ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Υποστήριξη παραλλήλων μεταδόσεων πακέτων σε "Multi – hop" Ασύρματα Δίκτυα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτρης Α. Παπαφιλίππου

Επιβλέπων : Συμεών Παπαβασιλείου
Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2005

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Υποστήριξη παραλλήλων μεταδόσεων πακέτων σε "Multi – hop" Ασύρματα Δίκτυα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτρης Α. Παπαφιλίππου

Επιβλέπων : Συμεών Παπαβασιλείου
Επίκουρος Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 2005.

.....	.....	.....
Παπαβασιλείου Συμεών	Μάγκλαρης Βασίλειος	Θεολόγου Μιχαήλ
Επίκουρος Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2005

.....

Δημήτρης Α. Παπαφιλίππου

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περιεχόμενα

Πίνακας σχημάτων

Περίληψη

Τα ασύρματα LANs (WLANs) που χρησιμοποιούν το πρωτόκολλο IEEE 802.11 MAC έχουν γίνει ένας από τους πλέον διαδεδομένους τρόπους για σύνδεση σε backbone infrastructure. Ωστόσο το πρωτόκολλο αυτό δε σχεδιάστηκε για multi-hop δίκτυα. Παρόλο που μπορεί να υποστηρίξει μερικές ad hoc αρχιτεκτονικές, δεν προορίζεται για να υποστηρίξει ασύρματα κινητά ad hoc δίκτυα, στα οποία η multi-hop διασύνδεση είναι το κύριο χαρακτηριστικό.

Το MACA-P είναι ένα πρωτόκολλο, το οποίο έχει προταθεί για την επίλυση αυτού του προβλήματος. Το πρωτόκολλο αυτό διατηρεί το βασικό μηχανισμό του 802.11 πρωτοκόλλου. Διατηρεί και βελτιώνει την τετραμερή χειραψία (RTS/CTS/DATA/ACK) καθώς επίσης και τη χρήση του exponential random backoff αλγορίθμου. Βασικός σχεδιαστικός στόχος του MACA-P είναι να επιτρέψει παράλληλες μεταδόσεις όταν αυτό είναι εφικτό, αυξάνοντας έτσι την απόδοση σε multi hop δίκτυα.

Σε αυτή τη διπλωματική εργασία έχουν μελετηθεί οι αιτίες που οδηγούν στη μειωμένη απόδοση του IEEE 802.11. Στη συνέχεια μελετήθηκε το πρωτόκολλο MACA-P και αναλύθηκαν όλες οι τροποποιήσεις και οι προσθήκες του πρωτοκόλλου αυτού σε σχέση με το 802.11. Επίσης, έχει αναπτυχθεί υλοποίηση του MACA-P στην πλατφόρμα προσομοίωσης ns. Με τη χρήση αυτού του κώδικα έχουν προσομοιωθεί σενάρια ασύρματης μετάδοσης σε ad hoc δίκτυα και γίνεται σύγκριση της ρυθμαπόδοσης των δύο πρωτοκόλλων, του MACA-P και του 802.11.

Λέξεις Κλειδιά

MACA-P, 802.11, multi-hop, ad hoc, RTS, CTS, ns, parallel transmission, wireless networks

Abstract

IEEE 802.11-based wireless LANs (WLANs) are clearly becoming a popular way for connecting to the backbone infrastructure. However, this protocol was not designed for multi-hop networks. Although it can support some ad hoc network architecture, it is not intended to support the wireless mobile ad hoc networks, in which multi-hop connectivity is one of the most prominent features.

MACA-P is a protocol that has been proposed for the resolution of the parallel transmission problem. This protocol maintains the basic functionality of the 802.11 protocol. It maintains and improves the 4-way handshake (RTS/CTS/DATA/ACK) and the use of exponential random backoff algorithm. Fundamental design aim of the MACA-P protocol is to allow parallel transmissions when they are feasible, increasing the throughput in multi hop networks.

In this diploma thesis, we have studied the causes of the decreased throughput of the IEEE 802.11 basic mechanism in multi-hop networks. We have also studied the MACA-P protocol and analyzed all the modifications and the additions of this protocol concerning the 802.11. Also, we have developed a MACA-P model for the ns network simulation platform. We have used this model to simulate scenarios of wireless transmissions in multi-hop ad hoc networks, and to perform a comparison of the two protocols, MACA-P and 802.11, based on the throughput they achieved.

Keywords

MACA-P, 802.11, multi-hop, ad hoc, RTS, CTS, ns, parallel transmission, wireless networks

Εισαγωγή

Το πρωτόκολλο MAC 802.11 το οποίο χρησιμοποιείται στα ασύρματα δίκτυα έχει γίνει το πιο διαδεδομένο πρωτόκολλο ασυρμάτων δικτύων. Το πρωτόκολλο αυτό σχεδιάστηκε για single hop δίκτυα και παρουσιάζει σημαντικές απώλειες στην απόδοση του όταν χρησιμοποιηθεί σε multi hop ad hoc ασύρματα δίκτυα. Πολλές μελέτες και προσπάθειες έγιναν μέχρι τώρα για σχεδιασμό ενός πρωτοκόλλου που να επιτρέπει να γίνονται παράλληλες μεταδόσεις σε αυτά τα δίκτυα, χωρίς όμως να προταθεί κάποιο πρωτόκολλο που να λύνει το πρόβλημα παράλληλης μετάδοσης και να αυξάνει τη ρυθμαπόδοση. Πρόσφατα προτάθηκε ένα πρωτόκολλο, το MACA-P, το οποίο υλοποιήθηκε για να επιτρέπει τις παράλληλες μεταδώσεις για κάποιες συγκεκριμένες περιπτώσεις.

Στην παρούσα διπλωματική, αντικειμενικός στόχος ήταν η μελέτη των δυο πρωτοκόλλων, του 802.11 και του MACA-P, και η υλοποίηση ενός κώδικα για το MACA-P, για να μπορέσουμε προσομοιώνοντας κάποια σενάρια, να διαπιστώσουμε αν το MACA-P, επιτρέπει όντως παράλληλες μεταδόσεις και βελτιώνει τη συνολική ρυθμαπόδοση του δικτύου.

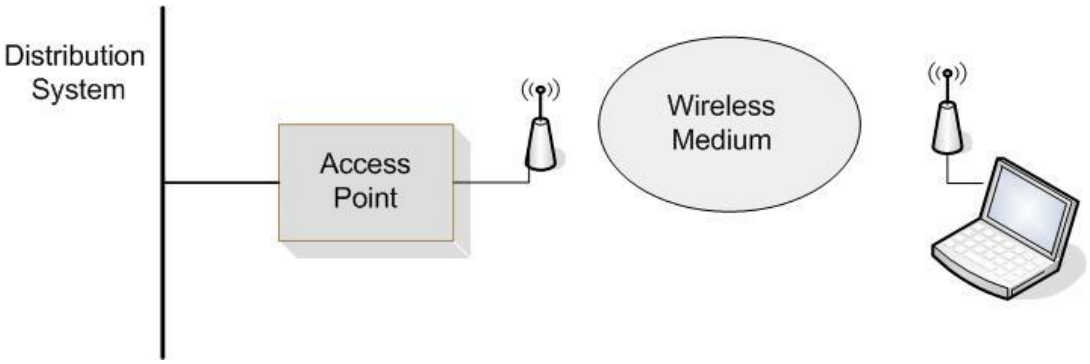
Στα δύο πρώτα κεφάλαια, γίνεται μια περιγραφή των δύο πρωτοκόλλων και εξηγούνται οι αιτίες που οδηγούν στην μειωμένη απόδοση του 802.11. Το τρίτο κεφάλαιο αναφέρεται στην υλοποίηση του κώδικα του MACA-P . Στο τέταρτο και τελευταίο κεφάλαιο αναφέρονται όλα τα σενάρια της προσομοίωσης και παρατίθενται τα αποτελέσματα καθώς και ο σχολιασμός των αποτελεσμάτων και τα συμπεράσματα της προσομοίωσης. Στα παραρτήματα υπάρχει ο κώδικας τόσο του MACA-P, όσο και των σεναρίων που προσομοιώθηκαν σε αυτή την εργασία.

Κλείνοντας την εισαγωγή, θα ήθελα να ευχαριστήσω θερμά τον σύμβουλο μου Καθηγητή Συμεών Παπαβασιλείου και τον υποψήφιο διδάκτορα Δημήτρη Βελένη για την πολύτιμη βοήθεια και καθοδήγηση που μου έδωσαν καθ’ όλη την διάρκεια της διπλωματικής αυτής εργασίας.

1. Το πρωτόκολλο 802.11

Το πρωτόκολλο 802.11 είναι ένα από τα πιο διαδεδομένα πρωτόκολλα που χρησιμοποιούνται για επικοινωνία σε ασύρματα δίκτυα. Σε αυτό το κεφάλαιο θα γίνει μια γενική περιγραφή του πρωτοκόλλου καθώς και μελέτη κάποιων βασικών μηχανισμών που χρησιμοποιεί το πρωτόκολλο αυτό.

Τα 802.11 δίκτυα περιλαμβάνουν 4 βασικά φυσικά επίπεδα (component), όπως φαίνεται και στο σχήμα 1.1.



Σχήμα 1.1: Τα μέρη του 802.11 LAN

Τα 4 αυτά επίπεδα είναι τα εξής:

- **Συστήματα Διανομής (Distribution System DS):** αποτελούν το λογικό συστατικό που χρησιμοποιείται για τη διασύνδεση πολλών σημείων πρόσβασης AP, για την επέκταση του δικτύου.

- **Σημείο Πρόσβασης (Access Point AP):** αποτελείται από τον Σταθμό Βάσης ο οποίος ελέγχει την κυψέλη του δικτύου. Οι λειτουργίες του σημείου πρόσβασης αποτελούν τη γέφυρα μεταξύ του ασύρματου σταθμού STA και του υπόβαθρου για την πρόσβαση στο δίκτυο. Το σημείο γεφυρώνει την επικοινωνία μεταξύ δύο ή περισσότερων STA. Οι συνδέσεις μεταξύ του AP μπορεί να είναι ενσύρματες ή ασύρματες.

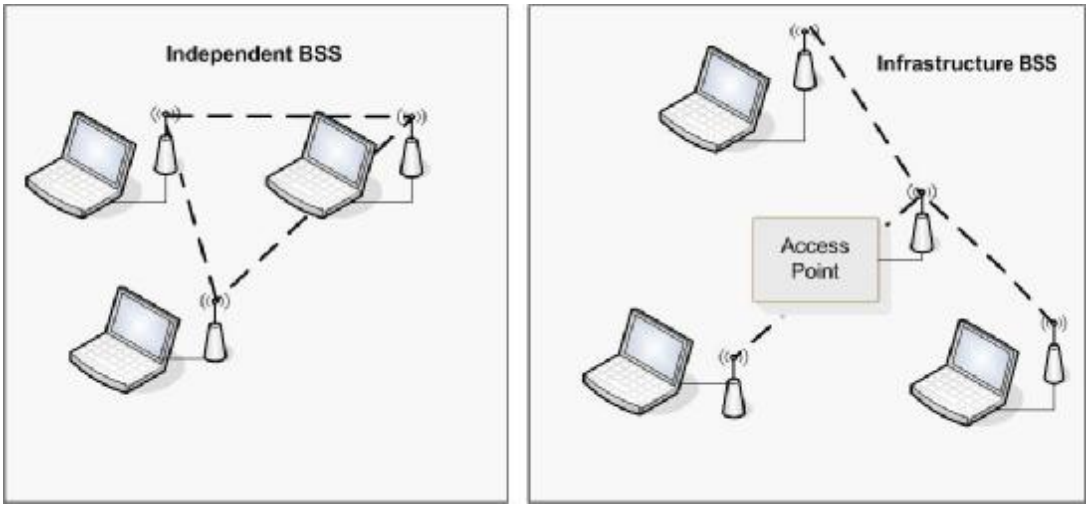
- **Ασύρματο Μέσο (Wireless Medium WM):** χρησιμοποιείται από το πρότυπο για κίνηση πλαισίων από σταθμό σε σταθμό.

- **Σταθμός (Station STA):** αποτελεί την ακρονησίδα ή το τερματικό πρόσβασης του δικτύου προς τον τελικό χρήστη. Αποτελείται από μια

κάρτα-προσαρμογέα δικτύου (Network Adapter Card) και ενσωματωμένο περιβάλλον για ασύρματη σύνδεση.

1.1 Τύποι δικτύων

Η βασική δομή του 802.11 δικτύου είναι το ***Basic Service Set (BSS)***, το οποίο είναι απλά ένα γκρουπ από σταθμούς που επικοινωνούν μεταξύ τους. Η επικοινωνία γίνεται μέσα σε μια περιοχή που ονομάζεται ***basic service area***, που καθορίζεται από τα χαρακτηριστικά μετάδοσης του ασύρματου μέσου. Όταν ένας σταθμός βρίσκεται στην basic service area, μπορεί να επικοινωνήσει με τα υπόλοιπα μέλη της BSS. Υπάρχουν δύο είδη BSS που φαίνονται σχήμα 1.2.



Σχήμα 1.2: Independent BSS και Infrastructure BSS

1.1.1 Independent networks

Στο *independent BSS - IBSS* (Ανεξάρτητο Βασικό Σετ Υπηρεσιών) οι σταθμοί επικοινωνούν κατευθείαν με τους υπόλοιπους και γι' αυτό το λόγο πρέπει να βρίσκονται στα όρια μετάδοσης των σταθμών που θέλουν να επικοινωνήσουν. Το μικρότερο δυνατό 802.11 δίκτυο είναι ένα IBSS με δύο σταθμούς. Τυπικά το IBSS δημιουργήθηκε για μικρό αριθμό σταθμών που επικοινωνούν για μικρή χρονική περίοδο και συγκεκριμένο σκοπό. Αυτά τα δίκτυα ονομάζονται και ad hoc δίκτυα.

1.1.2 Infrastructure networks

Τα *infrastructure BSS* χρησιμοποιούν σημείο πρόσβασης (access point) σε αντίθεση με τα IBSS. Το σημείο πρόσβασης χρησιμοποιείται για όλες τις επικοινωνίες ακόμα και αν πρόκειται για επικοινωνία μεταξύ ασυρμάτων κόμβων που βρίσκονται στην ίδια περιοχή εξυπηρέτησης. Αν ένας κινητός σταθμός σε αυτό το δίκτυο θέλει να επικοινωνήσει με έναν άλλο κινητό σταθμό, η επικοινωνία πρέπει να γίνει σε δυο hops. Αρχικά ο πρώτος σταθμός μεταφέρει το πλαίσιο στο σημείο πρόσβασης και έπειτα το σημείο πρόσβασης μεταφέρει το πλαίσιο στον προορισμό του. Έτσι οι multi hop μεταδόσεις χρειάζονται περισσότερη χωρητικότητα από την απευθείας μετάδοση του πλαισίου από τον αποστολέα στον παραλήπτη.

1.2 Το στρώμα MAC

Όπως όλα τα 802.x πρωτόκολλα, έτσι και το πρωτόκολλο 802.11 καλύπτει το Φυσικό Στρώμα και το Στρώμα MAC.

Το στρώμα MAC ορίζει δυο διαφορετικούς τρόπους προσπέλασης:

- **Την Κατανεμημένη Λειτουργία Συντονισμού (Distributed Coordination Function - DCF)**

- **Τη Σημειακή Λειτουργία Συντονισμού (Point Coordination Function - PCF)**

Η PCF παρέχει εξυπηρέτηση χωρίς ανταγωνισμό. Υπάρχει ένας κόμβος-αφέντης που ελέγχει την πρόσβαση μεταξύ ενός ή περισσότερων κόμβων-σκλάβων.

Παρακάτω θα αναλυθεί μόνο η DCF, αφού η PCF δεν μπορεί να χρησιμοποιηθεί στα ad hoc δίκτυα και δεν θα μελετηθεί περισσότερο σε αυτή την αναφορά.

1.2.1 Κατανεμημένη Λειτουργία Συντονισμού (Distributed Coordination Function - DCF)

Η DCF είναι ο βασικός μηχανισμός προσπέλασης του 802.11. Είναι ένας μηχανισμός πολλαπλής πρόσβασης με ανίχνευση φέροντος και αποφυγή σύγκρουσης (Carrier Sense Multiple Access with Collision Avoidance CSMA/CA). Η διαφορά του πρωτοκόλλου αυτού με το αντίστοιχο

πρωτόκολλο που χρησιμοποιεί το Ethernet, το CSMA/CD (όπου το CD σημαίνει ανίχνευση σύγκρουσης – Collision Detection) είναι ότι οι συγκρούσεις σπαταλούν χρήσιμη χωρητικότητα μετάδοσης και γι' αυτό στο 802.11 χρησιμοποιείται το CA.

Η λειτουργία του πρωτοκόλλου αυτού είναι η εξής:

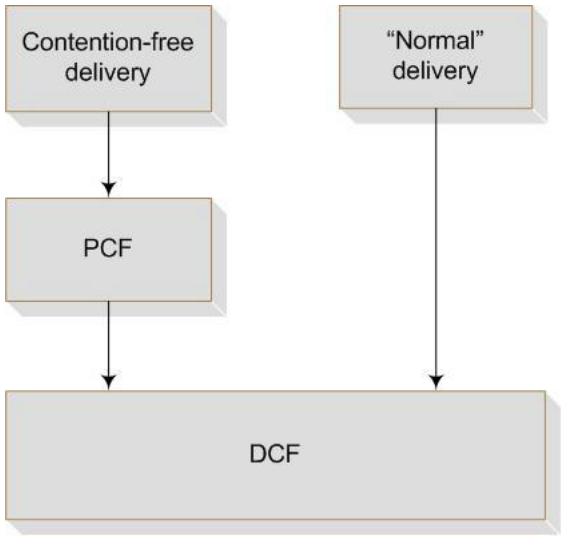
Κάθε σταθμός που επιθυμεί να μεταδώσει, ανιχνεύει το μέσο διάδοσης. Αν το μέσο είναι κατειλημμένο, δηλαδή κάποιος άλλος σταθμός εκπέμπει, τότε ο σταθμός αναβάλλει τη μετάδοση του για κάποιο χρονικό διάστημα. Ο σταθμός επιτρέπεται να εκπέμψει μόνο αν κανένας άλλος σταθμός δεν εκπέμπει, αφού μόνο τότε το μέσο θα είναι ελεύθερο.

Όταν το μέσο δεν είναι υπερβολικά φορτωμένο, το πρωτόκολλο είναι πολύ αποδοτικό γιατί επιτρέπει στους σταθμούς να εκπέμπουν με την ελάχιστη χρονική καθυστέρηση. Υπάρχει όμως και η πιθανότητα οι σταθμοί να ανιχνεύσουν ταυτόχρονα το μέσο ως ελεύθερο και να εκπέμψουν την ίδια χρονική στιγμή με αποτέλεσμα να υπάρξουν συγκρούσεις.

Στα ασύρματα δίκτυα η ανίχνευση σύγκρουσης δε λειτουργεί πολύ καλά σε αντίθεση με τα ενσύρματα, όπως το Ethernet, όπου μπορούν να ανιχνευθούν οι συγκρούσεις και να σταλεί ξανά το πακέτο χρησιμοποιώντας έναν exponential random backoff αλγόριθμο. Για να αποφευχθεί αυτό το πρόβλημα, το 802.11 χρησιμοποιεί το μηχανισμό αποφυγής σύγκρουσης όπως έχει αναφερθεί παραπάνω, και της Θετικής Επιβεβαίωσης (Positive ACK).

Ο μηχανισμός αυτός λειτουργεί ως εξής: Ο σταθμός που επιθυμεί να εκπέμψει ελέγχει το μέσο. Αν το μέσο είναι κατειλημμένο τότε αναβάλλει την εκπομπή του για κάποιο χρονικό διάστημα, το οποίο ονομάζεται Κατανεμημένο Διάστημα μεταξύ Πλαισίων (Distributed Inter Frame Space - DIFS). Μετά από αυτό το διάστημα μπορεί να εκπέμψει.

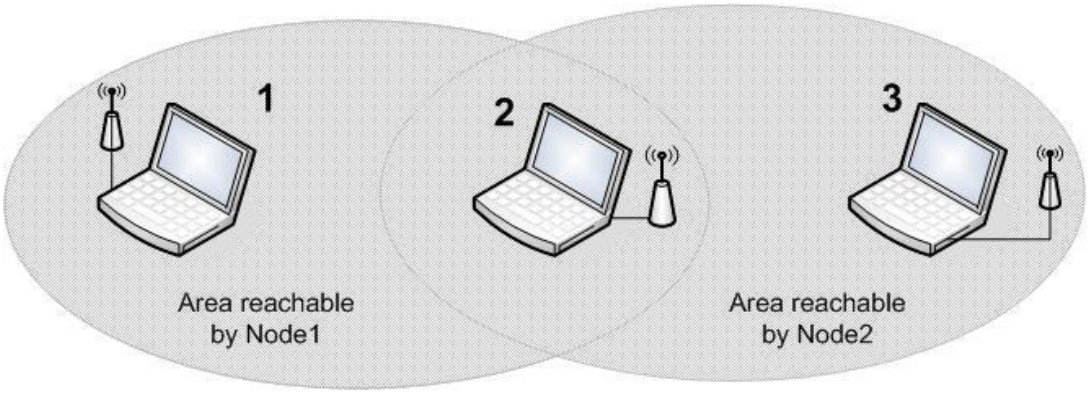
Ο σταθμός-δέκτης ελέγχει το CRC του λαμβανόμενου πακέτου και στέλνει έναν πακέτο Επιβεβαίωσης (ACK). Η παραλαβή του ACK δείχνει στον αποστολέα ότι το πακέτο λήφθηκε σωστά χωρίς να υπάρξει κάποια σύγκρουση. Αν δε ληφθεί πακέτο επιβεβαίωσης από τον αποστολέα για κάποιο καθορισμένο χρονικό διάστημα, τότε ο αποστολέας θεωρεί ότι υπήρξε σύγκρουση και το πακέτο δε λήφθηκε. Έτσι εκπέμπει ξανά το πακέτο μέχρι να λάβει ACK ή μέχρι ο αριθμός των προσπαθειών αναμετάδοσης ξεπεράσει κάποιον προκαθορισμένο αριθμό επαναλήψεων, όπου και απορρίπτεται το πακέτο.



Σχήμα 1.3: MAC coordination functions

1.2.2 Το πρόβλημα του "κρυμμένου" κόμβου (Hidden Node Problem)

Στα Ethernet δίκτυα, οι σταθμοί εξαρτώνται από τη λήψη της μετάδοσης για να εκτελέσουν τη συνάρτηση *carrier sensing* του CSMA/CD. Τα καλώδια στα φυσικά μέσα περιέχουν τα σήματα και τα διαμοιράζουν στους κόμβους του δικτύου. Τα ασύρματα δίκτυα δεν έχουν ξεκάθαρα όρια, μερικές φορές σε σημείο που κάθε κόμβος να μην είναι δυνατό να επικοινωνήσει με κανένα άλλο κόμβο στο ασύρματο δίκτυο, όπως για παράδειγμα στο σχήμα 1.4.



Σχήμα 1.4: Οι κόμβοι 1 και 3 είναι "κρυμμένοι"

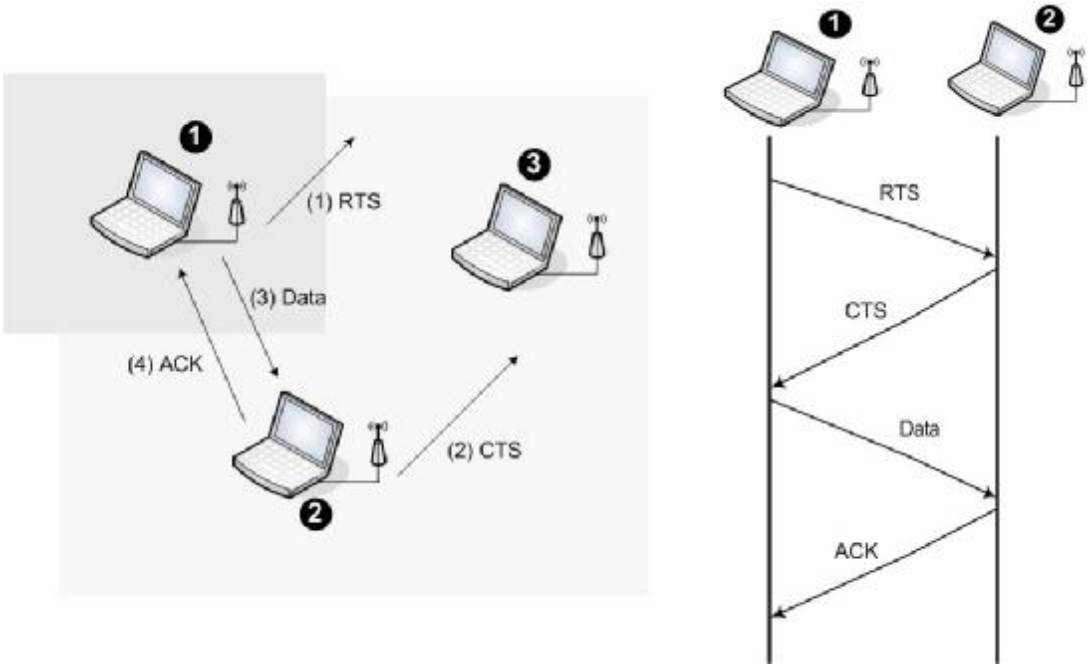
Σε αυτό το σχήμα, ο κόμβος 2 μπορεί να επικοινωνήσει και με τον κόμβο 1 και με τον κόμβο 3, όμως οι κόμβοι 1 και 3 δεν μπορούν να μεταδώσουν ταυτόχρονα. Για τον κόμβο 1, ο κόμβος 3 είναι ένας κρυμμένος κόμβος, αφού βρίσκεται στα όρια του παραλήπτη (του κόμβου 2), αλλά όχι στα δικά του όρια μετάδοσης. Αν γινόταν χρήση ενός πρωτοκόλλου χωρίς εγγύηση επιβεβαίωσης, θα ήταν εύκολο για τους κόμβους 1 και 3 να μεταδώσουν ταυτόχρονα, καθιστώντας έτσι τον κόμβο 2 ανίκανο να ανιχνεύσει οτιδήποτε. Επιπλέον οι κόμβοι 1 και 3 δε θα είχαν καμία ένδειξη λάθους γιατί η σύγκρουση θα γινόταν τοπικά στον κόμβο 2.

Τα αποτελέσματα των συγκρούσεων από τους κρυμμένους κόμβους είναι δύσκολο να ανιχνευθούν στα ασύρματα δίκτυα γιατί οι μεταδότες-παραλήπτες είναι γενικά half-duplex, δηλαδή δεν μπορούν να μεταδίδουν και να λαμβάνουν την ίδια χρονική στιγμή. Για να αποτρέπεται η σύγκρουση, το 802.11 επιτρέπει στους σταθμούς να χρησιμοποιούν τα Request to Send (RTS) και Clear to Send (CTS) μηνύματα. Στο σχήμα 1.5 φαίνεται η διαδικασία αυτή.

Στο σχήμα 1.5, ο κόμβος 1 έχει ένα πλαίσιο να στείλει, και ξεκινά τη διαδικασία στέλνοντας ένα RTS πλαίσιο. Το RTS πλαίσιο εξυπηρετεί πολλούς σκοπούς: εκτός του ότι κρατάει τη ραδιοζεύξη για τη μετάδοση, επιβάλλει σιγή σε όποιον σταθμό το ακούσει. Αν ο σταθμός-προορισμός λάβει το RTS, απαντά με ένα CTS. Όπως και το RTS έτσι και το CTS επιβάλλει σιγή στους σταθμούς της γειτονικής περιοχής. Όταν η ανταλλαγή RTS/CTS ολοκληρωθεί, ο κόμβος 1 μπορεί να μεταδώσει το πλαίσιο του χωρίς να ανησυχεί για παρεμβολή κάποιου κρυμμένου κόμβου. Οι κρυμμένοι κόμβοι έξω από τα όρια του αποστολέα θα σιγήσουν αφού έχουν ακούσει το CTS του παραλήπτη. Όταν χρησιμοποιείται η RTS/CTS ανταλλαγή, κάθε πλαίσιο πρέπει να είναι πλήρως επιβεβαιωμένο.

Η διαδικασία μετάδοσης RTS/CTS καταναλώνει ένα εύλογο ποσό χωρητικότητας, ειδικά λόγω της επιπρόσθετης καθυστέρησης που υφίσταται πριν ξεκινήσει η μετάδοση. Γι’ αυτό τον λόγο χρησιμοποιείται μόνο σε υψηλής χωρητικότητας περιβάλλοντα και σε περιβάλλοντα με σημαντικό ανταγωνισμό για μετάδοση. Για χαμηλότερης χωρητικότητας περιβάλλοντα δεν είναι αναγκαία.

Η RTS/CTS ανταλλαγή μπορεί να ελέγχεται θέτοντας ένα RTS κατώφλι (RTS threshold). Η RTS/CTS ανταλλαγή γίνεται μόνο για πλαίσια μεγαλύτερα από το κατώφλι αυτό. Τα πλαίσια που είναι μικρότερα από το κατώφλι στέλνονται απλά χωρίς την RTS/CTS ανταλλαγή, αποφεύγοντας έτσι την επιπρόσθετη αυτή καθυστέρηση για πακέτα μικρού μεγέθους.



Σχήμα 1.5: RTS/CTS ανταλλαγή

1.2.3 Συναρτήσεις Ανίχνευσης Φέροντος (Carrier – Sensing Functions) και Διάνυσμα Δέσμευσης πόρων του Δικτύου (Network Allocation Vector)

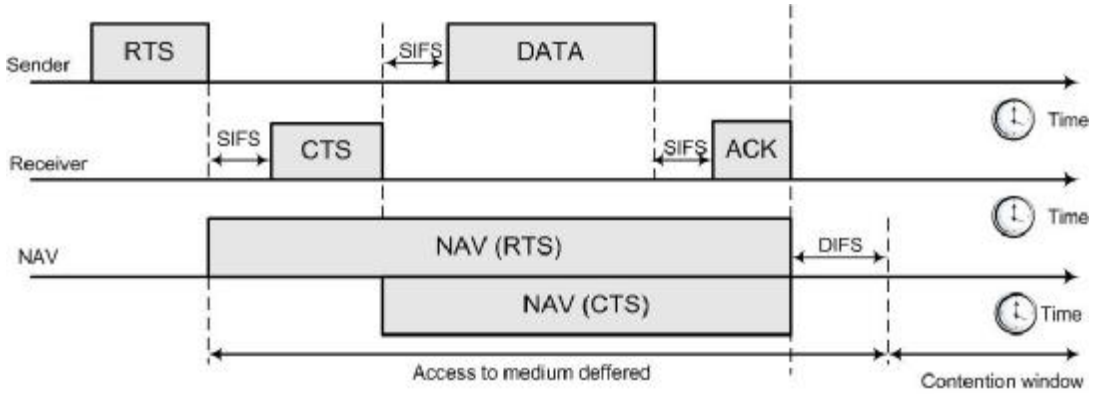
Οι Carrier sensing functions χρησιμοποιούνται για να διαπιστώσουν αν το μέσο είναι ελεύθερο. Υπάρχουν δυο τύποι carrier sensing συναρτήσεων του 802.11 που διαχειρίζονται αυτή τη διαδικασία: η physical carrier-sensing και η virtual carrier-sensing function. Αν κάποια από τις carrier-sensing functions δείξει ότι το μέσο είναι απασχολημένο, αυτό αναφέρεται από το MAC στο ανώτερο επίπεδο.

Οι Physical carrier-sensing συναρτήσεις παρέχονται από το φυσικό επίπεδο με ερώτηση και εξαρτώνται από το μέσο και τη διαμόρφωση που χρησιμοποιείται. Με τους κρυμμένους κόμβους να παραμονεύουν δυναμικά παντού, το physical carrier-sensing δεν μπορεί να περιέχει όλες τις απαραίτητες πληροφορίες.

Οι Virtual carrier-sensing παρέχονται από το Network Allocation Vector (NAV). Τα περισσότερα 802.11 πλαίσια μεταφέρουν ένα πεδίο χρονικής διάρκειας, το οποίο χρησιμοποιείται για να κρατεί το μέσο για μια προκαθορισμένη χρονική περίοδο. Το NAV είναι ένα χρονόμετρο που δείχνει για πόσο χρονικό διάστημα το μέσο θα είναι κατειλημμένο. Οι σταθμοί θέτουν

το NAV τους για χρονική περίοδο ίση με αυτήν που υπολογίζουν ότι θα χρησιμοποιήσουν το μέσο (συμπεριλαμβανομένων και των πλαισίων που είναι απαραίτητα για να ολοκληρωθεί η διαδικασία). Οι άλλοι σταθμοί μετρούν αντίστροφα από το NAV μέχρι το 0. Όταν το NAV δεν είναι μηδέν, η carrier-sensing function δείχνει ότι το μέσο είναι κατειλημμένο, ενώ όταν το NAV φτάσει στο μηδέν, η virtual carrier-sensing function δείχνει ότι το μέσο είναι ελεύθερο.

