρουμε τι bandwidth
         # θα πρέπει να επιστρέψουμε.

         heapq.heappush(times, ( current_time + duration , b[Ki])  )

         flag1 = True # Εδώ η κλήση έγινε αποδεκτή ,
                      # έτσι σηκώνεται ο σημαφόρος (γίνεται True)



  # Αν η κατηγορία Ki δεν διαθέτει Retries flag[Ki]==0 και
  # ο σημαφόρος flag1 παραμένει κατεβασμένος (False)
  #(δηλαδή η κλήση δεν έχει γίνει αποδεκτή μέχρι εδώ)
  # σημαίνει ότι η κλήση ΜΠΛΟΚΑΡΕΤΑΙ.
  if (flag1 == False and flag[Ki]==0):
         metritis2+=1

         if tr <=0 :    # Transit : όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
                        # τότε αρχίζουμε να παιρνουμε αποτελέσματα

             # εδώ μπλοκάρεται η κλήση γιατί δεν υπάρχουν διαθέσιμοι servers
             blocked_calls[Ki] += 1

             # μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[j] κατάσταση
             q[j]+=1
```

```
# Αν η κατηγορία Ki διαθέτει Retries flag[Ki]==1 και
# ο σημαφόρος flag1 παραμένει κατεβασμένος (False)
# ξεκινάμε να εξετάζουμε τα Retries της κατηγορίας Ki
if(flag1 == False and flag[Ki]==1):
      metritis3+=1
      for z in range(len(br[Ki])):

         # εδώ ελέγχουμε αν έχουμε διαθέσιμο πλήθος servers για το
         # bandwidth που ζητάμε, συνυπολογίζοντας το αντίστοιχο
         # reservation.
         if servers>0 and servers<=c and servers >= br[Ki][z] + reserve[Ki]:
                  metritis4+=1
                  if tr <= 0:             # Transit: όταν ο "χρονοδιακόπτης"
                                          # έχει πλέον "μηδενιστεί"
                                          # τότε αρχίζουμε να παιρνουμε αποτελέσματα

                      allowed_calls_with_retry[Ki][z]+=1   # αποδεκτή κλήση


                      q[j]+=1             # μετράμε πόσες φορές το σύστημα
                                          # θα βρίσκεται στην q[j] κατάσταση


                  #παραχωρούμε br[Ki][z] server για διάρκεια duration
                  servers -= br[Ki][z]

                  # χρονική διάρκεια διεκπεραίωσης της κλήσης (duration) με βάση την
                  # εκθετική κατανομή του μέσου ρυθμού εξυπηρέτησης m[Ki][z]
                  duration = seed_M.expovariate(mr[Ki][z])

                  # τοποθετούμε στη λίστα-πίνακα times το current_time
                  # μαζί με τη διάρκεια duration
                  # η λίστα είναι σωρός που στη θέση 0 τοποθετεί τη μικρότερη τιμή (min-heap)
                  # στη δεύτερη στήλη καταχωρούμε το αντίστοιχο bandwidth
                  # αυτό το χρειαζόμαστε κατά την εξαγωγή pop να ξέρουμε τι bandwidth
                  # θα πρέπει να επιστρέψουμε.

                  heapq.heappush(times, ( current_time + duration , br[Ki][z])  )

                  flag1 = True        # Εδώ η κλήση έγινε αποδεκτή ,
                                      # έτσι σηκώνεται ο σημαφόρος (γίνεται True)

                  break               # Δεν χρειάζεται να συνεχιστεί η επανάληψη
                                      # μιας και έγινε αποδεκτή η κλήση
                                      # γι' αυτό τη διακόποτουμε με break


         metritis5+=1
         if tr <= 0: # Αν έχουμε φτάσει εδώ , σημαίνει ότι δεν έχει γίνει αποδεκτή
                     # η κλήση με το συγκεκριμένο Retry και γι αυτό αυξάνεται
                     # η παρακάτω μετρική.
                     # Σημείωση: Αν υπάρχουν και άλλα Retries η επανάληψη συνεχίζεται.
             blocked_calls_with_retry[Ki][z]+=1



      if(flag1 == False): # Αν παρ' όλα τα παραπάνω ο σημαφόρος flag1
                          # παραμένει κατεβασμένος (=False)
                          # τότε η κλήση δεν μπορεί να εξυπηρετηθεί και μπλοκάρεται


          metritis6+=1
          if tr <=0 :     # Transit : όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
                          # τότε αρχίζουμε να παιρνουμε αποτελέσματα

              # εδώ μπλοκάρεται η κλήση γιατί δεν υπάρχουν διαθέσιμοι servers
              blocked_calls[Ki] += 1

              # μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[i] κατάσταση
              # το c είναι το όρισμα της συνάρτησης για το capacity.
              q[j]+=1


# εξάγουμε εκθετικά τον επόμενο χρόνο άφιξης της Ki κατηγορίας
arrivals[Ki] += seed_L.expovariate(l[Ki])

tr-=1 # μείωση σε κάθε επανάληψη του "χρονοδιακόπτη" κατά 1
```

Παράρτημα ΣΤ: Τμήματα κώδικα του μοντέλου *STM/MTM*

STM/MTM Τμήμα του κώδικα του αναλυτικού μοντέλου

```
#-----------------------------------------------------------------------------------------------
######################## Υλοποίηση του αναδρομικού τύπου της φόρμουλας ############################
try:
 for j in range (1,c+1):
  for i in range (1,k+1):
   if(j<=c-res[i-1]): # με reservations

