ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΤΗΛΕΠΙΚΟΙΚΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Ανάπτυξη και υλοποίηση τεχνικών παροχής υπηρεσιών σε ασύρματα δίκτυα με τον MadWifi driver

Μακρής Φ. Νικόλαος

Επιβλέποντες καθηγητές:

Κουτσόπουλος Ιορδάνης, Επίκουρος καθηγητής

Τασσιούλας Λέανδρος, Καθηγητής

Βόλος 2011

Ανάπτυξη και υλοποίηση τεχνικών παροχής υπηρεσιών σε ασύρματα δίκτυα με τον driver MadWifi

Μακρής Φ. Νικόλαος

Επιβλέποντες καθηγητές:

Κουτσόπουλος Ιορδάνης, Επίκουρος καθηγητής

Τασσιούλας Λέανδρος, Καθηγητής

Εγκρίθηκε από την εξεταστική επιτροπή την ……. Μαρτίου 2011

(Υπογραφή) *(Υπογραφή)*

…… ……

Βόλος 2011

Copyright Μακρής Νικόλαος, 2011 ©
Με επιφύλαξη παντός δικαιώματος. All rights reserved.
Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικό ή ερευνητικής φύσης.

Ευχαριστίες:

Η παρούσα διπλωματική εργασία δε θα μπορούσε ποτέ να ολοκληρωθεί χωρίς την πολύτιμη βοήθεια ανθρώπων που σ' όλη τη διάρκεια αυτών των μηνών ενασχόλησης και έρευνας μου προσέφεραν απλόχερα τις γνώσεις και τη βοήθειά τους και γι 'αυτό νιώθω την ανάγκη να τους ευχαριστήσω ειλικρινά γι όλη την υποστήριξη που μου παρείχαν.

Ευχαριστώ θερμά τους καθηγητές μου κύριο Αθανάσιο Κοράκη και κύριο Ιορδάνη Κουτσόπουλο, οι οποίοι με τις υποδείξεις, τις ιδέες και την καθοδήγησή τους άνοιξαν το μεγαλύτερο μέρος του δρόμου που οδήγησε εδώ στην περάτωση της εργασίας αυτής.

Ευχαριστώ πολύ τον επιβλέποντα καθηγητή μου κύριο Λέανδρο Τασιούλα που δέχτηκε να αναλάβει την επίβλεψη της διπλωματικής μου.

Ευχαριστώ επίσης τους Στράτο Κερανίδη, Απόστολο Αποστολάρα, Νίκο Γιαλελή και την υπόλοιπη ομάδα του Nitlab, για την υποστήριξη που έλαβα για το Testbed, καθώς και τον συμφοιτητή μου Νικόλα Μπαράτι για την βοήθεια και τις ιδέες του στην όλη προσπάθειά μου.

Μακρής Νικόλαος

Βόλος, 2011

Περιεχόμενα

Περίληψη:

Σκοπός της εργασίας αυτής είναι η μελέτη και η ανάλυση του μηχανισμού πρόσβασης στο μέσο (MAC) που χρησιμοποιείται από τα πρωτόκολλα IEEE 802.11a/b/g, και η ανάπτυξη νέων τεχνικών για την παροχή υπηρεσιών στους χρήστες του ασύρματου δικτύου. Κύριος στόχος μας είναι να μεγιστοποιήσουμε το συνολικό throughput μεταδιδόμενο πάνω από το δίκτυο, περιορίζοντας τη μετάδοση των σταθμών με μικρό φόρτο ή χαμηλό ρυθμό μετάδοσης. Αυτά τα καταφέρνουμε απλά αλλάζοντας την παράμετρο του Contention Window (CW) που χρησιμοποιείται στο μηχανισμό πρόσβασης στο μέσο. Οι μετρήσεις των νέων τεχνικών που παρουσιάζονται έγιναν σε πραγματικά δίκτυα, προσπαθώντας να προσομοιάσουμε όσο το δυνατόν καλύτερα τις συνθήκες του περιβάλλοντος που επηρεάζουν το ασύρματο μέσο.

Αρχικά γίνεται μια εισαγωγή στον τρόπο λειτουργίας των ασύρματων LAN δικτύων. Στη συνέχεια παρουσιάζεται συνοπτικά ο μηχανισμός πρόσβασης στο μέσο που χρησιμοποιείται από τα πρωτόκολλα IEEE 802.11 a/b/g, όπως επίσης και του πρωτοκόλλου IEEE 802.11e, που χρησιμοποιείται για παροχή υπηρεσιών. Έπειτα γίνεται μια μικρή παρουσίαση της πλατφόρμας ανοιχτού λογισμικού που έγιναν τα πειράματα, του MadWiFi driver.

Στη συνέχεια παρουσιάζονται κάποιες προηγούμενες έρευνες πάνω στο αντικείμενο και πως εμπνευστήκαμε από αυτές για την ανάπτυξη των δικών μας τεχνικών. Ακολουθούν μετά η ανάπτυξη των δύο πρώτων τεχνικών μας, και οι μετρήσεις γι αυτές που δείχνουν πως το συνολικό throughput μεταδιδόμενο πάνω από το δίκτυο αυξάνεται.

Τέλος, παρουσιάζεται μια εντελώς αυτόνομη τεχνική, όπου ξεκινώντας από τις προηγούμενες, και με κατάλληλη ανταλλαγή μηνυμάτων πάνω από το δίκτυο, μπορεί να προσφέρει ταυτόχρονα υψηλότερο throughput σε σχέση με το IEEE 802.11, αλλά ταυτόχρονα να είναι συνολικά πιο δίκαιο σε σχέση με τις προηγούμενες τεχνικές.

Abstract

Main purpose of this paper is to study and analyze the Medium Access Control (MAC) policies used by the IEEE 802.11 a/b/g protocols, and to design and develop new schemas able to offer QoS control to the users of the WLAN. Main target is to maximize the total throughput transmitted over the network, by suppressing transmission of the stations with low load or rate. This is done by changing the Contention Window (CW) value, which is used by the MAC policy in IEEE 802.11 protocol. The experiments made to test the new schemas are conducted in real-life networks, trying to simulate the environment conditions that may interfere with the medium.

In the beginning, we introduce the way WLANs work, and present the MAC policy used by IEEE 802.11 a/b/g, as well as IEEE 802.11e, the standard used for QoS. Moreover, we present the open-source software driver, where our new schemas where implemented, the MadWiFi wireless device driver.

In addition to these, former papers with similar work are presented, and the way these influenced the designing of our schemas. We present the first two techniques used for QoS, and the results from the experiments conducted show that total throughput over the network is raised.

Finally, a new schema is introduced which, starting from the former ones and appropriate exchange of messages over the network, can offer higher throughput compared to IEEE 802.11, and be overall more fair than the former techniques presented.

Κεφάλαιο 1

1.1 Εισαγωγή

Τα τελευταία χρόνια, τα ασύρματα δίκτυα γνωρίζουν τέτοια επιτυχία εφάμιλλη του Internet. Αποτέλεσμα αυτών, ασύρματες συσκευές χρησιμοποιούνται παντού για να παρέχουν φθηνά, κινητά και εύκολα στην υλοποίηση τους δίκτυα, με ή χωρίς πρόσβαση σε κάποιο ενσύρματο δίκτυο όπως πχ. το Internet. Τα ασύρματα δίκτυα μπορούν να χωριστούν σε δύο μεγάλες κατηγορίες: Τα κεντρικοποιημένα (δουλεύουν με τη βοήθεια συντονιστή - centralized) και τα κατανεμημένα (χωρίς συντονιστή - Ad-Hoc).

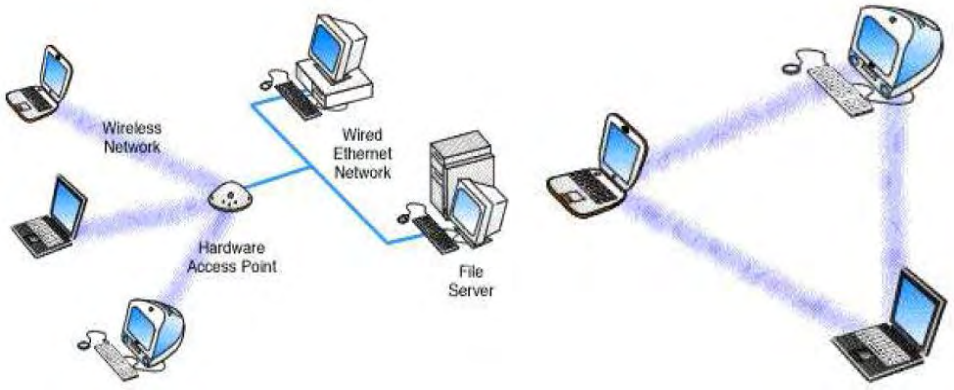


Figure 1: Example of an Infrastructure WLAN on the left and an Ad-hoc on the right

Τα ασύρματα δίκτυα που βασίζονται στο πρωτόκολλο IEEE 802.11 είναι αυτά που έχουν γνωρίσει την μεγαλύτερη διάδοση. Εξαιτίας όμως της φύσης του μέσου (αέρας), πρέπει η πρόσβαση στο μέσο να γίνεται με συντονισμένο τρόπο, ώστε να αποφεύγονται, όσο είναι δυνατό, οι συγκρούσεις (collisions). Στο πρωτόκολλο IEEE 802.11 οι βασικοί μηχανισμοί για την πρόσβαση στο μέσο είναι οι Distributed Coordination Function (DCF) και Point Coordination Function (PCF), οι οποίες αναλύονται εκτενώς παρακάτω.

1.2 IEEE 802.11 Distributed Coordination Function

Η **Distributed coordination function** (**DCF**) είναι η βασική τεχνική πρόσβασης στο μέσο για τα ασύρματα δίκτυα βασισμένα στο πρότυπο IEEE 802.11. Η DCF χρησιμοποιεί το πρωτόκολλο CSMA/CA με τον Binary exponential backoff αλγόριθμο. Η λειτουργία της είναι η εξής: Ένας σταθμός που θέλει να μεταδώσει πρέπει να παρακολουθήσει για την κατάσταση του καναλιού για χρόνο DIFS. Αν το κανάλι είναι κατειλημμένο από την μετάδοση κάποιου άλλου, τότε ο σταθμός αναστέλλει την μετάδοση του. Όμως σε ένα δίκτυο όπου πολλοί σταθμοί ανταγωνίζονται ο ένας τον άλλο για την πρόσβαση στο μέσο, αν πολλοί από αυτούς ακούσουν κάποιον να μεταδίδει και αναστείλουν την μετάδοση τους, θα προσπαθήσουν όλοι μαζί να πάρουν πρόσβαση στο μέσο όταν θα ελευθερωθεί το κανάλι. Σαν αποτέλεσμα αυτών, μπορεί να συμβούν συγκρούσεις (collisions). Για την αποφυγή τέτοιων συγκρούσεων, η DCF επιπλέον καθορίζει ένα τυχαίο χρόνο backoff, ο οποίος αναγκάζει ένα σταθμό να αναστείλει την πρόσβαση του στο μέσο για κάποια μικρή χρονική περίοδο. Κάθε σταθμός αναστέλλει την μετάδοση του για μια στιγμή στο μέλλον, και θα ακούσει το κανάλι ξανά λίγο πριν προσπαθήσει να στείλει. Σε κάθε σύγκρουση (collision), που εντοπίζεται από την έλλειψη επιβεβαίωσης (ACK) από τον προορισμό, ο σταθμός αυξάνει το όριο από το οποίο επιλέγεται ο χρόνος backoff, το οποίο ονομάζεται Contention Window (CW). Αυξάνοντας το CW μειώνει τον κίνδυνο περαιτέρω συγκρούσεων, υποθέτοντας ότι ο αριθμός των ανταγωνιστικών κόμβων για πρόσβαση στο μέσο είναι μεγάλος. Μετά από κάθε επιτυχή μετάδοση, ο σταθμός επαναφέρει το CW στο CW_{min} και ξαναπροσπαθεί να πάρει πρόσβαση στο μέσο ξανά με χαμηλή τιμή στο CW.

Η DCF έχει επίσης έναν πρόσθετο και κατ' επιλογήν μηχανισμό εικονικού carrier sensing που ανταλλάσει μικρά μηνύματα Request-to-send (RTS) και Clear-to-send (CTS) ανάμεσα στους σταθμούς που επικοινωνούν στα διαστήματα μεταξύ της αποστολής των frames των δεδομένων. Ένας σταθμός που επιθυμεί να στείλει δεδομένα ξεκινά την διαδικασία στέλνοντας μια αίτηση Request to Send (RTS). Ο κόμβος προορισμού

απαντά με ένα μήνυμα Clear To Send frame (CTS). Οποιοσδήποτε άλλος κόμβος παρέλαβε το RTS ή το CTS μήνυμα πρέπει να απέχει από το να στείλει δεδομένα για ένα δεδομένο χρονικό διάστημα (και έτσι επιλύεται και το hidden node πρόβλημα). Ο χρόνος για τον οποίο πρέπει να περιμένει ο σταθμός πριν προσπαθήσει να πάρει πρόσβαση στο μέσο περιλαμβάνεται στα RTS και CTS μηνύματα. Το πρωτόκολλο έχει σχεδιαστεί δεδομένου πως όλοι οι κόμβοι έχουν την ίδια εμβέλεια μετάδοσης.

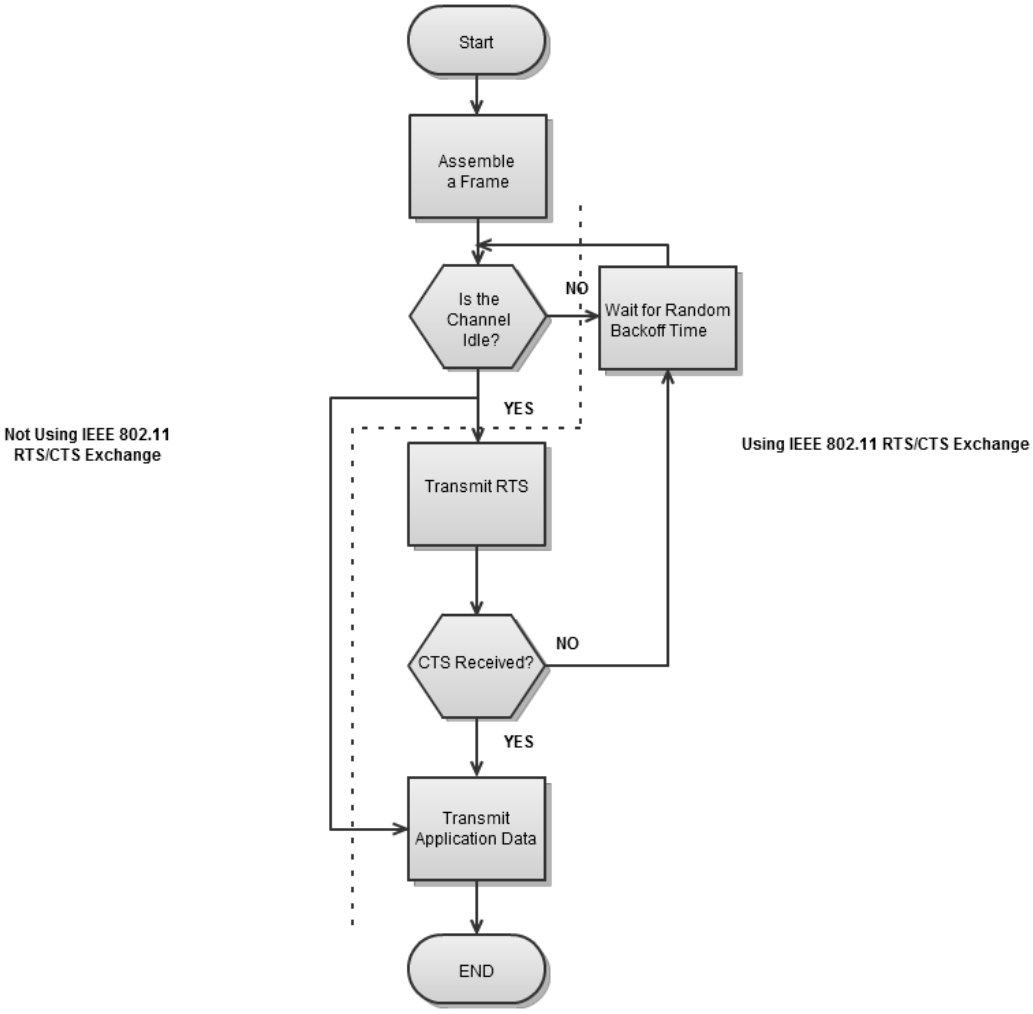


Figure 2: Schematic of CSMA/CA protocol

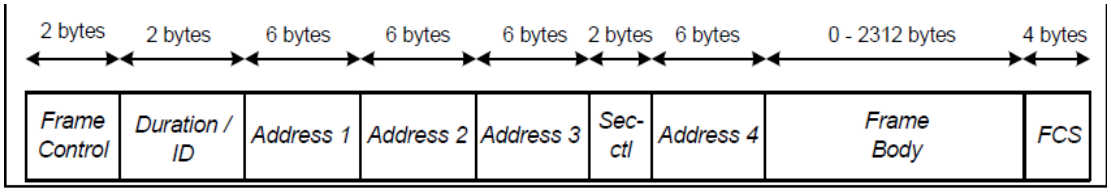


Figure 3: IEEE 802.11 standard frame

1.3 IEEE 802.11 Point Coordination Function

Η **Point Coordination Function (PCF)** υπάρχει στο Access Point (AP) και έχει ως σκοπό να συγχρονίσει την επικοινωνία στο δίκτυο. Η PCF τρέχει πάνω από την DCF και, αξιοποιώντας το AP, προσφέρει στα τερματικά όταν χρειάζεται πρόσβαση στο κοινό μέσο χωρίς ανταγωνισμό και συγκρούσεις. Το AP περιμένει για χρόνο PIFS αντί για DIFS για να πάρει πρόσβαση στο κανάλι. Ο χρόνος PIFS είναι μικρότερος από τον χρόνο DIFS και έτσι πάντα το AP έχει την μεγαλύτερη προτεραιότητα πρόσβασης στο μέσο. Το PCF ενεργοποιείται αυτόματα για συγκεκριμένα διαστήματα όταν το AP το κρίνει απαραίτητο ώστε, αν π.χ. πρόκειται να μεταδοθεί χρονικά κρίσιμη πληροφορία να εξασφαλιστεί ότι δε θα υπάρξουν συγκρούσεις για κάποιο διάστημα. Στην αρχή κάθε τέτοιας περιόδου χωρίς ανταγωνισμό το AP στέλνει σε όλους τους κόμβους ένα πλαίσιο συγχρονισμού (Beacon) και στη συνέχεια διαμοιράζει το χρόνο σε θυρίδες, και αναθέτει σε κάθε σταθμό μία θυρίδα κατά την οποία μόνο αυτός μπορεί να εκπέμψει ή να λάβει δεδομένα. Τα πλαίσια από έναν κόμβο Α σε έναν κόμβο Β μπορούν είτε να μεταδοθούν από τον Α στο AP (κατά τη θυρίδα του Α) και στη συνέχεια από τον AP στον Β (κατά τη θυρίδα του Β), είτε απευθείας από τον Α στον Β κατά τη θυρίδα του Α. Η έναρξη κάθε θυρίδας σηματοδοτείται από την αποστολή ενός πλαισίου ελέγχου Poll από τον AP στον κόμβο που του ανήκει η τρέχουσα θυρίδα.

1.4 Quality of Service (QoS) in IEEE 802.11

Παρότι απλά, τα πρωτόκολλα IEEE 802.11 (a/b/g) μπορούν να εξασφαλίσουν δικαιοσύνη στην πρόσβαση στο μέσο για όλους τους σταθμούς, μακροπρόθεσμα. Όμως η απόδοση τους δεν είναι πάντα καλή. Αυτό οφείλεται κυρίως στα σχεδιαστικά χαρακτηριστικά του πρωτοκόλλου MAC στο IEEE 802.11. Επομένως υπάρχει η ανάγκη για μια λεπτομερή ανάλυση του βασικού πρωτοκόλλου για πρόσβαση στο μέσο, έτσι ώστε να βελτιώσουμε την απόδοση του εφαρμόζοντας απλά αλλά αποδοτικά σχήματα, στην προϋπάρχουσα τεχνολογία.

Το πρωτόκολλο δεν προσπαθεί να εξασφαλίσει μέγιστο throughput στην προσπάθεια του να εξασφαλίσει δικαιοσύνη στην πρόσβαση στο μέσο. Δεδομένου της μεταβλητότητας του ασύρματου μέσου και της έλλειψης πόρων, είναι πρόκληση η προσπάθεια για να επιτύχουμε μέγιστο throughput, ώστε να πετύχουμε παροχή υπηρεσιών (QoS) στους χρήστες του μέσου, κάτι που είναι και το κύριο κομμάτι που θα μας απασχολήσει σε αυτή τη πτυχιακή. Το βασικό πρωτόκολλο που βελτιώνει το IEEE 802.11 και παρέχει QoS είναι το IEEE 802.11e.

1.4.1 IEEE 802.11e

Το 802.11e βελτιώνει την DCF και την PCF, μέσα από μια νέα λειτουργία συντονισμού: Την **hybrid coordination function (HCF)**. Στην HCF υπάρχουν δύο μέθοδοι πρόσβασης στο μέσο, παρόμοιες με αυτές ορισμένες στο 802.11 MAC: HCF Controlled Channel Access (HCCA) και Enhanced Distributed Channel Access (EDCA). Και οι δύο EDCA και HCCA ορίζουν τις Traffic Categories (TC- βλ. Figure 3). Για παράδειγμα, τα e-mail ανατίθενται σε μια κατηγορία χαμηλής προτεραιότητας, ενώ το Voice over Wireless LAN (VoWLAN) στην υψηλότερη.

Priority	UP (Same as 802.1D user priority)	802.1D designation	AC	Designation (informative)
Lowest ↓ Highest	1	BK	AC_BK	Background
	2	—	AC_BK	Background
	0	BE	AC_BE	Best Effort
	3	EE	AC_BE	Best Effort
	4	CL	AC_VI	Video
	5	VI	AC_VI	Video
	6	VO	AC_VO	Voice
	7	NC	AC_VO	Voice

Figure 4: IEEE 802.11e access categories

Με την EDCA, τα πακέτα υψηλής προτεραιότητας προς αποστολή έχουν υψηλότερη πιθανότητα να σταλούν από αυτά χαμηλότερης προτεραιότητας: Ένας σταθμός με υψηλότερης προτεραιότητας κίνηση περιμένει λιγότερο πριν στείλει το πακέτο του, κατά μέσο όρο, από ένα σταθμό με χαμηλότερης προτεραιότητας κίνηση. Αυτό επιτυγχάνεται χρησιμοποιώντας μικρότερο contention window (CW) και μικρότερο χρόνο arbitration inter-frame space (AIFS) για τα πακέτα υψηλότερης προτεραιότητας. Επιπλέον, η EDCA παρέχει ένα χρόνο πρόσβασης στο μέσο χωρίς

συναγωνισμό (contention free) το οποίο λέγεται Transmit Opportunity (TXOP). Μια TXOP είναι ένα χρονικό διάστημα στο οποίο ο σταθμός μπορεί να στείλει όσα πιο πολλά frames μπορεί (αρκεί η διάρκεια της μετάδοσης να μην ξεπερνά την μέγιστη διάρκεια της TXOP). Αν το frame είναι πολύ μεγάλο για να σταλεί σε ένα μόνο TXOP, θα πρέπει να σπάσει σε μικρότερα. Η χρήση των TXOPs μειώνει το πρόβλημα των σταθμών με χαμηλό ρυθμό μετάδοσης να παίρνουν πρόσβαση για υπερβολικό χρόνο στο κανάλι στο IEEE 802.11 DCF MAC.

