Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και
Μηχανικων Η/Υ

Εργασια στο Μεταπτυχιακο Μαθημα
Προχωρημενα Θεματα Προσομοιωσης Συστηματων

Προσομοίωση Πρωτοκόλλου Πολλαπλής Πρόσβασης σε Πολυδιαυλικά Δίκτυα

Δημήτριος Πολίτης (ΥΔ)

Επιβλέπων
Καθ. Ευστάθιος Συκάς

11 Απριλίου 2017

Περιεχόμενα

Κατάλογος Πινάκων

Κατάλογος Σχημάτων

Περίληψη

Στο παρόν μελετώνται τα ποιοτικά και ποσοτικά χαρακτηριστικά ενός πρωτοκόλλου πολλαπλής πρόσβασης τύπου (slotted aloha) σε πολυδιαυλικό δίκτυο, με τη βοήθεια μοντελοποίησης της κίνησης, βάση συγκεκριμένων παραδοχών. Η απόδοση του εν λόγω πρωτοκόλλου αξιολογείται, σε σχέση με τη συμφόρηση που δημιουργείται λόγω του πλήθους των σταθμών και με την καθυστέρηση μετάδοσης των πακέτων, λόγω συγκρούσεων, είτε εντός των διαύλων μετάδοσης είτε στον προορισμό.

Η μοντελοποίηση υπολογίζει τα μέτρα αποδοτικότητας του πρωτοκόλλου (διέλευση, καθυστέρηση) με βάση το πλήθος των σταθμών εκπομπής, το πλήθος των διαύλων επικοινωνίας (κανάλια) και το πλήθος των συντονιζόμενων πομπών ανά δέκτη.

Τα αποτελέσματα της προσομοίωσης καταδεικνύουν ότι, τα μέτρα αποδοτικότητας επηρεάζονται θετικά στην αύξηση του πλήθους των διαύλων ενώ η αύξηση του πλήθους των συντονιζόμενων πομπών ανά δέκτη δεν επιφέρει ορατά αποτελέσματα από ένα όριο και πάνω.

Λέξεις κλειδιά: Προσομοίωση, Πολυδιαυλικά δίκτυα, Συγκρούσεις, Πλήθος σταθμών, Πλήθος διαύλων, Εκτίμηση απόδοσης.

1 Εισαγωγή

1.1 Εισαγωγή

Τα δίκτυα Η/Υ ενσωματώνονται όλο και περισσότερο στην καθημερινότητά μας. Καθώς αυτή η διαδικασία προχωρά, η ανάγκη για μετάδοση μεγάλου όγκου πληροφοριών με τη μέγιστη δυνατή ταχύτητα γίνεται όλο και πιο σημαντική. Στα παραπάνω θα πρέπει να συνυπολογιστεί και η μεγάλη κινητικότητα που προσφέρουν οι συσκευές κινητής τηλεφωνίας, οι φορητοί Η/Υ και η εισαγωγή τα τελευταία χρόνια του διαδικτύου των αντικειμένων (Internet of Things). Όλα τα παραπάνω δημιουργούν συνεχώς νέες τεχνολογικές προκλήσεις στο τομέα της μετάδοσης δεδομένων.

Οι προκλήσεις αυτές αφορούν σε δύο κατευθύνσεις: αφενός μεν στη μεγιστοποίηση της ροής των δεδομένων εντός του δικτύου στη μονάδα του χρόνου, αφετέρου στην καλύτερη εξυπηρέτηση συνεχώς μετακινούμενων χρηστών, οι οποίοι εισέρχονται και εξέρχονται του δικτύου σε τυχαίους χρόνους (Quality of Experience σε αντιπαραβολή με το Quality of Service των στατικών δικτύων).

Για την αντιμετώπιση των προκλήσεων αυτών είναι απαραίτητη η εξέλιξη των αλγορίθμων κωδικοποίησης των δεδομένων, τεχνικών πολύπλεξης (TDM, FDM) αλλά και πρωτοκόλλων μετάδοσης και ελέγχου πρόσβασης του φυσικού μέσου (Medium Access Control, MAC).

Η παρούσα μελέτη εστιάζει στην αξιολόγηση της απόδοσης του πρωτοκόλλου πολλαπλής πρόσβασης τύπου Slotted Aloha. Τέτοιας μορφής πρωτόκολλα βρίσκουν εφαρμογή τόσο σε στατικά δίκτυα οπτικών ινών (WDM), όσο και σε ασύρματα δίκτυα με πολλούς χρήστες (π.χ. στην κινητή τηλεφωνία, δίκτυα 2.5G -3G) [1]. Τα κύρια σημεία της μελέτης και η ροή αυτής είναι όπως παρακάτω: Αρχικά, περιγράφονται επιγραμματικά αρχές που αφορούν στις προσομοιώσεις, (στοιχεία που αφορούν στην προσομοίωση συστημάτων, ουρές αναμονής). Στη συνέχεια, αναλύεται θεωρητικά το προς προσομοίωση πρωτόκολλο, περιγράφεται το μοντέλο της προσομοίωσης με τις παραδοχές του και ο κώδικας της προσομοίωσης με τα κύρια σημεία του. Τέλος, γίνεται η παρουσίαση των αποτελεσμάτων της προσομοίωσης, συγκρίνονται αυτά με τα αναμενόμενα θεωρητικά και εξάγονται τα συμπεράσματα της μελέτης.

1.2 Προσομοίωση Συστημάτων

1.2.1 Εισαγωγή

Ο όρος προσομοίωση ορίζει την αναπαραγωγή μιας διαδικασίας ή συστήματος, συνήθως του πραγματικού κόσμου, σε σχέση με το χρόνο. Η αναπαραγωγή αυτή γίνεται πάντα με συγκεκριμένες παραδοχές, ανάλογα με το σύστημα που θέλουμε να προσομοιώσουμε και την επιθυμητή ακρίβεια. Βασικός σκοπός της προσομοίωσης είναι εξαγωγή συμπερασμάτων σε σχέση με το σύστημα και τη συμπεριφορά του υπό διαφορετικές συνθήκες.

1.2.2 Πλεονεκτήματα και Μειονεκτήματα

Οι κύριοι λόγοι για τους οποίους χρησιμοποιείται η προσομοίωση διαδικασιών είναι κυρίως οι παρακάτω:

- Λόγοι που έχουν να κάνουν με το κόστος η τη δυσκολία υλοποίησης του πραγματικού συστήματος και αξιόπιστων δοκιμών.
- Λόγοι επικινδυνότητας υλοποίησης δοκιμών σε πραγματικό περιβάλλον.
- Όταν τα προς προσομοίωση συστήματα δεν είναι πραγματικά (conceptual).
- Όταν επιθυμούμε να συμπιέσουμε το χρόνο μελέτης ενός φαινομένου.

Η προσομοίωση επίσης, είναι σε θέση να προσφέρει βαθύτερη κατανόηση του συστήματος, να βοηθήσει το μελετητή να εξερευνήσει νέα πιθανά σενάρια εξέλιξης ενός φαινομένου και να διαγνώσει πιθανά προβλήματα, τα οποία υπό άλλες συνθήκες δε θα γινόταν αντιληπτά.

Η προσομοίωση παρουσιάζει και ορισμένα μειονεκτήματα. Τα κυριότερα αφορούν κυρίως στα κάτωθι:

- Η δημιουργία ενός μοντέλου, το οποίο είναι ακριβές, απλό και περιεκτικό είναι συνήθως μια δύσκολη και επίπονη διαδικασία. Πολλές φορές είναι δύσκολος ο συμπερασμός, για το κατά πόσο το παραγόμενο μοντέλο πλησιάζει επαρκώς τη συμπεριφορά του πραγματικού συστήματος.
- Τα αποτελέσματα της προσομοίωσης είναι ορισμένες φορές δύσκολο να μεταφραστούν σε συμπερασμούς για το πραγματικό σύστημα, καθώς είναι δύσκολη η διάκριση μεταξύ αυτών που προέρχονται από τις διεργασίες της προσομοίωσης ή από τυχαιότητα [5].
- Η διαδικασία ανάλυσης μπορεί να είναι χρονοβόρα ή να έχει υψηλό κόστος.
- Η διαδικασία προσομοίωσης είναι δυνατό να χρησιμοποιείται λανθασμένα σε περιπτώσεις, όπου ο συμπερασμός είναι δυνατός με αναλυτικές μεθόδους [5].

1.2.3 Κύριες Έννοιες

Η διαδικασία της προσομοίωσης μπορεί να γίνει με διάφορους τρόπους ανάλογα με τον τύπο του συστήματος. Συνήθη βοηθήματα για την διεξαγωγή προσομοιώσεων αποτελούν οι ηλεκτρονικοί υπολογιστές. Τα συστήματα που προσομοιώνονται σε αυτή την περίπτωση είναι διακριτού χρόνου, η συστήματα συνεχούς χρόνου που έχουν μετατραπεί σε μοντέλα διακριτού χρόνου με συγκεκριμένες παραδοχές.

Υπάρχουν συγκεκριμένες έννοιες που αφορούν στην προσομοίωση. Αυτές περιλαμβάνουν: Το σύστημα, το μοντέλο, της μεταβλητές κατάστασης του συστήματος, τις οντότητες, τις ιδιότητες κ.α. [5].

Το μοντέλο είναι μια απεικόνιση του προς εξέταση συστήματος. Ανάλογα με τις απαιτήσεις της προσομοίωσης αλλά και τις δυνατότητες των διαθέσιμων εργαλείων, αυτό μπορεί να είναι αρκετά λεπτομερές και πολύπλοκο. Συνήθως όμως το μοντέλο θα πρέπει να είναι σε θέση να απαντήσει στα ερωτήματα για τα οποία δημιουργήθηκε αλλά να μην είναι εξαιρετικά πολύπλοκο.

Τα γεγονότα είναι συμβάντα τα οποία αλλάζουν την κατάσταση του συστήματος. Για παράδειγμα, σε μια μοντελοποίηση ενός δικτυακού πρωτοκόλλου, γεγονός είναι η άφιξη ενός πακέτου ή η παράδοσή του στο σταθμό προορισμού.

Μεταβλητές συστήματος Περιλαμβάνουν όλες εκείνες τις πληροφορίες, οι οποίες είναι ικανές να περιγράψουν με την επιθυμητή ακρίβεια τι συμβαίνει τη συγκεκριμένη χρονική στιγμή στο σύστημα. Η επιλογή των μεταβλητών καθορίζεται από τον μελετητή, ανάλογα με τον επιθυμητό σκοπό ή ακρίβεια.

Οντότητες Αναπαριστούν αντικείμενα του συστήματος, που είναι απαραίτητα για την μοντελοποίηση του. Μπορούν να είναι είτε στατικά (π.χ. σταθμοί) ή δυναμικά (πακέτα που παραδίδονται και μετά εξαφανίζονται από το σύστημα). Τα αντικείμενα μπορούν να έχουν συγκεκριμένες *ιδιότητες*, οι οποίες καθορίζονται από το μελετητή.

Πόροι Είναι οντότητες, οι οποίες παρέχουν υπηρεσίες στις δυναμικές οντότητες του συστήματος. Οι δυναμικές οντότητες μπορούν να αιτηθούν από το σύστημα έναν ή περισσότερους πόρους για την εξυπηρέτησή τους. Εφόσον δεν καταφέρουν να εξασφαλίσουν κάποιο πόρο, εισέρχονται σε κάποια ουρά αναμονής, στην οποία παραμένουν μέχρι να εξυπηρετηθούν ή να απορριφθούν από το σύστημα. Περισσότερα για τις ουρές αναμονής σε προβλήματα τηλεπικοινωνιών, αναλύονται στο τμήμα 1.2.4.

Διεργασίες (activities) Είναι χρονικές περίοδοι των οποίων η διάρκεια είναι γνωστή εκ των προτέρων και το τέλος τους μπορεί να προγραμματισθεί [5]. Η διάρκεια αυτή μπορεί να είναι σταθερή, να προέρχεται από μια στατιστική κατανομή, να υπολογίζεται από μαθηματικές σχέσεις κτλ.

Καθυστερήσεις Είναι χρονικές περίοδοι αρχικά αδιευκρίνιστης διάρκειας, οι οποίες προκαλούνται από τις συνθήκες του συστήματος. Για παράδειγμα, ο χρόνος αναμονής ενός πακέτου εντός της ουράς αναμονής μέχρι να εξυπηρετηθεί είναι αρχικά άγνωστος, καθώς εξαρτάται από συνδυασμό γεγονότων, που είναι δυνατό να συμβούν εντός του συστήματος.

1.2.4 Θεωρία Ουρών στις Τηλεπικοινωνίες

Στα τηλεπικοινωνιακά δίκτυα, η θεωρία ουρών αναμονής χρησιμοποιείται για τη μοντελοποίηση ενός μεγάλου εύρους προβλημάτων. Ειδικότερα, χρησιμοποιείται οποτεδήποτε ένας πόρος του δικτύου διαμοιράζεται μεταξύ ανταγωνιζομένων οντοτήτων ή αιτημάτων. Στην περίπτωση που ο ρυθμός αφίξεων των αιτημάτων ξεπερνά, έστω και προσωρινά τη δυνατότητα εξυπηρέτησης του συστήματος, τότε χρησιμοποιούνται ουρές αναμονής, οι οποίες αποθηκεύουν προσωρινά αιτήματα, εώς ότου να εξυπηρετηθούν. Τυπικές περιπτώσεις δικτυακών προβλημάτων, ανά επίπεδο OSI, στις οποίες χρησιμοποιείται η θεωρία ουρών αναμονής είναι όπως παρακάτω:

- **OSI Layer 1:** Φαινόμενα συμφόρησης στην δικτυακή ροή, λόγω ανεπαρκών πόρων στη διαδρομή μεταξύ πηγής και προορισμού.
- **OSI Layer 2:** Περιπτώσεις στις οποίες διαφορετικά πακέτα χρησιμοποιούν στον ίδιο χρόνο κοινό δικτυακό μονοπάτι.
- **OSI Layer 3:** Συγκέντρωση αιτημάτων δρομολόγησης προς έλεγχο στον επεξεργαστή ενός δρομολογητή.

Η θεωρία ουρών αναμονής βρίσκει εφαρμογή περισσότερο σε packet switching δίκτυα από ότι σε circuit switching. Αυτό συμβαίνει γιατί στα circuit switching δίκτυα, όταν δεν υφίστανται οι απαραίτητοι πόροι για την εξυπηρέτηση των αιτημάτων, αυτά απορρίπτονται.

Εδώ η θεωρία ουρών αναμονής είναι περισσότερο χρήσιμη για τη στατιστική - μαθηματική μοντελοποίηση της διαδικασίας αφίξεων των αιτημάτων προς εξυπηρέτηση.

Αντίθετα στα packet switching δίκτυα είναι δυνατή η μοντελοποίηση σύνθετων διαδικασιών στα επίπεδα OSI 1 - 3. Αυτό συμβαίνει γιατί, σε αυτή την περίπτωση οι δυναμικές οντότητες (πακέτα) μπορούν να εισαχθούν σε κάποια ουρά μέχρι να εξυπηρετηθούν. Άλλος ένας λόγος είναι ότι τέτοιας μορφής δίκτυα μπορούν εύκολα να μοντελοποιηθούν ώς συστήματα διακριτού χρόνου.

Στα επόμενα παρουσιάζονται επιγραμματικά στοιχεία που αφορούν στο προς προσομοίωση πρωτόκολλο.

1.3 Δίκτυα Πολλαπλής Πρόσβασης

1.3.1 Εισαγωγή

Τα παραδοσιακά δίκτυα ήταν φτιαγμένα για συνδέσεις σημείο προς σημείο. Υπήρχαν συγκεκριμένοι δίαυλοι - κανάλια οι οποίοι συνέδεαν ανά δυο τους κόμβους μεταξύ τους. Σε αυτού του τύπου την αρχιτεκτονική δεν υπήρχε παρεμβολή μεταξύ των καναλιών επικοινωνίας, ενώ η τοπολογία ήταν λίγο ή πολύ γνωστή κατά τη φάση της σχεδίασης και δεν αναμένονταν να αλλάξει σημαντικά. Αυτού του τύπου οι συνδέσεις παρουσιάζουν δυο σημαντικά μειονεκτήματα:

- Δεν είναι οικονομικά αποδοτικές αφού επιβάλουν τη σύνδεση με αποκλειστικά (dedicated) κανάλια μεταξύ όλων των κόμβων.
- Δεν μπορούν να ανταποκριθούν σε δυναμικές τοπολογίες.

Σε αυτές τις περιπτώσεις χρησιμοποιούνται κανάλια broadcast. Σε αυτή την περίπτωση οι κόμβοι μοιράζονται ένα ή περισσότερα κανάλια επικοινωνίας (συνήθως λίγα σε αριθμό και σίγουρα λιγότερα από ότι το πλήθος των συνδυασμών μεταξύ τους). Σε αυτή την περίπτωση όμως υπάρχουν παρεμβολές, καθώς οι κόμβοι ανταγωνίζονται για την πρόσβαση στα κανάλια επικοινωνίας (συγκρούσεις - collisions. Την επίλυση των συγκρούσεων αναλαμβάνουν τα πρωτόκολλα ελέγχου πρόσβασης μέσου (Medium Access Control, MAC).

Τα πρωτόκολλα αυτά είναι συνήθη σε δίκτυα τύπου LAN - MAN. Συνήθως χρησιμοποιούνται διαφορετικά τέτοια πρωτόκολλα αναλόγως της δικτυακής τοπολογίας, του είδους του φυσικού μέσου και των χαρακτηριστικών της δικτυακής κίνησης. Παρακάτω παρουσιάζονται οι βασικές κατηγορίες πρωτοκόλλων ελέγχου πρόσβασης μέσου, αναλόγως με τον αλγόριθμο εκχώρησης του φυσικού μέσου στους κόμβους:

- **Πρωτόκολλα σταθερής εκχώρησης (Fixed access protocols)** Το πρωτόκολλο εκχωρεί κάθε φορά τη χρήση του φυσικού μέσου σε ένα σταθμό αποκλειστικά για την εκπομπή. Ο αλγόριθμος εκχώρησης μπορεί να αποφασίζει για το διαμοιρασμό του μέσου στο χώρο των συχνοτήτων, του χρόνου ή με άλλο τρόπο. Χαρακτηριστικά παραδείγματα τέτοιων πρωτοκόλλων είναι τα Time Division Multiple Access (TDMA), Frequency Division Multiple Access (FDMA), Code Division Multiple Access (CDMA).
- **Πρωτόκολλα συμφόρησης (Contention-based protocols)** Αυτά τα πρωτόκολλα εκχωρούν το μέσο σε περισσότερους από έναν σταθμούς τυατόχρονα, αλλά προβλέπουν έναν αλγόριθμο επίλυσης των συγκρούσεων. Συνήθως επιβάλλουν στους σταθμούς που απέτυχαν μια τυχαία καθυστέρηση επανεκπομπής του πακέτου που συγκρούστηκε. παραδείγματα τέτοιων πρωτοκόλλων είναι τα Pure Aloha, Slotted Aloha, Carrier Sense Multiple Access (CSMA).

- **Πρωτόκολλα ζήτησης (Demand-assignment protocol)** εκχωρούν το μέσο ανάλογα με τη ζήτηση των σταθμών. Τα πρωτόκολλα αυτά μπορούν να χωριστούν σε τρεις κατηγορίες: Polling Scheme, Token-Based Approach, και Reservation Aloha. Η πρώτη κατηγορία ρωτά με ένα συγκεκριμένο τρόπο τους σταθμούς αν έχουν να εκπέμψουν και εκχωρεί το μέσο σε έναν από αυτούς. Η δεύτερη κατηγορία χρησιμοποιείται κυρίως σε δίκτυα τοπολογίας δακτυλίου ή διαύλου. Ένα συγκεκριμένο σήμα (token) διακινείται με τη σειρά στους κόμβους του δικτύου και όποιος σταθμός το λαμβάνει, έχει δυνατότητα εκπομπής. Η τρίτη κατηγορία χρησιμοποιείται κυρίως στα ασύρματα δίκτυα, και χωρίζει το χρόνο εκπομπής σε δυο περιοδους. Κατά την πρώτη περίοδο οι σταθμοί υποβάλλουν στο σταθμό βάσης αιτήματα εκπομπής. Κατά τη δεύτερη φάση ο σταθμός βάσης εκχωρεί το μέσο σε έναν από τους σταθμούς.