    ##############################################################################################
    # Το πρώτο άθροισμα του αναδρομικού τύπου
    if j-b[i-1]>=0 :
         if(flag[i-1] == 0): # Δηλαδή b_k_c_s ==0
             q[j] += a[i-1]*b[i-1]*q[j-b[i-1]] / j

         else : # Δηλαδή b_k_c_s > 0
           if(j <= thresholds[0] + b[i-1]):
               q[j] += a[i-1]*b[i-1]*q[j-b[i-1]] / j


    ##############################################################################################
    # Το δεύτερο άθροισμα του αναδρομικού τύπου
    # Υπενθυμίζουμε ότι αν flag == 1 σημαίνει ότι υπάρχει (bc, mc)

    for z in range(1,len(thresholds)+1):
     if flag[i-1]==1 :
      if j-bc[i-1][z-1]>=0 and bc[i-1][z-1]>0 and mc[i-1][z-1]>0 and\
        (j > thresholds[z-1] + bc[i-1][z-1]):

        if (z<len(thresholds)):
           if (j <= thresholds[z] + bc[i-1][z-1]):
             q[j]+= (l[i-1]/mc[i-1][z-1])*bc[i-1][z-1]*q[j-bc[i-1][z-1]] / j

           # Το τελευταίο κατώφλι είναι το c-res[i-1] --> j <= c-res[i-1]
           if (thresholds[z-1] > thresholds[z]):
              q[j]+= (l[i-1]/mc[i-1][z-1]) * bc[i-1][z-1] * q[j-bc[i-1][z-1]] / j
       # Το τελευταίο κατώφλι είναι το c-res[i-1] --> j <= c-res[i-1]
        if (z==len(thresholds)):
            q[j]+= (l[i-1]/mc[i-1][z-1])*bc[i-1][z-1]*q[j-bc[i-1][z-1]] / j


  G += q[j]
except Exception as e:
 messagebox.showerror("ERROR","Πιθανό Σφάλμα με τα κατώφλια!")
 sys.exit()
```

STM/MTM Τμήμα του κώδικα της προσομοίωσης

```
#########################################################################################################
############## Η συνάρτηση της φόρμουλας Single_Multiple_Threshold Simulator (MTM_BR_SIM)################

def simulate_Thresholds(calls,c,k,thrs,l,m,b,reserve,seed_l,seed_m,transiT,l_m_b,r_m_b,l_m_mi,r_m_mi):

    #======================== Μέτρηση χρόνου διεκπεραίωσης ============================
    start = time.perf_counter()

    q = [0]*(c+1) # θα μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[i] κατάσταση
                  # Προσοχή! , έχουμε και την 0 κατάσταση άρα από 0 , 1, 2,...c
                  # συνολικά c+1
    ii=0

    p = [0]*(c+1)# θα μετράμε τις αντίστοιχες πιθανότητες των q καταστάσεων
                  # Προσοχή! , έχουμε και την 0 κατάσταση άρα από 0 , 1, 2,...c
                  # συνολικά c+1


    # Ορίζουμε ξεχωριστούς Random generators για λ και μ
    seed_L = random.Random(seed_l)
    seed_M = random.Random(seed_m)
    #==========================================================================

    # αρχικοποίηση μεταβλητών
    servers = c    # servers: διαθέσιμοι servers
    current_time = 0
    duration = 0.0

    # αρχικοποίηση πινάκων-λιστών
    blocked_calls = [0]*k
    allowed_calls = [0]*k
    arrival_time = [0.0]*k

    Blocking = [0.0]*k

    ###########################################################################################################
    # Εδώ δημιουργούμε τους πίνακες bc και mc που περιλαμβάνουν τις πολλαπλές τιμές
    # bandwidth και μ, ανάλογα την κατηγορία κίνησης K και τα κατώφλια.

    try:
       # Έλεγχος αν ισχύει η σχέση: len(l_m_b) != len(l_m_mi)
       # θα πρέπει δηλαδή το πλήθος των αντίστοιχων παραμέτρων να συμβαδίζει.
       if len(l_m_b) != len(l_m_mi) :
           messagebox.showerror("ERROR1","Πιθανό Σφάλμα με την σωστή αντιστοίχιση των \n\
Threshold b και Threshold μ")
           sys.exit()

       # Οι πίνακες  bc και mc είναι δύο διαστάσεων.
       counter = [0]*k
       for i in range(0,k):
         for j in range(len(l_m_mi)):
           if i+1 == l_m_mi[j]:
              counter[i]+=1

       bc = [[] for _ in range(k)]
       mc = [[] for _ in range(k)]

       for i in range(k):
         bc[i] = [0] * counter[i]
         mc[i] = [0.0] * counter[i]


       # Η flag χρησιμοποιείται για τις περιπτώσεις που
       # υπάρχει κατώφλι που επηρεάζει την κατηγορία.
       # Κάτι αντίστοιχο με τον αναδρομικό τύπο
       # b_k_c_s ==0 ή b_k_c_s > 0

       flag = [0]*k
```

```
        for i in range (1,k+1):
          count=0
          for r in range(0,len(l_m_b)):
            if (l_m_b[r]==i):
              bc[i-1][count]=r_m_b[r]
              mc[i-1][count]=r_m_mi[r]
              count=count+1
              flag[i-1]=1 # Χρησιμοποιούμε τον πίνακα-λίστα flag για να
                          # ξέρουμε ποια κατηγορία K έχει κατώφλι.