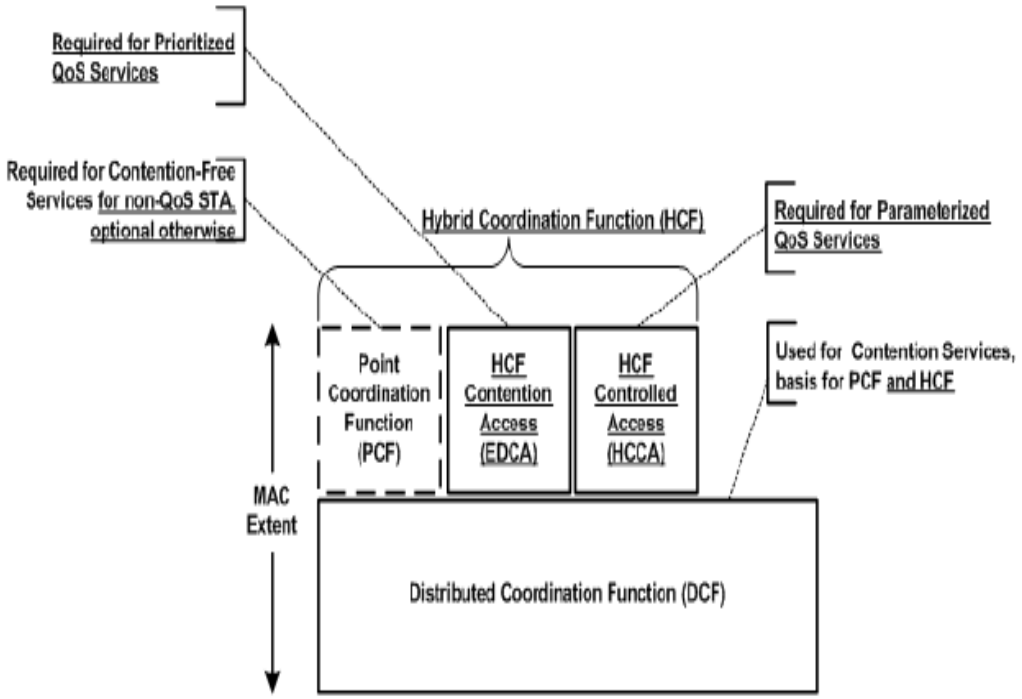


Figure 5: MAC level for 802.11e

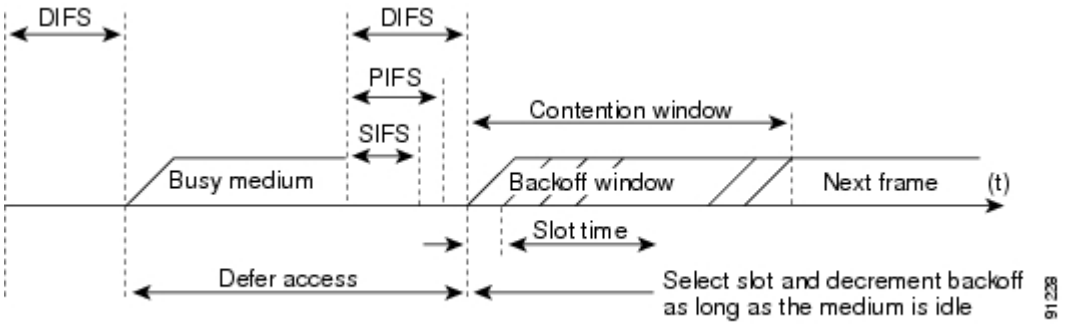


Figure 6: CSMA/CA Carrier Sense mechanism and all possible delay time

1.4.2 Ο ρυθμός μετάδοσης και το μέγεθος της ουράς ως παράμετροι για QoS

Ένα ακόμα σημαντικό χαρακτηριστικό που έχουν τα ασύρματα δίκτυα είναι πως μπορούν να υποστηρίξουν διαφορετικούς ρυθμούς μετάδοσης στο φυσικό επίπεδο. Για παράδειγμα, το IEEE 802.11g υποστηρίζει 14 ρυθμούς μετάδοσης, μεταξύ 1Mbps και 54Mbps. Ένας χαμηλός ρυθμός μετάδοσης χρησιμοποιεί λιγότερο πολύπλοκη και πιο αφαιρετική διαμόρφωση. Έτσι μπορεί και πετυχαίνει μικρότερο error rate με ένα tradeoff το χαμηλότερο throughput. Από την άλλη, ένας σταθμός με μεγαλύτερο ρυθμό μετάδοσης, έχει υψηλότερο throughput αλλά στην περίπτωση κακής ποιότητας ζεύξης υποφέρει από μεγαλύτερο error rate. Γι αυτούς τους λόγους, το πρωτόκολλο IEEE 802.11 χρησιμοποιεί κάποιους αλγόριθμους rate adaptation, έτσι ώστε να εξασφαλίσει ότι ο κάθε σταθμός μεταδίδει κάθε στιγμή με τον υψηλότερο δυνατό ρυθμό, εξισορροπώντας έτσι το error rate. (Παραδείγματα rate adaptation αλγορίθμων: AMRR, ONOE, Sample κ.ά.)

Μια άλλη παράμετρος, η οποία μπορεί να παίξει σημαντικό ρόλο στο συνολικό throughput που μετράμε, είναι το μέγεθος των ουρών σε ένα ασύρματο κόμβο και το ποσοστό του χρόνου που αυτές είναι κατειλημμένες. Η απόδοση σχετικά με το throughput ενός αλγορίθμου μπορεί να χαρακτηριστεί από το μεγαλύτερο σετ ρυθμών

άφιξης δεδομένων με τους οποίους ο αλγόριθμος μπορεί να κρατήσει τις ουρές στο δίκτυο σταθερές. Αφού πολλές εφαρμογές ασύρματων δικτύων έχουν αυστηρά όρια στο bandwidth, ο σχεδιασμός αλγορίθμων για την μεγιστοποίηση του throughput είναι μεγάλης σημασίας. Λαμβάνοντας υπ' όψη όλα τα παραπάνω, μια απλή βελτιστοποίηση στο IEEE 802.11 πρωτόκολλο είναι να αφήνουμε τους σταθμούς που έχουν την μεγαλύτερη κίνηση (μεγαλύτερη κίνηση ισοδυναμεί σε μεγαλύτερη ουρά) να μεταδίδει περισσότερα δεδομένα στο κανάλι, έτσι ώστε να προσπαθεί να κρατήσει την ουρά του όσο λιγότερο κατειλημμένη γίνεται.

1.4.3 Θέμα πτυχιακής

Λαμβάνοντας όλα τα παραπάνω υπόψη, στη παρούσα πτυχιακή θα προσπαθήσουμε χρησιμοποιώντας ένα συνδυασμό τους, να διαφοροποιήσουμε τον μηχανισμό πρόσβασης στο μέσο για όλους τους σταθμούς, προσπαθώντας να εξασφαλίσουμε το μέγιστο δυνατό throughput μεταδιδόμενο πάνω από το κανάλι. Η υλοποίηση του νέου μηχανισμού θα γίνει στον MadWiFi wireless device driver, ώστε να μπορέσουμε να κάνουμε μετρήσεις της νέας τεχνικής μας πάνω σε πραγματικά δίκτυα. Στη συνέχεια, παρουσιάζεται συνοπτικά ο MadWiFi driver. Μια πιο εκτενής παρουσίαση σχετικά με τις παραμέτρους που θα επηρεάσουμε για να αλλάξουμε το μηχανισμό πρόσβασης στο μέσο περιγράφεται στην ενότητα 2.2.

1.5 MadWiFi wireless device driver

Ο MadWiFi (*Multiband Atheros Driver for Wireless Fidelity)* είναι ένας Linux kernel device driver για Atheros-based Wireless LAN devices. Είναι ένας από τους πιο εξελιγμένους WLAN drivers διαθέσιμους για Linux σήμερα. Είναι σταθερός και έχει μια καθιερωμένη βάση χρηστών. Ο driver είναι open source αλλά εξαρτάται από το Hardware Abstraction Layer (HAL) (ιδιοκτησία της Atheros) που ήταν αρχικά διαθέσιμο

μόνο σε binary μορφή. Αυτό βέβαια μειώνει κάπως την δυνατότητα μεγαλύτερων παρεμβάσεων και πειραματισμών σ' όλο το εύρος των δυνατοτήτων αυτών των συσκευών αλλά ακόμα και στην παρούσα μορφή το Madwifi αποτελεί ένα πολύ χρήσιμο εργαλείο στην έρευνα στον χώρο των ασύρματων δικτύων ως μια ισχυρή και αποδεκτή εναλλακτική στα προγράμματα προσομοίωσης όπως Network Simulator ή ακόμα και Matlab.

Ο **Ath5k** είναι ένας νέος και «ανερχόμενος» driver και δεν εξαρτάται από το HAL. Θεωρείται πως θα αντικαταστήσει μακροχρόνια τον MadWifi και θα τον ξεπεράσει στο μέλλον.

OpenHAL

Ο παλιότερος madwifi driver αποτελούνταν από ένα BSD/GPL wrapper με ένα μη τροποποιήσιμο HAL (Hardware Abstraction Layer). Αυτό το HAL δεν ήταν binary firmware όπως στα Intel wireless chips, αλλά ένα κομμάτι κώδικα που έπρεπε να τρέχει στο Linux kernel. Αποτελούνταν από header files, για τα οποία δεν υπήρχε άδεια να τροποποιηθούν, και από pre-compiled object files. Η θέση της Atheros ήταν πως έπρεπε η Linux community να αποδεχθεί το sourceless HAL, αφού το chipset θα μπορούσε να ρυθμιστεί σε οποιαδήποτε συχνότητα, και έτσι να προκαλέσει RF παρεμβολές σε συστήματα που λειτουργούν σε αυτές τις συχνότητες.

Το binary HAL ήταν μη αποδεκτό στους Linux kernel developers, και σε αυτή τη μορφή ο driver της Atheros δεν θα μπορούσε ποτέ να έχει γίνει μέρος του official kernel. Κάποιοι OpenBSD developers, που αντιμετώπιζαν το ίδιο θέμα, έγραψαν το HAL ξανά από την αρχή και έφτιαξαν ένα νέο open source driver (ath5k), που στη συνέχεια συνεχίστηκε από την ομάδα του madwifi. Το νέο αυτό HAL, το **OpenHAL**, υπάρχει πλέον με τον Mad-WiFi αλλά παρότι ελεύθερο λογισμικό, είναι περιορισμένο πάλι σε κάποιες περιπτώσεις.

Κεφάλαιο 2

2.1 Σχετικές έρευνες

Πολλά MAC πρωτόκολλα έχουν σχεδιαστεί για να προσφέρουν υπηρεσίες QoS χρησιμοποιώντας τη δυναμική προσαρμογή του Contention Window στο CSMA/CA πρωτόκολλο. Το IEEE 802.11 χρησιμοποιεί ένα σταθερό παράθυρο CW για την πρόσβαση στο μέσο, πράγμα το οποίο δεν αντανακλά τον φόρτο του δικτύου. Στο IEEE 802.11e η EDCF [16] διαφοροποιεί την πρόσβαση στο μέσο για τις διάφορες κατηγορίες κίνησης χρησιμοποιώντας διαφορετικούς χρόνους για το contention window της καθεμιάς. Με εφαλτήριο αυτή την αλλαγή στο CW, πολλά πρωτόκολλα προσπαθούν να μεγιστοποιήσουν το συνολικό throughput στο δίκτυο, αλλάζοντας δυναμικά το CW που χρησιμοποιεί ο κάθε κόμβος.

Στο [4] εξετάζεται αρχικά η θεωρητική απόδοση του throughput και του delay σε περιπτώσεις κίνησης που αντιστοιχούν σε μέγιστο φόρτο στο δίκτυο. Τα τελικά αποτελέσματα δείχνουν την μεγάλη σημασία που έχει η διαδικασία του backoff στην μείωση των collisions. Όμως στο [3] εξετάζεται κατά πόσο είναι σωστός ο τρόπος με τον οποίο το contention window επαναφέρεται μετά από κάθε επιτυχημένη μετάδοση. Αναλύονται απλές συναρτήσεις που μειώνουν σταδιακά το CW αντί να το θέτουν στο CW_{min} και εξετάζεται η απόδοσή τους σε σχέση με το απλό IEEE 802.11. Μέσα από πειράματα αποδεικνύεται πως με τέτοιες βελτιστοποιήσεις μπορούμε να μειώσουμε τις μελλοντικές συγκρούσεις και επανεκπομπές πάνω από το δίκτυο.

Στο [5] εξετάζεται για ποιο μέσο μέγεθος CW μεγιστοποιείται το συνολικό throughput και αποδεικνύεται πως εξαιτίας των ρυθμίσεων στο δίκτυο, το πρωτόκολλο μπορεί να αποδίδει τελικά πολύ χαμηλότερα σε σχέση με το θεωρητικό όριο. Παρολαυτά με κατάλληλη ρύθμιση του αλγόριθμου backoff, το πρωτόκολλο μπορεί τελικά να αποδίδει κοντά στο θεωρητικό όριο.

Στο [2] προτείνεται ένας νέος και αποδοτικός μηχανισμός για την επιλογή του μεγέθους του backoff window χρησιμοποιώντας μια προσέγγιση με χρήση fuzzy reasoning, έχοντας παρατηρήσει την κίνηση στο δίκτυο και τον αριθμό των γειτόνων. Τελικά επιτυγχάνεται και δικαιοσύνη στην πρόσβαση στο μέσο.

Πολλά όμως από τα πρωτόκολλα που προτείνουν νέους αλγορίθμους προσαρμογής του CW χρησιμοποιούν επιπλέον το μέγεθος της ουράς του κάθε σταθμού, σαν μια μετρική για τον συνολικό φόρτο στο δίκτυο. Στο [1] υποθέτουμε αρχικά Ad Hoc δίκτυο με relay κόμβους. Η προσαρμογή στο CW γίνεται με βάση την κίνηση στους relay κόμβους, για την βελτιστοποίηση του thoughput end-to-end. Η κίνηση στο δίκτυο διαπιστώνεται εξετάζοντας περιοδικά την ουρά των relay κόμβων. Στο [8] προτείνεται μια βελτιστοποίηση στο IEEE 802.11e: Ένα σχήμα επανεκπομπής που προσπαθεί να μειώσει το delay και jitter των real-time πακέτων αλλάζοντας τα persistence factors (παράμετροι του πρωτοκόλλου για την διαδικασία backoff) δυναμικά, εξετάζοντας την ηλικία και τους χρόνους ζωής των πακέτων στις ουρές επανεκπομπής.

Στο [9] επιτυγχάνεται μια αύξηση του throughput κατά 20% απλά θέτοντας το CW κάθε σταθμού συναρτήσει του μεγέθους της ουράς του. Στο [7] προτείνεται ένας αλγόριθμος που προσαρμόζει το CW δυναμικά με τρόπο παρόμοιο με αυτό των TCP windows και παρουσιάζεται το σημαντικό κέρδος που μπορούμε να έχουμε σε throughput με αυτό τον τρόπο. Στο [14] παρέχονται υπηρεσίες QoS απλά αλλάζοντας το CW του κόμβου που μεταφέρει ροές δεδομένων που είναι delay-sensitive. Το CW προσαρμόζεται ανάλογα με την σχέση των ροών δεδομένων.

Τα [11], [12] και [13] προσπαθούν να βελτιώσουν το throughput υποθέτοντας ότι όλοι οι σταθμοί μεταδίδουν με τον ίδιο ρυθμό. Αυτό είναι ένα από τα θέματα που θα μας απασχολήσει σε αυτή τη πτυχιακή. Επιπλέον, στο [12] και [13] τίθεται το θέμα της χαμηλότερης από την αναμενόμενη απόδοση, εξαιτίας της βελτιστοποίησης τελικά συναρτήσεων κόστους (cost) και χρησιμότητας (utility) που μόνο εν μέρει περιγράφουν το πρόβλημα πρόσβασης στο μέσο.

2.2 Η δική μας έρευνα

Λαμβάνοντας υπόψη τα παραπάνω, σε αυτή τη πτυχιακή εξετάζουμε το θέμα της μεγιστοποίησης της απόδοσης του συνολικού throughput σε δίκτυα βασισμένα στο IEEE 802.11, αλλάζοντας δυναμικά το CW του σταθμού που θέλει να πάρει πρόσβαση στο μέσο. Υποθέτουμε δίκτυο με ένα Access Point (AP), και πολλαπλούς σταθμούς (STA) που ανταγωνίζονται ο ένας τον άλλο για το κοινό uplink κανάλι. Λαμβάνοντας υπόψη το μέγεθος της ουράς και το ρυθμό με τον οποίο μεταδίδει ο κάθε σταθμός, προτείνουμε ένα νέο σχήμα backoff, που προτείνει δυναμική αλλαγή στο CW που ο κάθε σταθμός θα χρησιμοποιήσει για τον αλγόριθμο του, προσπαθώντας να πάρει πρόσβαση στο μέσο. Για την υλοποίηση του μηχανισμού μας, χρησιμοποιήσαμε τον Mad-WiFi driver ανοιχτού κώδικα, σε αντίθεση με το σύνολο των προαναφερθέντων εργασιών που χρησιμοποιούν network simulators για την προσομοίωση των αποτελεσμάτων τους, και μέσα από πειράματα σε πραγματικά συστήματα, δείχνουμε πως ο αλγόριθμος μας αποδίδει καλύτερα του αρχικού IEEE 802.11.

Κεφάλαιο 3

3.1 Ανάπτυξη των αλγορίθμων

Ο αρχικός στόχος ήταν η ανάπτυξη παρόμοιων τεχνικών με αυτές που προτάθηκαν στη [16]. Τελικά φτάσαμε στην ανάπτυξη δύο τεχνικών, οι οποίες θα εξηγηθούν σε βάθος παρακάτω.

Ας σκεφτούμε αρχικά το Contention Window που χρησιμοποιείται από τον αλγόριθμο binary exponential backoff στο IEEE 802.11: Δεδομένου της αρχικής τιμής του contention window που χρησιμοποιείται στην πρώτη φάση του αλγορίθμου, w_1, η τιμή του παραθύρου διπλασιάζεται μετά από κάθε μη επιτυχημένη μετάδοση (συμβαίνει σύγκρουση), με βάση τον κανόνα: $w_j = \max(2 * w_{j-1}, W_{\max}),\ 1 < j \le m$, όπου W_{max} είναι η άνω οριακή τιμή για το contention window και m είναι ο αριθμός των φορών που θα διπλασιαστεί το contention window μετά από κάθε σύγκρουση.

Οι τεχνικές που αναπτύχθηκαν διαφέρουν σε σχέση με αυτές που αναφέρθηκαν στο ότι δεν χρησιμοποιούν το ίδιο CW για όλους τους σταθμούς, αντίθετα με το IEEE 802.11. Στο IEEE 802.11, αφού κάθε σταθμός χρησιμοποιεί το ίδιο CW στην διαδικασία του backoff, η πιθανότητα να συμβούν συγκρούσεις αυξάνεται καθώς μεγαλώνει ο αριθμός των σταθμών. Έτσι προτείνεται ένα νέο διαφορετικό σχήμα, όπου κάθε σταθμός αλλάζει το αρχικό του CW, συναρτήσει του μεγέθους της ουράς και τον ρυθμό μετάδοσης πάνω από το κανάλι (rate). Στόχος είναι να μεγιστοποιήσουμε το συνολικό throughput και να αφήνουμε τους σταθμούς με μεγαλύτερο μέγεθος στην ουρά και καλύτερη ποιότητα καναλιού να έχουν μεγαλύτερη πιθανότητα πρόσβασης στο μέσο. Παρότι φαίνεται ότι θα αλλάξουμε την δικαιοσύνη που υπάρχει στο IEEE 802.11 για να παρέχουμε υπηρεσίες QoS, μακροπρόθεσμα, οι κόμβοι που έχουν χαμηλή ποιότητα καναλιού θα είναι πιθανοί κόμβοι συμφόρησης, όπου για μεγαλύτερο μέγεθος ουράς θα έχουν μεγαλύτερη πιθανότητα πρόσβασης στο μέσο σε σχέση με αυτούς με καλύτερο κανάλι αλλά λιγότερη κίνηση.

3.1.1 1η τεχνική

Στόχος μας σε καθεμία από τις τεχνικές είναι τελικά η σχέση μεταξύ της στιγμιαίας τιμής του CW σε σχέση με το στιγμιαίο μέγεθος της ουράς (queue length), να είναι τελικά της υπερβολικής μορφής (convex συνάρτηση). Η πρώτη σκέψη ήταν αρχικά να εκφράσουμε το στιγμιαίο CW σε σχέση με το στιγμιαίο μέγεθος της ουράς σαν μια υπερβολή, και μετά να προσθέσουμε στην συνάρτηση μας την παράμετρο του ρυθμού μετάδοσης (rate) που εξαρτάται από τη φυσική ζεύξη μεταξύ των σταθμών. Η πιο απλή συνάρτηση που θα μπορούσαμε να χρησιμοποιήσουμε είναι η

$$f(x) = \frac{a}{x}\ ,\ x > 0 \qquad (3.1)$$

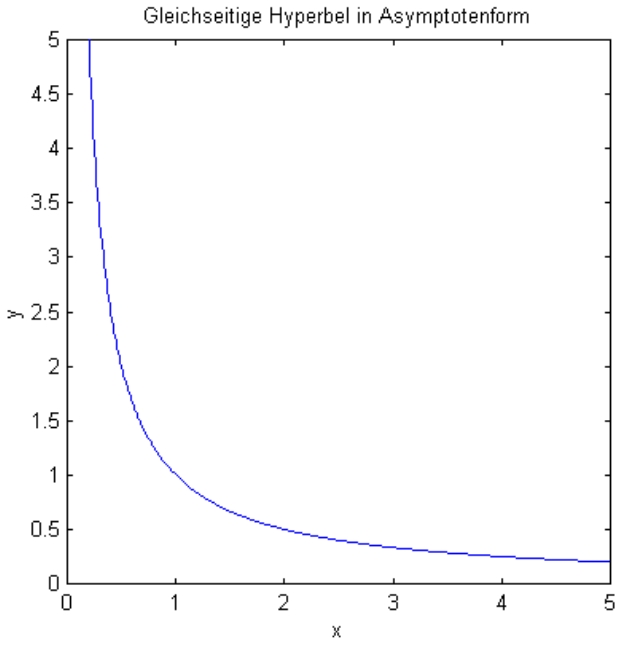


Figure 7: Example of the hyperbolic function we want to produce

Πριν προχωρήσουμε στον υπολογισμό της συνάρτησης, θα ορίσουμε το πεδίο ορισμού και το πεδίο τιμών της συνάρτησης μας. Έτσι η συνάρτηση μας θα είναι η:

$$cw(Q_i):[1,Q_{max}] \rightarrow [mincw,maxcw]$$

όπου Q_{max} είναι ο μέγιστος αριθμός των αποθηκευμένων στην ουρά frames που μπορεί ένας κόμβος να υποστηρίξει, mincw και maxcw είναι οι ελάχιστη και η μέγιστη τιμή στο contention window που μπορεί να χρησιμοποιήσει ένας κόμβος. Η ελάχιστη τιμή contention window είναι αυτή που θα χρησιμοποιηθεί όταν το στιγμιαίο μέγεθος της ουράς είναι Q_{max}.

Λαμβάνοντας υπόψη τα παραπάνω, και χρησιμοποιώντας δύο σταθερά σημεία, τα $(1, maxcw)$ and $(Qmax, mincw)$, παίρνουμε τελικά την συνάρτηση που υπολογίζει το contention window βασισμένο στην σχέση 3.1:

$$CW(Q_i) = (maxcw - mincw) * \frac{(Q_{max} - Q_i)^2}{Q_{max}^2} + mincw \qquad (3.2)$$

Ένα πρόβλημα που πρέπει να αντιμετωπιστεί, είναι σε ποια τιμή θα θέσουμε την μεταβλητή maxcw. Θα μπορούσαμε να το θέσουμε στην τιμή CW_{max}, την ανώτατη τιμή που ορίζεται για το contention window. Αυτό θα επέφερε πολλά προβλήματα, όπως τεράστια CW για τους σταθμούς που έχουν μηδαμινό ή μικρό μέγεθος στην ουρά τους. Παρόλο που αυτό μπορεί να οδηγήσει σε μικρότερο αριθμό συγκρούσεων, μπορεί επιπλέον να οδηγήσει σε μεγάλο χρόνο στον οποίο ο σταθμός απλά περιμένει, κάνοντας backoff, κάτι το οποίο σημαίνει και χαμηλότερο throughput. Μέσα από πειράματα, τα οποία εξηγούνται πλήρως σε επόμενο κεφάλαιο, συμπεραίνουμε πως μια τιμή κοντά στο 20% του CW_{max} είναι αυτή που ισορροπεί μεταξύ στη μεγιστοποίηση του throughput και της μείωσης των collisions.