Χρησιμοποιώντας το NAV, οι σταθμοί μπορούν να εξασφαλίζουν ότι η ατομική λειτουργία τους δε θα διακοπεί, όπως για παράδειγμα η RTS/CTS ανταλλαγή στο σχήμα 1.5. Το σχήμα 1.6 δείχνει πώς το NAV προστατεύει μια ακολουθία έτσι ώστε να μη διακοπεί. Η δραστηριότητα του μέσου φαίνεται από τα σκιασμένα κομμάτια, και σε κάθε κομμάτι αναγράφεται ο τύπος του πλαισίου. Το κενό μεταξύ των πλαισίων δείχνει την έλλειψη δραστηριότητας. Τέλος, ο μετρητής NAV απεικονίζεται από τη γραμμή NAV στο κάτω μέρος του σχήματος. Το NAV, που μεταφέρεται στην επικεφαλίδα των πλαισίων RTS και CTS, απεικονίζεται σε άλλη γραμμή για να δείξει με ποιό τρόπο το NAV σχετίζεται με την πραγματική μετάδοση στον αέρα. Στο σχήμα, όταν υπάρχει το σκιασμένο κομμάτι στη γραμμή του NAV οι σταθμοί πρέπει να αναβάλουν την πρόσβαση τους στο μέσο γιατί ο virtual carrier-sensing μηχανισμός δείχνει ότι το μέσο είναι κατειλημμένο.



Σχήμα 1.6: Χρήση του NAV σε virtual carrier sensing

Για να εξασφαλιστεί ότι η ακολουθία δε θα διακοπεί, ο κόμβος 1 στο σχήμα 1.5 θέτει το NAV του για να μπλοκάρει την πρόσβαση των άλλων κόμβων στο μέσο κατά τη διάρκεια μετάδοσης του RTS του. Όλοι οι σταθμοί που ακούνε το RTS αυτό, αναβάλλουν την προσπάθεια πρόσβασης τους στο μέσο μέχρι να τελειώσει το NAV.

Το RTS πλαίσιο δεν ακούγεται πάντα από όλους τους σταθμούς του δικτύου. Επομένως, ο παραλήπτης της προτιθέμενης μετάδοσης απαντά με ένα CTS που περιλαμβάνει ένα μικρότερο NAV. Αυτό το NAV εμποδίζει τους άλλους σταθμούς (οι οποίοι δεν έχουν ακούσει το RTS αλλά είναι στην γειτονιά του παραλήπτη) να έχουν πρόσβαση στο μέσο μέχρι να ολοκληρωθεί η μετάδοση. Μετά την ολοκλήρωση της ακολουθίας, το μέσο μπορεί να χρησιμοποιηθεί από οποιονδήποτε σταθμό μετά από χρονική περίοδο ίση με Distributed Interframe Space (DIFS), όπως φαίνεται στο δεξί μέρος του παραπάνω σχήματος, όπου και ξεκινά το Contention window (παράθυρο ανταγωνισμού).

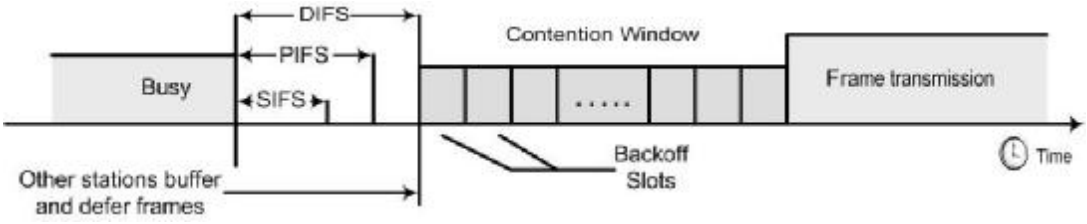
Η RTS/CTS ανταλλαγή είναι χρήσιμη σε "φορτωμένες" κυρίως περιοχές με πολλαπλά επικαλυπτόμενα δίκτυα. Κάθε σταθμός που βρίσκεται στο ίδιο φυσικό κανάλι, λαμβάνει το NAV και μεταθέτει κατάλληλα την πρόσβαση του, ακόμα και αν οι σταθμοί είναι συντονισμένοι να είναι σε διαφορετικά δίκτυα.

1.2.4 Επιβεβαιώσεις στο MAC επίπεδο (MAC Level Acknowledgments)

Το στρώμα MAC όπως αναφέρθηκε παραπάνω, περιμένει να λάβει επιβεβαίωση για κάθε πλαίσιο που στέλνει, το οποίο δεν είναι πλαίσιο ελέγχου. Αν δε λάβει την επιβεβαίωση τότε αναγνωρίζει ότι υπήρξε σύγκρουση στην εν λόγω μετάδοση. Σε broadcast πακέτα (πακέτα με πολλαπλούς προορισμούς), όπως για παράδειγμα τα Multicast, δε χρησιμοποιείται διαδικασία επιβεβαίωσης.

1.2.5 Μεσοδιάστημα μεταξύ πλαισίων (Inter frame Spacing)

Όπως και στο παραδοσιακό Ethernet, το inter frame spacing παίζει έναν πολύ σημαντικό ρόλο για την πρόσβαση στο μέσο μετάδοσης. Το 802.11 χρησιμοποιεί 4 διαφορετικά inter frame spaces μεταξύ των πλαισίων. Τα τρία από αυτά χρησιμοποιούνται για να προσδιορίσουν την πρόσβαση του μέσου. Η σχέση μεταξύ τους φαίνεται στο σχήμα 1.7.



Σχήμα 1.7: Οι χρονικές διάρκειες στο 802.11

Έχουμε ήδη δει ότι στο μηχανισμό αποφυγής συγκρούσεων, όπως αυτός υλοποιήθηκε στο 802.11 MAC, οι σταθμοί καθυστερούν τη μετάδοση τους μέχρι το μέσο να είναι ελεύθερο. Ποικίλα inter frame spacing δημιουργούν διαφορετικά επίπεδα προτεραιότητας για τους διαφορετικούς τύπους της κίνησης.

Η λογική που κρύβεται πίσω από αυτό είναι απλή:

> **Ø** Υψηλής προτεραιότητας κίνηση δε χρειάζεται να περιμένει τόσο μέχρι το μέσο να γίνει ελεύθερο. Επομένως, εάν υπάρχει κάποια υψηλής προτεραιότητας κίνηση να περιμένει, καταλαμβάνει το κανάλι πριν ένα πλαίσιο χαμηλής προτεραιότητας έχει την ευκαιρία να προσπαθήσει.

Για να βοηθήσει με την ανταλλαγή και χρήση πληροφοριών μεταξύ κόμβων με διαφορετικούς ρυθμούς δεδομένων, το inter frame space είναι μια σταθερή τιμή του χρόνου, ανεξάρτητο από την ταχύτητα μετάδοσης. Τα τέσσερα inter frame space περιγράφονται παρακάτω.

- **Short inter frame space – SIFS (Σύντομο Μεσοδιάστημα μεταξύ πλαισίων)**

Το SIFS χρησιμοποιείται για τις υψηλότερης προτεραιότητας μεταδόσεις (highest-priority transmissions), όπως τα RTS/CTS πλαίσια και οι θετικές επιβεβαιώσεις (ACKs). Οι υψηλής προτεραιότητας μεταδόσεις μπορούν να ξεκινήσουν όταν λήξει το SIFS. Όταν ξεκινήσουν αυτές οι υψηλής προτεραιότητας μεταδόσεις το μέσο γίνεται κατειλημμένο και έτσι τα πλαίσια μετάδοσης μετά τη λήξη του SIFS έχουν προτεραιότητα πριν από τα πλαίσια που μπορούν να μεταδοθούν μετά από μεγαλύτερης διάρκειας χρόνους. Το πλαίσιο αυτό είναι το πιο μικρό από τα τέσσερα πλαίσια.

Η τιμή του είναι καθορισμένη σε κάθε Φυσικό στρώμα και είναι υπολογισμένη έτσι ώστε ο σταθμός που εκπέμπει να είναι σε θέση να μεταβεί σε κατάσταση λήψης και να είναι ικανός να αποκωδικοποιήσει το εισερχόμενο πακέτο. Στο 802.11 Φυσικό στρώμα όπου έχουμε Αναπήδηση συχνότητας (FH PHY), η τιμή είναι καθορισμένη στα 28msec.

- **Point Coordination inter frame space (PCF inter frame space) – PIFS (Σημειακός Συντονιστής)**

Τα PIFS, που μερικές φορές λανθασμένα λέγονται priority interframe space, χρησιμοποιούνται από το PCF κατά τη διάρκεια της λειτουργίας χωρίς ανταγωνισμό. Σταθμοί που έχουν δεδομένα να μεταδώσουν σε περίοδο χωρίς ανταγωνισμό μπορούν να τα μεταδώσουν μετά από την λήξη του PIFS και αποτρέπουν έτσι οποιαδήποτε contention-based κίνηση.
Η τιμή του ισούται με SIFS συν μια Χρονοσχισμή, δηλαδή:

PIFS = SIFS + Slot Time

- **DCF interframe space (Distribute IFS) - DIFS (Κατανεμημένο IFS)**

Η DIFS είναι η μικρότερη διάρκεια χρόνου που είναι αδρανές το μέσο για contention-based services. Οι σταθμοί έχουν άμεση πρόσβαση στο μέσο αν αυτό μείνει ελεύθερο για περίοδο μεγαλύτερη από την DIFS.
Η τιμή του ισούται με PIFS συν μια Χρονοσχισμή, δηλαδή:

DIFS = PIFS + Slot Time

- **Extended interframe space – EIFS (Εκτεταμένος IFS)**

Το EIFS δε φαίνεται στο σχήμα 1.7 γιατί η χρονική διάρκεια δεν είναι καθορισμένη. Χρησιμοποιείται μόνο όταν υπάρχουν λάθη στη μετάδοση πλαισίου. Πιο συγκεκριμένα αν κάποιος σταθμός έλαβε ένα πακέτο και δεν το αντιλήφθηκε τότε θα υπήρχε σύγκρουση με ένα μελλοντικό πακέτο. Με αυτή τη χρονική διάρκεια, επιτυγχάνεται η αποφυγή αυτής της περίπτωσης. Το ότι ο σταθμός δεν αντιλήφθηκε το πακέτο σημαίνει ότι δεν μπόρεσε να καταλάβει την πληροφορία της χρονικής διάρκειας του Virtual Carrier Sense.

Interframe spacing και προτεραιότητα

Οι ατομικές λειτουργίες ξεκινούν σαν κανονικές μεταδόσεις, δηλαδή πρέπει να περιμένουν για χρονική περίοδο ίση με DIFS πριν ξεκινήσουν. Ωστόσο, στο δεύτερο και στα υπόλοιπα βήματα των ατομικών λειτουργιών, χρησιμοποιείται χρονική περίοδο ίση με SIFS, και όχι DIFS. Αυτό σημαίνει ότι το δεύτερο, και τα ακόλουθα κομμάτια της ατομικής λειτουργίας, θα καταλάβουν το μέσο πριν άλλος τύπος πλαισίου μπορέσει να μεταδοθεί. Χρησιμοποιώντας το SIFS και το NAV, οι σταθμοί μπορούν να καταλαμβάνουν το μέσο για όσο είναι αναγκαίο.

Στο σχήμα 1.6, για παράδειγμα, το short interframe space (SIFS) χρησιμοποιείται μεταξύ των διαφορετικών τμημάτων της ατομικής ανταλλαγής. Όταν ο αποστολέας κερδίσει πρόσβαση στο μέσο, ο παραλήπτης απαντά με ένα CTS μετά από SIFS. Κάθε σταθμός που προσπαθεί να πάρει

πρόσβαση στο μέσο στο τέλος του RTS θα πρέπει να περιμένει για μια DIFS διάρκεια. Κατά τη διάρκεια της DIFS περιόδου, η SIFS περίοδος θα τελειώσει και θα μεταδοθεί το CTS.

1.2.6 Πρόσβαση στο μέσο βασισμένη σε ανταγωνισμό, χρησιμοποιώντας το DCF

Οι περισσότερες κινήσεις χρησιμοποιούν το DCF (όπως και τα ad hoc δίκτυα), το οποίο παρέχει εξυπηρέτηση βασισμένη στον ανταγωνισμό. Το DCF επιτρέπει σε πολλαπλούς ανεξάρτητους σταθμούς να αλληλεπιδρούν χωρίς κεντρικό έλεγχο, και έτσι μπορούν να χρησιμοποιηθούν σε IBSS δίκτυα ή σε infrastructure δίκτυα.

Παρακάτω θα εξηγηθεί πιο αναλυτικά ο τρόπος λειτουργίας του.

Κάθε σταθμός, πριν προσπαθήσει να μεταδώσει ελέγχει αν το μέσο είναι ελεύθερο. Αν δεν είναι, οι σταθμοί μεταθέτουν τη μετάδοση τους και χρησιμοποιούν έναν backoff αλγόριθμο για αποφυγή των συγκρούσεων.

Μελετώντας τους κανόνες του 802.11 MAC, υπάρχει ένα βασικό σετ κανόνων που χρησιμοποιούνται πάντα, και κάποιοι επιπρόσθετοι κανόνες που χρησιμοποιούνται ανάλογα με κάποιες καταστάσεις. Οι δυο βασικοί κανόνες που χρησιμοποιούνται σε όλες τις μεταδόσεις που χρησιμοποιούν το DCF είναι οι παρακάτω:

1. Αν το μέσο είναι αδρανές για μεγαλύτερο χρόνο από το DIFS, η μετάδοση μπορεί να ξεκινήσει αμέσως. Ο Carrier sensing λειτουργεί χρησιμοποιώντας και τις δυο μεθόδους, την physical medium dependent και την virtual (NAV).

   I. Αν το προηγούμενο πλαίσιο έχει ληφθεί χωρίς λάθη, το μέσο πρέπει να είναι ελεύθερο για τουλάχιστον DIFS διάρκεια.

   II. Αν η προηγούμενη μετάδοση περιέχει λάθη, το μέσο πρέπει να είναι ελεύθερο για περίοδο ίση με EIFS.

2. Αν το μέσο είναι κατειλημμένο ο σταθμός πρέπει να περιμένει μέχρι το κανάλι να μείνει ελεύθερο. Στο 802.11 η αναμονή αναφέρεται σαν *access deferral*. Σε αυτή την περίπτωση ο σταθμός περιμένει να μείνει το μέσο ελεύθερο για χρονική διάρκεια ίση DIFS και προετοιμάζεται για την exponential backoff διαδικασία.

Οι επιπρόσθετοι κανόνες μπορεί να χρησιμοποιηθούν σε διάφορες περιπτώσεις. Πολλοί από αυτούς τους κανόνες εξαρτώνται από την "on the wire" κατάσταση, συγκεκριμένα από τα αποτελέσματα των προηγούμενων μεταδόσεων. Οι κανόνες είναι οι εξής:

1. Η αποκατάσταση των σφαλμάτων μετάδοσης είναι ευθύνη του σταθμού που μεταδίδει το πλαίσιο. Οι αποστολείς περιμένουν επιβεβαίωση για κάθε απεσταλμένο πλαίσιο και είναι υπεύθυνοι για να ξαναπροσπαθούν να μεταδώσουν μέχρι να το επιτύχουν.

   I. Θετική επιβεβαίωση είναι η μόνη ένδειξη για επιτυχή αποστολή. Οι ατομικές συναλλαγές πρέπει να ολοκληρωθούν εντελώς για να είναι επιτυχείς. Αν αναμένεται μια επιβεβαίωση και δε ληφθεί, ο αποστολέας θεωρεί ότι η μετάδοση χάθηκε και πρέπει να ξαναπροσπαθήσει.

   II. Όλα τα unicast δεδομένα πρέπει να επιβεβαιώνονται.

   III. Κάθε αποτυχία αυξάνει ένα retry μετρητή, και γίνεται προσπάθεια για αναμετάδοση. Η αποτυχία μπορεί να οφείλεται είτε στην αποτυχία του σταθμού να κερδίσει πρόσβαση στο μέσο είτε στην έλλειψη επιβεβαίωσης. Ωστόσο, το παράθυρο συμφόρησης είναι μεγαλύτερο όταν ξαναγίνεται προσπάθεια για μετάδοση.

2. Οι ακολουθίες των Multiframe ενημερώνουν το NAV σε κάθε βήμα στη λειτουργία μετάδοσης. Όταν ένας σταθμός λάβει μια ένδειξη ότι το μέσο είναι κατειλημμένο για μεγαλύτερη χρονική στιγμή από το τρέχον NAV, ενημερώνει το NAV με το νέο χρόνο.

3. Οι παρακάτω τύποι των πλαισίων μπορούν να μεταδοθούν μετά από το SIFS και έτσι λαμβάνουν μέγιστη προτεραιότητα: οι επιβεβαιώσεις (ACKs), τα CTS της RTS/CTS ανταλλαγής και τα fragments στις fragment sequences.

   I. Όταν ένας σταθμός μεταδώσει το πρώτο του πλαίσιο στην ακολουθία, κερδίζει τον έλεγχο του καναλιού. Κάθε επιπρόσθετο πλαίσιο και η επιβεβαίωση του μπορούν να σταλούν χρησιμοποιώντας το short interframe space (SIFS), το οποίο “κλειδώνει” έξω τους υπόλοιπους σταθμούς.

   II. Τα επιπρόσθετα πλαίσια στις ακολουθίες ενημερώνουν το NAV για τον επιπρόσθετο χρόνο στον οποίο το μέσο θα είναι κατειλημμένο.

4. Τα Extended frame sequences απαιτούνται για higher-level πακέτα τα οποία είναι μεγαλύτερα από το προκαθορισμένο κατώφλι .

I. Τα πακέτα που είναι μεγαλύτερα από το RTS κατώφλι πρέπει να κάνουν και RTS/CTS συναλλαγές.
II. Τα πακέτα που είναι μεγαλύτερα από το κατώφλι καταμερισμού (fragmentation threshold) πρέπει να τεμαχιστούν.

1.2.7 Διόρθωση λαθών με τον DCF

Ο εντοπισμός λαθών και η διόρθωση τους εξαρτάται από τον κάθε σταθμό που ξεκινά την ατομική ανταλλαγή πλαισίων. Όταν ανιχνευθεί ένα λάθος, ο σταθμός με τα δεδομένα πρέπει να ξαναστείλει το πλαίσιο. Τα λάθη πρέπει να ανιχνεύονται από το σταθμό-αποστολέα. Σε κάποιες περιπτώσεις, ο αποστολέας μπορεί να συμπεράνει ότι χάθηκε το πλαίσιο από τη μη λήψη της θετικής επιβεβαίωσης από τον παραλήπτη. Οι μετρητές επανάληψης αποστολής (Retry counters) αυξάνονται όταν ένα πλαίσιο στέλνεται ξανά.

Κάθε πλαίσιο ή τεμάχιο έχει ένα μοναδικό retry counter συνδεδεμένο μαζί του. Οι σταθμοί έχουν δυο retry counters: τον ***short retry count*** και τον ***long retry count***. Τα πλαίσια που είναι μικρότερα από το RTS κατώφλι θεωρούνται σαν μικρά, και πλαίσια μεγαλύτερα από το κατώφλι ως μεγάλα. Ανάλογα με το μέγεθος του πλαισίου, συνδέονται είτε με τον short είτε με τον long retry counter. Οι Frame retry counts ξεκινούν από το 0 και αυξάνονται όταν μια μετάδοση πλαισίου αποτύχει.

Ο short retry count μηδενίζεται όταν:

- Ένα CTS πλαίσιο λαμβάνεται σαν απάντηση του εκπεμπόμενου RTS.
- Μια MAC-layer επιβεβαίωση λαμβάνεται μετά από μια μη-τεμαχισμένη (non-fragmented) μετάδοση.
- Μια εκπομπή (broadcast) ή ένα multicast frame ληφθεί.

Ο long retry count μηδενίζεται όταν:

- Μια MAC-layer επιβεβαίωση λαμβάνεται για ένα πλαίσιο μεγαλύτερο από το RTS κατώφλι.
- Μια εκπομπή (broadcast) ή ένα multicast frame ληφθεί.

1.2.8 Χρήση των retry counters

Όπως τα περισσότερα από τα υπόλοιπα πρωτόκολλα, το 802.11 παρέχει αξιοπιστία κατά τις αναμεταδόσεις. Οι μεταδόσεις δεδομένων συμβαίνουν μέσα στα όρια μιας ατομικής ακολουθίας και ολόκληρη η ακολουθία πρέπει να ολοκληρωθεί για να γίνει επιτυχώς η μετάδοση. Όταν ένας σταθμός μεταδίδει ένα πλαίσιο πρέπει να λάβει μια επιβεβαίωση από τον παραλήπτη αλλιώς θα θεωρήσει ότι η μετάδοση απέτυχε. Οι αποτυχημένες μεταδόσεις αυξάνουν τον retry counter που είναι συνδεδεμένος με το πλαίσιο (ή το τεμάχιο). Αν φτάσει το όριο των προσπαθειών αναμετάδοσης, τότε το πλαίσιο απορρίπτεται και η απώλεια αναφέρεται στα πρωτόκολλα ανωτέρων επιπέδων.

Ένας από τους λόγους που υπάρχουν μικρά και μεγάλα πλαίσια είναι για να επιτρέπεται στο διαχειριστή του δικτύου να καθορίζει το εύρος των δικτύων για διαφορετικά μεγέθη πλαισίων. Τα μεγάλα πλαίσια χρειάζονται περισσότερο buffer space, έτσι μια δυναμική εφαρμογή που έχει δυο ξεχωριστά retry limits μειώνει το μεγάλο retry limit για να μειωθεί ο χώρος του buffer που χρειάζεται.

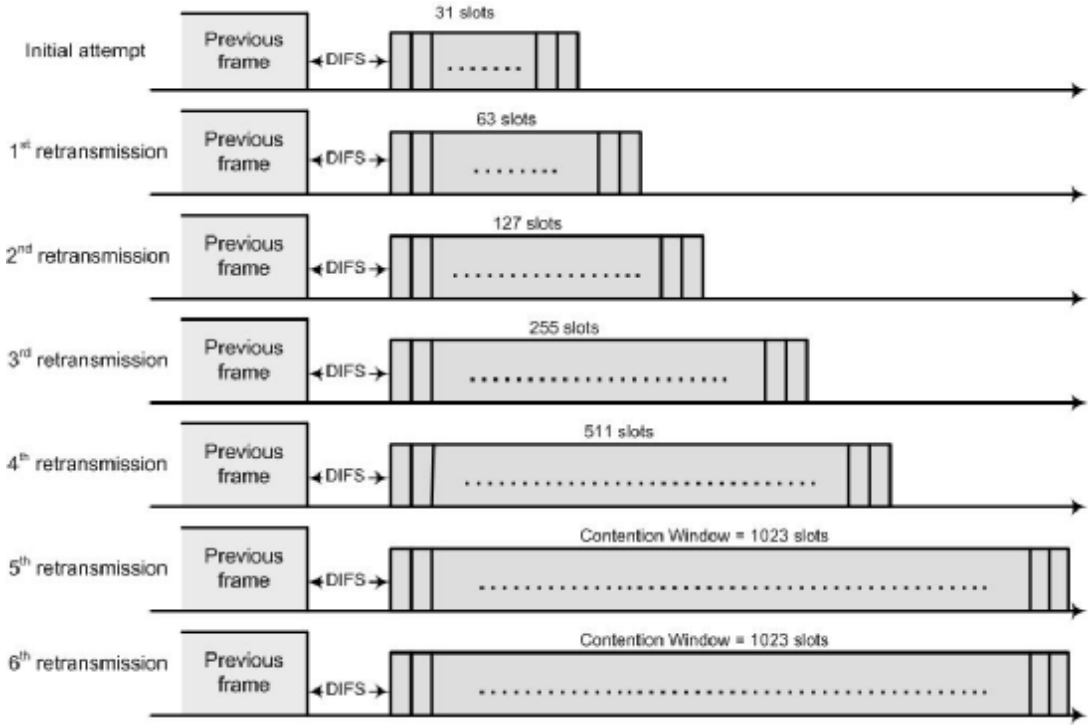
1.2.9 Αλγόριθμος τυχαίας εκθετικής υπαναχώρησης (Exponential random backoff Algorithm)

Το backoff είναι μια πολύ γνωστή μέθοδος, που χρησιμοποιείται για να καθοριστεί η σειρά με την οποία θα έχουν πρόσβαση στο μέσο οι διάφοροι σταθμοί που θέλουν να εκπέμψουν.

Μετά την ολοκλήρωση της μετάδοσης του πλαισίου και την εκπνοή του DIFS, οι σταθμοί μπορούν να προσπαθήσουν να μεταδώσουν δεδομένα. Μια περίοδος που ονομάζεται παράθυρο ανταγωνισμού (*contention window* ή *backoff window*) ακολουθεί το DIFS. Το παράθυρο διαιρείται σε σχισμές (slots). Το μέγεθος των σχισμών εξαρτάται από το μέσο. Για παράδειγμα, τα μεγαλύτερης ταχύτητας φυσικά επίπεδα χρησιμοποιούν μικρότερους slot times. Οι σταθμοί διαλέγουν μια τυχαία σχισμή και περιμένουν όσο διαρκεί αυτή η σχισμή πριν προσπαθήσουν να πάρουν πρόσβαση στο μέσο (όλες οι σχισμές έχουν την ίδια πιθανότητα να επιλεγούν). Όταν πολλοί σταθμοί προσπαθούν να μεταδώσουν, ο σταθμός ο οποίος διάλεξε την πρώτη σχισμή (ο σταθμός δηλαδή με το μικρότερο τυχαίο αριθμό) κερδίζει, και επιτυγχάνει να έχει πρόσβαση στο μέσο για να πραγματοποιήσει την μετάδοση του. Η διάρκεια της σχισμής αυτής είναι καθορισμένη με τέτοιο τρόπο, ώστε ο σταθμός να είναι πάντα σε θέση να αντιλαμβάνεται αν κάποιος σταθμός έχει

καταλάβει το μέσο στην αρχή του προηγούμενου Slot Time. Με αυτό τον τρόπο η πιθανότητα σύγκρουσης μειώνεται κατά 50%.

Όπως και στο Ethernet, έτσι και στο 802.11 ο backoff time διαλέγεται από μεγαλύτερο όριο κάθε φορά που η μετάδοση αποτυγχάνει. Το σχήμα 1.8 δείχνει την αύξηση του contention window καθώς ο αριθμός των μεταδόσεων αυξάνεται, χρησιμοποιώντας τους αριθμούς από direct-sequence spread-spectrum (DSSS) physical layer. Τα άλλα φυσικά επίπεδα χρησιμοποιούν διαφορετικό μέγεθος, αλλά η ιδέα είναι ακριβώς η ίδια. Το μέγεθος του contention window είναι πάντα μικρότερο κατά 1 από τις δυνάμεις του 2 (π.χ. 31, 63, 127, 255), γι' αυτό λέγεται εκθετικός αλγόριθμος. Κάθε φορά πού ο retry counter αυξάνεται, το contention window μετακινείται στην επόμενη μεγαλύτερη δύναμη του 2. Το μέγεθος του contention window είναι οριοθετημένο από το φυσικό επίπεδο. Για παράδειγμα, το DS φυσικό επίπεδο οριοθετεί το contention window στις 1023 σχισμές μετάδοσης.



Σχήμα 1.8: Μέγεθος παραθύρου συμφόρησης DSSS

Όταν το contention window φτάσει στο μέγιστο μέγεθος, παραμένει εκεί μέχρι να μπορεί να μηδενιστεί. Επιτρέπονται μεγάλα contention window όταν πολλοί ανταγωνιζόμενοι σταθμοί που προσπαθούν να κερδίσουν πρόσβαση στο μέσο κρατούν τον MAC αλγόριθμο τους σταθερό ακόμα και αν υπάρχει

μέγιστη φόρτωση. Το contention window επανέρχεται στο ελάχιστο μέγεθος του όταν τα πλαίσια μεταδοθούν επιτυχώς ή όταν ο συνδεδεμένος με αυτά retry counter φτάσει το όριο των προσπαθειών αναμετάδοσης, όπου και το πλαίσιο θα απορριφθεί.

Το πρότυπο 802.11 καθορίζει έναν **Exponential backoff Algorithm,** ο οποίος θα πρέπει να εκτελείται στις ακόλουθες περιπτώσεις:

- Όταν ο σταθμός ανιχνεύει το μέσο πριν από την πρώτη μετάδοση του πακέτου και διαπιστώνει ότι το μέσο είναι κατειλημμένο
- Μετά από κάθε αναμετάδοση (retransmission)
- Μετά από μια επιτυχημένη μετάδοση

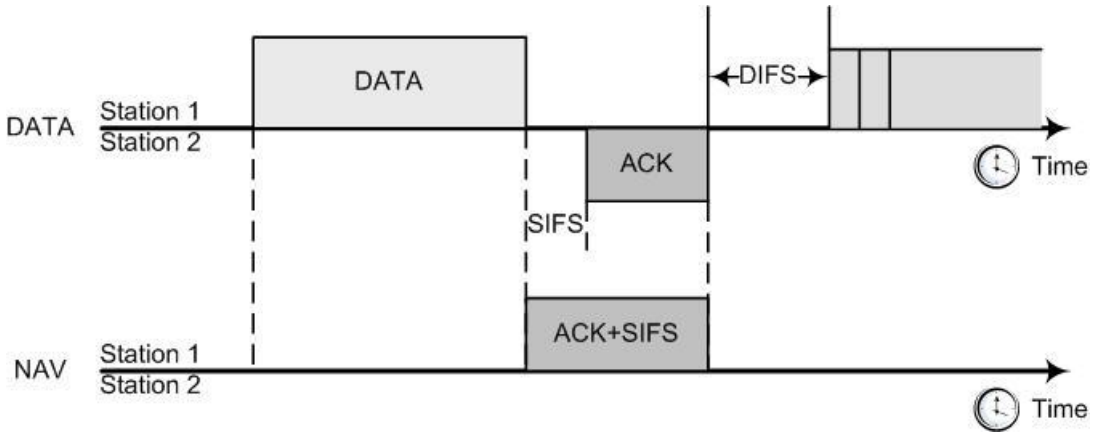
Η μόνη περίπτωση όπου ο μηχανισμός αυτός δε χρησιμοποιείται είναι όταν ο σταθμός αποφασίσει να εκπέμψει ένα νέο πακέτο και το μέσο ήταν ελεύθερο για χρονικό διάστημα μεγαλύτερο από DIFS.

1.2.10 Unicast Frames

Τα πλαίσια που προορίζονται για ένα απλό σταθμό ονομάζονται *directed* data από το πρότυπο του 802.11. Μια πιο κοινή ονομασία αυτών των πλαισίων είναι ***unicast.*** Τα unicast πλαίσια πρέπει να είναι επιβεβαιωμένα για να εξασφαλίζουν αξιοπιστία κάτι το οποίο σημαίνει ότι μπορεί να χρησιμοποιηθεί ποικιλία μηχανισμών για βελτίωση της αποδοτικότητας.

1.2.11 Θετική Επιβεβαίωση (positive acknowledgment)

Οι αξιόπιστες μεταδόσεις μεταξύ δυο σταθμών είναι βασισμένες σε απλές θετικές επιβεβαιώσεις. Τα unicast πλαίσια δεδομένων πρέπει να επιβεβαιώνονται, αλλιώς το πλαίσιο θεωρείται ότι έχει χαθεί. Η πιο βασική περίπτωση είναι ένα απλό πλαίσιο και η συνοδευτική επιβεβαίωση του όπως φαίνεται στο σχήμα 1.9.



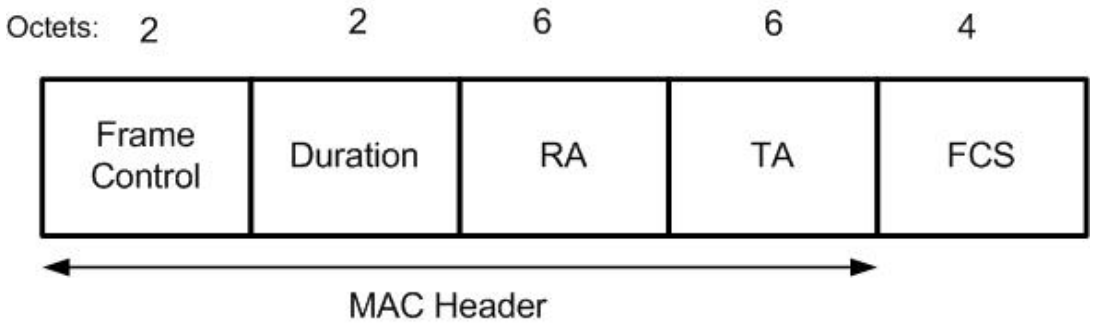
Σχήμα 1.9: Θετική επιβεβαίωση δεδομένων

Το πλαίσιο χρησιμοποιεί το NAV για να κρατήσει το μέσο για το πλαίσιο, την επιβεβαίωση του και το χρόνο SIFS. Θέτοντας ένα μεγάλο NAV, ο αποστολέας "κλειδώνει" τον virtual carrier για ολόκληρη την ακολουθία, εξασφαλίζοντας ότι ο παραλήπτης του πλαισίου θα μπορεί να στείλει την επιβεβαίωση. Επειδή η ακολουθία ολοκληρώνεται με το ACK, δεν είναι αναγκαίο πλέον το "κλείδωμα" του virtual Carrier, και έτσι το NAV στο ACK τίθεται ίσο με μηδέν.