        allowed_calls_with_thresholds = [[0 for j in range(len(bc[i]))] for i in range(k)]
        blocked_calls_with_thresholds = [[0 for j in range(len(bc[i]))] for i in range(k)]

        Multi_Blocking = [[0.0 for j in range(len(bc[i]))] for i in range(k)]
        Conditional_BP = [[0.0 for j in range(len(bc[i]))] for i in range(k)]
    except Exception as e:
      messagebox.showerror("ERROR2","Πιθανό Σφάλμα με την σωστή αντιστοίχιση των \n\
Thresholds , Threshold b και Threshold μ")
      sys.exit()

    ##############################################################################################

    # λίστα αποθήκευσης των χρόνων των κλήσεων
    times = []

    # δημιουργία σωρού αξιοποιώντας τη λίστα times
    # ο σωρός θα τοποθετεί τον μικρότερο χρόνο - τιμή
    # στη θέση 0 της λίστας times
    heapq.heapify(times)
    tr = transiT      # transiT είναι η μεταβλητή που θα μεταφερθεί από το χρήστη και δηλώνει
                      # τον αριθμό των αρχικών  κλήσεων που δεν θα ληφθούν υπόψη.
                      # Αυτόν τον αριθμό τον μεταφέρουμε σε μία μεταβλητή tr που δουλεύει
                      # σαν 'αντίθετος χρονοδιακόπτης' ,
                      # δηλαδή μέχρι να μηδενιστεί εκτελείται το πρόγραμμα χωρίς να αποθηκεύεται
                      # κάτι, μετά το μηδενισμό του αρχίζει η μέτρηση και η εξαγωγή
                      # των αποτελεσμάτων.


  #-------------------------------------------------------------------------------------------------
    total_calls_per_K = [0]*k # καταχώρηση των συνολικών κλήσεων ανά κατηγορία ,
                              # θα μας χρειαστεί στον υπολογισμό των Blockings

    # Αρχικοί χρόνοι αφίξεων για κάθε κατηγορία
    arrivals = [seed_L.expovariate(λ) for λ in l]


    for _ in range(calls):        # η κάτω παύλα στη for δηλώνει οτι
                                  # απλά χρησιμοποιούμε την επανάληψη
                                  # (loop) για calls φορές


      #========================== Άφιξη νέας κλήσης (Poisson) ====================================
          current_time = min(arrivals)        # βρίσκουμε κάθε φορά τη μικρότερη χρονικά άφιξη
                                              # αυτή η τιμή θα είναι κάθε φορά το current_time

          Ki = arrivals.index(current_time)   # βρίσκουμε τη θέση της μικρότερης τιμής των αφίξεων
                                              # που είναι και η κατηγορία της κλήσης

          total_calls_per_K[Ki]+=1
```

```
#-------------------------------------------------------------------------------------
# ελέγχει αν ειναι άδεια η λίστα times και μετά αν το στοιχείο στη θέση 0
# ειναι μικρότερο ή ίσο του current_time πράγμα που σημαίνει ότι έχει τελειώσει.
# Έτσι, κάνει εξαγωγή του στοιχείου που βρίσκεται στη μηδενική θέση της λίστας

# Αυτό επαναλαμβάνεται για όλον το σωρό, εξάγοντας όλες τις κλήσεις που έχουν
# λήξει χρονικά, έτσι αποδεσμεύονται servers, γι' αυτό αυξάνεται η τιμή των
# διαθέσιμων servers κατά b[Ki] σε κάθε επανάληψη
  while times and times[0][0] <= current_time:

    timE,bandwidtH = heapq.heappop(times)

    # επειδή έφυγε το στοιχείο 0 της λίστας
    # αυξάνουμε τους διαθέσιμους servers κατά b[Ki]
    # τα b[Ki] έχουν αποθηκευτεί ως δεύτερη στήλη
    # μέσω του σωρού (min-heap), στο times[0][1]
    # δηλαδή πάνω πάνω θα βρίσκεται ο μικρότερος χρόνος timE με το αντίστοιχο bandwidtH

    # Έλεγχος κατά την αποδέσμευση bandwidth για να μην ξεπεράσουμε το C
    if( servers + bandwidtH <= c) :
     servers += bandwidtH
    else:
     messagebox.showerror("Σφάλμα", "Σφάλμα με την αποδέσμευση bandwidth --> Ξεπερνά το C= ",c)
     sys.exit()

#-------------------------------------------------------------------------------------
 ######### Έλεγχος διαθεσιμότητας servers μαζί με το reservation της κατηγορίας ##########
         #-----------------------------------------------------------------------#

  j=c-servers # κατειλημμένες γραμμές

  # Σημαφόρος που θα μας χρειαστεί παρακάτω
  # κάθε φορά δηλαδή σε κάθε κλήση αρχικοποιείται
  # να ειναι κατεβασμένος (=False)
  flag1=False

  #=============================================================================================
  # Αρχικός έλεγχος, αν υπάρχει διαθέσιμος server και είτε δεν έχουμε κατώφλι
  # είτε έχουμε κατώφλι/α και είμαστε στα κανονικά πλαίσια του πρώτου κατωφλίου,

  if servers > 0  and servers <= c and servers >= b[Ki]+reserve[Ki]\
     and (flag[Ki]==0 or (flag[Ki]==1 and servers >= c-thrs[0])):

      if tr <= 0:# Transit: όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
                 # τότε αρχίζουμε να παιρνουμε αποτελέσματα