Το κύριο πρόβλημα που έχουν τα MAC πρωτόκολλα είναι ότι δε μπορούν να χρησιμοποιηθούν για όλους τους τύπους των εφαρμογών. Για παράδειγμα τα πρωτόκολλα σταθερής εκχώρησης είναι αποδοτικά σε σταθερή κίνηση δικτύου χωρίς ξαφνικές αυξομειώσεις. Τα πρωτόκολλα συμφόρησης, από την άλλη, λειτουργούν σωστά σε δικτυακή κίνηση με εξάρσεις. Τα πρωτόκολλα της τρίτης κατηγορίας αποδίδουν περισσότερο σε περιπτώσεις που διακινείται μεγάλος όγκος δεδομένων για σχετικά μεγάλο χρονικό διάστημα.

Στη βιβλιογραφία έχουν προταθεί διάφορες μέθοδοι πρόσβασης μέσου, αλλά δεν έχουν προταθεί μέχρι σήμερα μέθοδοι οι οποίες να αποδίδουν ικανοποιητικά σε διαφορετικά μοτίβα δικτυακής κίνησης [9]. Εν προκειμένω, ένα τέτοιο πρωτόκολλο θα πρέπει να ικανοποιεί ταυτόχρονα τα παρακάτω κριτήρια:

- **Προτεραιότητες:** Θα πρέπει να διαχειρίζεται διαφορετικούς τύπους δικτυακής κίνησης έτσι ώστε να ανταποκρίνεται σε πληθώρα απαιτήσεων Quality of Service.
- **Ορθή διαχείριση πρόσβασης μέσου:** Ισότιμος διαμοιρασμός των πόρων του δικτύου στους κόμβους.
- **Χαμηλά επίπεδα καθηστέρησης:** Άμεση ανταπόκριση σε πραγματικό χρόνο σε διακυμάνσεις στη δικτυακή κίνηση.
- **Απόδοση:** Χρήση όλων των πόρων του δικτύου με αποδοτικό τρόπο και υπό υψηλό φόρτο.
- **Σταθερότητα:** Ορθή διαχείριση των αιτημάτων εκπομπής, ώστε ο μέσος όρος των πακέτων που παράγονται από τους σταθμούς να είναι περίπου ίσος με το μέσο όρο αφίξεων στους προορισμούς.

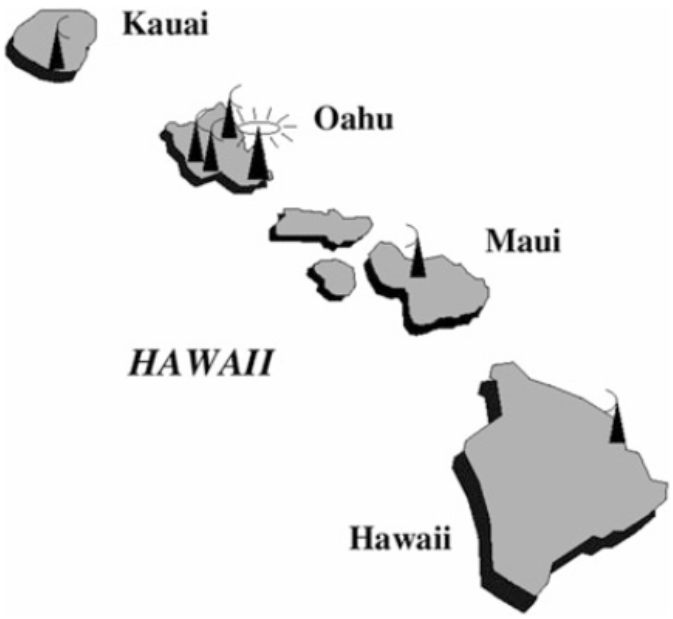
Τυπικές παράμετροι που εξετάζονται κατά την αξιολόγηση των πρωτοκόλλων MAC είναι:

- **Η απόδοση (Throughput, MAC level)** - Ποσοστό του χρόνου στο οποίο το διαμοιραζόμενο μέσο χρησιμοποιείται για την μεταφορά ωφέλιμου δικτυακού φορτίου. Καλό είναι να σημειωθεί ότι εδω η έννοια του throughput είναι διαφορετική από την αντίστοιχη του επιπέδου μεταφοράς, που αφορά στη δικτυακή κίνηση (bit-rate).
- **Η μέση καθυστέρηση των πακέτων (Mean Delay)** - Ο μέσος χρόνος που περνά από την γέννηση ενός πακέτου στο δίκτυο μέχρι την επιτυχή παράδοσή του στον προορισμό.

Οι μετρικές αυτές χρησιμοποιούνται για την εξέταση του πρωτοκόλλου Slotted Aloha κατά τη φάση της προσομοίωσης στο Κεφάλαιο 3.

1.3.2 Πρωτόκολλο Pure Aloha

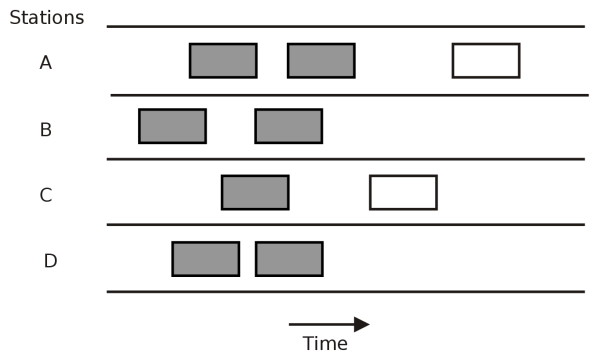
Το πρωτόκολλο Pure Aloha ανήκει στην ομάδα Contention-based protocols, όπως αναλύθηκε παραπάνω. Χρησιμοποιήθηκε σε ένα πειραματικό δίκτυο, το οποίο λειτουργούσε στην UHF μπάντα για τη διασύνδεση υπολογιστών του Πανεπιστημίου της Χαβάης, οι οποίοι βρίσκονταν σε διαφορετικά νησιά του συμπλέγματος [19].



Σχήμα 1.1: Το αρχικό δίκτυο aloha (αναπαραγωγή από [9])

Αναπτύχθηκε το 1970 από τον Norman Manuel Abramson. Αποτελούνταν από ένα κεντρικό ελεγκτή, ο οποίος βρισκόταν στη Χονολουλού και τους περιφερειακούς κόμβους. Κατά το πρωτόκολλο αυτό, ο κεντρικός ελεγκτής επικοινωνεί με τους υπόλοιπους με πακέτα broadcast. Ανάλογα, οι περιφερειακοί σταθμοί επικοινωνούν με τον κεντρικό στέλνοντας πακέτα οποτεδήποτε έχουν να εκπέμψουν. Εάν περισσότερα από ένα πακέτα παραληφθούν από τον κεντρικό σταθμό, τότε αυτά καταστρέφονται λόγω συγκρούσεων. Υπάρχει βέβαια και η περίπτωση, ένα πακέτο να παραληφθεί με μεγαλύτερη ισχύ από ότι άλλα πακέτα και να είναι δυνατή η αποκωδικοποίησή του. Σε αυτή την περίπτωση γίνεται αποδεκτό και απορρίπτονται τα υπόλοιπα. Κάθε περιφερειακός σταθμός αντιλαμβάνεται ότι το πακέτο του έχει παραληφθεί εάν λαμβάνει επιβεβαίωση από τον κεντρικό σταθμό με ένα broadcast πακέτο σε συγκεκριμένη συχνότητα - κανάλι, διαφορετική από αυτή των δεδομένων. Εάν δε λάβει κάποια επιβεβαίωση εντός συγκεκριμένου χρονικού διαστηματος, τότε το πακέτο έχει απορριφθεί. Εάν το πακέτο απορριφθεί για κάποιο λόγο, τότε ο σταθμός το επανεκπέμπει μετά από μια τυχαία καθυστέρηση (backoff algorithm), για να αποφύγει συνεχόμενες αποτυχίες - συγκρούσεις. Το πακέτο επανεκπέμπεται συνεχώς μέχρι να παραληφθεί επιτυχώς. Σε κάποιες πρακτικές εφαρμογές όμως το πακέτο απορρίπτεται μετά από συγκεκριμένο αριθμό αποτυχιών, για λόγους που αφορούν στη σταθερότητα του πρωτοκόλλου. Στο σχήμα 1.2 φαίνεται παραστατικά το πρωτόκολλο. Τα ορθογώνια παριστάνουν πακέτα δεδομένων. Τα σκιασμένα ορθογώνια δηλώνουν ότι τα πακέτα έχουν συγκρουστεί.

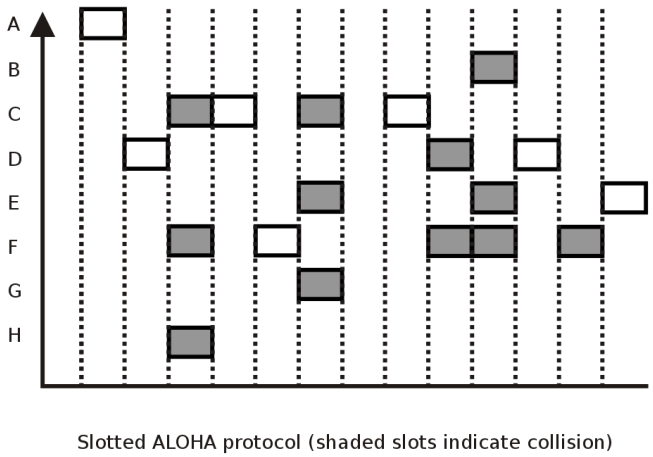
Τα δύο μεγάλα μειονεκτήματα του πρωτοκόλλου Pure Aloha είναι ότι υπάρχει σχετικά μεγάλη καθυστέρηση στη μετάδοση των πακέτων ενώ πολλά δεδομένα χάνονται λόγω συγκρούσεων.



Σχήμα 1.2: Χρονική αλληλοκάλυψη των πακέτων (pure aloha)

1.3.3 Πρωτόκολλο Slotted Aloha

Για να εξαλειφθούν τα μειονεκτήματα του πρωτοκόλλου Pure Aloha, το 1972 προτάθηκε ένα νέο βελτιωμένο πρωτόκολλο. Το πρωτόκολλο αυτό επιτύγχανε το διπλασιασμό της ωφέλιμης δικτυακής κίνησης με την παραδοχή ότι ο χρόνος χωρίζεται σε διακριτά time-slots με τη διάρκεια του καθενός να αντιστοιχεί στην εκπομπή ενός πακέτου. Οι εκπομπές των πακέτων από τους περιφερειακούς σταθμούς γίνονται με τέτοιο τρόπο ώστε τα πακέτα να παραλαμβάνονται από τον κεντρικό σταθμό σε προκαθορισμένες χρονικές στιγμές. Συγκεκριμένα, κάθε σταθμός μπορεί να εκπέμψει μόνο στην αρχή κάθε timeslot. Το νέο πρωτόκολλο ονομάστηκε έτσι από το τρόπο διαχωρισμού του χρόνου. Για να επιτύχει το συγχρονισμό εκπομπών μεταξύ των περιφερειακών σταθμών, ο κεντρικός σταθμός εκπέμπει σε τακτά χρονικά διαστήματα παλμούς συγχρονισμού. Εφόσον οι σταθμοί εκπέμπουν μόνο σε καθορισμένες χρονικές στιγμές, οι συγκρούσεις μπορούν να συμβούν μόνο όταν δύο ή περισσότεροι σταθμοί εκπέμπουν στην ίδια χρονοσχισμή (timeslot), όπως φαίνεται και στο σχήμα 1.3.



Σχήμα 1.3: Χρονική αλληλοκάλυψη των πακέτων (slotted aloha)

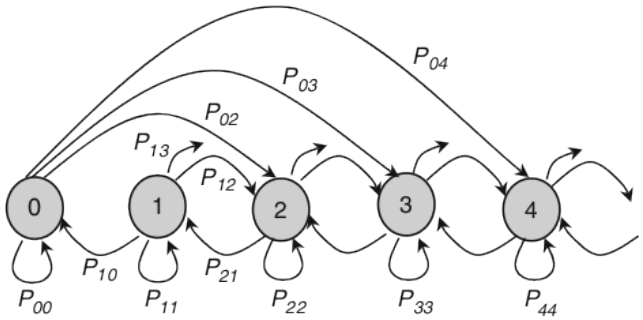
Οι σταθμοί που αποτυγχάνουν, αναπρογραμματίζουν την εκπομπή του πακέτου με μία τυχαία χρονική καθυστέρηση, σύμφωνα με έναν αλγόριθμο υπαναχώρησης (backoff algorithm).

1.4 Μαθηματική Ανάλυση του Πρωτοκόλλου

Γενικότερα, θα μπορούσαμε να διακρίνουμε τις κύριες μεθόδους ανάλυσης των πρωτοκόλλων τυχαίας πρόσβασης μέσου (random access protocols), εν προκειμένω και των τύπου aloha, όπως παρακάτω [9]:

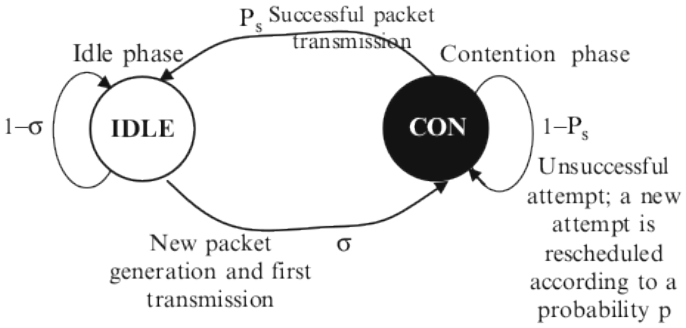
- **S – G ανάλυση:** υπολογίζεται μαθηματικά, χρησιμοποιώντας στοιχεία πιθανοτήτων, η σχέση μεταξύ των πακέτων που παράγονται συνολικά στο σύστημα (εφεξής G) και των πακέτων που παραδίδονται επιτυχώς στους προορισμούς τους. Η ανάλυση γίνεται υπό τις παραδοχές αφίξεων Poisson και απείρου πλήθους στοιχειώδεις σταθμούς εκπομπής. Αυτή είναι μια απλοποιημένη προσέγγιση και για πεπερασμένο πλήθος σταθμών, (με παραδοχές αφίξεων διωνυμικής κατανομής σε αυτή την περίπτωση) χωρίς άρση της γενικότητας. Σε αυτή την ανάλυση ωστόσο δεν λαμβάνονται υπόψη επιδράσεις από την ύπαρξη buffer αποθήκευσης πακέτων. Αυτή η μέθοδος είναι κυρίως κατάλληλη για την ανάλυση του mean access delay και του throughput S.

- **Ανάλυση με Μαρκοβιανές Αλυσσίδες**, σε αυτού του τύπου την ανάλυση αξιολογείται η συμπεριφορά του συστήματος ή ένος σταθμού, με τη χρήση Markov chains. Η μέθοδος αυτή εφαρμόζεται σε πεπερασμένο πλήθος σταθμών. Αυτή η μέθοδος συνήθως είναι η πιο ακριβής αλλά και η πιο περίπλοκη. Συνήθως εξετάζεται η σταθερότητα του πρωτοκόλλου μέσω της συμπεριφοράς του throughput S. Σε μερικές περιπτώσεις (για παράδειγμα στη μελέτη MAC πρωτοκόλλων για ασύρματα δίκτυα) η ανάλυση γίνεται με την προϋπόθεση ότι όλοι οι σταθμοί είναι backlogged (saturated conditions). Αυτού του τύπου η μελέτη χρησιμοποιείται για την αξιολόγηση του mean access delay και του throughput S [13]. Από την άλλη μοντέλα non-saturated χρησιμοποιούνται για την αξιολόγηση του mean packet delay και της σταθερότητας του πρωτοκόλλου.



Σχήμα 1.4: Μοντέλο ανάλυσης με μαρκοβιανές αλυσσίδες για το πρωτόκολλο slotted aloha (αναπαραγωγή από [9])

- **Equilibrium Point Ανάλυση** [9], κατά την οποία η συμπεριφορά κάθε σταθμού μοντελοποιείται με τη χρήση διαγραμμάτων κατάστασης. Αυτή η μέθοδος εφαρμόζεται σε πεπερασμένο αριθμό σταθμών M. Οι μεταβάσεις των καταστάσεων χαρακτηρίζονται από συγκεκριμένες πιθανότητες, ενώ συνθήκες ισορροπίας (Equilibrium conditions καθορίζονται για κάθε κατάσταση, λαμβάνοντας υπόψη ότι κάθε μία περιλαμβάνει ένα μέσο αριθμό σταθμών [18].



Σχήμα 1.5: Μοντέλο ανάλυσης με διαγράμματα κατάστασης για το πρωτόκολλο slotted aloha (αναπαραγωγή από [9])

1.5 Προηγούμενη Εργασία

Κατά καιρούς έχουν προταθεί στη βιβλιογραφία μέθοδοι ανάλυσης με μαθηματικά εργαλεία, των πρωτοκόλλων πρόσβασης μέσου, τύπου Aloha. Επειδή το πρωτόκολλο slotted aloha εμφανίζει αστάθειες, καθώς το πλήθος των σταθμών αυξάνει σημαντικά [8, 15], η αρχική έρευνα είχε στραφεί στην μελέτη της σταθερότητας του πρωτοκόλλου. Για παράδειγμα στο [17] παρουσιάζεται ένας Bayesian αλγόριθμος, που χρησιμοποιεί ανάδραση για να εκτιμήσει το πλήθος των backlogged σταθμών. Η σύγχρονη προσέγγιση χρησιμοποιεί διαδικασίες admission control για να περιορίσει το πλήθος των ενεργών σταθμών [8]. Άλλες μέθοδοι ανάλυσης του πρωτοκόλλου περιλαμβάνουν ανάλυση με μαρκοβιανές αλυσσίδες [16, 3] και ανάλυση με μεθόδους της θεωρίας παιγνίων [8, 10].

2 Μοντελοποίηση του Πρωτοκόλλου

2.1 Εισαγωγή

Κατά τη φάση της μοντελοποίησης καθορίζονται οι βασικές παραδοχές στις οποίες θα στηριχθεί η προσομοίωση. Επίσης καθορίζεται ο τρόπος παραγωγής της δικτυακής κίνησης και η στατιστική κατανομή που ακολουθούν τα παραγόμενα πακέτα. Αργότερα κατά τη φάση της προσομοίωσης χρησιμοποιούνται αυτές οι αποφάσεις για την υλοποίηση της γεννήτριας των τυχαίων αριθμών και τον καθορισμό της λεπτομέρειας προσομοίωσης (σε μακροσκοπικό ή μικροσκοπικό επίπεδο, π.χ. προσομοίωση σε επίπεδο κόμβων).