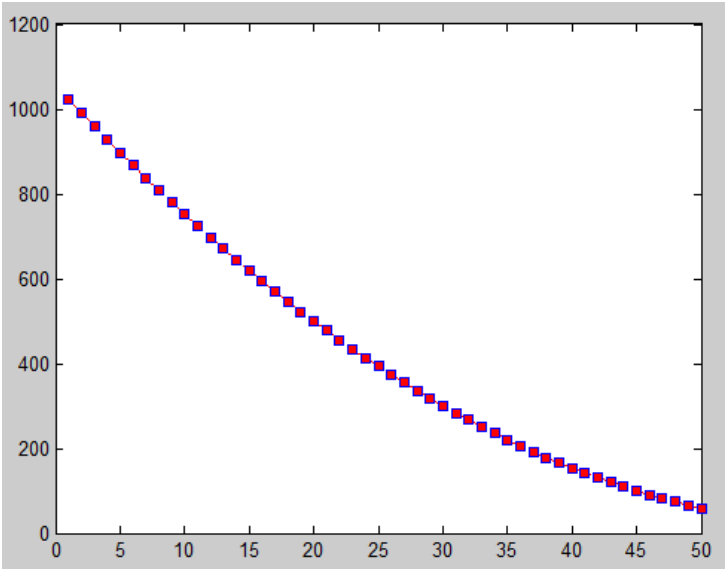


Figure 8: Graph of 3.2 for maxcw=1023

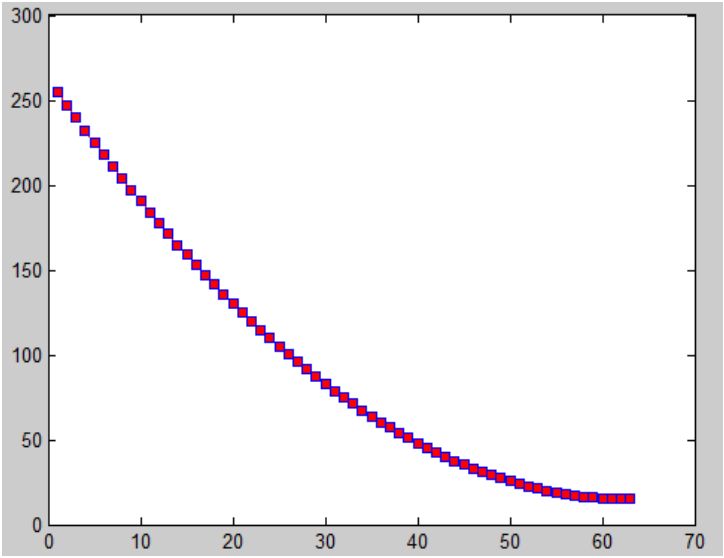


Figure 9: Graph of 3.2 for maxcw=255

Στην εξίσωση μας 3.2, πρέπει να περιλάβουμε με κάποιο τρόπο την ποιότητα της ζεύξης του καναλιού. Με την προϋπόθεση τελικά πως τρέχει κάποιος rate adaptation αλγόριθμος από πίσω, μπορούμε να υποθέσουμε πως το rate

αντικατοπτρίζει την ποιότητα του καναλιού. Στο IEEE 802.11g όπου έγιναν τα περισσότερα των πειραμάτων, οι υποστηριζόμενοι ρυθμοί μετάδοσης για ένα σταθμό είναι 6, 9, 12, 18, 24, 36, 48, 54 Mbps και οι ρυθμοί που χρησιμοποιούνται στο 802.11b 1, 2, 5.5, 11 Mbps. Ορίζουμε μια νέα μεταβλητή R_i, η οποία παίρνει τιμές από τα δύο σύνολα που μόλις αναφέραμε, και μια R_{max} που είναι ο μέγιστος ρυθμός που ένας σταθμός μπορεί να μεταδώσει (στο IEEE 802.11g είναι 54Mbps).

Για να προσθέσουμε την εξάρτηση από τον ρυθμό μετάδοσης στην εξίσωση μας, αλλάζουμε το τελευταίο μέρος της (3.2), όπου προσθέτουμε το mincw. Μπορούμε να πολλαπλασιάσουμε το mincw με μια τιμή ≥ 1, έτσι ώστε να διαφοροποιήσουμε κόμβους με ίδιο φόρτο δικτύου, με τρόπο τέτοιο ώστε ο κόμβος με το μεγαλύτερο ρυθμό, να έχει μεγαλύτερες πιθανότητες πρόσβασης στο μέσο. Στην περίπτωση όπου ο σταθμός μεταδίδει με μέγιστο ρυθμό R_{max}, η τιμή mincw θα πολλαπλασιαστεί με το 1. Λαμβάνοντας υπόψη αυτά, τελικά έχουμε την παρακάτω εξίσωση υπολογισμού του CW, βασιζόμενοι στο ρυθμό μετάδοσης και στο μέγεθος της ουράς του σταθμού:

$$CW(Q_i, R_i) = (maxcw - mincw) * \frac{(Q_{max} - Q_i)^2}{Q_{max}^2} + \frac{2 * R_{\max} - R_i}{R_{\max}} mincw \qquad (3.3)$$

Το πρόβλημα που είχαμε προηγουμένως με την τιμή της μεταβλητής maxcw υπάρχει ακόμα, αλλά μπορεί να επιλυθεί όπως και πριν. Τα σχήματα 3 και 4 θα είναι τα ίδια και για τη νέα μας συνάρτηση για τιμή $R_i=R_{max}$. Στην περίπτωση όπου ο ρυθμός είναι μικρότερος από R_{max}, οι γραφικές παραστάσεις θα είναι λίγο μετατοπισμένες προς τα πάνω.

3.1.2 2η τεχνική

Στην δεύτερη μας προσπάθεια, θα χρησιμοποιήσουμε τις κανονικοποιημένες τιμές για το μέγεθος της ουράς και το ρυθμό μετάδοσης. Οι νέες μεταβλητές θα είναι οι $R_i' = \frac{R_i}{R_{max}}$ για τον κανονικοποιημένο ρυθμό μετάδοσης και $Q_i' = \frac{Q_i}{Q_{max}}$ για το κανονικοποιημένο μέγεθος της ουράς. Η τιμή που μπορούν να πάρουν αυτές οι μεταβλητές είναι $0 < R_i', Q_i' \leq 1$.

Αν αντιστρέψουμε αυτές τις μεταβλητές, η τιμή που θα μπορούν να πάρουν είναι ≥ 1. Το χαμηλότερο όριο της τιμής των μεταβλητών συμβαίνει όταν είτε το μέγεθος της ουράς είτε ο ρυθμός μετάδοσης είναι μέγιστα αντίστοιχα. Επομένως, ας πούμε πως θέλουμε να εκφράσουμε την τιμή του contention window συναρτήσει του κανονικοποιημένου μεγέθους της ουράς μόνο. Θα μπορούσαμε να πούμε πως $CW(i) = \frac{1}{Q'(i)} * \text{mincw}$, όπου mincw είναι το η χαμηλότερη τιμή που θα λάβει το contention window όταν η ουρά είναι γεμάτη. Παρόμοια με αυτό, το στιγμιαίο contention window εκφρασμένο σαν συνάρτηση του κανονικοποιημένου ρυθμού μετάδοσης θα είναι $CW(i) = \frac{1}{R'(i)} * \text{mincw}$.

Μπορούμε τελικά να εκφράσουμε το contention window σαν συνάρτηση και των δύο μεταβλητών, απλά προσθέτοντας τις δυο παραπάνω συναρτήσεις και πολλαπλασιάζοντας κάθε μια από αυτές με κάποιες μεταβλητές, έστω K_1 και K_2, έτσι ώστε $K_1+K_2=1$. Τελικά έχουμε τη συνάρτηση

$$CW(i) = K_1 * \frac{1}{Q'(i)} * \min cw + K_2 * \frac{1}{R'(i)} * \min cw \qquad (3.4)$$

Μπορούμε να χρησιμοποιήσουμε τις μεταβλητές αυτές K_1 και K_2 έτσι ώστε να ρυθμίζουμε να δίνουμε μεγαλύτερη προτεραιότητα στο μέγεθος της ουράς ή στο ρυθμό μετάδοσης.

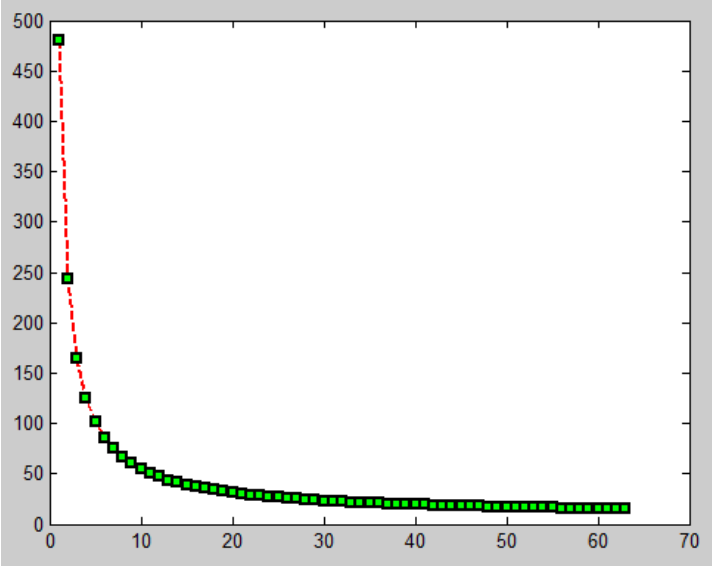


Figure 10: graph of (3.4) for K1=K2=0.5 and mincw=15[1]

Όπως παρατηρούμε από τη γραφική παράσταση, η σχέση που θέλουμε μεταξύ του contention window και του στιγμιαίου μεγέθους της ουράς είναι υπερβολικής μορφής, όπως είχαμε θέσει σαν αρχικό στόχο. Οι δυο μεταβλητές K_1, K_2 που χρησιμοποιούνται για να ρυθμίσουν την συνάρτηση, αλλάζουν την υψηλότερη τιμή του contention window όπως φαίνεται στα σχήματα 5, 6, 7 (Η τιμή για περαιτέρω κανονικοποίηση των δυο συναρτήσεων, αφού έχουν ελαφρώς διαφορετικά άνω όρια, περιλαμβάνεται στη τιμή των μεταβλητών). Επιπλέον μπορούν να χρησιμοποιηθούν για να «απλώσουν» τις τιμές της συνάρτησης, πράγμα πολύ χρήσιμο για εμάς, αφού τα πειράματα τα τρέχουμε στον MadWiFi Driver που δεν υποστηρίζει πράξεις με δεκαδικά ψηφία. Για παράδειγμα, η τιμή 6.2 θα κατανοηθεί το ίδιο για τον driver με το 6.9, σαν 6.

[1] mincw είναι η χαμηλότερη τιμή που θέτουμε στο CW. Του δίνουμε την τιμή 15, το CW που και το IEEE 802.11g πρωτόκολλο χρησιμοποιεί.

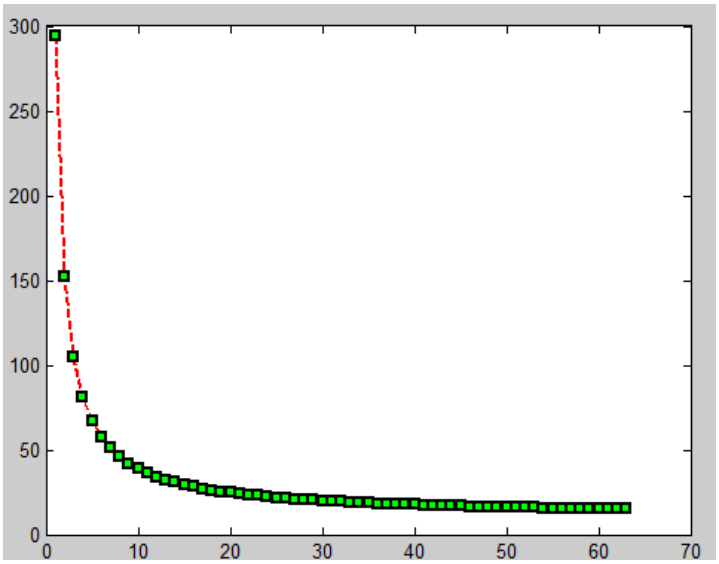


Figure 11: graph of 3.4 for K1=0.7, K2=0.3 and mincw=15

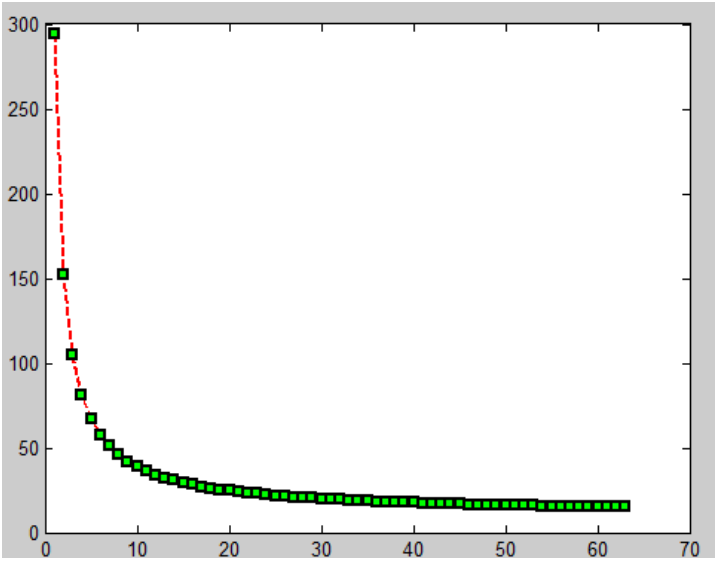


Figure 12: graph of 3.4 for K1=0.7, K2=0.3 and mincw=15

3.2 Επιλογή των τιμών K_1 και K_2

Μετά από μερικά απλά πειράματα (βλ. κεφάλαιο 4), μπορούμε να παρατηρήσουμε πως η σωστή επιλογή των τιμών των μεταβλητών K_1 και K_2 μπορεί να παίξει βασικό ρόλο στην επιτυχία της τεχνικής μας. Όπως δείχνουμε και σε επόμενο κεφάλαιο, όταν δίνουμε μεγαλύτερο βάρος στο κανονικοποιημένο μέγεθος της ουράς, μπορούμε μέχρι και να διπλασιάσουμε το throughput για τον σταθμό που μεταδίδει με τον υψηλότερο ρυθμό στο δίκτυο μας, σε σχέση με το αρχικό IEEE 802.11. Επομένως αναπτύξαμε ένα νέο σχήμα επικοινωνίας, έτσι ώστε να εξασφαλίσουμε πως όλοι οι σταθμοί έχουν της απαιτούμενες πληροφορίες, έτσι ώστε να υπολογίσουν μόνοι τους τις τιμές K_1 και K_2. Όπως θα δείξουμε και πιο εκτενώς στο κεφάλαιο 5, βασιζόμαστε στο ρυθμό όλων των άλλων σταθμών στο BSS που μας ενδιαφέρει.

Κεφάλαιο 4

4.1 Πειράματα

Για την αξιολόγηση των νέων τεχνικών, αρχικά έτρεξα τον νέο κώδικα χρησιμοποιώντας τρείς υπολογιστές, ένα σαν Access point (AP) και δύο άλλους σαν σταθμούς (STA), που θα συναγωνίζονται ο ένας τον άλλο για την πρόσβαση στο μέσο. Για να ρυθμίσουμε ένα υπολογιστή που τρέχει τον Mad-WiFi driver σαν AP τρέχουμε τις εντολές:

$$wlanconfig\ athX\ destroy$$

$$wlanconfig\ athX\ create\ wlandev\ wifiY\ wlanmode\ AP$$

όπου X είναι το όνομα που έχει το wireless interface όταν εκτελούμε την εντολή iwconfig και Y είναι η εγκατεστημένη συσκευή που θα χρησιμοποιήσουμε. Σε συστήματα που έχουν μόνο μια ασύρματη κάρτα είναι συνήθως.

Δεν χρειάζεται να ρυθμίσουμε τους σταθμούς, αφού ο Mad-WiFi driver αρχικά τα θέτει σαν STA μόλις φορτώνεται ο Linux Kernel.

Πρέπει να δημιουργήσουμε το ασύρματο δίκτυο με το AP το οποίο θα το βλέπουν οι STAs. Αρχικά αναθέτουμε IP σε όλους τους υπολογιστές με την εντολή

$$ifconfig\ athX\ address\ up$$

Όλες οι IP πρέπει να ανήκουν στο ίδιο το υποδίκτυο. Στη συνέχεια δημιουργούμε το δίκτυο μας με την εντολή iwconfig στο AP:

$$iwconfig\ athX\ channel\ Z\ essid\ \text{“}my\_network\text{”}$$

όπου Z είναι το κανάλι που θα χρησιμοποιήσουμε, συγκεκριμένα στο 802.11g τα 1-12.

Όταν τρέχουμε την εντολή που μας εμφανίζει όλα τα δίκτυα που βλέπει ο κάθε σταθμός, θα πρέπει να βλέπουμε το δικό μας. Συνδεόμαστε στο δίκτυο μας με την εντολή iwconfig.

$$wlanconfig\ athX\ list\ scan$$

$$iwconfig\ athX\ essid\ "my\_network"$$

Τώρα πρέπει να δημιουργήσουμε πακέτα προς αποστολή από τους STAs στο AP. Αυτό μπορεί να γίνει με την εντολή iperf. Δημιουργούμε έναν iperf-server στο AP, που θα δέχεται τα πακέτα που δημιουργήσαμε

$$iperf\ \ -s\ -u$$

Τώρα έχουμε ορίσει το AP να δέχεται τα UDP datagrams που θα του στέλνει το δίκτυο. Στη μεριά των σταθμών, επιλέγουμε το εύρος με το οποίο θα προσπαθήσει ο κάθε σταθμός να στείλει (ανεξάρτητα από το capacity του link) στο AP

$$iperf\ -c\ IP\_ADDRESS\_OF\ AP\ -u\ -b\ 10M\ -i\ 1\ -t\ 100$$

Στο παράδειγμα αυτό ο σταθμός προσπαθεί να στείλει πακέτα με ρυθμό 10Mbps για 100 δευτερόλεπτα, και η iperf μας εμφανίζει πληροφορίες κάθε 1 δευτερόλεπτο.

4.2.1 1η τεχνική

Πρέπει να κάνουμε compile τον driver εκ νέου για κάθε τεχνική που εφαρμόζουμε. Τα πειράματα εκτελέστηκαν αρκετές φορές για να είναι όσο το δυνατόν πιο έγκυρα τα αποτελέσματα. Επιπλέον πρέπει να πάρουμε μετρήσεις και για τον «απείραχτο» Mad-WiFi driver ώστε να έχουμε ένα σημείο αναφοράς για τα πειράματα μας.

Αρχικά τρέξαμε το πείραμα χρησιμοποιώντας μόνο 2 σταθμούς και ένα AP. Τα αποτελέσματα της πρώτης τεχνικής συνοψίζονται στον παρακάτω πίνακα. Οι τιμές στις παρενθέσεις είναι οι μετρήσεις που είχαμε πάρει με τον Mad-WiFi χωρίς να κάνουμε κάποια αλλαγή στον κώδικά του.

Table 1: Αποτελέσματα μετρήσεων για 2 σταθμούς για διάφορα rates

Σταθμός	Link Quality	Rate	10M	20M	50M
STA1	50/70	54M	10M (9.85M)	14.0M (11.9M)	13.9M (12,4M)
STA2	35/70	54M	10M (9.84M)	14.4M (12,4M)	13.8M (12.8M)
STA1	50/70	54M	9.58M (8,90M)	14.5M (9.12M)	14.8M (9.65M)
STA2	35/70	36M	9.96M (10M)	9.36M (11.7M)	9.27M (11.2M)

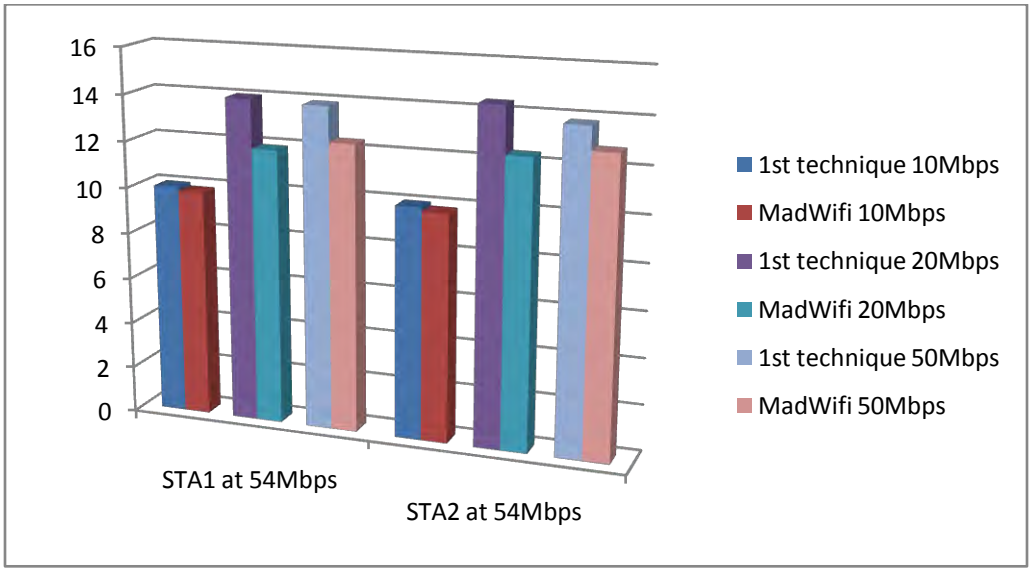


Figure 13: Γραφική αναπαράσταση throughput νέας τεχνικής για ίδιο rate

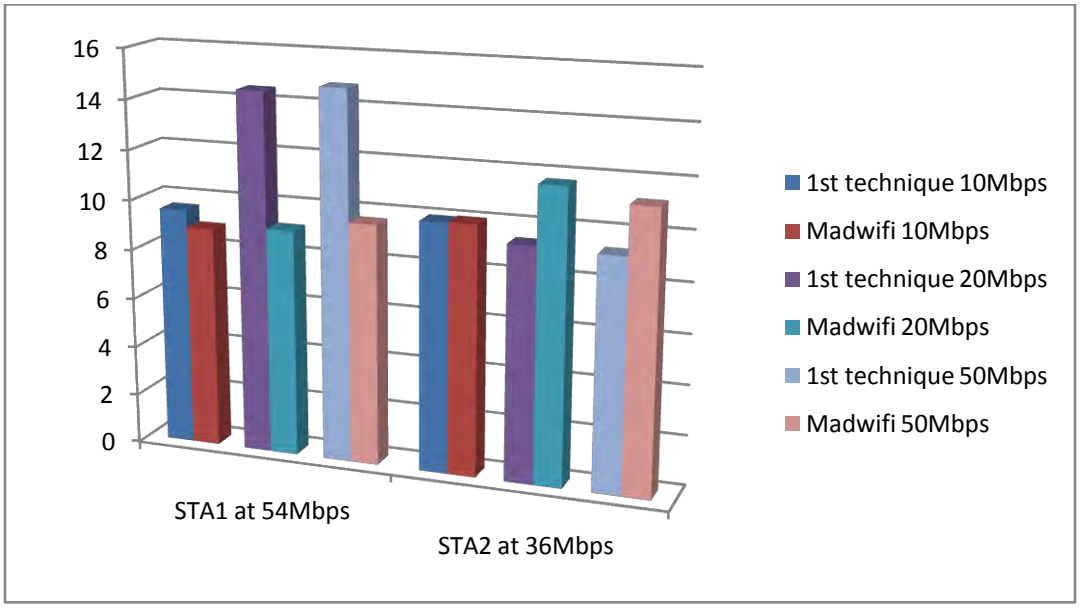


Figure 14: Γραφική αναπαράσταση αποτελεσμάτων νέας τεχνικής για διαφορετικά rates στον κάθε σταθμό

Παρατηρούμε μια μεγάλη αύξηση στο throughput στον σταθμό που στέλνει με μεγαλύτερο ρυθμό μετάδοσης. Χαμηλότερες τιμές για τους ρυθμούς μετάδοσης έχουν ανατεθεί στους υπολογιστές με το χειρότερο link quality, αφού υποθέτουμε πως κάποιος rate adaptation αλγόριθμος τρέχει.

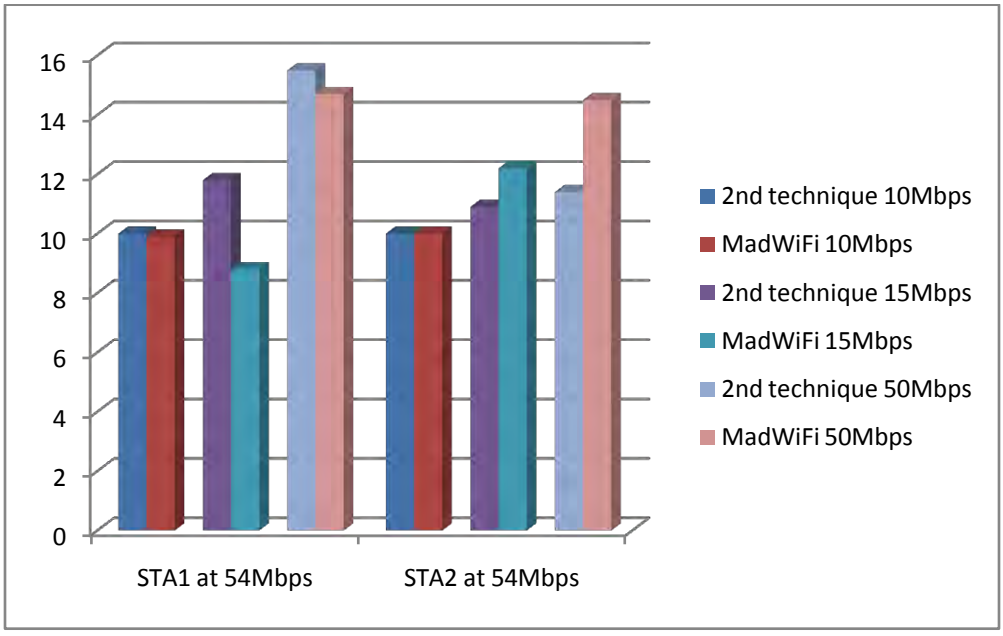
Το συνολικό throughput φαίνεται να έχει ανέβει. Επιβεβαιώσαμε τα αποτελέσματα μας χρησιμοποιώντας ένα packet sniffer, το Wireshark. Ελέγξαμε τις χρονοσφραγίδες των πακέτων για να δούμε για ποιο λόγο ανέβηκε το συνολικό throughput. Το συμπέρασμα είναι πως αφού αναθέτουμε μεγαλύτερο CW σε ένα σταθμό, έχει μεγαλύτερη πιθανότητα να δαπανά περισσότερο χρόνο στην διαδικασία του backoff. Επομένως ο ανταγωνιστικός σταθμός του, χρειάζεται απλά να περιμένει για DIFS, για να ελέγξει το κανάλι αν κάποιος μεταδίδει, και αφού το ακούσει άδειο θα μεταδώσει. Έτσι έχουμε τελικά συνολικά λιγότερο χρόνο να δαπανάται για την διαδικασία του backoff με αποτέλεσμα να έχουμε καλύτερο συνολικό thoughput.