1.2.12 Πλαίσια ελέγχου

1.2.12.1 RTS Πλαίσιο

Το RTS πλαίσιο έχει την παρακάτω μορφή:



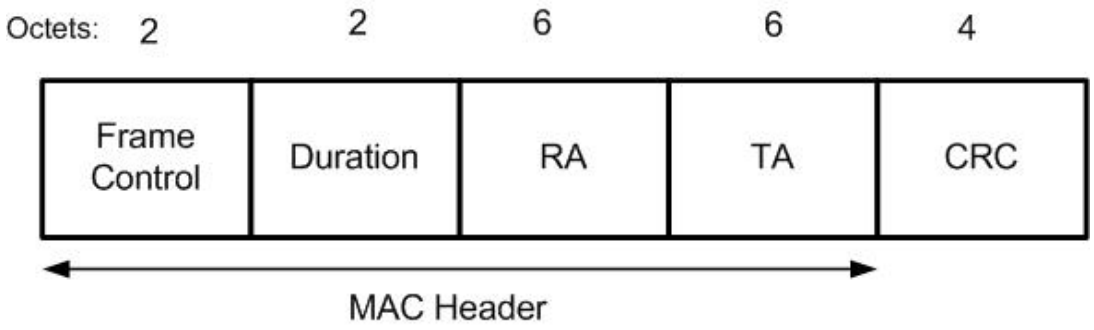
Σχήμα 1.10: RTS Πλαίσιο

Η διεύθυνση δέκτη (Receiver Address, RA) του RTS πλαισίου είναι η διεύθυνση του σταθμού (STA) στο ασύρματο μέσο, η οποία είναι ο άμεσος παραλήπτης του επόμενου πλαισίου Δεδομένων. Η διεύθυνση εκπομπής (Transmitter Address, TA) είναι η διεύθυνση του STA που εκπέμπει το RTS πλαίσιο. Η διάρκεια (Duration) ισούται με το άθροισμα των χρόνων που απαιτούνται για να σταλούν τα εξής πλαίσια: το επόμενο πλαίσιο δεδομένων, το πλαίσιο CTS, το πλαίσιο ACK και 3 SIFS.

$Duration_{RTS} = t_{transm,Data(frame)} + CTS + ACK + 3*SIFS$ [msec]

1.2.12.2 CTS Πλαίσιο

Το CTS πλαίσιο έχει την παρακάτω μορφή:



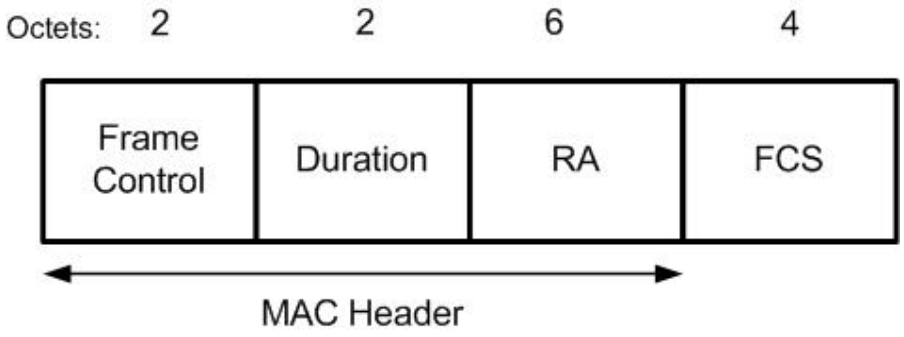
Σχήμα 1.11: CTS Πλαίσιο

Η διεύθυνση δέκτη (RA) του CTS πλαισίου έχει αντιγραφεί από το πεδίο TA (διεύθυνση εκπομπής) του αμέσως προηγούμενου RTS πλαισίου, του οποίου το CTS είναι η ανταπόκριση αυτού. Η διάρκεια (Duration) ισούται με την τιμή που λαμβάνεται από το πεδίο Duration του προηγούμενου RTS πλαισίου, μείον το χρόνο σε msec που απαιτείται για να γίνει η εκπομπή του CTS πλαισίου και του αντίστοιχου SIFS περιθωρίου.

$Duration_{CTS} = Duration\ field_{RTS\ frame} – CTS – SIFS$ [msec]

1.2.12.3 ACK Πλαίσιο

Το ACK πλαίσιο έχει την παρακάτω μορφή:



Σχήμα 1.12: ACK Πλαίσιο

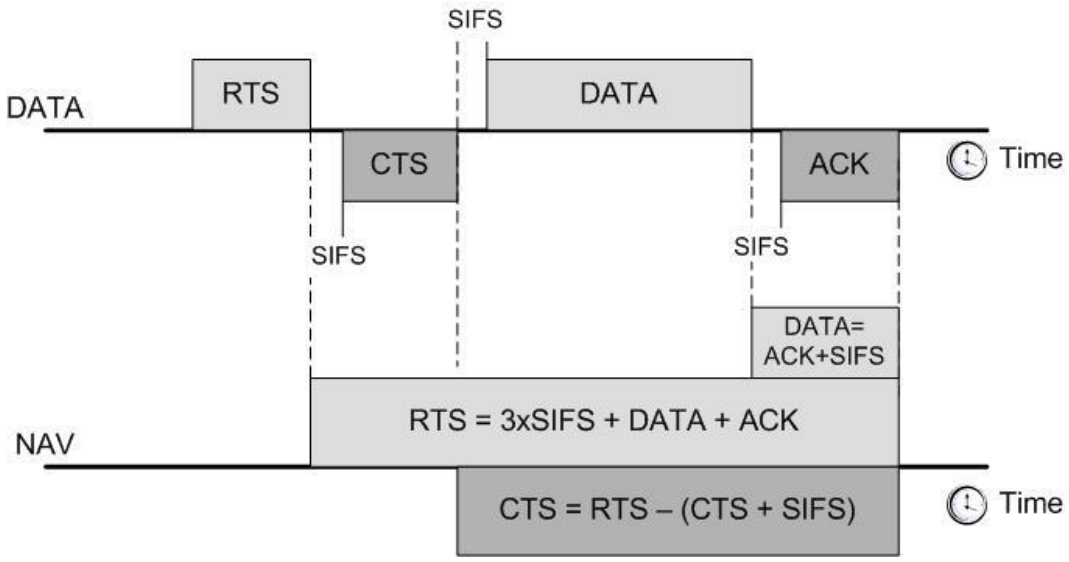
Η διεύθυνση δέκτη (RA) του ACK πλαισίου έχει αντιγραφεί από το πεδίο Διεύθυνση του αμέσως προηγούμενου πλαισίου. Αν το bit του More Fragment ισούται με 0 στο πεδίο ελέγχου του αμέσως προηγούμενου πλαισίου (Frame Control field), τότε η τιμή του Duration ισούται με 0. Αν δεν είναι 0 το bit τότε το Duration ισούται με την τιμή του πεδίου Duration του προηγούμενου πλαισίου που έλαβε, μείον το χρόνο σε msec που απαιτείται για να γίνει η εκπομπή του ACK πλαισίου και μείον μια χρονική διάρκεια ίση με SIFS.

$\text{Duration}_{\text{ACK}} = \text{Duration field}_{\text{frame}} - \text{ACK} - \text{SIFS}$ [msec]

1.2.13 RTS/CTS ανταλλαγή

Για εγγυημένη κατάληψη του μέσου και μετάδοση των δεδομένων χωρίς διακοπή, ο σταθμός μπορεί να χρησιμοποιήσει την RTS/CTS ανταλλαγή. Το σχήμα 1.13 δείχνει αυτή τη διαδικασία. Το RTS πλαίσιο ελέγχου, θέτει το NAV του για αρκετά μεγάλη περίοδο για να χωρέσει και το ACK των δεδομένων που θα στείλει (το θέτει ίσο με 3*SIFS + CTS + DATA + ACK, όπως φαίνεται και στο σχήμα και έχει αναφερθεί παραπάνω). Το επόμενο CTS πλαίσιο θέτει το NAV του μικρότερο από πριν κατά μια περίοδο SIFS και τη διάρκεια του CTS αφού έχει ήδη σταλεί. Ακολούθως όταν σταλούν τα δεδομένα, το NAV γίνεται ίσο με ACK + SIFS για να μείνει το μέσο κατειλημμένο μέχρι να ληφθεί και η επιβεβαίωση του πακέτου.

Το NAV στο RTS επιτρέπει στο CTS να ολοκληρωθεί, και το CTS χρησιμοποιείται για να κρατήσει πρόσβαση στο μέσο για το πλαίσιο δεδομένων.



Σχήμα 1.13: Διαδικασία RTS/CTS ανταλλαγής

Η RTS/CTS ανταλλαγή μπορεί να χρησιμοποιηθεί για όλες τις ανταλλαγές πλαισίων, για καμία από αυτές ή σε κάποιες από αυτές. Η συμπεριφορά του RTS/CTS ελέγχεται από ένα κατώφλι το οποίο τίθεται στο driver software. Πλαίσια μεγαλύτερα από το κατώφλι αυτό χρησιμοποιούν την RTS/CTS ανταλλαγή για να κρατήσουν το μέσο καθ' όλη τη διάρκεια της μετάδοσης, ενώ τα μικρότερα πλαίσια απλά μεταδίδονται.

1.2.14 Ad Hoc Δίκτυα

Σε ορισμένες περιπτώσεις οι χρήστες επιθυμούν να φτιάξουν ένα ασύρματο δίκτυο χωρίς να χρησιμοποιήσουν Access Point. Χαρακτηριστικό παράδειγμα είναι η ανταλλαγή αρχείων μεταξύ δυο ή περισσότερων φορητών υπολογιστών.

Αυτές οι περιπτώσεις χαρακτηρίζονται από το πρότυπο 802.11 ως ad hoc δίκτυα. Δεν υπάρχει δηλαδή Access Point, αλλά μέρος από τις λειτουργίες διενεργούνται από τους τελικούς χρήστες STA. Στα multi-hop ad hoc δίκτυα, οι κόμβοι επικοινωνούν με τους άλλους χρησιμοποιώντας multi-hop ασύρματες συνδέσεις και δεν υπάρχουν σταθερές υποδομές, όπως σταθμός βάσης. Κάθε κόμβος στο δίκτυο μπορεί να δρα σαν router, προωθώντας πακέτα δεδομένων στους άλλους κόμβους.

2. Το πρωτόκολλο MACA-P

2.1 Εισαγωγή

Το πρωτόκολλο 802.11 δε σχεδιάστηκε για multi-hop δίκτυα. Παρόλο που μπορεί να υποστηρίξει όπως αναφέρθηκε πιο πάνω μερικές ***ad hoc*** αρχιτεκτονικές, δεν προορίζεται για να υποστηρίξει ασύρματα κινητά ad hoc δίκτυα, στα οποία η multi-hop διασύνδεση είναι το κύριο χαρακτηριστικό. Διάφορες μελέτες που έγιναν για το multi-hop έδειξαν άλλωστε ότι χρησιμοποιώντας το IEEE 802.11 wireless LANs βασισμένο σε RTS/CTS ανταλλαγές παρουσιάζει σημαντικές απώλειες επίδοσης με αισθητή πτώση στη ρυθμαπόδοση του. Το MACA-P είναι ένα πρωτόκολλο βασισμένο στο 802.11, που χρησιμοποιεί RTS/CTS ανταλλαγή, και μπορεί να υποστηρίξει ταυτόχρονη μετάδοση. Το πρωτόκολλο αυτό προτάθηκε για να λύσει αυτό ακριβώς το πρόβλημα.

Πολλές μελέτες έδειξαν ότι τα multi-hops δίκτυα βασισμένα στο πρωτόκολλο MAC 802.11 παρουσιάζουν σημαντικές απώλειες επίδοσης. Για παράδειγμα, το multitcp έδειξε πως οι σύνοδοι TCP υποφέρουν από απότομη πτώση στη ρυθμαπόδοση όταν μεταδίδονται πολλά hops χρησιμοποιώντας το 802.11. Ένας σημαντικός λόγος για τη χαμηλή απόδοση είναι η υπερβολικά περιοριστική φύση του 802.11, που δεν επιτρέπει πολλαπλές ταυτόχρονες μεταδόσεις ακόμα και αν αυτές είναι “ιδανικά” εφικτές. Ο 802.11 CSMA-CA μηχανισμός για διανεμημένη πρόσβαση στο από “κοινού” κανάλι, σχεδιάστηκε κυρίως για single-hop ασύρματα LAN, όπου όλοι οι κόμβοι μπορούν να επικοινωνήσουν με μόνο ένα άλλο κόμβο. Προφανώς πολλαπλή ταυτόχρονη επικοινωνία δεν είναι εφικτή σε αυτή την περίπτωση. Τα multi-hop ασύρματα δίκτυα είναι εντελώς διαφορετικά, με διαφορετικούς κόμβους ικανούς να επικοινωνούν απ’ ευθείας με διαφορετικά σύνολα από one-hop γείτονες.

Πολλές μελέτες έχουν γίνει σε αυτή την περιοχή. Στο παρελθόν το macaw και το dfwmac έκαναν κάποια νύξη για παράλληλες μεταδόσεις αλλά δεν πρότειναν κάποια λύση. Πιο πρόσφατα το pcma περιγράφει ένα σχήμα δυναμικού ελέγχου για να αυξήσει τον αριθμό των ταυτόχρονων μεταδόσεων σε ένα ad-hoc ασύρματο δίκτυο. Οι προσπάθειες αυτές για βελτίωση των διαστημάτων για τα multi-hop δίκτυα εστιάστηκαν σε δύο προσεγγίσεις. Οι προσεγγίσεις αυτές είχαν θεμελιώδη σκοπό να μειώσουν το μέγεθος της γειτονιάς του ενός hop και να επιτρέψουν στο δίκτυο να διαχωρίζεται σε μεγαλύτερο αριθμό από ζώνες κατά τις ταυτόχρονες μεταδόσεις. Οι δύο αυτές προσεγγίσεις οι ακόλουθες:

I. Αλγόριθμοι δυναμικού ελέγχου (π.χ. pcma)
II. Χρησιμοποιώντας κατευθυντικές κεραίες (direct).

Ενώ οι έρευνες προσομοίωσης έδειξαν ότι οι δύο αυτές προσεγγίσεις μπορούν να αυξήσουν σημαντικά τη συνολική χωρητικότητα του καναλιού των multi-hop δικτύων, αυτές υποφέρουν από διάφορα μειονεκτήματα. Οι Distributed versions των πρωτοκόλλων δυναμικού ελέγχου απαιτούν οι κόμβοι να περιλαμβάνουν και να αποκρυπτογραφούν τα επίπεδα ισχύς της μετάδοσης στις επικεφαλίδες των MAC πακέτων ελέγχου. Σε πραγματικές καταστάσεις όμως, όπου τα όρια επέμβασης είναι μεγαλύτερα από τα όρια λήψης των πραγματικών πακέτων, οι κόμβοι υποφέρουν από επιδράσεις επέμβασης από γειτονικούς αποστολείς ακόμα και αν αυτοί δεν μπορούν να λάβουν σωστά τα πακέτα τους (και έτσι δεν μπορούν να εκτελέσουν σωστά τους κατάλληλους υπολογισμούς του επιπέδου ισχύος).

Οι κατευθυντικές κεραίες από την άλλη, χρησιμοποιούν πολύπλοκο hardware και στρατηγικές διαμόρφωσης φάσης, που δεν είναι οικονομικά συμφέρουσα λύση για μεγάλης κλίμακας ανάπτυξη, ειδικά για ασύρματες συσκευές.

Αυτές οι προσεγγίσεις εστιάστηκαν στον διαχωρισμό των δικτύων σε μικρότερα και περισσότερα τμήματα για να μπορούν να ενεργούν παράλληλα, ενώ η βασική προσπάθεια ήταν να γίνει απλά λιγότερο περιοριστικό το υπάρχον πρωτόκολλο MAC 802.11.

Αντίθετα, το MACA-P που θα μελετηθεί σε αυτό το κεφάλαιο δε χρησιμοποιεί μηχανισμούς δυναμικού ελέγχου, αλλά απλά επεκτείνει τα RTS/CTS μηνύματα ελέγχου για να αυξήσει την πιθανότητα για παράλληλη μετάδοση. Μια άλλη πρόσφατη δουλειά που έγινε είναι το seedex, του οποίου προσπάθεια είναι η αποφυγή των συγκρούσεων, με κάθε κόμβο να προετοιμάζει την αποστολή του σε τυχαίο χρόνο μέσα σε μια two-hop γειτονιά. Ωστόσο η θεμελιώδης διαφορά στην σχεδίαση και την επιτυχία του MACA-P σε σύγκριση με το seedex είναι ότι το MACA-P μοιράζεται τις κοινές πληροφορίες προωθώντας τις στους γειτονικούς κόμβους για καλύτερη πρόσβαση στο κανάλι.

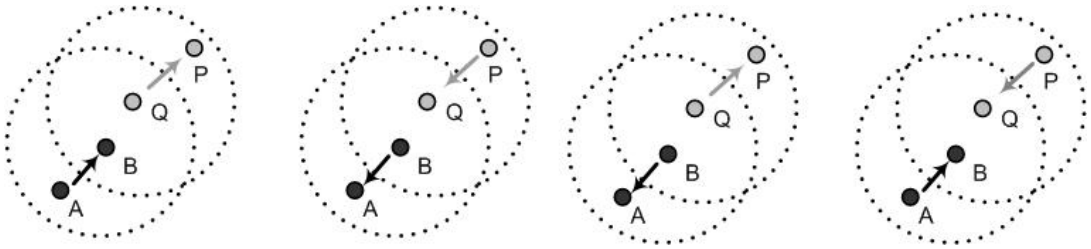
2.2 Παράλληλες μεταδόσεις στο 802.11

Στο 802.11 επιτρέπονται μεταδόσεις ανάμεσα σε δυο ζευγάρια κόμβων, μόνο όταν στη γειτονία του ενός ζευγαριού δεν υπάρχει ούτε ο αποστολέας ούτε ο παραλήπτης της μετάδοσης του άλλου ζευγαριού. Δεν επιτρέπεται δηλαδή η μετάδοση ακόμα και αν κόμβος-αποστολέας της δεύτερης μετάδοσης έχει στη γειτονιά του μόνο τον κόμβο-αποστολέα της πρώτης μετάδοσης.

Πολλά ζευγάρια αποστολέα-δέκτη μπορούν να είναι αποσυνδεδεμένα, δηλαδή κανένας κόμβος-αποστολέας να μην είναι γείτονας με κάποιον κόμβο-δέκτη διαφορετικής μετάδοσης. Σε αυτή την περίπτωση, αν το MAC επιτρέπει δύο ή περισσότερες ταυτόχρονες μεταδόσεις σε κάποιους από τους κόμβους-δέκτες που είναι γείτονες ή σε κάποιους από τους κόμβους-αποστολείς που είναι γείτονες, τότε θα έχει τεράστια βελτίωση η ρυθμαπόδοση του συστήματος. Σε αυτό ακριβώς το σημείο στηρίχθηκε και το MACA-P. Αντίθετα, η RTS/CTS ανταλλαγή που γίνεται στο MAC 802.11 πρωτόκολλο δεν επιτρέπει τέτοια παράλληλη λειτουργία. Πριν συνεχίσουμε και εξηγήσουμε γιατί συμβαίνει αυτό, πρέπει να εξετάσουμε πρώτα την παρακάτω παρατήρηση, η οποία πρέπει να υποστηρίζεται από ένα ασύρματο MAC.

- ***Παρατήρηση SRS:*** *αν κάποιος κόμβος ενεργεί σαν αποστολέας, μόνο ένας άλλος κόμβος (ο δέκτης της συγκεκριμένης μεταδιδόμενης πληροφορίας) στη γειτονιά του ενός* hop *μπορεί να παραλάβει την πληροφορία. Κανένας άλλος κόμβος στη γειτονιά μετάδοσης δεν μπορεί να παραλάβει μια ταυτόχρονη μετάδοση από κάποιο άλλο αποστολέα. Ομοίως, αν κάποιος κόμβος ενεργεί σαν δέκτης, μόνο ένας άλλος κόμβος (ο αποστολέας της συγκεκριμένης μεταδιδόμενης πληροφορίας) στη γειτονιά τους ενός* hop *επιτρέπεται να μεταδώσει.*

Παρακάτω θα μελετήσουμε κάποια παραδείγματα και θα δούμε πως κάποιες μεταδόσεις ενώ είναι “ιδανικά” εφικτές, το MAC 802.11 δεν τις επιτρέπει.



Σχήμα 2.1: Σενάρια παράλληλης μετάδοσης

Μελετώντας το σχήμα 2.1 που φαίνεται παραπάνω εξετάζουμε τις 4 περιπτώσεις. Στις περιπτώσεις αυτές ο B και ο Q είναι γείτονες του ενός hop, τα όρια μετάδοσης του A δεν περιλαμβάνουν τον Q (και το αντίθετο), και τα όρια μετάδοσης του P δεν περιλαμβάνουν το B (και το αντίθετο).

Στην πρώτη περίπτωση (α) είναι φανερό ότι η μετάδοση από το Q στο P θα παρέμβαινε της μετάδοσης από το A στο B, αφού το B είναι στα όρια μετάδοσης του Q. Αυτό παραβιάζει άλλωστε και την παραπάνω SRS παρατήρηση, αφού στη γειτονιά του Q (που στη συγκεκριμένη μετάδοση είναι ο αποστολέας) υπάρχουν δυο κόμβοι οι οποίοι έχουν το ρόλο του δέκτη, ο B και ο P.

Παρόμοια είναι και η δεύτερη περίπτωση (β), με την διαφορά ότι εδώ αποστολέας είναι ο B και παραλήπτης της συγκεκριμένης μετάδοσης είναι ο A, αλλά στη γειτονιά του ενός hop του B υπάρχει και άλλος ένας κόμβος παραλήπτης, ο Q. Οι περιπτώσεις αυτές δεν είναι εφικτές.

Ας μελετήσουμε προσεκτικά τώρα την τρίτη περίπτωση (γ). Αφού στα όρια μετάδοσης του B δε βρίσκεται ο P και στα όρια μετάδοσης του Q δε βρίσκεται ο A, οι δυο μεταδόσεις μπορούν να πραγματοποιηθούν παράλληλα. Δεν παραβιάζεται άλλωστε η SRS παρατήρηση αφού οι γειτονικοί κόμβοι που βρίσκονται σε διαφορετική μετάδοση (στην περίπτωση μας ο B και ο Q) έχουν τον ίδιο ρόλο, δηλαδή είναι και οι δυο κόμβοι-αποστολείς.

Το ίδιο ισχύει και στην τέταρτη περίπτωση (δ). Ο Q δεν είναι στα όρια μετάδοσης του A και ο B έξω από τα όρια μετάδοσης του P, άρα δεν παραβιάζεται η SRS και οι δυο μεταδόσεις μπορούν να γίνουν ταυτόχρονα.

Όπως είπαμε και παραπάνω, το 802.11 MAC δεν επιτρέπει σε δυο γείτονες ενός hop να είναι ταυτόχρονα αποστολείς ή δέκτες. Έχοντας υπόψη την περιγραφή που κάναμε παραπάνω για το DCF, θα εξηγήσουμε τώρα γιατί τα σενάρια μετάδοσης (γ) και (δ) του σχήματος 1 δεν επιτρέπονται από το MAC 802.11.

Στο σχήμα 2.1(γ), ο B στέλνει RTS στον A με σκοπό να του μεταδώσει δεδομένα. Το RTS που στέλνει ο B καταλαμβάνει το κανάλι για την επόμενη χρονική διάρκεια T_{RTS} και έτσι ο Q δεν μπορεί να στείλει RTS κατά τη διάρκεια αυτού του χρόνου στον P για να μπορέσει να του μεταδώσει τα δεδομένα που επιθυμεί. Έτσι ενώ φαίνεται δυνατό οι κόμβοι B και Q να μπορούν να πραγματοποιήσουν ταυτόχρονα τις μεταδόσεις τους, το MAC 802.11 εμποδίζει αυτή τη λειτουργία.

Ομοίως στο σχήμα 1(δ), ο B στέλνει ένα CTS σαν απάντηση για το RTS του A. Καθώς ο Q είναι στα όρια μετάδοσης του B, γνωρίζει τώρα ότι ο B έχει κατειλημμένο το κανάλι κατά τη διάρκεια του χρόνου T_{CTS}. Κατά τη διάρκεια αυτού του χρόνου ας υποθέσουμε ότι ο P στέλνει ένα RTS στον Q με σκοπό να μεταδώσει δεδομένα. Εντούτοις ο Q δεν μπορεί να απαντήσει εφόσον γνωρίζει ότι ο B έχει κατειλημμένο το κανάλι για κάποια περίοδο που

θα συμπίπτει με τη μετάδοση των δεδομένων από τον P. Έτσι πάλι δεν μπορεί να γίνει παράλληλη μετάδοση παρ' όλο που και εδώ φαίνεται ότι οι κόμβοι A και P μπορούν να πραγματοποιήσουν ταυτόχρονα τις μεταδόσεις τους.

Ένας λόγος για τον οποίο το 802.11 είναι υπερβολικά περιοριστικό, όπως φαίνεται και από τις δυο παραπάνω περιπτώσεις, είναι επειδή ο ρόλος κάθε κόμβου ως αποστολέα (tx) και ως παραλήπτη (rx) εναλλάσσεται πολλές φορές κατά τη διάρκεια της μεταφοράς ενός πακέτου, χωρίς να έχουν επακριβή επίγνωση του πότε γίνεται αυτή η αλλαγή του ρόλου τους.

Για παράδειγμα, στο σχήμα 1(δ), ο A έχει το ρόλο του tx για τη φάση του RTS και της μεταφοράς δεδομένων, ενώ ο B έχει τον rx ρόλο κατά τη διάρκεια αυτών των δυο φάσεων. Στις φάσεις του CTS και ACK ο B έχει τον tx ρόλο, ενώ ο A τον rx ρόλο. Θεωρώντας ότι η τετραμερής χειραψία μεταξύ του P και του Q ξεκινά ενώ η φάση μεταφοράς δεδομένων στην τετραμερή χειραψία του A με τον B είναι σε εξέλιξη, το RTS του P θα ληφθεί σωστά από το Q. Επομένως για να απαντήσει με CTS, ο Q πρέπει να πάρει τον tx ρόλο και αυτό θα παραβιάσει την παρατήρηση της SRS κατάστασης που είδαμε νωρίτερα. Για παράδειγμα, μετάδοση CTS από τον Q θα παρεμποδιστεί από τη μεταφορά δεδομένων του A στο B. Παρόμοιος συλλογισμός επίσης εξηγεί γιατί το σχέδιο μετάδοσης του σχήματος 1(γ) δεν επιτρέπεται. Θεωρώντας ότι ο Q θέλει να θέσει σε λειτουργία ένα RTS στο P κατά τη διάρκεια που η φάση μεταφοράς δεδομένων από το B στο A είναι σε λειτουργία, το RTS από το Q θα φτάσει στο P χωρίς παρεμβολή, αλλά το CTS από το P δε θα ληφθεί σωστά από το Q αφού το CTS και η μεταφορά δεδομένων από το B θα γίνουν ταυτόχρονα, και θα παραβιαστεί πάλι η παρατήρηση της SRS κατάστασης.

Ακόμη μια σημαντική επισήμανση που κάνουμε εδώ είναι ότι τα RTS/CTS που ανταλλάσσονται μεταξύ του ζευγαριού αποστολέα-παραλήπτη (π.χ. για μια μετάδοση μεταξύ P και Q) δεν μπορεί να πραγματοποιηθεί ταυτόχρονα με μια μετάδοση δεδομένων μεταξύ κάποιου άλλου ζεύγους (π.χ. μεταξύ A και B). Επίσης η ανταλλαγή RTS/CTS μεταξύ δυο ζευγαριών δεν μπορεί να γίνει ταυτόχρονα γιατί θα υπήρχαν συγκρούσεις.

2.3 Το πρωτόκολλο MACA-P

Το MACA-P είναι ένα MAC πρωτόκολλο βασισμένο στο CSMA-CA, σχεδιασμένο για να αυξήσει τη δυναμική για πολλαπλές ταυτόχρονες επικοινωνίες σε ένα ad hoc δίκτυο. Είναι βασισμένο σε μια απλή τροποποίηση του 802.11 MAC και διατηρεί τη βασική ιδιότητα της διανομής, βασιζόμενης στην πρόσβαση στο κοινό φυσικό κανάλι. Επιπλέον διατηρεί το μηχανισμό χειραψίας RTS/CTS για πολλαπλές μεταδόσεις, παρ' όλο που εισάγει την

πιθανότητα ενός επιπρόσθετου μηνύματος ελέγχου RTS' που μπορεί να χρειαστεί να σταλεί για προειδοποίηση των γειτονικών κόμβων, αν υπάρξει αλλαγή στο διάστημα μετάδοσης των δεδομένων. Το MACA-P σχεδιάσθηκε από την παρατήρηση ότι το κοινό κανάλι δικτύων, που χρησιμοποιεί η CSMA-CA αρχή, έχει μόνο ένα βασικό περιορισμό στην ταυτόχρονη πολλαπλή μετάδοση:

> Ø *Αν ένας κόμβος έχει ήδη αρχίσει τη λήψη πληροφορίας από ένα γειτονικό κόμβο, κανένας άλλος κόμβος (εκτός από τον κόμβο επικοινωνίας) στην* one-hoc *γειτονιά δεν μπορεί να μπει σε ταυτόχρονη επικοινωνία.*

Συνεπώς, η θεμελιώδης αρχή του MACA-P αλγορίθμου πολλαπλής πρόσβασης βασίζεται στο να σιγουρευτεί ότι σε γειτονικούς κόμβους οι χρόνοι εκπομπής ή λήψης αποφεύγουν τις συγκρούσεις που προκαλούνται από ταυτόχρονες πολλαπλές μεταδόσεις στους κόμβους λήψης.

2.3.1 Διαφορές MACA-P με 802.11

Παρακάτω θα αναφερθούν και θα αναλυθούν οι τροποποιήσεις που ορίζει το πρωτόκολλο MACA-P σε σχέση με το 802.11

2.3.1.1 Κενό ελέγχου (control gap)

Το MACA-P πρωτόκολλο που προτάθηκε για να επιτρέπει τις παράλληλες μεταδόσεις όπου αυτό είναι εφικτό, στηρίχθηκε στην φιλοσοφία και τον τρόπο λειτουργίας του MAC 802.11, διατηρώντας την τετραμερή χειραψία και τα μηνύματα ελέγχου RTS/CTS, τροποποιώντας όμως τη λειτουργία τους ώστε να μπορούν να γίνονται οι μεταδόσεις παράλληλα αποφεύγοντας τις πιθανές συγκρούσεις.

Αφού μελετήθηκε κάτω από ποιες προϋποθέσεις είναι δυνατό δύο γειτονικά ζευγάρια να προγραμματίζουν ταυτόχρονα τις μεταφορές δεδομένων τους και γιατί ο μηχανισμός RTS/CTS που χρησιμοποιείται από το 802.11 MAC αποτρέπει τις ταυτόχρονες μεταδόσεις, οι δημιουργοί του MACA-P πρότειναν την εισαγωγή ενός κενού μεταξύ της ανταλλαγή RTS/CTS και της επόμενης μετάδοσης δεδομένων. Αυτό το κενό, το οποίο αποκαλείται φάση ελέγχου (*control phase)* εξυπηρετεί τους παρακάτω σκοπούς:

- Κατά τη διάρκεια μιας ανταλλαγής RTS/CTS από ένα ζευγάρι (π.χ. A, B), επιτρέπει σε άλλα γειτονικά ζευγάρια (π.χ. Q, P), τα οποία μπορούν να προγραμματίσουν μια ταυτόχρονη μεταφορά δεδομένων να

ανταλλάξουν τα μηνύματα ελέγχου RTS/CTS για το συγχρονισμό τους, κατά τη διάρκεια του «κενού» της φάσης ελέγχου, αφού δεν μπορεί να γίνει ταυτόχρονα η ανταλλαγή RTS/CTS μεταξύ δυο ζευγαριών.

- Επιτρέπει σε επόμενο ζευγάρι, (όπως το P με το Q) να προγραμματίσει τη φάση μεταφοράς δεδομένων του με τη φάση μεταφοράς δεδομένων του πρώτου ζευγαριού, προγραμματίζοντας τη δική του μεταφορά δεδομένων στο τέλος του κενού της φάσεως ελέγχου. Το κενό ελέγχου τοποθετείται σε θέση από το πρώτο ζευγάρι (A, B), ενώ η επόμενη ανταλλαγή RTS/CTS από γειτονικό ζευγάρι (P, Q) δεν ορίζει ξανά νέο κενό. Απλά, το επόμενο ζευγάρι χρησιμοποιεί το τμήμα του κενού ελέγχου που έχει μείνει για να προγραμματίσει τη δική του μεταφορά δεδομένων με αυτήν του πρώτου ζευγαριού.