2.2 Παραγωγή Τυχαίων Αριθμών

Οι τυχαίοι αριθμοί είναι ίσως το σημαντικότερο μέρος της προσομοίωσης σε Η/Υ. Χρησιμοποιούνται για την δημιουργία των γεγονότων στο σύστημα, για διακλαδώσεις στον κώδικα προσομοίωσης, που εξαρτώνται από την τυχαιότητα κ.α. Τυπικά η τυχαιότητα παράγεται από μια γεννήτρια ψευδοτυχαίων αριθμών και θεωρείται ότι ακολουθεί ομοιόμορφη κατανομή $U(0,1)$. Στη συνέχεια, οι παραγόμενοι τυχαίοι αριθμοί υφίστανται μετασχηματισμό, ώστε να ακολουθούν μία ή περισσότερες επιθυμητές στατιστικές κατανομές, οι οποίες καθορίζονται κατά τη φάση της μοντελοποίησης. Γενικότερα η παραγωγή των τυχαίων αριθμών κατά την προσομοίωση στηρίζεται στην παραδοχή ότι, η γεννήτρια ψευδοτυχαίων αριθμών παράγει ομοιόμορφη κατανομή $U(0,1)$. Ο ισχυρισμός αυτός δεν ισχύει γενικότερα γιατί η γεννήτρια ψευδοτυχαίων είναι ένα ντετερμινιστικό πρόγραμμα σε Η/Υ. Μπορούμε όμως να αποδεχθούμε τους αριθμούς, που παράγονται από την υπόψη διαδικασία, εφόσον ο κύκλος περιόδου της γεννήτριας είναι αρκετά μεγάλος. Εκτεταμένη ανάλυση και έλεγχοι τυχαιότητας για διάφορες υλοποιήσεις γεννητριών τυχαίων αριθμών παρουσιάζονται στο [11], ενώ στο τμήμα 3.2 γίνεται εκτενής αναφορά στην γεννήτρια ψευδοτυχαίων αριθμών που επιλέχθηκε για την προσομοίωση.

2.2.1 Στοιχεία Στατιστικών Κατανομών

2.2.1.1 Μέθοδοι Παραγωγής Στατιστικών Κατανομών

Για το μετασχηματισμό των αριθμών, που παράγονται από τη γεννήτρια ψευδοτυχαίων αριθμών, σε αριθμούς που ακολουθούν συγκεκριμένη κατανομή ακολουθούνται διάφοροι αλγόριθμοι, οι οποίοι παρουσιάζονται συνοπτικά παρακάτω.

Υπολογισμός μέσω της Αντίστροφης Συναρτησης Όταν η τυχαία μεταβλητή X είναι συνεχής, η κατανομή της είναι δυνατό να αναπαρασταθεί από την συνάρτηση πυκνότητας πιθανότητας $f(x)$. η πιθανότητα το X να βρίσκεται μεταξύ δυο δεδομένων τιμών α, β με $\beta > \alpha$ δίδεται από τη σχέση:

$$\Pr(\alpha \leq X \leq \beta) = \int_{\alpha}^{\beta} f(x)dx$$

Από τον παραπάνω ορισμό προκύπτει ότι η η πιθανότητα αυτή περιγράφεται από την συνάρτηση αθροιστικής κατανομής:

$$\Pr(\alpha \leq X \leq \beta) = F(x) = \int_{-\infty}^{X} f(z)dz$$

Λόγω του ότι η $f > 0, \forall x$, η F είναι συνεχής και γνησίως αύξουσα. Άρα είναι ένα προς ένα και ορίζεται η αντίστροφή της. Εάν είναι εύκολος ο υπολογισμός της αντίστροφης, τότε για την παραγωγή των τυχαίων αριθμών, που ακολουθούν συγκεκριμένη κατανομή, παράγονται ψευδοτυχαίοι αριθμοί $u \sim U(0,1)$ και χρησιμοποιείται η σχέση:

$$x = F^{-1}(u)$$

Η παραπάνω μέθοδος είναι δυνατό να επεκταθεί ώστε να περιλαμβάνει και περιπτώσεις στις οποίες η κατανομή είναι διακριτή. Έστω ότι η τυχαία μεταβλητή X παίρνει μόνο διακριτές τιμές $x_1, x_2, \cdots, x_n$ με αντίστοιχες πιθανότητες

$$p_i = \Pr(X \leq x) = \sum_{i:x_i \leq x} p_i$$

Μπορεί να αποδειχθεί [5] σε αυτή την περίπτωση ότι η αντίστροφη συνάρτηση ορίζεται από τη σχέση:

$$F^{-1}(u) = min\{x|u \leq F(x)\}$$

Ο αλγόριθμος, ο οποίος υλοποιεί την διαδικασία υπολογισμού είναι όπως παρακάτω:

Algorithm 1 Inverse Tranform Method

1: $U \leftarrow R(0,1)$
2: $i \leftarrow 1$
3: **while** $(F(x_i) < U)$ **do**
4: $\quad i \leftarrow i + 1$
5: $X \leftarrow x_i$

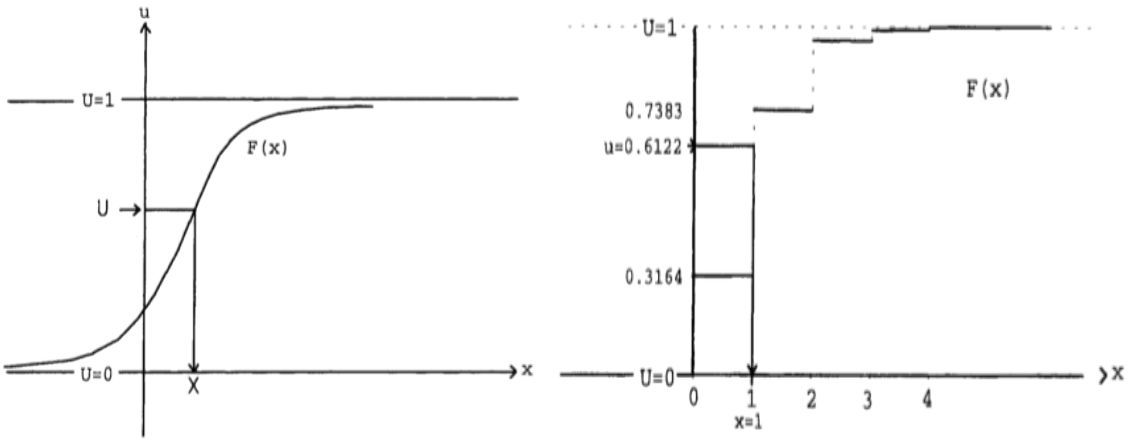
Για το λόγο ότι η συγκεκριμένη μέθοδος απαιτεί γραμμική αναζήτηση, είναι μη αποδοτική όταν το n είναι μεγάλος αριθμός.

Στο σχήμα 2.1 περιγράφεται η μέθοδος για τη συνεχή και τη διακριτή περίπτωση.

Μέθοδος Αποδοχής - Απόρριψης Η μέθοδος αυτή χρησιμοποιείται όταν είναι δύσκολος ο υπολογισμός της αντίστροφης αθροιστικής συνάρτησης πιθανότητας. Για την εφαρμογή της θα πρέπει να ισχύουν τα παρακάτω:

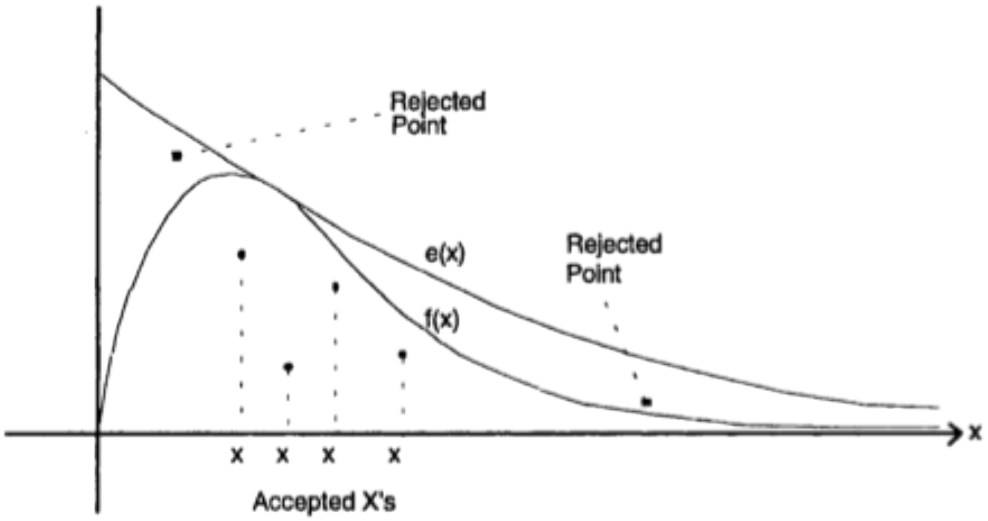
- Η συνάρτηση πυκνότητας πιθανότητας είναι άνω φραγμένη από μια άλλη συνάρτηση πυκνότητας πιθανότητας $g(x)$ $\forall x$.
- Είναι δυνατή η παραγωγή τυχαίων αριθμών που ακολουθούν ομοιόμορφη κατανομή σε όλο το γεωμετρικό χώρο μεταξύ της καμπύλης $g(x)$ και του άξονα των x. Έστω ότι σημειώνουμε τα παραγόμενα σημεία με (X, Y).
- Εάν η γραφική παράσταση της $f(x)$ σχεδιαστεί στο ίδιο διάγραμμα με την $g(x)$ τότε κάποια σημεία θα βρεθούν πάνω από την καμπύλη της $f(x)$ ενώ κάποια σημεία κάτω της, δηλαδή θα είναι $Y > f(x)$ ή $Y \leq f(x)$. αποδεκτά γίνονται μόνο τα σημεία που βρίσκονται κάτω από την $f(x)$.

Η μέθοδος παριστάνεται διαγραμματικά στο σχήμα 2.2.



(α′) Συνεχής Περίπτωση (β′) Διακριτή Περίπτωση

Σχήμα 2.1: Μέθοδος Αντίστροφης Συνάρτησης



Σχήμα 2.2: Μέθοδος Αποδοχής - Απόρριψης

Συνθετική Μέθοδος Ορισμένες φορές είναι επιθυμητή η παραγωγή τυχαίων αριθμών σε μη γνωστή κατανομή, η οποία όμως μπορεί να αναλυθεί σε ένα αλγεβρικό άθροισμα διαφορετικών γνωστών κατανομών. Σε αυτές τις περιπτώσεις, εάν είναι δυνατή η παραγωγή τυχαίων αριθμών από τις γνωστές κατανομές, τότε χρησιμοποιείται ο παρακάτω αλγόριθμος για την παραγωγή της σύνθετης κατανομής [5]:

Algorithm 2 Composition Method

1: $F_j \leftarrow \sum_{i=1}^{j} p_i$ for $j = 1, 2, \cdots, r$
2: $U \leftarrow R(0, 1)$
3: $i \leftarrow 1$
4: **while** $F_i < U$ **do**
5: $\quad i \leftarrow i + 1$
6: Return $X = X_i$ ▷ A variate drawn from the density f_i

Μετασχηματισμοί Κατανομών Ορισμένες φορές είναι δυνατό κάποιες κατανομές να προσεγγίζονται ικανοποιητικά από άλλες γνωστές κατανομές με πιο απλή μαθηματική

διατύπωση ή με τρόπο ώστε να είναι δυνατός ο υπολογισμός της αντίστροφης συνάρτησης πιθανότητας. Τέτοιες περιπτώσεις αναφέρονται στο τμήμα 2.2.1.2.

2.2.1.2 Στατιστικές Κατανομές στην Προσομοίωση

Οι κύριες στατιστικές κατανομές που χρησιμοποιούνται κατά την προσομοίωση δικτυακής κίνησης είναι οι εξής:

Διωνυμική Κατανομή $B(n,p)$ Οι εφαρμογές της περιλαμβάνουν κατά κύριο λόγο:

- Εκτίμηση πιθανοτήτων σε δοκιμές επιτυχίας - αποτυχίας (Bernoulli trials).
- Εκτίμηση πιθανοτήτων σε παίγνια τύχης.
- Δειγματοληψία ιδιοτήτων [7].

Όταν μια δοκιμή Bernoulli επαναλαμβάνεται για πεπερασμένο πλήθος n φορών μας ενδιαφέρει τις περισσότερες φορές ο αριθμός των επιτυχιών - αποτυχιών. Σε αυτή την περίπτωση η τυχαία μεταβλητή $B(n,p)$ εκφράζει αυτόν ακριβώς τον αριθμό των επιτυχιών και ορίζεται ως ο αριθμός των επιτυχιών σε n ανεξάρτητες δοκιμές Bernoulli, όπου η πιθανότητα επιτυχίας για κάθε δοκιμή είναι p, $0 < p < 1$ και η πιθανότητα αποτυχίας $q = 1 - p$. Σε αυτή την περίπτωση λέμε ότι ο αριθμός ακολουθεί την διωνυμική κατανομή $X \sim B(n,p)$.

Η πιθανότητα να έχουμε k επιτυχίες σε n ανεξάρτητα πειράματα με πιθανότητα επιτυχίας p κάθε φορά είναι (συνάρτηση πυκνότητας πιθανότητας):

$$f(k;n,p) = \Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Η αθροιστική συνάρτηση πιθανότητας δίνεται από τη σχέση:

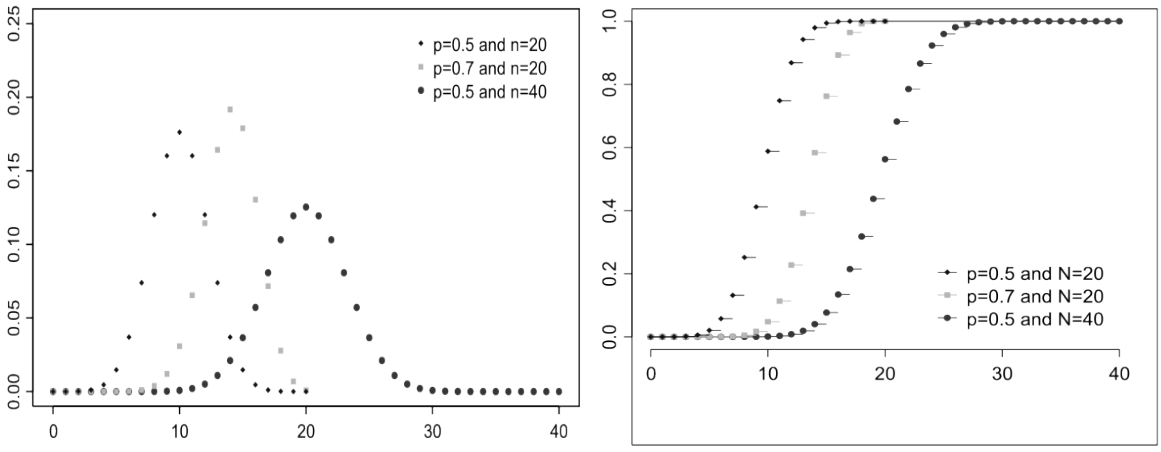
$$F(k;n,p) = \Pr(X \leq k) = \sum_{i=0}^{\lfloor k \rfloor} \binom{n}{i} p^i (1-p)^{n-i}$$

Η μέση τιμή της κατανομής είναι np και η διακύμανση $np(1-p)$. Όταν ισχύει $np(1-p) > 5$ και $0.1 \geq p \geq 0.9$ ή $min(np, n(1-p)) > 10$ η κατανομή συγκλίνει σύμφωνα με το Θεώρημα De Moivre - Laplace στην κανονική κατανομή με μέση τιμή np και τυπική απόκλιση $\sqrt{np(1-p)}$. Η σύγκλιση ισχύει για κάθε p εφόσον $np(1-p) > 25$ [7].

$$\mathcal{N}(np,\, np(1-p))$$

Κατά παρόμοιο τρόπο όταν $p < 0.1$ και $np < 10$ η κατανομή συγκλίνει σε κατανομή Poisson με μέση τιμή $\lambda = np$ [7].

$$\begin{aligned}
\lim_{n\to\infty} P(X=k) &= \lim_{n\to\infty} \frac{n!}{k!\,(n-k)!} \left(\frac{\lambda}{n}\right)^k \left(1-\frac{\lambda}{n}\right)^{n-k} \\
&= \lim_{n\to\infty} \left(\frac{\lambda^k}{k!}\right) \left(\frac{n(n-1)(n-2)\cdots(n-k+1)}{n^k}\right) \left(1-\frac{\lambda}{n}\right)^n \left(1-\frac{\lambda}{n}\right)^{-k} \\
&= \frac{\lambda^k}{k!} \cdot \lim_{n\to\infty} \underbrace{\left(\frac{n}{n} \cdot \frac{n-1}{n} \cdot \frac{n-2}{n} \cdots \frac{n-k+1}{n}\right)}_{\to 1} \underbrace{\left(1-\frac{\lambda}{n}\right)^n}_{\to e^{-\lambda}} \underbrace{\left(1-\frac{\lambda}{n}\right)^{-k}}_{\to 1} \\
&= \frac{\lambda^k \mathrm{e}^{-\lambda}}{k!}
\end{aligned}$$



(α′) Συνάρτηση πυκνότητας πιθανότητας (β′) Συνάρτηση αθροιστικής πιθανότητας

Σχήμα 2.3: Διωνυμική Κατανομή

Στο σχήμα 2.3 εμφανίζονται οι συναρτήσεις πυκνότητας πιθανότητας και αθροιστικής πιθανότητας της κατανομής.

Για την παραγωγή τυχαίων αριθμών που ακολουθούν τη διωνυμική κατανομή, ακολουθούνται οι παρακάτω μέθοδοι [7]:

- Μέθοδος Αποδοχής - Απόρριψης. Παράγονται n αριθμοί $U(0,1)$ από τη γεννήτρια ψευδοτυχαίων. Ο αριθμός ο οποίος είναι μικρότερος από p ακολουθεί την $B(n,p)$ κατανομή.

- Μέθοδος της Γεωμετρικής Κατανομής. Εάν το p είναι αρκετά μικρό, μια γρηγορότερη μέθοδος είναι η άθροιση x αριθμών που ακολουθούν γεωμετρική κατανομή εώς ότου το άθροισμα να ξεπεράσει το $n - x$. Το πλήθος των αθροιζόμενων αριθμών είναι ένας αριθμός διωνυμικής κατανομής.

Κατανομή Poisson Η κατανομή Poisson χρησιμοποιείται σε περιπτώσεις γεγονότων που είναι σχετικά μικρής συχνότητας αλλά έχουν ανοιχτό ορίζοντα τέλους. Ένα χαρακτηριστικό παράδειγμα είναι τα σφάλματα σε μια παρτίδα παραγωγής. Χρησιμοποιείται επίσης για την αναπαράσταση του αριθμού των αφίξεων ανά μονάδα του χρόνου, σε ένα κέντρο εξυπηρέτησης. Ο αριθμός αυτός ακολουθεί κατανομή Poisson, εφόσον ο μέσος όρος του ρυθμού των αφίξεων δεν εμφανίζει συμαντικές διακυμάνσεις κατά την πάροδο του χρόνου. Στην περίπτωση που τα χρονικά διαστήματα μεταξύ των αφίξεων ακολουθούν εκθετική κατανομή, το πλήθος των αφίξεων ακολουθεί κατανομή Poisson [7]. Η κατανομή Poisson μπορεί να χρησιμοποιηθεί και όταν οι ρυθμοί αφίξεων εμφανίζουν εποχικότητα. Σε αυτή την περίπτωση μπορούμε να θεωρήσουμε ότι ισχύει η κατανομή κατά τμήματα, όπου υπάρχει σχετική ομοιογένεια. Η μέση τιμή και η διακύμανση είναι ίσες και είναι δυνατό να εκτιμηθούν από τα χαρακτηριστικά των αφίξεων.