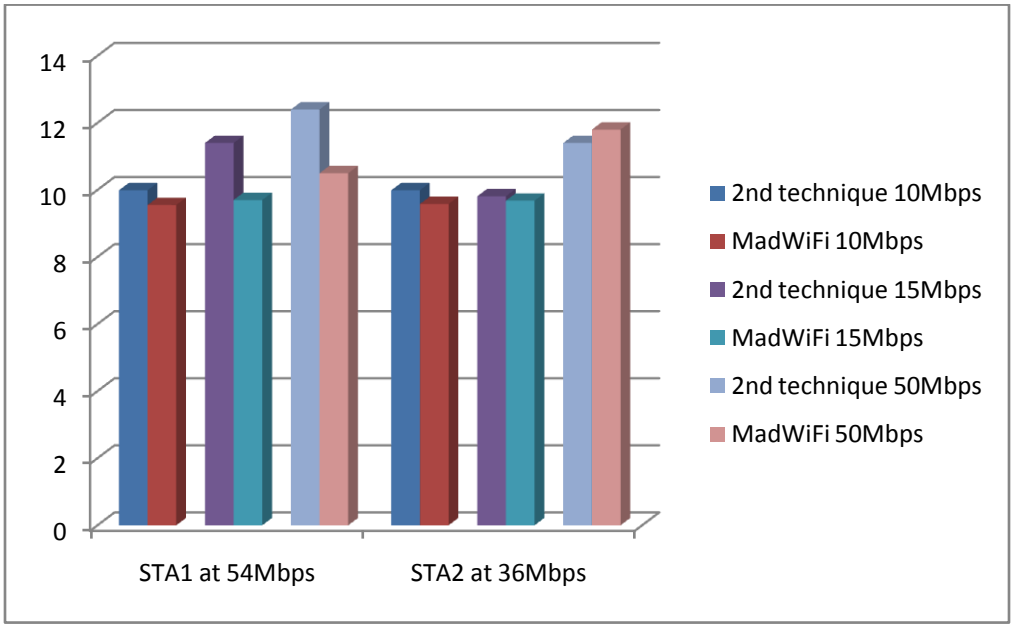
4.2.2 2η τεχνική

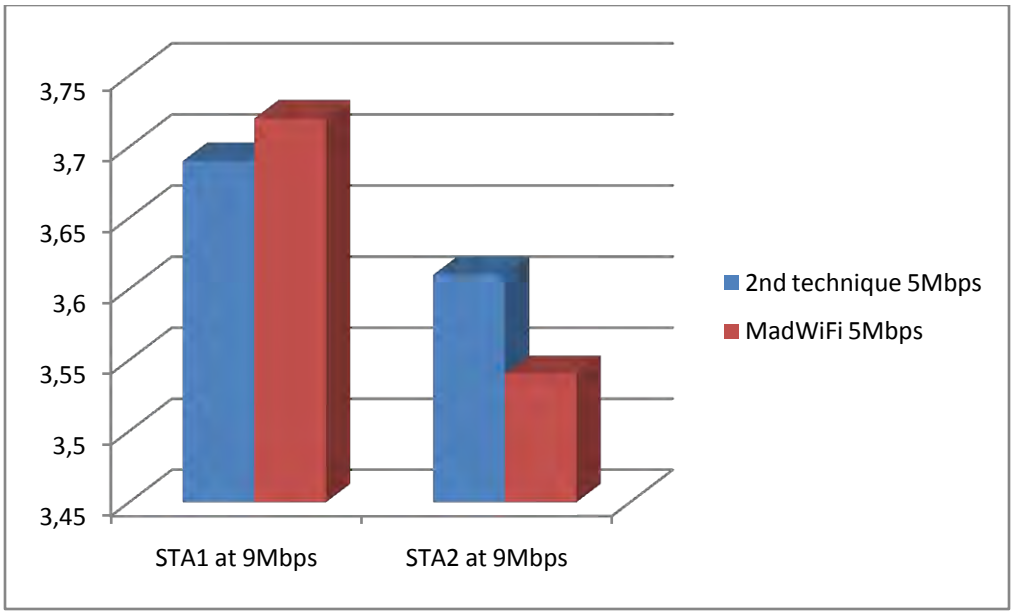
Παρόμοια με την πρώτη μας τεχνική, τρέξαμε τα πειράματα χρησιμοποιώντας 2 STAs και ένα AP. Τα αποτελέσματα για την δεύτερη μας τεχνική, για διαφορετικές τιμές των μεταβλητών K_1 και K_2 παρουσιάζονται στους επόμενους πίνακες:

Για ίσα **K1=K2=0.5** (συμπεριλαμβάνουμε μέσα στις μεταβλητές ένα μικρό offset για την κανονικοποίηση των μεγεθών ώστε να συνεισφέρουν το ίδιο στο τελικό αποτέλεσμα, αφού μέγιστο μέγεθος ουράς για τις συσκευές που έγιναν οι μετρήσεις ήταν το 63 και μέγιστο rate το 54Mbps)

Σταθμός	Link Quality	Rate	10M	15M	50M
STA1	50/70	54M	9.99M (9.55M)	11.4M (9.7M)	12,4M (10.5M)
STA2	35/70	36M	9.99M (9.58M)	9.81 (9.68M)	11,4M (11.8M)
STA1	50/70	54M	10M (9,90M)	11.8M (8.80M)	15.5M (14.7M)
STA2	35/70	54M	10M (10M)	10,9M (12.2M)	11.4M (14.5M)
			5M		
STA1	50/70	9M	3.69M (3.72M)		
STA2	35/70	9M	3.61M (3.54M)		


16
14
12
10
8
6
4
2
0
STA1 at 54Mbps
STA2 at 54Mbps
2nd technique 10Mbps
MadWiFi 10Mbps
2nd technique 15Mbps
MadWiFi 15Mbps
2nd technique 50Mbps
MadWiFi 50Mbps

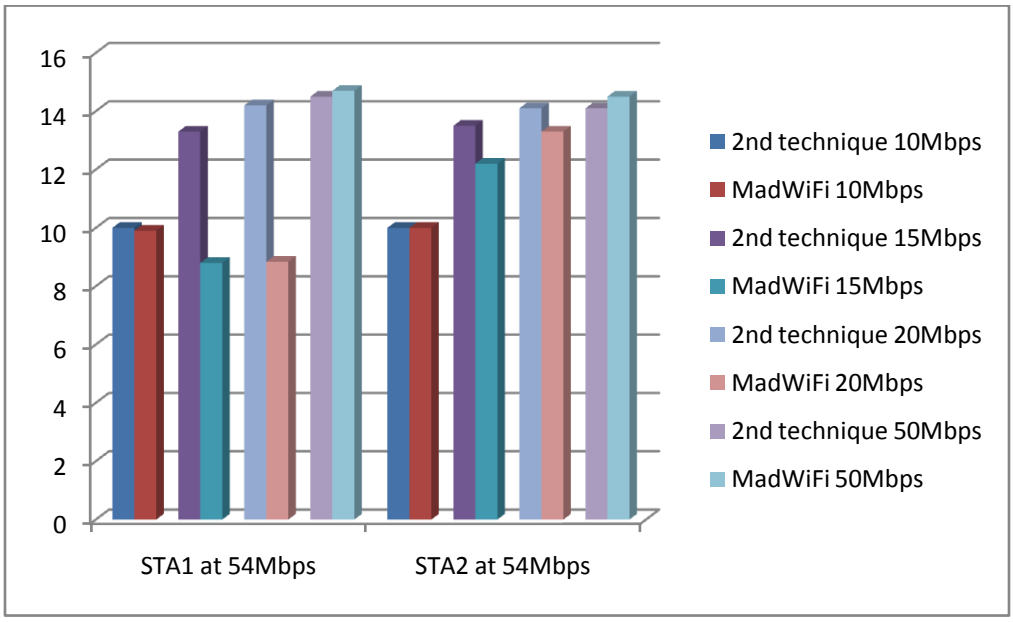

14
12
10
8
6
4
2
0
STA1 at 54Mbps
STA2 at 36Mbps
2nd technique 10Mbps
MadWiFi 10Mbps
2nd technique 15Mbps
MadWiFi 15Mbps
2nd technique 50Mbps
MadWiFi 50Mbps

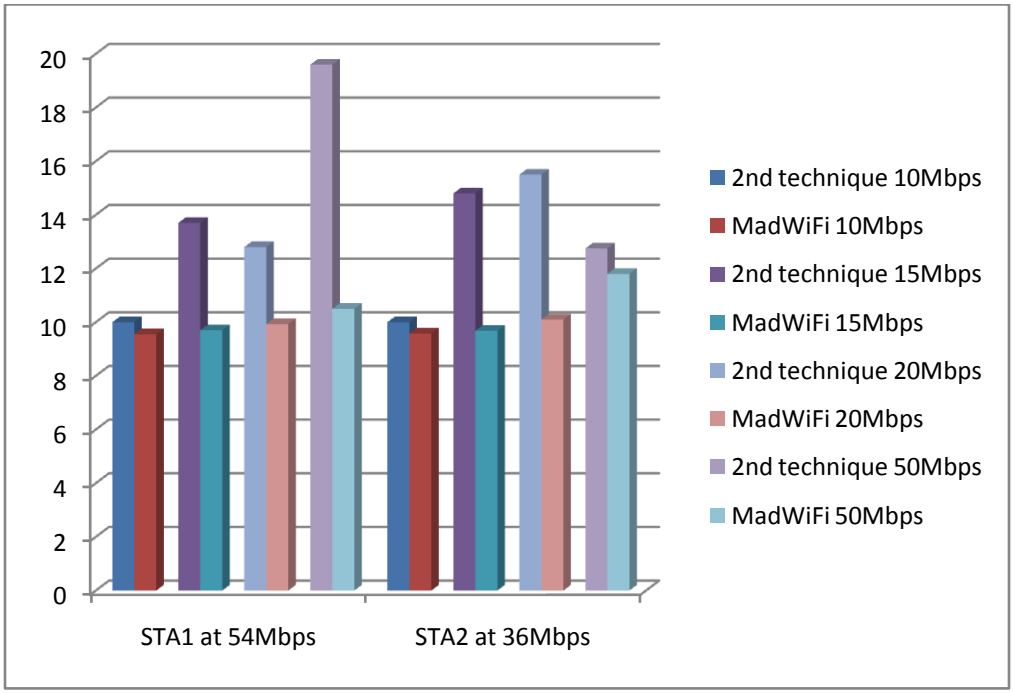


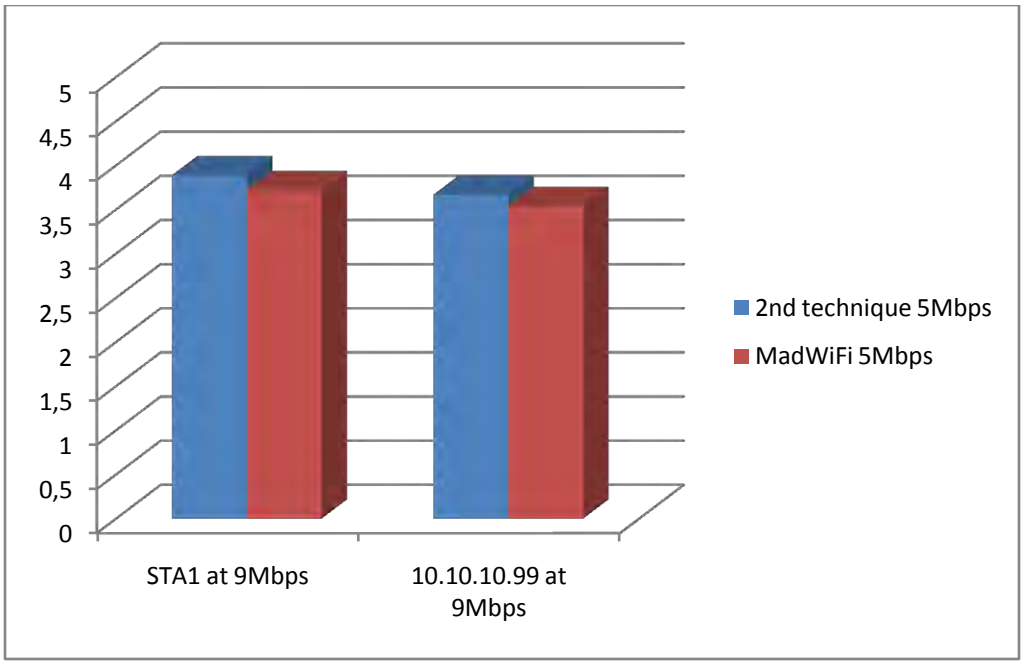
Όπως μπορούμε να δούμε, το συνολικό throughput αυξάνεται ξανά, όπως και με την πρώτη τεχνική. Όμως, όταν χρησιμοποιούμε χαμηλότερους ρυθμούς μετάδοσης, οι STAs κάνουν backoff για περισσότερο χρόνο (δηλ. με μεγαλύτερη πιθανότητα) απ' ότι όπως είναι στο IEEE 802.11. Γι αυτό τον λόγο το συνολικό throughput είναι πεσμένο σε σχέση με τον Mad-WiFi χωρίς αλλαγές. Αυτό το γεγονός το εκμεταλλεύονται οι πιο γρήγοροι σταθμοί, οι οποίοι στέλνοντας περισσότερα δεδομένα στη μονάδα του χρόνου, συνεισφέρουν σημαντικά στη μεγιστοποίηση του throughput.

Όταν χρησιμοποιούμε **K1=0.7, K2=0.3**

Σταθμός	Link Quality	Rate	10M	15M	20M	50M
STA1	50/70	54M	10M (9.55M)	13.7M (9.7M)	12,8M (9.93M)	19,6M (10.5M)
STA2	35/70	36M	10M (9.58M)	14.8 (9.68M)	15,5M (10.1M)	12,75M (11.8M)
STA1	50/70	54M	10M (9,90M)	13.3M (8.80M)	14.2M (8.84M)	14.5M (14.7M)
STA2	35/70	54M	10M (10M)	13.5M (12.2M)	14.1M (13.3M)	14.4M (14.5M)
			5M			
STA1	50/70	9M	3.89M (3.72M)			
STA2	35/70	9M	3.675M (3.54M)			


16
14
12
10
8
6
4
2
0
STA1 at 54Mbps
STA2 at 54Mbps
2nd technique 10Mbps
MadWiFi 10Mbps
2nd technique 15Mbps
MadWiFi 15Mbps
2nd technique 20Mbps
MadWiFi 20Mbps
2nd technique 50Mbps
MadWiFi 50Mbps

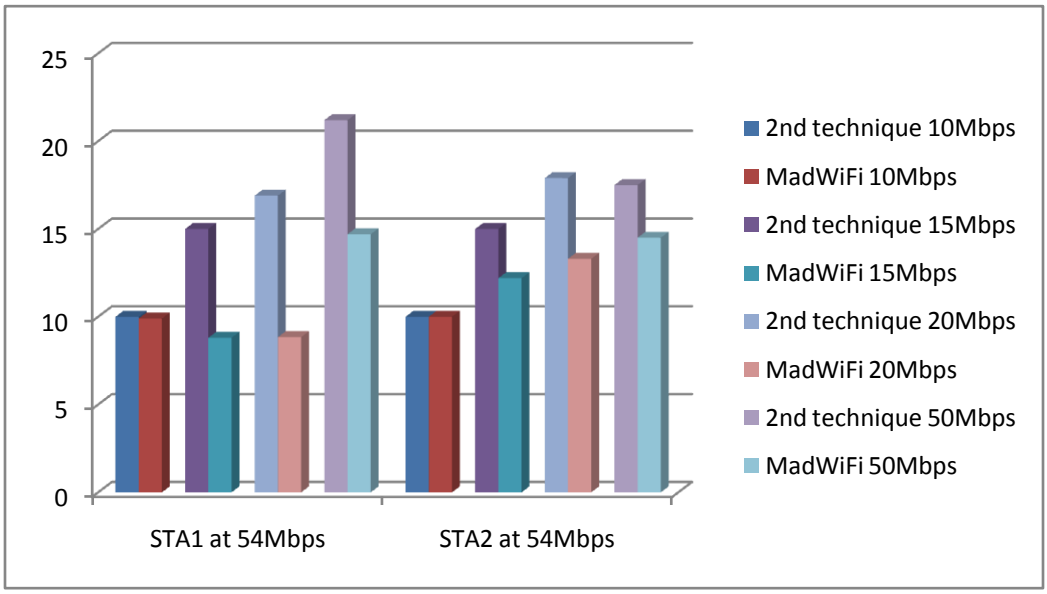

20
18
16
14
12
10
8
6
4
2
0
STA1 at 54Mbps
STA2 at 36Mbps
2nd technique 10Mbps
MadWiFi 10Mbps
2nd technique 15Mbps
MadWiFi 15Mbps
2nd technique 20Mbps
MadWiFi 20Mbps
2nd technique 50Mbps
MadWiFi 50Mbps

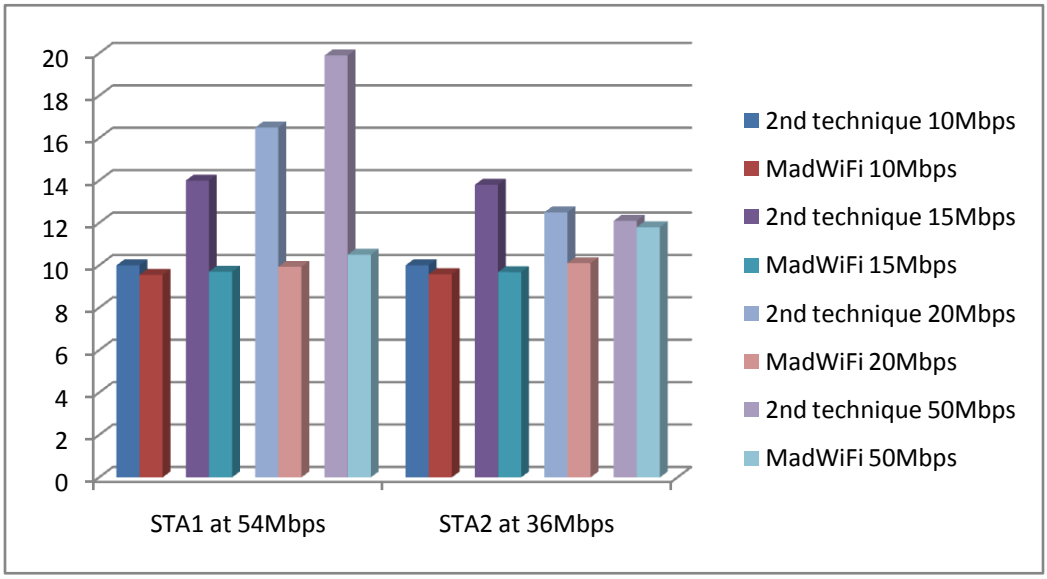


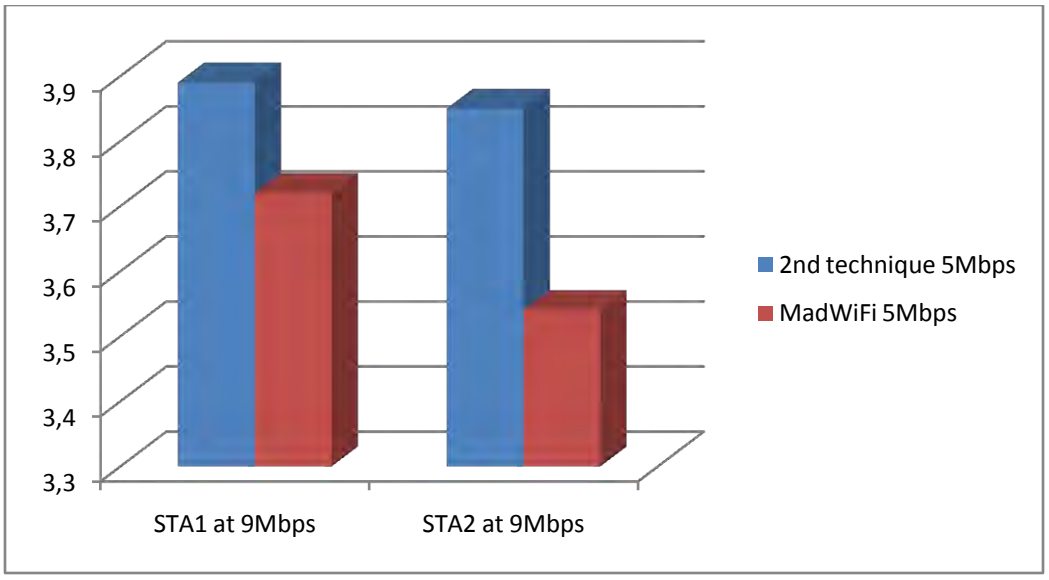
Εδώ παρατηρούμε πως για τους ίδιους ρυθμούς μετάδοσης, το σύστημα μας είναι δίκαιο μεταξύ των σταθμών αυτών. Όμως για διαφορετικούς ρυθμούς μετάδοσης, τελικά το σύστημα μας καταφέρνει να δίνει προτεραιότητα για μικρές ροές δεδομένων στον πιο αργό σταθμό. Αυτό γίνεται γιατί κατά την προσπάθεια για την πρώτη πρόσβαση στο κανάλι, έχει περισσότερα πακέτα στην ουρά (αφού στέλνει με πιο αργό ρυθμό) και τελικά του ανατίθεται χαμηλότερο CW. Το γεγονός αυτό αλλάζει στη περίπτωση που έχουμε πιο μεγάλες ροές δεδομένων, αφού σχεδόν ταυτόχρονα γεμίζουν οι ουρές, και μένει μόνη περίπτωση διαφοροποίησης το rate του κάθε σταθμού, όπου έτσι τελικά παίρνει μεγαλύτερη προτεραιότητα ο πιο γρήγορος.

Όταν χρησιμοποιούμε **K1=**0.3**, K2=**0.7

Σταθμός	Link Quality	Rate	10M	15M	20M	50M
STA1	50/70	54M	10M (9.55M)	14M (9.7M)	16,5M (9.93M)	19,9M (10.5M)
STA2	35/70	36M	10M (9.58M)	13.8 (9.68M)	12,5M (10.1M)	12,1M (11.8M)
STA1	50/70	54M	10M (9,90M)	15M (8.80M)	16.9M (8.84M)	21.2M (14.7M)
STA2	35/70	54M	10M (10M)	15M (12.2M)	18.7M (13.3M)	17.5M (14.5M)
			5M			
STA1	50/70	9M	3.89M (3.72M)			
STA2	35/70	9M	3.85M (3.54M)			


25
20
15
10
5
0
STA1 at 54Mbps
STA2 at 54Mbps
2nd technique 10Mbps
MadWiFi 10Mbps
2nd technique 15Mbps
MadWiFi 15Mbps
2nd technique 20Mbps
MadWiFi 20Mbps
2nd technique 50Mbps
MadWiFi 50Mbps


20
18
16
14
12
10
8
6
4
2
0
STA1 at 54Mbps
STA2 at 36Mbps
2nd technique 10Mbps
MadWiFi 10Mbps
2nd technique 15Mbps
MadWiFi 15Mbps
2nd technique 20Mbps
MadWiFi 20Mbps
2nd technique 50Mbps
MadWiFi 50Mbps



Στις νέες αυτές μετρήσεις παρατηρούμε πως τελικά καταπνίγουμε τον πιο αργό σταθμό, και παίρνουμε έτσι σημαντικά μεγαλύτερο throughput πάνω από το κανάλι. Στην κατεύθυνση αυτή συντελεί επίσης και το χαμηλό overhead από τις αλλαγές στον driver, όπου αλλάζει κατά πολύ μικρούς παράγοντες το CW για κάθε νέο πακέτο στην ουρά, δεδομένου ότι διαχειρίζεται και μόνο ακεραίους σαν αριθμούς, και όχι δεκαδικούς.