        # αποδεκτή κλήση
        allowed_calls[Ki]+=1


        # μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[j] κατάσταση
        q[j]+=1

      # παραχωρούμε b[Ki] servers για διάρκεια duration
      servers -= b[Ki]

      # χρονική διάρκεια διεκπεραίωσης της κλήσης (duration) με βάση την
      # εκθετική κατανομή του μέσου ρυθμού εξυπηρέτησης μ
      duration = seed_M.expovariate(m[Ki])


      # τοποθετούμε στη λίστα-πίνακα times το current_time μαζί με τη διάρκεια duration
      # η λίστα είναι σωρός που στη θέση 0 τοποθετεί τη μικρότερη τιμή (min-heap)
      # στη δεύτερη στήλη καταχωρούμε το αντίστοιχο bandwidth που έχουμε παραχωρήσει,
      # αυτό το χρειαζόμαστε κατά την εξαγωγή pop να ξέρουμε τι bandwidth
      # θα πρέπει να επιστρέψουμε.
      heapq.heappush(times, ( current_time + duration , b[Ki])  )
```

```
# αν για τη συγκεκριμένη κατηγορία κλήσης δεν υπάρχει κατώφλι
elif (flag[Ki]==0 and (servers<=0 or servers < b[Ki]+reserve[Ki] or servers>c)):
  if tr <=0 :# Transit : όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
             # τότε αρχίζουμε να παιρνουμε αποτελέσματα

       # εδώ μπλοκάρεται η κλήση γιατί δεν υπάρχουν διαθέσιμοι servers
       blocked_calls[Ki] += 1
       # μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[j] κατάσταση
       q[j]+=1


#=============================================================================================
# Όταν έχω ένα κατώφλι και το έχουμε ξεπεράσει --> Περίπτωση single threshold
elif ( flag[Ki]==1 and len(thrs)==1 ):

   # εδώ ΘΑ ΕΙΧΑΜΕ μπλοκ αν ΔΕΝ ΥΠΗΡΧΑΝ Thresholds
   # χρησιμοποιούμε απλά μία μετρική
   #blocked_calls[Ki] += 1

   if(servers>0  and servers >= bc[Ki][0]+reserve[Ki] and servers < c-thrs[0])  :

     if tr <= 0:# Transit: όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
                # τότε αρχίζουμε να παιρνουμε αποτελέσματα

       # αποδεκτή κλήση με ένα κατώφλι
       allowed_calls_with_thresholds[Ki][0]+=1
       # μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[j] κατάσταση
       q[j]+=1

     # παραχωρούμε bc[Ki][0] servers για διάρκεια duration
     servers -= bc[Ki][0]

     # χρονική διάρκεια διεκπεραίωσης της κλήσης (duration) με βάση την
     # εκθετική κατανομή του μέσου ρυθμού εξυπηρέτησης mc[Ki][0]
     duration = seed_M.expovariate(mc[Ki][0])
     heapq.heappush(times, ( current_time + duration , bc[Ki][0])  )

   else:
      if tr <=0 :# Transit : όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
                 # τότε αρχίζουμε να παιρνουμε αποτελέσματα

       # εδώ μπλοκάρεται η κλήση γιατί δεν υπάρχουν διαθέσιμοι servers
       blocked_calls_with_thresholds[Ki][0]+=1
       # μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[j] κατάσταση
       q[j]+=1


#=============================================================================================
# Όταν έχω περισσότερα από ένα threshold --> περίπτωση multi thresholds

elif (flag[Ki]==1 and len(thrs)>1 ):

 try:

  flag1=False                 # Αρχικοποιούμε τον σημαφόρο ώστε να είναι 0 δηλαδή κατεβασμένος

  for z in range (len(thrs)): # Επαναληπτικά εξετάζονται τα διαστήματα μεταξύ των κατωφλιών

   if z+1 <= len(thrs)-1 :    # ΠΡΟΣΟΧΗ! Το τελευταίο διάστημα εξετάζεται ξεχωριστά παρακάτω

    if(servers>0 and  servers >= bc[Ki][z]+reserve[Ki] and \
       servers >= c-thrs[z+1] and servers < c-thrs[z]):
```

```
    if tr <= 0:# Transit: όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
               # τότε αρχίζουμε να παιρνουμε αποτελέσματα

      # αποδεκτή κλήση με συγκεκριμένο κατώφλι
      allowed_calls_with_thresholds[Ki][z]+=1


      flag1=True            # Αν η κλήση έγινε αποδεκτή τότε σηκώνεται ο σημαφόρος (γίνεται 1)



      # μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[j] κατάσταση
      q[j]+=1

    # παραχωρούμε bc[Ki[z] servers για διάρκεια duration
    servers -= bc[Ki][z]

    # χρονική διάρκεια διεκπεραίωσης της κλήσης (duration) με βάση την
    # εκθετική κατανομή του μέσου ρυθμού εξυπηρέτησης mc[Ki][z]
    duration = seed_M.expovariate(mc[Ki][z])
    heapq.heappush(times, ( current_time + duration , bc[Ki][z])  )

    break  # Εφόσον, η κλήση γίνεται δεκτή δεν χρειάζεται να συνεχίσουμε
           # και κάνουμε break στην επανάληψη .