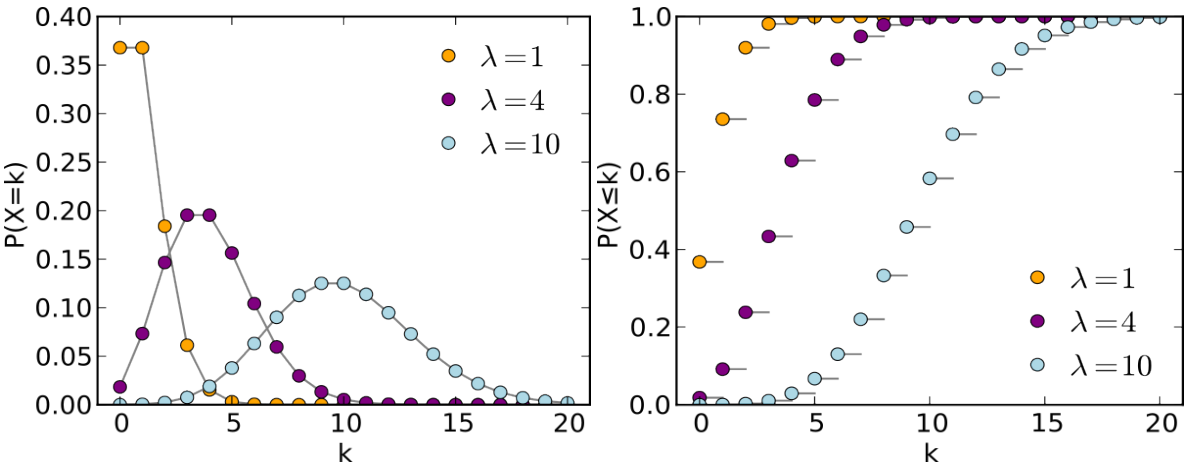
Ένα γεγονός μπορεί να συμβεί κ φορές μέσα σε ένα συγκεκριμένο χρονικό διάστημα. Το μέσο πλήθος των γεγονότων που συμβαίνουν στη μονάδα του χρόνου παριστάνονται από την παράμετρο λ. η συνάρτηση πυκνότητας πιθανότητας δίνεται από τη σχέση:

$$f(k;\lambda) = \Pr(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Η αθροιστική συνάρτηση πιθανότητας δίνεται από τη σχέση:

$$F(x;\alpha,c) = \Pr(X \leq k) = e^{-\lambda} \sum_{i=0}^{\lfloor k \rfloor} \frac{\lambda^i}{i!}$$

Στο σχήμα 2.4 εμφανίζονται οι συναρτήσεις πυκνότητας πιθανότητας και αθροιστικής πιθανότητας της κατανομής.



(α′) Συνάρτηση πυκνότητας πιθανότητας (β′) Συνάρτηση αθροιστικής πιθανότητας

Σχήμα 2.4: Κατανομή Poisson

Για την παραγωγή τυχαίων αριθμών που ακολουθούν την κατανομή Poisson χρησιμοποιείται ο εξής αλγόριθμος. Αρχικά υπολογίζεται η συνάρτηση κατανομής $F(x)$ για $x = 0, 1, 2, \cdots, N$, όπου N είναι ένας αυθαίρετα μεγάλος αριθμός. Στη συνέχεια επιλέγονται τυχαίοι αριθμοί που ακολουθούν την ομοιόμορφη κατανομή $R \sim U(0,1)$. Εάν ισχύει η σχέση $F(x) \leq R < F(x+1)$, τότε ο αριθμός x ακολουθεί την κατανομή Poisson.

Κατανομή Pareto Η κατανομή Pareto περιγράφεται συχνά ως η βάση του κανόνα 80/20 [7]. Για παράδειγμα 80% των πελατών που αφορούν σε ένα κατασκευαστή αυτοκινήτων, παραπονιέται για προβλήματα που προκύπτουν από το 20% των εξαρτημάτων. Αλλες εφαρμογές περιλαμβάνουν την κατανομή του εισοδήματος και την ταξινόμηση του αποθέματος σε μια αποθήκη με βάση τη συχνότητα κίνησης των αγαθών.

Εάν θεωρήσουμε παραμέτρους α, c ώστε να ισχύουν οι σχέσεις $\alpha \leq x < \infty$, $\alpha > 0$ και $c > 0$, η συνάρτηση πυκνότητας πιθανότητας δίνεται από τη σχέση:

$$f(x;\alpha,c) = \Pr(X = x) = \frac{c\alpha^c}{x^{c+1}}$$

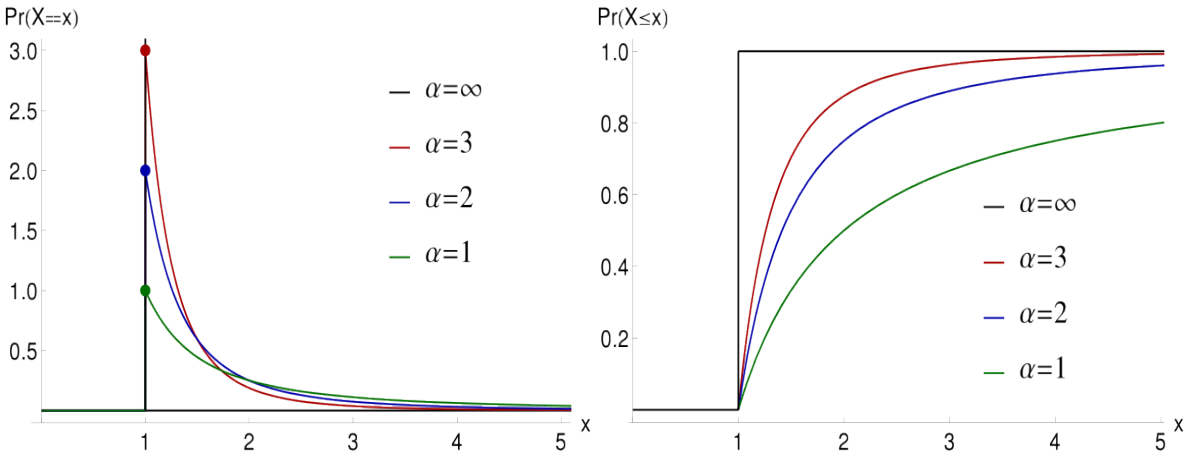
Η αθροιστική συνάρτηση πιθανότητας δίνεται από τη σχέση:

$$F(x;\alpha,c) = \Pr(X \leq x) = 1 - \left(\frac{\alpha}{x}\right)^c$$

Η μέση τιμή της κατανομής είναι $\frac{c\alpha}{c-1}$, $c > 1$ και η διακύμανση $\frac{c\alpha^2}{(c-1)^2(c-2)}$, $c > 2$. Υπάρχουν τρία είδη κατανομών Pareto. Τα σταθερότερα είδη είναι για τιμές $0 < c < 2$.

Στο σχήμα 2.5 εμφανίζονται οι συναρτήσεις πυκνότητας πιθανότητας και αθροιστικής πιθανότητας της κατανομής.

Για την παραγωγή τυχαίων αριθμών που ακολουθούν την κατανομή Pareto, μπορεί να χρησιμοποιηθεί η προσέγγιση $\alpha(1-R)^{-1/c}$ [7], όπου $R \sim U(0,1)$.



(α′) Συνάρτηση πυκνότητας πιθανότητας (β′) Συνάρτηση αθροιστικής πιθανότητας

Σχήμα 2.5: Κατανομή Pareto

2.2.2 Εξομοίωση Γεγονότων Δικτυακής Κίνησης

Η ανάλυση της δικτύακής κίνησης παρέχει αρκετές πληροφορίες για το μέσο φορτίο, τις απαιτήσεις εύρους ζώνης για συγκεκριμένες εφαρμογές και διάφορες άλλες λεπτομέρειες. Τα μοντέλα κίνησης επιτρέπουν στους σχεδαστές δικτύων να εξάγουν συμπεράσματα σχετικά με το δίκτυο, βασιζόμενοι σε παλαιότερες παρατηρήσεις και να κάνουν προβλέψεις για την μελλοντική απόδοση του δικτύου. Τα μοντέλα δικτυακής κίνησης είναι δυνατό να χρησιμοποιηθούν με τους παρακάτω τρόπους:

- Ως μέρος ενός αναλυτικού μοντέλου του δικτύου
- Για τον καθορισμό μιας προσομοίωσης διακριτών γεγονότων.

Σε αντιστοιχία με τις κατανομές του τμήματος 2.2.1.2 μπορούμε να διακρίνουμε τους τρόπους παραγωγής δικτυακής κίνησης (μοντέλα) κατά την προσομοίωση ως εξής:

Μοντέλο κίνησης Poisson . Είναι από τα παλαιότερα και πιο διαδεδομένα μοντέλα δικτυακής κίνησης. Χρησιμοποιήθηκε αρχικά για την μοντελοποίηση της κίνησης σε τηλεφωνικά δίκτυα. Το δικτυακό μοντέλο κίνησης Poisson είναι κατάλληλο όταν η αφίξεις των πακέτων προέρχονται από ένα μεγάλο πλήθος ανεξάρτητων πηγών γνωστών και ως πηγές Poisson. Η κατανομή Poisson είναι πολύ διαδεδομένη σε εφαρμογές δικτυακής κίνησης που παράγονται από ανεξάρτητες πηγές. Ο λόγος προκύπτει από το Θεώρημα του Palm [6]. Το θεώρημα καθορίζει ότι υπό ορισμένες συνθήκες όταν πολυπλέκονται πολλές το πλήθος, ανεξάρτητες μεταξύ τους διεργασίες, η προκύπτουσα διεργασία πλησιάζει την κατανομή Poisson. Δεν είναι βέβαια απαραίτητο κάθε δικτυακή κίνηση να προσομοιάζει διεργασία Poisson.

Οι δυο κύριες παραδοχές που λαμβάνονται υπόψη, ώστε να ισχύει το μοντέλο κίνησης Poisson είναι ως εξής:

- Το πλήθος των πηγών είναι άπειρο.
- Το μοτίβο αφίξεων είναι τυχαίο.

Θα πρέπει να σημειωθεί ότι, διεργασίες Poisson με μέσες τιμές μεγαλύτερες από 30 πλησιάζουν αρκετά την κανονική κατανομή, η οποία μπορεί να χρησιμοποιηθεί σε αυτές τις περιπτώσεις, αναλόγως της επιθυμητής ακρίβειας [6].

Διωνυμικό Μοντέλο κίνησης Η Διωνυμική Κατανομή μπορεί να χρησιμοποιηθεί αντί της Κατανομής Poisson σε περιπτώσεις όπου έχουμε πεπερασμένο αριθμό πηγών και επιθυμούμε να μοντελοποιήσουμε διακριτά γεγονότα δικτυακής κίνησης. Αυτό προκύπτει από το νόμο των σπάνιων γεγονότων (οριακό Θεώρημα Poisson), το οποίο καθορίζει ότι όταν $n \to \infty$ και $p \to 0$, ώστε το $np = \lambda$ να παραμένει σταθερό τότε η Διωνυμική Κατανομή προσεγγίζει την Poisson, όπως αναφέρθηκε στο τμήμα 2.2.1.2. Επίσης το παραπάνω συμβαίνει και ως απόρροια του γεγονότος ότι, το ανάλογο του μοντέλου Poisson για διακριτό χρόνο είναι οι διεργασίες Bernoulli. Σε αυτή την περίπτωση, οι χρόνοι μεταξύ των αφίξεων ακολουθούν γεωμετρική κατανομή (έναντι της εκθετικής των διεργασιών Poisson).

Κίνηση Self-Similar Για να επιτευχθεί πιο αληθοφανής προσομοίωση της κίνησης που παρατηρείται σε σύγχρονα δίκτυα είναι πιο δόκιμη η χρήση του μοντέλου Self-Similar. Η έρευνα πάνω σε αυτό το θέμα έχει δείξει ότι η δικτυακή κίνηση που παράγεται σε πραγματικά τοπικά δίκτυα, για παράδειγμα κίνηση από μεταφορές αρχείων ή αναπαραγωγή πολυμέσων, παρουσιάζει τις ιδιότητες self-similarity και long-range dependency (LRD) [4]. Ο τύπος αυτος της κίνησης χαρακτηρίζεται από εξάρσεις οι οποίες ακολουθούνται από περιόδους ηρεμίας (bursty, ενώ ο κύκλος αυτός επαναλαμβάνεται σύμφωνα με μια διεργασία τύπου ON-OFF. Για την προσομοίωση τέτοιου τύπου δικτυακής κίνησης χρησιμοποιείται συνήθως η κατανομή Pareto.

2.3 Ισχύουσες Παραδοχές

Στο παρόν θα παρουσιαστούν οι γενικές παραδοχές που ακολουθούμε κατά τη μοντελοποίηση του πρωτοκόλλου slotted aloha σε τρεις φάσεις. Κατά την πρώτη φάση (stage1) το μοντέλο προβλέπει πεπερασμένο αριθμό σταθμών M, οι οποίοι δεν αποθηκεύουν πακέτα και επικοινωνούν μεταξύ τους με ένα διαμοιραζόμενο κανάλι. Στη δεύτερη φάση οι σταθμοί χρησιμοποιούν περισσότερα τους ενός κανάλια (stage2), ενώ στην τρίτη φάση οι σταθμοί διαθέτουν ουρές αναμονής (buffers) για την αποθήκευση των πακέτων που φτάνουν στο ίδιο προορισμό στο ίδιο timeslot (stage3).

- Δεχόμαστε πεπερασμένο πλήθος σταθμών M, ότι ο χρόνος είναι κβαντισμένος σε χρονοσχισμές (timeslot) και ότι οι αφίξεις ακολουθούν την διωνυμική κατανομή σε κάθε timeslot [9].
- Κάθε σταθμός είναι εξοπλισμένος με ένα receiving buffer και ένα transmiting buffer, χωρητικότητας ενός πακέτου τη φορά. Εάν το transmiting buffer είναι άδειο, τότε ο σταθμός χαρακτηρίζεται ως free, αλλιώς ως backlogged ή busy.
- Κάθε πακέτο έχει πληροφορίες για τη διεύθυνση σταθμού εκκίνησης και προορισμού.
- Εάν ένας busy σταθμός επιτύχει να εκπέμψει, τότε γίνεται free στο επόμενο timeslot. Ένας free σταθμός γίνεται busy στο επόμενο timeslot, εάν έχει αποτυχημένη εκπομπή.
- Τα κανάλια θεωρούμε ότι δεν εισάγουν σφάλματα κατά τη μετάδοση. Η καθυστέρηση διάδοσης εντός του καναλιού θεωρείται αμελητέα.

Μεγαλύτερο ερευνητικό ενδιαφέρον παρουσιάζει η μελέτη του stage3, καθώς αξιολογεί τη μεταβολή στις μετρικές απόδοσης του πρωτοκόλλου με δύο μεταβλητές (πλήθος receiving buffer και καναλιών επικοινωνίας). Η μαθηματική μοντελοποίση του stage3

αναλύεται στο [14] χρησιμοποιώντας μαρκοβιανή ανάλυση, για την εκτίμηση της απόδοσης, της καθυστέρησης και της πιθανότητας απόρριψης στον προορισμό. Στη συνέχεια περιγράφονται οι επιμέρους παραδοχές για κάθε ένα από τα stage3.

2.3.1 M σταθμοί / 1 κανάλι / 1 receiving buffer

Για τη μελέτη του πρωτοκόλλου θεωρούμε ότι:

- Τα πακέτα απορρίπτονται μόνο λόγω συγκρούσεων στο κανάλι.
- Όλοι οι σταθμοί παράγουν το ίδιο δικτυακό φορτίο.

2.3.2 M σταθμοί / N κανάλια / 1 receiving buffer

Για την μοντελοποίηση του πρωτοκόλλου για N κανάλια, ισχύει ότι και στην προηγούμενη περίπτωση με τις εξής επιπλέον παραδοχές:

- Υπάρχουν N το πλήθος παράλληλα κανάλια ίδιας χωρητικότητας, κάθε ένα από τα οποία συνιστά ένα ξεχωριστό broadcast domain.
- Κάθε σταθμός συνδέεται με κάθε ένα από τα N κανάλια με ξεχωριστά interfaces.
- Κάθε σταθμός επιλέγει τυχαία και ισοπίθανα ένα κανάλι για να εκπέμψει. Εάν δύο ή περισσότεροι σταθμοί επιλέξουν το ίδιο κανάλι για να εκπέμψουν στο ίδιο timeslot, τότε έχουμε σύγκρουση και απόρριψη των πακέτων πάνω στο κανάλι.

2.3.3 M σταθμοί / N κανάλια / F receiving buffers

Σε αυτή την περίπτωση ισχύουν ότι και στα προηγούμενα, επιπλέον όμως τα πακέτα απορρίπτονται είτε λόγω εκπομπής στο ίδιο κανάλι είτε λόγω του ότι το receiving buffer στο προορισμό είναι γεμάτο.

3 Προσομοίωση του Δικτύου

3.1 Εισαγωγή

Στο παρόν περιγράφεται η υλοποίηση της προσομοίωσης του πρωτοκόλλου slotted aloha χρησιμοποιώντας το αντίστοιχο μοντέλο και λαμβάνοντας υπόψη συγκεκριμένες παραδοχές.

3.2 Περιγραφή - Πάραδοχές Προσομοίωσης

3.2.1 Γενική Περιγραφή Μοντέλου

Για την προσαρμογή του μοντέλου θεωρούμε ως πόρους τα κανάλια και τους σταθμούς. Δυναμικές οντότητες είναι τα παραγόμενα πακέτα. Ως γεγονότα, θεωρούμε τις γεννήσεις πακέτων από τους σταθμούς, τις συγκρούσεις - απορρίψεις και τις επιτυχείς παραδόσεις των πακέτων στους προορισμούς. Ως μεταβλητές συστήματος καθορίζονται ανά timeslot: το πλήθος των επιτυχών παραδόσεων και το πλήθος των free - busy σταθμών.

Η προσομοίωση πραγματοποιείται με την δημιουργία και τήρηση του πίνακα προσομοίωσης του συστήματος. Ο πίνακας αυτός συμπληρώνεται λαμβάνοντας υπόψη, ότι κάθε γραμμή του αντιστοιχεί σε μια εκπομπή ενός νέου πακέτου στο σύστημα. Συγκεκριμένα ο πίνακας συμπληρώνεται ανά timeslot και ένα χαρακτηριστικό στιγμιότυπο του αποτυπώνεται στον πίνακα 3.1 (αφορά στο stage3, F=2).

arbitrary source	channel	boolean busy	dest	bool success	total packets on dest	successful packets on dest	bool destruction on channel
1	6	1	15	0	0	0	1
2	7	0	5	1	2	2	0
3	10	0	5	1	2	2	0
4	5	1	6	1	1	1	0
5	6	1	7	0	0	0	1

Πίνακας 3.1: Στιγμιότυπο Πίνακα Προσομοίωσης (stage3, F=2)

Ο παραπάνω πίνακας ενημερώνεται κατάλληλα από τον κώδικα προσομοίωσης, κάθε φορά που συμβαίνει ένα γεγονός, το οποίο επηρεάζει την προσομοίωση. Στο τέλος κάθε timeslot, ενημερώνονται κατάλληλα οι μεταβλητές κατάστασης του συστήματος με βάση τα συμπεράσματα που προκύπτουν από την ανάλυση του πίνακα προσομοίωσης.

3.2.2 Παραδοχές Προσομοίωσης

Για την μοντελοποίηση του πρωτοκόλλου χρησιμοποιούμε όλες τις συνθήκες που ορίστηκαν στο τμήμα 2.3 και επιπλέον:

- Η παραγωγή τυχαίων αριθμών σε Η/Υ δεν είναι πραγματικά τυχαία, αλλά ψευδοτυχαία.

- Η ακρίβεια των μαθηματικών πράξεων, κατά τη διάρκεια της προσομοίωσης εξαρτάται από το hardware, την αρχιτεκτονική του Η/Υ, την γλώσσα προγραμματισμού (εν προκειμένω την C), τον μεταγλωττιστή (compiler και το λειτουργικό σύστημα. Αυτό σημαίνει, ότι είναι δυνατό ο ίδιος πηγαίος κώδικας προσομοίωσης να παράγει ελαφρώς διαφορετικά αποτελέσματα, εφόσον αλλάζουν ένα ή περισσότερα από τα παραπάνω.

3.3 Υλοποίηση Προσομοίωσης σε Γλώσσα C

Στο τμήμα αυτό παρουσιάζονται τα κύρια σημεία του κώδικα της προσομοίωσης, έτσι όπως αυτός παρουσιάζεται στα Παραρτήματα Α′, Β′, Γ′. Στο Παράρτημα Δ′, παρουσιάζεται ο κώδικας, ο οποίος καλεί το εκτελέσιμο της προσομοίωσης με τα σωστά ορίσματα και παράγει τα σχετικά γραφήματα.