2.3.1.2 RTS/CTS πλαίσια

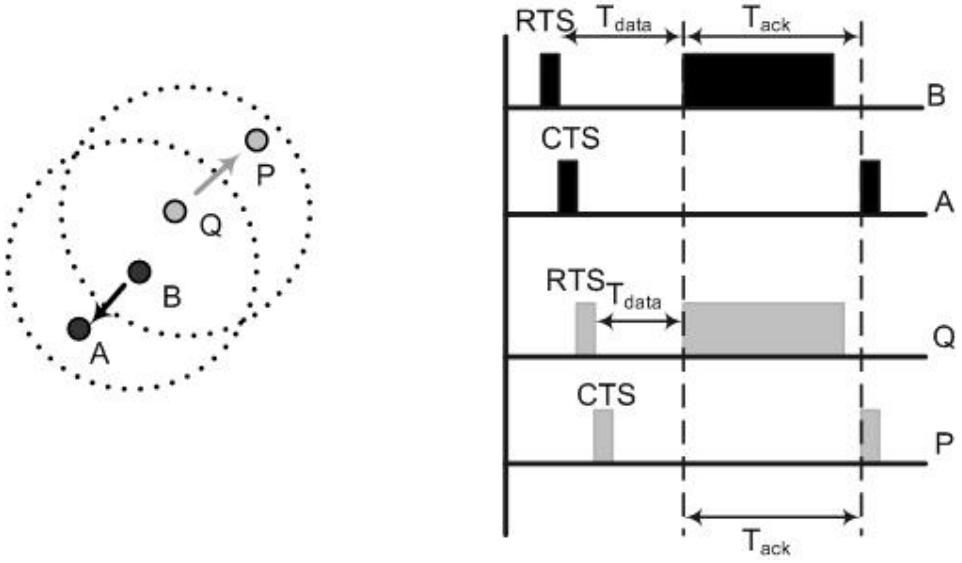
2.3.1.2.1 Προσθήκη χρονικών διαρκειών

Η πρώτη βελτίωση που έγινε στο πρωτόκολλο ήταν η προσθήκη επιπλέον πληροφορίας στα μηνύματα RTS και CTS ώστε να περιγράφουν λεπτομερώς τη διάρκεια της φάσης μεταφοράς δεδομένων και της φάσης του ACK, ούτως ώστε να επιτρέπεται στους άλλους κόμβους να γνωρίζουν ακριβώς πότε οι δύο κόμβοι που συσχετίζονται με την υπό εξέταση μετάδοση αλλάζουν τους ρόλους τους σε tx και rx. Αυτό έγινε με την λεπτομερή σύσταση δυο χρονικών διαρκειών στα μηνύματα ελέγχου RTS και CTS αντί για μια χρονική διάρκεια που είχε το MAC 802.11 :

- T_{DATA} : καθορίζεται ως η χρονική διάρκεια μετά τη λήψη του μηνύματος ελέγχου, που δείχνει το χρόνο έναρξης της μεταφοράς δεδομένων.
- T_{ACK} : καθορίζεται ως η χρονική διάρκεια μετά τη λήψη του μηνύματος ελέγχου, που δείχνει το χρόνο έναρξης του μηνύματος ACK.

Με την εισαγωγή των δυο χρονικών διαρκειών, διαχωρίζεται λεπτομερώς η φάση ελέγχου, η φάση μεταφοράς δεδομένων και το ACK. Η χρονική διάρκεια T_{DATA} ενημερώνει όλους τους γειτονικούς κόμβους για την διάρκεια της φάσης ελέγχου και επιτρέπει την πιθανότητα μια παράλληλη μετάδοση που έχει προγραμματιστεί, να ξεκινήσει τη μεταφορά δεδομένων της με το τέλος της χρονικής αυτής διάρκειας. Η δεύτερη χρονική διάρκεια T_{ACK} είναι απαραίτητη για την «ευθυγράμμιση» των ACK πακέτων των παραλλήλων μεταδόσεων. Στο σχήμα 2.2 παρακάτω, ο B ακούει το RTS που στέλνει ο Q στον P, και είναι ενήμερος ότι ένας γειτονικός κόμβος ξεκινά μια μετάδοση. Αν έχει πακέτο για να μεταδώσει αυτό το χρονικό διάστημα, θα στείλει ένα

RTS του οποίου ο T_{DATA} είναι “ευθυγραμμισμένος” με το χρόνο εκκίνησης της μεταφοράς δεδομένων του Q. Τόσο το μήνυμα RTS όσο και το CTS μεταφέρουν τις δύο χρονικές διάρκειες έτσι ώστε οι κόμβοι που είναι γείτονες είτε του αποστολέα είτε του παραλήπτη να μαθαίνουν για τις προγραμματισμένες μεταδόσεις δεδομένων και ACK, γνωρίζοντας τις τιμές των χρονικών διαρκειών T_{DATA} και T_{ACK}.



Σχήμα 2.2: Παράλληλη μετάδοση στο MACA-P

2.3.1.2.2 Κατάσταση γειτονικών κόμβων

Ο κάθε κόμβος χρειάζεται να διατηρεί την κατάσταση των γειτονικών του κόμβων, ακούγοντας τις συναλλαγές RTS/CTS από τους γείτονες του. Μελετώντας το σχήμα 2.2 ο B ξεκινά μια RTS/CTS ανταλλαγή με τον A. Καθώς ο Q ακούει το RTS από το B, θα ενημερώσει το NAV του ώστε να δείχνει ότι ο B έχει προγραμματίσει μια μετάδοση στον A. Για κάθε κόμβο που έχει ακούσει ένα RTS ή ένα CTS, ο κόμβος διατηρεί μια είσοδο στο NAV που αποτελείται από τη MAC διεύθυνση του γείτονα, αποστολέα ή παραλήπτη και τις χρονικές διάρκειες T_{DATA} και T_{ACK}. Αυτή η πληροφορία χρησιμοποιείται ως ακολούθως: αν κάποιος κόμβος επιθυμεί να στείλει ένα πακέτο δεδομένων, πρέπει να ελέγξει ότι καμία είσοδος στο NAV του δεν είναι σημειωμένη σαν παραλήπτης. Διαφορετικά θα παραβιαζόταν η SRS επισήμανση που αναφέραμε νωρίτερα. Ομοίως αν κάποιος κόμβος παραλάβει ένα RTS, δεν μπορεί να ανταποκριθεί με ένα CTS αν κάποια είσοδος στο NAV είναι σημειωμένη σαν αποστολέας. Επιπλέον από αυτόν το βασικό έλεγχο, το NAV επιτρέπει στους κόμβους να γνωρίζουν αν υπάρχει ήδη

μετάδοση προγραμματισμένη από κάποιο κόμβο της γειτονιάς τους και να χρησιμοποιούν την πληροφορία αυτή για να προγραμματίσουν μια δική τους παράλληλη μεταφορά δεδομένων. Για παράδειγμα, στο σχήμα 2.2, ο Q ενημερώνει το NAV του για το RTS που έχει ακούσει να στέλνει ο B στον A, και μετά χρησιμοποιεί την πληροφορία αυτή για να προγραμματίσει τη δική του παράλληλη μεταφορά δεδομένων, όπως εξηγήθηκε πιο πάνω.

2.3.1.2.3 Inflexible bit στο RTS

Το μήνυμα RTS βελτιώνεται επιπλέον κρατώντας ένα bit που το αποκαλούμε **inflexible bit**. Ο σκοπός αυτού του bit είναι να δείχνει στον παραλήπτη του RTS μηνύματος κατά πόσο η μετάδοση που πρόκειται να πραγματοποιηθεί μπορεί να αλλάξει, ανάλογα με το αν η συγκεκριμένη μετάδοση είναι ευθυγραμμισμένη με μια ήδη υπάρχουσα ή όχι. Αν το bit είναι "set" τότε δεν μπορεί να αλλάξει, γιατί η μετάδοση έχει προγραμματιστεί να ευθυγραμμιστεί με μια άλλη μετάδοση που υπάρχει στη γειτονιά του αποστολέα.

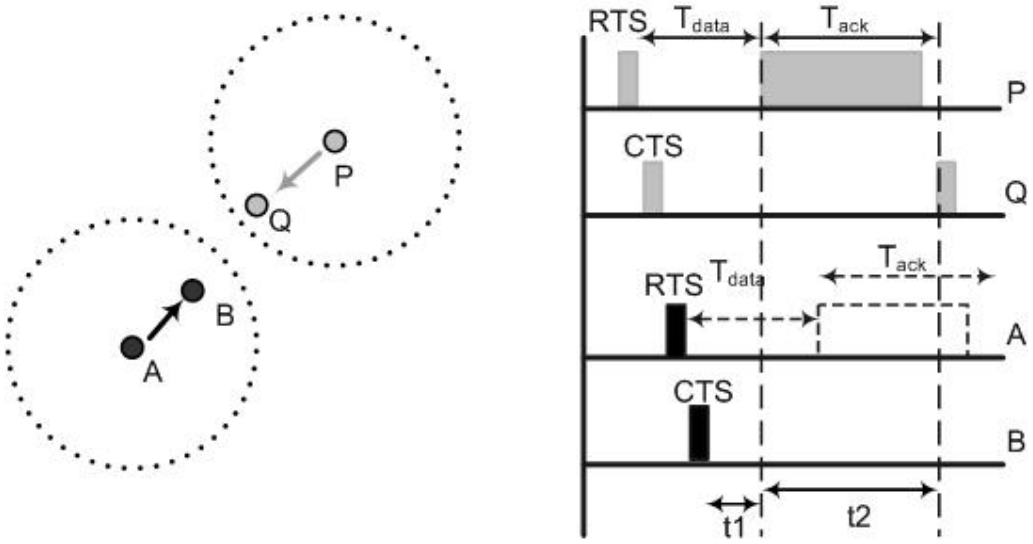
2.3.1.2.4 Τροποποίηση των T_{DATA} και T_{ACK} από το CTS

Μελετούμε πάλι το σχήμα 2.2. Όταν ο B στέλνει το RTS του στον A, το bit είναι "unset" αφού δεν υπάρχει άλλη μετάδοση στη γειτονιά του B. Ωστόσο μετά από αυτό, θεωρούμε ότι ο Q επιθυμεί να στείλει ένα πακέτο στο P. Το NAV του Q έχει ήδη ενημερωθεί με την προγραμματισμένη μετάδοση του B στο A, αφού έχει ακούσει το RTS που έστειλε ο B. Συνεπώς το Q στέλνει RTS στο P, ευθυγραμμίζοντας τη μετάδοση δεδομένων του με αυτή του B με το inflexible bit του RTS του να είναι set. Υπάρχει βέβαια η πιθανότητα ο προτεινόμενος προγραμματισμός από τον αποστολέα να είναι ανέφικτος για το δέκτη, λόγω της πληροφορίας της δικής του γειτονιάς, αλλά το τροποποιημένο «πρόγραμμα» να είναι εφικτό για τον παραλήπτη. Ωστόσο, αν το inflexible bit είναι set στο RTS, τότε ο παραλήπτης πρέπει είτε να δεχθεί τον προτεινόμενο προγραμματισμό (στέλνοντας ένα CTS πίσω, με τα ίδια T_{DATA} και T_{ACK} όπως και αυτά του RTS) είτε να την απορρίψει εντελώς (να μη στείλει πίσω CTS). Δεν μπορεί να στείλει πίσω ένα τροποποιημένο πρόγραμμα με το CTS.

Τροποποιημένο πρόγραμμα με το CTS μπορεί να στείλει ο παραλήπτης μόνο όταν το inflexible bit δεν είναι set. Μπορεί δηλαδή να αλλάξει τον προτεινόμενο προγραμματισμό του αποστολέα έτσι ώστε να μπορεί να ευθυγραμμίσει την προτεινόμενη μετάδοση δεδομένων από τον αποστολέα με μια υπάρχουσα προγραμματισμένη λήψη στη γειτονιά του. Αυτό γίνεται με

την τροποποίηση των λαμβανόμενων χρονικών διαρκειών T_{DATA} και T_{ACK} του RTS μηνύματος, και στέλνοντας πίσω τις τροποποιημένες τιμές τους μέσω του CTS μηνύματος.

Ας μελετήσουμε τώρα το σχήμα 2.3 που φαίνεται παρακάτω. Ο Β έχει ακούσει το CTS από το Q και είναι ενήμερος για την προγραμματισμένη λήψη του στη γειτονιά του. Άρα, όταν λαμβάνει ένα RTS από τον Α με το inflexible bit να είναι unset, απαντά με τα τροποποιημένα T_{DATA} και T_{ACK} (τα οποία φαίνονται στο σχήμα ως t1 και t2) έτσι ώστε η λήψη δεδομένων του Β από τον Α να γίνει ταυτόχρονα με την λήψη του Q.



Σχήμα 2.3: Τροποποίηση T_{data} και T_{ack} από το CTS

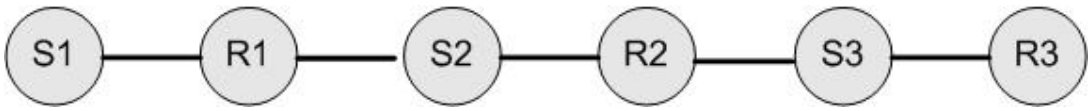
2.3.1.2.5 RTS' μήνυμα ελέγχου

Όταν ο αποστολέας μεταδίδει ένα RTS μήνυμα, οι γείτονες του αποστολέα ενημερώνουν τα NAV τους για αυτή τη μετάδοση, αφού έχουν πρώτα ακούσει το RTS μήνυμα που έχει σταλεί. Ωστόσο αν το προτεινόμενο πρόγραμμα αποστολής του RTS έχει τροποποιηθεί από τον παραλήπτη του RTS μηνύματος (όπως είχαμε αναφέρει νωρίτερα), οι γείτονες του αποστολέα δεν ενημερώνονται για το τροποποιημένο σχέδιο. Για να αποφευχθεί αυτή η

κατάσταση, ο αποστολέας στέλνει πάντα ένα άσκοπο RTS μήνυμα που περιέχει τα ενημερωμένα T_{DATA} και T_{ACK} που παραλαμβάνονται από το CTS. Ο σκοπός του συγκεκριμένου RTS (το οποίο ονομάζεται **RTS'**) δεν είναι να στείλει επιβεβαίωση για το CTS στον παραλήπτη αλλά για να ενημερώσει τους υπόλοιπους γείτονες του αποστολέα για το τροποποιημένο σχέδιο. Το RTS' στέλνεται ακολουθούμενο από μια SIFS (Short Inter-frame Space) χρονική περίοδο μετά από τη λήψη του τροποποιημένου CTS.

Μια δεύτερη χρησιμότητα του RTS' μηνύματος είναι να ακυρώνει το προηγούμενο σχέδιο αποστολής που είχε γίνει δια μέσου του πρώτου RTS. Αυτό συμβαίνει όταν ο αποστολέας δε λαμβάνει το CTS από τον προοριζόμενο παραλήπτη του RTS. Υπάρχει πιθανότητα ο παραλήπτης να μην μπορεί να απαντήσει στον κόμβο που έχει στείλει το RTS για διάφορους λόγους, είτε να υπάρχει μια «τρέχουσα» λήψη στη γειτονιά του παραλήπτη είτε το προτεινόμενο σχέδιο να παραβιάζει την SRS επισήμανση στον παραλήπτη. Το CTS μπορεί επίσης να χαθεί από λάθη στο κανάλι. Ωστόσο, οι γείτονες του αποστολέα έχουν ενημερώσει τις αντίστοιχες NAV αφού έχουν ήδη ακούσει το αρχικό RTS. Σε αυτή την περίπτωση, μετά την αναμονή της SIFS περιόδου συν της CTS περιόδου, ο αποστολέας μεταδίδει ένα RTS' μήνυμα με μηδενισμένα τα T_{DATA} και T_{ACK}. Στη λήψη αυτού του μηνύματος, οι γείτονες απομακρύνουν στην αντίστοιχη είσοδο τους τον αποστολέα από τη δική τους αντίστοιχη NAV. Αυτό επιτρέπει στους γείτονες να χρησιμοποιούν το κανάλι κατά τη διάρκεια του χρόνου που αρχικά ήταν κρατημένο από το RTS, αφού η μετάδοση που κρατούσε το κανάλι έχει ακυρωθεί.

Ένα σημαντικό προτέρημα της χρησιμοποίησης του RTS' για ακύρωση ενός προτεινόμενου σχεδίου που έγινε νωρίτερα είναι να αποφεύγει να "κλειδώνει" κάποιες μεταδόσεις οι οποίες είναι προφανές ότι μπορούν να γίνουν παράλληλα. Για να το εξηγήσουμε αυτό ας θεωρήσουμε μια αλυσίδα από κόμβους όπως στο παρακάτω σχήμα.



Σχήμα 2.4: Αλυσίδα

Ο αποστολέας S3 είχε επιτυχή ανταλλαγή RTS/CTS με τον παραλήπτη R3. Αν κατά τη διάρκεια της φάσης ελέγχου ή της φάσης ανταλλαγής δεδομένων του S3-R3, υποθέτουμε ότι ο S2 στέλνει RTS στον R2 για μια προγραμματισμένη S2-R2 μετάδοση. Ο R2 δεν μπορεί να απαντήσει με ένα

CTS, ενόσω υπάρχει μια προγραμματισμένη μετάδοση στη γειτονιά του (και ο γειτονικός του κόμβος είναι αποστολέας και όχι παραλήπτης). Μετά την αποστολή του RTS από το S2, θεωρούμε ότι ο S1 στέλνει ένα RTS στο R1 για την προγραμματισμένη S1-R1 μετάδοση. Αφού ο R1 έχει ακούσει το RTS του S2, δεν μπορεί να απαντήσει στον S1. Ως αποτέλεσμα η μετάδοση S3-R3 έχει "κλειδώσει" τόσο την S2-R2 μετάδοση όσο και την S1-R1. Ωστόσο, είναι φανερό ότι η μετάδοση S3-R3 και η S1-R1 μπορούν να γίνουν παράλληλα, χωρίς να υπάρξει το παραμικρό πρόβλημα. Αυτό επιτυγχάνεται με τη χρήση του RTS'. Όταν ο S2 δε λάβει το CTS από τον R2, στέλνει ένα RTS', ελευθερώνοντας έτσι το κανάλι για να χρησιμοποιηθεί από οποιονδήποτε γείτονα. Αυτός είναι ο λόγος που οδήγησε στην ανάγκη να μπει ένα επιπρόσθετο μήνυμα RTS' στο MACA-P.

2.3.1.3 Βασικός Μηχανισμός Πρόσβασης

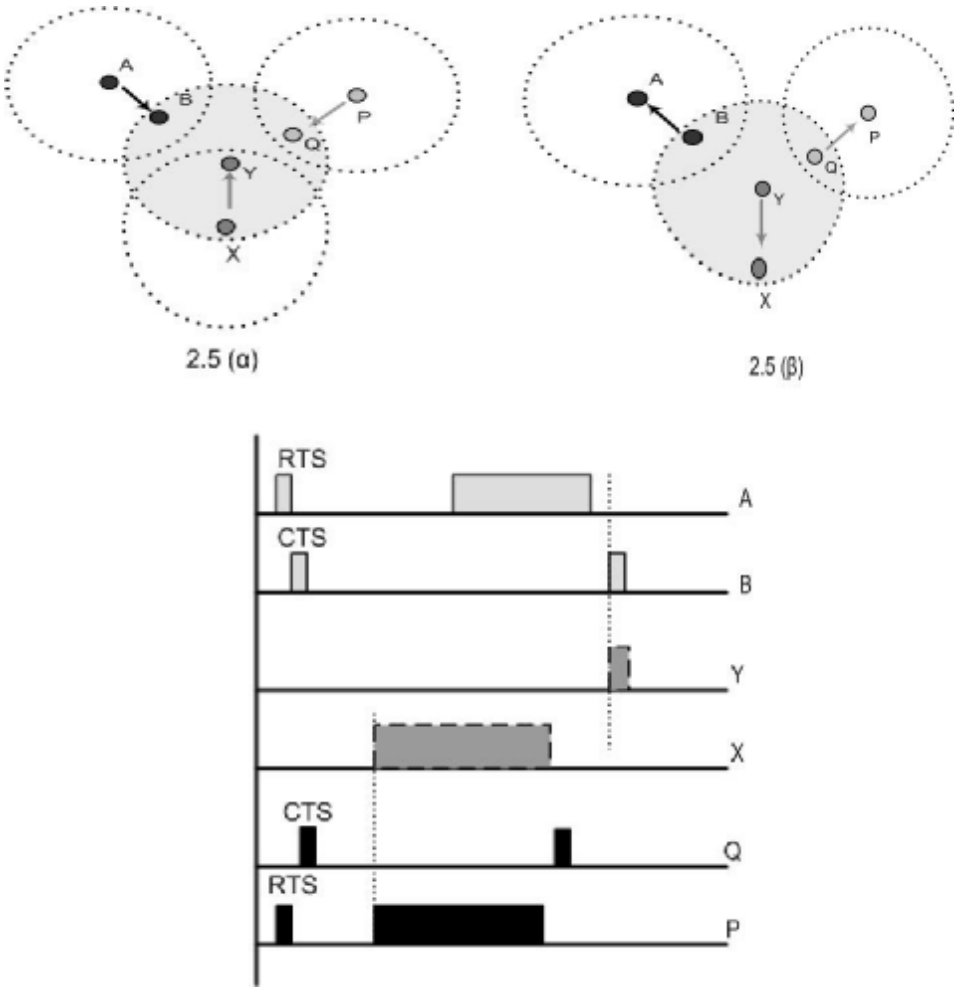
Στο MAC 802.11, όπως αναφέρθηκε και στο πρώτο κεφάλαιο, όταν ένας κόμβος είναι ο πρώτος που κάνει αίτηση για να μεταδώσει ένα πακέτο, τότε ανιχνεύει ότι το κανάλι είναι αδρανές. Δεν μπορεί όμως να μεταδώσει αμέσως, αλλά πρέπει να περιμένει μια χρονική περίοδο ίση με DIFS (Distributed Inter-Frame Space) και μετά να μεταδώσει. Ένας τυχαίος backoff μετρητής σε διάταξη (0, Congestion Window) επιλέγεται με έναν τυχαίο αριθμό από time slots πριν ξεκινήσει η μετάδοση. Αν οι κόμβοι που περιμένουν είναι περισσότεροι, τότε ο κόμβος που έχει διαλέξει το μικρότερο αριθμό από slots θα έχει προτεραιότητα να μεταδώσει. Αν το κανάλι είναι αδρανές στην εκπνοή του χρονομέτρου, η μετάδοση πακέτου θα τεθεί σε λειτουργία. Το χρονόμετρο παγώνει οποιαδήποτε στιγμή το κανάλι καταληφθεί. Αλλιώς το παράθυρο συμφόρησης διπλασιάζεται και ο τυχαίος μετρητής διαλέγεται από καινούργιο παράθυρο. Το MACA-P σχεδιάστηκε έτσι ώστε να διατηρεί αυτό το μηχανισμό του MAC 802.11.

Η χρονική περίοδος που πρέπει να περιμένει ένας κόμβος πριν μεταδώσει το μήνυμα, μετά την ανίχνευση ότι το κανάλι είναι αδρανές στο 802.11 DCF MAC ονομάζεται DIFS. Στο MACA-P, από τη στιγμή που υπάρχει η δυνατότητα ενός RTS' ακολουθούμενο του RTS, ο κόμβος που ακούει το RTS και μετά ανιχνεύει το κανάλι αδρανές πρέπει να περιμένει αρκετά ώστε να δώσει τη δυνατότητα στον αποστολέα του RTS να μεταδώσει αν χρειαστεί ένα RTS'. Έτσι επιβάλλεται ο κόμβος που ανιχνεύει το κανάλι αδρανές μετά το άκουσμα του RTS να χρησιμοποιήσει μια αρκετά μεγάλη DIFS περίοδο για να δώσει τη δυνατότητα να ακούσει το RTS'. Για παράδειγμα πρέπει ο κόμβος που έχει διαλέξει να μεταδώσει το RTS από μόνος του, να το κάνει μόνο μετά το χρόνο που επιτρέπεται στον κόμβο που βρίσκεται στη γειτονιά του και προηγουμένως είχε στείλει RTS να μπορεί να στείλει και RTS'.

2.3.1.4 Πρόγραμμα των master transmissions

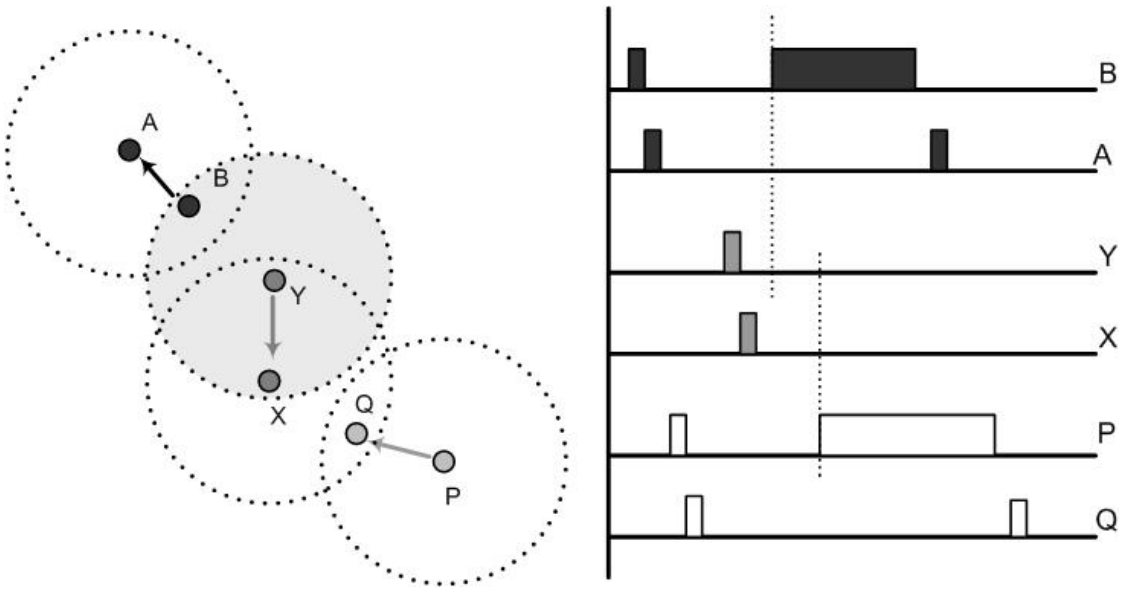
Το πρόγραμμα των master transmissions είναι αυτό που επιτρέπει στους γείτονες τόσο του αποστολέα όσο και του παραλήπτη να προγραμματίζουν τις δικές τους μεταδόσεις δεδομένων ευθυγραμμισμένες με τον master της γειτονιάς τους. Ένα σετ από ευθυγραμμισμένες μεταδόσεις δεδομένων αναφέρεται σαν σετ μεταδόσεων (transmission set). Ο αποστολέας δεδομένων του master transmission αναφέρεται ως master sender, και ομοίως ο παραλήπτης δεδομένων ως master recipient. Γενικά, η λέξη "master" χρησιμοποιείται όταν είναι ξεκάθαρο αν αναφέρεται για τον αποστολέα ή τον παραλήπτη. Παρακάτω αναγράφεται η απαραίτητη προϋπόθεση για το πότε ένα ζευγάρι αποστολέα/παραλήπτη μπορεί να προγραμματίσει μια μετάδοση εξαρτώμενη από τον αριθμό των masters στη γειτονιά τους:

Ένα ζευγάρι αποστολέα/παραλήπτη μπορεί να προγραμματίσει μια μετάδοση μόνο αν υπάρχει ένας μόνο master transmission *στη γειτονιά του αποστολέα ή ένας μόνο* master reception *στη γειτονιά του παραλήπτη, αλλά όχι και τα δυο.*



Σχήμα 2.5: Μετάδοση με 2 master transmission

Για να δούμε γιατί υπάρχει μια τέτοια προϋπόθεση, μελετούμε το σχήμα 2.5. Στο σχήμα 2.5(α), ο Y είναι γείτονας του B και του Q, αλλά ο B δεν είναι γείτονας του Q. Οι δύο μεταδόσεις από τον A στο B και από τον P στον Q έχουν προγραμματιστεί. Άρα ο Y έχει δύο masters, το B και τον Q. Ο X στέλνει ένα RTS στον Y. Για να μπορεί ο Y να ανταποκριθεί σε αυτή τη μετάδοση, πρέπει να ευθυγραμμίσει τη μετάδοση δεδομένων του X με αυτήν του P (με τον Q παραλήπτη) και να ευθυγραμμίσει επίσης τη δική του αποστολή ACK προς τον X με το ACK που στέλνει ο B στον A. Γενικά, αν ένας κόμβος έχει περισσότερους από ένα masters, πρέπει να ευθυγραμμίσει την προτεινόμενη μετάδοση δεδομένων με αυτήν του master με την πιο γρήγορη μετάδοση δεδομένων και να ευθυγραμμίσει το ACK με αυτήν του master με την πιο αργή ACK. Πρώτον, αυτό προσθέτει μεγάλη πολυπλοκότητα για να εφαρμοστεί κάποια λύση. Δεύτερον, όλοι οι masters (παραλήπτες) κόμβοι εκτός από αυτόν με την πιο αργή ACK, είναι μπλοκαρισμένοι να προγραμματίσουν κάποια λήψη μέχρι ο master transmission με την πιο αργή ACK τελειώσει. Στο σχήμα αυτό σημαίνει ότι ο Q δεν μπορεί να προγραμματίσει καμία άλλη λήψη (από τον P για παράδειγμα) πριν ο Y στείλει το ACK του στον X (ευθυγραμμίζοντας το, με το ACK από τον B στον A). Διαφορετικά, το επόμενο CTS από τον Q μπορεί να παρέμβει στη λήψη δεδομένων του Y. Γι' αυτό το MACA-P είναι πιο συντηρητικό και απλά απαγορεύει στους κόμβους με περισσότερους από ένα master να συμμετέχουν σε μια παράλληλη μετάδοση/λήψη. Στο σχήμα 2.5(β) φαίνεται κάτι ανάλογο για ένα κόμβο με περισσότερους από ένα master sender.



Σχήμα 2.6: Μετάδοση με master sender και master recipient

Ας μελετήσουμε τώρα το σχήμα 2.6, όπου και ο αποστολέας και ο παραλήπτης της προτιθέμενης μετάδοσης, έχουν ένα master ο καθένας. Στο σχήμα 2.6 θεωρούμε ότι και οι δυο μεταδόσεις από το B στο A και από το P στο Q έχουν προγραμματιστεί. Ο B είναι ο master του Y, και ο Q ο master του X. Ο Y στέλνει ένα RTS με το inflexible bit set. Καθώς ο X έχει ήδη ένα master, πρέπει να ευθυγραμμίσει όλες τις λήψεις του στο δικό του master, τον Q. Ωστόσο η προτιθέμενη προγραμματισμένη μετάδοση του Y είναι ευθυγραμμισμένη με τον master του Y, το B. Σε αυτή την περίπτωση ο X δε θα μπορεί να απαντήσει στο RTS του Y. Αυτό το παράδειγμα δείχνει ότι όταν και ο αποστολέας και ο παραλήπτης έχουν ένα master transmission και πρόγραμμα μετάδοσης στις αντίστοιχες γειτονιές τους, τότε δεν είναι δυνατό στο ζευγάρι αποστολέα / παραλήπτη να έχουν μετάδοση χωρίς σύγκρουση.

2.3.1.5 Μέγεθος πακέτων των master transmission

Μια σημαντική συνέπεια των πιο πάνω είναι ότι οι αποστολείς οι οποίοι έχουν master transmission μπορούν να εκπέμπουν πακέτα δεδομένων μόνο αν το μέγεθος των πακέτων αυτών είναι μικρότερο από το μέγεθος του πακέτου που εκπέμπει ο master transmission. Διαφορετικά, η εκπομπή δεδομένων σε μια παράλληλη μετάδοση θα παρεμβληθεί με την αποστολή ACK του master. Επίσης, το MACA-P συνιστά ένα κενό ελέγχου πριν τη φάση μεταφοράς δεδομένων.

Επομένως προκύπτουν δυο παρατηρήσεις:

1. Ο αποστολέας που δεν έχει master transmission στη γειτονιά του εφαρμόζει το MACA-P σε ένα πακέτο αν το μέγεθος του πακέτου είναι μεγαλύτερο από το κατώφλι. Για παράδειγμα το MACA-P εφαρμόζεται σε ένα “μεγάλο” πακέτο αφού ο μηχανισμός RTS/CTS δε χρησιμοποιείται για “μικρά” πακέτα

2. Ένας κόμβος μπορεί να κάνει μια παράλληλη μετάδοση με τον master μόνο αν ο χρόνος μετάδοσης δεδομένων του κόμβου δεν είναι μεγαλύτερος από αυτόν του master.

Η δεύτερη παρατήρηση υποδηλώνει ότι αν ο αποστολέας έχει ένα μικρό πακέτο να στείλει και υπάρχει master transmission, τότε ο αποστολέας θα προσπαθήσει να κάνει παράλληλα την εκπομπή του πακέτου με αυτήν του master.

Το MACA-P μειώνει τον RTS backoff timer μόνο όταν μπορεί να στείλει ένα RTS, όταν δηλαδή το κανάλι είναι ελεύθερο (συμπεριλαμβανομένης και

της φάσης ελέγχου) ή όταν ο αποστολέας είναι γνώστης για το πολύ ενός επιπλέον master transmission προγράμματος και για κανένα γειτονικό παραλήπτη. Η συμπεριφορά εξαρτάται από το μέγεθος του πακέτου. Ακόμα και για μικρού μεγέθους πακέτα (μικρότερα από το κατώφλι), το MACA-P προσπαθεί να ευθυγραμμίσει το πακέτο με το υπάρχον πρόγραμμα μετάδοσης αν αυτό είναι δυνατό. Αν ο χρόνος του χρονομέτρου τελειώσει και η προσπάθεια του RTS αποτύχει, τότε το MACA-P αμέσως χρησιμοποιεί το μηχανισμό MAC 802.11 διπλασιάζοντας τον backoff timer.

2.3.1.6 Αναμετάδοση του RTS πλαισίου

Όταν ένας κόμβος στέλνει ένα RTS και δε λαμβάνει ένα CTS ως απάντηση, θα εκπέμψει ένα RTS' για να ακυρώσει το πρόγραμμα μετάδοσης που "ανακοίνωσε" το RTS. Ο βασικός μηχανισμός πρόσβασης του 802.11 που είδαμε νωρίτερα, θα διπλασιάσει το congestion window (παράθυρο συμφόρησης) και θα ξαναπροσπαθήσει χρησιμοποιώντας μια τυχαία τιμή μέσα στο νέο congestion window. Κατά τη διάρκεια αυτής της περιόδου είναι πιθανό η κατάσταση της γειτονιάς να έχει αλλάξει. Για παράδειγμα, την πρώτη φορά ο κόμβος μπορεί να ήταν master transmitter (κανένας στη γειτονιά δεν είχε προγραμματισμένη μετάδοση). Ωστόσο, κατά τη διάρκεια της backoff περιόδου, κάποιος γείτονας μπορεί να είχε στείλει ένα RTS και να έγινε master transmitter σε αυτή τη διαδικασία. Έτσι, όταν ο κόμβος ξαναπροσπαθεί να στείλει ένα RTS, ίσως να μην μπορεί να χρησιμοποιήσει το ίδιο T_{DATA} όπως χρησιμοποίησε στην προηγούμενη προσπάθεια. Πρέπει να ξαναπροσπαθήσει να στείλει το RTS βασισμένος στην τωρινή κατάσταση της γειτονιάς.