#-------------------------------------------------------------------------
# έλεγχος του τελευταίου διαστήματος με την τιμή του τελευταίου κατωφλιού
elif z+1 > len(thrs)-1 :

 if  (servers>0 and  servers >= bc[Ki][z]+reserve[Ki] and servers < c-thrs[z]):

  if tr <= 0:# Transit: όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
             # τότε αρχίζουμε να παιρνουμε αποτελέσματα


    # αποδεκτή κλήση με το τελευταίο κατώφλι
    allowed_calls_with_thresholds[Ki][z]+=1

    flag1=True # Αν η κλήση έγινε αποδεκτή τότε σηκώνεται ο σημαφόρος (γίνεται 1)

    # μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[j] κατάσταση
    q[j]+=1

  # παραχωρούμε bc[Ki][z] servers για διάρκεια duration
  servers -= bc[Ki][z]

  # χρονική διάρκεια διεκπεραίωσης της κλήσης (duration) με βάση την
  # εκθετική κατανομή του μέσου ρυθμού εξυπηρέτησης mc[Ki][z]
  duration = seed_M.expovariate(mc[Ki][z])
  heapq.heappush(times, ( current_time + duration , bc[Ki][z])  )

  break

# Αν ο σημαφόρος παραμένει κατεβασμένος (0) σημαίνει ότι στους παραπάνω ελέγχους ΜΕΧΡΙ ΤΩΡΑ
# η κλήση δεν έχει γίνει αποδεκτή, οπότε αυξάνεται η αντίστοιχη μετρικη των
# blocked_calls_with_thresholds
if (flag1==False) :
  if tr <=0 :# Transit : όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
             # τότε αρχίζουμε να παιρνουμε αποτελέσματα

    # εδώ μετραμε το blocking της κλήσης σε συγκεκριμένο κατώφλι
    # η κλήση μπορεί να δοκιμάσει με άλλο κατώφλι , αν υπάρχει.
    blocked_calls_with_thresholds[Ki][z]+=1
```

```
            # Αν τελικά μετά από ΟΛΟΥΣ τους ελέγχους δεν έχει σηκωθεί ο σημαφόρος flag1
            # τότε η κλήση μπλοκάρεται αυξάνοντας τη μετρική blocked_calls
            if (flag1==False) :
               if tr <=0 :# Transit : όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
                          # τότε αρχίζουμε να παιρνουμε αποτελέσματα

                  # εδώ μπλοκάρεται η κλήση γιατί δεν υπάρχουν διαθέσιμοι servers
                  # και έχουν εξαντληθεί όλα τα κατώφλια
                  blocked_calls[Ki]+=1

                  # μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[j] κατάσταση
                  # j είναι οι κατειλημμένες γραμμές
                  q[j]+=1

           except Exception as e:
             messagebox.showerror("Σφάλμα", f"Πρόβλημα με Thresholds ή Reservations\n\
και τις κατειλημμένες γραμμές C!")
             sys.exit()


          else:
             messagebox.showerror("Σφάλμα", "Σφάλμα με το πλήθος των servers!!!")
             sys.exit()

          # εξάγουμε εκθετικά τον επόμενο χρόνο άφιξης της Ki κατηγορίας
          arrivals[Ki] += seed_L.expovariate(l[Ki])

          tr-=1 # μείωση σε κάθε επανάληψη του "χρονοδιακόπτη" κατά 1

    #=== Τέλος χρόνου διεκπεραίωσης ===
    end = time.perf_counter()
```

Παράρτημα Z: Τμήματα κώδικα του μοντέλου *CDTM*

CDTM : Τμήμα του κώδικα του αναλυτικού μοντέλου

```
#-----------------------------------------------------------------------------------------------
################### Υλοποίηση του αναδρομικού τύπου της φόρμουλας CDTM ############################
try:

 for j in range (1,c+1):
  for i in range (1,k+1):
   if(j<=c-res[i-1]): # για reservation

    ##########################################################################################
    # Το πρώτο άθροισμα του αναδρομικού τύπου

    if j-b[i-1]>=0:

      if(flag[i-1] == 0): # Δηλαδή b_k_c_s ==0
          q[j] += a[i-1]*b[i-1]*q[j-b[i-1]] / j

      else : # Δηλαδή b_k_c_s > 0
         if (j <= threshc[i-1][0] + b[i-1]):
           q[j] += a[i-1]*b[i-1]*q[j-b[i-1]] / j


    ##########################################################################################
    # Το δεύτερο άθροισμα του αναδρομικού τύπου
    # Υπενθυμίζουμε ότι αν flag == 1 σημαίνει ότι υπάρχει (bc, mc)

    for z in range(1,len(threshc[i-1])+1):
      metritis1+=1

      if flag[i-1]==1 :
        if j > threshc[i-1][z-1] + bc[i-1][z-1] and \
             j-bc[i-1][z-1]>=0 and bc[i-1][z-1]>0 and mc[i-1][z-1]>0 :

            if z<len(threshc[i-1]):
```

```
              if (j <= threshc[i-1][z] + bc[i-1][z-1]):
                  q[j]+= (l[i-1]/mc[i-1][z-1]) * bc[i-1][z-1] * q[j-bc[i-1][z-1]] / j

              # Το τελευταίο κατώφλι είναι το c-res[i-1] --> j <= c-res[i-1]
              if (threshc[i-1][z-1] > threshc[i-1][z]):
                  q[j]+= (l[i-1]/mc[i-1][z-1]) * bc[i-1][z-1] * q[j-bc[i-1][z-1]] / j