Επιγραμματικά, ο κώδικας του Παραρτήματος Δ′ αλλάζει τα ορίσματα M, N, FB για το πλήθος των σταθμών, των καναλιών και των receiving buffers. Στη συνέχεια μεταγλωττίζει τον κώδικα και εκτελεί όλες τις προσομοιώσεις για όλα τα ορίσματα ταυτόχρονα, με χρήση όλων των διαθέσιμων πυρήνων του επεξεργαστή προς κέρδος χρόνου. Στο τέλος της εκτέλεσης καλεί το πακέτο gnuplot για τη σχεδίαση των γραφημάτων της προσομοίωσης.

Ο κώδικας της προσομοίωσης σε C (Παραρτήματα Α′, Β′, Γ′) και των τριών περιπτώσεων, χωρίζεται σε τρία τμήματα: τη γεννήτρια ψευδοτυχαίων αριθμών για την προσομοίωση, τη συνάρτηση υπολογισμού των διωνυμικών συντελεστών και το κύριο μέρος της προσομοίωσης. Στα επόμενα αναλύεται ο κώδικας για την περίπτωση του stage3, κατά τμήμα. Ανάλογα ισχύουν και για τις άλλες περιπτώσεις της προσομοίωσης.

3.3.1 Γεννήτρια Τυχαίων Αριθμών

Για την παραγωγή των ψευδοτυχαίων αριθμών, που είναι απαραίτητοι για την προσομοίωση, χρησιμοποιήθηκε ένας αλγόριθμος Mersenne Twister. Ο κώδικας τροποποιήθηκε κατάλληλα για τις ανάγκες του προγράμματος. Η αρχική μορφή του κώδικα για 64-bit επεξεργαστές είναι διαθέσιμη στη σελίδα http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/C-LANG/mt19937-64.c.

Η επιλογή του αλγορίθμου αυτού έγινε έπειτα από αξιολόγηση των περισσότερο διαδεδομένων αλγορίθμων και της ενσωματωμένης συνάρτησης rand της γλώσσας C. Συγκεκριμένα, αναλύθηκαν χαρακτηριστικά όπως το μήκος της περιόδου και η ταχύτητά τους.

Linear Congruential Generators Ίσως από τους πιο διαδεδομένους αλγόριθμους. Είναι αρκετά γρήγοροι και με μικρές απαιτήσεις σε μνήμη, ιδίως αν το m είναι δύναμη του 2, $m = 2^{32}$ ή $m = 2^{64}$, γιατί επιτρέπει τον υπολογισμό του υπολοίπου με τη χρήση bitshift [2]. Ο Knuth στο [11] έχει ένα εκτεταμένο κείμενο για την επιλογή όλων των σταθερών του αλγορίθμου με κάποια κριτήρια και διεξάγει αναλυτικά τεστ τυχαιότητας στον αλγόριθμο. Η ενσωματωμένη συνάρτηση rand() της C χρησιμοποιεί τον ίδιο αλγόριθμο (linear congruential generator) με μήκος περιόδου 2^{31} ή έναν additive feedback generator με πολύ μεγαλύτερη περίοδο ανάλογα με τον τρόπο κλήσης της [2]. Στο [12] παρέχονται εκτεταμένοι πίνακες με καλές επιλογές σταθερών για τον αλγόριθμο. Τέλος, σύμφωνα με το [2], ο αλγόριθμος αυτός δεν θα πρέπει να χρησιμοποιείται όπου είναι αναγκαία υψηλής ποιότητας τυχαιότητα.

Mersenne Twister Είναι ο πιο συχνά χρησιμοποιούμενος αλγόριθμος. Χρησιμοποιείται από πολλά επαγγελματικά μαθηματικά πακέτα, όπως τα Matlab, Maple, Mathematica, Stata, R, Octave κ.α. Το μήκος περιόδου του είναι $2^{219937}-1$ (σε αντιδιαστολή με το 2^{32} του linear congruential generator). Τα μειονεκτήματά του είναι ότι είναι σχετικά αργός και δεν είναι κρυπτογραφικά ασφαλής (όπως ούτε και ο linear congruential generator).

Για να γίνει μια αντιπροσωπευτική δοκιμή των δύο αλγορίθμων παραγωγής ψευδοτυχαίων αριθμών, πραγματοποιήθηκαν προσομοιώσεις του πρωτοκόλλου με τη χρήση του κώδικα του Παραρτήματος Α΄. Σε μία περίπτωση χρησιμοποιήθηκε ένας linear congruential generator με μήκος περιόδου $2^{32}-5$ και στη δεύτερη περίπτωση ο mersenne twister του Παραρτήματος Α΄. Στη συνέχεια πραγματοποιήθηκε έλεγχος RSS για να διαπιστωθεί πόσο κοντά στις αναμενόμενες τιμές είναι αυτές που παράγονται από την προσομοίωση. Ο στατιστικός έλεγχος έδειξε ότι, για τις ίδιες τιμές των timeslots, καλύτερη προσαρμογή στο θεωρητικό μοντέλο έχει η mersenne twister γεννήτρια (μικρότερη απόκλιση). Επίσης, η linear congruential φαίνεται να έχει μεγάλες διακυμάνσεις στο throughput και το delay, παρά το μεγάλο πλήθος των timeslots R=100000000. Συνοπτικά, τα αποτελέσματα του ελέγχου προσαρμογής φαίνονται στους πίνακες 3.2, 3.3.

Throughput S	Delay D	Mean Busy B
1.30918E-06	3678.685	0.001775641

Πίνακας 3.2: Έλεγχος RSS (mersenne twister)

Throughput S	Delay D	Mean Busy B
0.003817639	51803.66	6.184416289

Πίνακας 3.3: Έλεγχος RSS (linear congruential)

3.3.2 Υπολογισμός Διωνυμικών Συντελεστών

Για τον υπολογισμό των διωνυμικών συντελεστών, οι οποίοι χρειάζονται για τον υπολογισμό της αντίστροφης συνάρτησης δυωνυμικής κατανομής, δοκιμάστηκαν διάφορες μέθοδοι.

Αρχικά για μικρό πλήθος σταθμών M χρησιμοποιήθηκε ο ορισμός των διωνυμικών συντελεστών με τη χρήση συνάρτησης παραγοντικού.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Η μέθοδος αυτή γρήγορα φτάνει τον H/Υ στα όριά του καθώς δε μπορούν να αποθηκευτούν σε κατάλληλο τύπο δεδομένων (long long) τα ενδιάμεσα αποτελέσματα της συνάρτησης παραγοντικού για ακέραιους μεγαλύτερους από 21. Αυτό μπορεί να ξεπεραστεί με την τροποποίηση του κώδικα για τη χρησιμοποίηση του GNU Multiple Precision Arithmetic Library, GMP αλλά και πάλι ο υπολογισμός του παραγοντικού είναι μη αποδοτικός, καθώς είναι αναδρομικός και έχει μεγάλες απαιτήσεις σε προσωρινή μνήμη. Υπάρχει βεβαίως, μια εναλλακτική λύση για τον υπολογισμό του παραγοντικού η οποία απλοποιεί τις πράξεις και τις υπολογιστικές απαιτήσεις και βασίζεται στην προσέγγιση Stirling. Σε αυτή την περίπτωση οι διωνυμικοί συντελεστές προσεγγίζονται ικανοποιητικά (σχεδόν ισότητα) από τη σχέση:

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Παρόλα αυτά, η χρήση του ορισμού για τον υπολογισμό των διωνυμικών συντελεστών ενέχει δύο υπολογισμούς παραγοντικού και μια διαίρεση, πράγμα το οποίο καθιστά τον αλγόριθμο υπολογιστικά εντατικό καθώς αυξάνεται το πλήθος των σταθμών.

Μια δεύτερη πιθανή υλοποίηση της συνάρτησης είναι η χρήση ενός προσυμπληρωμένου πίνακα διωνυμικών συντελεστών (επί της ουσίας ένας τετραγωνικός πίνακας με στοιχεία αυτά του τριγώνου Pascal), ενσωματωμένου στον πρόγραμμα. Είναι μια γρήγορη υλοποίηση με μικρό κόστος σε μνήμη ακόμα και για μεγάλους ακεραίους. Επίσης η πράξη που εκτελείται κάθε φορά είναι απλά πρόσβαση σε συγκεκριμένη διεύθυνση μνήμης. Το μειονέκτημά της είναι ότι δεν είναι δυναμική, καθώς πρέπει να αλλάζει ο πίνακας με την αύξηση του πλήθους των σταθμών και δεν είναι κομψή προγραμματιστικά.

Υπάρχει άλλος ένας διαδεδομένος αλγόριθμος για τον υπολογισμό των διωνυμικών συντελεστών, ο οποίος βασίζεται στην συνάρτηση Γάμμα. Οι διωνυμικοί συντελεστές υπολογίζονται από τη σχέση:

$$\binom{n}{k} = \frac{\Gamma(n+1)}{\Gamma(k+1)\Gamma(n-k+1)}$$

Επειδή όμως η συνάρτηση Γάμμα δεν είναι ενσωματωμένη στην C και ο υπολογισμός της είναι γενικά πολύπλοκος, δεν προτιμήθηκε για την προσομοίωση.

Αντίθετα, σε αυτή την περίπτωση είναι βολική η χρήση δυναμικού προγραμματισμού για τη δημιουργία με δυναμικό τρόπο του πίνακα διωνυμικών συντελεστών, που περιγράφηκε προηγουμένως. Ο δυναμικός προγραμματισμός κρύβεται πίσω από τον αλγόριθμο πλήρωσης του πίνακα. Κάθε γραμμή συμπληρώνεται αναδρομικά σύμφωνα με τον αλγόριθμο:

Algorithm 3 Dynamic Programming Binomial

```
1: function BINOMIAL(n, k)
2:     for i ← 0, n do                          ▷ fill out the table row wise
3:         for i ← 0, min(i, k) do
4:             if j==0 or j==i then
5:                 C[i, j] ← 1
6:             else
7:                 C[i, j] ← C[i − 1, j − 1] + C[i − 1, j]      ▷ recursive relation
8:     return C[n, k]
```

Μόνο η μισή γραμμή χρειάζεται υπολογισμό γιατί οι τιμές είναι ίσες αντιδιαμετρικά. Οπότε οι εντατικοί υπολογισμοί εκτελούνται μόνο στο πρώτο μισό του πίνακα και η κύρια πράξη είναι η πρόσθεση. Ο κώδικας υλοποίησης του αλγορίθμου φαίνεται στο Παράρτημα Γ′ (γραμμές 126 - 133).

3.3.3 Κύριο Μέρος

Η προσομοίωση εξετάζει μακροσκοπικά το δικτυακό πρωτόκολλο, από την άποψη ότι δεν μοντελοποιείται ξεχωριστά κάθε σταθμός που συμμετέχει στο δίκτυο. Το κύριο μέρος ξεκινά από τη γραμμή 134 με την προετοιμασία των μεταβλητών για την εκτέλεση της προσομοίωσης και του επόμενου timeslot. Εκτελούνται 1000 κύκλοι των R timeslots, με κάθε κύκλο να εκτελείται με αυξανόμενη πιθανότητα επανεκπομπής των busy σταθμών. Σε κάθε timeslot υπολογίζονται αριθμητικά το πλήθος των busy, free σταθμών και με τη βοήθεια της αντίστροφης συνάρτησης διωνυμικής κατανομής υπολογίζεται το πλήθος των πακέτων που εκπέμπουν οι σταθμοί ανά κατηγορία. Η διαδικασία αυτή εκτελείται από τις γραμμές 152 - 169.

Στη συνέχεια ενημερώνεται ο πίνακας της προσομοίωσης, όπως περιγράφηκε στο τμήμα 3.2. Το k στο for loop της γραμμής 174 αντιπροσωπεύει το σταθμό που εκπέμπει. Επειδή δεν έχει σημασία για τον υπολογισμό, οι δείκτες δεν αντιστοιχούν σε κάποιο συγκεκριμένο σταθμό, αλλά σε αυτόν που εξέπεμψε με σειρά k. Το k εξετάζεται μόνο κατά την επιλογή του σταθμού προορισμού, όπου ο σταθμός στην k γραμμή του πίνακα επιλέγει να στείλει σε έναν οποιοδήποτε από τους M-1 προορισμούς πλην του k. Ο πίνακας συμπληρώνεται με τέτοιο τρόπο, ώστε γραμμές που αντιστοιχούν σε εκπομπές busy ή free σταθμών να είναι με τυχαία σειρά και όχι πρώτα οι busy ή το αντίστροφο. Αυτό εξασφαλίζει ότι δεν υπάρχει μεροληψία κατά την επιλογή των σταθμών που θα απορριφθούν ή θα γίνουν αποδεκτοί στη φάση της αξιολόγησης του πίνακα, καθώς αυτός εξετάζεται διαδοχικά γραμμή προς γραμμή. Οι αντίστοιχες γραμμές του κώδικα είναι 171 - 196.

Έπειτα, ο πίνακας αξιολογείται για συγκρούσεις πάνω στα κανάλια, όταν δηλαδή δύο ή περισσότεροι σταθμοί επιλέγουν το ίδιο κανάλι για να εκπέμψουν (γραμμές 198 - 210). Στο επόμενο for loop (γραμμές 212 - 222) πραγματοποιείται ένας πρώτος έλεγχος για να υπολογισθεί το πλήθος των πακέτων που προορίζονται για κάθε σταθμό.

Στο επόμενο for loop εξετάζεται ο πίνακας για συγκρούσεις στον προορισμό, όταν δηλαδή σε έναν προορισμό καταφθάνουν περισσότερα πακέτα από όσα μπορούν να αποθηκευτούν στο receiving buffer του. Σε αυτή την περίπτωση επιλέγονται τυχαία τα πακέτα τα οποία θα γίνουν αποδεκτά και όταν το receiving buffer γεμίσει, τυχόν πακέτα που εμφανίζονται αργότερα, απορρίπτονται (γραμμές 227 - 241).

Για τον υπολογισμό του throughput του πρωτοκόλλου, είναι αναγκαίο όταν το πλήθος των πακέτων που καταφθάνουν σε ένα σταθμό, είναι ίσα ή περισσότερα από το receiving buffer, τότε αυτό να εμφανίζεται γεμάτο. Αυτό τον έλεγχο πραγματοποιεί το for loop των γραμμών 243 - 252. Εξασφαλίζει ότι οι τυχαίες απορρίψεις πακέτων του προηγούμενου for loop δεν ήταν τόσες ώστε το receiving buffer να είναι άδειο (όταν φυσικά έχει φτάσει ικανος αριθμός πακέτων στον προορισμό).

Οι επόμενες γραμμές υπολογίζουν τελικά πόσα πακέτα από busy ή free σταθμούς παραδόθηκαν επιτυχώς (γραμμές 254 - 260). Τέλος, στις γραμμές 262 - 273 καταχωρούνται οι τιμές των busy σταθμών για το επόμενο timeslot και ο αριθμός των συνολικών επιτυχιών, ο οποίος χρησιμοποιείται στο τέλος της προσομοίωσης για τον υπολογισμό του throughput S. Η γραμμή 275 επισημαίνει το τέλος του timeslot και η 278 τυπώνει τα αποτελέσματα του κύκλου. Το throughput υπολογίζεται ως το σύνολο των επιτυχώς παραδομένων πακέτων στον κύκλο προς το πλήθος των timeslots και το delay εξάγεται από το νόμο του Little.

Στο επόμενο κεφάλαιο παρουσιάζονται τα αποτελέσματα της προσομοίωσης για συγκεκριμένους συνδυασμούς ορισμάτων και εξάγονται τα συμπεράσματα της μελέτης.

4 Παρουσίαση Αποτελεσμάτων

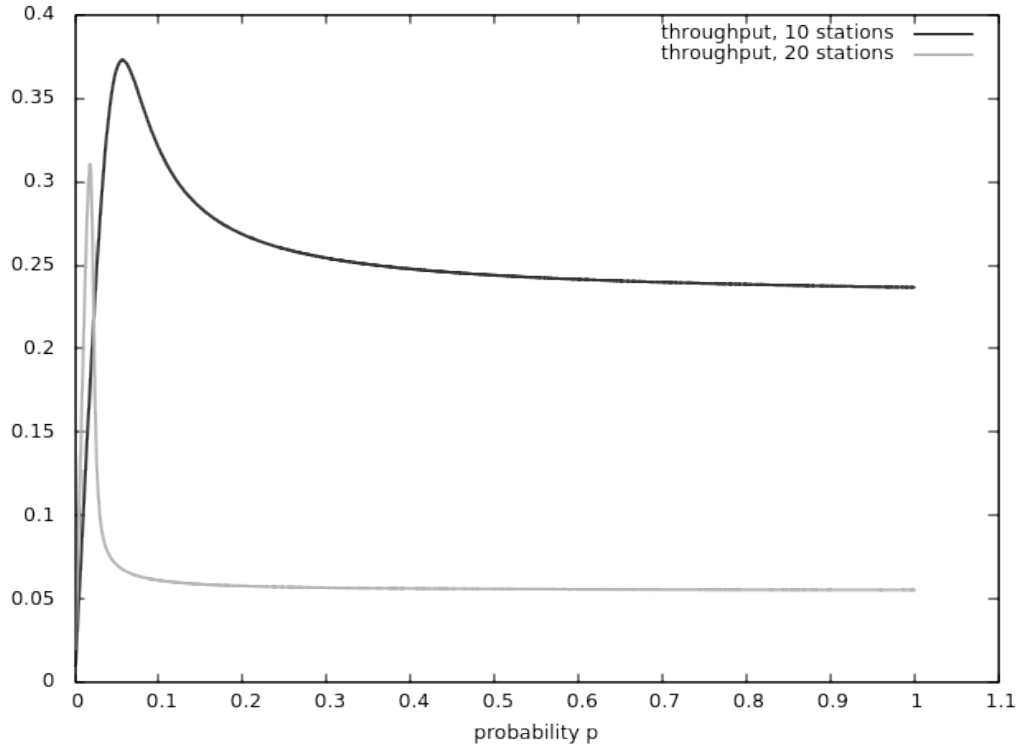
4.1 Εισαγωγή

Στο τμήμα αυτό παρουσιάζονται τα γραφήματα που παρήχθησαν μέσω της προσομοίωσης για συγκεκριμένα ορίσματα. Αρχικά θα παρουσιαστούν τα αποτελέσματα του stage1 για πλήθος σταθμών 10 και 20. Στη συνέχεια τα αποτελέσματα του stage2 για πλήθος σταθμών 50, 100 και τέλος αυτά του stage3 για πλήθος σταθμών 50, 100.

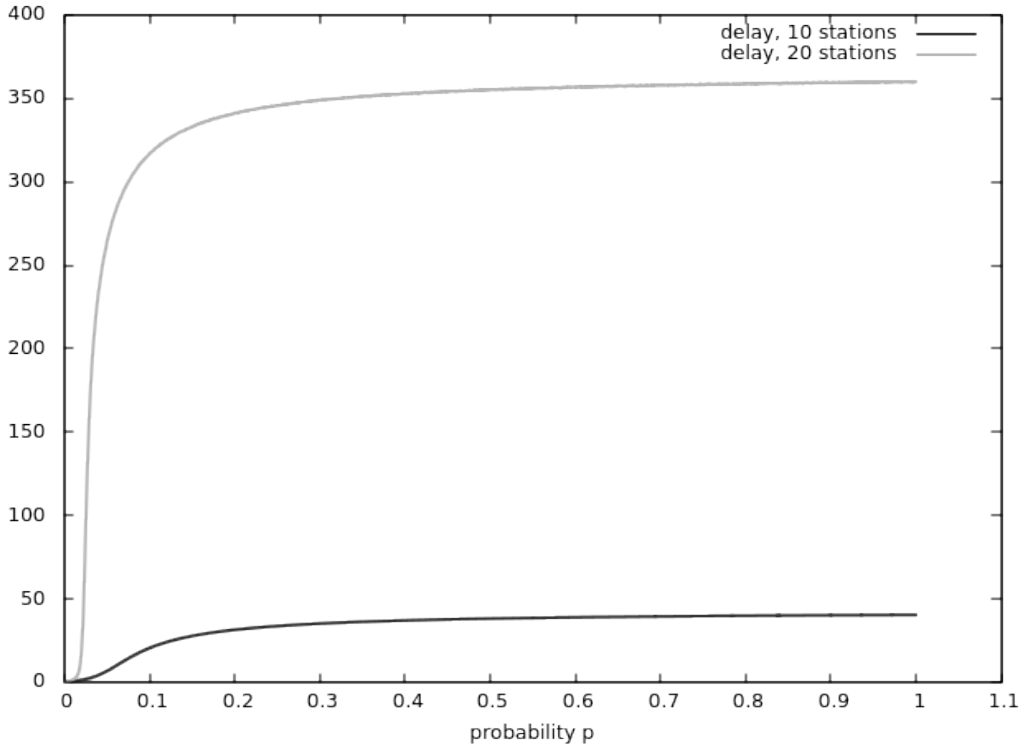
4.2 Μέτρηση Απόδοσης για Μεταβαλλόμενα Μεγέθη

4.2.1 M σταθμοί / 1 κανάλι / 1 receiving buffer (stage1)

Στα παρακάτω σχήματα παρουσιάζεται η εξέλιξη των delay, throughput σε σχέση με την πιθανότητα επανεκπομπής των busy σταθμών. Είναι εμφανές ότι σε αυτή την περίπτωση υπάρχει μεγάλος ανταγωνισμός για το κανάλι και για αυτό το λόγο υπάρχει υψηλή καθυστέρηση. Σε σχέση όμως με το pure aloha παρατηρείται διπλασιασμός του throughput S.