3. Υλοποίηση MACA-P πρωτοκόλλου

Ο αλγόριθμος της υλοποίησης του πρωτοκόλλου στηρίχθηκε ακριβώς στις αλλαγές και τις προσθήκες του MACA-P σε σχέση με το 802.11, όπως αυτές αναλύθηκαν στο προηγούμενο κεφάλαιο.

3.1 Αλγόριθμος υλοποίησης

Ο αλγόριθμος του MACA-P χωρίζεται σε δυο μέρη, σχετικά με τις ενέργειες που πρέπει να ακολουθήσει ο κόμβος-αποστολέας και αυτές που πρέπει να ακολουθήσει ο κόμβος-παραλήπτης μιας μετάδοσης.

Παρακάτω παρατίθενται οι δύο αλγόριθμοι καθώς και τα διαγράμματα ροής τους.

Κόμβος αποστολέας:

```
Step 1 :
If (node N has packet P to send)
then wait till no neighbour is a recipient.

Step 2 :
If (#(master transmissions) > 1)
then idle till exactly one master scheduled.

Step 3 :
If ( exactly 1 master transmission scheduled)
then
        if (size of P) > (master transmission size)
        then
                wait till master transmission complets
        else
                send RTS with inflexible-bit set with Tdata and Tack aligned with
        master
                wait till CTS is received
                if no CTS is received
                then
                        send RTS' canceling prior RTS
                        retry RTS using 802.11 basic access method
                else
                        Schedule transmission of P at Tdata
                        Wait for ACK at Tack
                        If (ACK is not received at Tack)
```

```
                    then Goto NoSuccess

else    /* there are no master transmissions scheduled */
        if (size of packet) < threshold
        then
                use 802.11 RTS/CTS DCF /* no control gap before data*/
        else
                Node N is the master
                Send RTS with inflexible-bit unset
                Wait till CTS is received
                If (CTS is not received)
                then
                        send RTS' canceling prior RTS
                        N is no longer a master
                        retry RTS using 802.11 basic access method
                else
                        if (CTS is received with modified Tdata and Tack)
                        then
                            send RTS' with inflexible-bit set with modified Tdata
                        and Tack
                            schedule transmission of P after modified Tdata
                            wait for ACK after modified Tack
                            If (ACK is not received at modified Tack)
                            then
                              Goto NoSuccess
                            else
                              schedule transmission after Tdata
                              Wait for ACK after Tack
                              If (ACK is not received at Tack)
                              then Goto NoSuccess

Goto Step1
NoSuccess :
                Update #tries for packet P
                If #tries > max-tries
                then Drop P from transmission queue
                Goto Step1
```

Κόμβος Παραλήπτης :

Step 1 :
Wait till RTS is received from a node N
If inflexible bit is set
then
if (R has no master recipient))
then
Set R as a master recipient
Send CTS back with same Tdata and Tack as RTS
Else / cannot schedule a inflexible sender with an existing master */*
No CTS is sent
Goto Step1
else
if #(master recipients for R) == 0
then
set R as a master recipient
Send CTS back same Tdata and Tack as RTS
Goto WaitforData

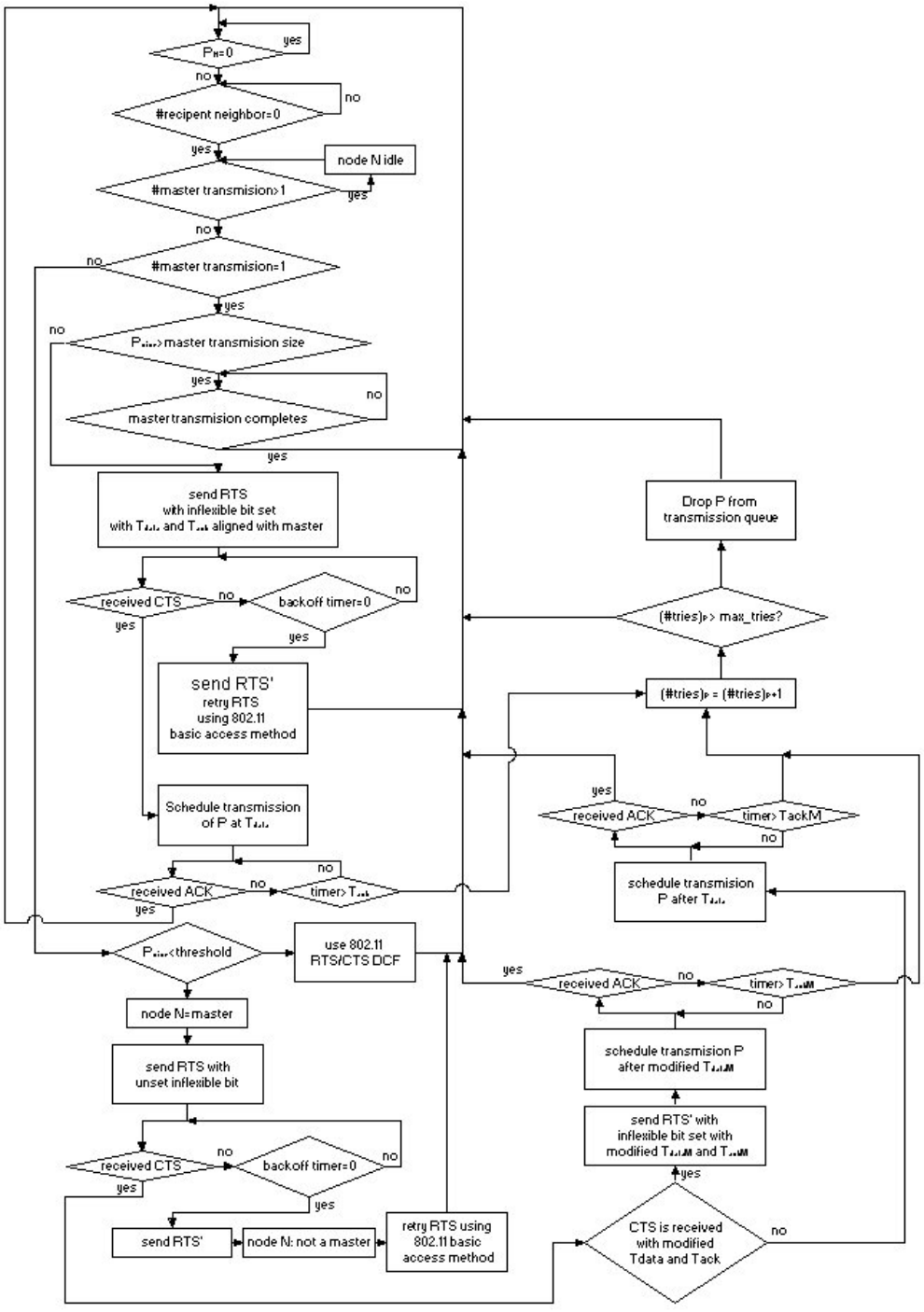
If #(master recipients for R) == 1
then
Send CTS with modified Tdata and Tack aligned with existing master
Else / (more than 1 master recipients) */*
Do not send CTS
Goto Step1

WaitforData :

Wait for transmission of data from N at (modified) Tdata
If transmission received correctly, then send ACK at (modified) Tack
Goto Step1

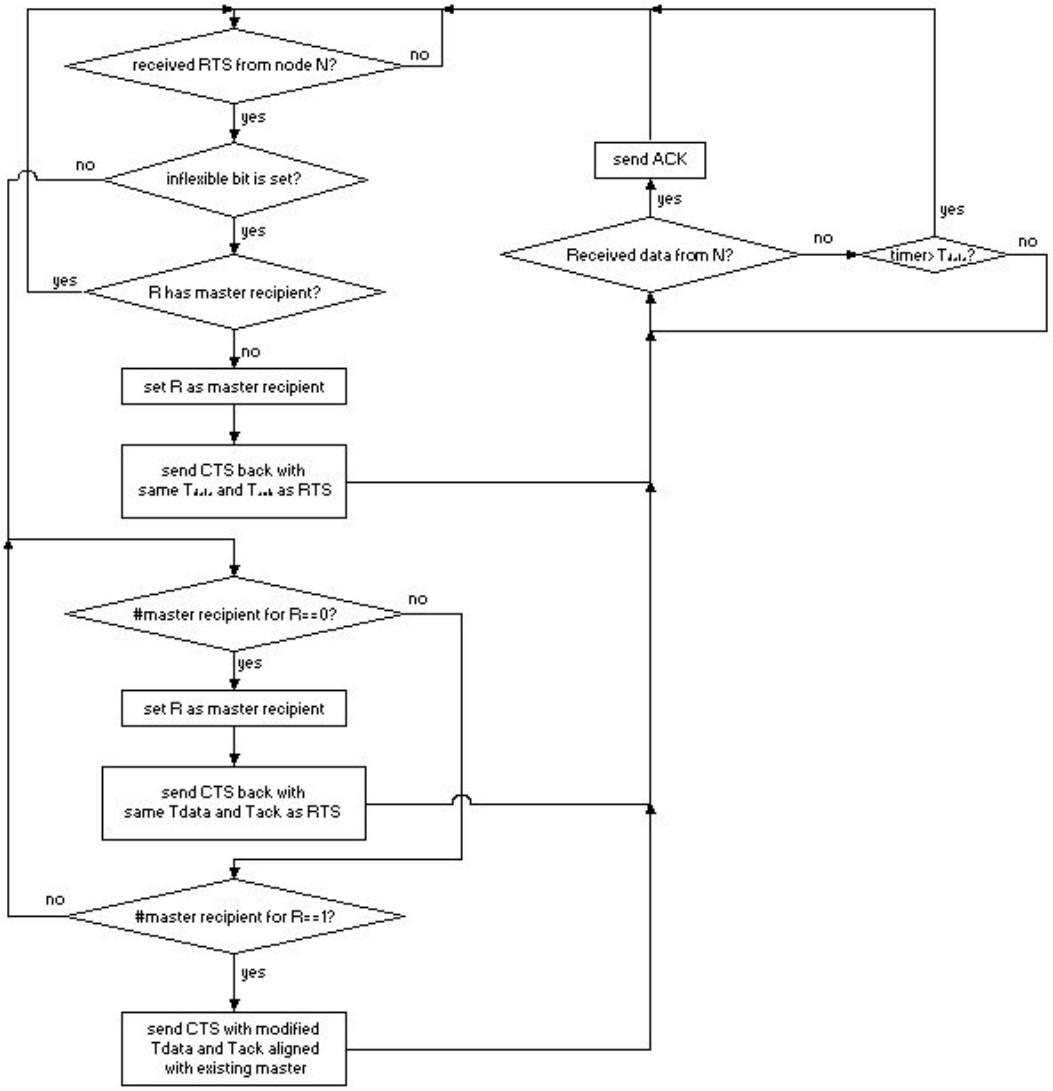
Διαγράμματα Ροής

1. Κόμβος Αποστολέας



Σχήμα 3.1: Διάγραμμα ροής Κόμβου-Αποστολέα

2. Κόμβος παραλήπτης



Σχήμα 3.2: Διάγραμμα ροής Κόμβου-Παραλήπτη

3.2 Υλοποίηση κώδικα στο ns2.28

Η υλοποίηση στηρίχθηκε στον ήδη υπάρχοντα κώδικα του ns2.28 για το 802.11. Έγιναν προσθήκες κάποιων αναγκαίων συναρτήσεων καθώς και overwrite κάποιες συναρτήσεις του 802.11 έτσι ώστε να προσαρμοστούν στις ανάγκες του MACA-P. Οι συναρτήσεις αυτές αναφέρονται παρακάτω.
Πιο συγκεκριμένα έγιναν οι εξής αλλαγές-προσθήκες:

Συναρτήσεις που προστέθηκαν:

- **navDataTimer ( ):** Ο timer αυτός κρατάει τη χρονική διάρκεια T_{DATA}. Λειτουργεί όπως ακριβώς και ο navTimer του 802.11, τίθεται όμως με διαφορετική τιμή (την T_{DATA})
- **is_idleMacap ( ):** Η συνάρτηση αυτή είναι αντίστοιχη της is_idle ( ) του 802.11. Ελέγχει αν κάποιος κόμβος βρίσκεται στην περίοδο που μεταδίδει βρίσκεται ακόμη στην χρονική περίοδο που διαρκεί το T_{DATA}
- **sendRRTS ( ):** Δημιουργεί το packet header του RTS' πακέτου.
- **check_pktRRTS ( ):** Η συνάρτηση αυτή καλείται για να σταλεί το RTS' πακέτο
- **recvRRTS ( ):** Η συνάρτηση που καλείται όταν το πακέτο που έχει ληφθεί είναι RTS'

Προστέθηκαν ακόμη κάποιες μεταβλητές, απαραίτητες για τη σωστή λειτουργία του πρωτοκόλλου. Οι σημαντικότερες είναι οι εξής:

- **rx_master_ :** κάθε κόμβος ενημερώνει αυτή τη μεταβλητή όταν υπάρχει κάποιος master recipient στη γειτονιά του.
- **tx_master_:** κάθε κόμβος ενημερώνει αυτή τη μεταβλητή όταν υπάρχει κάποιος master sender στη γειτονιά του.
- **tx_master_size_:** αποθηκεύετε το packet size του master sender, ώστε να γίνεται ο έλεγχος αν το μέγεθος του πακέτου για μια παράλληλη μετάδοση είναι μικρότερο από αυτό του master.

Κάποιες από τις συναρτήσεις που χρησιμοποιεί το 802.11 έγιναν overwrite για να τροποποιηθούν ώστε να μπορούν να χρησιμοποιηθούν για το MACA-P. Οι συναρτήσεις αυτές είναι οι ακόλουθες:

- send_timer ( )
- check_pktCTRL ( )
- check_pktRTS ( )

- check_pktTx ( )
- sendRTS ( )
- sendCTS ( )
- send ( )
- recv ( )
- recv_timer ( )
- recvRTS ( )
- recvCTS ( )
- recvDATA ( )

Η υλοποίηση του κώδικα του MACA-P έγινε σε γλώσσα προγραμματισμού C++ και στηρίχθηκε στον κώδικα υλοποίησης του 802.11 που υπάρχει στο πρόγραμμα προσομοίωσης ns2.28. Ο κώδικας των αρχείων Maca_p.cc και Maca_p.h βρίσκεται στο παράρτημα Α. Έχουν τροποποιηθεί επίσης κάποια αρχεία του ns για να οριστούν κάποιες σταθερές. Τα αρχεία αυτά είναι το ns-default.tcl και το ns-packet.tcl και βρίσκονται στο tcl/lib folder του ns2.28. Οι προσθήκες που έγιναν σε αυτά τα αρχεία παρατίθενται στο ΠΑΡΑΡΤΗΜΑ Α.

Για την προσομοίωση χρησιμοποιήθηκε το πρόγραμμα προσομοίωσης ns2.28 και τα σενάρια γράφτηκαν σε TCL κώδικα. Ο κώδικας των σεναρίων βρίσκεται στο ΠΑΡΑΡΤΗΜΑ Β.

4. Προσομοίωση – Ρυθμαπόδοσης πρωτοκόλλων

Ακολούθως έγινε προσομοίωση των δύο πρωτοκόλλων, του 802.11 και του MACA-P, ώστε να γίνει σύγκριση στη ρυθμαπόδοση τους. Η προσομοίωση έγινε για διαφορετικές τοπολογίες και διαφορετική κυκλοφορία δεδομένων. Παρακάτω θα αναπτυχθούν τα διάφορα σενάρια προσομοίωσης και θα γίνει σύγκριση και σχολιασμός των αποτελεσμάτων.

Σε όλα τα παρακάτω σενάρια οι κόμβοι επικοινωνούσαν ανά ζεύγη. Οι κόμβοι τοποθετήθηκαν σε αποστάσεις 200m από τους γειτονικούς τους και τα όρια επικοινωνίας των κόμβων ήταν 250m. Αυτό σημαίνει ότι οι κόμβοι ήταν στα όρια επικοινωνίας μόνο των γειτονικών τους, και μπορούσαν να ακούσουν μόνο αυτούς.

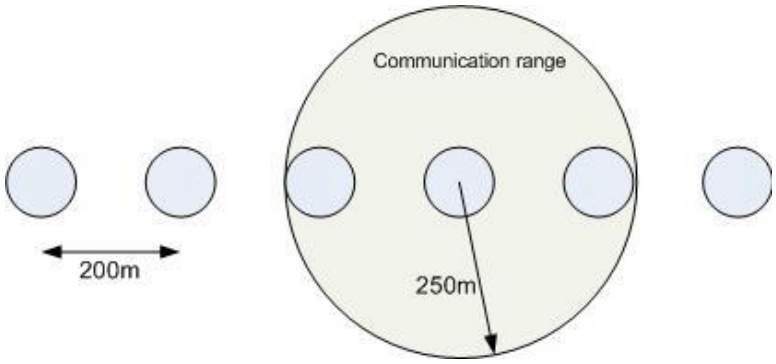
Στα σενάρια προσομοίωσης χρησιμοποιήθηκαν πηγές κίνησης CBR ή PARETO.

Ως πρωτόκολλο μεταφοράς χρησιμοποιήθηκε το UDP. Η επιλογή του UDP έναντι του TCP έγινε με σκοπό η κίνηση που παράγουν οι πηγές να μεταφέρεται στο δίκτυο ως έχει. Στην περίπτωση του TCP ο έλεγχος συμφόρησης του πρωτοκόλλου θα αλλοίωνε τη μορφή της κίνησης που θα κατέληγε στο δίκτυο.

4.1 Σενάριο 1

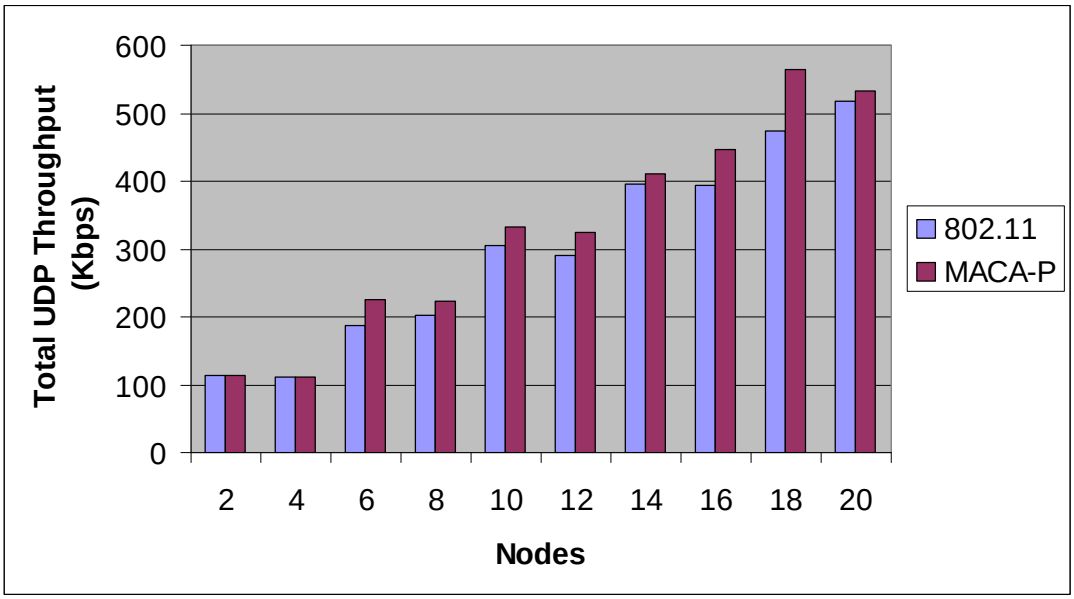
Στο πρώτο σενάριο της προσομοίωσης οι κόμβοι τοποθετήθηκαν γραμμικά σε μια ευθεία. Οι αποστάσεις των κόμβων είναι σε απόσταση 200m όπως αναφέρθηκε και παραπάνω. Η τοπολογία φαίνεται στο σχήμα 4.1.

Για την παραγωγή δεδομένων χρησιμοποιήθηκε CBR πηγή. Ο ρυθμός παραγωγής των πακέτων ήταν 2Mbps για κάθε σύνδεση. Το σενάριο αυτό προσομοιώθηκε για 2, 4, 6, 8, 10, 12, 14, 16, 18 και 20 κόμβους.



Σχήμα 4.1: Σενάριο 1 – Τοπολογία κόμβων

Τα αποτελέσματα της προσομοίωσης φαίνονται στο σχήμα 4.2.



Σχήμα 4.2: Σενάριο 1 – Υψηλή κίνηση – Πηγή CBR

Σχολιασμός αποτελεσμάτων:

Όπως φαίνεται από τη γραφική παράσταση, το πρωτόκολλο MACA-P έχει πιο υψηλή συνολική ρυθμαπόδοση από το 802.11, εκτός από τις περιπτώσεις όπου ο αριθμός των κόμβων ήταν 2 και 4. Για αυτές τις περιπτώσεις είναι αναμενόμενο να συμβαίνει αυτό. Στην περίπτωση των 2 κόμβων το MACA-P λειτουργεί όπως και το 802.11, αφού υπάρχει μόνο ένας αποστολέας δεδομένων και έτσι δεν υπάρχει καμία παράλληλη μετάδοση. Στην περίπτωση των 4 κόμβων, υπάρχουν δύο αποστολείς και ο αριθμός των δυνατών

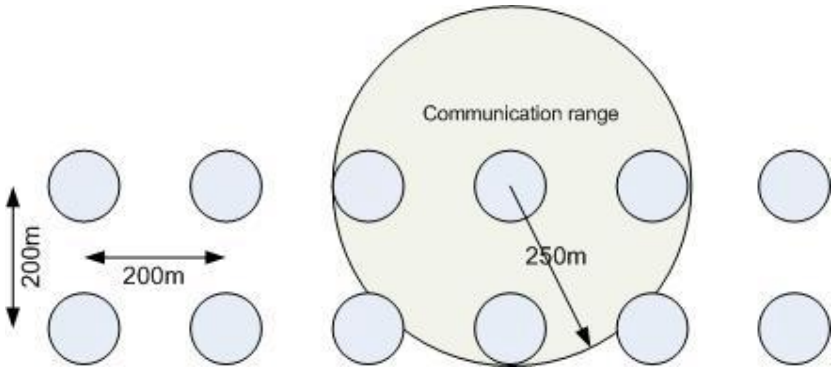
παραλλήλων μεταδόσεων είναι τόσο μικρός που δε δίνουν την αναμενόμενη αύξηση στη ρυθμαπόδοση.

Παρ' όλο που παρατηρείται αύξηση της ρυθμαπόδοσης του MACA-P για τις υπόλοιπες περιπτώσεις, η αύξηση αυτή δεν είναι μεγάλη. Αυτό συμβαίνει γιατί οι κόμβοι έχουν ως γείτονες μόνο δύο άλλους κόμβους. Έτσι η πιθανότητα μιας μετάδοσης να γίνει παράλληλα με κάποια άλλη είναι σχετικά περιορισμένη, αφού ο κάθε κόμβος πριν μεταδώσει κάνει backoff, και για να γίνει παράλληλη μετάδοση πρέπει δύο γειτονικοί να επιλέξουν τέτοια slot time ώστε η λήξη του backoff timer του ενός κόμβου να συμπέσει με το κενό ελέγχου του κόμβου που κέρδισε πρόσβαση στο μέσο, για να είναι εφικτή η παράλληλη μετάδοση.

4.2 Σενάριο 2

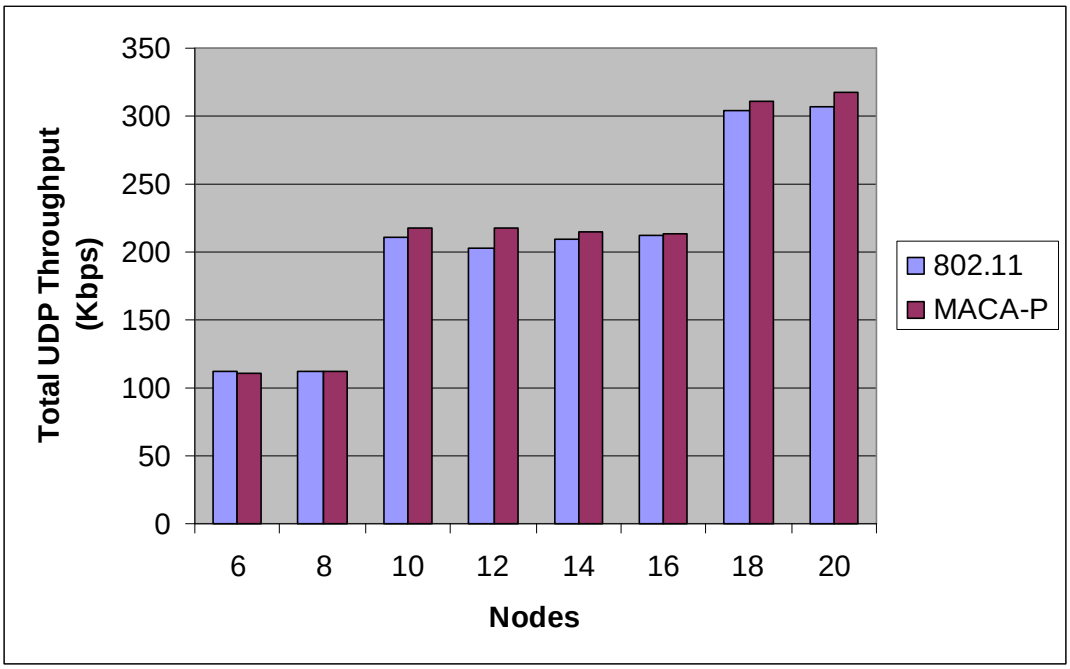
Στο δεύτερο σενάριο της προσομοίωσης οι κόμβοι τοποθετήθηκαν πάλι γραμμικά, αλλά σε δύο ευθείες, με την απόσταση μεταξύ των δύο ευθειών και μεταξύ των γειτονικών κόμβων να είναι ίση με 200m. Η τοπολογία φαίνεται στο σχήμα 4.3.

Για την κυκλοφορία δεδομένων χρησιμοποιήθηκε CBR πηγή. Ο ρυθμός παραγωγής των πακέτων ήταν 2Mbps για κάθε σύνδεση. Το σενάριο αυτό προσομοιώθηκε για 6, 8, 10, 12, 14, 16, 18 και 20 κόμβους.



Σχήμα 4.3: Σενάριο 2 – Τοπολογία κόμβων

Τα αποτελέσματα της προσομοίωσης φαίνονται στο σχήμα 4.4.



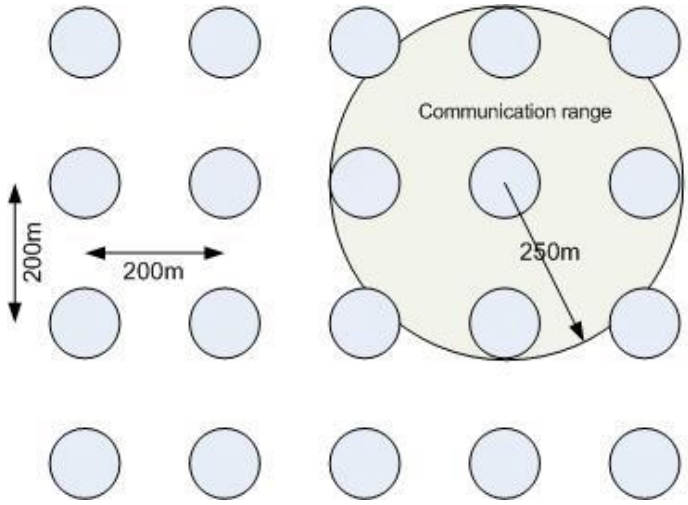
Σχήμα 4.4: Σενάριο 2 – Υψηλή κίνηση – Πηγή CBR

Σχολιασμός αποτελεσμάτων:

Και σε αυτό το σενάριο, το MACA-P έχει μεγαλύτερη ρυθμαπόδοση από το αντίστοιχο 802.11, αλλά και πάλι η αύξηση είναι σχετικά μικρή. Ο λόγος είναι παρόμοιος με αυτόν του προηγούμενου σεναρίου. Οι κόμβοι επιτυγχάνουν να κάνουν κάποιες παράλληλες μεταδόσεις αυξάνοντας έτσι τη ρυθμαπόδοση του πρωτοκόλλου. Σε αυτό το σενάριο οι κόμβοι έχουν 3 γειτονικούς κόμβους σε αντίθεση με τους δύο του προηγούμενου σεναρίου, αλλά και πάλι η μετάδοση των πακέτων δεν είναι τέτοια ώστε το MACA-P επιτύχει τέτοιο αριθμό παραλλήλων μεταδόσεων που θα αύξαναν αισθητά την απόδοση του πρωτοκόλλου.

4.3 Σενάριο 3

Το τρίτο σενάριο της προσομοίωσης παρουσιάζει και το μεγαλύτερο ενδιαφέρον. Στο σενάριο αυτό, οι κόμβοι είναι τοποθετημένοι σε πλέγμα, με τις ίδιες αποστάσεις όπως και στις προηγούμενες περιπτώσεις. Το σενάριο αυτό προσομοιώθηκε για 12, 16 και 20 κόμβους και η τοπολογία των κόμβων φαίνεται στο σχήμα 4.5.



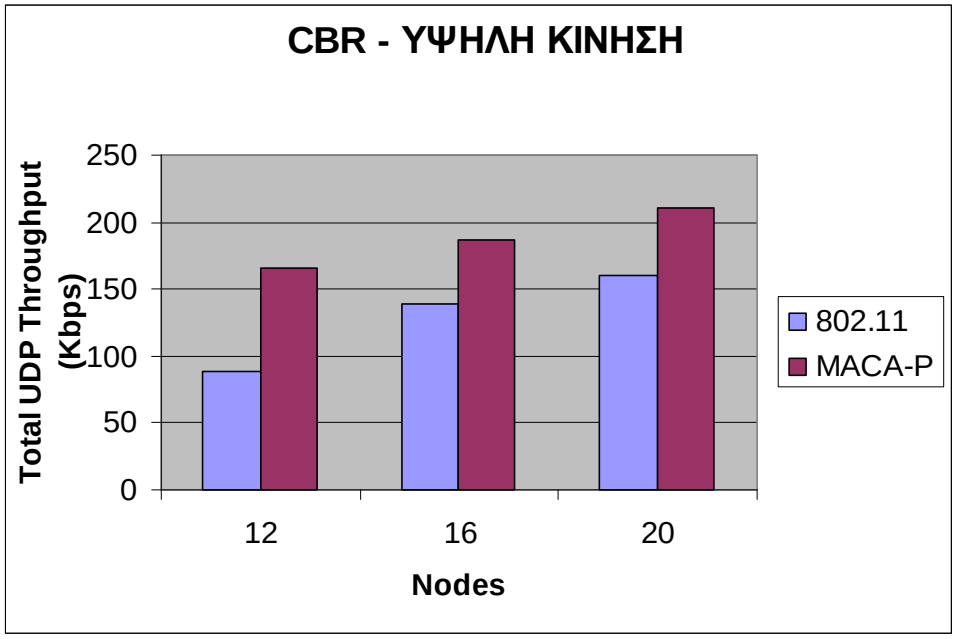
Σχήμα 4.5: Σενάριο 3 – Τοπολογία κόμβων

4.3.1 Υψηλή Κίνηση

4.3.1.1 Πηγή CBR

Στην πρώτη προσομοίωση χρησιμοποιήθηκε πηγή CBR με ρυθμό παραγωγής πακέτων 2Mbps για κάθε σύνδεση.

Τα αποτελέσματα αυτά φαίνονται στη γραφική παράσταση του σχήματος 4.6.



Σχήμα 4.6: Σενάριο 3 – Υψηλή κίνηση – Πηγή CBR

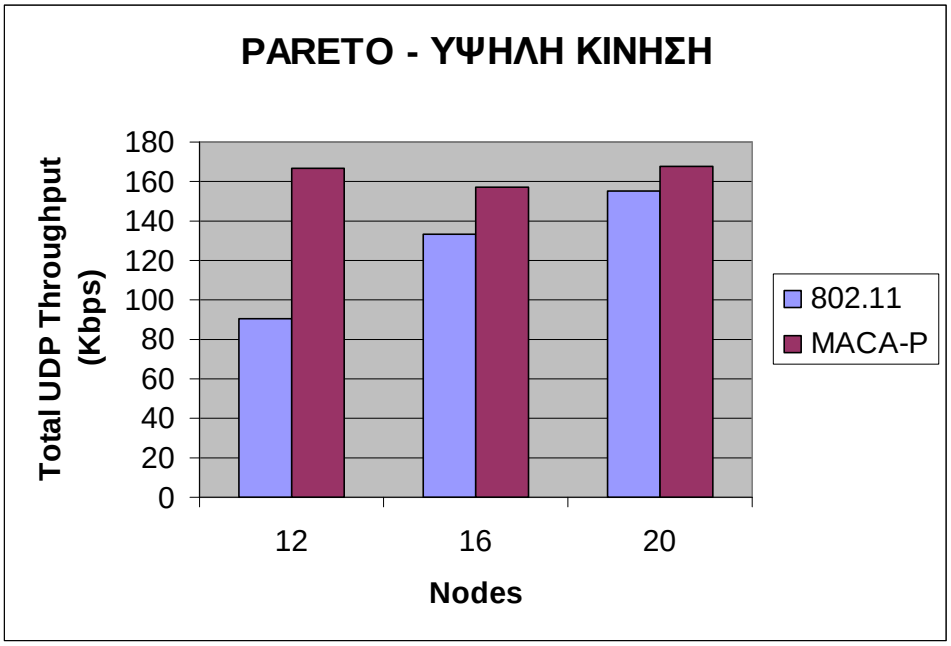
Σχολιασμός αποτελεσμάτων:

Στην περίπτωση αυτή βλέπουμε ότι η απόδοση του MACA-P έχει βελτιωθεί πάρα πολύ σε σχέση με το 802.11. Ειδικά στην περίπτωση των 12 κόμβων, η απόδοση σχεδόν έχει διπλασιαστεί. Σε αυτό το σενάριο προσομοίωσης οι κόμβοι έχουν περισσότερους γειτονικούς κόμβους, ο καθένας έχει 4 γείτονες. Αυτό δίνει τη δυνατότητα για περισσότερες παράλληλες μεταδόσεις στο πρωτόκολλο MACA-P. Ο λόγος είναι ο εξής: οι κόμβοι που προσπαθούν να μεταδώσουν έχουν άλλους τέσσερις γειτονικούς κόμβους που κάνουν και αυτοί προσπάθεια για μετάδοση. Μόλις κάποιος από αυτούς καταλάβει το μέσο, το 802.11 δεν επιτρέπει στους υπόλοιπους τέσσερις να μεταδώσουν. Αντίθετα, στο MACA-P αν κάποιος από αυτούς τους τέσσερις τηρεί τις προϋποθέσεις, που αναλύθηκαν σε προηγούμενο κεφάλαιο, τότε επιτρέπει να γίνει αυτή η μετάδοση. Αυξάνεται δηλαδή η πιθανότητα ένας κόμβος να τελειώσει το backoff του τη στιγμή που κάποιος γείτονας του μεταδίδει και εκείνη την στιγμή βρίσκεται στην περίοδο του control gap.

4.3.1.2 Πηγή PARETO

Μετά από αυτό, προσομοιώσαμε το ίδιο σενάριο με τον ίδιο ρυθμό παραγωγής πακέτων, χρησιμοποιώντας πηγή ON/OFF που ακολουθούσε την κατανομή PARETO. Οι πηγή αυτή έχει δύο περιόδους, την ON όπου γίνεται παραγωγή πακέτων, και την OFF όπου σταματάει η παραγωγή. Η μέση διάρκεια των διαστημάτων αυτών έχει οριστεί ίση με 100ms τόσο για την ON όσο και για την OFF περίοδο.

Η γραφική παράσταση του σχήματος 4.7 δείχνει τα αποτελέσματα αυτής της κίνησης.



Σχήμα 4.7: Σενάριο 3 – Υψηλή κίνηση – Πηγή PARETO

Σχολιασμός αποτελεσμάτων:

Και εδώ τα αποτελέσματα είναι παρόμοια με τα πιο πάνω αποτελέσματα όπου είχε χρησιμοποιηθεί CBR πηγή. Το MACA-P διατηρεί την δυνατότητα της παράλληλης μετάδοσης του και για τυχαία πηγή παραγωγής πακέτων, όπως η PARETO. Τέτοιου τύπου πηγές προσεγγίζουν περισσότερο τις πραγματικές συνθήκες, όπου η κίνηση δεν είναι σταθερή αλλά υπάρχουν χρονικές περίοδοι μέγιστης φόρτωσης του δικτύου και αδρανείς περίοδοι.

Ο ρυθμός παραγωγής πακέτων που χρησιμοποιήθηκε σε αυτά τα σενάρια ήταν πάρα πολύ υψηλός που δημιουργούσε υπερφόρτωση δικτύου με αποτέλεσμα να απορρίπτονταν πολλά πακέτα. Γι' αυτό το λόγο στη συνέχεια προσομοιώθηκε η ίδια τοπολογία-πλέγμα για μέτρια και χαμηλή κίνηση.

Σε ένα δίκτυο 802.11 ορίζεται ένας ρυθμός μετάδοσης, (π.χ. στο 802.11b 11Mbps, στο ns 1Mbps) ο οποίος καθορίζει το ρυθμό που μεταδίδεται ένα πακέτο όταν υπάρχει μετάδοση. Ωστόσο λόγω της χρήσης του κοινού καναλιού από πολλούς κόμβους και της CSMA/CA πρόσβασης στο κοινό αυτό μέσο, ο εφικτός ρυθμός μετάδοσης ακόμα και με ένα μόνο αποστολέα δεδομένων είναι σημαντικά κατώτερος από την τιμή του ρυθμού μετάδοσης ενός πακέτου. Επιπλέον αυτός ο εφικτός ρυθμός μετάδοσης δεν είναι εύκολο να προβλεφθεί εκ των προτέρων.

Στα προηγούμενα σενάρια προσομοίωσης ο ρυθμός ήταν μεγαλύτερος από τη μέγιστη χωρητικότητα του καναλιού. Από τα αποτελέσματα, θεωρήσαμε ότι ο εφικτός ρυθμός μετάδοσης είναι η μέγιστη ρυθμαπόδοση που πήραμε στις μετρήσεις μας.

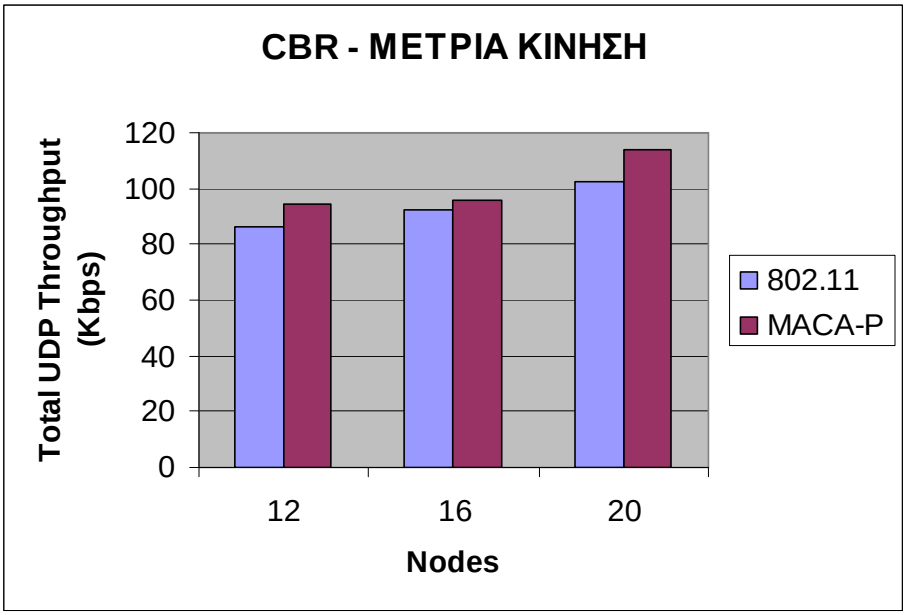
Σε αυτό στηριχθήκαμε για να υπολογίσουμε τις τιμές για τους ρυθμούς παραγωγής πακέτων της μέτριας και της χαμηλής κίνησης.

Για την προσομοίωση μας, η μέτρια κίνηση ορίστηκε στο 70% του εφικτού ρυθμού μετάδοσης, και υπολογίστηκε ίση με 24Kbps και η χαμηλή κίνηση στο 30% που είναι ίση με 11Kbps.

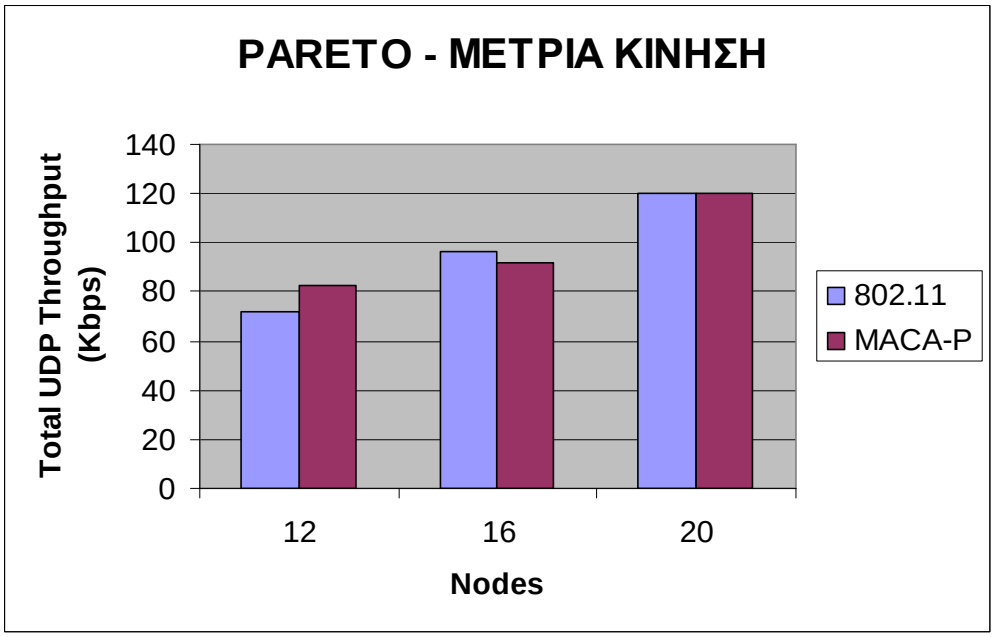
4.3.2 Μέτρια Κίνηση

Χρησιμοποιήθηκαν και πάλι οι δύο πηγές, CBR και PARETO, για τον ίδιο αριθμό κόμβων. Ο ρυθμός παραγωγής πακέτων είναι 24Kbps, όπως αναφέρθηκε παραπάνω.

Τα αποτελέσματα αυτά φαίνονται στα σχήματα 4.8 και 4.9 που ακολουθούν:



Σχήμα 4.8: Σενάριο 3 – Μέτρια κίνηση – Πηγή CBR



Σχήμα 4.9: Σενάριο 3 – Μέτρια κίνηση – Πηγή PARETO

Σχολιασμός αποτελεσμάτων:

Μελετώντας τις γραφικές παραστάσεις, τα αποτελέσματα που προκύπτουν έχουν αρκετό ενδιαφέρον και αξίζει να γίνει ένας επιπλέον σχολιασμός.

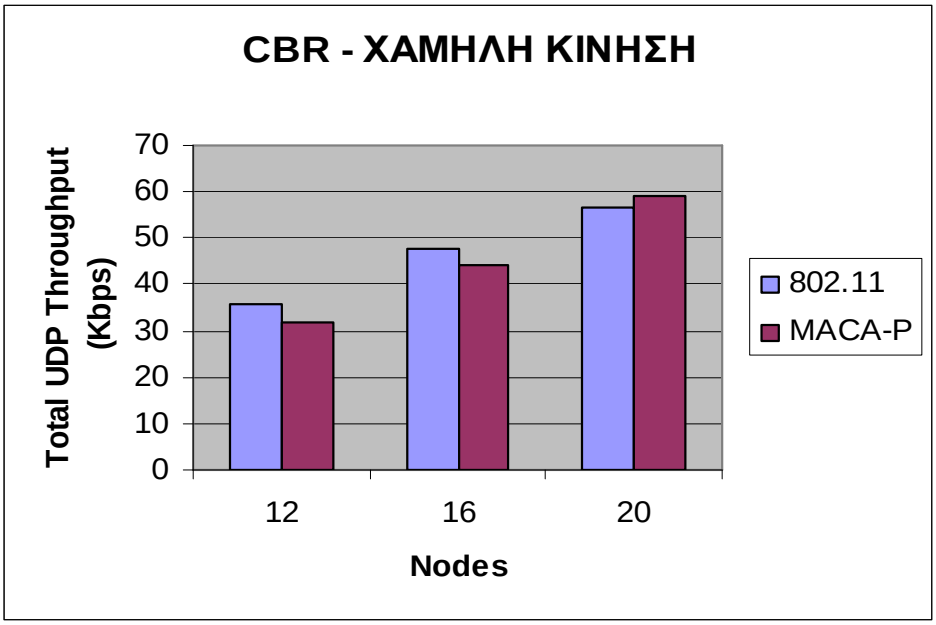
Ο ρυθμός παραγωγής των πακέτων είναι πολύ πιο μικρός από τις προηγούμενες περιπτώσεις. Έτσι ο αριθμός των παραλλήλων μεταδόσεων του MACA-P μειώνεται αρκετά, αφού μειώνεται πάρα πολύ η πιθανότητα κάποια παράλληλη μετάδοση να τηρεί τις προϋποθέσεις και να μπορεί να γίνει. Το αξιοσημείωτο είναι ότι σε κάποιες περιπτώσεις το 802.11 είναι πιο αποδοτικό από το MACA-P. Στις περιπτώσεις αυτές, αυτό που συμβαίνει είναι ότι οι παράλληλες μεταδόσεις που πραγματοποιήθηκαν ήταν ελάχιστες. Στην υλοποίηση όμως του MACA-P έχει προστεθεί ένα κενό ελέγχου, το οποίο αυξάνει τη διάρκεια της μετάδοσης του πακέτου σε σχέση με το 802.11. Υπενθυμίζουμε ότι στο 802.11 η μεταφορά των δεδομένων ξεκινούσε αμέσως μετά την ανταλλαγή των πακέτων ελέγχου (RTS/CTS). Αυτό έχει ως αποτέλεσμα το 802.11 να είναι πιο αποδοτικό, αφού η κάθε μετάδοση που γίνεται σε αυτό κρατάει λιγότερο χρόνο το κανάλι κατειλημμένο, ενώ ταυτόχρονα στο MACA-P δε γίνεται κάποια παράλληλη μετάδοση.

4.3.3 Χαμηλή Κίνηση

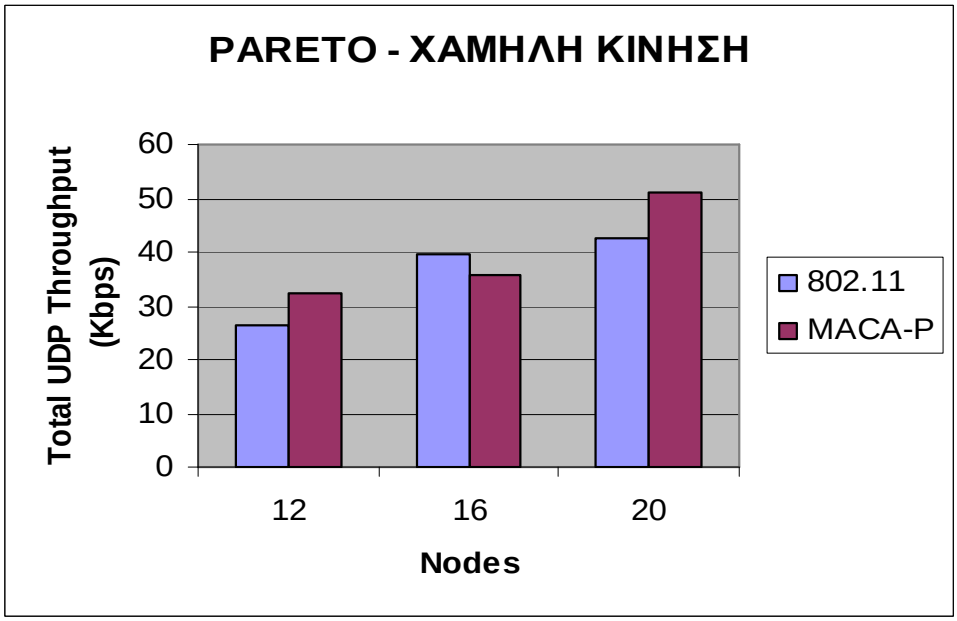
Το ίδιο σενάριο ίσχυσε και στην περίπτωση της χαμηλής κίνησης. Ο ρυθμός που χρησιμοποιήθηκε εδώ ήταν 11Kbps όπως προαναφέρθηκε.

Χρησιμοποιήθηκαν και πάλι οι δύο πηγές, CBR και PARETO, για τον ίδιο αριθμό κόμβων.

Τα αποτελέσματα της προσομοίωσης αυτής φαίνονται στις γραφικές παραστάσεις των σχημάτων 4.10 και 4.11 που ακολουθούν.



Σχήμα 4.10: Σενάριο 3 – Χαμηλή κίνηση – Πηγή CBR



Σχήμα 4.11: Σενάριο 3 – Χαμηλή κίνηση – Πηγή PARETO

Σχολιασμός αποτελεσμάτων:

Τα αποτελέσματα έχουν μεγάλη ομοιότητα με τα αποτελέσματα της μέτριας κίνησης. Για τον ίδιο ακριβώς λόγο, οι παράλληλες μεταδόσεις είναι πολύ λίγες με αποτέλεσμα το MACA-P να μην είναι πολύ πιο αποδοτικό από το 802.11, ενώ σε κάποιες περιπτώσεις να συμβαίνει το αντίθετο, δηλαδή το 802.11 να είναι πιο αποδοτικό.

4.4 Συμπεράσματα

Το πρωτόκολλο MACA-P όπως φαίνεται και από τα αποτελέσματα των σεναρίων προσομοίωσης μπορεί να υποστηρίξει παράλληλες μεταδόσεις σε συγκεκριμένες περιπτώσεις, γεγονός που μπορεί όντως να αυξήσει την απόδοση του δικτύου.

Όπως φαίνεται από τα σενάρια μας, η αποδοτικότητα του MACA-P σε σχέση με το 802.11 αυξάνεται όταν η κίνηση του δικτύου είναι μεγάλη. Το τρίτο σενάριο, το οποίο προσομοιώθηκε για διαφορετικούς ρυθμούς κίνησης δείχνει πόσο μπορεί να επηρεάσει αυτό τη γενικότερη αύξηση της ρυθμαπόδοσης.

Άλλος ένας σημαντικός παράγοντας που επηρεάζει αισθητά την απόδοση του πρωτοκόλλου, είναι ο αριθμός των γειτονικών κόμβων. Είναι φυσικό η αύξηση των γειτόνων να αυξάνει την πιθανότητα ένας κόμβος να μπορεί να συγχρονίσει τη μετάδοση του με κάποιαν άλλη μετάδοση που έχει αρχίσει.

Οι όποιες μη αναμενόμενες αυξομειώσεις στη ρυθμαπόδοση για διαφορετικούς αριθμούς κόμβων σε ίδια σενάρια, οφείλεται στο γεγονός ότι οι συνδέσεις των κόμβων ανά ζεύγη δεν ήταν απολύτως συμμετρικές. Αυτό είχε ως αποτέλεσμα κάποιες συνδέσεις να μην ήταν οι ιδανικές για το MACA-P, αφού το πρωτόκολλο αυτό δεν επιτρέπει σε όλες τις περιπτώσεις παράλληλες μεταδόσεις, παρά μόνο για κάποιες συγκεκριμένες. Αυτό όμως δεν επηρεάζει τα συμπεράσματα μας για την απόδοση του MACA-P. Πέρα απ΄αυτό θέλαμε να μελετήσουμε γενικότερα το πρωτόκολλο, ειδικά σε τυχαία σενάρια. Σκοπός μας δεν ήταν να βρούμε τα ιδανικά σενάρια που επιτρέπουν παράλληλες μεταδόσεις και να κάνουμε μετρήσεις οι οποίες στην ουσία να μην ήταν αντιπροσωπευτικές για την πραγματική απόδοση του MACA-P. Τα τυχαία αυτά σενάρια προσεγγίζουν περισσότερο την πραγματική κίνηση του δικτύου και τα αποτελέσματα μας είναι αρκετά ρεαλιστικά.

Αυτό που μπορούμε να πούμε με βεβαιότητα είναι ότι η ιδέα υλοποίησης του πρωτοκόλλου αυτού για την επίλυση του προβλήματος που υπάρχει στην

παράλληλη μετάδοση πακέτων στα ad hoc δίκτυα, ξεφεύγει από τις προηγούμενες απόπειρες. Προτείνει μια συγκεκριμένη λύση, η οποία είναι εφαρμόσιμη και σχετικά αποδοτική.

ΠΑΡΑΡΤΗΜΑ Α – ΚΩΔΙΚΑΣ MACA-P

Στο παράρτημα αυτό παρατίθενται οι πλήρεις κώδικες των αρχείων Maca_p.cc και Maca_p.h που υλοποιούν το πρωτόκολλο. Επίσης παρατίθενται οι αλλαγές που έγιναν στα αρχεία ns-default.tcl και ns-packet.tcl ώστε να οριστούν κάποιες απαραίτητες μεταβλητές για το πρόγραμμα προσομοίωσης ns2.28.