            # Το τελευταίο κατώφλι είναι το c-res[i-1] --> j <= c-res[i-1]
            if (z==len(threshc[i-1])):
                q[j]+= (l[i-1]/mc[i-1][z-1]) * bc[i-1][z-1] * q[j-bc[i-1][z-1]] / j

  G += q[j]
except Exception as e:
 messagebox.showerror("ERROR","Πιθανό Σφάλμα με τα κατώφλια!")
 sys.exit()
```

CDTM : Τμήμα του κώδικα της προσομοίωσης

```
    # λίστα αποθήκευσης των χρόνων των κλήσεων
    times = []

    # δημιουργία σωρού αξιοποιώντας τη λίστα times
    # ο σωρός θα τοποθετεί τον μικρότερο χρόνο - τιμή
    # στη θέση 0 της λίστας times
    heapq.heapify(times)


    tr = transiT      # transiT είναι η μεταβλητή που θα μεταφερθεί από το χρήστη και δηλώνει
                      # τον αριθμό των αρχικών  κλήσεων που δεν θα ληφθούν υπόψη.
                      # Αυτόν τον αριθμό τον μεταφέρουμε σε μία μεταβλητή tr που δουλεύει
                      # σαν 'αντίθετος χρονοδιακόπτης' ,
                      # δηλαδή μέχρι να μηδενιστεί εκτελείται το πρόγραμμα χωρίς να αποθηκεύεται
                      # κάτι, μετά το μηδενισμό του αρχίζει η μέτρηση και η εξαγωγή
                      # των αποτελεσμάτων.


#-------------------------------------------------------------------------------------------------------
    total_calls_per_K = [0]*k # καταχώρηση των συνολικών κλήσεων ανά κατηγορία


    # Αρχικοί χρόνοι αφίξεων για κάθε κατηγορία
    arrivals = [seed_L.expovariate(λ) for λ in l]


    for _ in range(calls):        # η κάτω παύλα στη for δηλώνει οτι
                                  # απλά χρησιμοποιούμε την επανάληψη
                                  # (loop) για calls φορές


      #========================== Άφιξη νέας κλήσης (Poisson) ====================================
          current_time = min(arrivals)         # βρίσκουμε κάθε φορά τη μικρότερη χρονικά άφιξη
                                               # αυτή η τιμή θα είναι κάθε φορά το current_time

          Ki = arrivals.index(current_time)    # βρίσκουμε τη θέση της μικρότερης τιμής των αφίξεων
                                               # που είναι και η κατηγορία της κλήσης

          total_calls_per_K[Ki]+=1             # καταχώρηση των συνολικών κλήσεων ανά κατηγορία


        #-----------------------------------------------------------------------------------
        # ελέγχει αν ειναι άδεια η λίστα times και μετά αν το στοιχείο στη θέση 0
        # ειναι μικρότερο ή ίσο του current_time πράγμα που σημαίνει ότι έχει τελειώσει.
        # Έτσι, κάνει εξαγωγή του στοιχείου που βρίσκεται στη μηδενική θέση της λίστας

        # Αυτό επαναλαμβάνεται για όλον το σωρό, εξάγοντας όλες τις κλήσεις που έχουν
        # λήξει χρονικά, έτσι αποδεσμεύονται servers, γι' αυτό αυξάνεται η τιμή των
        # διαθέσιμων servers κατά b[Ki] σε κάθε επανάληψη
          while times and times[0][0] <= current_time:

            timE,bandwidtH = heapq.heappop(times)

            # επειδή έφυγε το στοιχείο 0 της λίστας
            # αυξάνουμε τους διαθέσιμους servers κατά b[Ki]
            # τα b[Ki] έχουν αποθηκευτεί ως δεύτερη στήλη
            # μέσω του σωρού (min-heap), στο times[0][1]
            # δηλαδή πάνω πάνω θα βρίσκεται ο μικρότερος χρόνος timE με το αντίστοιχο bandwidtH

            # Ελεγχος κατά την αποδέσμευση bandwidth για να μην ξεπεράσουμε το C
            if( servers + bandwidtH <= c) :
             servers += bandwidtH
            else:
             messagebox.showerror("Σφάλμα", "Σφάλμα με την αποδέσμευση bandwidth --> Ξεπερνά το C= ",c)
             sys.exit()
```

```
#-------------------------------------------------------------------------------------------
######### Έλεγχος διαθεσιμότητας servers μαζί με το reservation της κατηγορίας ##########
        #-------------------------------------------------------------------#

  # Σημαφόρος που θα μας χρειαστεί παρακάτω
  # κάθε φορά δηλαδή σε κάθε κλήση αρχικοποιείται
  # να ειναι κατεβασμένος (=False)
  flag1= False

  j=c-servers # κατειλημμένες γραμμές


  # Αρχικός έλεγχος: ΕΝΤΟΣ ΟΡΙΩΝ διάθεσης servers
  # είτε δεν έχουμε κατώφλι --> flag[Ki]==0
  # και είτε έχουμε κατώφλι/α --> flag[Ki]==1 και είμαστε στα κανονικά πλαίσια του πρώτου κατωφλίου,
  # σ'αυτές τις περιπτώσεις γίνεται ΑΠΟΔΟΧΗ της κλήσης και παραχωρείται το b[Ki]
  # για χρόνο με ρυθμό m[Ki].