Σχήμα 4.1: Stage1 Throughput



Σχήμα 4.2: Stage1 Delay

4.2.2 M σταθμοί / N κανάλια / 1 receiving buffer (stage2)

Σε αυτή την περίπτωση επαυξάνουμε την δυνατότητα μεταγωγής πακέτων του συστήματος αυξάνοντας τα διαθέσιμα κανάλια πάνω στα οποία γίνεται ο ανταγωνισμός των σταθμών. Τα αποτελέσματα της προσομοίωσης παρουσιάζονται στα σχήματα 4.3, 4.4, 4.5 και 4.6.

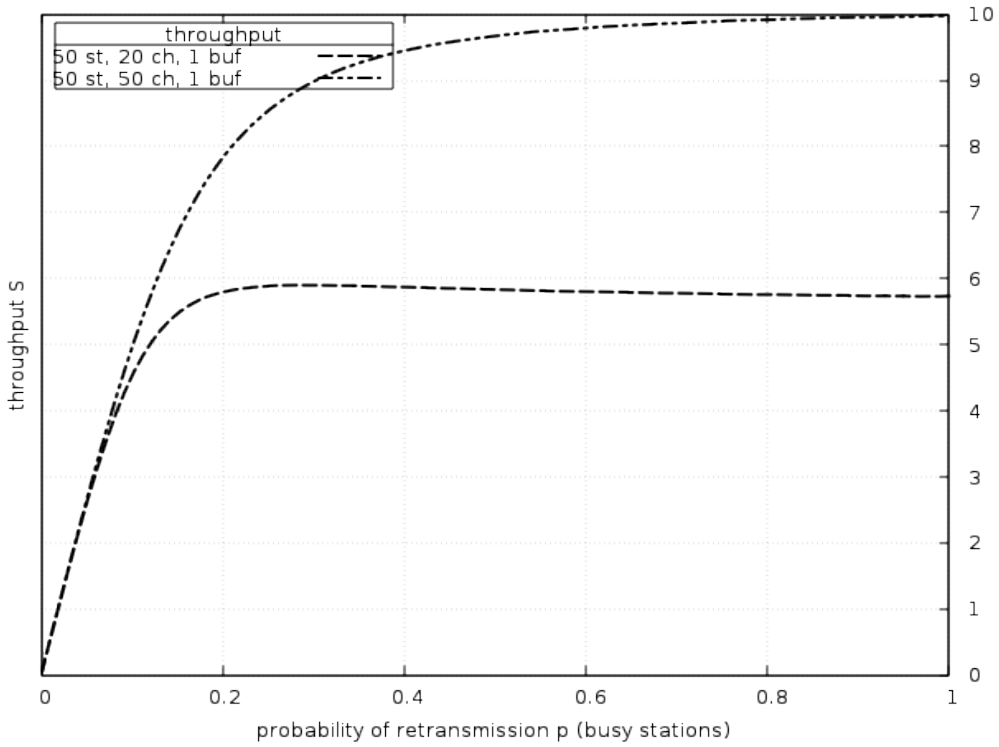
Πράγματι είναι εμφανής η αύξηση του throughput καθώς αυξάνεται το πλήθος των διαθέσιμων καναλιών.

4.2.3 M σταθμοί / N κανάλια / F receiving buffer (stage3)

Στο stage3 γίνεται αύξηση του αριθμού των πακέτων που μπορεί ένας σταθμός να αποθηκεύσει στο receiving buffer του, ώστε να τα επεξεργαστεί σε δεύτερο χρόνο. Η διαφορά από την αύξηση του receiving buffer από 1 σε 2 είναι σημαντική, όμως η αύξηση του σε 3 και πάνω δεν δίνει εμφανή αποτελέσματα. Οι απεικονίσεις για το stage3 φαίνονται στα γραφήματα 4.7, 4.8, 4.9 και 4.10:

4.3 Συμπεράσματα

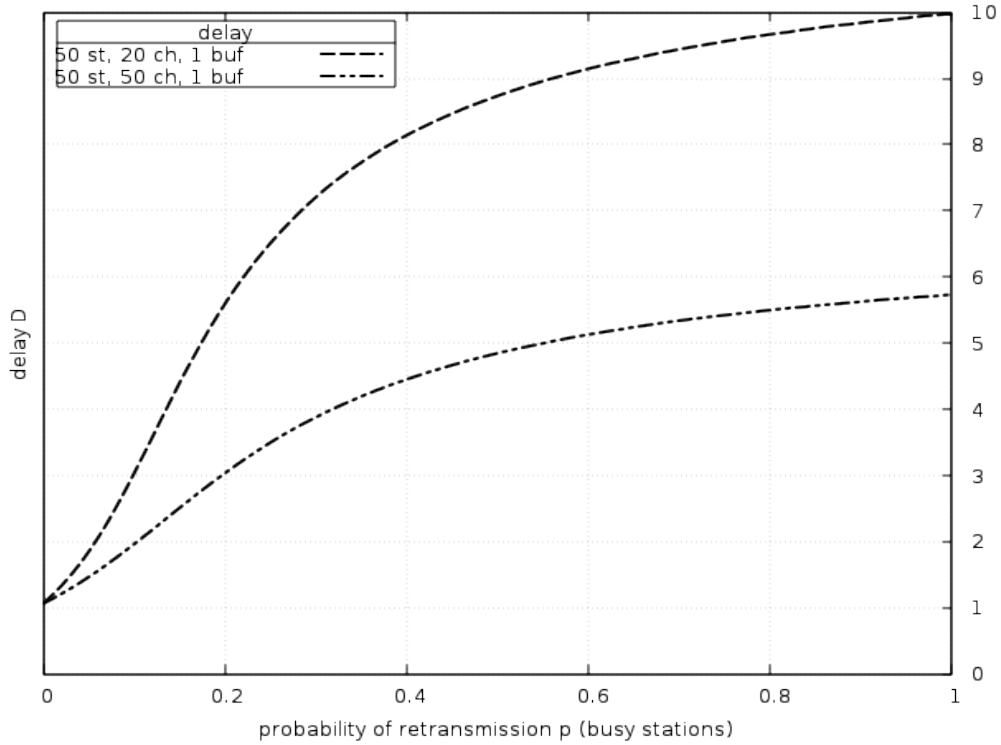
Στο παρόν, παρουσιάστηκε ένα μοντέλο προσομοίωσης του πρωτοκόλλου πολλαπλής πρόσβασης μέσου (MAC) τύπου slotted aloha. Τα αποτελέσματα της προσομοίωσης έδειξαν τιμές πολύ κοντά στις θεωρητικά αναμενόμενες. Συγκεκριμένα, η αύξηση των μνημών προσωρινής αποθήκευσης πακέτων (receiving buffers) στους σταθμούς, είχε θετική επίδραση στην απόδοση του συστήματος S, η οποία όμως εμφανίζονταν φθίνουσα, καθώς αυξάνονταν το πλήθος των σταθμών M. Επίσης, η αύξηση του πλήθους των receiving buffers πάνω από δύο, δεν είχε κάποια σημαντική επίδραση στην αποδοτικότητα του συστήματος. Αντιθέτως, η αύξηση των διαύλων επικοινωνίας μεταξύ των κόμβων είχε σημαντικά θετική επίδραση στην διεκπεραιωτική δυνατότητα του συστήματος και τη



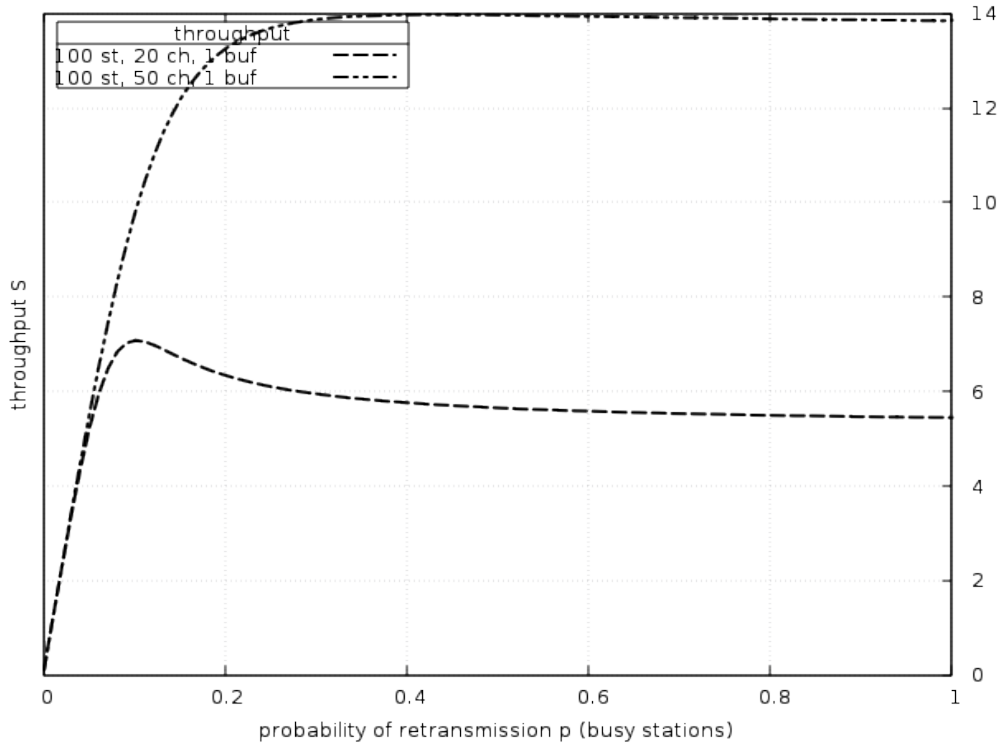
Σχήμα 4.3: Stage2 Throughput M=50

μείωση της καθυστέρησης, ανεξάρτητα από το πλήθος των σταθμών.

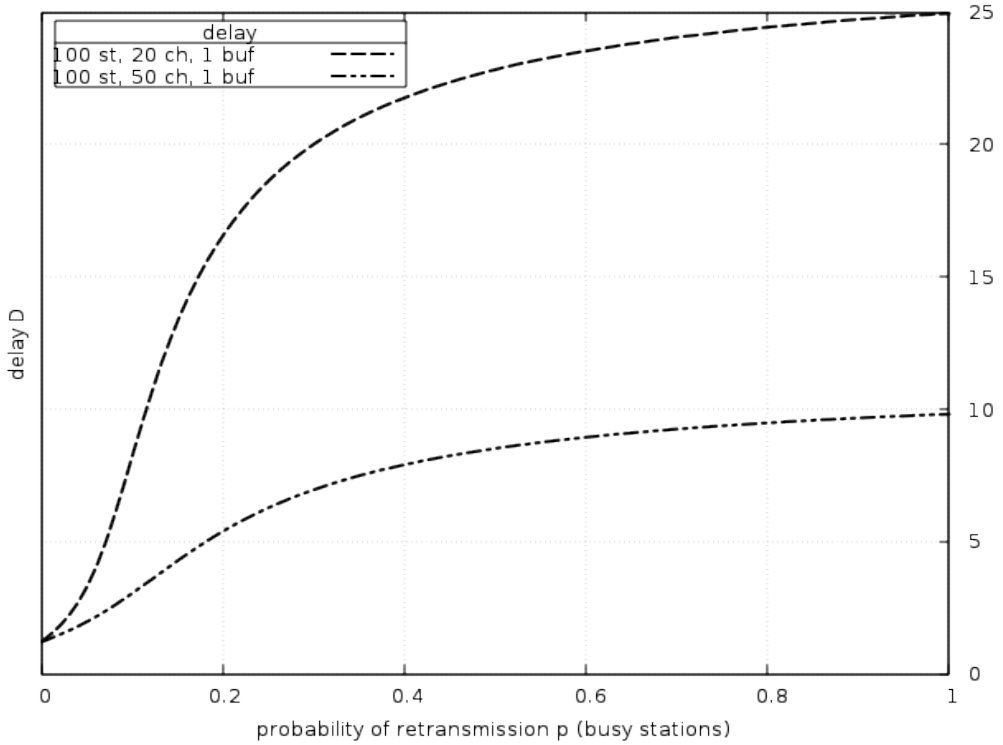
Τα παραπάνω καταδεικνύουν ότι η βέλτιστη απόδοση του πρωτοκόλλου επιτυγχάνεται όταν οι σταθμοί έχουν δύο το πλήθος receiving buffers και ικανό αριθμό καναλιών - διαύλων επικοινωνίας μεταξύ τους (ο οποίος μπορεί να περιορίζεται από τεχνικούς περιορισμούς ή περιορισμούς κόστους). Στην περίπτωση των ασύρματων δικτύων αυτό μπορεί να μεταφραστεί: είτε σε ικανό αριθμό πομποδεκτών ανά σταθμό, οι οποίοι θα λειτουργούν σε διαφορετικές συχνότητες για λόγους αποφυγής παρεμβολών, είτε σε κατάλληλο σχήμα πολύπλεξης (TDM, FDM), επαναχρησιμοποίηση φάσματος με αναπήδηση συχνότητας κ.α. Κατά αντιστοιχία, στα οπτικά δίκτυα είναι δυνατή η χρησιμοποίηση πολύτροπων οπτικών ινών και εκπομπή των δεδομένων σε διαφορετικά μήκη κύματος για την ταυτόχρονη χρησιμοποίηση του μέσου.