Maca_p.cc

```
#include "Maca_p.h"
#include "tfrc.h"

/*****************************************************************************/

 #ifdef USE_SLOT_TIME
 #error "Incorrect slot time implementation - don't use USE_SLOT_TIME..."
 #endif

#define ROUND_TIME()      \
        {                                                               \
                assert(slottime);                                       \
                double rmd = remainder(s.clock() + rtime, slottime);          \
                if(rmd > 0.0)                                           \
                        rtime += (slottime - rmd);                      \
                else                                                    \
                        rtime += (-rmd);                                \
        }


/*==================MacapTimers============================== */

void
MacapTimer::start(double time)
{
        Scheduler &s = Scheduler::instance();
        assert(busy_ == 0);

        busy_ = 1;
        paused_ = 0;
        stime = s.clock();
        rtime = time;
        assert(rtime >= 0.0);


        s.schedule(this, &intr, rtime);
}
```

```
void
MacapTimer::stop(void)
{
        Scheduler &s = Scheduler::instance();

        assert(busy_);

        if(paused_ == 0)
                s.cancel(&intr);

        busy_ = 0;
        paused_ = 0;
        stime = 0.0;
        rtime = 0.0;
}


/*==================NAV_DATA_Timer============================*/

void
NavDataTimer::handle(Event *)
{
        busy_ = 0;
        paused_ = 0;
        stime = 0.0;
        rtime = 0.0;

        macap->navDataHandler();
}

/*************************************************************************/

PHY_MIB2::PHY_MIB2(Maca_p* parent)
{
        /*
         * Bind the phy mib objects.  Note that these will be bound
         * to Mac/802_11 variables
         */

        parent->bind("CWMin_", &CWMin);
        parent->bind("CWMax_", &CWMax);
        parent->bind("SlotTime_", &SlotTime);
        parent->bind("SIFS_", &SIFSTime);
        parent->bind("PreambleLength_", &PreambleLength);
        parent->bind("PLCPHeaderLength_", &PLCPHeaderLength);
        parent->bind_bw("PLCPDataRate_", &PLCPDataRate);
}
```

```
/*****************************************************************************/

MAC_MIB2::MAC_MIB2(Maca_p *parent)
{
        /*
         * Bind the phy mib objects.  Note that these will be bound
         * to Mac/802_11 variables
         */

        parent->bind("RTSThreshold_", &RTSThreshold);
        parent->bind("ShortRetryLimit_", &ShortRetryLimit);
        parent->bind("LongRetryLimit_", &LongRetryLimit);
}

/*******************MACA-P constructor ********************************/

Maca_p::Maca_p() :
        Mac802_11(), phymib_(this), macmib_(this), mhNavData_(this)
{
        navData_ = 0.0;
        tx_master_=0;
        rx_master_=0;
        tx_master_size_ = 0.0;
        rx_master_duration_ack = 0;
        tx_state_ = rx_state_ = MAC_IDLE;
        tx_active_ = 0;
        eotPacket_ = NULL;
        pktRTS_ = 0;
        pktRRTS_= 0;
        pktCTRL_ = 0;
        cw_ = phymib_.getCWMin();
        ssrc_ = slrc_ = 0;
      et_ = new EventTrace();

        sta_seqno_ = 1;
        cache_ = 0;
        cache_node_count_ = 0;

        // chk if basic/data rates are set
        // otherwise use bandwidth_ as default;

        Tcl& tcl = Tcl::instance();
        tcl.evalf("Mac/Maca_p set basicRate_");
        if (strcmp(tcl.result(), "0") != 0)
                bind_bw("basicRate_", &basicRate_);
        else
                basicRate_ = bandwidth_;

        tcl.evalf("Mac/Maca_p set dataRate_");
```

```
if (strcmp(tcl.result(), "0") != 0)
                bind_bw("dataRate_", &dataRate_);
        else
                dataRate_ = bandwidth_;

      EOTtarget_ = 0;
        bss_id_ = IBSS_ID;

}

/* ================= Timer Handler Routines====================== */

void
Maca_p::capture(Packet *p)
{
        /*
         * Update the NAVDATA so that this does not screw
         * up carrier sense.
         */
        set_nav(usec(phymib_.getEIFS() + txtime(p)));
        set_navData(usec(phymib_.getEIFS() + txtime(p)));
        Packet::free(p);
}

inline int
Maca_p::is_idleMacap()
{
        if(navData_ > Scheduler::instance().clock())
                return 0;

        return 1;
}


void
Maca_p::navDataHandler()
{

}


void
Maca_p::tx_resume()
{
        double rTime;
        assert(mhSend_.busy() == 0);
        assert(mhDefer_.busy() == 0);

        if(pktCTRL_) {
```

```
                /*
                 *  Need to send a CTS or ACK.
                 */
                mhDefer_.start(phymib_.getSIFS());
        } else if(pktRTS_) {
                if (mhBackoff_.busy() == 0) {
                        rTime = (Random::random() % cw_) * phymib_.getSlotTime();
                        mhDefer_.start( phymib_.getDIFS() + rTime);
                }
        } else if(pktTx_) {
                if (mhBackoff_.busy() == 0) {
                        hdr_cmn *ch = HDR_CMN(pktTx_);
                        struct hdr_mac802_11 *mh = HDR_MAC802_11(pktTx_);

                        if ((u_int32_t) ch->size() < macmib_.getRTSThreshold()
                           || (u_int32_t) ETHER_ADDR(mh->dh_ra) ==
MAC_BROADCAST) {
                                rTime = (Random::random() % cw_)
                                        * phymib_.getSlotTime();
                                mhDefer_.start(phymib_.getDIFS() + rTime);
                        } else {
                                mhDefer_.start(phymib_.getSIFS());
                        }
                        }
        } else if(callback_) {
                Handler *h = callback_;
                callback_ = 0;
                h->handle((Event*) 0);
        }
        setTxState(MAC_IDLE);
}

void
Maca_p::rx_resume()
{
        assert(pktRx_ == 0);
        assert(mhRecv_.busy() == 0);

        setRxState(MAC_IDLE);
}

/* ============The "real" Timer Handler Routines====================== */

void
Maca_p::send_timer()
{
        struct hdr_mac802_11* dh;
        struct hdr_tfrc *macaprh;
        Handler *h = callback_;
        double rTime;
```

```
        switch(tx_state_) {
        /*
         * Sent a RTS, but did not receive a CTS.
         */
        case MAC_RTS:
        macaprh = hdr_tfrc::access(pktRTS_);
        dh = HDR_MAC802_11(pktTx_);
                if (macaprh->UrgentFlag<2)
                {
                sendRRTS(ETHER_ADDR(dh->dh_ra),0,0);
                check_pktRRTS();
        }
        RetransmitRTS();
                break;

        /*
         * Sent a CTS, but did not receive a DATA packet.
         */
        case MAC_CTS:
                assert(pktCTRL_);
                Packet::free(pktCTRL_);
                pktCTRL_ = 0;
                break;
        /*
         * Sent DATA, but did not receive an ACK packet.
         */
        case MAC_SEND:
                RetransmitDATA();
                break;
        /*
         * Sent an ACK, and now ready to resume transmission.
         */
        case MAC_ACK:
                assert(pktCTRL_);
                if (tx_master_ > 0) {tx_master_--;}
                else if (rx_master_ > 0) {rx_master_ --;}
                Packet::free(pktCTRL_);
                pktCTRL_ = 0;
                break;
        case MAC_IDLE:
                break;
        default:
                assert(0);
        }
        tx_resume();
}
```

```
/* ========== Outgoing Packet Routines============================= */

int
Maca_p::check_pktCTRL()
{
        struct hdr_mac802_11 *mh;
        double timeout;

        if(pktCTRL_ == 0)
                return -1;
        if(tx_state_ == MAC_CTS || tx_state_ == MAC_ACK)
                return -1;

        mh = HDR_MAC802_11(pktCTRL_);

        struct hdr_cmn *ch = HDR_CMN(pktCTRL_);

        u_int32_t dst = ETHER_ADDR(mh->dh_ra);
        u_int32_t src = ETHER_ADDR(mh->dh_ta);

        switch(mh->dh_fc.fc_subtype) {
        /*
         *  If the medium is not IDLE, don't send the CTS.
         */
        case MAC_Subtype_CTS:
             if (  ( (! is_idle()) && rx_master_>2 ) || ( (!is_idle()) && (is_idleMacap())) ){
                        discard(pktCTRL_, DROP_MAC_BUSY); pktCTRL_ = 0;
                        return 0;
                }
                setTxState(MAC_CTS);
                /*
                 * timeout:  cts + data tx time calculated by
                 *          adding cts tx time to the cts duration
                 *          minus ack tx time -- this timeout is
                 *          a guess since it is unspecified
                 *          (note: mh->dh_duration == cf->cf_duration)
                 */
                 timeout = txtime(phymib_.getCTSlen(), basicRate_)
                + DSSS_MaxPropagationDelay                  // XXX
                + sec(mh->dh_duration)
                + DSSS_MaxPropagationDelay                  // XXX
               - phymib_.getSIFS()
               - txtime(phymib_.getACKlen(), basicRate_);
                break;
                /*
                 * IEEE 802.11 specs, section 9.2.8
                 * Acknowledments are sent after an SIFS, without regard to
                 * the busy/idle state of the medium.
                 */
```

```
        case MAC_Subtype_ACK:
                setTxState(MAC_ACK);
                timeout = txtime(phymib_.getACKlen(), basicRate_);
                break;
        default:
                fprintf(stderr, "check_pktCTRL:Invalid MAC Control subtype\n");
                exit(1);
        }
        transmit(pktCTRL_, timeout);
        return 0;
}


int
Maca_p::check_pktRRTS()
{
        struct hdr_mac802_11 *mh;
        double timeout;

        assert(mhBackoff_.busy() == 0);

        if(pktRRTS_ == 0)
                return -1;
        mh = HDR_MAC802_11(pktRRTS_);

        struct hdr_cmn *ch = HDR_CMN(pktRRTS_);

        u_int32_t dst = ETHER_ADDR(mh->dh_ra);
        u_int32_t src = ETHER_ADDR(mh->dh_ta);

        switch(mh->dh_fc.fc_subtype) {
        case MAC_Subtype_RTS:
                if(! is_idle()) {
                        inc_cw();
                        mhBackoff_.start(cw_, is_idle());
                        return 0;
                }
                setTxState(MAC_RTS);
                timeout = txtime(phymib_.getRTSlen(), basicRate_)
                        + DSSS_MaxPropagationDelay                  // XXX
                        + phymib_.getSIFS();
                break;
        default:
                fprintf(stderr, "check_pktRTS:Invalid MAC Control subtype\n");
                exit(1);
        }
        transmit(pktRRTS_, timeout);
        return 0;
}
```

```
int
Maca_p::check_pktRTS()
{
        struct hdr_mac802_11 *mh;
        double timeout;

        assert(mhBackoff_.busy() == 0);

        if(pktRTS_ == 0)
                return -1;
        mh = HDR_MAC802_11(pktRTS_);
        struct hdr_tfrc *macaprh = hdr_tfrc::access(pktRTS_);

        struct hdr_cmn *ch = HDR_CMN(pktRTS_);
        u_int32_t dst = ETHER_ADDR(mh->dh_ra);
        u_int32_t src = ETHER_ADDR(mh->dh_ta);

        switch(mh->dh_fc.fc_subtype) {
        case MAC_Subtype_RTS:
        if (macaprh->UrgentFlag<2)
                {
            if (  ( (! is_idle()) && tx_master_>2 ) || ( (!is_idle()) && (is_idleMacap())) )  {
                        inc_cw();
                        mhBackoff_.start(cw_, is_idle());
                        return 0;
                }
                timeout = txtime(phymib_.getRTSlen(), basicRate_)
                        + DSSS_MaxPropagationDelay                      // XXX
                        + phymib_.getSIFS()
                        + txtime(phymib_.getCTSlen(), basicRate_)
                        + DSSS_MaxPropagationDelay;

if (tx_master_ == 2 ){
        if (    ( mhNavData_.expire() < timeout) || ( (macaprh->psize - mhNavData_.expire())
< (phymib_.getSIFS()
                        + txtime(phymib_.getCTSlen(), basicRate_))) )
                        {
                                inc_cw();
                                mhBackoff_.start(cw_, is_idle());
                                return 0;
                        }
                }
                setTxState(MAC_RTS);
                break;
        }

        default:
                fprintf(stderr, "check_pktRTS:Invalid MAC Control subtype\n");
                exit(1);
        }
```

```
        transmit(pktRTS_, timeout);
        return 0;
}


int
Maca_p::check_pktTx()
{
        struct hdr_mac802_11 *mh;
        double timeout;

        assert(mhBackoff_.busy() == 0);
        if(pktTx_ == 0)
                return -1;

        mh = HDR_MAC802_11(pktTx_);

        struct hdr_cmn *ch = HDR_CMN(pktTx_);

        u_int32_t dst = ETHER_ADDR(mh->dh_ra);
        u_int32_t src = ETHER_ADDR(mh->dh_ta);

        switch(mh->dh_fc.fc_subtype) {
        case MAC_Subtype_Data:
                if( (! is_idleMacap()) || tx_master_==3) {
                        sendRTS(ETHER_ADDR(mh->dh_ra),1,0);
                        inc_cw();
                        mhBackoff_.start(cw_, is_idle());
                        return 0;
                }
                setTxState(MAC_SEND);
                if((u_int32_t)ETHER_ADDR(mh->dh_ra) != MAC_BROADCAST){
                timeout = txtime(pktTx_)
                       + DSSS_MaxPropagationDelay              // XXX
                      + phymib_.getSIFS()
                      + txtime(phymib_.getACKlen(), basicRate_)
                      + DSSS_MaxPropagationDelay;             // XXX
                }
                else {
                        timeout = txtime(pktTx_);
                }
                break;
        default:
                fprintf(stderr, "check_pktTx:Invalid MAC Control subtype\n");
                exit(1);
        }
        transmit(pktTx_, timeout);
        return 0;
}
```

```
/*
 * Low-level transmit functions that actually place the packet onto
 * the channel.
 */


void
Maca_p::sendRRTS(int dst, int inflexible_bit, u_int16_t cts_duration)
{
        Packet *p = Packet::alloc();
        hdr_cmn* ch = HDR_CMN(p);
        struct rts_frame *rf = (struct rts_frame*)p->access(hdr_mac::offset_);
        struct hdr_tfrc *macaprrh = hdr_tfrc::access(p);

        assert(pktTx_);

        /*
         *  If the size of the packet is larger than the
         *  RTSThreshold, then perform the RTS/CTS exchange.
         */
        if( (u_int32_t) HDR_CMN(pktTx_)->size() < macmib_.getRTSThreshold() ||
        (u_int32_t) dst == MAC_BROADCAST) {
                Packet::free(p);

                return;
        }

        ch->uid() = 0;
        ch->ptype() = PT_MAC;
        ch->size() = phymib_.getRTSlen();
        ch->iface() = -2;
        ch->error() = 0;

        bzero(rf, MAC_HDR_LEN);

        rf->rf_fc.fc_protocol_version = MAC_ProtocolVersion;
        rf->rf_fc.fc_type       = MAC_Type_Control;
        rf->rf_fc.fc_subtype    = MAC_Subtype_RTS;
        rf->rf_fc.fc_to_ds      = 0;
        rf->rf_fc.fc_from_ds    = 0;
        rf->rf_fc.fc_more_frag          = 0;
        rf->rf_fc.fc_retry      = 0;
        rf->rf_fc.fc_pwr_mgt    = 0;
        rf->rf_fc.fc_more_data          = 0;
        rf->rf_fc.fc_wep        = 0;
        rf->rf_fc.fc_order      = 0;
```

```
        if (inflexible_bit == 0) {
                macaprrh->UrgentFlag = 2;
                macaprrh->round_id = 0;
                macaprrh->rate = 1.0;

                //rf->rf_duration = RTS_DURATION(pktTx_);
                STORE4BYTE(&dst, (rf->rf_ra));

                /* store rts tx time */
                ch->txtime() = txtime(ch->size(), basicRate_);

                STORE4BYTE(&index_, (rf->rf_ta));

                /* calculate rts duration field */
                macaprrh->psize = 0;

                macaprrh->seqno = 0;

                rf->rf_duration = 0;
        }
        else if (inflexible_bit == 1) {
                macaprrh->UrgentFlag = 3;
                //rf->rf_duration = RTS_DURATION(pktTx_);

                macaprrh->round_id = 0;
                macaprrh->rate = 1.0;

                STORE4BYTE(&dst, (rf->rf_ra));

                /* store rts tx time */
                ch->txtime() = txtime(ch->size(), basicRate_);

                STORE4BYTE(&index_, (rf->rf_ta));

                /* calculate rts duration field */
                macaprrh->psize = usec(mhNavData_.expire());

                macaprrh->seqno = cts_duration;

                rf->rf_duration = (macaprrh->psize + macaprrh->seqno);

                }

        pktRRTS_ = p;
}
```

```
void
Maca_p::sendRTS(int dst, int inflexible_bit, u_int16_t cts_duration)
{
        double control_gap;
        Packet *p = Packet::alloc();
        hdr_cmn* ch = HDR_CMN(p);
        struct rts_frame *rf = (struct rts_frame*)p->access(hdr_mac::offset_);
        struct hdr_tfrc *macaprh = hdr_tfrc::access(p);

        assert(pktTx_);
        assert(pktRTS_ == 0);
        /*
         *  If the size of the packet is larger than the
         *  RTSThreshold, then perform the RTS/CTS exchange.
         */
        if( (u_int32_t) HDR_CMN(pktTx_)->size() < macmib_.getRTSThreshold() ||
        (u_int32_t) dst == MAC_BROADCAST) {
                Packet::free(p);
                return;
        }

        ch->uid() = 0;
        ch->ptype() = PT_MAC;
        ch->size() = phymib_.getRTSlen();
        ch->iface() = -2;
        ch->error() = 0;

        bzero(rf, MAC_HDR_LEN);

        rf->rf_fc.fc_protocol_version = MAC_ProtocolVersion;
        rf->rf_fc.fc_type       = MAC_Type_Control;
        rf->rf_fc.fc_subtype    = MAC_Subtype_RTS;
        rf->rf_fc.fc_to_ds      = 0;
        rf->rf_fc.fc_from_ds    = 0;
        rf->rf_fc.fc_more_frag          = 0;
        rf->rf_fc.fc_retry      = 0;
        rf->rf_fc.fc_pwr_mgt    = 0;
        rf->rf_fc.fc_more_data          = 0;
        rf->rf_fc.fc_wep        = 0;
        rf->rf_fc.fc_order      = 0;

        if (is_idle()){

        macaprh->UrgentFlag = 0;   //inflexible bit=0
        macaprh->round_id = 1;          //master transmission = 1
        macaprh->rate = 0.0;
        tx_master_ = 1;
        //rf->rf_duration = RTS_DURATION(pktTx_);
        STORE4BYTE(&dst, (rf->rf_ra));
```

```
        /* store rts tx time */
        ch->txtime() = txtime(ch->size(), basicRate_);

        STORE4BYTE(&index_, (rf->rf_ta));

        control_gap = (3*txtime(phymib_.getCTSlen(), basicRate_)+
3*txtime(phymib_.getRTSlen(), basicRate_)+2*phymib_.getSIFS());
        /* calculate rts duration field */
        macaprh->psize = usec(phymib_.getSIFS()
                        + txtime(phymib_.getCTSlen(), basicRate_)
                        +control_gap);

        macaprh->seqno = usec( phymib_.getSIFS()
                                                + txtime(pktTx_)
                        + phymib_.getSIFS());
        rx_master_duration_ack = macaprh->seqno;

        rf->rf_duration = macaprh->psize + macaprh->seqno;

        tx_master_size_ = (u_int32_t) HDR_CMN(pktTx_)->size();

        }
        else if ((!is_idleMacap()) && tx_master_==1) {

                if ((u_int32_t) HDR_CMN(pktTx_)->size() > tx_master_size_) {

                        tx_master_ = 3;
                        macaprh->round_id = 1;          //master transmission = 1
                        macaprh->UrgentFlag = 0;
                        macaprh->rate = 0.0;


                        STORE4BYTE(&dst, (rf->rf_ra));

                        /* store rts tx time */
                        ch->txtime() = txtime(ch->size(), basicRate_);

                        STORE4BYTE(&index_, (rf->rf_ta));

                        control_gap = (3*txtime(phymib_.getCTSlen(), basicRate_)+
3*txtime(phymib_.getRTSlen(), basicRate_)+2*phymib_.getSIFS());
                        /* calculate rts duration field */
                        macaprh->psize = usec(phymib_.getSIFS()
                            + txtime(phymib_.getCTSlen(), basicRate_)
                            +control_gap);

                        macaprh->seqno = usec( phymib_.getSIFS()
                                                + txtime(pktTx_)
                            + phymib_.getSIFS());
```

```
                        rx_master_duration_ack = macaprh->seqno;

                        rf->rf_duration = macaprh->psize + macaprh->seqno;
                }
                else {
                        tx_master_ = 2;
                        macaprh->UrgentFlag = 1;
                        macaprh->round_id = 2;          //master transmission = 2
                        macaprh->rate = 0.0;

                        STORE4BYTE(&dst, (rf->rf_ra));

                        /* store rts tx time */
                        ch->txtime() = txtime(ch->size(), basicRate_);

                        STORE4BYTE(&index_, (rf->rf_ta));

                        /* calculate rts duration field */
                        macaprh->psize = usec(mhNavData_.expire());

                        macaprh->seqno = rx_master_duration_ack;

                        rf->rf_duration = macaprh->psize + macaprh->seqno;
                        }

        }

        else if (tx_master_==2) {

        tx_master_ = 3;
        macaprh->round_id = 1;          //master transmission = 1
        macaprh->UrgentFlag = 0;
        macaprh->rate = 0.0;

        STORE4BYTE(&dst, (rf->rf_ra));

        /* store rts tx time */
        ch->txtime() = txtime(ch->size(), basicRate_);

        STORE4BYTE(&index_, (rf->rf_ta));

        control_gap = (3*txtime(phymib_.getCTSlen(), basicRate_)+
3*txtime(phymib_.getRTSlen(), basicRate_)+2*phymib_.getSIFS());
        /* calculate rts duration field */
        macaprh->psize = usec(phymib_.getSIFS()
                        + txtime(phymib_.getCTSlen(), basicRate_)
                        +control_gap);

        macaprh->seqno = usec( phymib_.getSIFS()
                                                + txtime(pktTx_)
```

```
                                + phymib_.getSIFS());
        rx_master_duration_ack = macaprh->seqno;

        rf->rf_duration = macaprh->psize + macaprh->seqno;
        }
        else{
        macaprh->UrgentFlag = 0;   //inflexible bit=0
        macaprh->round_id = 1;          //master transmission = 1
        tx_master_ = 1;
        macaprh->rate = 0.0;

        //rf->rf_duration = RTS_DURATION(pktTx_);
        STORE4BYTE(&dst, (rf->rf_ra));

        /* store rts tx time */
        ch->txtime() = txtime(ch->size(), basicRate_);

        STORE4BYTE(&index_, (rf->rf_ta));

        control_gap = (3*txtime(phymib_.getCTSlen(), basicRate_)+
3*txtime(phymib_.getRTSlen(), basicRate_)+2*phymib_.getSIFS());
        /* calculate rts duration field */
        macaprh->psize = usec(phymib_.getSIFS()
                                + txtime(phymib_.getCTSlen(), basicRate_)
                                +control_gap);

        macaprh->seqno = usec( phymib_.getSIFS()
                                                        + txtime(pktTx_)
                                + phymib_.getSIFS());
        rx_master_duration_ack = macaprh->seqno;

        rf->rf_duration = macaprh->psize + macaprh->seqno;

        tx_master_size_ = (u_int32_t) HDR_CMN(pktTx_)->size();

        }

        pktRTS_ = p;
}

void
Maca_p::sendCTS(int dst, double rts_Tdata_duration, double rts_Tack_duration, int
inflexible_bit)
{
        Packet *p = Packet::alloc();
        hdr_cmn* ch = HDR_CMN(p);
        struct cts_frame *cf = (struct cts_frame*)p->access(hdr_mac::offset_);
        struct hdr_tfrc *macapch = hdr_tfrc::access(p);
```

```
assert(pktCTRL_ == 0);
ch->uid() = 0;
ch->ptype() = PT_MAC;
ch->size() = phymib_.getCTSlen();


ch->iface() = -2;
ch->error() = 0;
//ch->direction() = hdr_cmn::DOWN;
bzero(cf, MAC_HDR_LEN);

cf->cf_fc.fc_protocol_version = MAC_ProtocolVersion;
cf->cf_fc.fc_type       = MAC_Type_Control;
cf->cf_fc.fc_subtype    = MAC_Subtype_CTS;
cf->cf_fc.fc_to_ds      = 0;
cf->cf_fc.fc_from_ds    = 0;
cf->cf_fc.fc_more_frag          = 0;
cf->cf_fc.fc_retry      = 0;
cf->cf_fc.fc_pwr_mgt    = 0;
cf->cf_fc.fc_more_data          = 0;
cf->cf_fc.fc_wep        = 0;
cf->cf_fc.fc_order      = 0;


//cf->cf_duration = CTS_DURATION(rts_duration);
STORE4BYTE(&dst, (cf->cf_ra));

/* store cts tx time */
ch->txtime() = txtime(ch->size(), basicRate_);

if (inflexible_bit==0){

macapch->UrgentFlag = 0;
macapch->round_id = 0;
/* calculate cts duration */
macapch->psize = usec(sec(rts_Tdata_duration)
                - phymib_.getSIFS()
                - txtime(phymib_.getCTSlen(), basicRate_));

macapch->seqno = usec(sec(rts_Tack_duration));
cf->cf_duration = macapch->psize + macapch->seqno ;
}
else if (inflexible_bit==1){
          macapch->UrgentFlag = 1;
          macapch->round_id = 1;
          macapch->psize = usec(mhNavData_.expire());
          macapch->seqno = usec(sec(rts_Tack_duration));
          cf->cf_duration = macapch->psize + macapch->seqno ;
}
```

```
        pktCTRL_ = p;
}


/* ==============  Incoming Packet Routines========================== */

void
Maca_p::send(Packet *p, Handler *h)
{
        double rTime;
        struct hdr_mac802_11* dh = HDR_MAC802_11(p);
        EnergyModel *em = netif_->node()->energy_model();
        if (em && em->sleep()) {
                em->set_node_sleep(0);
                em->set_node_state(EnergyModel::INROUTE);
        }

        callback_ = h;
        sendDATA(p);
        sendRTS(ETHER_ADDR(dh->dh_ra),0,0);

        /*
         * Assign the data packet a sequence number.
         */
        dh->dh_scontrol = sta_seqno_++;

        /*
         *  If the medium is IDLE, we must wait for a DIFS
         *  Space before transmitting.
         */

        if(mhBackoff_.busy() == 0) {
                if(is_idleMacap() && is_idle()) {
                        if (mhDefer_.busy() == 0) {
                                /*
                                 * If we are already deferring, there is no
                                 * need to reset the Defer timer.
                                 */
                                if ((u_int32_t) HDR_CMN(pktTx_)->size() <
macmib_.getRTSThreshold() )  {
                                mhBackoff_.start(cw_, is_idle());
                                Mac802_11::send(p,h);      /// CALL mac_802.11
                                return;
                                }
                                rTime = (Random::random() % cw_)
                                        * (phymib_.getSlotTime());
                                mhDefer_.start(phymib_.getDIFS() + rTime);
                        }
                } else
```

```
                if ( (!is_idleMacap()) && rx_state_==MAC_IDLE &&  tx_master_==2)
                        {
                        deferHandler();
                        }
                        else {
                        /*
                         * If the medium is NOT IDLE, then we start
                         * the backoff timer.
                         */
                        mhBackoff_.start(cw_, is_idle());
                        }
        }
}

void
Maca_p::recv(Packet *p, Handler *h)
{
        struct hdr_cmn *hdr = HDR_CMN(p);
        /*
         * Sanity Check
         */
        assert(initialized());

        /*
         *  Handle outgoing packets.
         */
        if(hdr->direction() == hdr_cmn::DOWN) {
           send(p, h);
           return;
      }
        /*
         *  Handle incoming packets.
         *
         *  We just received the 1st bit of a packet on the network
         *  interface.
         *
         */

        /*
         *  If the interface is currently in transmit mode, then
         *  it probably won't even see this packet.  However, the
         *  "air" around me is BUSY so I need to let the packet
         *  proceed.  Just set the error flag in the common header
         *  to that the packet gets thrown away.
         */

        if(tx_active_ &&hdr->error() == 0) {
                hdr->error() = 1;
        }
```

```
        if(rx_state_ == MAC_IDLE ) {
                setRxState(MAC_RECV);
                pktRx_ = p;
                /*
                 * Schedule the reception of this packet, in
                 * txtime seconds.
                 */
        if (mhRecv_.busy())             mhRecv_.stop();
                mhRecv_.start(txtime(p));
        } else {
                /*
                 *  If the power of the incoming packet is smaller than the
                 *  power of the packet currently being received by at least
            *  the capture threshold, then we ignore the new packet.
                 */
                if(pktRx_->txinfo_.RxPr / p->txinfo_.RxPr >= p->txinfo_.CPThresh) {
                        capture(p);
                } else {
                        collision(p);
                }
        }
}

void
Maca_p::recv_timer()
{
        u_int32_t src;
        hdr_cmn *ch = HDR_CMN(pktRx_);
        hdr_mac802_11 *mh = HDR_MAC802_11(pktRx_);
        u_int32_t dst = ETHER_ADDR(mh->dh_ra);
        struct hdr_tfrc *macaprh = hdr_tfrc::access(pktRx_);


        u_int8_t  type = mh->dh_fc.fc_type;
        u_int8_t  subtype = mh->dh_fc.fc_subtype;

        assert(pktRx_);
        assert(rx_state_ == MAC_RECV || rx_state_ == MAC_COLL);
      /*
       *  If the interface is in TRANSMIT mode when this packet
       *  "arrives", then I would never have seen it and should
       *  do a silent discard without adjusting the NAV.
       */
      if(tx_active_) {
            Packet::free(pktRx_);
            goto done;
      }
```

```
        /*
         * Handle collisions.
         */
        if(rx_state_ == MAC_COLL) {
                discard(pktRx_, DROP_MAC_COLLISION);
                set_nav(usec(phymib_.getEIFS()));
        set_navData(usec(phymib_.getEIFS()));
                goto done;
        }

        /*
         * Check to see if this packet was received with enough
         * bit errors that the current level of FEC still could not
         * fix all of the problems - ie; after FEC, the checksum still
         * failed.
         */
        if( ch->error() ) {
                Packet::free(pktRx_);
                set_nav(usec(phymib_.getEIFS()));
set_navData(usec(phymib_.getEIFS()));

                goto done;
        }

        /*
         * IEEE 802.11 specs, section 9.2.5.6
         *      - update the NAV (Network Allocation Vector)
         */


        if(dst != (u_int32_t)index_) {
                if ( subtype == MAC_Subtype_RTS && macaprh->round_id ==2) {
                        tx_master_ = 2;
                }
                else if ( subtype == MAC_Subtype_CTS && macaprh->round_id ==2) {
                        rx_master_ = 2;
                }
                set_nav(mh->dh_duration);
                if ( subtype == MAC_Subtype_RTS || subtype == MAC_Subtype_CTS)
                        set_navData(macaprh->psize);

        }


      /* tap out - */

      if (tap_ && type == MAC_Type_Data &&
         MAC_Subtype_Data == subtype )
                tap_->tap(pktRx_);
```

```
/*
         * Adaptive Fidelity Algorithm Support - neighborhood infomation
         * collection
         *
         * Hacking: Before filter the packet, log the neighbor node
         * I can hear the packet, the src is my neighbor
         */
        if (netif_->node()->energy_model() &&
           netif_->node()->energy_model()->adaptivefidelity()) {
                src = ETHER_ADDR(mh->dh_ta);
                netif_->node()->energy_model()->add_neighbor(src);
        }
        /*
         * Address Filtering
         */
        if(dst != (u_int32_t)index_ && dst != MAC_BROADCAST) {
                /*
                 *  We don't want to log this event, so we just free
                 *  the packet instead of calling the drop routine.
                 */
                discard(pktRx_, "---");
                goto done;
        }

        switch(type) {

        case MAC_Type_Management:
                discard(pktRx_, DROP_MAC_PACKET_ERROR);
                goto done;
        case MAC_Type_Control:
                switch(subtype) {
                case MAC_Subtype_RTS:
                        if ( macaprh->rate==1.0) {
                        recvRRTS(pktRx_);
                }
                else {
                         recvRTS(pktRx_);
                        }
                        break;
                case MAC_Subtype_CTS:
                        recvCTS(pktRx_);
                        break;
                case MAC_Subtype_ACK:
                        recvACK(pktRx_);
                        break;
                default:
                        fprintf(stderr,"recvTimer1:Invalid MAC Control Subtype %x\n",
                                subtype);
                        exit(1);
                }
```

```
                break;
        case MAC_Type_Data:
                switch(subtype) {
                case MAC_Subtype_Data:
                        recvDATA(pktRx_);
                        break;
                default:
                        fprintf(stderr, "recv_timer2:Invalid MAC Data Subtype %x\n",
                                subtype);
                        exit(1);
                }
                break;
        default:
                fprintf(stderr, "recv_timer3:Invalid MAC Type %x\n", subtype);
                exit(1);
        }
 done:
        pktRx_ = 0;
        rx_resume();
}

void
Maca_p::recvRTS(Packet *p)
{
        hdr_cmn *ch = HDR_CMN(p);
        hdr_mac802_11 *mh = HDR_MAC802_11(p);
        u_int32_t dst = ETHER_ADDR(mh->dh_ra);
        u_int32_t src = ETHER_ADDR(mh->dh_ta);
        struct rts_frame *rf = (struct rts_frame*)p->access(hdr_mac::offset_);
        struct hdr_tfrc *macaprh = hdr_tfrc::access(p);

        if (tx_state_ != MAC_IDLE) {
                discard(p, DROP_MAC_BUSY);
                return;
        }

        /*
         *  If I'm responding to someone else, discard this RTS.
         */
        if(pktCTRL_) {
                discard(p, DROP_MAC_BUSY);
                return;
        }
        if (macaprh->UrgentFlag == 1) {
                if (rx_master_==0 || rx_master_==1) {
                        rx_master_ = 1;
                        sendCTS(ETHER_ADDR(rf->rf_ta), macaprh->psize, macaprh-
>seqno, 0);
                        }
```

```
                        else {
                                discard(p, DROP_MAC_BUSY);
                                return;
                        }
                }
                else
                        if (macaprh->UrgentFlag == 0) {
                                if (rx_master_==0 ||rx_master_==1) {
                                rx_master_ = 1;
                                rx_master_Tdata = macaprh->psize;
                                rx_master_Tack = macaprh->seqno;
                                sendCTS(ETHER_ADDR(rf->rf_ta), macaprh->psize, macaprh-
>seqno, 0);
                                }
                                else
                                if (rx_master_==2) {
                                rx_master_ = 2;
                                sendCTS(ETHER_ADDR(rf->rf_ta), rx_master_Tdata,
rx_master_Tack, 1);
                                }
                        else {
                                        discard(p, DROP_MAC_BUSY);
                        return;
                        }
                }


                /*
                 *  Stop deferring - will be reset in tx_resume().
                 */
                if(mhDefer_.busy()) mhDefer_.stop();

                tx_resume();

                mac_log(p);
}


void
Maca_p::recvRRTS(Packet *p)
{
        hdr_cmn *ch = HDR_CMN(p);
        hdr_mac802_11 *mh = HDR_MAC802_11(p);
        u_int32_t dst = ETHER_ADDR(mh->dh_ra);
        u_int32_t src = ETHER_ADDR(mh->dh_ta);

        struct rts_frame *rf = (struct rts_frame*)p->access(hdr_mac::offset_);
        struct hdr_tfrc *macaprrh = hdr_tfrc::access(p);
```

```
/*
         *  If I'm responding to someone else, discard this RTS.
         */

mhSend_.stop();

        if (macaprrh->UrgentFlag == 2) {
                mhNav_.stop();
                mhNavData_.stop();
                tx_master_ --;
        }

        /*
         *  Stop deferring - will be reset in tx_resume().
         */
        if(mhDefer_.busy()) mhDefer_.stop();

        tx_resume();

        mac_log(p);
}



void
Maca_p::recvCTS(Packet *p)
{
        hdr_cmn *ch = HDR_CMN(p);
        struct hdr_mac802_11* dh = HDR_MAC802_11(p);
        u_int32_t dst = ETHER_ADDR(dh->dh_ra);
        u_int32_t src = ETHER_ADDR(dh->dh_ta);

        struct hdr_tfrc *macapch = hdr_tfrc::access(p);
        double rTime;

        if(tx_state_ != MAC_RTS) {
                discard(p, DROP_MAC_INVALID_STATE);
                return;
        }

        assert(pktRTS_);
        Packet::free(pktRTS_); pktRTS_ = 0;

        assert(pktTx_);
        mhSend_.stop();
```

```
        /*
         * The successful reception of this CTS packet implies
         * that our RTS was successful.
         * According to the IEEE spec 9.2.5.3, you must
         * reset the ssrc_, but not the congestion window.
         */

        if ((tx_master_ == 1 || tx_master_ ==0 ) && (macapch->UrgentFlag == 1)){
//EUTHIGRAMMISI ME TON PARALIPTI
                sendRRTS(ETHER_ADDR(dh->dh_ra),1,macapch->seqno);
                check_pktRRTS();
                return ;
        }

        ssrc_ = 0;
        tx_resume();

        mac_log(p);
}


void
Maca_p::recvDATA(Packet *p)
{
        struct hdr_mac802_11 *dh = HDR_MAC802_11(p);
        u_int32_t dst, src, size;
        struct hdr_cmn *ch = HDR_CMN(p);

        dst = ETHER_ADDR(dh->dh_ra);
        src = ETHER_ADDR(dh->dh_ta);
        size = ch->size();
        /*
         * Adjust the MAC packet size - ie; strip
         * off the mac header
         */
        ch->size() -= phymib_.getHdrLen11();
        ch->num_forwards() += 1;

        /*
         *  If we sent a CTS, clean up...
         */
        if(dst != MAC_BROADCAST) {
                if(size >= macmib_.getRTSThreshold()) {
                        if (tx_state_ == MAC_CTS ) {
                                assert(pktCTRL_);
                                Packet::free(pktCTRL_); pktCTRL_ = 0;
                                mhSend_.stop();
                                /*
                                 * Our CTS got through.
                                 */
```

```
                        } else {
                                discard(p, DROP_MAC_BUSY);
                                return;
                        }
                        sendACK(src);
                        tx_resume();
                } else {
                        /*
                         *  We did not send a CTS and there's no
                         *  room to buffer an ACK.
                         */
                if(pktCTRL_) {
                                discard(p, DROP_MAC_BUSY);
                                return;
                        }
                        sendACK(src);
                        if(mhSend_.busy() == 0)
                                tx_resume();
                }
        }

/* ======= Make/update an entry in our sequence number cache.============= */

      if(dst != MAC_BROADCAST) {
            if (src < (u_int32_t) cache_node_count_) {
                Host *h = &cache_[src];

                if(h->seqno && h->seqno == dh->dh_scontrol) {
                     discard(p, DROP_MAC_DUPLICATE);
                     return;
                }
                h->seqno = dh->dh_scontrol;
            } else {
                        static int count = 0;
                        if (++count <= 10) {
                                printf ("MAC_802_11: accessing MAC cache_ array out of
range (src %u, dst %u, size %d)!\n", src, dst, cache_node_count_);
                                if (count == 10)
                                        printf ("[suppressing additional MAC cache_
warnings]\n");
                        };
                };
        }

        /*
         *  Pass the packet up to the link-layer.
         *  XXX - we could schedule an event to account
         *  for this processing delay.
         */
```

```
        /* in BSS mode, if a station receives a packet via
         * the AP, and higher layers are interested in looking
         * at the src address, we might need to put it at
         * the right place - lest the higher layers end up
         * believing the AP address to be the src addr! a quick
         * grep didn't turn up any higher layers interested in
         * the src addr though!
         * anyway, here if I'm the AP and the destination
         * address (in dh_3a) isn't me, then we have to fwd
         * the packet; we pick the real destination and set
         * set it up for the LL; we save the real src into
         * the dh_3a field for the 'interested in the info'
         * receiver; we finally push the packet towards the
         * LL to be added back to my queue - accomplish this
         * by reversing the direction!*/

        if ((bss_id() == addr()) && ((u_int32_t)ETHER_ADDR(dh->dh_ra)!=
MAC_BROADCAST)&& ((u_int32_t)ETHER_ADDR(dh->dh_3a) != addr())) {
                struct hdr_cmn *ch = HDR_CMN(p);
                u_int32_t dst = ETHER_ADDR(dh->dh_3a);
                u_int32_t src = ETHER_ADDR(dh->dh_ta);
                /* if it is a broadcast pkt then send a copy up
                 * my stack also
                 */
                if (dst == MAC_BROADCAST) {
                        uptarget_->recv(p->copy(), (Handler*) 0);
                }

                ch->next_hop() = dst;
                STORE4BYTE(&src, (dh->dh_3a));
                ch->addr_type() = NS_AF_ILINK;
                ch->direction() = hdr_cmn::DOWN;
        }

        uptarget_->recv(p, (Handler*) 0);
}



/* ================== TCL Hooks for the simulator=================== */

static class Maca_pClass : public TclClass {
public:
        Maca_pClass() : TclClass("Mac/Maca_p") {}
        TclObject* create(int, const char*const*) {
                return (new Maca_p());
        }
} class_Maca_p;
```

Maca_p.h

```
#ifndef ns_maca_p_h
#define ns_maca_p_h

#include "delay.h"
#include "connector.h"
#include "packet.h"
#include "random.h"
#include "mobilenode.h"

// #define DEBUG 99

#include "arp.h"
#include "ll.h"
#include "mac.h"
#include "mac-timers.h"
#include "mac-802_11.h"
#include "cmu-trace.h"

// Added by Sushmita to support event tracing
#include "agent.h"
#include "basetrace.h"

class Maca_p;

/****************** MACA-P TIMER ************************************/

class MacapTimer : public Handler {
public:
        MacapTimer(Maca_p* m) : macap(m) {
                busy_ = paused_ = 0; stime = rtime = 0.0;
        }

        virtual void handle(Event *e) = 0;

        virtual void start(double time);
        virtual void stop(void);
        virtual void pause(void) { assert(0); }
        virtual void resume(void) { assert(0); }

        inline int busy(void) { return busy_; }
        inline int paused(void) { return paused_; }
        inline double expire(void) {
                return ((stime + rtime) - Scheduler::instance().clock());
        }
```

```
protected:
        Maca_p                  *macap;
        int             busy_;
        int             paused_;
        Event           intr;
        double          stime;  // start time
        double          rtime;  // remaining time
};

class NavDataTimer : public MacapTimer {
public:
        NavDataTimer(Maca_p *m) : MacapTimer(m) {}

        void    handle(Event *e);
};


/************************* PHY_MIB2 ***********************************/

class PHY_MIB2 {
public:
        PHY_MIB2(Maca_p *parent);

        inline u_int32_t getCWMin() { return(CWMin); }
        inline u_int32_t getCWMax() { return(CWMax); }
        inline double getSlotTime() { return(SlotTime); }
        inline double getSIFS() { return(SIFSTime); }
        inline double getPIFS() { return(SIFSTime + SlotTime); }
        inline double getDIFS() { return(SIFSTime + 2 * SlotTime); }
        inline double getEIFS() {
                // see (802.11-1999, 9.2.10)
                return(SIFSTime + getDIFS()
                + (8 *  getACKlen())/PLCPDataRate);
        }
        inline u_int32_t getPreambleLength() { return(PreambleLength); }
        inline double getPLCPDataRate() { return(PLCPDataRate); }

        inline u_int32_t getPLCPhdrLen() {
                return((PreambleLength + PLCPHeaderLength) >> 3);
        }

        inline u_int32_t getHdrLen11() {
                return(getPLCPhdrLen() + sizeof(struct hdr_mac802_11)
                + ETHER_FCS_LEN);
        }

        inline u_int32_t getRTSlen() {
                return(getPLCPhdrLen() + sizeof(struct rts_frame));
        }
```

```
inline u_int32_t getCTSlen() {
        return(getPLCPhdrLen() + sizeof(struct cts_frame));
    }

    inline u_int32_t getACKlen() {
        return(getPLCPhdrLen() + sizeof(struct ack_frame));
    }

 private:
    u_int32_t    CWMin;
    u_int32_t    CWMax;
    double       SlotTime;
    double       SIFSTime;
    u_int32_t    PreambleLength;
    u_int32_t    PLCPHeaderLength;
    double       PLCPDataRate;
};

/************************ MAC_MIB2 ******************************/

class MAC_MIB2 {
public:

    MAC_MIB2(Maca_p*);

private:
    u_int32_t    RTSThreshold;
    u_int32_t    ShortRetryLimit;
    u_int32_t    LongRetryLimit;
public:
    u_int32_t    FailedCount;
    u_int32_t    RTSFailureCount;
    u_int32_t    ACKFailureCount;
 public:
   inline u_int32_t getRTSThreshold() { return(RTSThreshold);}
   inline u_int32_t getShortRetryLimit() { return(ShortRetryLimit);}
   inline u_int32_t getLongRetryLimit() { return(LongRetryLimit);}
};

/************************ MACA-P*************************************/

class Maca_p : public Mac802_11 {

    friend class NavDataTimer;

public:
    Maca_p();

    void         recv(Packet *p, Handler *h);
protected:
```

```
        void    navDataHandler();

        NavDataTimermhNavData_;

        Packet          *pktRRTS_;

        void            rx_resume(void);
        void            tx_resume(void);
        inline int      is_idleMacap(void);

        /*
         * Called by the timers.
         */
public:
        void            send_timer(void);
        void            recv_timer(void);

        int             check_pktCTRL();
        int             check_pktRTS();
        int             check_pktRRTS();
        int             check_pktTx();

        /*
         * Packet Transmission Functions.
         */
        void    send(Packet *p, Handler *h);
        void    sendRTS(int dst, int inflexible_bit,u_int16_t cts_duration);
        void    sendRRTS(int dst, int inflexible_bit,u_int16_t cts_duration);
        void    sendCTS(int dst, double rts_Tdata_duration, double rts_Tack_duration, int
inflexible_bit);


        /*
         * Packet Reception Functions.
         */
        void    recvRTS(Packet *p);
        void    recvRRTS(Packet *p);
        void    recvCTS(Packet *p);
        void    recvDATA(Packet *p);
        void    capture(Packet *p);
```

```
/* ================== Internal MAC State====================== */

        double          navData_;       // maca-p   Network Allocation Vector

        int                     rrts;
        int             tx_master_;
        int                     rx_master_;
        double tx_master_size_;

        u_int32_t       rx_master_duration_ack;
        double                  rx_master_Tdata;
        double                  rx_master_Tack;


inline void set_navData(u_int16_t us) {
        double now = Scheduler::instance().clock();
        double t = us * 1e-6;
        if((now + t) > navData_) {
                navData_ = now + t;
                if(mhNavData_.busy())
                        mhNavData_.stop();
                mhNavData_.start(t);
        }
}

protected:
                        PHY_MIB2        phymib_;
                        MAC_MIB2         macmib_;


};


#endif /* __maca_p_h__ */
```

ns-default.tcl

```
…….
…….
Mac/802_11 set RTSThreshold_       0          ;# bytes
Mac/802_11 set ShortRetryLimit_    7          ;# retransmittions
Mac/802_11 set LongRetryLimit_     4          ;# retransmissions


#change by Demetris
# MACA-P MIB parameters
#

Mac/Maca_p set CWMin_                       31
Mac/Maca_p set CWMax_                       1023
Mac/Maca_p set SlotTime_                    0.000020        ;# 20us
Mac/Maca_p set SIFS_                  0.000010        ;# 10us
Mac/Maca_p set PreambleLength_              144             ;# 144 bit
Mac/Maca_p set PLCPHeaderLength_      48              ;# 48 bits
Mac/Maca_p set PLCPDataRate_                1.0e6           ;# 1Mbps
Mac/Maca_p set RTSThreshold_                0               ;# bytes
Mac/Maca_p set ShortRetryLimit_             7               ;# retransmittions
Mac/Maca_p set LongRetryLimit_               4              ;# retransmissions

#
# Support for Abstract LAN
#

Classifier/Replicator set direction_ false
……
…….
```

ns-packet.tcl

```
……
……
UMP
        Pushback
        SCTP
        Smac
        NV
        MACAP
} {
        add-packet-header $prot
}
……
……...
```

ΠΑΡΑΡΤΗΜΑ Β – ΣΕΝΑΡΙΑ ΠΡΟΣΟΜΟΙΩΣΗΣ

Στο παράρτημα αυτό παρατίθενται οι κώδικες των σεναρίων 1 και 2 καθώς και ο κώδικας του τρίτου σεναρίου τόσο για CBR πηγή όσο και για PARETO.

Scenario1.tcl

```
#############################################################################
# Get command line arguments and create trace path
#
# Create $nodes downlink gateways and $connections TCP conn per gateway

set USAGE "ns Scenario1.tcl <nodes> <protocol> <seed> "
if {$argc != 3} {puts $USAGE; exit 0}

set nodes       [lindex $argv 0];
set protocol    [lindex $argv 1];
set seed        [lindex $argv 2];

if {$nodes < 2} {append n 0 $nodes} else {set n $nodes}
if {$protocol < 10} {append p 0 $protocol} else {set p $protocol}

append scenario "Scenario1/" $n "-nodes/"  $p "-protocol/" $seed "-seed/"
puts "Creating $scenario..."
file mkdir $scenario
set proto "Mac/"
append proto $p



global defaultRNG
$defaultRNG seed $seed


#############################################################################

# ======================================================================
# Define options
# ======================================================================

set val(chan)       Channel/WirelessChannel         ;# channel type
set val(prop)       Propagation/TwoRayGround        ;# radio-propagation model
set val(netif)      Phy/WirelessPhy                 ;# network interface type
set val(mac)        $proto                          ;# MAC type
set val(ifq)        Queue/DropTail/PriQueue         ;# interface queue type
set val(ll)         LL                              ;# link layer type
```

```
set val(ant)       Antenna/OmniAntenna              ;# antenna model
set val(ifqlen)    50                               ;# max packet in ifq
set val(nn)        $n                               ;# number of mobilenodes
set val(rp)        DSDV                             ;# routing protocol

# ======================================================================
# Main Program
# ======================================================================

Phy/WirelessPhy set Pt_ 0.2818
add-packet-header SCTP

#
# Initialize Global Variables
#

set ns_          [new Simulator]
set tracefd      [open $scenario/trace.tr w]


$ns_ use-newtrace
$ns_ trace-all $tracefd


# set up topography object
set topo      [new Topography]

$topo load_flatgrid     4500 4500

#
# Create God
#
create-god $val(nn)

#
#  Create the specified number of mobilenodes [$val(nn)] and "attach" them
#  to the channel.
#  Here two nodes are created : node(0) and node(1)

# configure node

$ns_ node-config -adhocRouting $val(rp) \
           -llType $val(ll) \
           -macType $val(mac) \
           -ifqType $val(ifq) \
           -ifqLen $val(ifqlen) \
           -antType $val(ant) \
           -propType $val(prop) \
           -phyType $val(netif) \
```

```
            -channelType $val(chan) \
            -topoInstance $topo \
            -agentTrace ON \
            -routerTrace OFF \
            -macTrace OFF \
            -movementTrace OFF


for {set i 0} {$i < $val(nn) } {incr i} {
        set node_($i) [$ns_ node]
        $node_($i) random-motion 0          ;# disable random motion
        $node_($i) set X_ [expr (10+(200.0 * ($i)))]
        $node_($i) set Y_ 10.0
        $node_($i) set Z_ 0.0
}

#
# Progress report
#
set progress_interval   60.0
set start_time          [clock seconds]
set sim_duration        300.1

proc progress_report {} {
        global  progress_interval       \
                start_time              \
                ns_

        set sim_now [$ns_ now]
        set tmp2 [clock seconds]
        set tmp [expr $tmp2 - $start_time]
        set now [clock format $tmp -format %H:%M:%S -gmt 1]
        set tmp [expr $sim_now / 60.0]
        puts "$now : $tmp \bmin"
        $ns_ at [expr $sim_now + $progress_interval] "progress_report"
}

$ns_ at 0.0 "progress_report"
$ns_ at $sim_duration "stop"



set j 3
set k -1
for {set i 0} {$i < $val(nn)} {set i [expr $i + 2]} {
incr k
puts "LOOP i: $i"
```

```
if {[expr  ($j)%3 == 0] || [expr  ($j)%3 == 1]} {
puts "creating UDP & CBR $k on node $i"
        set udp($k) [new Agent/UDP]
        $udp($k) set packetSize_ 2000
        $ns_ attach-agent $node_($i) $udp($k)

        set cbr($k) [new Application/Traffic/CBR]
        $cbr($k) set packetSize_ 2000
        $cbr($k) set interval_ 0.001
        $cbr($k) attach-agent $udp($k)

        set null($k) [new Agent/Null]
        $ns_ attach-agent $node_([expr $i + 1]) $null($k)
        $ns_ connect $udp($k) $null($k)

        incr j
} else {
set tmp [expr $i + 1]
puts "creating UDP & CBR $k on node $tmp"
        set udp($k) [new Agent/UDP]
        $udp($k) set packetSize_ 2000
        $ns_ attach-agent $node_([expr $i + 1]) $udp($k)

        set cbr($k) [new Application/Traffic/CBR]
        $cbr($k) set packetSize_ 2000
        $cbr($k) set interval_ 0.001
        $cbr($k) attach-agent $udp($k)

        set null($k) [new Agent/Null]
        $ns_ attach-agent $node_($i) $null($k)
        $ns_ connect $udp($k) $null($k)

        incr j
        }
}

for {set i 0} {$i < ([expr $val(nn)/2]) } {incr i} {
puts "starting cbr $i"
        $ns_ at 11.0 "$cbr($i) start"
        $ns_ at 31.1 "$cbr($i) stop"
        }

#
# Tell nodes when the simulation ends
#
for {set i 0} {$i < $val(nn) } {incr i} {
   $ns_ at 150.0 "$node_($i) reset";
}
```

```
#$ns_ at 150.0 "stop"
#$ns_ at 150.01 "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
   global ns_ tracefd
puts "stop1"
   $ns_ flush-trace
   close $tracefd
puts "stop2"
         exit 0
}

puts "Starting Simulation..."
$ns_ run
```

Scenario2.tcl

```
#============================================================================
# Get command line arguments and create trace path
#
# Create $nodes downlink gateways and $connections TCP conn per gateway

set USAGE "ns Scenario2.tcl <nodes> <protocol> <seed>"
if {$argc != 3} {puts $USAGE; exit 0}

set nodes       [lindex $argv 0];
set protocol    [lindex $argv 1];
set seed        [lindex $argv 2];


if {$nodes < 2} {append n 0 $nodes} else {set n $nodes}
if {$protocol < 10} {append p 0 $protocol} else {set p $protocol}

append scenario "Scenario2/" $n "-nodes/"  $p "-protocol/" $seed "-seed/"
puts "Creating $scenario..."
file mkdir $scenario
set proto "Mac/"
append proto $p
global defaultRNG
$defaultRNG seed $seed
```

```
#######################################################################
# ====================================================================
# Define options
# ====================================================================

set val(chan)       Channel/WirelessChannel          ;# channel type
set val(prop)       Propagation/TwoRayGround         ;# radio-propagation model
set val(netif)      Phy/WirelessPhy                  ;# network interface type
set val(mac)        $proto                           ;# MAC type
set val(ifq)        Queue/DropTail/PriQueue          ;# interface queue type
set val(ll)         LL                               ;# link layer type
set val(ant)        Antenna/OmniAntenna              ;# antenna model
set val(ifqlen)     50                               ;# max packet in ifq
set val(nn)         $n                               ;# number of mobilenodes
set val(rp)         DSDV                             ;# routing protocol

#======================================================================
# Main Program
#======================================================================

Phy/WirelessPhy set Pt_ 0.2818
add-packet-header SCTP


#
# Initialize Global Variables
#

set ns_         [new Simulator]
set tracefd     [open $scenario/trace.tr w]

$ns_ use-newtrace
$ns_ trace-all $tracefd

# set up topography object
set topo      [new Topography]

$topo load_flatgrid 1500 1500

#
# Create God
#
create-god $val(nn)

#
#  Create the specified number of mobilenodes [$val(nn)] and "attach" them
#  to the channel.
#  Here two nodes are created : node(0) and node(1)
```

```
# configure node

$ns_ node-config -adhocRouting $val(rp) \
           -llType $val(ll) \
           -macType $val(mac) \
           -ifqType $val(ifq) \
           -ifqLen $val(ifqlen) \
           -antType $val(ant) \
           -propType $val(prop) \
           -phyType $val(netif) \
           -channelType $val(chan) \
           -topoInstance $topo \
           -agentTrace ON \
           -routerTrace OFF \
           -macTrace OFF \
           -movementTrace OFF

set j 0
for {set i 0} {$i <[expr  $val(nn)/2] } {incr i} {
          incr j
          set node_($i) [$ns_ node]
          $node_($i) random-motion 0            ;# disable random motion
          $node_($i) set X_ [expr (10+(200.0 * ($j)))]
          $node_($i) set Y_ 10.0
          $node_($i) set Z_ 0.0
          set node_([expr ($val(nn)-($i+1))]) [$ns_ node]
          $node_([expr ($val(nn)-($i+1))]) random-motion 0             ;# disable random
motion
          $node_([expr ($val(nn)-($i+1))]) set X_ [expr (10+(200.0 * ($j)))]
          $node_([expr ($val(nn)-($i+1))]) set Y_ 210.0
          $node_([expr ($val(nn)-($i+1))]) set Z_ 0.0
}

for {set i 0} {$i < $val(nn) } {incr i} {
          set tmpX [$node_($i) set X_]
          set tmpY [$node_($i) set Y_]
          puts "node:$i X:$tmpX Y:$tmpY"
}

#
# Progress report
#
set progress_interval  60.0
set start_time         [clock seconds]
set sim_duration       300.1

proc progress_report {} {
          global  progress_interval       \
                  start_time              \
                  ns_
```

```
        set sim_now [$ns_ now]
        set tmp2 [clock seconds]
        set tmp [expr $tmp2 - $start_time]
        set now [clock format $tmp -format %H:%M:%S -gmt 1]
        set tmp [expr $sim_now / 60.0]
        puts "$now : $tmp \bmin"
        $ns_ at [expr $sim_now + $progress_interval] "progress_report"
}

$ns_ at 0.0 "progress_report"
$ns_ at $sim_duration "stop"


set j 3
set k -1
for {set i 0} {$i < [expr ($val(nn)/2  )]} {set i [expr $i + 2]} {
incr k
puts "LOOP i: $i"
set tmp [expr $i + 1]
if {[expr  ($j)%3 == 0] || [expr  ($j)%3 == 1]} {
puts "creating UDP & CBR $k  node $i -> $tmp"
        set udp($k) [new Agent/UDP]
        $udp($k) set packetSize_ 2000
        $ns_ attach-agent $node_($i) $udp($k)

        set cbr($k) [new Application/Traffic/CBR]
        $cbr($k) set packetSize_ 2000
        $cbr($k) set interval_ 0.001
        $cbr($k) attach-agent $udp($k)

        set null($k) [new Agent/Null]
        $ns_ attach-agent $node_([expr $i + 1]) $null($k)
        $ns_ connect $udp($k) $null($k)

        incr j
} else {
set tmp [expr $i + 1]
puts "creating UDP & CBR $k  node $tmp -> $i"
        set udp($k) [new Agent/UDP]
        $udp($k) set packetSize_ 2000
        $ns_ attach-agent $node_([expr $i + 1]) $udp($k)

        set cbr($k) [new Application/Traffic/CBR]
        $cbr($k) set packetSize_ 2000
        $cbr($k) set interval_ 0.001
        $cbr($k) attach-agent $udp($k)
```

```
        set null($k) [new Agent/Null]
        $ns_ attach-agent $node_($i) $null($k)
        $ns_ connect $udp($k) $null($k)

        incr j
        }
}


set j 3
set k -1
for {set i [expr ($val(nn)-1)]} {$i > [expr ($val(nn)/2 )]} {set i [expr $i - 2]} {
incr k
set temp1 [expr (($val(nn)/2)-$k-1)]
set temp [expr $i - 1]
puts "LOOP i: $i"
if {[expr  ($j)%3 == 0] || [expr  ($j)%3 == 1]} {
puts "creating UDP & CBR $temp1  node $i -> $temp"
        set udp([expr (($val(nn)/2)-$k-1)]) [new Agent/UDP]
        $udp([expr (($val(nn)/2)-$k-1)]) set packetSize_ 2000
        $ns_ attach-agent $node_($i) $udp([expr (($val(nn)/2)-$k-1)])

        set cbr([expr (($val(nn)/2)-$k-1)]) [new Application/Traffic/CBR]
        $cbr([expr (($val(nn)/2)-$k-1)]) set packetSize_ 2000
        $cbr([expr (($val(nn)/2)-$k-1)]) set interval_ 0.001
        $cbr([expr (($val(nn)/2)-$k-1)]) attach-agent $udp([expr (($val(nn)/2)-$k-1)])

        set null([expr (($val(nn)/2)-$k-1)]) [new Agent/Null]
        $ns_ attach-agent $node_([expr $i - 1]) $null([expr (($val(nn)/2)-$k-1)])
        $ns_ connect $udp([expr (($val(nn)/2)-$k-1)]) $null([expr (($val(nn)/2)-$k-1)])

        incr j
} else {
set tmp [expr $i - 1]
set tmp1 [expr (($val(nn)/2)-$k-1)]
puts "creating UDP & CBR $tmp1 node $tmp -> $i"
        set udp([expr (($val(nn)/2)-$k-1)]) [new Agent/UDP]
        $udp([expr (($val(nn)/2)-$k-1)]) set packetSize_ 2000
        $ns_ attach-agent $node_([expr $i - 1]) $udp([expr (($val(nn)/2)-$k-1)])

        set cbr([expr (($val(nn)/2)-$k-1)]) [new Application/Traffic/CBR]
        $cbr([expr (($val(nn)/2)-$k-1)]) set packetSize_ 2000
        $cbr([expr (($val(nn)/2)-$k-1)]) set interval_ 0.001
        $cbr([expr (($val(nn)/2)-$k-1)]) attach-agent $udp([expr (($val(nn)/2)-$k-1)])

        set null([expr (($val(nn)/2)-$k-1)]) [new Agent/Null]
        $ns_ attach-agent $node_($i) $null([expr (($val(nn)/2)-$k-1)])
        $ns_ connect $udp([expr (($val(nn)/2)-$k-1)]) $null([expr (($val(nn)/2)-$k-1)])
```

```
incr j
        }
}


for {set i 0} {$i < ([expr $val(nn)/2]) } {incr i} {
puts "starting cbr $i"
        $ns_ at 11.0 "$cbr($i) start"
        $ns_ at 31.1 "$cbr($i) stop"

        }
#
# Tell nodes when the simulation ends
#
for {set i 0} {$i < $val(nn) } {incr i} {
   $ns_ at 150.0 "$node_($i) reset";
}
#$ns_ at 150.0 "stop"
#$ns_ at 150.01 "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
   global ns_ tracefd
puts "stop1"
   $ns_ flush-trace
   close $tracefd
puts "stop2"
        exit 0
}

puts "Starting Simulation..."
$ns_ run
```

Scenario3CBR.tcl

```
#############################################################################
# Get command line arguments and create trace path
#
# Create $nodes downlink gateways and $connections TCP conn per gateway

set USAGE "ns Scenario3CBR.tcl <nodes> <protocol> <seed>"
if {$argc != 3} {puts $USAGE; exit 0}

set nodes       [lindex $argv 0];
set protocol    [lindex $argv 1];
set seed        [lindex $argv 2];

if {$nodes < 2} {append n 0 $nodes} else {set n $nodes}
if {$protocol < 10} {append p 0 $protocol} else {set p $protocol}
```

```
append scenario "Scenario3CBR/" $n "-nodes3nd/"  $p "-protocol/" $seed "-seed/"
puts "Creating $scenario..."
file mkdir $scenario
set proto "Mac/"
append proto $p


global defaultRNG
$defaultRNG seed $seed


#########################################################################
# ======================================================================
# Define options
# ======================================================================

set val(chan)       Channel/WirelessChannel          ;# channel type
set val(prop)       Propagation/TwoRayGround         ;# radio-propagation model
set val(netif)      Phy/WirelessPhy                  ;# network interface type
set val(mac)        $proto                           ;# MAC type
set val(ifq)        Queue/DropTail/PriQueue          ;# interface queue type
set val(ll)         LL                               ;# link layer type
set val(ant)        Antenna/OmniAntenna              ;# antenna model
set val(ifqlen)     50                               ;# max packet in ifq
set val(nn)         $n                               ;# number of mobilenodes
set val(rp)         DSDV                             ;# routing protocol



# ======================================================================
# Main Program
# ======================================================================

Phy/WirelessPhy set Pt_ 0.2818
add-packet-header SCTP


#
# Initialize Global Variables
#

set ns_           [new Simulator]
set tracefd     [open $scenario/trace.tr w]
set namfd       [open $scenario/trace.nam w]

$ns_ use-newtrace
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namfd 1500 1500
```

```
# set up topography object
set topo       [new Topography]

$topo load_flatgrid 1500 1500

#
# Create God
#
create-god $val(nn)

#
#  Create the specified number of mobilenodes [$val(nn)] and "attach" them
#  to the channel.
#  Here two nodes are created : node(0) and node(1)

# configure node

$ns_ node-config -adhocRouting $val(rp) \
            -llType $val(ll) \
            -macType $val(mac) \
            -ifqType $val(ifq) \
            -ifqLen $val(ifqlen) \
            -antType $val(ant) \
            -propType $val(prop) \
            -phyType $val(netif) \
            -channelType $val(chan) \
            -topoInstance $topo \
            -agentTrace ON \
            -routerTrace OFF \
            -macTrace OFF \
            -movementTrace OFF

set j 0
for {set i 0} {$i <[expr  $val(nn)/4] } {incr i} {
          incr j
          set node_($i) [$ns_ node]
          $node_($i) random-motion 0             ;# disable random motion
          $node_($i) set X_ [expr (10+(200.0 * ($j)))]
          $node_($i) set Y_ 10.0
          $node_($i) set Z_ 0.0

          set node_([expr ($val(nn)/4+($i))]) [$ns_ node]
          $node_([expr ($val(nn)/4+($i))]) random-motion 0              ;# disable random
motion
          $node_([expr ($val(nn)/4+($i))]) set X_ [expr (10+(200.0 * ($j)))]
          $node_([expr ($val(nn)/4+($i))]) set Y_ 210.0
          $node_([expr ($val(nn)/4+($i))]) set Z_ 0.0
```

```
        set node_([expr (2*$val(nn)/4+($i))]) [$ns_ node]
        $node_([expr (2*$val(nn)/4+($i))]) random-motion 0              ;# disable
random motion
        $node_([expr (2*$val(nn)/4+($i))]) set X_ [expr (10+(200.0 * ($j)))]
        $node_([expr (2*$val(nn)/4+($i))]) set Y_ 410.0
        $node_([expr (2*$val(nn)/4+($i))]) set Z_ 0.0

        set node_([expr ($val(nn)/4+($i))]) [$ns_ node]
        $node_([expr (3*$val(nn)/4+($i))]) random-motion 0              ;# disable
random motion
        $node_([expr (3*$val(nn)/4+($i))]) set X_ [expr (10+(200.0 * ($j)))]
        $node_([expr (3*$val(nn)/4+($i))]) set Y_ 610.0
        $node_([expr (3*$val(nn)/4+($i))]) set Z_ 0.0
}



for {set i 0} {$i < $val(nn) } {incr i} {
        set tmpX [$node_($i) set X_]
        set tmpY [$node_($i) set Y_]
        puts "node:$i X:$tmpX Y:$tmpY"
}

#
# Progress report
#
set progress_interval   60.0
set start_time          [clock seconds]
set sim_duration        300.1

proc progress_report {} {
        global  progress_interval       \
                start_time              \
                ns_

        set sim_now [$ns_ now]
        set tmp2 [clock seconds]
        set tmp [expr $tmp2 - $start_time]
        set now [clock format $tmp -format %H:%M:%S -gmt 1]
        set tmp [expr $sim_now / 60.0]
        puts "$now : $tmp \bmin"
        $ns_ at [expr $sim_now + $progress_interval] "progress_report"
}

$ns_ at 0.0 "progress_report"
$ns_ at $sim_duration "stop"

set inter_ 0.001
```

```
set j 3
set k -1
for {set i 0} {$i < $val(nn)} {set i [expr $i + 2]} {
incr k
puts "LOOP i: $i"

if {[expr  ($j)%3 == 0] || [expr  ($j)%3 == 1]} {
puts "creating UDP & CBR $k on node $i"
        set udp($k) [new Agent/UDP]
        $udp($k) set packetSize_ 2000
        $ns_ attach-agent $node_($i) $udp($k)

        set cbr($k) [new Application/Traffic/CBR]
        $cbr($k) set packetSize_ 2000
        $cbr($k) set interval_ 0.001
        $cbr($k) attach-agent $udp($k)

        set null($k) [new Agent/Null]
        $ns_ attach-agent $node_([expr $i + 1]) $null($k)
        $ns_ connect $udp($k) $null($k)

        incr j
} else {
set tmp [expr $i + 1]
puts "creating UDP & CBR $k on node $tmp"
        set udp($k) [new Agent/UDP]
        $udp($k) set packetSize_ 2000
        $ns_ attach-agent $node_([expr $i + 1]) $udp($k)

        set cbr($k) [new Application/Traffic/CBR]
        $cbr($k) set packetSize_ 2000
        $cbr($k) set interval_ 0.001
        $cbr($k) attach-agent $udp($k)

        set null($k) [new Agent/Null]
        $ns_ attach-agent $node_($i) $null($k)
        $ns_ connect $udp($k) $null($k)

        incr j
        }
}


for {set i 0} {$i < ([expr $val(nn)/2]) } {incr i} {
puts "starting cbr $i"
        $ns_ at 11.0 "$cbr($i) start"
        $ns_ at 31.1 "$cbr($i) stop"
        }
```

```
#
# Tell nodes when the simulation ends
#
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at 150.0 "$node_($i) reset";
}
#$ns_ at 150.0 "stop"
#$ns_ at 150.01 "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
    global ns_ tracefd
puts "stop1"
    $ns_ flush-trace
    close $tracefd
puts "stop2"
            exit 0
}
puts "Starting Simulation..."
$ns_ run
```

Scenario3PARETO.tcl

```
#############################################################################
# Get command line arguments and create trace path
#
# Create $nodes downlink gateways and $connections TCP conn per gateway

set USAGE "ns Scenario3PARETO.tcl <nodes> <protocol> <seed>"
if {$argc != 3} {puts $USAGE; exit 0}

set nodes       [lindex $argv 0];
set protocol    [lindex $argv 1];
set seed        [lindex $argv 2];

if {$nodes < 2} {append n 0 $nodes} else {set n $nodes}
if {$protocol < 10} {append p 0 $protocol} else {set p $protocol}

append scenario "Scenario3PARETO/" $n "-nodes/"  $p "-protocol/" $seed "-seed/"
puts "Creating $scenario..."
file mkdir $scenario
set proto "Mac/"
append proto $p

global defaultRNG
$defaultRNG seed $seed
```

```
#######################################################################
# ====================================================================
# Define options
#=====================================================================

set val(chan)       Channel/WirelessChannel          ;# channel type
set val(prop)       Propagation/TwoRayGround         ;# radio-propagation model
set val(netif)      Phy/WirelessPhy                  ;# network interface type
set val(mac)        $proto                           ;# MAC type
set val(ifq)        Queue/DropTail/PriQueue          ;# interface queue type
set val(ll)         LL                               ;# link layer type
set val(ant)        Antenna/OmniAntenna              ;# antenna model
set val(ifqlen)     50                               ;# max packet in ifq
set val(nn)         $n                               ;# number of mobilenodes
set val(rp)         DSDV                             ;# routing protocol
# ====================================================================
# Main Program
# ====================================================================

Phy/WirelessPhy set Pt_ 0.2818
add-packet-header SCTP


#
# Initialize Global Variables
#

set ns_           [new Simulator]
set tracefd     [open $scenario/trace.tr w]
set namfd       [open $scenario/trace.nam w]

$ns_ use-newtrace
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namfd 1500 1500


# set up topography object
set topo      [new Topography]

$topo load_flatgrid 1500 1500

#
# Create God
#
create-god $val(nn)
```

```
#
#  Create the specified number of mobilenodes [$val(nn)] and "attach" them
#  to the channel.
#  Here two nodes are created : node(0) and node(1)

# configure node

$ns_ node-config -adhocRouting $val(rp) \
           -llType $val(ll) \
           -macType $val(mac) \
           -ifqType $val(ifq) \
           -ifqLen $val(ifqlen) \
           -antType $val(ant) \
           -propType $val(prop) \
           -phyType $val(netif) \
           -channelType $val(chan) \
           -topoInstance $topo \
           -agentTrace ON \
           -routerTrace OFF \
           -macTrace OFF \
           -movementTrace OFF

set j 0
for {set i 0} {$i <[expr  $val(nn)/4] } {incr i} {
          incr j
          set node_($i) [$ns_ node]
          $node_($i) random-motion 0             ;# disable random motion
          $node_($i) set X_ [expr (10+(200.0 * ($j)))]
          $node_($i) set Y_ 10.0
          $node_($i) set Z_ 0.0

          set node_([expr ($val(nn)/4+($i))]) [$ns_ node]
          $node_([expr ($val(nn)/4+($i))]) random-motion 0              ;# disable random
motion
          $node_([expr ($val(nn)/4+($i))]) set X_ [expr (10+(200.0 * ($j)))]
          $node_([expr ($val(nn)/4+($i))]) set Y_ 210.0
          $node_([expr ($val(nn)/4+($i))]) set Z_ 0.0

          set node_([expr (2*$val(nn)/4+($i))]) [$ns_ node]
          $node_([expr (2*$val(nn)/4+($i))]) random-motion 0                    ;# disable
random motion
          $node_([expr (2*$val(nn)/4+($i))]) set X_ [expr (10+(200.0 * ($j)))]
          $node_([expr (2*$val(nn)/4+($i))]) set Y_ 410.0
          $node_([expr (2*$val(nn)/4+($i))]) set Z_ 0.0
```

```
        set node_([expr ($val(nn)/4+($i))]) [$ns_ node]
        $node_([expr (3*$val(nn)/4+($i))]) random-motion 0              ;# disable
random motion
        $node_([expr (3*$val(nn)/4+($i))]) set X_ [expr (10+(200.0 * ($j)))]
        $node_([expr (3*$val(nn)/4+($i))]) set Y_ 610.0
        $node_([expr (3*$val(nn)/4+($i))]) set Z_ 0.0
}



for {set i 0} {$i < $val(nn) } {incr i} {
        set tmpX [$node_($i) set X_]
        set tmpY [$node_($i) set Y_]
        puts "node:$i X:$tmpX Y:$tmpY"
}

#
# Progress report
#
set progress_interval   60.0
set start_time          [clock seconds]
set sim_duration        300.1

proc progress_report {} {
        global  progress_interval       \
                start_time              \
                ns_

        set sim_now [$ns_ now]
        set tmp2 [clock seconds]
        set tmp [expr $tmp2 - $start_time]
        set now [clock format $tmp -format %H:%M:%S -gmt 1]
        set tmp [expr $sim_now / 60.0]
        puts "$now : $tmp \bmin"
        $ns_ at [expr $sim_now + $progress_interval] "progress_report"
}

$ns_ at 0.0 "progress_report"
$ns_ at $sim_duration "stop"



set pk_size     2000
set rate 65000
set burst       1.9
set idle 100ms
set shape       100ms
```

```
set j 3
set k -1
for {set i 0} {$i < $val(nn)} {set i [expr $i + 2]} {
incr k
puts "LOOP i: $i"

if {[expr  ($j)%3 == 0] || [expr  ($j)%3 == 1]} {
puts "creating UDP & CBR $k on node $i"
        set udp($k) [new Agent/UDP]
        $udp($k) set packetSize_ 2000
        $ns_ attach-agent $node_($i) $udp($k)


        set so_udp_pareto($k) [new Application/Traffic/Pareto]
        $so_udp_pareto($k) set packetSize_ $pk_size
        $so_udp_pareto($k) set rate_ $rate
        $so_udp_pareto($k) set burst_time_ $burst
        $so_udp_pareto($k) set idle_time_ $idle
        $so_udp_pareto($k) set shape_ $shape
        $so_udp_pareto($k) attach-agent $udp($k)

        set null($k) [new Agent/Null]
        $ns_ attach-agent $node_([expr $i + 1]) $null($k)
        $ns_ connect $udp($k) $null($k)

        incr j
} else {
set tmp [expr $i + 1]
puts "creating UDP & CBR $k on node $tmp"
        set udp($k) [new Agent/UDP]
        $udp($k) set packetSize_ 2000
        $ns_ attach-agent $node_([expr $i + 1]) $udp($k)

set so_udp_pareto($k) [new Application/Traffic/Pareto]
        $so_udp_pareto($k) set packetSize_ $pk_size
        $so_udp_pareto($k) set rate_ $rate
        $so_udp_pareto($k) set burst_time_ $burst
        $so_udp_pareto($k) set idle_time_ $idle
        $so_udp_pareto($k) set shape_ $shape
        $so_udp_pareto($k) attach-agent $udp($k)

        set null($k) [new Agent/Null]
        $ns_ attach-agent $node_($i) $null($k)
        $ns_ connect $udp($k) $null($k)

        incr j
        }
}
```

```
for {set i 0} {$i < ([expr $val(nn)/2]) } {incr i} {
puts "starting cbr $i"
        $ns_ at 11.0 "$so_udp_pareto($i) start"
        $ns_ at 31.1 "$so_udp_pareto($i) stop"

        }
#
# Tell nodes when the simulation ends
#
for {set i 0} {$i < $val(nn) } {incr i} {
   $ns_ at 150.0 "$node_($i) reset";
}
#$ns_ at 150.0 "stop"
#$ns_ at 150.01 "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
   global ns_ tracefd
puts "stop1"
   $ns_ flush-trace
   close $tracefd
puts "stop2"
        exit 0
}

puts "Starting Simulation..."
$ns_ run
```

ΠΑΡΑΠΟΜΠΕΣ

[1] “MACA-P: A MAC Protocol to Improve Parallelism in Multi-Hop Wireless Networks”, IBM Research Report RC 22325, September 2002.
[2] A. Acharya, A. Mishra, and S. Bansal. “Supporting Concurrent Transmissions in Multihop Wireless Networks”, at 2nd NY Metro Area Networking Workshop, Columbia
University, Sept2002.
[3] S. Xu, and T. Saadawi, “Does the IEEE 802.11 MAC Protocol Work Well in Multihop Wireless Ad Hoc Networks?”, IEEE Communication Magazine June 2001.
[4] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification, ANSI/IEEE Std 802.11, 1999 Edition
[5] Arup Acharya, Archan Misra and Sorav Bansal, “MACA-P: A MAC for Concurrent Transmissions in Multi-hop Wireless Networks”, IEEE Percom 2003
[6] Arup Acharya and Sorav Bansal, “Design and Analysis of o Cooperative Medium Access Scheme for Wireless Mesh Networks”
[7] Arup Acharya, Archan Misra and Sorav Bansal, “Challenges in high performance data forwarding in multi-hop wireless networks”, RC22506 (W0207-012) July 2002
[8] “The network simulator”, available at ***http://www.isi.edu/nsnam/ns***
[9] “Oreilly - 802.11 Wireless Networks The Definitive Guide”