  #-------------------------------------------------------------------------------------------
  # Συγκεκριμένα, εδώ ελέγχουμε αν έχουμε διαθέσιμο πλήθος servers για το
  # bandwidth που ζητάμε, συνυπολογίζοντας το αντίστοιχο
  # reservation και δεν έχουμε κατώφλι/α --> flag[Ki]==0
  # Αν ισχύουν οι παραπάνω προυποθέσεις τότε η κλήση γίνεται αποδεκτή
  # παραχωρόντας b[Ki] με ρυθμό εξυπηρέτησσς μ[Ki]
  if servers > 0 and servers <=c and servers >= b[Ki]+reserve[Ki] and\
      flag[Ki]==0 :

           metritis1+=1
           if tr <= 0:  # Transit: όταν ο "χρονοδιακόπτης" έχει πλέον "μηδενιστεί"
                        # τότε αρχίζουμε να παιρνουμε αποτελέσματα

               # αποδεκτή κλήση
               allowed_calls[Ki]+=1


               # μετράμε πόσες φορές το σύστημα θα βρίσκεται στην q[j] κατάσταση
               q[j]+=1

           # παραχωρούμε b[Ki] servers για διάρκεια duration
           servers -= b[Ki]

           # χρονική διάρκεια διεκπεραίωσης της κλήσης (duration) με βάση την
           # εκθετική κατανομή του μέσου ρυθμού εξυπηρέτησης μ
           duration = seed_M.expovariate(m[Ki])


           # τοποθετούμε στη λίστα-πίνακα times το current_time μαζί με τη διάρκεια duration
           # η λίστα είναι σωρός που στη θέση 0 τοποθετεί τη μικρότερη τιμή (min-heap)
           # στη δεύτερη στήλη καταχωρούμε το αντίστοιχο bandwidth που έχουμε παραχωρήσει,
           # αυτό το χρειαζόμαστε κατά την εξαγωγή pop να ξέρουμε τι bandwidth
           # θα πρέπει να επιστρέψουμε.
           heapq.heappush(times, ( current_time + duration , b[Ki])  )

           ######## ΣΗΜΑΝΤΙΚΟ ####################
           # Ενεργοποιούμε έναν σημαφόρο (=True)
           # ώστε να μην συνεχίσει ο κώδικας
           # με άλλους ελέγχους μιας και η κλήση
           # έγινε αποδεκτή
           flag1 = True


  # Αν η κατηγορία Ki δεν διαθέτει Thresholds flag[Ki]==0 και
  # ο σημαφόρος flag1 παραμένει κατεβασμένος (False)
  #(δηλαδή η κλήση δεν έχει γίνει αποδεκτή μέχρι εδώ)
  # σημαίνει ότι η κλήση ΜΠΛΟΚΑΡΕΤΑΙ.
  if flag1 == False and flag[Ki] == 0:
           metritis2+=1
           if tr <= 0:
```

```
                blocked_calls[Ki]+=1
                q[j]+=1

#-------------------------------------------------------------------------------------------------------
# Αν υπάρχουν κατώφλι/α --> flag[Ki]==1  αλλά η κλήση μέχρι εδώ δεν έχει γίνει αποδεκτή
# δηλαδή ο σημαφόρος flag1==False είναι κατεβασμένος, τότε ξεκινάμε να εξετάζουμε τις
# περιπτώσεις με τα κατώφλια της κατηγορίας Ki
if flag1 == False and flag[Ki]!=0 :
      metritis3+=1
      # Σημαφόρος που θα μας χρειαστεί παρακάτω
      # κάθε φορά δηλαδή σε κάθε κλήση αρχικοποιείται
      # να ειναι κατεβασμένος (=False)
      flag2=False

      # Εδώ είμαστε στα κανονικά πλαίσια του πρώτου κατωφλίου (πριν από το 1ο κατώφλι)
      # και ελέγχουμε αν έχουμε διαθέσιμο πλήθος servers για το bandwidth της κατηγορίας
      # συνυπολογίζοντας το αντίστοιχο reservation.
      # Υπενθυμίζουμε ότι j είναι c-servers δηλαδή οι κατειλημμένοι servers
      # Αν η κλήση γίνει αποδεκτή τότε της παραχωρείται b[Ki] και μ[Ki]
      # αλλιώς ξεκινά η διερεύνηση στα επόμενα στάδια κατωφλιών.
      if servers > 0 and servers <=c and servers >= b[Ki]+reserve[Ki] and\
         j <= threshc[Ki][0]:

            metritis4+=1
            if tr <= 0:

               allowed_calls[Ki]+=1                # Έχουμε αποδεκτή  κλήση

               q[j]+=1                             # Εδώ μετράμε πόσες φορές το σύστημα θα
                                                   # βρίσκεται στην q[j] κατάσταση


               servers -= b[Ki]                    # παραχώριση του αντίστοιχου bandwidth

               duration = seed_M.expovariate(m[Ki])  # παραχώριση του αντίστοιχου

                heapq.heappush(times, ( current_time + duration , b[Ki])  ) # push στο σωρό

                flag2=True                                 # σημαφόρος για το αν έχουμε αποδοχή (=True)
                                                           # της κλήσης πριν το πρώτο κατώφλι

            #else:              # Εδώ υπενθυμίζουμε ότι εξετάζεται το πρώτο κομμάτι κάτω από το κατώφλι 0
                                # της κάθε κατηγορίας και αν βρεθεί ότι δεν ικανοποιεί τις
                                # προυποθέσεις αυξάνεται η μετρική για blocked_calls_with_thresholds[Ki][0]
                                # σημειώνουμε ότι Ki είναι η εκάστοτε κατηγορία.