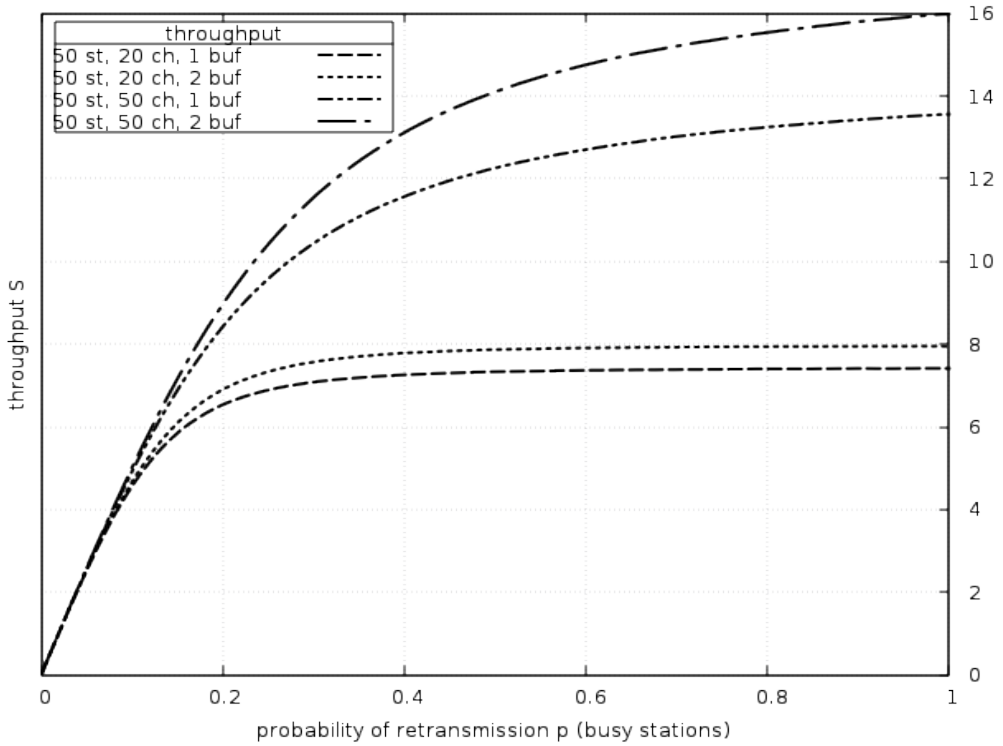
Σχήμα 4.4: Stage2 Delay M=50



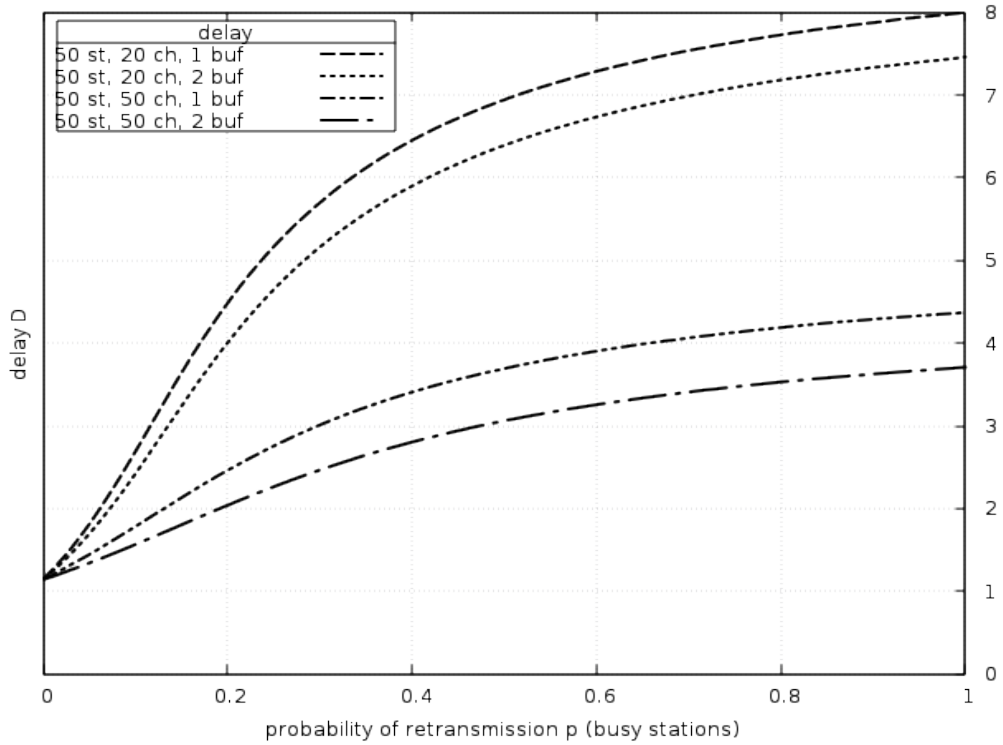
Σχήμα 4.5: Stage2 Throughput M=100



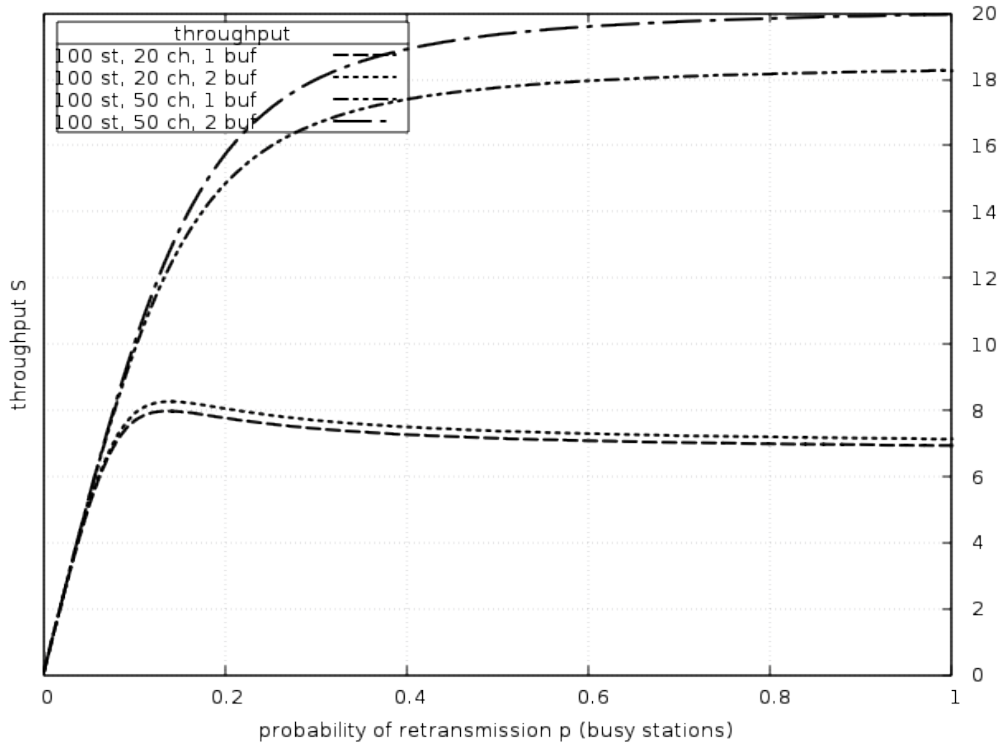
Σχήμα 4.6: Stage2 Delay M=100



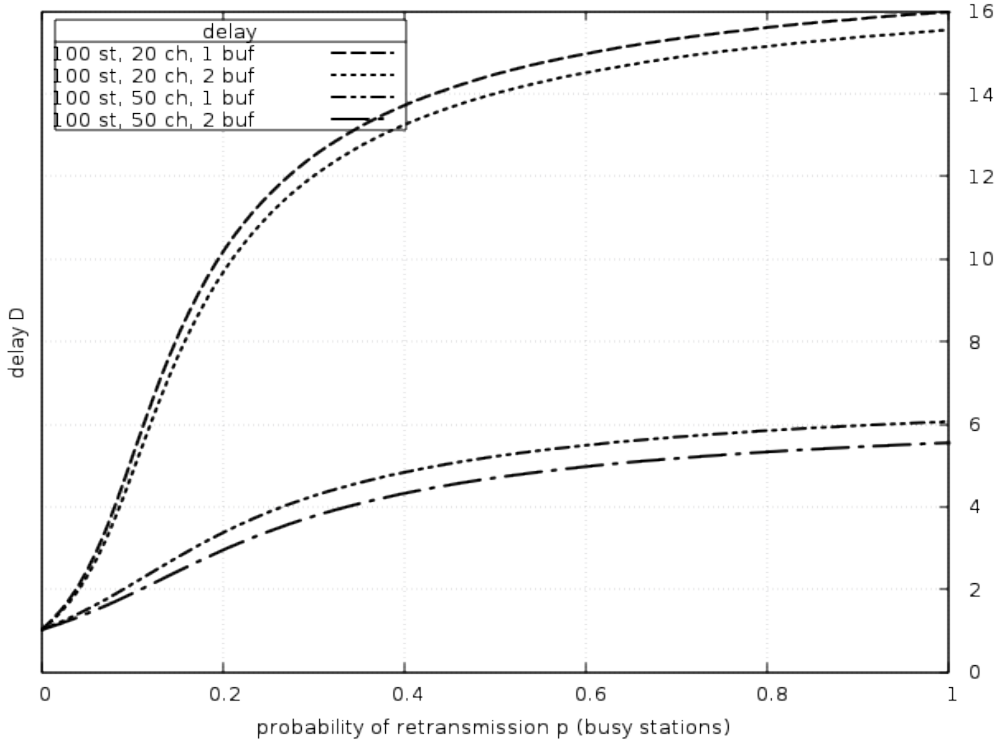
Σχήμα 4.7: Stage3 Throughput M=50



Σχήμα 4.8: Stage3 Delay M=50



Σχήμα 4.9: Stage3 Throughput M=100



Σχήμα 4.10: Stage3 Delay M=100

Παραρτήματα

Α΄ Κώδικας Προσομοίωσης σε C, stage1

```c
/*****************************************************************************************************
* slottedAloha.c
* Computes simulated throughput, delay, mean busy stations of slotted ALOHA
* protocol, based on certain assumptions.
*
* To compile, run 'gcc slottedAloha.c -O3 -o slottedAloha -lm', on linux or
* other *nix command line.
*
* Politis Dimitrios, Mar 2017
*
* Uses parts from Mersenne Twister Random Number Generator, found at
* http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/C-LANG/mt19937-64.c
*
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*
*
* The source code can be found at
* https://github.com/dpolitis/Slotted-Aloha-Simulation
\*****************************************************************************************************/

/* header includes */
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>

/* defines */
#define NN 312
#define MM 156
#define MATRIX_A 0xB5026F5AA96619E9ULL
#define UM 0xFFFFFFFF80000000ULL          // most significant 33 bits
#define LM 0x7FFFFFFFULL                  // least significant 31 bits

#define M 20                              // number of stations
#define R 100000000                       // number of time-slots per invocation

/* global variable definition */
int i, j;                                 // counters of transmissions per invocation
int B;                                    // counter of backlogged stations
unsigned int C;                           // total time-slot counter
unsigned int S;                           // total succesful time-slot counter
unsigned int busy;                        // total busy stations in each invocation
double bc[M+1][M+1];                      // binomial coefficients array

double A, F;                              // IDF functions
double X1, X2;                            // random numbers [0,1]
double p = 0.0;                           // probability of free stations
double r = 0.3;                           // probability of busy stations

/* The array for the state vector */
static unsigned long long mt[NN];

/* mti==NN+1 means mt[NN] is not initialized */
static int mti=NN+1;

/* initializes mt[NN] with a seed */
void init_genrand64(unsigned long long seed) {
  mt[0] = seed;
```

```
70    for (mti=1; mti<NN; mti++)
71      mt[mti] =  (6364136223846793005ULL * (mt[mti-1] ^ (mt[mti-1] >> 62)) + mti);
72  }
73
74  /* generates a random number on [0, 2^64-1]-interval */
75  unsigned long long genrand64_int64(void) {
76    int i;
77    unsigned long long x;
78    static unsigned long long mag01[2]={0ULL, MATRIX_A};
79
80    if (mti >= NN) {                        // generate NN words at one time
81
82      /* if init_genrand64() has not been called, a default initial seed is used     */
83      if (mti == NN+1)
84        init_genrand64(5489ULL);
85
86      for (i=0;i<NN-MM;i++) {
87        x = (mt[i]&UM)|(mt[i+1]&LM);
88        mt[i] = mt[i+MM] ^ (x>>1) ^ mag01[(int)(x&1ULL)];
89      }
90      for (;i<NN-1;i++) {
91        x = (mt[i]&UM)|(mt[i+1]&LM);
92        mt[i] = mt[i+(MM-NN)] ^ (x>>1) ^ mag01[(int)(x&1ULL)];
93      }
94      x = (mt[NN-1]&UM)|(mt[0]&LM);
95      mt[NN-1] = mt[MM-1] ^ (x>>1) ^ mag01[(int)(x&1ULL)];
96
97      mti = 0;
98    }
99
100   x = mt[mti++];
101
102   x ^= (x >> 29) & 0x5555555555555555ULL;
103   x ^= (x << 17) & 0x71D67FFFEDA60000ULL;
104   x ^= (x << 37) & 0xFFF7EEE000000000ULL;
105   x ^= (x >> 43);
106
107   return x;
108 }
109
110 /* generates a random number on [0,1]-real-interval */
111 double genrand64_real1(void) {
112   return (genrand64_int64() >> 11) * (1.0/9007199254740991.0);
113 }
114
115 /* begin code */
116 int main () {
117   /* fill binomial coefficients table */
118   for (i = 0; i < M + 1; i++) {           // fill each line iteratively
119     for (j = 0; j < M + 1; j++) {         // using dynamic programming
120       if (j > i) bc[i][j] = 0.0;
121       else if ((j == 0) || (j == i)) bc[i][j] = 1.0;
122       else bc[i][j] = (j <= 1 + i / 2) ? bc[i - 1][j - 1] + bc[i -1][j]: bc[i][i-j];
123     }
124   }
125
126   do {
127
128     /* initialize with random seed */
129     init_genrand64((unsigned)time(NULL));
130
131     p = p + 0.001;                        // increase probability at each invocation
132     B = 0;
133     C = 0;
134     S = 0;
135     busy = 0;
136
137     /* until simulation is over */
138     do {
139       /* initialize variables */
140       A = 0.0, i = 0;
141       X1 = genrand64_real1();
142
143       /* compute IDF to find how many free stations transmitted */
144       do {
145         A = A + bc[M - B][i] * pow(p, (double)i) * pow((1.0 - p), (double)(M - B - i));
```

```
146
147           if (X1 < A) { break; }
148             i++;
149         } while (i <= (M - B));
150
151         /* initialize variables */
152         F = 0.0, j = 0;
153         X2 = genrand64_real1();
154
155         /* compute IDF to find how many busy stations transmitted */
156         do {
157           F = F + bc[B][j] * pow(r, (double)j) * pow((1.0 - r), (double)(B - j));
158
159           if (X2 < F) { break; }
160             j++;
161         } while (j <= B);
162
163         /* define the outcome */
164         if ((i == 1) && (j == 0)) {
165           /* success, free station transmitted, increase successful timeslot counter */
166           S++;
167         }
168         else if ((i == 0) && (j == 1)) {
169           /* success, busy station transmitted */
170           S++;
171           B--;
172         }
173         else {
174           /* collision occured, increase busy stations by free stations transmitted,
175           increase fail counter for busy stations */
176           B+= i;
177         }
178
179         /* end of timeslot, proceed to next one */
180         C++;
181         busy+= B;
182
183       } while (C < R);                    // end of simulation cycle
184
185       /* printout simulated throughput, delay, mean busy stations */
186       printf("%.16f \t %.16f \t %.16f %.16f\n", p, (double)S / (double)R, 1.0 + (double)(busy) / (double)S,
          ↪   (double)busy / (double)R);
187
188     } while (p < 1.0);
189
190     return 0;
191   }
```

Β′ Κώδικας Προσομοίωσης σε C, stage2

```c
/*****************************************************************************************************
* slottedAloha.c
* Computes simulated throughput, delay, mean busy stations of slotted ALOHA
* protocol, based on certain assumptions.
*
* To compile, run 'gcc slottedAloha.c -O3 -o slottedAloha -lm', on linux or
* other *nix command line.
*
* Politis Dimitrios, Mar 2017
*
* Uses parts from Mersenne Twister Random Number Generator, found at
* http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/C-LANG/mt19937-64.c
*
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*
*
* The source code can be found at
* https://github.com/dpolitis/Slotted-Aloha-Simulation
\*****************************************************************************************************/

/* header includes */
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>

/* defines */
#define NN 312
#define MM 156
#define MATRIX_A 0xB5026F5AA96619E9ULL
#define UM 0xFFFFFFFF80000000ULL          // most significant 33 bits
#define LM 0x7FFFFFFFULL                  // least significant 31 bits

#define M 20                              // number of stations
#define N 15                              // number of channels
#define R 100000000                       // number of time-slots per invocation

/* global variable definition */
int i, ii, j, jj;                         // counters of transmissions per invocation
int k, f;                                 // general purpose counters
int B;                                    // counter of backlogged stations
int C;                                    // total time-slot counter
int S;                                    // total succesful time-slot counter
int busy;                                 // total busy stations in each invocation
int sim[N][4];                            // simulation array
double bc[M+1][M+1];                      // binomial coefficients array

double A, F;                              // IDF functions
double X1, X2;                            // random numbers [0,1]
double p = 0.0;                           // probability of free stations
double r = 0.3;                           // probability of busy stations

/* The array for the state vector */
static unsigned long long mt[NN];

/* mti==NN+1 means mt[NN] is not initialized */
static int mti=NN+1;

```

```
70  /* initializes mt[NN] with a seed */
71  void init_genrand64(unsigned long long seed) {
72    mt[0] = seed;
73    for (mti=1; mti<NN; mti++)
74      mt[mti] =  (6364136223846793005ULL * (mt[mti-1] ^ (mt[mti-1] >> 62)) + mti);
75  }
76
77  /* generates a random number on [0, 2^64-1]-interval */
78  unsigned long long genrand64_int64(void) {
79    int i;
80    unsigned long long x;
81    static unsigned long long mag01[2]={0ULL, MATRIX_A};
82
83    if (mti >= NN) {                        // generate NN words at one time
84
85      /* if init_genrand64() has not been called, a default initial seed is used */
86      if (mti == NN+1)
87        init_genrand64(5489ULL);
88
89      for (i=0;i<NN-MM;i++) {
90        x = (mt[i]&UM)|(mt[i+1]&LM);
91        mt[i] = mt[i+MM] ^ (x>>1) ^ mag01[(int)(x&1ULL)];
92      }
93      for (;i<NN-1;i++) {
94        x = (mt[i]&UM)|(mt[i+1]&LM);
95        mt[i] = mt[i+(MM-NN)] ^ (x>>1) ^ mag01[(int)(x&1ULL)];
96      }
97      x = (mt[NN-1]&UM)|(mt[0]&LM);
98      mt[NN-1] = mt[MM-1] ^ (x>>1) ^ mag01[(int)(x&1ULL)];
99
100     mti = 0;
101   }
102
103   x = mt[mti++];
104
105   x ^= (x >> 29) & 0x5555555555555555ULL;
106   x ^= (x << 17) & 0x71D67FFFEDA60000ULL;
107   x ^= (x << 37) & 0xFFF7EEE000000000ULL;
108   x ^= (x >> 43);
109
110   return x;
111 }
112
113 /* generates a random number on [0,1]-real-interval */
114 double genrand64_real1(void) {
115   return (genrand64_int64() >> 11) * (1.0/9007199254740991.0);
116 }
117
118 /* generates a random number on [0,maxint]-integer-interval */
119 int genrand64_int1(unsigned int maxint) {
120   return (int)(genrand64_real1() * (double)maxint + 0.5);
121 }
122
123 /* begin code */
124 int main () {
125   /* fill binomial coefficients table */
126   for (i = 0; i < M + 1; i++) {           // fill each line iterativelly
127     for (j = 0; j < M + 1; j++) {         // using dynamic programming
128       if (j > i) bc[i][j] = 0.0;
129       else if ((j == 0) || (j == i)) bc[i][j] = 1.0;
130       else bc[i][j] = (j <= 1 + i / 2) ? bc[i - 1][j - 1] + bc[i -1][j]: bc[i][i-j];
131     }
132   }
133
134   do {
135
136     /* initialize with random seed */
137     init_genrand64((unsigned)time(NULL));
138
139     p = p + 0.001;                        // increase probability at each invocation
140     B = 0;
141     C = 0;
142     S = 0;
143     busy = 0;
144
145     /* until simulation is over */
```

```
146       do {
147         /* initialize variables */
148         A = 0.0, i = 0;
149         X1 = genrand64_real1();
150
151         /* compute IDF to find how many free stations transmitted */
152         do {
153           A = A + bc[M - B][i] * pow(p, (double)i) * pow((1.0 - p), (double)(M - B - i));
154
155           if (X1 < A) { break; }
156             i++;
157         } while (i <= (M - B));
158
159         /* initialize variables */
160         F = 0.0, j = 0;
161         X2 = genrand64_real1();
162
163         /* compute IDF to find how many busy stations transmitted */
164         do {
165           F = F + bc[B][j] * pow(r, (double)j) * pow((1.0 - r), (double)(B - j));
166
167           if (X2 < F) { break; }
168             j++;
169         } while (j <= B);
170
171         /* fill simulation matrix: channel, boolean busy, destination, boolean success */
172         for (k = 0; k < i + j; k++) {
173           sim[k][0] = genrand64_int1(N);
174           sim[k][1] = 1;
175           /* choose any other station as destination */
176           do {
177             sim[k][2] = genrand64_int1(M - 1);
178           } while (sim[k][2] == k);
179           /* busy sources are statistically more than free */
180           sim[k][3] = 1;
181         }
182         /* fill simulation matrix, (boolean busy = 0) at random
183         until number of free stations is reached */
184         f = 0;
185         do {
186           k = genrand64_int1(i + j);
187           if (sim[k][1] == 1) {
188             f++;
189             sim[k][1] = 0;
190           }
191         } while (f < i);
192
193         /* evaluate simulation matrix */
194         for (k = 0; k < i + j; k++) {
195           for (f = k + 1; f < i + j; f++) {
196             /* find duplicates (packets on same channel or same destination) */
197             if ((sim[k][0] == sim[f][0]) || (sim[k][2] == sim[f][2])) {
198               /* mark as failed */
199               sim[k][3] = 0;
200               sim[f][3] = 0;
201             }
202           }
203         }
204
205         /* calculate successful transmissions */
206         ii = i; jj = j;
207
208         for (k = 0; k < i + j; k++) {
209           if (sim[k][3] == 0) {
210             if (sim[k][1] == 0) ii--;
211             else jj--;
212           }
213         }
214
215         /* define the outcome */
216         /* success, free stations transmitted */
217         S+= ii;
218         /* success, busy stations transmitted */
219         S+= jj;
220         B-= jj;
221         /* collisions occured, increase fail counter for busy stations */
```

```
222         B+= i - ii;
223
224         /* end of timeslot, proceed to next one */
225         C++;
226         busy+= B;
227
228       } while (C < R);                    // end of simulation cycle
229
230       /* printout simulated throughput, delay, mean busy stations */
231       printf("%.16f \t %.16f \t %.16f %.16f\n", p, (double)S / (double)R, 1.0 + (double)(busy) / (double)S,
          ↪   (double)busy / (double)R);
232
233     } while (p < 1.0);
234
235     return 0;
236   }
```

Γ′ Κώδικας Προσομοίωσης σε C, stage3

```c
/*****************************************************************************************************
* slottedAloha.c
* Computes simulated throughput, delay, mean busy stations of slotted ALOHA
* protocol, based on certain assumptions.
*
* To compile, run 'gcc slottedAloha.c -O3 -o slottedAloha -lm', on linux or
* other *nix command line.
*
* Politis Dimitrios, Mar 2017
*
* Uses parts from Mersenne Twister Random Number Generator, found at
* http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/C-LANG/mt19937-64.c
*
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*
*
* The source code can be found at
* https://github.com/dpolitis/Slotted-Aloha-Simulation
\*****************************************************************************************************/

/* header includes */
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>

/* defines */
#define NN 312
#define MM 156
#define MATRIX_A 0xB5026F5AA96619E9ULL
#define UM 0xFFFFFFFF80000000ULL          // most significant 33 bits
#define LM 0x7FFFFFFFULL                  // least significant 31 bits

#define M 100                             // number of stations
#define N 50                              // number of channels
#define FB 3                              // receiving buffer length
#define R 100000000                       // number of time-slots per invocation

/* global variable definition */
int i, ii, j, jj;                         // counters of transmissions per invocation
int k, f, g;                              // general purpose counters
int B;                                    // counter of backlogged stations
int C;                                    // total time-slot counter
int S;                                    // total succesful time-slot counter
int busy;                                 // total busy stations in each invocation
int sim[M+1][7];                          // simulation array
double bc[M+1][M+1];                      // binomial coefficients array

double A, F;                              // IDF functions
double X1, X2;                            // random numbers [0,1]
double p = 0.0;                           // probability of free stations
double r = 0.3;                           // probability of busy stations

/* The array for the state vector */
static unsigned long long mt[NN];

/* mti==NN+1 means mt[NN] is not initialized */
static int mti=NN+1;
```

```
70
71  /* initializes mt[NN] with a seed */
72  void init_genrand64(unsigned long long seed) {
73   mt[0] = seed;
74   for (mti=1; mti<NN; mti++)
75    mt[mti] =  (6364136223846793005ULL * (mt[mti-1] ^ (mt[mti-1] >> 62)) + mti);
76  }
77
78  /* generates a random number on [0, 2^64-1]-interval */
79  unsigned long long genrand64_int64(void) {
80    int i;
81    unsigned long long x;
82    static unsigned long long mag01[2]={0ULL, MATRIX_A};
83
84    if (mti >= NN) {                        // generate NN words at one time
85
86      /* if init_genrand64() has not been called, a default initial seed is used */
87      if (mti == NN+1)
88      init_genrand64(5489ULL);
89
90      for (i=0;i<NN-MM;i++) {
91        x = (mt[i]&UM)|(mt[i+1]&LM);
92        mt[i] = mt[i+MM] ^ (x>>1) ^ mag01[(int)(x&1ULL)];
93      }
94      for (;i<NN-1;i++) {
95        x = (mt[i]&UM)|(mt[i+1]&LM);
96        mt[i] = mt[i+(MM-NN)] ^ (x>>1) ^ mag01[(int)(x&1ULL)];
97      }
98      x = (mt[NN-1]&UM)|(mt[0]&LM);
99      mt[NN-1] = mt[MM-1] ^ (x>>1) ^ mag01[(int)(x&1ULL)];
100
101     mti = 0;
102   }
103
104   x = mt[mti++];
105
106   x ^= (x >> 29) & 0x5555555555555555ULL;
107   x ^= (x << 17) & 0x71D67FFFEDA60000ULL;
108   x ^= (x << 37) & 0xFFF7EEE000000000ULL;
109   x ^= (x >> 43);
110
111   return x;
112 }
113
114 /* generates a random number on [0,1]-real-interval */
115 double genrand64_real1(void) {
116   return (genrand64_int64() >> 11) * (1.0/9007199254740991.0);
117 }
118
119 /* generates a random number on [0,maxint]-integer-interval */
120 int genrand64_int1(unsigned int maxint) {
121   return (int)(genrand64_real1() * (double)maxint + 0.5);
122 }
123
124 /* begin code */
125 int main () {
126   /* fill binomial coefficients table */
127   for (i = 0; i < M + 1; i++) {           // fill each line iterativelly
128     for (j = 0; j < M + 1; j++) {         // using dynamic programming
129       if (j > i) { bc[i][j] = 0.0; }
130       else if ((j == 0) || (j == i)) { bc[i][j] = 1.0; }
131       else bc[i][j] = (j <= 1 + i / 2) ? bc[i - 1][j - 1] + bc[i -1][j]: bc[i][i-j];
132     }
133   }
134
135   do {
136     /* initialize with random seed */
137     init_genrand64((unsigned)time(NULL));
138
139     p+= 0.001;                            // increase probability at each invocation
140     B = 0;
141     C = 0;
142     S = 0;
143     busy = 0;
144
145     /* until simulation cycle is over */
```

```
146       do {
147         /* initialize variables */
148         A = 0.0, i = 0;
149         X1 = genrand64_real1();
150
151         /* compute IDF to find how many free stations transmitted */
152         do {
153           A = A + bc[M - B][i] * pow(p, (double)i) * pow((1.0 - p), (double)(M - B - i));
154
155           if (X1 < A) { break; }
156             i++;
157         } while (i <= (M - B));
158
159         /* initialize variables */
160         F = 0.0, j = 0;
161         X2 = genrand64_real1();
162
163         /* compute IDF to find how many busy stations transmitted */
164         do {
165           F = F + bc[B][j] * pow(r, (double)j) * pow((1.0 - r), (double)(B - j));
166
167           if (X2 < F) { break; }
168             j++;
169         } while (j <= B);
170
171         /* fill simulation matrix (k is arbitrary source): channel, boolean busy, destination,
172         boolean success, total packet counter on destination, successful packet counter on
173         destination (initially all packets are considered successful), boolean destruction on channels */
174         for (k = 0; k < i + j; k++) {
175           sim[k][0] = genrand64_int1(N);
176           sim[k][1] = 1;
177           /* choose any other station as destination */
178           do {
179             sim[k][2] = genrand64_int1(M - 1);
180           } while (sim[k][2] == k);
181           /* busy sources are statistically more than free */
182           sim[k][3] = 1;
183           sim[k][4] = 0;
184           sim[k][5] = 0;
185           sim[k][6] = 0;
186         }
187         /* fill simulation matrix, (boolean busy = 0) at random
188         until number of free stations is reached */
189         f = 0;
190         do {
191           k = genrand64_int1(i + j);
192           if (sim[k][1] == 1) {
193             f++;
194             sim[k][1] = 0;
195           }
196         } while (f < i);
197
198         /* evaluate simulation matrix */
199         for (k = 0; k < i + j; k++) {
200           for (f = k + 1; f < i + j; f++) {
201             /* find duplicates (packets on same channel) */
202             if (sim[k][0] == sim[f][0]) {
203               /* mark as failed, because of same channel use */
204               sim[k][3] = 0;
205               sim[k][6] = 1;
206               sim[f][3] = 0;
207               sim[f][6] = 1;
208             }
209           }
210         }
211
212         for (k = 0; k < M; k++) {
213           /* check for same destination k */
214           for (f = 0; f < i + j; f++) {
215             if ((sim[f][2] == k) && (sim[f][6] != 1)) {
216             /* increase receiving buffer packet counter on all instances */
217               for (g = 0; g < i + j; g++) {
218                 if ((sim[g][2] == k) && (sim[g][3] != 0)) { sim[g][4]++; }
219               }
220             }
221           }
```

```
222         }
223
224         /* calculate successful transmissions */
225         ii = i; jj = j;
226
227         /* check for buffer overflow */
228         for (k = 0; k < i + j; k++) {
229           /* mark as failed, because of receiving buffer overflow, if not failed already */
230           if ((sim[k][4] > FB) && (sim[k][6] != 1)) {
231             /* throw dice to say if package accepted */
232             if (sim[k][5] >= FB) { sim[k][3] = 0; }
233             else { sim[k][3] = genrand64_int1(1); }
234           }
235           /* package accepted */
236           if (sim[k][3] == 1) {
237             for (g = 0; g < i + j; g++) {
238               if (sim[g][2] == sim[k][2]) { sim[g][5]++; }
239             }
240           }
241         }
242
243         /* check for buffer underflow */
244         for (k = 0; k < i + j; k++) {
245           if ((sim[k][3] == 0) && (sim[k][6] != 1) && (sim[k][4] > FB) && (sim[k][5] < FB)) {
246             /* accept package that was randomly rejected at previous step */
247             sim[k][3] = 1;
248             for (g = 0; g < i + j; g++) {
249               if (sim[g][2] == sim[k][2]) { sim[g][5]++; }
250             }
251           }
252         }
253
254         for (k = 0; k < i + j; k++) {
255           /* check for failed transmissions */
256           if (sim[k][3] == 0) {
257             if (sim[k][1] == 0) { ii--; }
258             else { jj--; }
259           }
260         }
261
262         /* define the outcome */
263         /* success, free stations transmitted */
264         S+= ii;
265         /* success, busy stations transmitted */
266         S+= jj;
267         B-= jj;
268         /* collisions occured */
269         B+= i - ii;
270
271         /* end of timeslot, proceed to next one */
272         C++;
273         busy+= B;
274
275       } while (C < R);                      // end of simulation cycle
276
277       /* printout simulated throughput, calculated delay, mean busy stations */
278       printf("%.16f \t %.16f \t %.16f \t %.16f\n", p, (double)S / (double)R, 1.0 + (double)(busy) /
          ↪   (double)S, (double)busy / (double)R);
279
280     } while (p < 1.0);
281
282     return 0;
283   }
```

Δ΄ Κώδικας Αποτύπωσης Γραφημάτων σε Bash

```
#!/bin/bash

# This script acts as a wrapper to the simulation executable

##### Constants
M1=50; M2=70; M3=100;
N1=20; N2=30; N3=50;
F1=1; F2=2; F3=3;

##### Functions
function plot_results {

  until [ $(pidof slottedAloha_stage3.out | wc -l) -eq 0 ]; do
    sleep 10
  done

  for M in $M1 $M2 $M3; do
    gnuplot -p -e "set xlabel 'probability of retransmission p (busy stations)'; set ylabel 'throughput S';\
    set xrange [0:1]; set yrange [0:]; set grid x y2; set key Left left top box\
    title 'throughput'; unset ytics; set y2tics; plot\
    'total_M${M}_N${N1}_FB${F1}_stage3.out' using 1:2 with lines title '$M st, $N1 ch, $F1 buf' lw 2 smooth bezier,\
    'total_M${M}_N${N1}_FB${F2}_stage3.out' using 1:2 with lines title '$M st, $N1 ch, $F2 buf' lw 2 smooth bezier,\
    'total_M${M}_N${N1}_FB${F3}_stage3.out' using 1:2 with lines title '$M st, $N1 ch, $F3 buf' lw 2 smooth bezier,\

    'total_M${M}_N${N2}_FB${F1}_stage3.out' using 1:2 with lines title '$M st, $N2 ch, $F1 buf' lw 2 smooth bezier,\
    'total_M${M}_N${N2}_FB${F2}_stage3.out' using 1:2 with lines title '$M st, $N2 ch, $F2 buf' lw 2 smooth bezier,\
    'total_M${M}_N${N2}_FB${F3}_stage3.out' using 1:2 with lines title '$M st, $N2 ch, $F3 buf' lw 2 smooth bezier,\

    'total_M${M}_N${N3}_FB${F1}_stage3.out' using 1:2 with lines title '$M st, $N3 ch, $F1 buf' lw 2 smooth bezier,\
    'total_M${M}_N${N3}_FB${F2}_stage3.out' using 1:2 with lines title '$M st, $N3 ch, $F2 buf' lw 2 smooth bezier,\
    'total_M${M}_N${N3}_FB${F3}_stage3.out' using 1:2 with lines title '$M st, $N3 ch, $F3 buf' lw 2 smooth bezier"

    gnuplot -p -e "set xlabel 'probability of retransmission p (busy stations)'; set ylabel 'delay D';\
    set xrange [0:1]; set yrange [0:]; set grid x y2; set key Left left top box\
    title 'delay'; unset ytics; set y2tics; plot\
    'total_M${M}_N${N1}_FB${F1}_stage3.out' using 1:3 with lines title '$M st, $N1 ch, $F1 buf' lw 2 smooth bezier,\
    'total_M${M}_N${N1}_FB${F2}_stage3.out' using 1:3 with lines title '$M st, $N1 ch, $F2 buf' lw 2 smooth bezier,\
    'total_M${M}_N${N1}_FB${F3}_stage3.out' using 1:3 with lines title '$M st, $N1 ch, $F3 buf' lw 2 smooth bezier,\

    'total_M${M}_N${N2}_FB${F1}_stage3.out' using 1:3 with lines title '$M st, $N2 ch, $F1 buf' lw 2 smooth bezier,\
    'total_M${M}_N${N2}_FB${F2}_stage3.out' using 1:3 with lines title '$M st, $N2 ch, $F2 buf' lw 2 smooth bezier,\
    'total_M${M}_N${N2}_FB${F3}_stage3.out' using 1:3 with lines title '$M st, $N2 ch, $F3 buf' lw 2 smooth bezier,\

    'total_M${M}_N${N3}_FB${F1}_stage3.out' using 1:3 with lines title '$M st, $N3 ch, $F1 buf' lw 2 smooth bezier,\
    'total_M${M}_N${N3}_FB${F2}_stage3.out' using 1:3 with lines title '$M st, $N3 ch, $F2 buf' lw 2 smooth bezier,\
    'total_M${M}_N${N3}_FB${F3}_stage3.out' using 1:3 with lines title '$M st, $N3 ch, $F3 buf' lw 2 smooth bezier"

```

```
48      gnuplot -p -e "set xlabel 'probability of retransmission p (busy stations)'; set ylabel 'mean busy
        ↪  stations';\
49      set xrange [0:1]; set yrange [0:]; set grid x y2; set key Left right bottom box\
50      title 'mean busy stations'; unset ytics; set y2tics; plot\
51      'total_M${M}_N${N1}_FB${F1}_stage3.out' using 1:4 with lines title '$M st, $N1 ch, $F1 buf' lw 2
    ↪  smooth bezier,\
52      'total_M${M}_N${N1}_FB${F2}_stage3.out' using 1:4 with lines title '$M st, $N1 ch, $F2 buf' lw 2
    ↪  smooth bezier,\
53      'total_M${M}_N${N1}_FB${F3}_stage3.out' using 1:4 with lines title '$M st, $N1 ch, $F3 buf' lw 2
    ↪  smooth bezier,\
54
55      'total_M${M}_N${N2}_FB${F1}_stage3.out' using 1:4 with lines title '$M st, $N2 ch, $F1 buf' lw 2
    ↪  smooth bezier,\
56      'total_M${M}_N${N2}_FB${F2}_stage3.out' using 1:4 with lines title '$M st, $N2 ch, $F2 buf' lw 2
    ↪  smooth bezier,\
57      'total_M${M}_N${N2}_FB${F3}_stage3.out' using 1:4 with lines title '$M st, $N2 ch, $F3 buf' lw 2
    ↪  smooth bezier,\
58
59      'total_M${M}_N${N3}_FB${F1}_stage3.out' using 1:4 with lines title '$M st, $N3 ch, $F1 buf' lw 2
    ↪  smooth bezier,\
60      'total_M${M}_N${N3}_FB${F2}_stage3.out' using 1:4 with lines title '$M st, $N3 ch, $F2 buf' lw 2
    ↪  smooth bezier,\
61      'total_M${M}_N${N3}_FB${F3}_stage3.out' using 1:4 with lines title '$M st, $N3 ch, $F3 buf' lw 2
    ↪  smooth bezier"
62
63      gnuplot -p -e "set xlabel 'throughput S'; set ylabel 'delay D';\
64      set xrange [0:]; set yrange [0:]; set grid x y2; set key Left right top box\
65      title 'delay / throughput'; unset ytics; set y2tics; plot\
66      'total_M${M}_N${N1}_FB${F1}_stage3.out' using 2:3 with lines title '$M st, $N1 ch, $F1 buf' lw 2
    ↪  smooth bezier,\
67      'total_M${M}_N${N1}_FB${F2}_stage3.out' using 2:3 with lines title '$M st, $N1 ch, $F2 buf' lw 2
    ↪  smooth bezier,\
68      'total_M${M}_N${N1}_FB${F3}_stage3.out' using 2:3 with lines title '$M st, $N1 ch, $F3 buf' lw 2
    ↪  smooth bezier,\
69
70      'total_M${M}_N${N2}_FB${F1}_stage3.out' using 2:3 with lines title '$M st, $N2 ch, $F1 buf' lw 2
    ↪  smooth bezier,\
71      'total_M${M}_N${N2}_FB${F2}_stage3.out' using 2:3 with lines title '$M st, $N2 ch, $F2 buf' lw 2
    ↪  smooth bezier,\
72      'total_M${M}_N${N2}_FB${F3}_stage3.out' using 2:3 with lines title '$M st, $N2 ch, $F3 buf' lw 2
    ↪  smooth bezier,\
73
74      'total_M${M}_N${N3}_FB${F1}_stage3.out' using 2:3 with lines title '$M st, $N3 ch, $F1 buf' lw 2
    ↪  smooth bezier,\
75      'total_M${M}_N${N3}_FB${F2}_stage3.out' using 2:3 with lines title '$M st, $N3 ch, $F2 buf' lw 2
    ↪  smooth bezier,\
76      'total_M${M}_N${N3}_FB${F3}_stage3.out' using 2:3 with lines title '$M st, $N3 ch, $F3 buf' lw 2
    ↪  smooth bezier"
77    done
78  }  # end of plot_results
79
80  function usage {
81      echo "usage: $0 [[-r (runs simulation and plots results)] | [-p (plots results)]] | [-h (this
        ↪  message)]]"
82  }  # end of usage
83
84  function run_simulation {
85    for M in $M1 $M2 $M3; do
86      sed -i -- "s/define M [0-9][0-9]*/define M ${M}/g" ./slottedAloha_stage3.c
87
88      for N in $N1 $N2 $N3; do
89        sed -i -- "s/define N [0-9][0-9]*/define N ${N}/g" ./slottedAloha_stage3.c
90
91        for F in $F1 $F2 $F3; do
92          sed -i -- "s/define FB [0-9]*/define FB ${F}/g" ./slottedAloha_stage3.c
93
94          gcc slottedAloha_stage3.c -O3 -lm -o slottedAloha_stage3.out
95          ./slottedAloha_stage3.out > total_M${M}_N${N}_FB${F}_stage3.out&
96        done
97      done
98    done
99  }  # end of run_simulation
100
101 ##### Main
102 case $1 in
103   -r | --run )  run_simulation
```

```
104                 plot_results
105                 exit
106                 ;;
107     -p | --plot ) plot_results
108                 exit
109                 ;;
110     -h | --help ) usage
111                 exit
112                 ;;
113     * )         usage
114                 exit 1
115   esac
```

Βιβλιογραφία

[1] *Alohanet*, Mar 2017. https://en.wikipedia.org/wiki/ALOHAnet.

[2] *Linear congruential generator*, Mar 2017. https://en.wikipedia.org/wiki/Linear_congruential_generator.

[3] Al-Naami, R.: *Queueing analysis of slotted aloha with finite buffer capacity.* Proceedings of GLOBECOM '93. IEEE Global Telecommunications Conference.

[4] Bai, Xiaofeng και Abdallah Shami: *Modeling self-similar traffic for network simulation.* CoRR, abs/1308.3842, 2013.

[5] Banks, Jerry: *Handbook of simulation: principles, methodology, advances, applications, and practice.* John Wiley & Sons Inc, 2007.

[6] Chandrasekaran, Balakrishnan: *Survey of network traffic models.* http://www.cse.wustl.edu/~jain/cse567-06/ftp/traffic_models3/index.html.

[7] Forbes, Catherine και Merran Evans: *Statistical distributions.* Wiley, 2010.

[8] Ghez, S., S. Verdu, και S.c. Schwartz: *Stability properties of slotted aloha with multipacket reception capability.* IEEE Transactions on Automatic Control, 33(7):640–649, 1988.

[9] Giambene, Giovanni: *Queuing Theory and Telecommunications Networks and Applications.* Springer US, 2014.

[10] Jin, Youngmi και G. Kesidis: *Equilibria of a noncooperative game for heterogeneous users of an aloha network.* IEEE Communications Letters, 6(7):282–284, 2002.

[11] Knuth, D. E.: *The art of computer programming Volume 2: seminumerical algorithms.* Addison-Wesley, 1998.

[12] L'ecuyer, Pierre: *Tables of linear congruential generators of different sizes and good lattice structure.* Mathematics of Computation, 68(225):249–261, Jan 1999.

[13] Ma, R.t.b., V. Misra, και D. Rubenstein: *An analysis of generalized slotted-aloha protocols.* IEEE/ACM Transactions on Networking, 17(3):936–949, 2009.

[14] P. Baziana, G. Fragkouli, E. Sykas: *Analytical receiver collisions performance modeling of a multi-channel network.* Proceedings of 2017 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW), (8), Feb 2017.

[15] Patel, A: *Performance analysis of multiple access protocols.* Computer Communications, 10(1):39–40, 1987.

[16] Pountourakis, Ioannis E.: *Throughput evaluation of multichannel slotted aloha-type protocols with receiver collisions.* Telecommunication Systems, 5(2):413–419, 1996.

[17] Rivest, R.: *Network control by bayesian broadcast.* IEEE Transactions on Information Theory, 33(3):323–328, 1987.

[18] Saunders, Peter T.: *An introduction to catastrophe theory.* Cambridge Univ. Press, 2003.

[19] Tanenbaum, Andrew S. και David Wetherall: *Computer networks.* Pearson, 2014.