Σημείωση: Όπως παρατηρούμε, το συνολικό throughput πάνω από το μέσο έχει σχεδόν διπλασιαστεί, όταν χρησιμοποιούμε την δικιά μας τεχνική αντί για τον Mad-WiFi. Αυτό προκαλείται εξαιτίας του ότι όταν μεταδίδουμε ένα frame σε χαμηλότερο ρυθμό, συνεπάγεται ότι το frame θα χρειαστεί περισσότερο χρόνο για να σταλεί και επομένως όσοι άλλοι σταθμοί ανταγωνίζονται για πρόσβαση στο μέσο θα πρέπει να σταματήσουν την διαδικασία του backoff τους για περισσότερο χρόνο απ' ότι θα έκαναν για ένα σταθμό που θα μετέδιδε με υψηλότερο rate (υποθέτουμε ότι όλοι οι σταθμοί χρησιμοποιούν ίδιο μέγεθος frames για τα πειράματα μας). Όμως όταν αναθέτουμε μικρότερο CW στους σταθμούς με το υψηλότερο ρυθμό μετάδοσης, η πιθανότητα πρόσβασης στο μέσο είναι μεγαλύτερη και επομένως η ποσότητα των δεδομένων που

στάλθηκε πάνω από το δίκτυο είναι μεγαλύτερη (δεδομένου ότι αναθέτουμε μεγαλύτερο CW στον σταθμό που μεταδίδει με χαμηλότερο ρυθμό). Επομένως οι σταθμοί κάνουν backoff για συνολικά λιγότερο χρόνο από την default διαδικασία και το συνολικό throughput είναι ανεβασμένο.

Κεφάλαιο 5

5.1 Επιλογή των τιμών των K_1 και K_2

Όπως παρατηρούμε από τα πειράματα μας, η επιλογή των τιμών των K_1 και K_2 μπορεί να παίξει σημαντικό ρόλο στην επιτυχία της τεχνικής μας. Για την επιλογή των τιμών αυτών, θα αποφασίσουμε με βάση τους ρυθμούς μετάδοσης που χρησιμοποιούν όλοι οι σταθμοί στο δίκτυο μας.

Το πρωτόκολλο CSMA/CA, ακούει το μέσο για να διαπιστώσει αν κάποιος μεταδίδει. Επομένως ο κάθε σταθμός θα πρέπει να γνωρίζει ποιοι άλλοι σταθμοί είναι μέσα στην sensing area του. Όμως, όταν χρησιμοποιούμε τον Mad-WiFi driver η πληροφορία αυτή δεν είναι διαθέσιμη, αφού όλο το carrier sensing υλοποιείται στο hardware. Ο μόνος που μπορεί να γνωρίζει, σε επίπεδο driver, ποιοι σταθμοί είναι συνδεδεμένοι στο δίκτυο μας, είναι το AP. Επομένως, την πληροφορία αυτή μπορεί με κάποιο τρόπο να «διαφημίσει» το AP στα beacons που στέλνει.

Όπως αποφασίσαμε, η επιλογή των K_1 και K_2 θα εξαρτάται από το διάνυσμα των ρυθμών μετάδοσης όλων των σταθμών στο δίκτυο μας. Πρέπει να ορίσουμε ένα τρόπο με τον οποίο ο κάθε σταθμός θα μπορεί να γνωρίζει τον ρυθμό με τον οποίο όλοι οι υπόλοιποι σταθμοί μεταδίδουν. Αυτό ίσως θα μπορούσε να γίνει με την χρήση των debug messages που έχει ο driver, όμως αυτό θα πρόσθετε παραπάνω overhead στην υλοποίηση μας. Επομένως, αποφασίσαμε την χρήση των management πακέτων, που χρησιμοποιούνται για να καθιερώσουν/τερματίσουν την επικοινωνία στο δίκτυο, με σκοπό την μεταφορά της πληροφορίας που θέλουμε (ρυθμό μετάδοσης) στο AP. Έτσι η παραπάνω αυτή πληροφορία θα μπορεί να «διαφημίζεται» μαζί με τους σταθμούς στα beacons.

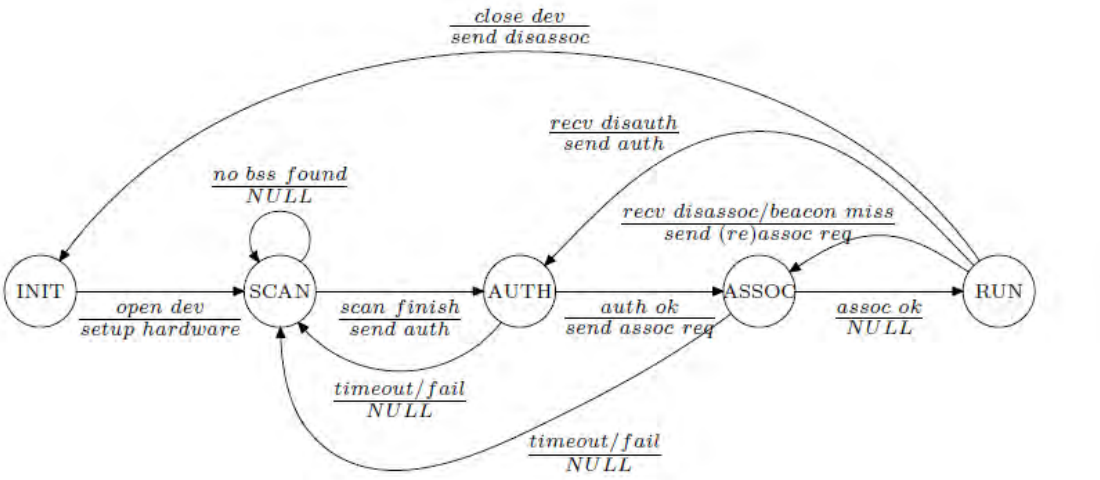


Figure 15: Διάφορες καταστάσεις και η ανταλλαγή management μηνυμάτων για την καθιέρωση της επικοινωνίας

Το σχήμα μας δείχνει τα μηνύματα τα οποία κάθε σταθμός στέλνει στο AP όταν συνδέεται στο BSS, μέχρι τελικά να φτάσει στη κατάσταση RUN, όπου μπορεί πλέον να ανταλλάξει δεδομένα. Όμως, η ιδέα να στέλνουμε περιοδικά το δικό μας management πακέτο, που περιέχει και την πληροφορία για τους ρυθμούς μετάδοσης, δεν μπορεί να λειτουργήσει αφού όλα τα bits που χρησιμοποιούνται για να ορίσουν κάθε πακέτο είναι δεσμευμένα σύμφωνα και με τον παρακάτω πίνακα.

Type Value b3 b2	Type Description	Subtype Value b7 b6 b5 b4	Subtype Description
00	Management	0000	Association Request
00	Management	0001	Association Response
00	Management	0010	Reassociation Request
00	Management	0011	Reassociation Response
00	Management	0100	Probe Request
00	Management	0101	Probe Response
00	Management	0110-0111	Reserved
00	Management	1000	Beacon
00	Management	1001	ATIM
00	Management	1010	Disassociation
00	Management	1011	Authentication
00	Management	1100	Deauthentication
00	Management	1101-1111	Reserved
01	Control	0000-1001	Reserved
01	Control	1010	PS-Poll
01	Control	1011	RTS
01	Control	1100	CTS
01	Control	1101	ACK
01	Control	1110	CF End
01	Control	1111	CF End + CF-ACK
10	Data	0000	Data
10	Data	0001	Data + CF-Ack
10	Data	0010	Data + CF-Poll
10	Data	0011	Data + CF-Ack + CF-Poll
10	Data	0100	Null Function (no data)
10	Data	0101	CF-Ack (no data)
10	Data	0110	CF-Poll (no data)
10	Data	0111	CF-Ack + CF-Poll (no data)
10	Data	1000-1111	Reserved
11	Reserved	0000-1111	Reserved

Figure 16: Bits used to distinguish between different types of packets

Επομένως τελικά σαν λύση σε όλα αυτά τα προβλήματα ήταν η χρήση ενός ήδη προϋπάρχοντος management πακέτου, που θα μπορεί να ξεχωριστεί από άλλα ίδια management πακέτα χρησιμοποιώντας κάποια δικά μας flags. Τελικά επέλεξα το πακέτο που θα μεταφέρει αυτή την πληροφορία να είναι ένα Authentication πακέτο, και η περίοδος στην οποία θα στέλνεται θα είναι κάθε 30 ληφθέντα beacons.

5.2 Αλγόριθμος υπολογισμού K_1 και K_2

Το συμπέρασμα των πειραμάτων μας έδειξε πως όταν αναθέτουμε μεγαλύτερη τιμή στην μεταβλητή K_2 στον σταθμό που μεταδίδει με τον μεγαλύτερο ρυθμό μετάδοσης, το συνολικό throughput πάνω από το μέσο είναι σημαντικά αυξημένο σε σχέση με της εφαρμογή του IEEE 802.11. Ο αλγόριθμος που θα χρησιμοποιήσουμε έτσι ώστε να αλλάζουμε δυναμικά τις τιμές των μεταβλητών μας είναι ο εξής:

- If my_rate is the highest one in the BSS then K_2=0.8, K_1=0.2
- Else if my_rate is more than the half of the highest in the BSS then K_2=0.6, K_1=0.4
- Else K_2=0.3, K_1=0.7

Τα πειράματα που ακολουθούν δείχνουν την αποτελεσματικότητα του σχήματος μας, όταν επιλέγουμε τις τιμές K_1 και K_2 με βάση τον παραπάνω αλγόριθμο.

Χρησιμοποιώντας 2 σταθμούς και ένα AP, παίρνουμε τα παρακάτω αποτελέσματα. Εντός της παρένθεσης, είναι οι μετρήσεις που έγιναν με το κώδικα του MadWiFi χωρίς αλλαγές.

Table 2: Μετρήσεις της νέας τεχνικής με 2 σταθμούς

Σταθμός	Link Quality	Rate	10M	15M	20M	50M
STA1	35/70	54M	10M (9.90M)	14.8M (9.10M)	11.1M (8.90M)	4.64M (14.2M)
STA2	40/70	54M	10M (10M)	15M (12.5M)	20M (13.3M)	28.8M (15M)
STA1	35/70	36M	10M (9.55M)	11.8M (9.68M)	8.87M (10.5M)	4.36M (10.5M)
STA2	40/70	54M	10M (10M)	15M (9.7M)	20M (10.4M)	28M (11.8M)
			5M			
STA1	35/70	9M	3.39M (3.72M)			
STA2	40/70	9M	3.45M (3.54M)			

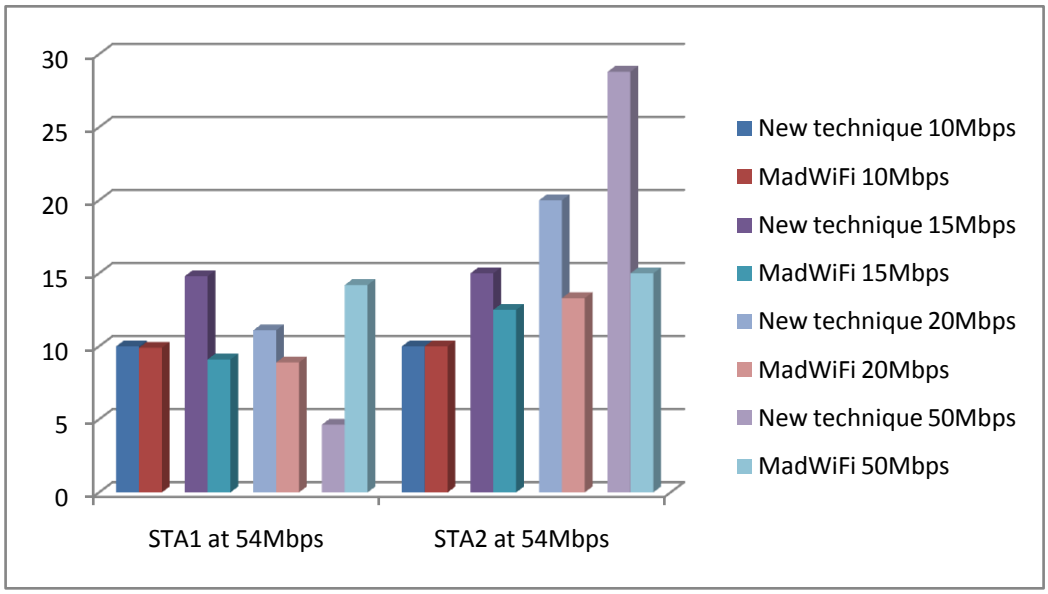


Figure 17: Αποτελέσματα για ίδιο rate και στους 2 σταθμούς

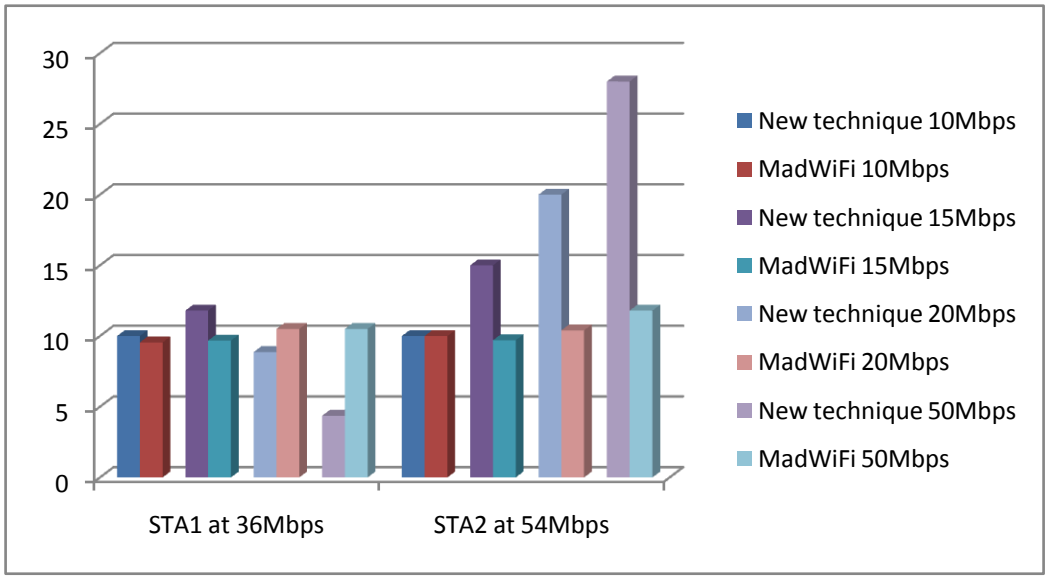
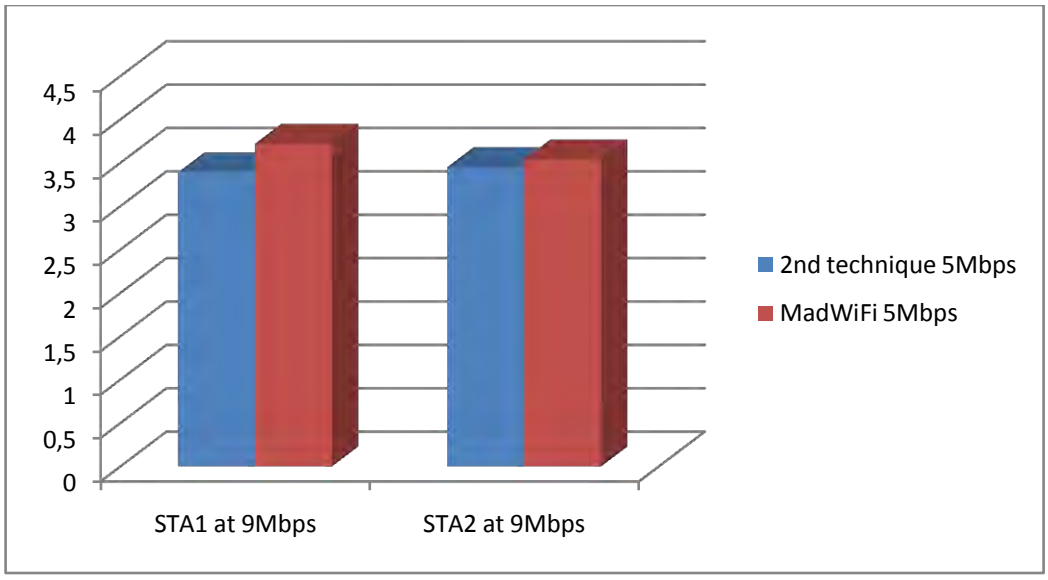


Figure 18: Αποτελέσματα για διαφορετικό rate σε κάθε σταθμό



Όπως παρατηρούμε, για επικοινωνία 2 σταθμών σε ένα AP, το συνολικό throughput που παίρνουμε είναι ανεβασμένο σε σχέση με τον απλό κώδικα του MadWiFi. Αυτό οφείλεται στους ίδιους λόγους με τα προηγούμενα αποτελέσματα (βλ. κεφ. 4). Για να δούμε κατά πόσο το νέο σχήμα αυτό υπερτερεί σε σχέση με το απλό 802.11, οι μετρήσεις έγιναν και για την επικοινωνία 3 σταθμών με ένα AP, ώστε να προσομοιάσουμε συνθήκες με παραπάνω κίνηση στο μέσο, όπου η νέα τεχνική δείχνει να υπερτερεί.

Παρακάτω παρατίθενται και τα αποτελέσματα των μετρήσεων, με χρήση 3 σταθμών και ένα AP.

Table 3: Μετρήσεις νέας τεχνικής για 3 σταθμούς

Σταθμός	Link Quality	Rate	10M	15M	20M	50M
STA1	50/70	54M	10M (2.37M)	14.5M (3.02M)	16.2M (3M)	17.8M (2.98M)
STA2	35/70	54M	5.84M (2.36M)	3.66M (3.32M)	3.25M (3.33M)	3.37M (3.38M)
STA3	22/70	36M	5.77M (1.51M)	3.25M (2.15M)	2.69M (2.2M)	3.32M (2.59M)
STA1	50/70	54M	10M (3.56M)	15M (4.74M)	18.9M (4.82M)	18.6M (4.42M)
STA2	35/70	36M	7.08M (7.06M)	4.75M (5.72M)	2.64M (5.62M)	3.3M (5.72M)
STA3	22/70	18M	4.18M (3.22M)	2.8M (4.03M)	1.66M (4.11M)	1.91M (4.37M)
STA1	50/70	36M	10M (1.76M)	11.9M (1.78M)	12M (1.76M)	
STA2	35/70	24M	4.49M (7.02M)	3.22M (11.4M)	3.43M (11.3M)	
STA3	22/70	9M	1.47M (1.3M)	1.07M (1.38M)	1.16M (1.39M)	

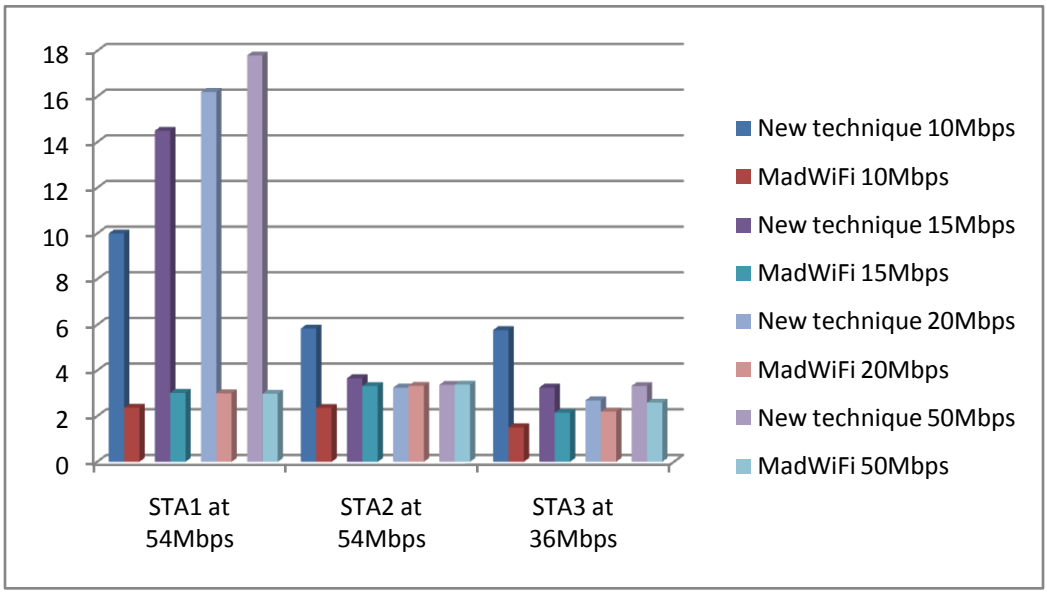


Figure 19: Αποτελέσματα νέας τεχνικής για 3 σταθμούς για διάφορα rates

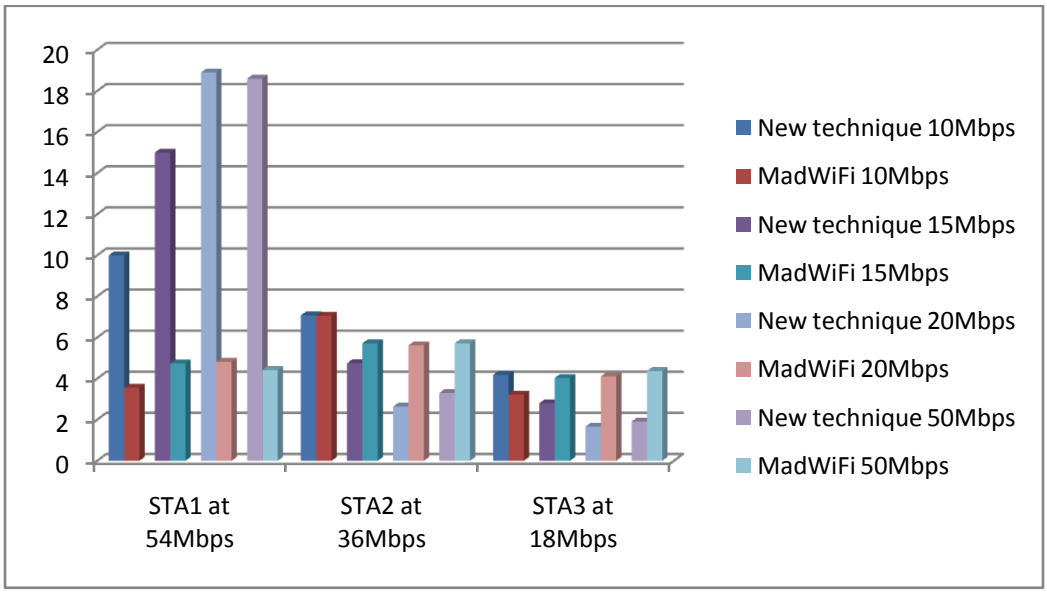


Figure 20: Αποτελέσματα νέας τεχνικής για 3 σταθμούς για διάφορα rates

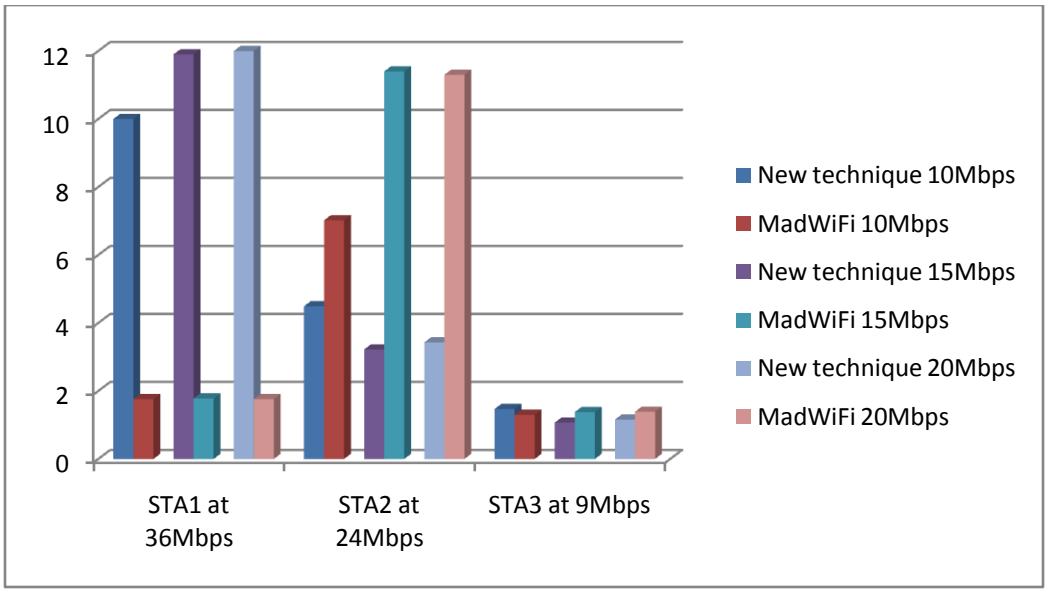


Figure 21: Αποτελέσματα νέας τεχνικής για 3 σταθμούς για διάφορα rates

Παρατηρώντας τα αποτελέσματα της νέας τεχνικής μας, βλέπουμε πως για περίπου ίδια rate όλων των σταθμών, παίρνουμε αρκετά μεγαλύτερο συνολικό throughput σε σχέση με το απλό 802.11. Αυτό οφείλεται στο γεγονός, όπως έχουμε εξηγήσει και νωρίτερα, ότι με το μεταβαλλόμενο backoff που έχουμε σε κάθε σταθμό, γίνεται μια προσπάθεια να εξαλειφθεί το άνω όριο throughput που επιτάσσει ο σταθμός με το χαμηλότερο ρυθμό μετάδοσης (βλ. σελ. 41).

Για τις περιπτώσεις των σταθμών που έχουμε διαφορετικούς ρυθμούς μετάδοσης σε όλους τους σταθμούς, βλέπουμε ότι για μεγάλη κίνηση στο κανάλι, παίρνουμε και μεγαλύτερο συνολικό throughput, σχεδόν 50% περισσότερο σε σχέση με το απλό MadWiFi. Η τεχνική αυτή αναθέτει το μεγαλύτερο throughput στο σταθμό με το μεγαλύτερο ρυθμό μετάδοσης, αλλά μόνο για τις περιπτώσεις με μεγάλη κίνηση στο κανάλι. Για μικρότερη κίνηση, ο αλγόριθμος αυτός πέφτει θύμα των μεγάλων χρόνων του backoff που αναθέτουμε, και έτσι μπορεί να αποδώσει ελάχιστα χειρότερα του MadWiFi.

Τέλος, τα αποτελέσματα που πήραμε με την νέα αυτή τεχνική, είναι ελαφρώς χειρότερα σε σχέση με την τεχνική με σταθερά K_1 και K_2. Αυτό οφείλεται στον

αλγόριθμο επιλογής των K_1 και K_2, ο οποίος για την περίπτωση μας είναι ένας απλός που έγινε μόνο για τον έλεγχο των τεχνικών μας και της αποδοτικότητας του νέου σχήματος επικοινωνίας. Έχοντας κατά νου τα αποτελέσματα που είχαμε πάρει στο Κεφάλαιο 4, έχουμε ένα άνω όριο το οποίο μπορούμε να φτάσουμε με την επιλογή του κατάλληλου αλγορίθμου για την επιλογή των σταθερών αυτών.

5.3 Επεκταμένος αλγόριθμος υπολογισμού K_1 και K_2

Προχωρώντας λίγο βαθύτερα στον τρόπο υπολογισμού των μεταβλητών K_1 και K_2, μπορούμε εύκολα να διαπιστώσουμε πως η συνάρτηση η οποία είναι βέλτιστη να εφαρμόσουμε είναι μια αύξουσα concave συνάρτηση ως προς τον ρυθμό μετάδοσης του κάθε σταθμού (βλ. Figure 21).

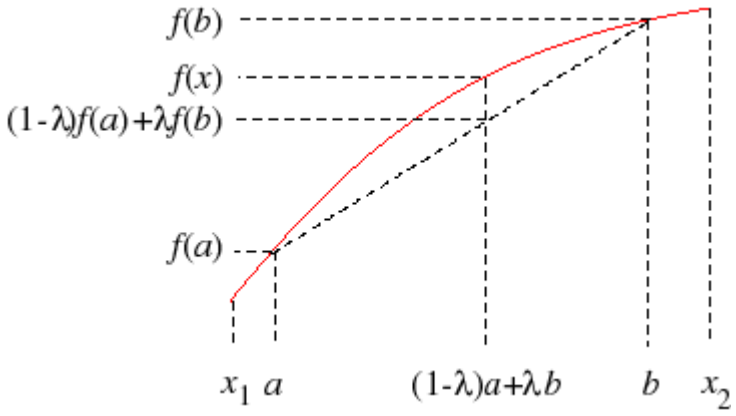


Figure 22: example of a concave function

Οι σταθμοί οι οποίοι θα έχουν μεγαλύτερο ρυθμό μετάδοσης, θα έχουν και μεγαλύτερη τιμή K_2. Η ιδιομορφία των συναρτήσεων αυτών μας επιτρέπει κατά κάποιο τρόπο να ομαδοποιούνται οι καλύτερες τιμές σε σχέση με τις μικρότερες. Αυτό είναι χρήσιμο και για την περίπτωση όπου προστίθεται κάποιος νέος σταθμός στο δίκτυο μας.

Μια τέτοια απόλυτη ομαδοποίηση θα έκανε η λογαριθμική συνάρτηση (βλ. Figure 22). Δεδομένου όμως πως ο MadWiFi driver δεν υποστηρίζει πράξεις με λογαρίθμους, η ιδέα αυτή εγκαταλείφθηκε.

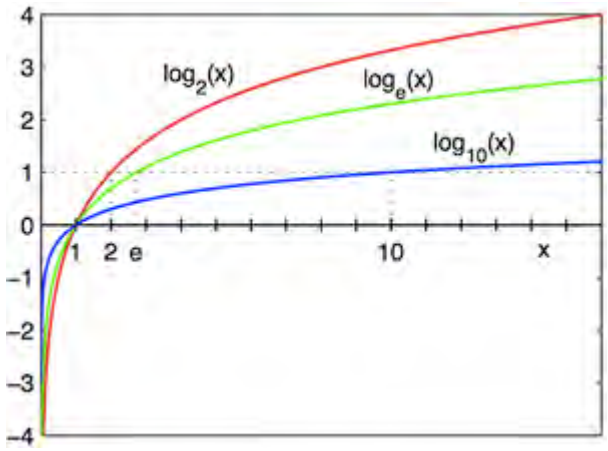


Figure 23: Logarithm Functions

Επόμενη επιλογή ήταν η χρήση μιας παραβολικής συνάρτησης, που η κορυφή της θα είναι στο σημείο που υποδεικνύεται από το μεγαλύτερο rate στο δίκτυο μας και το άνω όριο της μεταβλητής K_2. Η συνάρτηση που θα χρησιμοποιηθεί θα είναι η

$$f(x) = -a * x^2 + b * x + c$$

Επομένως κάθε φορά που θα έχουμε ένα προς μετάδοση πακέτο θα πρέπει να λύνουμε ένα μικρό σύστημα για να υπολογίζουμε τους νέους συντελεστές a,b,c, οι οποίοι τελικά θα εφαρμόζονται στην εξίσωση

$$K_2 = -a * (current\_rate)^2 + b * (current\_rate) + c$$

Θεωρούμε και τα όρια που μπορεί να πάρει η τιμή K_2 είναι 0.8 η υψηλότερη τιμή (HB) και 0.2 η χαμηλότερη (LB). Η χαμηλότερη τιμή θα ισχύει όταν το rate είναι το

χαμηλότερο δυνατό, δηλαδή 1Mbps. Κάνοντας εδώ μια μικρή παραδοχή πως αντί για 1 είναι 0, μπορούμε να δούμε με απλές πράξεις πως

$$a = \frac{HB - LB}{\max\ \_\text{rate}^2}$$

$$b = 2 * \max\ \_rate * a$$

$$c = LB$$

και άρα θα είναι και απλό υπολογιστικά η επίλυση του συστήματος (όπου max_rate είναι η μέγιστη τιμή ρυθμού μετάδοσης στο δίκτυο μας).

Η διάταξη στο χώρο της συνάρτησης αυτής φαίνεται για όλους τους δυνατούς ρυθμούς μετάδοσης στο Figure 23.

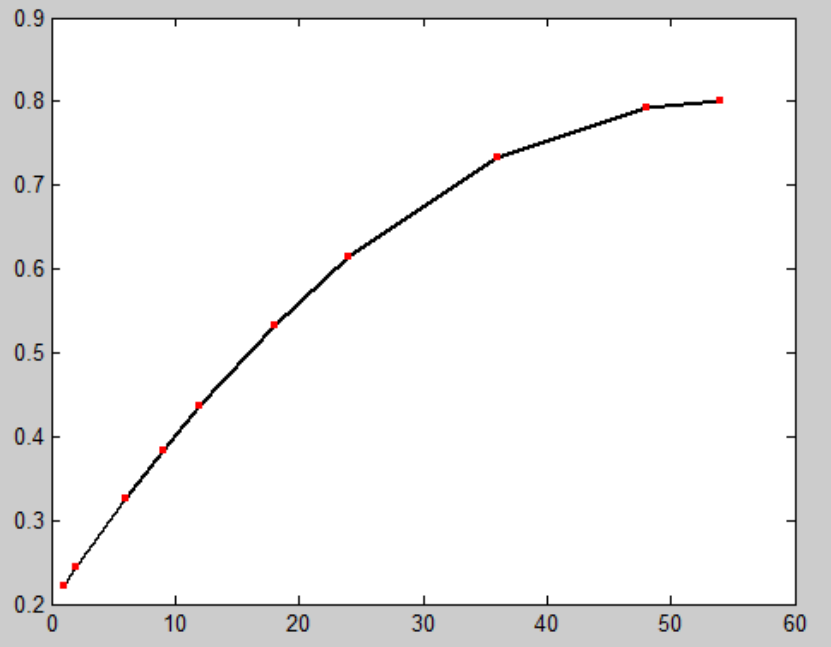


Figure 24: Γραφική αναπαράσταση όλων των ρυθμών μετάδοσης και οι τιμές K2 που αντιστοιχούν

Αν υποθέσουμε πως έχουμε 2 σταθμούς, τότε η γραφική παράσταση εκφυλίζεται σε μια γραμμή, που διέρχεται από το (0,0.2) και το (max_rate, HB). Επομένως για 2 σταθμούς μόνο, όσο πιο κοντά είναι ο ρυθμός μετάδοσης του 2^{ου}

σταθμού κοντά στο μεγαλύτερο, τόσο πιο δίκαιος θα είναι ο ανταγωνισμός για το κανάλι.

Για παραπάνω σταθμούς, μπορούμε να δούμε παρακάτω τι μορφή παίρνει η εξίσωση για διάφορους ρυθμούς μετάδοσης (βλ. Figure 24, 25,26)

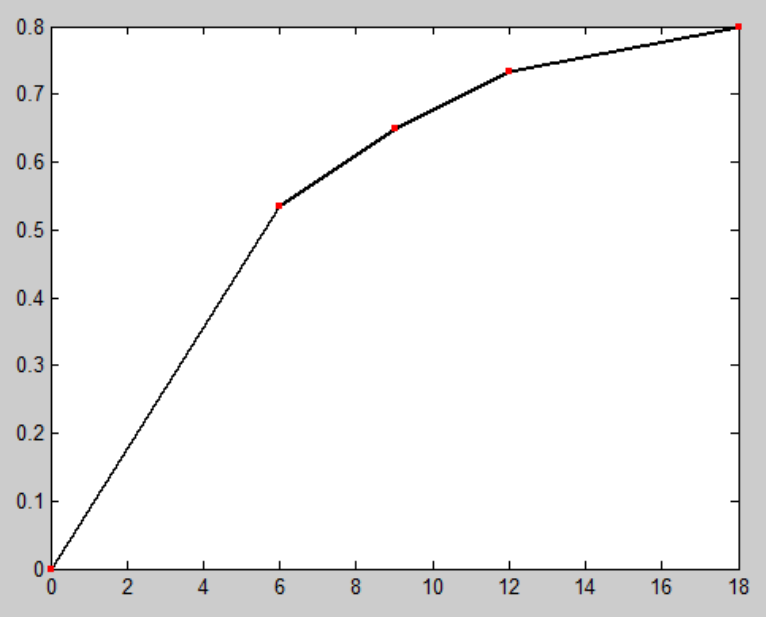


Figure 25

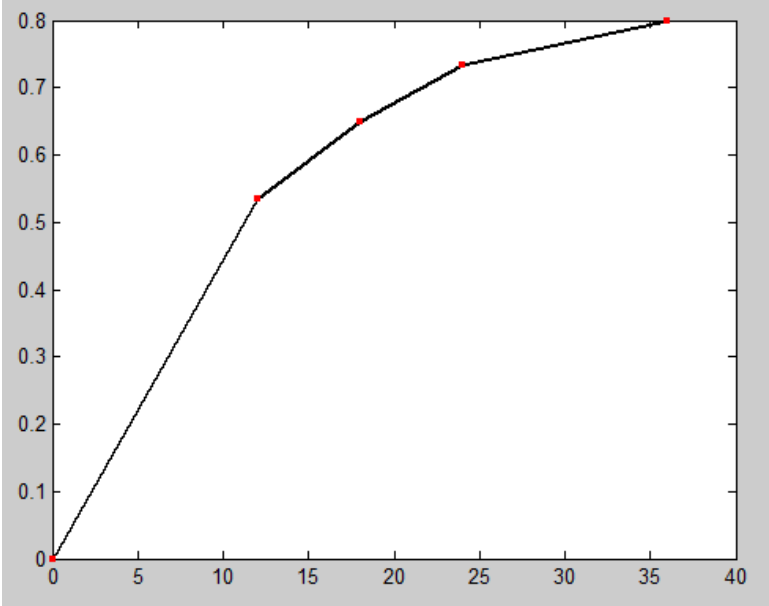


Figure 26

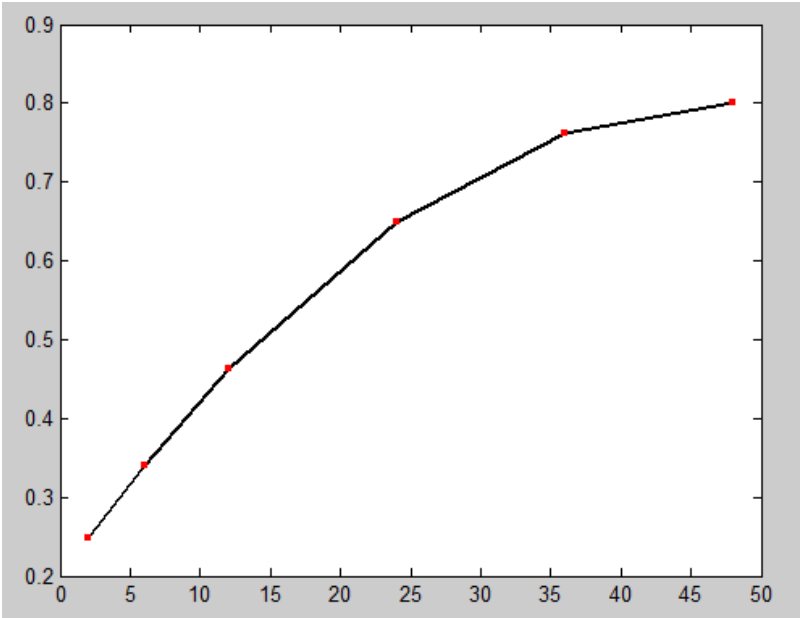


Figure 27

Τα αποτελέσματα για τη νέα αυτή τεχνική φαίνονται στον παρακάτω πίνακα. Πειράματα με δυο σταθμούς δεν έγιναν, αφού θα είχαν παρόμοια αποτελέσματα με τις προηγούμενες μεθόδους, όπου χρησιμοποιούσαμε σταθερή τιμή στις μεταβλητές μας.

Table 4: Μετρήσεις νέας τεχνικής για 3 σταθμούς

Σταθμός	Link Quality	Rate	10M	15M	20M	50M
STA1	50/70	54M	9.77M (2.37M)	13.5M (3.02M)	13.8M (3M)	13.7M (2.98M)
STA2	35/70	54M	8.68M (2.36M)	5.32M (3.32M)	5.58M (3.33M)	4.59M (3.38M)
STA3	22/70	36M	6.77M (1.51M)	7.55M (2.15M)	7.00M (2.2M)	7.90 M (2.59M)
STA1	50/70	54M	7.22M (3.56M)	7.27M (4.74M)	7.24M (4.82M)	7.34M (4.42M)
STA2	35/70	36M	7.60M (7.06M)	7.69M (5.72M)	7.70M (5.62M)	7.71M (5.72M)
STA3	22/70	18M	3.92M (3.22M)	3.91M (4.03M)	3.90M (4.11M)	3.86M (4.37M)
STA1	50/70	36M	6.52M (1.76M)	6.80 M (1.78M)	6.76M (1.76M)	
STA2	35/70	24M	4.26M (7.02M)	4.70M (11.4M)	4.66M (11.3M)	
STA3	22/70	9M	2.04M (1.3M)	1.86 M (1.38M)	1.96M (1.39M)	

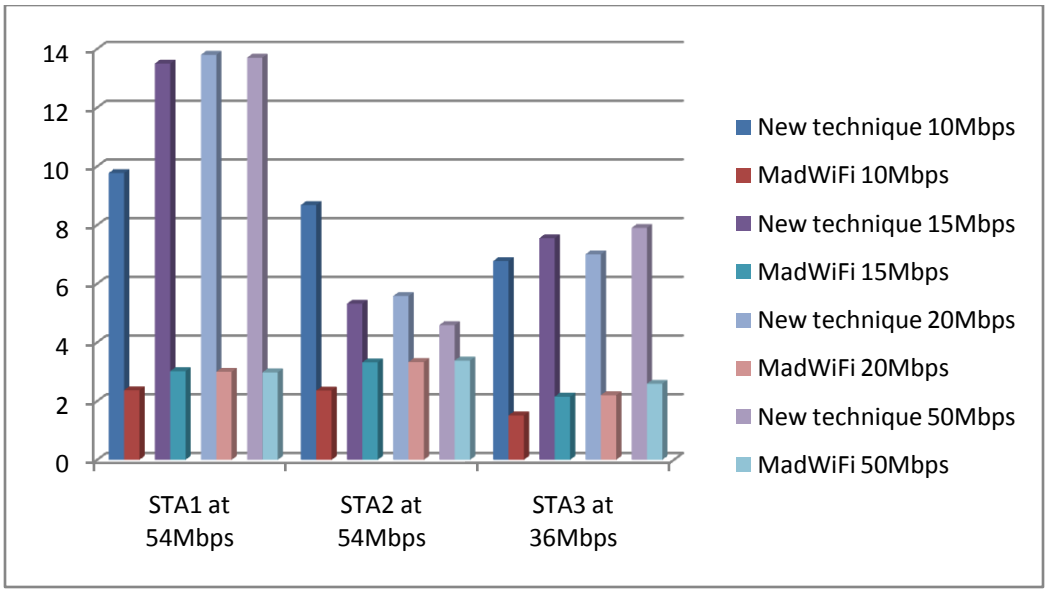


Figure 28: Αποτελέσματα επεκταμένης τεχνικής για 3 σταθμούς με διάφορα rates

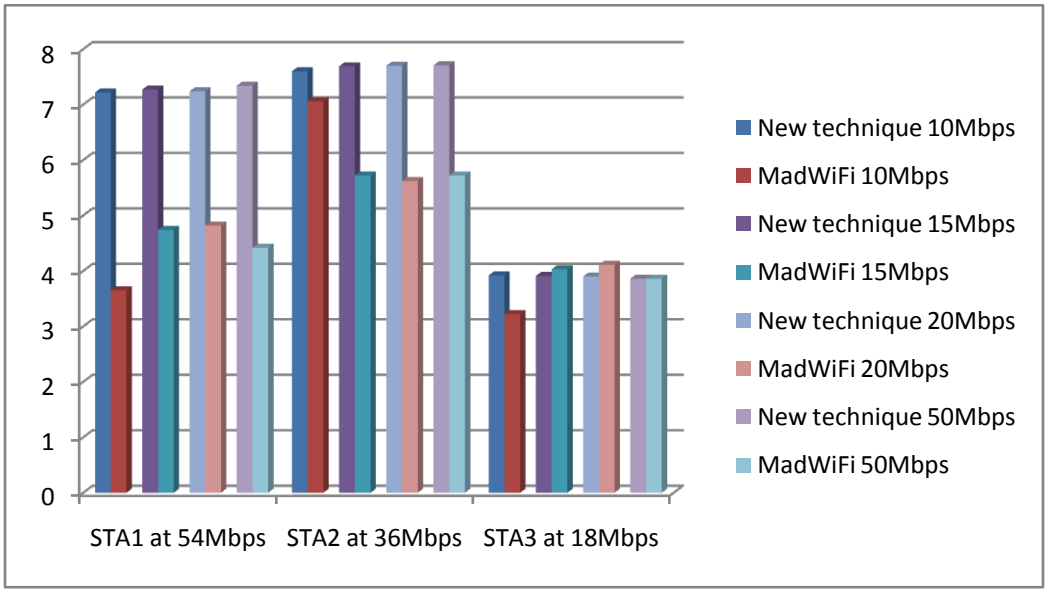


Figure 28: Αποτελέσματα επεκταμένης τεχνικής για 3 σταθμούς με διάφορα rates

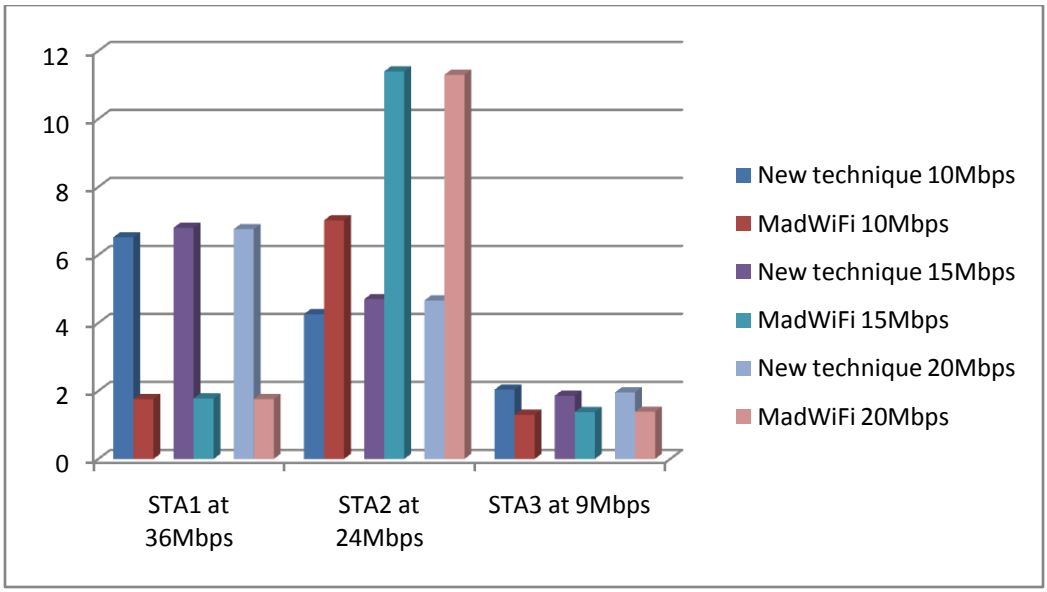


Figure 299: Αποτελέσματα νέας τεχνικής για 3 σταθμούς για διάφορα rates

Παρατηρούμε πως, για χαμηλούς σχετικά ρυθμούς μετάδοσης, οι σταθμοί που μεταδίδουν με μεγαλύτερο ρυθμό είναι αυτοί που τελικά καταφέρνουν και κερδίζουν το κανάλι για περισσότερο χρόνο. Για τις περιπτώσεις όπου έχουμε μεγαλύτερους ρυθμούς μετάδοσης, βλέπουμε πως αν οι ρυθμοί είναι πάνω από ένα όριο, συναρτήσει του μεγαλυτέρου ρυθμού στο δίκτυο, τότε οι σταθμοί που μεταδίδουν με τέτοιους ρυθμούς καταλαμβάνουν το κανάλι για περισσότερο από τους σταθμούς με χαμηλότερο rate.

Όμως, σε σχέση με το πρωτόκολλο 802.11, συνολικά καταφέρνουμε να μεταδώσουμε περισσότερη πληροφορία, άρα και μεγαλύτερο throughput. Λαμβάνοντας επιπλέον υπόψη το γεγονός πως τελικά το 802.11 ευνοεί τον σταθμό αυτόν με τη καλύτερη ποιότητα καναλιού, αφού θα έχει περισσότερες επιτυχημένες μεταδόσεις, τελικά το σχήμα μας αυτό καταφέρνει και είναι και συνολικά πιο δίκαιο.

Κεφάλαιο 6

6.1 MadWiFi Driver programming difficulties

Για τις μετρήσεις των νέων σχημάτων QoS που φτιάξαμε, επιλέχθηκε σαν πλατφόρμα ο driver ανοιχτού λογισμικού Mad-WiFi για κάρτες ασύρματου δικτύου στης Atheros. Παρά το ότι ο driver είναι ανοιχτού λογισμικού, υπάρχουν πολλά πράγματα στα οποία σαν προγραμματιστές δεν μπορούμε να έχουμε πρόσβαση, αφού όπως έχουμε αναφέρει ήδη αρκετές φορές, είναι υλοποιημένα αποκλειστικά σε hardware (όπως πχ. ACKs κ.ά.)

Η διαδικασία του backoff είναι υλοποιημένη μόνο σε hardware. Ο driver μόνο αναθέτει τις ιδιότητες στην ουρά που θα χρησιμοποιηθεί για να γίνει αποστολή του frame. Επιπλέον, το carrier sense γίνεται και αυτό στο hardware. Επομένως όταν προγραμματίζουμε τον Driver, μπορούμε να γνωρίζουμε σαν παραμέτρους τις ιδιότητες της ουράς, όπως είναι το στιγμιαίο μέγεθος της ουράς και το αρχικό Contention Window που θα χρησιμοποιηθεί για το επόμενο προς αποστολή frame. Δεν μπορούμε να γνωρίζουμε όμως πότε θα γίνει η αποστολή του επόμενου frame. Επομένως, έπρεπε να υλοποιηθεί μια τεχνική που θα άλλαζε το contention window αποδοτικά και ανεξάρτητα από το carrier sensing.

Μια πρώτη ιδέα ήταν να χρησιμοποιήσουμε έναν kernel timer, ο οποίος θα πυροδοτεί τον υπολογισμό του CW και να το αναθέτει σαν ιδιότητα στην ουρά. Ο timer θα μπορεί να έχει άμεση πρόσβαση στο στιγμιαίο μέγεθος της ουράς και τον στιγμιαίο ρυθμό μετάδοσης, επομένως θα μπορεί να αναθέσει αποδοτικά το CW. Ο timer υλοποιήθηκε, όμως οι μετρήσεις που πήραμε γι αυτόν έδειξαν πως το παραπάνω overhead που προσέθετε στην συνολική μας επικοινωνία ήταν καταστροφικό για τα αποτελέσματα μας.

Επομένως, η νέα σκέψη ήταν να υπολογίζεται και να αναθέτεται το CW κάθε φορά που έχουμε ένα νέο πακέτο προς αποστολή. Όταν το frame έχει μόλις ενθυλακωθεί, υπολογίζουμε και αναθέτουμε το CW σαν ιδιότητα της ουράς προς

αποστολή. Με αυτό τον τρόπο εξασφαλίζουμε ότι το πρώτο πακέτο που υπάρχει στην ουρά (η ουρά είναι FCFS) θα μεταδοθεί χρησιμοποιώντας το CW που υπολογίστηκε με βάση όλα τα πακέτα στην ουρά μέχρι στιγμής. Οι μετρήσεις που έγιναν έδειξαν πως το CW αλλάζει για το επόμενο πακέτο προς μετάδοση, και το overhead της διαδικασίας αυτής ήταν αμελητέο.

Ένα άλλο πρόβλημα που υπήρχε προγραμματίζοντας τον driver ήταν οι αλλαγές που κάναμε μπορεί να αλλάζονταν ή ακόμα και να κόβονταν από το HAL. Επομένως, ψάχνοντας το OpenHAL βρήκαμε που ορίζονται οι ιδιότητες της ουράς, και διορθώσαμε τον κώδικα που μας εμπόδιζε να αλλάζουμε την τιμή στο CW.

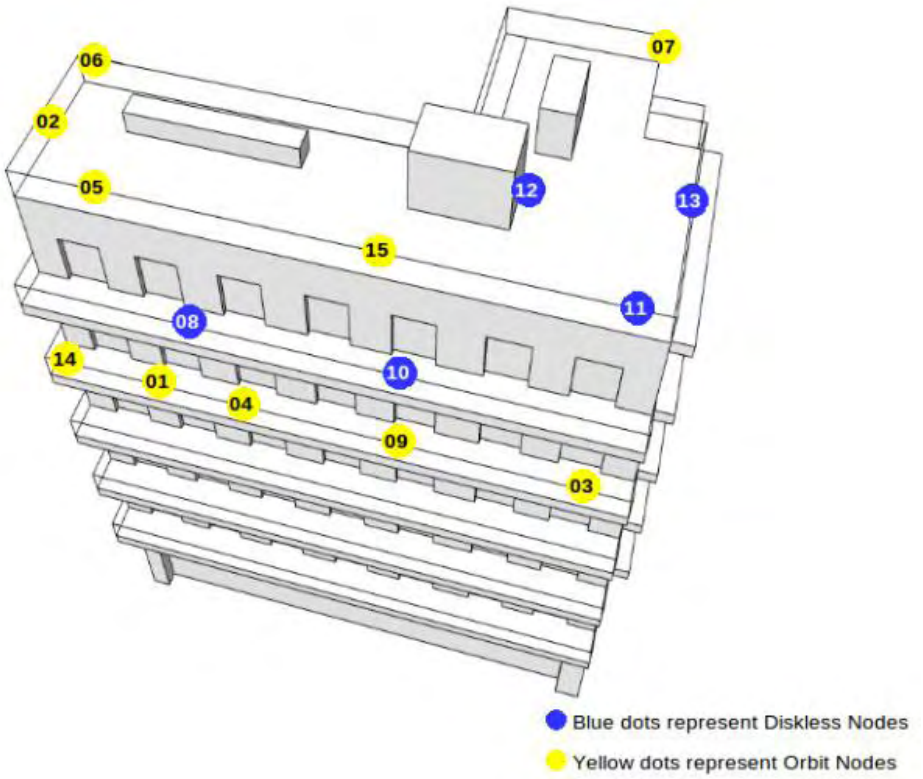
Εξαιτίας του γεγονότος ότι ολόκληρη η διαδικασία του backoff εκτελείται στο hardware, δεν ήταν δυνατόν να ελέγξουμε χρησιμοποιώντας τον driver τι τιμή είχε ο backoff timer και έτσι να ελέγξουμε αν το σχήμα μας δουλεύει τελικά. Η τιμή του backoff timer στον Mad-WiFi μπορεί να διαβαστεί μόνο στα 802.11a AR5210 chipsets. Όμως, η τιμή του backoff timer αφορά time slots, ενώ το να διαβάσουμε τον καταχωρητή που περιέχει αυτή την τιμή θα έπαιρνε παραπάνω χρόνο απ' ότι στο hardware να στείλει το πακέτο. Σαν αποτέλεσμα αυτών, έπρεπε να βρεθεί κάποιος άλλος τρόπος για να ελέγχουμε πότε ένα πακέτο έχει σταλεί και πόση ώρα πήρε για να μεταδοθεί. Για όλους αυτούς τους λόγους, χρησιμοποιήσαμε το εργαλείο για packet sniffing Whireshark. Στον κόμβο που ήταν ρυθμισμένος σαν AP, χρησιμοποιήσαμε και μια δεύτερη κάρτα σε monitor mode, ώστε να μπορεί να δέχεται όλη την εισερχόμενη κίνηση. Κάθε πακέτο που καταγράφονταν, είχε μια χρονοσφραγίδα. Ελέγχοντας τις χρονοσφραγίδες συνεχόμενων πακέτων, μπορούμε κατά εκτίμηση να δούμε αν το συγκεκριμένο frame περίμενε μόνο χρόνο DIFS για να σταλεί ή παραπάνω χρόνο, κάνοντας backoff πριν να πάρει πρόσβαση στο μέσο.

Για την υλοποίηση του νέου σχήματος επικοινωνίας που χρησιμοποιήσαμε (βλ. Κεφάλαιο 5), έπρεπε να αλλάζουμε τα beacons για να μπορούν να φέρουν την δική μας πληροφορία. Όμως δεν μπορούμε τα beacons να τα κάνουμε όσο μεγάλα θέλουμε, αφού προβλήματα όπως stuck beacons θα μπορεί να συμβούν. Επομένως,

χρησιμοποιήσαμε μόνο κάποια πεδία του 802.11 header τα οποία δεν χρησιμοποιούνται από τα beacons (όπως πχ. duration field) και υλοποιήσαμε μια κυκλική διαδικασία που «διαφημίζει» μόνο ένα σταθμό κάθε φορά. Όταν ένας συγκεκριμένος σταθμός λαμβάνει μια συγκεκριμένη ποσότητα beacons (στην υλοποίηση μας ήταν 30), στέλνει ένα Authentication πακέτο που περιέχει το rate του. Για να μπορέσει το AP να ξεχωρίσει αν το πακέτο ήταν ένα δικό μας authentication πακέτο ή ένα κανονικό, χρησιμοποιήσαμε και ένα δεύτερο πεδίο. Αυτά τα δύο πεδία έχουν υλοποιηθεί σαν μια νέα επικεφαλίδα πακέτων, που τοποθετείται στα πακέτα ακριβώς πριν προστεθεί το 802.11 header.

Συνεχίζοντας στην υλοποίηση, κάποιες μεταβλητές που πρέπει να χρησιμοποιούμε, έπρεπε να έχουν την κατάλληλη τιμή όταν θα τις χρησιμοποιήσουμε. Επομένως φτιάξαμε δύο kernel timers, ένα για το AP και ένα για τους STAs, οι οποίοι προετοιμάζουν τις μεταβλητές που θα χρησιμοποιήσουμε. Στο παράρτημα μπορεί ο αναγνώστης να βρει όλες οι αλλαγές στον κώδικα του driver, καθώς επίσης και ότι άλλο συζητήθηκε σε αυτό το κεφάλαιο.

6.2 Nitos Wireless Testbed

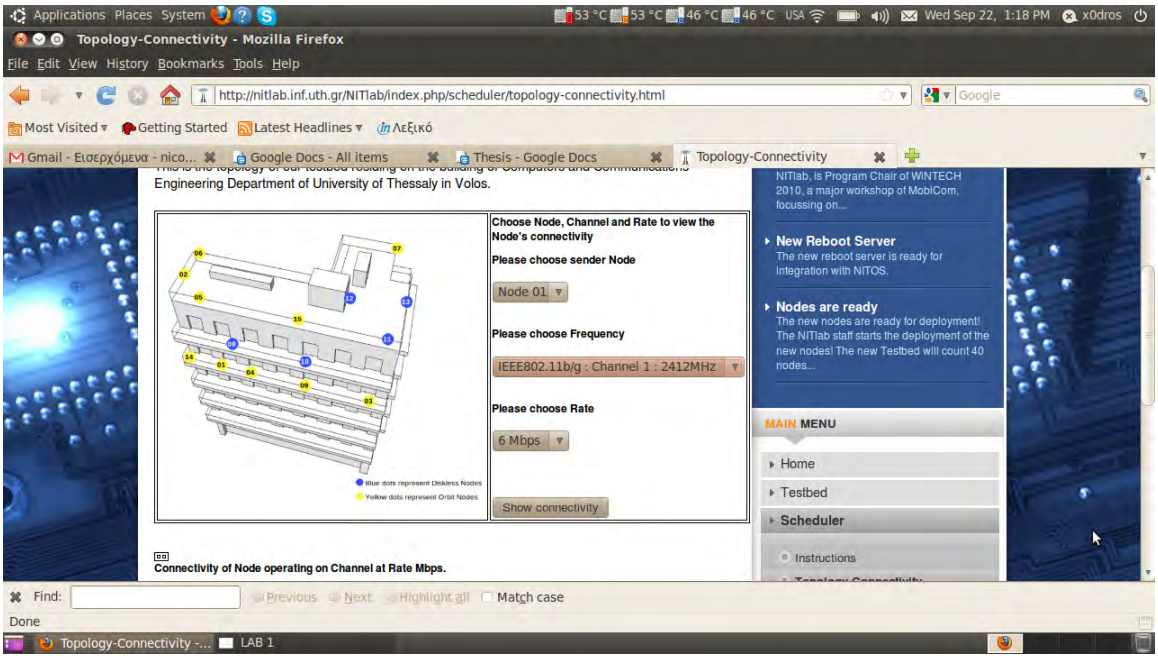


Το Nitos (Network Implementation Testbed using Open Source code) είναι μια διάταξη ασύρματων κόμβων η οποία μπορεί να χρησιμοποιηθεί για πειράματα και μετρήσεις σε πραγματικές συνθήκες και προσφέρει την ευκαιρία στους ερευνητές να εξετάσουν τα αποτελέσματα της δουλειάς τους, χωρίς την “προστασία” που προσφέρουν τα προγράμματα προσομοίωσης, εξετάζοντάς τα ως προς τη λειτουργικότητα και απόδοση απέναντι σε προβλήματα που προκύπτουν σε ρεαλιστικά σενάρια και πραγματικές καταστάσεις λειτουργίας.

Αποτελείται από 15 κόμβους, οι οποίοι αναμένεται να αυξηθούν στους 40, εγκαταστημένοι στην ταράτσα και στους 2 τελευταίους ορόφους του κτιρίου του τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων στην οδό Γκλαβάνη και αναπτύχθηκε από το Nitlab.

Οι κόμβοι αυτοί είναι 2 ειδών, Orbit και Diskless. Οι πρώτοι διαθέτουν σκληρό δίσκο ενώ οι δεύτεροι όχι. Ελέγχονται και λειτουργούν από έναν κεντρικό server μέσω switches. Το σύστημα είναι εξοπλισμένο με εργαλεία τόσο για παροχή δικτυακών υπηρεσιών όσο για τη διαχείριση του Testbed. Στην πρώτη κατηγορία ανήκει το NITOS nfs, ένα δικτυακό λειτουργικό σύστημα βασισμένο σε linux, PXE, εργαλείο που χρησιμεύει για το network booting, DHCP, DNS και APACHE. Στην δεύτερη ανήκουν το OMF (cOntrol and Management Framework), με το οποίο κανείς μπορεί να σχεδιάσει και να ελέγχει τα πειράματά του. Τέλος, το σύστημα παρέχει έναν scheduler ο οποίος αποτελεί ένα εργαλείο ενημέρωσης σχετικά με τη διαθεσιμότητα των κόμβων και των ελεύθερων συχνοτήτων ενώ παρέχει την ταυτόχρονη αξιοποίηση τους προσφέροντας διαμοιρασμό του φάσματος των συχνοτήτων. Μέσω μιας εύχρηστης διαδικτυακής διεπαφής ο ερευνητής μπορεί να δεσμεύσει κόμβους στις επιθυμητές συχνότητες και για την επιθυμητή διάρκεια η οποία για λόγους δικαιοσύνης δεν μπορεί να υπερβεί τις 4 ώρες.

Όλα αυτά είναι στη διάθεση του καθένα, αρκεί να μπει στην ιστοσελίδα του Nitlab, http://nitlab.inf.uth.gr, και κάνει εγγραφή και καταχώρηση των στοιχείων που θα του ζητηθούν. Στην παρακάτω εικόνα φαίνεται ένα κομμάτι από την διεπαφή δέσμευσης των κόμβων.



Όπως ήταν προφανές, θα ήταν ιδανικό να χρησιμοποιηθεί το Nitos για την εκτέλεση των μετρήσεων μας, πάνω στις νέες τεχνικές που αναπτύχθηκαν. Όμως, διάφορα προβλήματα ήταν αυτά που απέτρεψαν την λύση αυτή, αφού δοκιμάζοντας να τρέξουμε τα πειράματα μας, το throughput που παίρναμε ήταν πολύ χαμηλό, αφού όπως φαινόταν, δεν χρησιμοποιούνταν ποτέ πάνω από ένα πακέτο στην ουρά του συστήματος. Ακόμα και σε ιδανικές συνθήκες, χωρίς θόρυβο, σε ένα contention free περιβάλλον, το throughput εξακολουθούσε να είναι χαμηλό, πράγμα που δείχνει πως δεν έφταιγε η υλοποίηση της τεχνικής μας, αφού δεν θα χρησιμοποιούνταν ο αλγόριθμος του backoff. Ο κανονικός κώδικας madwifi που είχαμε στη διάθεση μας, χωρίς τις αλλαγές μας, έτρεχε ανορθόδοξα, αφού την μία στιγμή έδινε το σωστό throughput και την άλλη έδινε πάλι πολύ χαμηλό. Λαμβάνοντας υπόψη όλα αυτά, και χρησιμοποιώντας και ένα δεύτερο κώδικα MadWiFi με το binary HAL που έτρεξε απροβλημάτιστα, καταλήξαμε στο συμπέρασμα πως το πρόβλημα δημιουργούνταν από τον κώδικα που χρησιμοποιήθηκε. Όμως, εξαιτίας των αλλαγών που ήταν επιβεβλημένοι να γίνουν και σε επίπεδο HAL, καταλήξαμε τελικά τα πειράματα για τις μετρήσεις μας να γίνουν με απλούς υπολογιστές.

Κεφάλαιο 7

7.1 Επίλογος

Εξαιτίας της μεγάλης εξάπλωσης των ασυρμάτων δικτύων στις μέρες μας, πολλές φορές είναι απαραίτητη η διαφοροποίηση της πρόσβασης στο μέσο για κάθε σταθμό, για την περίπτωση της παροχής υπηρεσιών. Τα πρωτόκολλα που χρησιμοποιούνται κυρίως στα ασύρματα δίκτυα είναι αυτά βασισμένα στο πρότυπο IEEE 802.11. Το πρωτόκολλο αυτό, προσπαθεί να είναι δίκαιο σε σχέση με την πρόσβαση στο μέσο για τους χρήστες του. Στην πτυχιακή αυτή, ασχοληθήκαμε με την διαφοροποίηση της πρόσβασης στο μέσο για κάθε σταθμό, αλλάζοντας μια παράμετρο της διαδικασίας πρόσβασης στο μέσο του IEEE 802.11, την CW_{min}.

Η αλλαγή αυτής της παραμέτρου, έγινε με βάση το μέγεθος της ουράς κάθε σταθμού, και με βάση τον ρυθμό μετάδοσης του. Στόχος ήταν ο σταθμός με το μεγαλύτερο φόρτο, άρα και μεγαλύτερο μέγεθος ουράς, και με το καλύτερο κανάλι, άρα καλύτερο ρυθμό μετάδοσης, να έχει μεγαλύτερη πιθανότητα πρόσβασης στο μέσο. Η πρόσβαση γίνεται διαφοροποιημένα σε κάθε σταθμό, και έτσι οι δυο τεχνικές που παρουσιάστηκαν υπερτερούν του κανονικού πρωτοκόλλου IEEE802.11.

Στη συνέχεια, και λαμβάνοντας υπόψη πως η επιτυχία της δεύτερης τεχνικής μας εξαρτάται άμεσα από δύο νέες σταθερές, παρουσιάστηκε ένας νέος αλγόριθμος για τον έλεγχο αυτών των σταθερών. Με βάση αυτόν τον αλγόριθμο, έπρεπε όλοι οι σταθμοί να γνωρίζουν τον ρυθμό μετάδοσης όλως των υπολοίπων σταθμών. Επομένως, αναπτύχθηκε ένα νέο σχήμα επικοινωνίας, όπου το AP «διαφημίζει» τους σταθμούς συνδεδεμένους πάνω του, και τον ρυθμό μετάδοσης που έχει ο καθένας. Για να μπορέσει το AP να γνωρίζει αυτή την πληροφορία, υλοποιήθηκε μηχανισμός αποστολής αυτών των δεδομένων από κάθε σταθμό στο AP, ανά μια χρονική περίοδο, που ορίζεται από την λήψη beacons.

Πάνω στο σχήμα αυτό, εφαρμόσαμε ένα απλό αλγόριθμο για τον υπολογισμό των μεταβλητών αυτών, εξαρτώμενο από το μέγιστο ρυθμό μετάδοσης στο δίκτυο. Οι

μετρήσεις για την τελευταία αυτή τεχνική έδειξαν πως το σχήμα μας δουλεύει πολύ καλά για την περίπτωση μεγάλης κίνησης προς το AP, αλλά όχι τόσο καλά για τις περιπτώσεις μικρότερης κίνησης. Αυτό οφείλεται και στο αλγόριθμο που εφαρμόζουμε για την επιλογή των σταθερών που μας αφορούν. Τέλος, αναπτύχθηκε και ένας δεύτερος αλγόριθμος πάνω σε αυτό το σχήμα μετάδοσης, όπου πάλι αναλογικά με το μέγιστο ρυθμό μετάδοσης πάνω στο δίκτυο, χρησιμοποιούμε μια παραβολή με το μέγιστο σημείο της πάνω σε αυτό, και επηρεάζουμε κατάλληλα τις σταθερές σε όλους τους σταθμούς. Αν κάποιος βρίσκεται κοντά στο μέγιστο ρυθμό μετάδοσης, θα δώσει μεγαλύτερη βαρύτητα στο ρυθμό μετάδοσης. Αντίθετα, αν έχει αρκετά χαμηλότερο ρυθμό, θα δώσει μεγαλύτερο βάρος στο μέγεθος της ουράς του. Οι μετρήσεις για τη νέα αυτή τεχνική έδειξαν πως οι γρήγοροι σταθμοί καταφέρνουν να στέλνουν περισσότερα δεδομένα πάνω από το κανάλι από τους πιο αργούς, οι οποίοι αντίθετα με τις προηγούμενες τεχνικές δεν καταπνίγονται τόσο πολύ.

Για τις μετρήσεις χρησιμοποιήθηκε ο MadWiFi driver ανοιχτού κώδικα, επομένως οι μετρήσεις έγιναν σε πραγματικό περιβάλλον. Για τους ερευνητές υπάρχει διαθέσιμη η πλατφόρμα του Nitos testbed, αλλά δυστυχώς δεν ήταν εφικτό το τρέξιμο κάποιων μετρήσεων με το εργαλείο αυτό, αφού τα αποτελέσματα χαλούσαν εξαιτίας κάποιων αλλαγών στον κώδικα του MadWiFi, τις οποίες οι κόμβοι του Nitos δεν μπορούν να υποστηρίξουν. Επομένως οι μετρήσεις έγιναν με χρήση κοινών υπολογιστών, ένας ρυθμισμένος σαν AP, και ακόμα 3 να έχουν τον ρόλο του σταθμού.

7.2 Μελλοντική έρευνα

Όπως φάνηκε από την εργασία αυτή, μπορούμε επηρεάζοντας την παράμετρο του CW να μεταδίδουμε περισσότερα δεδομένα πάνω στο κανάλι, δεδομένου ότι οι σταθμοί με μεγαλύτερο ρυθμό μετάδοσης μεταδίδουν περισσότερα δεδομένα σε λιγότερο χρόνο. Σαν μελλοντική έρευνα θα μπορούσε να συνδυαστεί με δυναμική επιλογή συχνότητας, όπου ανάλογα με το RSSI του κάθε καναλιού, θα χωρίζουμε τους

αργούς από τους γρήγορους σταθμούς κατανεμημένα, αναθέτοντας τους γρήγορους σε πιο ήρεμο κανάλι, και θα εφαρμόζουμε κατάλληλα τον αλγόριθμο επιλογής CW. Βέβαια η αποτελεσματικότητα του σχήματος που παρουσιάσαμε εδώ, θα πρέπει να δοκιμαστεί ξανά σε πιο βαριές συνθήκες καναλιού, με πιο πολλούς σταθμούς, κάτι το οποίο δεν ήταν εφικτό να γίνει σε αυτή την εργασία.

Αναφορές

1. Jaeseon Hwang Hyuk Lim: Queue-based Contention Control for Congested Multihop Wireless Networks
2. Jenhui Chen and Wenchiao Wu: Dynamic Contention Window Selection Scheme to achieve a Theoretical Throughput Limit in Wireless Networks: A Fuzzy Reasoning Approach
3. Imad Aad, Qiang Ni, Chadi Barakat, Thierry Turletti: Enhancing IEEE 802.11 MAC in congested environments
4. Eustathia Ziouva and Theodore Antonakopoulos: CSMA/CA Performance under High Traffic Conditions: Throughput and Delay analysis
5. Frederico Calì, Marco Conti, Enrico Gregori: Dynamic Tuning of the IEEE 802.11 Protocol to Achieve a Theoretical Throughput Limit
6. Jian Ni, Bo Tan, and R. Srikant: Q-CSMA: Queue length Based CSMA/CA Algorithms for Achieving Maximum Throughput and Low Delay in Wireless Networks
7. Qixiang Pang, Soung Chang Liew, Jack Y. B. Lee, S.-H. Gary Chan: A TCP-like Adaptive Contention Window Scheme for WLAN
8. George W. Wong and Robert W. Donaldson: Improving the QoS Performance of EDCF in IEEE 802.11e Wireless LANs
9. Chang-Keun Seo, Weidong Wang and Sang-Jo Yoo: Load Based Dynamic Backoff Algorithm for QoS Support in Wireless Ad Hoc Networks
10. Iordanis Koutsopoulos, Oliveira Rodolfo : Maximum Throughput Access Control in Wireless LANS through Max-Weight Inspired Policies
11. Chonggang Wang, Bo Li and Lemin Li: A new collision resolution mechanism to enhance the performance of IEEE 802.11 DF
12. Luciano Bononi, Marco Conti, Enrico Gregori: Runtime optimization of IEEE 802.11 Wireless LANs Performance
13. Martin Heusse, Franck Rousseau, Romaric Guillier, Andrzej Duda: Idle sense: an optimal access method for high throughput and fairness in rate diverse wireless LANs

14. Yaling Yang, Jun Wang and Robin Kravets: Distributed optimal Contention Window control for elastic traffic in wireless LANs
15. Iordanis Koutsopoulos, Oliveira Rodolfo : Queue and Channel State Awareness for maximum Throughput Access Control in CSMA/CA-based Wireless LANs
16. Stefan Mangold, Sunghyun Choi, Peter May, Ole Klein, Guido Hiertz, Lothar Stibor : IEEE 802.11e Wireless LAN for Quality of Service
17. A technical tutorial on the IEEE 802.11 Protocol, by Pablo Brenner, director of engineer BreezeCom
18. MadWiFi summary, Wenhua Zhao

ΠΑΡΑΡΤΗΜΑ

Παρακάτω παρουσιάζονται οι αλλαγές που έγιναν στα αρχεία του MadWiFi driver για να επιτύχουμε την υλοποίηση των τεχνικών που έχουν περιγραφεί παραπάνω.

1η τεχνική:

Για την 1η μας τεχνική, οι αλλαγές στον κώδικα του MadWiFi έχουν ως εξής:

Στο αρχείο ath/if_ath.c:

(line 3554, συνάρτηση ath_hardstart): Δήλωση μεταβλητών

```
#ifdef QUEUE_RATE
    struct ieee80211com *my_ic = &sc->sc_ic;
    struct ath_txq *my_txq = sc->sc_ac2q[WME_AC_BE];
    struct wmeParams *wmep = &my_ic-
              >ic_wme.wme_chanParams.cap_wmeParams[WME_AC_BE];
    struct ath_hal *ah = sc->sc_ah;
    HAL_TXQ_INFO qi;
    int current_queue = 0;
    int current_rate = 0;
    int my_cwmin = 0;
    static int qmax = 63; //kind of a cheat here, for the
beginning of cw calculations,
                          //// use the most common length
    static int prev_cw_min = 0;
    int max_cw=1023;
    int min_cw=15;
    int Rmax=54;
    int Rmin=1;
    int Qmin=1;
    int my_const=0;


#endif
```

(line 3778, συνάρτηση ath_hardstart): Βρίσκουμε το μέγιστο μέγεθος ουράς

```
#ifdef QUEUE_RATE
                qmax=my_txq->axq_depth;
#endif
```

(line 3783, συνάρτηση ath_hardstart): Υπολογισμός cw και ανάθεση του στις ιδιότητες της ουράς

```
 #ifdef QUEUE_RATE
     if(ic->ic_opmode == IEEE80211_M_STA){


         current_queue=my_txq->axq_depth;
         current_rate= (ni->ni_rates.rs_rates[ni->ni_txrate] &
 IEEE80211_RATE_VAL)/2; //stored as double

         //following code checks mode and for given rates set
 calculates new cwmin


              //using 3 calculations to avoid floating point
 numbers, might get inaccurate results
              if(current_queue != 0){


                my_cwmin = (max_cw-min_cw)*(qmax-current_queue);

                my_const = current_queue*(qmax-1);

                my_cwmin = my_cwmin/my_const + 15;

                   //just calculating the possible new CW. Whether
 this is going to be used
                   //for transmission will be decided in HAL

                    //NOT NEEDED TAKEN ALREADY CARE OF IN THE
 EQUATION
                //  my_cwmin=MAX(15,my_cwmin);
                //  my_cwmin=MIN(my_cwmin,1024);
```

```
                    printk("new cwmin to be proposed: %u\n",
my_cwmin);
                         printk("old cwmin :%u \n",prev_cw_min);

                    if(my_cwmin != prev_cw_min){

                     my_ath_txq_update(sc, sc->sc_ac2q[WME_AC_BE],
WME_AC_BE, my_cwmin);

                         prev_cw_min=my_cwmin; //used for next
packet
                    }//end of cw comparison
               }//end of queue equal to zero

    }//end of outer if_opmode
#endif
```

(line 3485): Συνάρτηση my_ath_txq_update για την ανάθεση των τιμών στην ουρά

```
static int
my_ath_txq_update(struct ath_softc *sc, struct ath_txq *txq, int
ac, int cwmin)
{
    struct ieee80211com *ic = &sc->sc_ic;
    struct wmeParams *wmep = &ic->
         ic_wme.wme_chanParams.cap_wmeParams[ac];
    struct ath_hal *ah = sc->sc_ah;
    HAL_TXQ_INFO qi;

    ath_hal_gettxqueueprops(ah, txq->axq_qnum, &qi);
    qi.tqi_aifs = wmep->wmep_aifsn;
    qi.tqi_cwmin = cwmin;
    if(ac==WME_AC_BE) printk("if_ath ath_txq_update: cwmin =
%u\n",qi.tqi_cwmin);
    qi.tqi_cwmax = (1 << wmep->wmep_logcwmax) - 1;
    qi.tqi_burstTime = IEEE80211_TXOP_TO_US(wmep->wmep_txopLimit);

    if (!ath_hal_settxqueueprops(ah, txq->axq_qnum, &qi)) {
         EPRINTF(sc, "Unable to update hardware queue "
              "parameters for %s traffic!\n",
              ieee80211_wme_acnames[ac]);
              printk("Not able to change fucking values\n");
```

```
            return 0;
      } else {
            ath_hal_resettxqueue(ah, txq->axq_qnum); /* push to h/w */
            return 1;
      }
  }
```

Στο αρχείο ath_hal/ah.c (OpenHAL):

(line 698, συνάρτηση ath_hal_setTxQProps):

```
if (qInfo->tqi_cwmin != HAL_TXQ_USEDEFAULT) {
 #ifndef QUEUE_RATE
            cw = AH_MIN(qInfo->tqi_cwmin, 1024);
            /* make sure that the CWmin is of the form (2^n - 1) */
            qi->tqi_cwmin = 1;
            while (qi->tqi_cwmin < cw){
                  qi->tqi_cwmin = (qi->tqi_cwmin << 1) | 1;
                   printk("Hello from hal: inner while printing cwmin
      : %u\n", qInfo->tqi_cwmin);
            }
 #else
            //cw = AH_MAX(15,qInfo->tqi_cwmin);         //MOVED    TO
      DRIVER CODE
            //qi->tqi_cwmin=AH_MIN(cw,1023);      //MOVED  TO  DRIVER
      CODE
            qi->tqi_cwmin = qInfo->tqi_cwmin;
            printk("HAL:Final cwmin is: %u\n",qi->tqi_cwmin);

 #endif
 }
```

2η τεχνική:

Για την 2η τεχνική χωρίς την αυτόματη ανάθεση τιμών στις μεταβλητές K_1 και K_2 οι αλλαγές είναι οι εξής:

Στο αρχείο ath/if_ath.c:

(line 3554, συνάρτηση ath_hardstart): Δήλωση μεταβλητών

```
#ifdef QUEUE_RATE

            struct ath_txq *my_txq = sc->sc_ac2q[WME_AC_BE];
            int current_queue = 0;
            int current_rate = 0;
            int my_cwmin = 0;
            static int qmax = 63; //for the beginning of cw
                                  //  calculations,
                                   // use the most common length
            static int prev_cw_min = 0;
            int min_cw=15;
            int max_rate=54;
            int k1=5;//Divide by 10 before using
            int k2=5;//Divide by 10 before using
            int i=0;

#endif
```

(line 3778, συνάρτηση ath_hardstart): Βρίσκουμε το μέγιστο μέγεθος ουράς

```
if (bf == NULL) {

      ATH_TXQ_UNLOCK_IRQ_EARLY(txq);
                  /* All DMA buffers full, safe to try again. */

//WILL BE USED TO GIVE SOME DYNAMIC BEHAVIOR TO CW ADAPTATION

//WILL EXTRACT IMPLICITILY MAX_SIZE OF QUEUE
```

```
#ifdef QUEUE_RATE

          qmax=my_txq->axq_depth;

#endif

          requeue = 1;
          goto hardstart_fail;
     }
```

(line 3783, συνάρτηση ath_hardstart): Υπολογισμός cw και ανάθεση του στις ιδιότητες της ουράς

```
#ifdef QUEUE_RATE

     if(ic->ic_opmode == IEEE80211_M_STA){

           nt = ni->ni_table;
           current_queue=my_txq->axq_depth;
           current_rate=  (ni->ni_rates.rs_rates[ni->ni_txrate]  &
     IEEE80211_RATE_VAL)/2; //stored as double

            printk("CURRENT queue length: %u CURRENT rate: %u\n",
                                    current_queue, current_rate);

             //calculation of new cwmin

           if(current_queue != 0){

               my_cwmin = ((qmax/current_queue)*k1*min_cw)/10 +
                          ((max_rate/current_rate)*k2*min_cw)/10;

                         //just calculating the possible new CW.
                          //Whether this is going to be used
                        //for transmission will be decided in HAL

              printk("new cwmin to be proposed: %u, Old
                           cwmin :%u \n", my_cwmin, prev_cw_min);

               if(my_cwmin != prev_cw_min){

                    my_ath_txq_update(sc, sc->sc_ac2q[WME_AC_BE],
                            WME_AC_BE, my_cwmin);

                    prev_cw_min=my_cwmin; //used for next packet
```

```
                    }//end of cw comparison
                }//end of queue equal to zero

            }//end of outer if_opmode
#endif
```

(line 3485): Συνάρτηση my_ath_txq_update για την ανάθεση των τιμών στην ουρά

```
  static int
  my_ath_txq_update(struct ath_softc *sc, struct ath_txq *txq, int
  ac, int cwmin)
  {
      struct ieee80211com *ic = &sc->sc_ic;
      struct wmeParams *wmep = &ic->
            ic_wme.wme_chanParams.cap_wmeParams[ac];
      struct ath_hal *ah = sc->sc_ah;
      HAL_TXQ_INFO qi;

      ath_hal_gettxqueueprops(ah, txq->axq_qnum, &qi);
      qi.tqi_aifs = wmep->wmep_aifsn;
      qi.tqi_cwmin = cwmin;
      if(ac==WME_AC_BE) printk("if_ath ath_txq_update: cwmin =
  %u\n",qi.tqi_cwmin);
      qi.tqi_cwmax = (1 << wmep->wmep_logcwmax) - 1;
      qi.tqi_burstTime = IEEE80211_TXOP_TO_US(wmep->wmep_txopLimit);

      if (!ath_hal_settxqueueprops(ah, txq->axq_qnum, &qi)) {
            EPRINTF(sc, "Unable to update hardware queue "
                  "parameters for %s traffic!\n",
                  ieee80211_wme_acnames[ac]);
                  printk("Not able to change fucking values\n");
            return 0;
      } else {
            ath_hal_resettxqueue(ah, txq->axq_qnum); /* push to h/w */
            return 1;
      }
  }
```

Στο αρχείο ath_hal/ah.c (OpenHAL):

(line 698, συνάρτηση ath_hal_setTxQProps):

```
if (qInfo->tqi_cwmin != HAL_TXQ_USEDEFAULT) {
```

```
#ifndef QUEUE_RATE
          cw = AH_MIN(qInfo->tqi_cwmin, 1024);
          /* make sure that the CWmin is of the form (2^n - 1) */
          qi->tqi_cwmin = 1;
          while (qi->tqi_cwmin < cw){
                qi->tqi_cwmin = (qi->tqi_cwmin << 1) | 1;
                 printk("Hello from hal: inner while printing cwmin
    : %u\n", qInfo->tqi_cwmin);
          }
#else
          //cw = AH_MAX(15,qInfo->tqi_cwmin);         //MOVED    TO
    DRIVER CODE
          //qi->tqi_cwmin=AH_MIN(cw,1023);      //MOVED  TO  DRIVER
    CODE
          qi->tqi_cwmin = qInfo->tqi_cwmin;
          printk("HAL:Final cwmin is: %u\n",qi->tqi_cwmin);

#endif
}
```

2η τεχνική-επέκταση:

Για την τελευταία τεχνική, με την ανταλλαγή των μηνυμάτων πάνω από το δίκτυο για τις μεταβλητές μας K_1 και K_2, ο κώδικας που θα χρειαστούμε είναι αυτός της δεύτερης τεχνικής μαζί με τα παρακάτω:

Στο αρχείο ath/if_ath.c:

(line 364): Ορισμός του prototype της συνάρτησης για τον timer.

```
#ifdef QUEUE_RATE
static void host_calc(unsigned long arg);
#endif
```

(line 859, συνάρτηση ath_attach): Αρχικοποίηση timer.

```
#ifdef QUEUE_RATE
      init_timer(&sc->host_timer);
      sc->host_timer.function = host_calc;
      sc->host_timer.data =  (unsigned long) ic;
      sc->host_timer.expires = jiffies + 1000;
      add_timer(&sc->host_timer);
#endif
```

(line 1210, συνάρτηση ath_detach): Διαγραφή timer.

```
#ifdef QUEUE_RATE
      del_timer_sync(&sc->host_timer);
#endif
```

(line 3783, συνάρτηση ath_hardstart): Αλλαγή στον υπολογισμό του cw και ανάθεση του στις ιδιότητες της ουράς

```
#ifdef QUEUE_RATE
      if(ic->ic_opmode == IEEE80211_M_STA){
             nt = ni->ni_table;
             current_queue=my_txq->axq_depth;
             current_rate=  (ni->ni_rates.rs_rates[ni->ni_txrate]  &
      IEEE80211_RATE_VAL)/2; //stored as double
              printk("CURRENT queue length: %u CURRENT rate: %u\n",
                                        current_queue, current_rate);
               //calculation of new cwmin
            if(current_queue != 0){
```

```
                    if(current_rate>=ic->max_rate){
                        k1=2;
                        k2=8;
                    }
                    else if(current_rate>(5*ic->max_rate)/10){
                        k1=4;
                        k2=6;
                    }
                    else{
                        k1=7;
                        k2=3;
                    }

                  my_cwmin = ((qmax/current_queue)*k1*min_cw)/10 +
                             ((max_rate/current_rate)*k2*min_cw)/10;

                            //just calculating the possible new CW.
                             //Whether this is going to be used
                            //for transmission will be decided in HAL

                  printk("new cwmin to be proposed: %u, Old
                              cwmin :%u \n", my_cwmin, prev_cw_min);

                  if(my_cwmin != prev_cw_min){

                        my_ath_txq_update(sc, sc->sc_ac2q[WME_AC_BE],
                                WME_AC_BE, my_cwmin);

                        prev_cw_min=my_cwmin; //used for next packet
                      }//end of cw comparison
                 }//end of queue equal to zero

            }//end of outer if_opmode
#endif
```

(line 13063): Συνάρτηση Host_calc

```
#ifdef QUEUE_RATE
static void host_calc(unsigned long arg){

      struct ath_softc *sc = (struct ath_softc *)arg;

       struct ieee80211com *ic = &sc->sc_ic;
      struct ieee80211_node *cur=NULL;
      struct ieee80211_node_table *nt=NULL;
      int i=0;



      if(ic->ic_opmode == IEEE80211_M_HOSTAP){
        IEEE80211_LOCK_IRQ(ic);
            nt = &ic->ic_sta;

//////////////////////////////////////////////////////////////
//////////////////////////////
        //printk("******Try #1 to print all
rates********\n");

      IEEE80211_NODE_TABLE_LOCK_IRQ(nt);
      TAILQ_FOREACH(cur, &nt->nt_node, ni_list) {


  //printk("Sta "MAC_FMT" transmit rate:",MAC_ADDR(cur-
>ni_macaddr));

 // printk("%u \n", (cur->ni_rates.rs_rates[cur->ni_txrate] &
IEEE80211_RATE_VAL)/2);

  if(!IEEE80211_ADDR_EQ(cur->ni_macaddr,cur->ni_vap->iv_myaddr)){

       IEEE80211_ADDR_COPY(ic->mac[i],cur->ni_macaddr);

      i++;
      if(i==32){
           break;
      }

     }
    }

       IEEE80211_NODE_TABLE_UNLOCK_IRQ(nt);
```

```
        ic->length=i-1;


             //end of outer if_opmode

             IEEE80211_UNLOCK_IRQ(ic);
      //printk("End of my timer\n");
          //if(ic->ic_opmode == IEEE80211_M_HOSTAP){
             mod_timer(&sc->host_timer,msecs_to_jiffies(5000));
          //}
           }
           else{

             //an einai stathmos ypologizw me ayto to timer to max
      rate twn ypoloipwn

             //u_int8_t prev;
             ic->max_rate=ic->rates[0];

             for(i=0; i<32; i++){

                 if(ic->rates[i]>ic->max_rate) ic->max_rate=ic-
      >rates[i];

             }

             mod_timer(&sc->host_timer,msecs_to_jiffies(1500));
           }



      }
#endif
```

<u>Στο αρχείο ath/if_athvar.h</u>:

<u>(line 863, struct ath_softc):</u> Ορισμός του timer μας

```
#ifdef QUEUE_RATE

      struct timer_list host_timer;

#endif
```

Στο αρχείο net80211/ieee80211.h: (ορισμός του frame που θα χρησιμοποιήσουμε για τα authentication πακέτα μας)

```
#define QUEUE_RATE

#ifdef QUEUE_RATE

      struct my_frame{

       u_int8_t rate;

       u_int8_t flag;

      }__packed;

#endif
```

Στο αρχείο net80211/ieee80211_beacon.c:

(line 265, συνάρτηση ieee80211_beacon_alloc): Αλλαγή του beacon μας για να ξεχωρίζει

```
 #ifndef QUEUE_RATE
     wh->i_dur = 0;
 #else
     wh->i_dur = 3;
 #endif
```

(line 286, συνάρτηση ieee80211_beacon_update): Αλλαγή των πεδίων του beacon που μας ενδιαφέρουν

```
#ifdef QUEUE_RATE

           static int cnt=0;
           struct ieee80211_frame *wh;
#endif
```

(line 595):

```
#ifdef QUEUE_RATE

  if(ic->length>=0){

    wh = (struct ieee80211_frame *)skb->data;

      if(cnt>ic->length){

        cnt=0;
      }


      if(ic->mac[cnt]!=NULL){
        IEEE80211_ADDR_COPY(wh->i_addr1, ic->mac[cnt]);
        wh->i_dur = ic->rates[cnt];


        printk("Stelnw beacon thn MAC "MAC_FMT" transmit rate: ",
  MAC_ADDR(wh->i_addr1));
        printk("%u \n", wh->i_dur);
      }

      cnt++;
      printk("cnt %u \n", cnt);

     }
 #endif
```

<u>Στο αρχείο net80211/ieee80211_input.c</u>:

<u>(line 2960, συνάρτηση ieee80211_recv_mgmt):</u> Δήλωση μεταβλητής που θα κρατάει τον αριθμό των εισερχόμενων beacons

```
#ifdef QUEUE_RATE

       static int cnt=0;
       int i=0;
       int flag=0;
       u_int8_t max_rate = 0;
#endif
```

(line 2984, συνάρτηση ieee80211_recv_mgmt): Χειρισμός δικών μας beacons

```
#ifdef QUEUE_RATE
        if(((subtype==IEEE80211_FC0_SUBTYPE_BEACON))){
           printk("Received my beacon \n");
           wh = (struct ieee80211_frame *)skb->data;

           printk("Station "MAC_FMT" transmits using %u Mbps\n",
      MAC_ADDR(wh->i_addr1), wh->i_dur);
           cnt++;
           max_rate=ic->rates[0];
            for(i=0; i<32; i++){
                if(IEEE80211_ADDR_EQ(wh->i_addr1, ic->mac[i])){
                        ic->rates[i]=(u_int8_t) wh->i_dur;
                        flag=1;
                        break;
                      }
           }
           if(flag!=1){
              IEEE80211_ADDR_COPY(ic->mac[ic->length],wh->i_addr1);
              ic->rates[ic->length]=wh->i_dur;
              ic->length++;
              break;
           }
           ic->max_rate=max_rate;
           wh->i_dur = (u_int8_t) 0;

           if(cnt==30){
             //printk("Going to send my auth packet\n");
             cnt=0;
             ic->mgmt_flag=1;
             IEEE80211_SEND_MGMT(ni, IEEE80211_FC0_SUBTYPE_AUTH,
      0);
           }

           frm = (u_int8_t *)&wh[1];
           efrm = skb->data + skb->len;
         }
#endif
```

(line 3566, συνάρτηση ieee80211_recv_mgmt): Χειρισμός δικών μας authentication πακέτων

```
case IEEE80211_FC0_SUBTYPE_AUTH: {
          u_int16_t algo, seq, status;
          /*
           * auth frame format
           *    [2] algorithm
           *    [2] sequence
           *    [2] status
           *    [tlv*] challenge
           */

#ifdef QUEUE_RATE
           struct my_frame *mf;
           mf = (struct my_frame *) (skb-
    >data+ieee80211_hdrsize(wh));
           wh = (struct ieee80211_frame *)skb->data;

           if(mf->flag==102){
          printk("Got auth req. My mgmt packet\n");
          printk("Station "MAC_FMT" transmits at %u Mbps \n",
    MAC_ADDR(wh->i_addr2), mf->rate);
          int i=0;

              for(i=0; i<32; i++){
                   if(IEEE80211_ADDR_EQ(wh->i_addr2, ic->mac[i])){
                     ic->rates[i]=(u_int8_t) mf->rate;
                      break;
                   }
                   if(ic->mac[i]==NULL){
                    IEEE80211_ADDR_COPY(ic->mac[i],wh->i_addr1);
                    ic->rates[i]=mf->rate;
                    break;
                   }
                }


          return 0;
           }

 #endif
```

Στο αρχείο net80211/ieee80211_output.c:

(line 424, συνάρτηση ieee80211_mgmt_output): Δήλωση μεταβλητών

```
#ifdef QUEUE_RATE
      struct my_frame *mf;
#endif
```

(line 432, συνάρτηση ieee80211_mgmt_output): Κατασκευή του δικού μας authentication πακέτου

```
#ifdef QUEUE_RATE

        if(ic->mgmt_flag==1){
        ic->mgmt_flag=0;
        mf = (struct my_frame *)
      skb_push(skb, sizeof(struct my_frame));
        mf->rate=(ni->ni_rates.rs_rates[ni->ni_txrate] &
  IEEE80211_RATE_VAL)/2;
        mf->flag=102;
        printk("Sending my auth packet at %u Mbps\n",mf->rate);
        }
 #endif
```

Στο αρχείο net80211/ieee80211_var.h:

(line 533, struct ieee80211_com): Δήλωση μεταβλητών που θα χρειαστούν για τους υπολογισμούς στους timers και για την αποστολή του authentication πακέτου

```
#ifdef QUEUE_RATE
    int mgmt_flag;
    u_int8_t max_rate;
    u_int8_t mac[32][IEEE80211_ADDR_LEN];
    u_int8_t rates[32];
    int length;
#endif
```

3η τεχνική-επέκταση:

Για την τελευταία τεχνική που υλοποιήσαμε, θα κρατήσουμε όλες τις αλλαγές που κάναμε στην προηγούμενη ακριβώς τεχνική, αλλά θα αλλάξουμε στο σημείο όπου αλλάζουμε το CW και θα προσθέσουμε τη συνάρτηση υπολογισμού των δύο μεταβλητών K_1 και K_2.

Στο αρχείο ath/if_ath.c:

(line 3783, συνάρτηση ath_hardstart): Αλλαγή στον υπολογισμό του cw και ανάθεση του στις ιδιότητες της ουράς

```
#ifdef QUEUE_RATE
  if(ic->ic_opmode == IEEE80211_M_STA){

    nt = ni->ni_table;
      current_queue=my_txq->axq_depth;
      current_rate= (ni->ni_rates.rs_rates[ni->ni_txrate] &
  IEEE80211_RATE_VAL)/2; //stored as double

      //printk("CURRENT queue length: %u CURRENT rate: %u\n",
  current_queue, current_rate);


      //calculation of new cwmin

     //printk("ic->max_rate is %u\n", ic->max_rate);

         if( (current_queue != 0) && (ic->max_rate != 0) ){

          a=60000/((ic->max_rate)*(ic->max_rate));
          b=2*(ic->max_rate)*a;

          k2=c+b*(current_rate)-a*(current_rate)*(current_rate);
          k1=100000-k2;
          //printk("k2 %d \n, k1 %d \n", k2,k1);

           my_cwmin = (((qmax/current_queue)*k1*min_cw) +
  ((max_rate/current_rate)*k2*min_cw))/100000;


              //just calculating the possible new CW. Whether this
  is going to be used
             //for transmission will be decided in HAL
```

```
            //printk("new cwmin to be proposed: %u, Old cwmin :%u
  \n", my_cwmin, prev_cw_min);

            if(my_cwmin != prev_cw_min){

                  my_ath_txq_update(sc, sc->sc_ac2q[WME_AC_BE],
  WME_AC_BE, my_cwmin);

                        prev_cw_min=my_cwmin; //used for next packet
           }//end of cw comparison
        }//end of queue equal to zero

  }//end of outer if_opmode
#endif
```