              #if tr <= 0:

                   #blocked_calls_with_thresholds[Ki][0]+=1


      if flag2==False :               # Εδώ ο σημαφόρος flag2 (=False) μας δηλώνει
                                      # ότι πριν το πρώτο κατώφλι δεν έχουμε αποδοχή
                                      # και πρέπει να συνεχίστει η διερεύνηση στα
                                      # επόμενα διαστήματα κατωφλιών.
                                      # ( δηλαδή πάνω από το πρώτο κατώφλι )

           ##########################################################################
           #if tr <= 0:

                 #blocked_calls_with_thresholds[Ki][0]+=1
           ##########################################################################

           metritis5+=1
           flag3=False    # Όπως και πριν ο σημαφόρος αυτός αν γίνει True
                          # σε κάποιο από τα στάδια διερεύνησης
                          # σημαίνει ότι η κλήση έχει γίνει αποδεκτή.
                          # Αν μετά τις επαναλήψεις παραμένει False
                          # τότε σημαίνει ότι η κλήση μπλοκάρεται.
```

```
            # με την επανάληψη, ξεκινάμε να εξετάζουμε τις περιπτώσεις με τα υπόλοιπα κατώφλια.
            for i in range (len(bc[Ki])) :
                metritis6+=1
                # Έτσι, έχουμε τα νέα  όρια κατωφλιών με ανώτερο το c
                # και ταυτόχρονα, ελέγχουμε αν έχουμε διαθέσιμο πλήθος servers
                # για το bandwidth που ζητάμε, συνυπολογίζοντας το
                # αντίστοιχο reservation.
                # Αν η κλήση γίνει αποδεκτή τότε της παραχωρείται bc[Ki][i] για mc[Ki][i]
                # και γίνεται break στην επανάληψη.
                # Αλλιώς αν δεν γίνει break αυξάνεται η μετρική
                # blocked_calls_with_thresholds[Ki][i] κάθε φορά και
                # συνεχίζουμε τη διερεύνηση στα επόμενα στάδια κατωφλιών
                if servers>0 and servers <=c and servers >= bc[Ki][i]+reserve[Ki] and\
                   j > threshc[Ki][i] and j <= threshc[Ki][i+1]:
                      metritis7+=1
                      if tr <= 0:

                         allowed_calls_with_thresholds[Ki][i]+=1  # Αποδεκτή  κλήση για τα νέα όρια


                         q[j]+=1                                  # μετράμε πόσες φορές το σύστημα
                                                                  # θα βρίσκεται στην q[j] κατάσταση


                         servers -= bc[Ki][i]                     # παραχωρούμε bc[Ki][i] servers
                                                                  # για διάρκεια duration


                         # χρονική διάρκεια διεκπεραίωσης της κλήσης (duration) με βάση την
                         # εκθετική κατανομή του μέσου ρυθμού εξυπηρέτησης mc[Ki][i]
                         duration = seed_M.expovariate(mc[Ki][i])

                         # τοποθετούμε στη λίστα-πίνακα times το current_time μαζί με τη διάρκεια duration
                         # η λίστα είναι σωρός που στη θέση 0 τοποθετεί τη μικρότερη τιμή (min-heap)
                         # στη δεύτερη στήλη καταχωρούμε το αντίστοιχο bandwidth που έχουμε παραχωρήσει,
                         # αυτό το χρειαζόμαστε κατά την εξαγωγή pop να ξέρουμε τι bandwidth
                         # θα πρέπει να επιστρέψουμε.
                         heapq.heappush(times, ( current_time + duration , bc[Ki][i])  )

                         flag3 = True # ενεργοποιείται ο σημαφόρος flag3

                         break   # Δεν χρειάζεται να συνεχιστεί η επανάληψη μιας και έγινε αποδεκτή η κλήση
                                 # γι' αυτό τη διακόποτουμε με break


                # Αν δεν έχει γίνει break μέχρι εδώ, σημαίνει ότι δεν έχει γίνει αποδεκτή η κλήση
                # στο συγκεκριμένο πλαίσιο ορίων των κατωφλίων, οπότε αυξάνεται η αντίστοιχη μετρική
                # blocking --> blocked_calls_with_thresholds[Ki][i]
                # και συνεχίζουμε για την επόμενη επανάληψη, δηλαδή στα επόμενα στάδια διερεύνησης.
                if tr <= 0:
                          metritis8+=1
                          blocked_calls_with_thresholds[Ki][i]+=1


            if flag3 == False :      # Εδώ σημαίνει ότι μετά τις παραπάνω επαναλήψεις
                                     # δεν είχαμε διαθέσιμους servers οπότε γίνεται
                                     # block της κλήσης της συγκεκριμένης κατηγορίας
                metritis9+=1
                if tr <= 0:
                    blocked_calls[Ki]+=1
                    q[j]+=1
#=============================================================================================

 # εξάγουμε εκθετικά τον επόμενο χρόνο άφιξης της Ki κατηγορίας
 arrivals[Ki] += seed_L.expovariate(l[Ki])

 tr-=1 # μείωση σε κάθε επανάληψη του "χρονοδιακόπτη" κατά 1
```

Υπεύθυνη Δήλωση Συγγραφέα:

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης.