Alexander T.E.I. of Thessaloniki
School of Technological Applications
Department of Informatics

Μελέτη απόδοσης ασύρματων δικτύων IEEE 802.11e με το λογισμικό προσομοίωσης Network Simulator 3 (ns-3)

Πτυχιακή Εργασία

Του

Κωνσταντίνου Θεοδοσιάδη

Επιβλέπων Καθηγητής
Δρ. Περικλής Χατζημίσιος
Αναπληρωτής Καθηγητής

Αλεξάνδρειο Τ.Ε.Ι. Θεσσαλονίκης
Τμήμα Μηχανικών Πληροφορικής
Τ.Θ. 141, 574 00, Θεσσαλονίκη

Μάιος 2014

ΠΡΟΛΟΓΟΣ

Το IEEE 802.11e είναι μία αναθεώρηση του αρχικού προτύπου 802.11 του οργανισμού IEEE (Institute of Electrical and Electronics Engineers), που καθορίζει ένα σύνολο από βελτιώσεις στον τομέα της ποιότητας υπηρεσίας (Quality of Service - QoS) στα ασύρματα τοπικά δίκτυα, μέσω τροποποιήσεων στην λειτουργία του επιπέδου MAC (Medium Access Control). Η παραπάνω αναβάθμιση θεωρείται ιδιαίτερα σημαντική για τις εφαρμογές που είναι ευαίσθητες στις καθυστερήσεις (delay), όπως οι εφαρμογές μετάδοσης φωνής (Voice over WLAN) και ροής πολυμέσων (streaming multimedia). [1]

Το λογισμικό προσομοίωσης δικτύων ns-3 (Network Simulator 3), είναι ένας προσομοιωτής διακριτών γεγονότων, ο οποίος απευθύνεται κυρίως στην έρευνα και στην εκπαίδευση. Είναι ελεύθερο λογισμικό, το οποίο διατίθεται κάτω από την άδεια χρήσης GNU GPLv2, για έρευνα, ανάπτυξη και χρήση. Κατά την ανάπτυξη του ns-3, εγκαταλείφθηκε πλήρως η προς τα πίσω συμβατότητα με το προϋπάρχον ns-2 και ο προσομοιωτής γράφηκε από την αρχή, χρησιμοποιώντας την γλώσσα C++. Η ανάπτυξή του ξεκίνησε τον Ιούλιο του 2006 και η πρώτη διανομή, η ns-3.1, έγινε διαθέσιμη τον Ιούνιο του 2008, με την τρέχουσα διανομή να είναι η ns-3.19, η οποία έγινε διαθέσιμη τον Δεκέμβριο του 2013. [2]

Η παρούσα πτυχιακή εργασία εκπονήθηκε κάνοντας χρήση της διανομής ns-3.17.

---

1 IEEE 802.11e, http://en.wikipedia.org/wiki/IEEE_802.11e-2005

2 ns (simulator), http://en.wikipedia.org/wiki/Ns-3

ΠΕΡΙΛΗΨΗ

Η απαίτηση για παροχή ποιότητας υπηρεσίας (QoS) στα ασύρματα τοπικά δίκτυα προς υποστήριξη των εφαρμογών χαμηλής ανοχής στις καθυστερήσεις του δικτύου, όπως αυτές της μετάδοσης φωνής ή βίντεο, οδήγησε στην αναθεώρηση του αρχικού προτύπου και την έκδοση του IEEE 802.11e.

Το ανοιχτό λογισμικό προσομοίωσης δικτύων ns-3, από την διανομή ns-3.15 και έπειτα, υποστηρίζει την προσομοίωση των βασικών λειτουργιών επιπέδου Medium Access Control (MAC) του IEEE 802.11e.

Στην πτυχιακή εργασία αρχικά θα πραγματοποιηθεί μια λεπτομερής εισαγωγή στη λειτουργία του επιπέδου MAC στο πρωτόκολλο IEEE 802.11 καθώς και σε αυτή που προβλέπεται στην αναθεωρημένο πρότυπο IEEE 802.11e.

Επίσης, θα πραγματοποιηθεί μία εισαγωγή στο λογισμικό προσομοίωσης ns-3 και κυρίως στο μοντέλο προσομοίωσης ασύρματων τοπικών δικτύων IEEE 802.11.

Τέλος, χρησιμοποιώντας το λογισμικό ns-3, θα πραγματοποιηθεί προσομοίωση ασύρματου τοπικού δικτύου IEEE 802.11e καθώς και προσομοίωση ασύρματου τοπικού δικτύου IEEE 802.11 χωρίς υποστήριξη QoS, ακολουθούμενη από συγκριτική μελέτη των αποτελεσμάτων για τις μετρικές απόδοσης, της καθυστέρησης, μέσης καθυστέρησης, μεταβλητότητας της καθυστέρησης και απώλειας πακέτων (delay, mean delay, jitter και packet loss).

ABSTRACT

The demand for Quality of Service (QoS) in Wireless Local Area Networks (WLANs) to support applications with low tolerance to network delay, such as the transmission of voice or video, led to the IEEE 802.11e amendment.

The open source network simulation software ns-3, since distribution ns-3.15 and later, supports the simulation of the basic functions of the IEEE 802.11e Medium Access Control (MAC) layer.

In the current thesis we carry out a detailed introduction about the operation of the IEEE 802.11 MAC layer as well as for the IEEE 802.11e amendment. Furthermore, we provide an introduction to the ns-3 simulation software, discussing mainly the simulation model for IEEE 802.11 WLANs.

Finally, by employing the software ns-3, we carry out simulations for IEEE 802.11e as well as for IEEE 802.11 WLANs without considering QoS, followed by a comparative study and analysis of the results for the performance metrics of delay, mean delay, jitter and packet loss.

ΕΥΡΕΤΗΡΙΟ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΡΟΛΟΓΟΣ 2

ΠΕΡΙΛΗΨΗ 3

ABSTRACT 4

ΕΥΡΕΤΗΡΙΟ ΠΕΡΙΕΧΟΜΕΝΩΝ 5

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ & ΕΙΚΟΝΩΝ 8

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ 9

ΕΥΡΕΤΗΡΙΟ ΣΥΝΤΟΜΟΓΡΑΦΙΩΝ 10

ΚΕΦΑΛΑΙΟ 1 ΤΟ ΕΠΙΠΕΔΟ MAC ΣΤΟ IEEE 802.11 13

1.1 Λειτουργίες διαχείρισης σύνδεσης στο BSS 15

1.1.1 Το πλαίσιο φάρου (Beacon) 15

1.1.2 Η διαδικασία σάρωσης 16

1.1.3 Η διαδικασία ταυτοποίησης 17

1.1.4 Η διαδικασία σύνδεσης 17

1.1.5 Η διαδικασία επανασύνδεσης 18

1.1.6 Η αποσύνδεση 18

1.2 Η κατανεμημένη διαδικασία πρόσβασης στο κανάλι (DCF) 18

1.2.1 Ο Βασικός χρονισμός της DCF 20

1.2.1.1 SIFS (Short Inter-Frame space) 21

1.2.1.2 Χρόνος θυρίδας (Slot time) 21

1.2.1.3 PIFS (Point Coordination Function Inter-Frame Space) 22

1.2.1.4 DIFS (DCF Inter-Frame Space) 22

1.2.1.5 Ο τυχαίος χρόνος οπισθοχώρησης (Random Backoff time) 23

1.2.2 Η διαδικασία της οπισθοχώρησης 23

1.3 Ανταλλαγή πλαισίων DATA/ACK 24

1.3.1 Κατακερματισμός (Fragmentation) 25

1.3.2 Ανίχνευση διπλοτύπων πλαισίων 26
1.3.3 Δικαιοσύνη στην πρόσβαση στο μέσο 27
1.4 Το πρόβλημα του κρυφού κόμβου 28
1.4.1 Το Διάνυσμα Κατανομής Δικτύου (Network Allocation Vector - NAV) 29
1.4.2 Ανταλλαγή πλαισίων RTS/CTS 29
1.4.3 EIFS (Extended Inter-Frame Space) 30
1.5 Η Βελτιωμένη κατανεμημένη πρόσβαση καναλιού (EDCA) 31
1.5.1 Η έννοια της Ευκαιρίας εκπομπής 33
1.5.2 Ο χρονισμός πρόσβασης στην EDCA 33
1.5.3 Παράμετροι πρόσβασης της EDCA 34
1.5.4 Η αναθεώρηση του EIFS 35
1.5.5 Ανίχνευση σύγκρουσης 36
1.5.6 Το πλαίσιο δεδομένων QoS 36
1.6 Το πρωτόκολλο ομαδικής επιβεβαίωσης (Block ACK) 37
1.7 Ανταλλαγή ομάδας πλαισίων δεδομένων 40
ΚΕΦΑΛΑΙΟ 2 Ο ΠΡΟΣΟΜΟΙΩΤΗΣ ns-3 42
2.1 Τι είναι το ns-3 42
2.2 Οι τεχνολογίες του ns-3 42
2.2.1 Περιτυλίγματα Python 42
2.2.2 Σχεδιασμός υψηλού επιπέδου 43
2.3 Το βασικό μοντέλο δικτύου 46
2.3.1 Κόμβος 46
2.3.2 Εφαρμογή 46
2.3.3 Κανάλι 47
2.3.4 Συσκευή δικτύου 48
2.4 Οι κλάσεις βοηθοί 49
2.5 Το 802.11 WiFi στο ns-3 50

2.5.1 Επισκόπηση του μοντέλου 50
2.5.2 Η διαδικασία εκπομπής και λήψης πλαισίων 53
ΚΕΦΑΛΑΙΟ 3 Η ΠΡΟΣΟΜΟΙΩΣΗ 59
3.1 Το δίκτυο που προσομοιώνεται 59
3.2 Δομή του σεναρίου προσομοίωσης (ns-3 script) 60
3.3 Τροποποίηση της κλάσης ns3::OnOffApplication. 69
3.4 Εκτέλεση της προσομοίωσης 70
ΚΕΦΑΛΑΙΟ 4 ΑΠΟΤΕΛΕΣΜΑΤΑ ΤΗΣ ΠΡΟΣΟΜΟΙΩΣΗΣ 71
4.1 Αποτελέσματα για τη Μέση Απόδοση (Average Throughput) 71
4.2 Αποτελέσματα για την Απώλεια Πακέτων (Packets Lost) 72
4.3 Αποτελέσματα για τη Μέση Καθυστέρηση (Mean Delay) 73
4.4 Αποτελέσματα για τη Διακύμανση της καθυστέρησης (Jitter) 74
ΚΕΦΑΛΑΙΟ 5 ΣΥΜΠΕΡΑΣΜΑΤΑ 76
ΒΙΒΛΙΟΓΡΑΦΙΑ 77
ΠΑΡΑΡΤΗΜΑ 1 – ΚΩΔΙΚΑΣ 78
Α) Το σενάριο προσομοίωσης – QoSSim.cc 78
Β) Η κλάση OnOffApplication – OnOffApplication.cc 86
Γ) Το αρχείο header – OnOffApplication.h 97
Δ) Το αρχείο WSCRIPT 101
ΠΑΡΑΡΤΗΜΑ 2 – ΑΡΧΕΙΑ ΑΠΟΤΕΛΕΣΜΑΤΩΝ 103
Α) Περιεχόμενα αρχείου RESULTS-BE-35.dat 103
Β) Περιεχόμενα αρχείου RESULTS-VO-35.dat 104

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ & ΕΙΚΟΝΩΝ

Εικόνα 1 - IEEE 802.11 [1] 15

Εικόνα 2 – Inter Frame Spaces [9]..... 21

Εικόνα 3 – Η διαδικασία τυχαίας οπισθοχώρησης (backoff procedure) [1] 24

Εικόνα 4 - Το πρόβλημα του κρυφού κόμβου..... 28

Εικόνα 5 Μετάδοση με χρήση του μηχανισμού RTS/CTS [9]..... 30

Εικόνα 6 - Οι κατηγορίες πρόσβασης στο κανάλι και ο σχετικός χρονισμός τους [1]..... 34

Εικόνα 7 - Η διαδικασία ομαδικής ανταλλαγής πλαισίων [1]..... 41

Εικόνα 8 - Η βασική αρχιτεκτονική του NS-3 [2] 43

Εικόνα 9 - Η εσωτερική αρχιτεκτονική του ns-3 [2] 44

Εικόνα 10 – Το μοντέλο σύνδεσης καναλιού – συσκευών [6] 47

Εικόνα 11 - Το βασικό μοντέλο δικτύου του ns-3 [6]..... 49

Εικόνα 12 - Η διαδικασία εκπομπής/λήψης πακέτων [3] 54

Εικόνα 13 - Το βασικό δίκτυο της προσομοίωσης..... 59

Εικόνα 14 – Διάγραμμα Μέσης Απόδοσης (Avg Throughput)..... 72

Εικόνα 15 - Διάγραμμα Απολεσθέντων Πακέτων (Packets Lost) 73

Εικόνα 16 - Διάγραμμα Μέσης Καθυστέρησης (Mean Delay)..... 74

Εικόνα 17 - Διάγραμμα Διακύμανσης της Καθυστέρησης (Jitter)..... 75

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

Πίνακας 1 – Οι κατηγορίες προτεραιότητας (ACs) [1] 32

Πίνακας 2 - Οι προκαθορισμένες παράμετροι πρόσβασης της EDCA για τα PHY των 802.11a, 802.11g και 802.11n [1] 35

Πίνακας 3 - Η βασική διαδικασία εκπομπής [2]..... 56

Πίνακας 4 - Η βασική διαδικασία λήψης..... 58

ΕΥΡΕΤΗΡΙΟ ΣΥΝΤΟΜΟΓΡΑΦΙΩΝ

AARF	Adaptive Auto Rate Fallback
AC	Access Category
AC_BE	Access Category Best Effort
AC_BK	Access Category Background
AC_VI	Access Category Video
AC_VO	Access Category Voice
ACK	Acknowledgement
ADDBA	Add Block ACK
AIFS	Arbitration Inter-frame Space
AIFSN	Arbitration Inter-frame Space Number
AP	Access point
API	Application Programming Interface
ARF	Auto Rate Fallback
ASCII	American Standard Code for Information Interchange
BA	Block ACK
BAR	Block ACK Request
BSS	Basic Service Set
BSSID	Basic Service set Identifier
CARA	Collision Aware Rate Adaption
CBR	Constant Bit Rate
CCA	Clear Chanel Assessment
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection

CTS	Clear To Send
CW	Contention Window
DA	Destination Address
DCE	Distributed Coordination Function
DCF	Distributed Coordination Function
DELBA	Delete Block ACK
DIFS	DCF Inter-frame Space
DS	Distribution system
EDCA	Enhanced Distributed Coordination Function
EIFS	Extended Interface Space
ESS	Extended Service Set
FCS	Frame Control Sequence
GNU	GNU's Not Unix
GPL	General public License
GTK	GIMP Toolkit
IBSS	Independent Basic Service set
IDE	Integrated Development Environment
IEEE	Institute of Electrical & Electronic Engineers
IFS	Inter-frame Space
IP	Internet Protocol
LLC	Logical Link Control
MAC	Medium Access Control
MPDU	MAC Protocol Data Unit
MSDU	MAC Service Data Unit
NAV	Network Allocation vector
NIC	Network Interface Card

NS	Network Simulator
PHY	Physical Layer
PIFS	Point Coordination Inter-frame Space
POSIX	Portable Operating System Interface for Unix
PPDU	Physical Protocol Data Unit
QoS	Quality of Service
RRAA	Robust Rate Adaption Algorithm
RTS	Request To Send
Rx	Receive
SIFS	Short Inter-frame Space
SSC	Starting Sequence Control
SSID	Service set Identifier
STA	Station
TBTT	Target Beacon Transmission Time
TC	Traffic Category
TID	Traffic Identifier
Tx	Transmit
TXOP	Transmit Opportunity
WEP	Wireless Equivalent Privacy
WiFi	Wireless Fidelity
WLAN	Wireless Local Area Network
WMM	Wireless Multimedia Extensions
XML	Extensible Markup Language
YANS	Yet Another Network Simulator

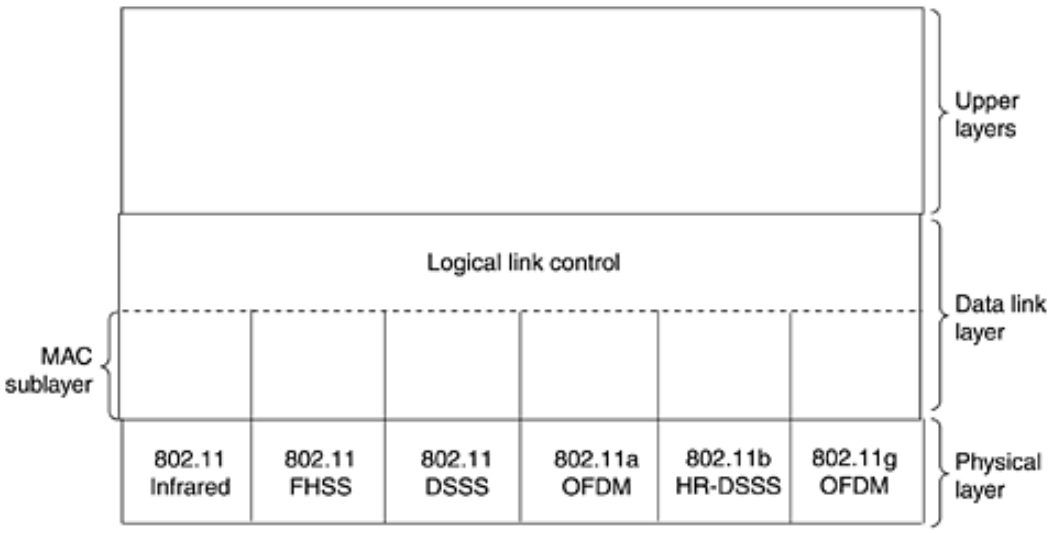
ΚΕΦΑΛΑΙΟ 1 ΤΟ ΕΠΙΠΕΔΟ MAC ΣΤΟ IEEE 802.11

Το επίπεδο ελέγχου πρόσβασης στο μέσο (Medium Access Control - MAC) παρέχει, μεταξύ άλλων, τον έλεγχο πρόσβασης στο κανάλι και καθιστά δυνατή την επικοινωνία σε ένα δίκτυο με μεγάλο αριθμό σταθμών. Το πρωτόκολλο IEEE 802.11, συχνά αναφέρεται και ως ασύρματο Ethernet και από την άποψη της πρόσβασης στο κανάλι, το 802.11 είναι πράγματι παρόμοιο με Ethernet, το οποίο τυποποιήθηκε ως IEEE 802.3. Ως μέλος της IEEE 802 οικογένειας τοπικών δικτύων, το 802.11 κάνει χρήση του 48-bit καθολικού χώρου διευθύνσεων του IEEE 802, γεγονός που το καθιστά συμβατό με το Ethernet στο επίπεδο ζεύξης (LLC). Το MAC του 802.11 επίσης υποστηρίζει την κοινόχρηστη πρόσβαση στο ασύρματο μέσο, μέσω μιας τεχνικής που ονομάζεται CSMA/CA (Carrier Sense Multiple Access / Collision Avoidance), η οποία είναι παρόμοια με την αρχική CSMA/CD (Carrier Sense Multiple Access / Collision Detection) του Ethernet. Και με τις δύο τεχνικές, εάν το κανάλι ανιχνεύεται ότι είναι ελεύθερο, ο σταθμός επιτρέπεται να εκπέμψει, αλλά αν το κανάλι ανιχνεύεται ότι είναι κατειλημμένο, τότε ο σταθμός αναβάλλει την μετάδοση. Ωστόσο, τα πολύ διαφορετικά φυσικά μέσα στα οποία λειτουργούν τα Ethernet και 802.11 επιβάλουν κάποιες σημαντικές διαφοροποιήσεις.

Το πρωτόκολλο πρόσβασης του Ethernet στο κανάλι, βασίζεται στο γεγονός ότι ο σταθμός πριν αρχίσει να μεταδίδει περιμένει να γίνει ελεύθερο το μέσο και εάν ανιχνευθεί μια σύγκρουση κατά τη διάρκεια εκπομπής, θα σταματήσει να εκπέμπει, για να ξαναρχίσει μετά από μια τυχαία χρονική περίοδο αναμονής. Δεν είναι όμως εφικτό για έναν πομπό να ανιχνεύσει μια σύγκρουση κατά τη μετάδοση σε ένα ασύρματο μέσο, οπότε στο IEEE 802.11 απαιτείται να αποφεύγονται οι συγκρούσεις. Μόλις το μέσο ελευθερωθεί, ο σταθμός περιμένει για μία τυχαία χρονική περίοδο, κατά την οποία συνεχίζει να «ακούει» το μέσο, και αν στο τέλος της περιόδου αυτής το μέσο εξακολουθεί να είναι ελεύθερο, ξεκινά τη μετάδοση. Η τυχαία χρονική περίοδος μειώνει τις πιθανότητες μιας σύγκρουσης, λόγω του ότι οι άλλοι σταθμοί που περιμένουν για να έχουν πρόσβαση στο μέσο, πιθανότατα θα έχουν επιλέξει διαφορετικό χρονικό διάστημα αναμονής. Σε αυτό συνίσταται η πτυχή της αποφυγής σύγκρουσης της μεθόδου CSMA/CA.

Το απλό αυτό κατανεμημένο, και ανταγωνιστικής φύσης, πρωτόκολλο πρόσβασης στο μέσο, που υποστηρίζεται από την τεχνική CSMA/CA, είναι η βάση για το MAC του πρωτοκόλλου 802.11 και επίσης το σημείο όπου η ομοιότητα με το Ethernet τελειώνει. Το ασύρματο μέσο, είναι πολύ διαφορετικό από το ενσύρματο και απαιτεί μια σειρά από πρόσθετα χαρακτηριστικά:

- Το ασύρματο μέσο είναι επιρρεπές σε σφάλματα και ωφελείται σημαντικά από ένα χαμηλής καθυστέρησης μηχανισμό διόρθωσης σφαλμάτων, στο επίπεδο ζεύξης (Logical Link Control - LLC).
- Σε ένα ασύρματο μέσο όλοι οι σταθμοί δεν μπορούν να «ακούσουν» όλους τους άλλους σταθμούς. Ορισμένοι σταθμοί μπορεί να μπορούν να «ακούνε» το σταθμό που βρίσκεται στο ένα άκρο της ανταλλαγής δεδομένων, αλλά δεν μπορούν να «ακούσουν» το σταθμό που βρίσκεται στο άλλο απομακρυσμένο άκρο (αυτό είναι το πρόβλημα του κρυμμένου κόμβου).
- Ο ρυθμός μετάδοσης δεδομένων που μπορεί να υποστηρίξει ένα κανάλι επηρεάζεται σε μεγάλο βαθμό από την απόσταση και από τις περιβαλλοντικές συνθήκες. Επίσης, οι συνθήκες στο κανάλι μπορεί να αλλάξουν με το χρόνο λόγω της κινητικότητάς του σταθμού ή μεταβολές στο περιβάλλον. Οι σταθμοί πρέπει να προσαρμόζουν συνεχώς το ρυθμό μετάδοσης, με τον οποίο ανταλλάσσουν πληροφορίες για τη βελτιστοποίηση της απόδοσης.
- Οι σταθμοί, οι οποίοι συχνά είναι κινητοί, χρειάζονται ένα μηχανισμό για τη διαχείριση της σύνδεσης και την αποσύνδεσης από ένα WLAN καθώς αυτοί αλλάζουν θέση.



Εικόνα 1 - IEEE 802.11 [1]

1.1 Λειτουργίες διαχείρισης σύνδεσης στο BSS

Ένας σταθμός μπορεί να αντιληφθεί την ύπαρξη ενός Basic Service Set (BSS) μέσω σάρωσης, που είναι η παθητική αναζήτηση των εκπομπών φάρου (Beacon). Επίσης, μπορεί να ερευνήσει ενεργά για την ύπαρξη ενός Access Point (AP) μέσω ενός μηχανισμού ανταλλαγής Probe Request/Probe Response.

Η ένταξη ενός σταθμού σε ένα BSS είναι δυναμική. Ο σταθμός μπορεί να τεθεί εντός/εκτός λειτουργίας ή μπορεί να κινείται μέσα ή έξω από την περιοχή που καλύπτεται από το BSS. Ένας σταθμός γίνεται μέλος ενός BSS με το να συσχετιστεί-συνδεθεί (Assosiation) με το BSS. Με την έξοδο από το BSS, ο σταθμός αποσυσχετίζεται-αποσυνδέεται (disassociation) από αυτό. Σε ένα Extended Service Set (ESS), που αποτελείται από πολλαπλά BSS , ένας σταθμός μπορεί να μεταναστεύσει από ένα BSS σε ένα άλλο BSS, με τη διαδικασία της επανασυσχέτισης-επανασύνδεσης (Reassosiation).

1.1.1 Το πλαίσιο φάρου (Beacon)

Το Access Point σε ένα infrastructure BSS μεταδίδει περιοδικά beacon πλαίσια. Η περίοδος των Beacon ορίζει ένα σταθερό χρονοδιάγραμμα των επιδιωκόμενων χρονικών στιγμών εκπομπής των Beacon (Target Beacon Transmission Time - TBTT) και το ίδιο το πλαίσιο Beacon μεταδίδεται, όσο το δυνατόν κοντά ή πάνω στο TBTT, εφόσον το μέσο είναι ελεύθερο. Το πλαίσιο Beacon φέρει πληροφορίες

ρυθμίσεων, πληροφορίες για την ικανότητα των συσκευών και πληροφορίες για τη διαχείριση του BSS.

1.1.2 Η διαδικασία σάρωσης

Η σάρωση είναι η διαδικασία με την οποία ένας σταθμός ανακαλύπτει ένα BSS και γνωρίζει τα χαρακτηριστικά που έχει το BSS αυτό. Οι δυνατοί τύποι σάρωσης είναι δύο: η παθητική και η ενεργητική σάρωση.

Η παθητική σάρωση είναι μια διαδικασία που περιλαμβάνει μόνο λήψη και που είναι συμβατή με όλα τα κατά τόπους πλαίσια κανονισμών. Με την παθητική σάρωση, ο σταθμός αναζητά εκπομπές Beacon και μπορεί να αλλάζει κανάλια για να βρει αυτές τις εκπομπές. Τα πλαίσια Beacon περιλαμβάνουν, μεταξύ άλλων, πληροφορίες σχετικά με τον κωδικό της χώρας, τη μέγιστη επιτρεπόμενη ισχύ εκπομπής, και τα κανάλια που θα χρησιμοποιηθούν σύμφωνα με το κατά τόπους ισχύον πλαίσιο κανονισμών. Όταν ο σταθμός ανακαλύψει το AP μέσω των εκπομπών Beacon και αποκτήσει τις κανονιστικές πληροφορίες που αυτό περιλαμβάνει, μπορεί να ρωτήσει άμεσα το AP για πρόσθετες πληροφορίες, ξεκινώντας μια ανταλλαγή Probe Request/Probe Response, εάν οι αναγκαίες επιπλέον πληροφορίες δεν περιλαμβάνονται στο ίδιο το πλαίσιο Beacon.

Η ενεργητική σάρωση μπορεί να χρησιμοποιηθεί όταν αυτό επιτρέπεται από το κανονιστικό πλαίσιο στο οποίο ο σταθμός λειτουργεί. Με την ενεργητική σάρωση ο σταθμός εκπέμπει πλαίσια Probe Request, σε κάθε ένα από τα κανάλια, στα οποία αναζητεί ένα BSS. Ανάλογα με τη γενικότητα της αναζήτησης, το πλαίσιο Probe Request περιλαμβάνει τις ακόλουθες πληροφορίες διευθυνσιοδότησης:

- **SSID** (Service Set Identifier): Το SSID στο Probe Request μπορεί να είναι το SSID του συγκεκριμένου ESS για το οποίο ο σταθμός αναζητεί ένα BSS ή μπορεί να είναι το SSID μπαλαντέρ.
- **BSSID** (BSS Identifier): Το BSSID στο Probe Request μπορεί να είναι το BSSID ενός συγκεκριμένου BSS ή μπορεί να είναι το BSSID μπαλαντέρ.
- **DA** (Destination Address): Η διεύθυνση προορισμού του πλαισίου Probe Request είναι η διεύθυνση broadcast ή η συγκεκριμένη διεύθυνση MAC του ζητούμενου AP.

Ένα AP, το οποίο δέχεται ένα broadcast αίτημα Probe Request, στέλνει μια απάντηση Probe Response στο σταθμό υποβάλλει την αίτηση εάν ισχύουν και οι δύο ακόλουθες συνθήκες:

A. Το SSID είναι το SSID μπαλαντέρ ή αυτό ταιριάζει με το SSID του ESS στο οποίο το AP ανήκει.

B. Το BSSID είναι το BSSID μπαλαντέρ ή το BSSID του AP.

Περισσότερα από ένα διαφορετικά AP μπορεί να ανταποκριθούν στο αίτημα Probe Request, χρησιμοποιώντας την κανονική διαδικασία πρόσβασης στο κανάλι για την αποφυγή συγκρούσεων.

1.1.3 Η διαδικασία ταυτοποίησης

Η ταυτοποίηση (Authentication) είναι η διαδικασία με την οποία δύο σταθμοί που επιθυμούν να επικοινωνήσουν, πιστοποιούν την ταυτότητά τους σε ένα αμοιβαία αποδεκτό επίπεδο. Το αρχικό IEEE 802.11 υποστηρίζει δύο μεθόδους ελέγχου ταυτότητας που λειτουργούν στο επίπεδο ζεύξης δεδομένων, την ταυτοποίηση ανοικτού συστήματος και την ταυτοποίηση μέσω κοινόχρηστου κλειδιού. Με την πρώτη μέθοδο, κάθε σταθμός μπορεί να γίνει δεκτός ως μέλος σε ένα BSS. Με την δεύτερη, οι σταθμοί μέσω του πρωτοκόλλου WEP (Wired Equivalent Privacy) καλούνται να επιδείξουν τη γνώση ενός κοινού κλειδιού κρυπτογράφησης. Το πρωτόκολλο WEP αποδείχτηκε ανασφαλές και αντικαταστάθηκε από νεότερες τεχνικές, με το πρότυπο IEEE 802.11X.

1.1.4 Η διαδικασία σύνδεσης

Πριν επιτραπεί σε ένα σταθμό να στείλει δεδομένα μέσω ενός AP, πρέπει να συνδεθεί σε αυτό το AP. Η σύνδεση αυτή παρέχει μια αντιστοίχιση μεταξύ του σταθμού και του AP που επιτρέπει στα μηνύματα εντός του DS (Distribution System) να φτάσουν στο κατάλληλο AP, στο οποίο συνδέεται ο σταθμός και τελικά στον ίδιο το σταθμό. Σε κάθε δεδομένη χρονική στιγμή, ένας σταθμός μπορεί να συνδέεται με ένα μόνο AP.

Η σύνδεση προκαλείται από τον σταθμό, με την αποστολή μιας αίτησης σύνδεσης (Association Request) προς το AP. Εάν ο σταθμός γίνει δεκτός, το AP απαντά με ένα πλαίσιο Association Response. Με την ανταλλαγή αίτησης/απάντησης σύνδεσης, ο σταθμός και το AP ανταλλάσουν πληροφορίες ικανότητας

υποστήριξης για διάφορες προαιρετικές λειτουργίες και το AP ενημερώνει το σταθμό για συγκεκριμένες παραμέτρους λειτουργίας εντός του BSS.

1.1.5 Η διαδικασία επανασύνδεσης

Η λειτουργία της επανασύνδεσης υποστηρίζει την κινητικότητα αλλαγής BSS, επιτρέποντας σε ένα σταθμό να περάσει από μια τρέχουσα σύνδεση με ένα AP σε ένα άλλο AP μέσα στο ίδιο ESS. Αυτή η λειτουργία κρατά το κεντρικό σύστημα διανομής (Distribution System - DS) ενήμερο για την τρέχουσα αντιστοίχιση μεταξύ AP και σταθμών. Η επανασύνδεση μπορεί επίσης να εκτελείται και για να γίνει αλλαγή στις ιδιότητες της σύνδεσης του σταθμού, όπως στις πληροφορίες ικανότητας υποστήριξης λειτουργιών.

Η επανασύνδεση προκαλείται από το σταθμό, με την αποστολή ενός πλαισίου αιτήματος επανασύνδεσης (Re-association Request) προς το AP. Το AP απαντά με ένα πλαίσιο Re-association Response.

1.1.6 Η αποσύνδεση

Η αποσύνδεση τερματίζει μια υπάρχουσα σύνδεση και μπορεί να πραγματοποιηθεί είτε από τον σταθμό, είτε από το AP. Οι σταθμοί οφείλουν να προσπαθήσουν να αποσυνδεθούν όταν εγκαταλείπουν το δίκτυο. Ωστόσο, επειδή η απώλεια της επικοινωνίας μπορεί να μην το επιτρέπει, ένας μηχανισμός επιτρέπει στο AP να αποσυνδέσει το σταθμό χωρίς να απαιτείται ανταλλαγή μηνυμάτων, όταν ο σταθμός να καταστεί απρόσιτος πέραν κάποιου χρονικού ορίου.

Για να αποσυνδεθεί ένας σταθμός από το BSS, το AP ή ο σταθμός στέλνει ένα πλαίσιο αποσύνδεσης (Disassociation). Το πλαίσιο αυτό δεν είναι αίτηση, έτσι το άλλο μέρος απλώς επιβεβαιώνει την ορθή λήψη του.

1.2 Η κατανεμημένη διαδικασία πρόσβασης στο κανάλι (DCF)

Ο μηχανισμός CSMA/CA που χρησιμοποιείται στο MAC του 802.11, ονομάζεται κατανεμημένη λειτουργία συντονισμού (Distributed Coordination Function - DCF). Ένας σταθμός που επιθυμεί να μεταδώσει πρώτα εκτελεί μια λειτουργία εκτίμησης της κατάστασης χρήσης του μέσου, την Clear Chanel Assessment (CCA), ακούγοντας το μέσο για μια καθορισμένη χρονική περίοδο, την DCF Inter-Frame

Space (DIFS). Εάν το μέσο είναι ελεύθερο, ο σταθμός υποθέτει ότι μπορεί να λάβει την κυριότητα του μέσου και να αρχίσει μια ακολουθία ανταλλαγής πλαισίων. Εάν το μέσο είναι απασχολημένο, ο σταθμός περιμένει να ελευθερωθεί το μέσο, αναμένει για χρόνο DIFS και στη συνέχεια για ένα τυχαίο χρονικό διάστημα αναμονής (backoff period). Εάν το μέσο παραμένει ελεύθερο κατά το διάστημα αναμονής DIFS καθώς και του τυχαίου χρόνου αναμονής, ο σταθμός υποθέτει ότι μπορεί να πάρει την κυριότητα του μέσου και να αρχίσει μια ακολουθία ανταλλαγής πλαισίων.

Η τυχαία αυτή περίοδος οπισθοχώρησης (backoff) παρέχει την διάσταση αποφυγής της σύγκρουσης της DCF. Όταν το δίκτυο είναι φορτωμένο, πολλαπλοί σταθμοί αναμένουν το μέσο να ελευθερωθεί, έχοντας συσσωρεύσει πακέτα για αποστολή, ενώ το μέσο είναι κατειλημμένο. Δεδομένου ότι κάθε σταθμός επιλέγει, σύμφωνα με τις πιθανότητες, ένα διαφορετικό διάστημα οπισθοχώρησης, συγκρούσεις όπου περισσότεροι από ένας σταθμοί αρχίζουν τη μετάδοση την ίδια ακριβώς στιγμή είναι ελάχιστα πιθανό να συμβούν.

Μόλις ένας σταθμός έχει αποκτήσει πρόσβαση στο μέσο, διατηρεί τον έλεγχο του μέσου κρατώντας μια ελάχιστη χρονική απόσταση, την Short Inter-Frame Space (SIFS), μεταξύ των πλαισίων στην ακολουθία που εκπέμπει. Ένας άλλος σταθμός δεν θα δύναται να αποκτήσει πρόσβαση στο μέσο κατά τη διάρκεια αυτής της ακολουθίας, διότι θα πρέπει να αναμένει για μια καθορισμένη διάρκεια (DIFS) που είναι μεγαλύτερη από την SIFS. Υπάρχουν κανόνες που περιορίζουν τους επιτρεπτούς τύπους των ακολουθιών ανταλλαγής πλαισίων και τη διάρκεια των ακολουθιών αυτών για να αποτρέπεται ένας σταθμός από το να μονοπωλεί το μέσο.

Πρωτεύουσας σημασίας για το CSMA/CA είναι η αντίληψη της κατάστασης του φορέα. Η λειτουργία DCF χρησιμοποιεί τόσο φυσική όσο και εικονική λειτουργία αίσθησης για να καθορίσει την κατάσταση του μέσου. Η φυσική αίσθηση του φορέα βρίσκεται στο PHY και χρησιμοποιεί ανίχνευση της ενέργειας της εκπομπής και ανίχνευση του προοιμίου (Preamble) της εκπομπής για να καθορίσει πότε το μέσο είναι απασχολημένο. Η εικονική αίσθηση του φορέα υλοποιείται στο MAC και χρησιμοποιεί τις πληροφορίες δέσμευσης του μέσου που μεταφέρονται στο πεδίο Διάρκεια (Duration) της κεφαλίδας MAC, που ανακοινώνουν τη δέσμευση του μέσου για ορισμένο χρόνο. Ο εικονικός μηχανισμός αντίληψης του φορέα καλείται

Διάνυσμα Κατανομής Δικτύου (Network Allocation Vector - NAV). Το μέσο θεωρείται ότι παραμένει ελεύθερο μόνο όταν τόσο ο φυσικός, όσο και ο εικονικός μηχανισμός αίσθησης του μέσου δείχνουν να είναι έτσι.

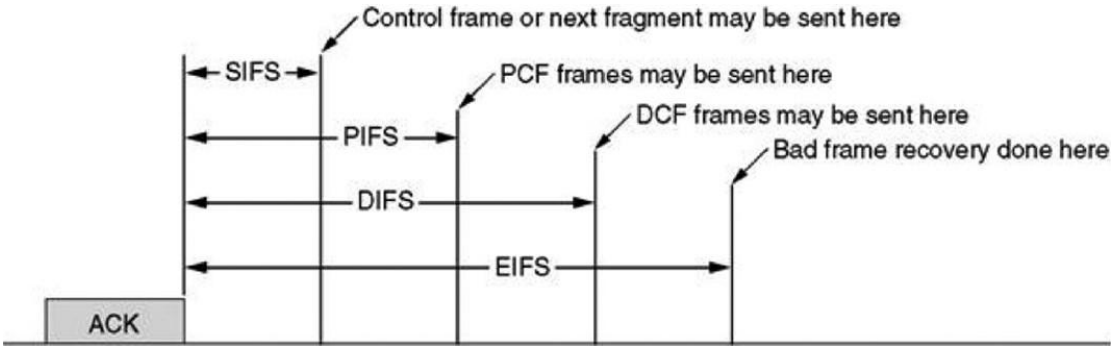
Η DCF κάνει επίσης χρήση της άμεσης ανατροφοδότησης από το βασικό μηχανισμό επιβεβαίωσης ορθής λήψης, που προβλέπει ο αποδέκτης να στέλνει ένα πλαίσιο ACK για την επιβεβαίωση της ορθής λήψης πλαισίου δεδομένων ή διαχείρισης από τον αποστολέα. Η μη λήψη πλαισίου ACK είναι μια πιθανή ένδειξη ότι η εκπομπή του αποστολέα δεν έχει ληφθεί σωστά, είτε λόγω σύγκρουσης, είτε λόγω κακών συνθηκών του καναλιού κατά τη στιγμή της διαβίβασης των δεδομένων.

Για να ελαχιστοποιηθεί περαιτέρω η πιθανότητα συγκρούσεων και, ως ένας πιο ισχυρός μηχανισμός ανίχνευσης της σύγκρουσης, ο σταθμός μπορεί να ξεκινήσει μια ακολουθία πλαισίων με μια σύντομη ανταλλαγή πλαισίων ελέγχου, χρησιμοποιώντας τα πιο ισχυρής διαμόρφωσης πλαίσια RTS (Ready To Send) και CTS (Clear To Send). Αυτό ενημερώνει τους NAV στους σταθμούς που περιβάλλουν τόσο τον αποστολέα όσο και τον αποδέκτη, μερικοί από τούς οποίους μπορεί να είναι κρυμμένοι κόμβοι που δεν είναι σε θέση να εντοπίσουν τις εκπομπές του πιο απομακρυσμένου σταθμού.

Η DCF παρέχει μια κατανεμημένη και ανταγωνιστικής φύσης λειτουργία πρόσβασης στο κανάλι. Οι σταθμοί ανταγωνίζονται για την πρόσβαση στο κανάλι, χωρίς την ανάγκη για ένα κεντρικό συντονιστή ή διαιτητή. Αυτός ο μηχανισμός διακρίνεται από αξιοσημείωτη αποτελεσματικότητα και κατανέμει αρκετά δίκαια το εύρος ζώνης μεταξύ των ενεργών σταθμών.

1.2.1 Ο Βασικός χρονισμός της DCF

Ο βασικός χρονισμός πρόσβασης καναλιού από την αρχική προδιαγραφή IEEE 802.11 απεικονίζεται στην Εικόνα 2. Οι διαφορετικές χρονικές αποστάσεις μεταξύ των πλαισίων (Inter-Frame Spaces - IFS) παρέχουν πρόσβαση στο ασύρματο μέσο, σε διαφορετικά επίπεδα προτεραιότητας.



Εικόνα 2 – Inter Frame Spaces [9]

1.2.1.1 SIFS (Short Inter-Frame space)

Το διάστημα SIFS χρησιμοποιείται για το διαχωρισμό ενός πλαισίου απάντησης από το πλαίσιο που έχει προκαλέσει την απάντηση αυτή, για παράδειγμα μεταξύ ενός πλαισίου δεδομένων και το πλαίσιο επιβεβαίωσης ACK. Το SIFS έχει σχεδιαστεί να είναι όσο το δυνατόν συντομότερο, διατηρώντας την ικανότητα να χωρέσει τις καθυστερήσεις που προκύπτουν σε μια πραγματική υλοποίηση. Αυτές οι καθυστερήσεις περιλαμβάνουν την καθυστέρηση του PHY στην αποκωδικοποίηση και την αποδιαμόρφωση του λαμβανόμενου πλαισίου, τον απαιτούμενο χρόνο επεξεργασίας για το ληφθέν πλαίσιο και την δόμηση της απάντησης στο MAC, καθώς επίσης και τον χρόνο εκκίνησης του πομπού για να στείλει την απάντηση αυτή.

Το SIFS χρησιμοποιείται επίσης για τον διαχωρισμό μεμονωμένων πλαισίων, σε μια ριπή συνεχόμενων πλαισίων δεδομένων. Οι σταθμοί που έχουν πρόσβαση στο μέσο χρησιμοποιώντας χρονισμό SIFS, δεν ελέγχουν αν το μέσο είναι απασχολημένο, αλλά απλώς μεταβαίνουν σε λειτουργία εκπομπής (αν δεν είναι ήδη) και αρχίζουν τη μετάδοση στο όριο λήξης του SIFS.

Η διάρκεια του SIFS για κάθε ένα συγκεκριμένο PHY ορίζεται από την παράμετρο aSIFSTime. Για τα PHY των 802.11a, 802.11g και 802.11n, η τιμή της παραμέτρου αυτής είναι 16 μs.

1.2.1.2 Χρόνος θυρίδας (Slot time)

Ο χρονισμός για τα άλλα διαστήματα μεταξύ πλαισίων (IFS) είναι SIFS συν έναν ακέραιο αριθμό χρόνων θυρίδας, με έναρξη μετάδοσης στο όριο λήξης της τελευταίας θυρίδας. Στην πράξη, οι καθυστερήσεις στη διάδοση του σήματος και

σε μικρότερο μικρό βαθμό οι διαφορές στην ακρίβεια της κάθε υλοποίησης, έχουν ως αποτέλεσμα ο κάθε σταθμός να αντιλαμβάνεται ένα ελαφρώς διαφορετικό όριο της θυρίδας. Η διάρκεια της θυρίδας έχει σχεδιαστεί έτσι ώστε να χωράει αυτές τις διαφορές και να εξασφαλίζει αρκετό χρόνο για την ανίχνευση από γειτονικούς σταθμούς του προοιμίου (Preamble) του σταθμού που εκπέμπει, πριν από την έναρξη της επόμενης θυρίδας. Κατά τη διάρκεια κάθε χρονοθυρίδας, οι σταθμοί που δεν εκπέμπουν, παραμένουν σε κατάσταση λήψης και ελέγχουν αν το μέσο παραμένει ελεύθερο.

Ο χρόνος της θυρίδας για ένα συγκεκριμένο PHY ορίζεται από την παράμετρο aSlotTime. Για τα PHY των 802.11a, 802.11g και 802.11n, η τιμή της παραμέτρου αυτής είναι 9 μs.

1.2.1.3 PIFS (Point Coordination Function Inter-Frame Space)

Η απόσταση μεταξύ πλαισίων κατά χρονικό διάστημα PIFS, παρέχει την επόμενη υψηλότερη προτεραιότητα πρόσβασης, μετά από το SIFS και χρησιμοποιείται για να αποκτηθεί κατά προτεραιότητα πρόσβαση στο μέσο.

Το PIFS ορίζεται από την εξίσωση : PIFS = aSIFSTime + aSlotTime

Το AP χρησιμοποιεί αναμονή κατά χρόνο PIFS για να αποκτήσει πρόσβαση στο μέσο κατά προτεραιότητα και να στείλει ένα Beacon, να ξεκινήσει μια περίοδο ελεύθερη ανταγωνισμού ή να ανακτήσει την πρόσβαση στο μέσο εάν δεν ληφθεί το αναμενόμενο πλαίσιο απάντησης κατά τη διάρκεια μιας τέτοιας περιόδου. Το PIFS χρησιμοποιείται επίσης και για άλλες ενέργειες προτεραιότητας, όπως από ένα σταθμό που πρέπει να στείλει ένα πλαίσιο ανακοίνωσης αλλαγής καναλιού.

1.2.1.4 DIFS (DCF Inter-Frame Space)

Το DCF χρησιμοποιείται από τους σταθμούς που λειτουργούν σύμφωνα με τη DCF, για να μεταδίδουν πλαίσια δεδομένων ή πλαίσια διαχείρισης.

Το DIFS ορίζεται από την εξίσωση: DIFS = aSIFSTime + 2 × aSlotTime

Ένας σταθμός που κάνει χρήση της DCF, επιτρέπεται να μεταδίδει, εφόσον κρίνει ότι το μέσο είναι ελεύθερο κατά τη διάρκεια του DIFS ή αν κρίνει ότι το μέσο είναι ελεύθερο κατά τη διάρκεια του DIFS συν τον εναπομένοντα χρόνο οπισθοχώρησης μετά την λήψη ενός ορθά ληφθέντος πλαισίου.

1.2.1.5 Ο τυχαίος χρόνος οπισθοχώρησης (Random Backoff time)

Όταν το μέσο αλλάζει κατάσταση από κατειλημμένο σε ελεύθερο, περισσότεροι του ενός σταθμοί μπορεί να είναι έτοιμοι να στείλουν δεδομένα. Για να ελαχιστοποιηθούν οι συγκρούσεις, οι σταθμοί που επιθυμούν να ξεκινήσουν τη μετάδοση επιλέγουν ένα τυχαίο αριθμό οπισθοχώρησης και αναβάλλουν την εκπομπή για αυτό τον αριθμό χρονοθυρίδων (Slot times). Ο αριθμός οπισθοχώρησης έχει επιλεγεί ως ένας ψευδοτυχαίος ακέραιος προερχόμενος από μια κανονική κατανομή των τιμών στο διάστημα [0,CW], όπου CW, είναι μια ακέραια τιμή που ονομάζεται παράθυρο ανταγωνισμού (Contention Window).

Η παράμετρος CW λαμβάνει την αρχική τιμή CWmin και διπλασιάζεται σε κάθε αποτυχημένη προσπάθεια μετάδοσης δεδομένων, π.χ. κάθε φορά που δεν έχει ληφθεί απάντηση ACK για ένα πλαίσιο δεδομένων. Αν το CW φτάσει να γίνει ίσο με το CWmax, παραμένει στην τιμή αυτή μέχρι την επαναφορά του στην αρχική τιμή. Το CW επαναφέρεται στην αρχική CWmin μετά από κάθε επιτυχημένη μετάδοση δεδομένων.

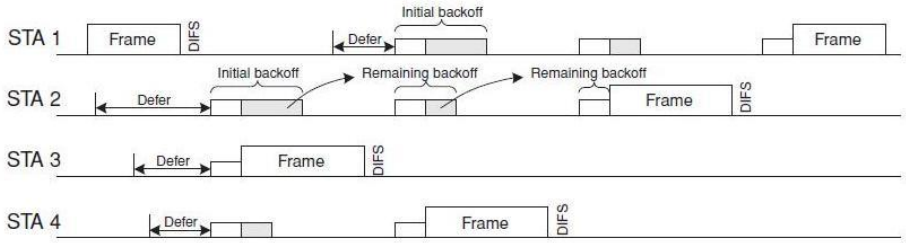
Τα CW, CWmin και CWmax μπορεί να πάρουν τιμές που είναι δυνάμεις του 2, μείον 1 (2^n-1). Για την DCF, τα CWmin και CWmax είναι καθορισμένα από το συγκεκριμένο PHY που χρησιμοποιείται. Για τα PHY των 802.11a, 802.11g και 802.11n, το CWmin είναι 15 και το CWmax είναι 1023. Το CW θα ξεκινήσει έτσι με την τιμή 15 και όταν διπλασιαστεί θα λάβει την επόμενη υψηλότερη δύναμη του 2, μείον 1, μέχρι να φτάσει την τιμή 1023, δηλ. 15,31,63,..., 1023.

1.2.2 Η διαδικασία της οπισθοχώρησης

Ξεκινώντας τη διαδικασία τυχαίας οπισθοχώρησης, ο σταθμός επιλέγει ένα τυχαίο αριθμό οπισθοχώρησης μέσα από το διάστημα [0,CW]. Όλες οι χρονοθυρίδες οπισθοχώρησης (Backoff Slots) ακολουθούν μετά από χρονικό διάστημα DIFS, κατά τη διάρκεια του οποίου το μέσο αναμένεται να παραμένει ελεύθερο. Κατά τη διάρκεια κάθε χρονοθυρίδας οπισθοχώρησης ο σταθμός συνεχίζει να παρακολουθεί το μέσο.

Εάν το μέσο γίνει απασχολημένο κατά τη διάρκεια μια χρονοθυρίδας οπισθοχώρησης, τότε η διαδικασία οπισθοχώρησης αναστέλλεται. Ο μετρητής του αριθμού υποχώρησης συνεχίζει από την τιμή που είχε κατά την αναστολή, όταν το

μέσο ελευθερώνεται και πάλι για περίοδο DIFS. Η διαδικασία αυτή παρουσιάζεται στο παρακάτω σχήμα.



Εικόνα 3 – Η διαδικασία τυχαίας οπισθοχώρησης (backoff procedure) [1]

Όταν είναι πολλαπλοί οι σταθμοί που αναβάλουν την μετάδοση και μεταβαίνουν σε κατάσταση τυχαίας οπισθοχώρησης, τότε ο σταθμός που επιλέγει το μικρότερο αριθμό οπισθοχώρησης (ο STA 3) θα κερδίσει τον ανταγωνισμό και θα διαβιβάσει πρώτος. Οι υπόλοιποι σταθμοί αναστέλλουν τη διαδικασία οπισθοχώρησης και επανέρχονται μετά από χρόνο DIFS, μετά την απελευθέρωση του μέσου. Ο σταθμός με την επόμενη μεγαλύτερη τιμή στο μετρητή οπισθοχώρησης θα κερδίσει την επόμενη φορά (STA 4) και, τελικά, στη συνέχεια, ο επόμενος σταθμός με το μεγαλύτερο αριθμό στο μετρητή (STA 2).

Ένας σταθμός που ξεκινά μια νέα διαδικασία πρόσβασης (ο STA 1 εκ νέου) θα επιλέξει έναν τυχαίο αριθμό για το μετρητή οπισθοχώρησης, από το πλήρες CW και έτσι θα έχει την τάση να επιλέξει μια μεγαλύτερη τιμή από ότι οι υπόλοιποι σταθμοί (όπως ο STA 2) που έχουν ήδη αναστείλει την διαδικασία οπισθοχώρησης τους, από μια προηγούμενη προσπάθεια πρόσβασης.

1.3 Ανταλλαγή πλαισίων DATA/ACK

Μετάδοση σε ένα ασύρματο μέσο είναι εγγενώς επιρρεπής σε σφάλματα. Η μεταφορά των δεδομένων επωφελείται από έναν, χαμηλής καθυστέρησης, μηχανισμό επανάληψης, στο επίπεδο της ζεύξης δεδομένων (LLC), που να επιτρέπει την αναμετάδοση των πλαισίων που δεν έχουν επιτυχώς αποδιαμορφωθεί από τον παραλήπτη. Ο βασικός μηχανισμός με τον οποίο αυτό επιτυγχάνεται, είναι ο σταθμός που λαμβάνει σωστά ένα πλαίσιο δεδομένων που απευθύνεται σε αυτόν, να στείλει μια άμεση επιβεβαίωση με τη μορφή ενός

πλαισίου ACK. Εάν ο σταθμός που στέλνει το πλαίσιο δεδομένων δεν λάβει πλαίσιο ACK, θεωρεί ότι το πλαίσιο δεδομένων δεν ελήφθη σωστά και μπορεί να το στείλει ξανά.

Όλα τα πλαίσια δεδομένων δεν επιβεβαιώνονται με τον ίδιο παραπάνω τρόπο. Τα πλαίσια δεδομένων broadcast και multicast, απευθύνονται προς το σύνολο ή ένα υποσύνολο των σταθμών σε ένα WLAN και δεν μπορεί να επιβεβαιωθεί η άφιξη τους με αυτόν τον τρόπο. Στα ασύρματα τοπικά δίκτυα 802.11, τα πλαίσια broadcast και multicast δεν επωφελούνται από την πρόσθετη αξιοπιστία που παρέχει ο μηχανισμός επιβεβαίωσης.

Ο αριθμός των επιτρεπτών προσπαθειών αναμετάδοσης για ένα συγκεκριμένο MSDU (MAC Service Data Unit) είναι περιορισμένος. Ο σταθμός εκπομπής διατηρεί ένα μετρητή του αριθμού των προσπαθειών αναμετάδοσης για ένα MSDU και όταν αυτός υπερβεί το προκαθορισμένο όριο αναμεταδόσεων, το MSDU απορρίπτεται.

Για να ενισχυθεί η αξιοπιστία του μηχανισμού επιβεβαίωσης, το πλαίσιο ACK είναι ισχυρής διαμόρφωσης, δηλαδή αποστέλλεται χρησιμοποιώντας ένα χαμηλότερο ρυθμό δεδομένων στο PHY, από ότι τα πλαίσια δεδομένων. Η πρόσθετη επιβάρυνση που προκύπτει από την χρήση ισχυρής διαμόρφωσης είναι σχετικά μικρή δεδομένου ότι το ίδιο το πλαίσιο ACK είναι πολύ σύντομο.

1.3.1 Κατακερματισμός (Fragmentation)

Ο κατακερματισμός χρησιμοποιείται για να χωριστεί ένα μεγάλο MSDU (MAC Service Data Unit) σε μικρότερα τμήματα, ώστε να βελτιωθεί η πιθανότητα ότι τα MSDU θα λαμβάνονται σωστά και να μειωθεί η επιβάρυνση από την αναμετάδοση. Σε χαμηλούς ρυθμούς δεδομένων, μια μη κατακερματισμένη MSDU μπορεί να καταλάβει ένα μεγάλο μέρος του χρόνου του αέρα. Για παράδειγμα, στην περίπτωση που ένα πλαίσιο δεδομένων μεγέθους 1500 bytes, που αποστέλλεται με ρυθμό 1 Mbps του IEEE 802.11b, η μετάδοση διαρκεί 12 ms, πράγμα που την καθιστά επιρρεπή στις μεταβαλλόμενες συνθήκες του καναλιού. Ένα σφάλμα σε ένα bit του πλαισίου, θα οδηγούσε στην αναμετάδοση ολόκληρου του πλαισίου.

Με τον κατακερματισμό, το MSDU θα σπάσει σε μικρότερα τμήματα/θραύσματα και κάθε θραύσμα θα ενθυλακωθεί σε ένα MPDU (MAC Protocol Data Unit). Κάθε

MPDU θα σταλεί σε ξεχωριστό PPDU (Physical Protocol data Unit). Ένα σφάλμα σε ένα bit, θα έχει ως αποτέλεσμα την αναμετάδοση μόνο της MPDU που φέρει το σφάλμα.

Τα Θραύσματα που αποτελούν ένα ολόκληρο MSDU αποστέλλονται ως μεμονωμένα MPDU. Ένας σταθμός μπορεί να στείλει κάθε ένα θραύσμα σε μια ξεχωριστή πρόσβαση στο κανάλι ή η κατακερματισμένη MSDU μπορεί να αποσταλεί ως μια ριπή από MPDU μετά από μία και μόνο πρόσβαση στο κανάλι.

Μια MSDU κατακερματίζεται όταν το μήκος της υπερβαίνει το όριο που καθορίζεται από την παράμετρο *dot11FragmentationThreshold*. Κάθε θραύσμα περιέχει άρτιο αριθμό bytes και όλα τα θραύσματα έχουν το ίδιο μέγεθος, εκτός από το τελευταίο θραύσμα, το οποίο επιτρέπεται να είναι μικρότερο. Τα θραύσματα παραδίδονται στη σειρά.

Ο αποστολέας ενός κατακερματισμένου MSDU διατηρεί ένα χρονόμετρο. Η παράμετρος d*ot11MaxTransmitMSDULifetime*, καθορίζει το μέγιστο χρονικό διάστημα που επιτρέπεται να διαρκεί η μετάδοση ενός MSDU. Ο αποστολέας ξεκινά το χρονόμετρο με την πρώτη προσπάθεια για τη μετάδοση το πρώτου θραύσματος της MSDU. Αν το χρονόμετρο υπερβεί την τιμή της dot11MaxTransmitMSDULifetime, τότε απορρίπτονται όλα τα υπόλοιπα θραύσματα.

Ο αποδέκτης ενός κατακερματισμένου MSDU διατηρεί επίσης ένα χρονόμετρο. Η παράμετρος *aMaxReceiveLifetime* καθορίζει το μέγιστο χρονικό διάστημα που επιτρέπεται να διαρκέσει η λήψη μιας MSDU. Το χρονόμετρο του λήπτη της MSDU ξεκινά με την υποδοχή του πρώτου θραύσματος. Εάν το χρονόμετρο υπερβεί την τιμή της *aMaxReceiveLifetime*, τότε απορρίπτονται όλα τα θραύσματα του MSDU. Επιπλέον θραύσματα που μπορεί να ληφθούν αργότερα, επίσης απορρίπτονται.

1.3.2 Ανίχνευση διπλοτύπων πλαισίων

Κατά τη μετάδοση, υπάρχει η πιθανότητα ένα πλαίσιο που είχε ληφθεί σωστά να ληφθεί και πάλι, π.χ. αν ο αποστολέας εκπέμψει ξανά ένα πλαίσιο, διότι η ίδια η απάντηση ACK, δεν αποδιαμορφώθηκε σωστά από αυτόν. Για τον εντοπισμό διπλότυπων πλαισίων, το Πλαίσιο δεδομένων περιλαμβάνει ένα ψηφίο Retry (flag) και ένα πεδίο Sequence Control, το οποίο αποτελείται από έναν

αύξοντα αριθμό σειράς και έναν αύξοντα αριθμό θραύσματος. Στο ψηφίο Retry τίθεται η τιμή 1, σε κάθε πλαίσιο που επανεκπέμπεται. Ο αριθμός σειράς παράγεται ως μια αύξουσα ακολουθία ακεραίων αριθμών που ανατίθενται στα MSDU και στα πλαίσια διαχείρισης. Εάν ένα MSDU ή πλαίσιο διαχείρισης είναι κατακερματισμένο, τότε κάθε θραύσμα λαμβάνει τον ίδιο αριθμό σειράς και έναν διαφορετικό αύξοντα αριθμό θραύσματος.

Για τον εντοπισμό των διπλότυπων πλαισίων, ο σταθμός λήψης παρακολουθεί τον αύξοντα αριθμό σειράς και τον αριθμό θραύσματος της τελευταίας MSDU ή του πλαισίου διαχείρισης που έλαβε από κάθε σταθμό με τον οποίο επικοινωνεί. Δηλαδή διατηρεί ένα χώρο προσωρινής αποθήκευσης των πλειάδων { διεύθυνση αποστολής, αριθμός σειράς, αριθμός θραύσματος }, για κάθε θραύσμα που έλαβε για τον τελευταίο αριθμό σειράς που εμφανίζεται από κάθε διεύθυνση αποστολής, εκτός των διευθύνσεων broadcast και multicast. Εάν ο σταθμός λάβει ένα MPDU με το ψηφίο επανάληψης (Retry bit) να είναι ίσο με 1, και να ταιριάζει με μια από τις προσωρινά αποθηκευμένες πλειάδες, τότε απορρίπτει την MPDU ως διπλότυπο.

1.3.3 Δικαιοσύνη στην πρόσβαση στο μέσο

Ο μηχανισμός της κατανεμημένης πρόσβασης στο κανάλι, προωθεί τη δικαιοσύνη ανάμεσα στους σταθμούς, με την έννοια ότι όλοι οι σταθμοί του δικτύου που έχουν δεδομένα για να στείλουν, θα στείλουν κατά μέσο όρο τον ίδιο αριθμό πλαισίων δεδομένων. Αν όλοι χρησιμοποιούν το ίδιο μέγεθος πακέτου, θα έχουν όλοι την ίδια απόδοση, ανεξαρτήτως του ρυθμού δεδομένων του συγκεκριμένου PHY που χρησιμοποιούν.

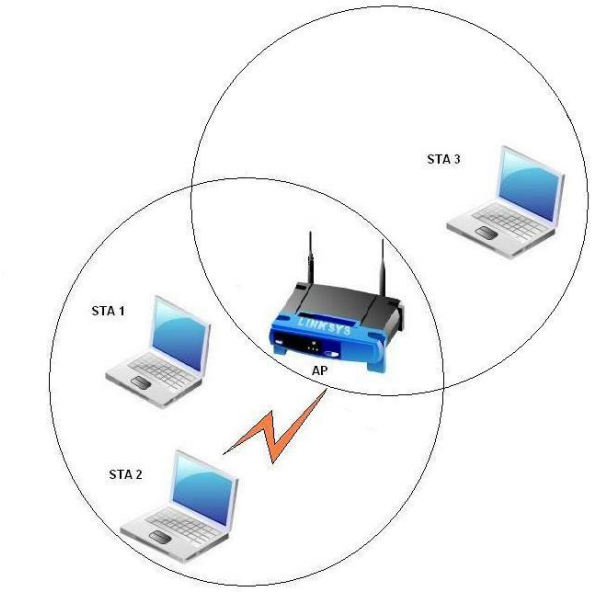
Για παράδειγμα, ας υποθέσουμε ότι υπάρχουν δύο σταθμοί του δικτύου, οι STA 1 και STA 2, οι οποίοι ανταγωνίζονται μεταξύ τους για την πρόσβαση στο κανάλι προκειμένου να στείλουν δεδομένα στον σταθμό STA 3. Ας υποθέσουμε επίσης, ότι ο STA 1 χρησιμοποιεί ένα υψηλού ρυθμού δεδομένων PHY, ενώ ο STA 2 χρησιμοποιεί ένα χαμηλού ρυθμού μετάδοσης. Κάθε σταθμός ανταγωνίζεται για την πρόσβαση στο κανάλι προκειμένου να στείλει ένα πλαίσιο δεδομένων και κάθε σταθμός κατά μέσο όρο θα πάρει τον ίδιο αριθμό ευκαιριών μετάδοσης. Ωστόσο, επειδή ο STA 2 χρησιμοποιεί χαμηλότερο ρυθμό μετάδοσης δεδομένων, θα

χρησιμοποιήσει περισσότερο χρόνο αέρα για να στείλει τα πλαίσια του, από ότι ο STA 1.

1.4 Το πρόβλημα του κρυφού κόμβου

Η κατανεμημένη φύση της διαδικασίας πρόσβασης στο κανάλι, κάνει τον μηχανισμό αντίληψης της κατάστασης του φορέα, ιδιαίτερα κρίσιμο για την λειτουργία χωρίς συγκρούσεις. Η φυσική ανίχνευση του φορέα, η οποία υλοποιείται στο PHY, είναι υπεύθυνη για τον εντοπισμό των εκπομπών των άλλων σταθμών. Σε ορισμένες όμως περιπτώσεις μπορεί να μην είναι δυνατή η φυσική ανίχνευση των εκπομπών όλων των σταθμών.

Σε ένα σενάριο, όπως αυτό στο παρακάτω σχήμα, όπου υπάρχει μεταφορά δεδομένων μεταξύ του STA 2 και του AP. Η εκπομπή από τον STA 2 μπορεί να ανιχνευθεί από το AP και τον STA 1. Ένας άλλος κόμβος, ο STA 3, μπορεί να ανιχνεύσει τις εκπομπές από το AP, αλλά όχι όμως και από τον STA 1. Ο STA 3 είναι ένας κρυφός κόμβος όσον αφορά την επικοινωνία μεταξύ του STA 2 και του AP. Όταν ο STA 2 εκπέμψει ένα πλαίσιο προς το AP, υπάρχει η πιθανότητα ο STA 3 να εξακολουθεί να βλέπει το μέσο ως ελεύθερο και να αρχίσει μια εκπομπή πλαισίου, με πιθανό αποτέλεσμα την σύγκρουση.



Εικόνα 4 - Το πρόβλημα του κρυφού κόμβου

Για την αντιμετώπιση του προβλήματος των κρυφών κόμβων χρησιμοποιείται ένας συνδυασμός από τους παρακάτω μηχανισμούς προστασίας.

1.4.1 Το Διάνυσμα Κατανομής Δικτύου (Network Allocation Vector - NAV)

Ένας μηχανισμός που καθορίζεται για να ξεπεραστεί το πρόβλημα του κρυμμένου κόμβου είναι το διάνυσμα κατανομής του δικτύου (NAV). Ο NAV είναι μια λειτουργία που βρίσκεται στο επίπεδο MAC και παρέχει μια εικονική αίσθηση του φορέα για να συμπληρώσει τη φυσική αίσθηση του φορέα.

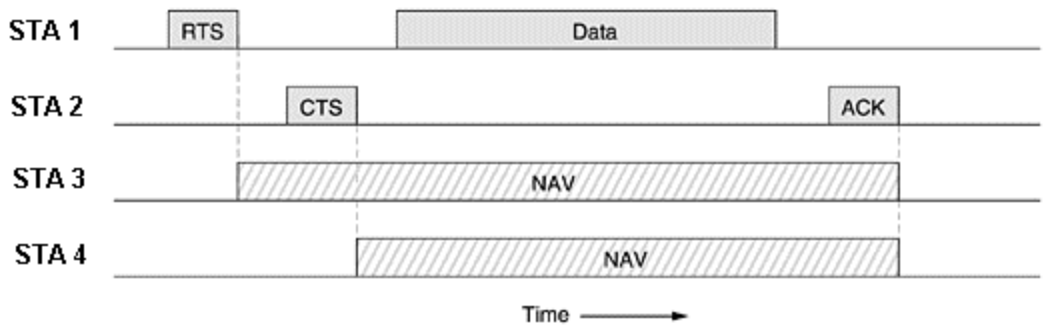
Κάθε πλαίσιο MAC φέρει ένα πεδίο διάρκειας (Duration) που χρησιμοποιείται για την ενημέρωση των NAV σε κάθε σταθμό, εκτός από το σταθμό στον οποίο απευθύνεται το πλαίσιο. Το πεδίο Duration φέρει μια τιμή χρόνου, που δείχνει τη διάρκεια για την οποία ο σταθμός αποστολής αναμένει ότι θα απασχολήσει το μέσο.

Όλα τα πλαίσια περιλαμβάνουν το πεδίο Duration και μπορεί να ενημερώσουν τους NAV σε γειτονικούς σταθμούς. Ωστόσο, για να το πράξουν αυτό, το πλαίσιο πρέπει να αποδιαμορφωθεί επιτυχώς από τους γειτονικούς σταθμούς. Οι NAV ενημερώνονται πιο αποτελεσματικά στους γειτονικούς σταθμούς, όταν χρησιμοποιούνται πλαίσια ελέγχου με ισχυρή διαμόρφωση, όπως στην ανταλλαγή RTS/CTS, αντί για κοινά πλαίσια δεδομένων.

1.4.2 Ανταλλαγή πλαισίων RTS/CTS

Για την προστασία της εκπομπής του, από κρυμμένους κόμβους, ένας σταθμός μπορεί να ξεκινήσει την μετάδοση δεδομένων με μια ανταλλαγή πλαισίων RTS/CTS όπως απεικονίζεται στο σχήμα 4. Το RTS (Request To Send) στέλνεται από τον σταθμό που εκπέμπει (STA 1), και ο σταθμός στον οποίο απευθύνεται το RTS (STA 2) αποκρίνεται με ένα πλαίσιο CTS (Clear To Send). Το πλαίσιο RTS καταλαμβάνει λιγότερο χρόνο στον αέρα από ότι το πλαίσιο δεδομένων και είναι επομένως λιγότερο επιρρεπές σε συγκρούσεις. Επίσης η απώλεια του RTS λόγω σύγκρουσης θα εντοπιστεί γρήγορα και τα πλαίσια RTS και CTS είναι ισχυρής

διαμόρφωσης.



Εικόνα 5 Μετάδοση με χρήση του μηχανισμού RTS/CTS [9]

Το πεδίο διάρκειας (Duration) του πλαισίου RTS φέρει την αναγκαία ρύθμιση χρόνου για τους NAV, ώστε να καλύπτεται ο χρόνος απάντησης με το CTS και ο χρόνος που απαιτείται για την επακόλουθη ανταλλαγή πλαισίων. Το πλαίσιο απάντησης CTS έχει το δικό του πεδίο διάρκειας, του οποίου η τιμή τίθεται ίση με τη διάρκεια στο RTS, μειωμένη κατά SIFS και κατά το χρόνο διάρκειας της ίδιας της απόκρισης CTS. Στο διάγραμμα, ο κρυφός κόμβος (STA 4), θα λάβει το CTS πλαίσιο και θα ενημερώσει τον NAV του, ώστε να αναμένει κατά την επερχόμενη ανταλλαγή πλαισίων, ενώ ο STA 3 θα λάβει τόσο το RTS, όσο και το CTS.

Η χρήση RTS/CTS απαιτείται όταν το μήκος ενός πλαισίου δεδομένων ή πλαισίου διαχείρισης υπερβαίνει το όριο που καθορίζεται από την παράμετρο dot11RTSThreshold. Η dot11RTSThreshold είναι μια παράμετρος διαχείρισης και μπορεί να ρυθμιστεί με τιμή από 0, ώστε όλες οι MPDU να μεταδίδονται με μια ανταλλαγή RTS/CTS, έως το μέγιστο επιτρεπόμενο μήκος MPDU, έτσι ώστε να μη χρειάζεται να χρησιμοποιηθεί καθόλου ανταλλαγή RTS/CTS, ή οποιαδήποτε άλλη τιμή στο μεταξύ.

1.4.3 EIFS (Extended Inter-Frame Space)

Ένας άλλος μηχανισμός που χρησιμοποιείται για την προστασία από τους κρυφούς κόμβους είναι το EIFS. Ένας σταθμός χρησιμοποιεί EIFS αντί DIFS για να αναβάλει τη μετάδοση του όταν ένα πλαίσιο έχει ανιχνευθεί, αλλά δεν ήταν δυνατή η ορθή λήψη του, δηλαδή το MAC διαπιστώσει ότι η ακολουθία ελέγχου πλαισίου (Frame Control Sequence - FCS) δεν είναι έγκυρη.

Το EIFS ορίζεται ως εξής: EIFS = aSIFSTime + ACKTxTime + DIFS

όπου ACKTxTime είναι ο χρόνος που απαιτείται για τη μετάδοση ενός πλαισίου ACK με το χαμηλότερο υποχρεωτικό ρυθμό δεδομένων του PHY. Το EIFS αποσκοπεί στο να αποτρέπεται ένας σταθμός από το να εκπέμψει πάνω στη μετάδοση ενός ACK από ένα κρυφό κόμβο, όταν ο σταθμός αυτός δεν είναι σε θέση να αποδιαμορφώσει το πλαίσιο δεδομένων και συνεπώς να ρυθμιστεί σωστά τον NAV του.

Εάν κατά τη διάρκεια της αναμονής κατά EIFS, ληφθεί ένα έγκυρο πλαίσιο (π.χ. το ACK), τότε χρησιμοποιείται αναμονή κατά DIFS μετά το συγκεκριμένο πλαίσιο, αντί να συνεχίζεται η διαδικασία με EIFS.

1.5 Η Βελτιωμένη κατανεμημένη πρόσβαση καναλιού (EDCA)

Η EDCA είναι μια επέκταση της βασικής λειτουργίας DCF, που εισήχθη με το 802.11e για την υποστήριξη QoS μέσω καθορισμού προτεραιοτήτων. Ο μηχανισμός της EDCA ορίζει τέσσερις κατηγορίες πρόσβασης (Access Categories - ACs). Κάθε AC χαρακτηρίζεται από συγκεκριμένες τιμές για ένα σύνολο παραμέτρων πρόσβασης που στατιστικά δίνουν μεγαλύτερη προτεραιότητα πρόσβασης στο κανάλι για ένα AC έναντι ενός άλλου. Οι σχετικές προτεραιότητες πρόσβασης των τεσσάρων AC δίνονται στον παρακάτω πίνακα. Ένα MSDU με μία συγκεκριμένη προτεραιότητα χρήστη, θεωρείται ότι ανήκει σε μια ξεχωριστή κατηγορία κίνησης (Traffic Category - TC), με αυτή την προτεραιότητα.

Priority	**802.11D User priority**	**AC**	**Designation**
Lowest	1	AC_BK	Background
	2		
	0	AC_BE	Best Effort
	3		

	4	AC_VI	Video
	5		
Highest	6	AC_VO	Voice
	7		

Πίνακας 1 – Οι κατηγορίες προτεραιότητας (ACs) [1]

Σύμφωνα με την EDCA, η εξερχόμενη από το σύστημα κίνηση ταξινομείται λογικά σε τέσσερις ουρές, μία για κάθε AC. Ένα στιγμιότυπο της διαδικασίας πρόσβασης της EDCA λειτουργεί για κάθε ουρά που ανταγωνίζεται για πρόσβαση, με τις παραμέτρους πρόσβασης του συγκεκριμένου AC, όταν η ουρά δεν είναι κενή.

Οι διαδικασίες πρόσβασης της EDCA, όπως και στην DCF, ανταγωνίζονται για την πρόσβαση το μέσο, αναβάλλοντας για ένα καθορισμένο χρονικό διάστημα, το AIFS (Arbitration Inter-Frame Space) και στη συνέχεια για ένα τυχαίο χρονικό διάστημα υποχώρησης, όταν το μέσο ελευθερώνεται. Οι παράμετροι πρόσβασης στην EDCA είναι παρόμοιες με αυτές που χρησιμοποιούνται στην DCF, αλλά ορίζονται ανά AC.

Η τιμή του AIFS για κάθε AC συμβολίζεται με AIFS[AC]. Το Contention Window (CW) από το οποίο επιλέγεται το τυχαίο backoff count συμβολίζεται ως CW[AC].

Το CW για το κάθε συγκεκριμένο AC συμβολίζεται ως CW[AC] και αρχίζει με την τιμή CWmin[AC]. Αν η μετάδοση του frame για ένα συγκεκριμένο AC δεν είναι επιτυχής, το CW[AC] διπλασιάζεται. Αν το CW[AC] φτάσει το CWmax[AC], παραμένει σε αυτή την τιμή μέχρι την επαναφορά στην αρχική. Το CW[AC] επανέρχεται στο CWmin[AC] μετά από μια επιτυχημένη μετάδοση MPDU από αυτό το AC.

Αν δύο (ή περισσότερα) στιγμιότυπα της λειτουργίας EDCA αποκτήσουν πρόσβαση ταυτόχρονα, η εσωτερική σύγκρουση επιλύεται με το υψηλότερης προτεραιότητας AC να αποκτά πρόσβαση και το άλλο AC να συμπεριφέρεται σαν να συνέβη μια εξωτερική σύγκρουση, με τον διπλασιασμό του CW και επιχειρώντας εκ' νέου την πρόσβαση.

1.5.1 Η έννοια της Ευκαιρίας εκπομπής

Μία από τις έννοιες που εισάγεται με το 802.11e είναι η ευκαιρία εκπομπής (Transmit Opportunity - TXOP). Η TXOP είναι ένα ορισμένο χρονικό διάστημα κατά το οποίο ένας σταθμός μπορεί να μεταδώσει δεδομένα ενός συγκεκριμένου AC. Σύμφωνα με την EDCA, μια TXOP αποκτάται από το σταθμό μέσω της διαδικασίας πρόσβασης στο κανάλι, χρησιμοποιώντας τις παραμέτρους πρόσβασης για τη συγκεκριμένη AC, για την οποία η TXOP θα χρησιμοποιηθεί. Μόλις η TXOP έχει ληφθεί, ο σταθμός μπορεί να αρχίσει να μεταδίδει συνεχώς πλαίσια δεδομένων, ελέγχου, και διαχείρισης και να λαμβάνει πλαίσια απάντησης, εφόσον η χρονική διάρκεια της ακολουθίας των πλαισίων δεν υπερβαίνει το χρονικό όριο της TXOP που έχει οριστεί για το συγκεκριμένο AC. Όταν το χρονικό όριο της TXOP είναι ίσο με το μηδέν, σημαίνει ότι μόνο ένα MSDU ή πλαίσιο διαχείρισης μπορεί να μεταδοθεί, πριν από τον εκ' νέου ανταγωνισμό για την πρόσβαση.

Η έννοια της TXOP προωθεί την δικαιοσύνη στη χρήση των πόρων και όχι τη δικαιοσύνη στην ποσότητα του διακινούμενου φορτίου, έτσι ώστε όλοι οι σταθμοί στο δίκτυο με κυκλοφορία της ίδιας κατηγορίας κατά μέσο όρο θα λάβουν το ίδιο χρόνο στον αέρα.

Αν υποθέσουμε ότι δύο σταθμοί ανταγωνίζονται για την πρόσβαση, ο ένας με ένα PHY (φυσικό επίπεδο) υψηλού ρυθμού μετάδοσης δεδομένων και ο άλλος με ένα PHY χαμηλού ρυθμού, και οι δύο σταθμοί, κατά μέσο όρο, θα λαμβάνουν τον ίδιο χρόνο στον αέρα, αλλά ο σταθμός με το υψηλότερου ρυθμού PHY θα πετυχαίνει υψηλότερη απόδοση. Χωρίς TXOP, και οι δύο σταθμοί θα πετυχαίνουν την ίδια απόδοση στη μεταφορά φορτίου, αλλά με άνιση χρήση των πόρων.

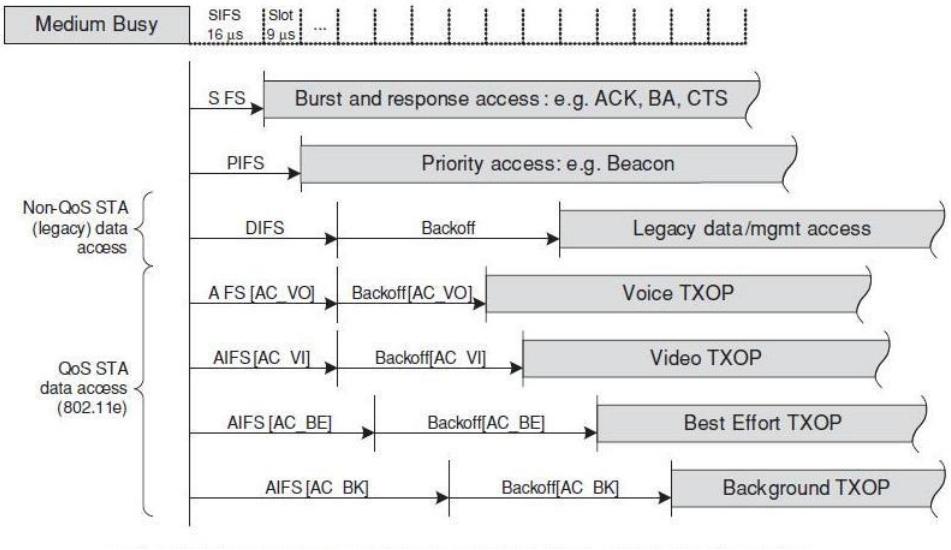
1.5.2 Ο χρονισμός πρόσβασης στην EDCA

Στην Εικόνα 6 διακρίνονται οι προτεραιότητες πρόσβασης για τις τέσσερις AC σε σχέση με την DCF και μεταξύ τους.

Το AIFS για ένα συγκεκριμένο AC ορίζεται από το εξίσωση:

AIFS [AC] = aSIFSTime + AIFSN [AC] × aSlotTime,

όπου AIFSN [AC] είναι ο αριθμός των slot.



Εικόνα 6 - Οι κατηγορίες πρόσβασης στο κανάλι και ο σχετικός χρονισμός τους [1]

1.5.3 Παράμετροι πρόσβασης της EDCA

Οι παράμετροι πρόσβασης της EDCA παρέχονται στο στοιχείο πληροφοριών παραμέτρων EDCA, που υπάρχει στα πλαίσια Beacon και Probe Response. Οι σταθμοί στο BSS χρησιμοποιούν την τελευταία έκδοση των παραμέτρων και το AP μπορεί να προσαρμόσει τις παραμέτρους στην πάροδο του χρόνου, π.χ. με βάση το φορτίο του δικτύου και τον αριθμό των συνεργαζόμενων σταθμών.

Οι προεπιλεγμένες παράμετροι πρόσβασης του EDCA για τα 802.11a , 802.11 b, και 802.11n PHY, δίνονται στον Πίνακα 2. Οι προεπιλεγμένες παράμετροι EDCA χρησιμοποιούνται εάν το AP δεν έχει ανακοινώσει ένα διαφορετικό σύνολο παραμέτρων. Αν και δεν αποτελεί κατηγορία πρόσβασης, ο πίνακας δείχνει επίσης, για σύγκριση, και τις ισοδύναμες παραμέτρους για το DCF (legacy).

Οι παράμετροι πρόσβασης EDCA προσδιορίζουν το βαθμό στον οποίο ένα AC έχει προτεραιότητα πάνω από ένα άλλο. Η παράμετρος AIFSN παρέχει σχετικά μικρή διαφοροποίηση. Οι AC με χαμηλότερη τιμή αποκτούν πρόσβαση πιο συχνά από οι AC με υψηλότερη τιμή, όταν οι άλλες παράμετροι είναι ίδιες. Η παράμετρος CWmin παρέχει πολύ μεγαλύτερη διαφοροποίηση. Ο τυχαίος χρόνος υποχώρησης (backoff count) επιλέγεται από το διάστημα [0,CW], όπου το CW συνήθως είναι ίσο με το CWmin. Η Αύξηση της περιοχής από την οποία η επιλέγεται το backoff count, έχει μεγαλύτερη επίδραση επί της συνολικής περιόδου

αναβολής (defer period), από ότι η διαφορά στην AIFSN. Το όριο της TXOP είναι επίσης ένα ισχυρό μέσο διαφοροποίησης. Μια AC με μεγάλο όριο TXOP θα λάβει περισσότερο χρόνο του αέρα από μια AC με ένα μικρό όριο TXOP, αν έχουμε ίση κατανομή των TXOP.

AC	CWmin	CWmax	AIFSN	Όριο TXOP
AC_BK	31	1023	7	0
AC_BE	31	1023	3	0
AC_VI	15	31	2	3.008ms
AC_VO	7	15	2	1.504ms
legacy	15	1023	2	0

Πίνακας 2 - Οι προκαθορισμένες παράμετροι πρόσβασης της EDCA για τα PHY των 802.11a, 802.11g και 802.11n [1]

1.5.4 Η αναθεώρηση του EIFS

Σύμφωνα με το DCF, ένας σταθμός πρέπει να αναβάλει τη μετάδοση για χρόνο EIFS αντί για DIFS μετά από ένα πλαίσιο που ανιχνεύθηκε αλλά δεν αποδιαμορφώθηκε επιτυχώς. Ο σκοπός είναι να προστατευθεί μια πιθανή απάντηση ACK, ενός πιο μακρινού κόμβου, ο οποίος μπορεί να μην αποκωδικοποίησε επιτυχώς το πλαίσιο από το σταθμό και δεν ενημέρωσε το NAV του. Για να επιτευχθεί ισοδύναμη συμπεριφορά στην EDCA, ένας σταθμός πρέπει να αναβάλει για EIFS - DIFS + AIFS[AC], μετά από ένα πλαίσιο που εντοπίζεται αλλά δεν αποδιαμορφώνεται επιτυχώς.

Τα EIFS και EIFS - DIFS + AIFS[AC], είναι οι δύο αντίστοιχοι τρόποι για να εκφραστεί ότι μετά από ένα ανεπιτυχώς αποδιαμορφωμένο πλαίσιο ένας σταθμός πρέπει να περιμένει για SIFS συν τη διάρκεια ενός πλαισίου ACK, πριν από την εκτέλεση της συνήθους διαδικασίας αναμονής κατά DIFS ή AIFS.

1.5.5 Ανίχνευση σύγκρουσης

Όταν ένας σταθμός αποκτά μια TXOP τότε μπορεί να διαβιβάσει για τη διάρκεια της TXOP, είτε ένα ενιαίο πλαίσιο, είτε μια ριπή διαδοχικών πλαισίων δεδομένων. Υπάρχει, ωστόσο η πιθανότητα δύο σταθμοί να αποκτήσουν πρόσβαση στο κανάλι ταυτόχρονα και οι συγκρουόμενες μεταδόσεις που θα προκύψουν είναι πιθανό να είναι ακατάληπτες για τους σταθμούς που τις λαμβάνουν.

Για να ελαχιστοποιηθεί η απώλεια χρόνου αέρα λόγω αυτών των συγκρούσεων, οι σταθμοί πρέπει να εκτελούν μια σύντομη ανταλλαγή πλαισίων κατά την έναρξη της TXOP, ώστε να ανιχνεύεται μια σύγκρουση. Η σύντομη ανταλλαγή πλαισίων μπορεί να είναι μία ανταλλαγή RTS/CTS ή μια ανταλλαγή DATA/ACK, μόνο με ένα μικρό πλαίσιο δεδομένων.

Η ανίχνευση σύγκρουσης είναι επίσης απαραίτητη για τη σωστή λειτουργία της διαδικασίας οπισθοχώρησης (backoff), στην οποία πρέπει να διπλασιαστεί το μέγεθος του CW σε περίπτωση σύγκρουσης. Ο διπλασιασμός του παραθύρου CW μειώνει την πιθανότητα να ξανασυμβεί σύγκρουση την επόμενη φορά που δύο σταθμοί επιχειρούν πρόσβαση στο κανάλι.

Η σύντομη ανταλλαγή πλαισίων που εκτελείται κατά την έναρξη της TXOP επιτρέπει επίσης στους δύο σταθμούς που συμμετέχουν στην ανταλλαγή, να ενημερώσουν τους NAV των γειτονικών τους σταθμών.

1.5.6 Το πλαίσιο δεδομένων QoS

Για την υποστήριξη των χαρακτηριστικών του QoS και της ομαδικής επιβεβαίωσης (Block ACK), το 802.11e καθόρισε ένα νέο πλαίσιο δεδομένων, το πλαίσιο δεδομένων QoS. Το QoS πλαίσιο δεδομένων έχει τα ίδια πεδία, όπως το κοινό πλαίσιο δεδομένων του 802.11, αλλά περιλαμβάνει και ένα επιπλέον πεδίο ελέγχου QoS. Το πεδίο ελέγχου QoS περιλαμβάνει διάφορα υποπεδία για τη διαχείριση του QoS και άλλων χαρακτηριστικών που εισάγονται με το 802.11e.

Το TID ή αναγνωριστικό κυκλοφορίας, προσδιορίζει την κατηγορία κυκλοφορίας (Traffic Category - TC) στην οποία ανήκει το πλαίσιο. Στην EDCA, το πεδίο TID φέρει την προτεραιότητα χρήστη (User Priority), που αντιστοιχιστεί στο AC σύμφωνα με τον Πίνακα 2.

Το υποπεδίο πολιτικής επιβεβαίωσης (ACK policy), καθορίζει τον τρόπο με τον οποίο επιβεβαιώνεται η παραλαβή από τον παραλήπτη και φέρει μία από τις ακόλουθες τιμές:

- **Normal ACK –** όταν γίνει σωστή λήψη, ο παραλήπτης ανταποκρίνεται στο πλαίσιο δεδομένων QoS με ένα πλαίσιο ACK.
- **No ACK -** ο παραλήπτης δεν θα ανταποκριθεί στο πλαίσιο δεδομένων QoS. Αυτό μπορεί να είναι χρήσιμο για κυκλοφορία που έχει μικρή ανοχή στη διακύμανση της καθυστέρησης (jitter) ή την καθυστέρηση και δεν επωφελείται από την αναμετάδοση των δεδομένων.
- **No Explicit ACK -** μπορεί να υπάρξει ένα πλαίσιο ανταπόκρισης, αλλά δεν θα είναι ένα πλαίσιο ACK. Αυτή η πολιτική χρησιμοποιείται όταν εφαρμόζεται κεντρικός συντονισμός της πρόσβασης στο κανάλι, αντί των DCF ή EDCA.
- **Block ACK -** ο παραλήπτης δεν κάνει καμία ενέργεια σχετικά με το ληφθέν πλαίσιο, εκτός από την καταγραφή της ορθής λήψης. Η πολιτική αυτή χρησιμοποιείται στα πλαίσια του πρωτοκόλλου ομαδικής επιβεβαίωσης.

1.6 Το πρωτόκολλο ομαδικής επιβεβαίωσης (Block ACK)

Το πρωτόκολλο ομαδικής επιβεβαίωσης (Block ACK), το οποίο εισήχθη με την τροπολογία 802.11e, βελτιώνει την αποδοτικότητα με την μεταβίβαση μιας ομάδας πλαισίων δεδομένων που επιβεβαιώνεται με ένα και μόνο πλαίσιο ACK, αντί ενός ACK για καθένα από τα επιμέρους πλαίσια δεδομένων. Σε αντίθεση με τον κανονικό μηχανισμό επιβεβαίωσης, στο μηχανισμό ομαδικής επιβεβαίωσης ένας σταθμός πρέπει να δημιουργήσει μια συνεδρία ομαδικής επιβεβαίωσης με τον σταθμό που επικοινωνεί, για κάθε αναγνωριστικό κυκλοφορίας (TID), για την οποία πρόκειται να γίνει μεταφορά δεδομένων. Μια συγκεκριμένη συνεδρία ομαδικής επιβεβαίωσης προσδιορίζεται από το τρίδυμο {Διεύθυνση εκπομπής - Διεύθυνση Λήψης - TID}.

Το 802.11e εισήγαγε δύο εναλλακτικές εκδοχές του πρωτοκόλλου ομαδικής επιβεβαίωσης: την άμεση ομαδική επιβεβαίωση (Immediate Block ACK) και την καθυστερημένη ομαδική επιβεβαίωση (Delayed Block ACK). Οι δύο εκδοχές

διαφέρουν στον τρόπο με τον οποίο ανταλλάσσονται τα πλαίσια ελέγχου της ομαδικής επιβεβαίωσης.

Στην άμεση ομαδική επιβεβαίωση, ένα πλαίσιο αίτησης ομαδικής επιβεβαίωσης (Block ACK Request - BAR) προκαλεί την άμεση ανταπόκριση με αποστολή πλαισίου BA, έτσι ώστε το BA να έχει επιστρέψει εντός SIFS από την παραλαβή του BAR και συνεπώς εντός της ίδιας TXOP. Στην καθυστερημένη ομαδική επιβεβαίωση, το BAR έχει σταλεί σε μία TXOP και η απάντηση BA επιστρέφει σε ξεχωριστή, επόμενη TXOP. Η άμεση ομαδική επιβεβαίωση παρέχει μικρότερη καθυστέρηση και βελτιωμένη απόδοση σε σχέση με την καθυστερημένη ομαδική επιβεβαίωση, η οποία ορίστηκε για λόγους ευκολίας της υλοποίησης της.

Η ομαδική επιβεβαίωση ενεργοποιείται, προς μία κατεύθυνση και για ένα συγκεκριμένο TID, με μια ανταλλαγή ADDBA Request/ADDBA Response. Ο σταθμός που έχει δεδομένα προς αποστολή, αποστέλλει ένα αίτημα ADDBA στο σταθμό που θα λάβει τα δεδομένα. Ο παραλήπτης επιβεβαιώνει την ορθή λήψη της αίτησης ADDBA με ένα ACK και στη συνέχεια απαντά λίγο αργότερα με μια απάντηση ADDBA, την οποία ο αποστολέας επιβεβαιώνει επίσης με ένα ACK. Η ανταλλαγή ADDBA επιτρέπει στους αποστολέα και παραλήπτη την ανταλλαγή παραμέτρων, όπως το μέγεθος του προσωρινού χώρου αναδιάταξης (reorder buffer). Για την διακοπή μιας συνεδρίας ομαδικής επιβεβαίωσης (Block ACK session), ένας από τους δύο σταθμούς στέλνει ένα αίτημα DELBA, το οποίο αν ληφθεί σωστά επιβεβαιώνεται με ένα ACK.

Η ομαδική μεταφορά πλαισίων δεδομένων λαμβάνει χώρα ως ακολούθως. Ο αποστολέας μεταδίδει ένα ή περισσότερα πλαίσια δεδομένων QoS, με τη διεύθυνση προορισμού του παραλήπτη, που περιέχουν το TID της συνεδρίας. Το πεδίο πολιτικής επιβεβαίωσης (ACK Policy) στα πλαίσια αυτά έχει ρυθμιστεί να είναι Block ACK. Τα πλαίσια δεδομένων της ομάδας δεν απαιτείται να αποσταλούν στη σειρά και μπορεί να περιλαμβάνονται και αναμεταδόσεις πλαισίων. Ο παραλήπτης είναι υπεύθυνος για την αναδιάταξη των πλαισίων και την παράδοση τους στο υψηλότερο επίπεδο δικτύου, στη σωστή σειρά. Εκτελεί αυτή τη λειτουργία χρησιμοποιώντας τον προσωρινό χώρο της μνήμης αναδιάταξης (reorder buffer). Ο παραλήπτης θα αποθηκεύει τα πλαίσια στο buffer αυτό, έως ότου συμπληρωθούν τα κενά με τους αύξοντες αριθμούς πλαισίων που λείπουν (sequence numbers). Ο αποστολέας περιορίζει το εύρος των αριθμών σειράς, για

τους οποίους εκκρεμεί επιβεβαίωση, έτσι ώστε να μην γίνει υπέρβαση του reorder buffer.

Μετά την αποστολή μιας ομάδας πλαισίων δεδομένων, ο αποστολέας στέλνει ένα πλαίσιο ομαδικής επιβεβαίωσης BAR (Block ACK Request). Το πλαίσιο BAR εκτελεί δύο λειτουργίες: Καθαρίζει το reorder buffer του παραλήπτη και αιτείται ανταπόκριση με ένα πλαίσιο BA. Ο buffer αναδιάταξης του παραλήπτη μπορεί να χρειαστεί να καθαριστεί για να ξεπεραστούν τα όποια κενά στην σειρά των αυξόντων αριθμών που προκύπτουν από MSDU που δεν κατάφεραν να περάσουν μετά την εξάντληση του μέγιστου αριθμού αναμεταδόσεων ή τη λήξη διάρκειας της ζωής. Η διαδικασία καθαρισμού του buffer αναδιάταξης, ξεμπλοκάρει τα MSDU που μπορεί να έχουν κολλήσει πίσω από τα παραπάνω κενά.

Το πλαίσιο BAR περιλαμβάνει ένα πεδίο ελέγχου έναρξης ακολουθίας (Starting Sequence Control - SSC), που περιέχει τον αύξοντα αριθμό σειράς του παλαιότερου MSDU στην ομάδα, για το οποίο αναμένεται επιβεβαίωση. Τα MSDU στο buffer του παραλήπτη, με αριθμούς σειράς που προηγούνται του αριθμού αυτού, είτε προωθούνται στο επίπεδο LLC (αν έχουν ολοκληρωθεί) ή απορρίπτονται (εάν ένα ή περισσότερα θραύσματα λείπουν). Το προκαλούμενο πλαίσιο BA περιέχει ένα bitmap που αντιπροσωπεύει την κατάσταση επιβεβαίωσης των πλαισίων δεδομένων που έχουν ήδη παραληφθεί, αρχίζοντας από τον αριθμό σειράς (sequence number) που δόθηκε από το πλαίσιο BAR.

Με την παραλαβή του πλαισίου BA, ο αποστολέας απορρίπτει τα πλαίσια δεδομένων για τα οποία έχει λάβει επιβεβαίωση και προωθεί για αναμετάδοση αυτά για τα οποία δεν έχει λάβει επιβεβαίωση. Ο αποστολέας μπορεί επίσης να απορρίψει πλαίσια δεδομένων που έχουν υπερβεί το όριο αναμεταδόσεων ή τη διάρκεια ζωής τους.

Κατά την συνεδρία ομαδικής επιβεβαίωσης, ο αποστολέας διατηρεί τη δυνατότητα να προκαλέσει μια επιβεβαίωση με κανονικό ACK, για πλαίσια δεδομένων QoS, θέτοντας Normal ACK στο πεδίο πολιτικής επιβεβαίωσης.

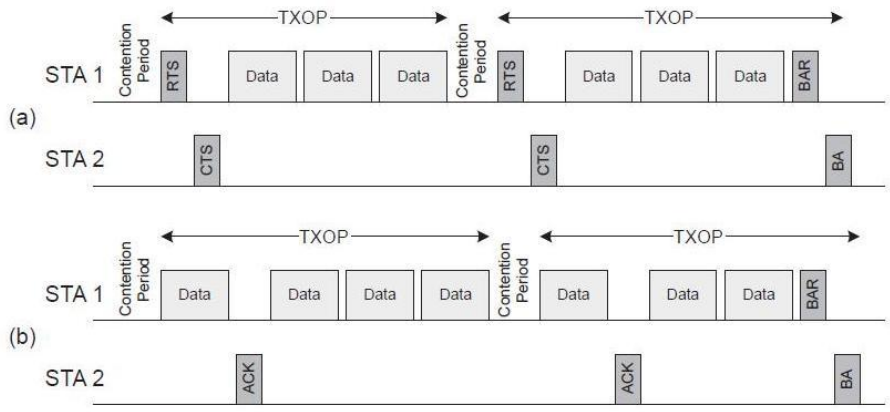
1.7 Ανταλλαγή ομάδας πλαισίων δεδομένων

Η ανταλλαγή ομάδας πλαισίων δεδομένων χρησιμοποιώντας το πρωτόκολλο άμεσης επιβεβαίωσης (Immediate Block ACK) απεικονίζεται στην Εικόνα 6(α), με τον STA 1 να μεταδίδει δεδομένα στον STA 2.

Μετά από μια περίοδο ανταγωνισμού για το μέσο, το STA 1 αποκτά μία TXOP. Ως ένας μηχανισμός ανίχνευσης σύγκρουσης και για να ενημερωθούν οι NAV στους γειτονικούς σταθμούς, ο STA 1 ξεκινάει μια σύντομη ανταλλαγή πλαισίων, σε αυτή την περίπτωση RTS/CTS. Ο STA 1 στη συνέχεια στέλνει μία ριπή συνεχόμενων πλαισίων δεδομένων με χρόνο SIFS μεταξύ των μεμονωμένων μεταδόσεων, μέχρι να φτάσει το όριο της TXOP. Με δεδομένο ότι έχει και άλλα δεδομένα για αποστολή, ο STA 1 αποκτά πρόσβαση και πάλι στο ασύρματο μέσο και κερδίζει μια νέα TXOP. Πραγματοποιείται και πάλι μία ανταλλαγή RTS/CTS, που ακολουθείται από τα υπόλοιπα πλαίσια της ομάδας να αποστέλλονται ως ριπή διαδοχικών πλαισίων. Ο STA 1 στη συνέχεια στέλνει ένα πλαίσιο BAR, με το οποίο ζητάει ένα BA ως απάντηση από τον STA 2. Η απάντηση BA υποδεικνύει ποια από τα πλαίσια δεδομένων από την ομάδα είχαν ληφθεί σωστά.

Ως εναλλακτική της εκτέλεσης της ανταλλαγής RTS/CTS, ο STA 1 μπορεί να ξεκινήσει μια ανταλλαγή DATA/ACK, για να πραγματοποιήσει την ανίχνευση σύγκρουσης, όπως απεικονίζεται στην εικόνα 8(b). Η ανίχνευση σύγκρουσης είναι απαραίτητο να γίνει, με έναν από αυτούς τους μηχανισμούς, καθώς μειώνει την υποβάθμιση της απόδοσης (throughput) του δικτύου που διαφορετικά θα προκύψει, εάν δύο σταθμοί συγκρούονταν για όλη τη διάρκεια της TXOP. Η ανταλλαγή DATA/ACK προβλέπει πιο περιορισμένη προστασία στην πλευρά του πομπού (λόγω της υψηλότερης τάξης διαμόρφωσης που χρησιμοποιείται για το πλαίσιο δεδομένων), αλλά είναι πιο αποδοτική από την ανταλλαγή RTS/CTS με την οποία δεν υπάρχει μεταφορά πληροφοριών όταν πραγματοποιείται.

Θα πρέπει να σημειωθεί ότι η μεταφορά της ομάδας πλαισίων είναι ανεξάρτητη από την TXOP. Η μεταφορά της ομάδας πλαισίων μπορεί να γίνει σε πολλαπλές TXOP ή μπορεί να περιέχεται εντός μιας ενιαίας TXOP.



Εικόνα 7 - Η διαδικασία ομαδικής ανταλλαγής πλαισίων [1]

ΚΕΦΑΛΑΙΟ 2 Ο ΠΡΟΣΟΜΟΙΩΤΗΣ ns-3

2.1 Τι είναι το ns-3

Το ns-3 είναι ένας προσομοιωτής διακριτών γεγονότων δικτύου, προορίζεται πρωτίστως για την έρευνα και την εκπαιδευτική χρήση. Το ns-3 είναι ελεύθερο λογισμικό, υπό την άδεια χρήσης GNU (GPLv2) και είναι διαθέσιμο στο κοινό για την έρευνα, την ανάπτυξη και τη χρήση.

Ο στόχος του ns-3 είναι να αναπτυχθεί ένα δημοφιλές, ανοικτό περιβάλλον προσομοίωσης για την έρευνα στα δίκτυα. Να ευθυγραμμιστεί με τις ανάγκες της σύγχρονης έρευνας δικτύων για προσομοίωση και να ενθαρρύνει τη συμβολή της κοινότητας των χρηστών του, την αξιολόγηση από ομότιμους, και την επικύρωση του παραγόμενου λογισμικού.

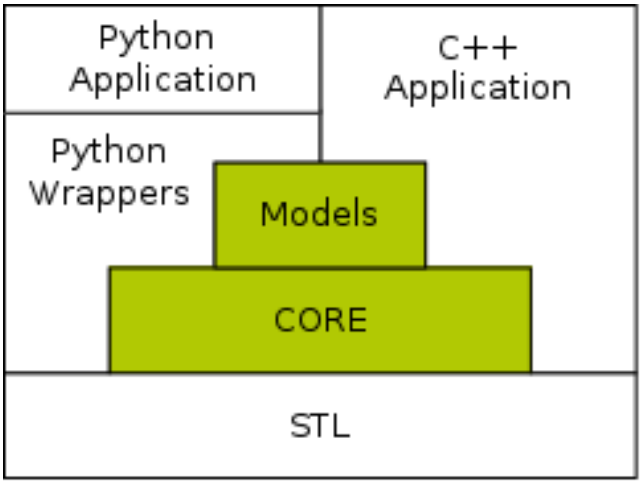
2.2 Οι τεχνολογίες του ns-3

Το ns-3 είναι μια βιβλιοθήκη της γλώσσας C++, η οποία παρέχει μια σειρά από μοντέλα προσομοίωσης δικτύου, που υλοποιούνται ως αντικείμενα της C++, καθώς και τα περιτυλίγματα τους σε γλώσσα python.

Οι χρήστες συνήθως αλληλεπιδρούν με αυτή τη βιβλιοθήκη γράφοντας ένα πρόγραμμα σε C++ ή python που αρχικοποιεί μια σειρά από μοντέλα προσομοίωσης για να δημιουργήσει το επίμαχο σενάριο προσομοίωσης, εισέρχεται στον κύριο βρόγχο επανάληψης της προσομοίωσης, και εξέρχεται όταν η προσομοίωση έχει ολοκληρωθεί.

2.2.1 Περιτυλίγματα Python

Η βιβλιοθήκη ns-3 είναι περιτυλιγμένη σε python χάρη στη βιβλιοθήκη pybindgen οποία διαβιβάζει τις κεφαλίδες C++ του ns-3 στα gccxml και pygccxml ώστε να δημιουργήσει αυτόματα τον απαραίτητο κώδικα σύνδεσης με τη C++. Αυτά τα αυτομάτως δημιουργούμενα αρχεία C++, τελικά μεταφράζονται σε μια μονάδα python του ns-3, ώστε να επιτρέπεται στους χρήστες να αλληλεπιδρούν με τα γραμμένα σε C++ μοντέλα και πυρήνα του ns-3, μέσω σεναρίων γραμμένων σε python.



Εικόνα 8 - Η βασική αρχιτεκτονική του NS-3 [2]

2.2.2 Σχεδιασμός υψηλού επιπέδου

Σε σύγκριση με άλλους προσομοιωτές διακριτών γεγονότων δικτύου, το ns-3 διακρίνεται από τους παρακάτω υψηλού επιπέδου στόχους, στο σχεδιασμού του:

- **Έμφαση στη C++ και την Python**

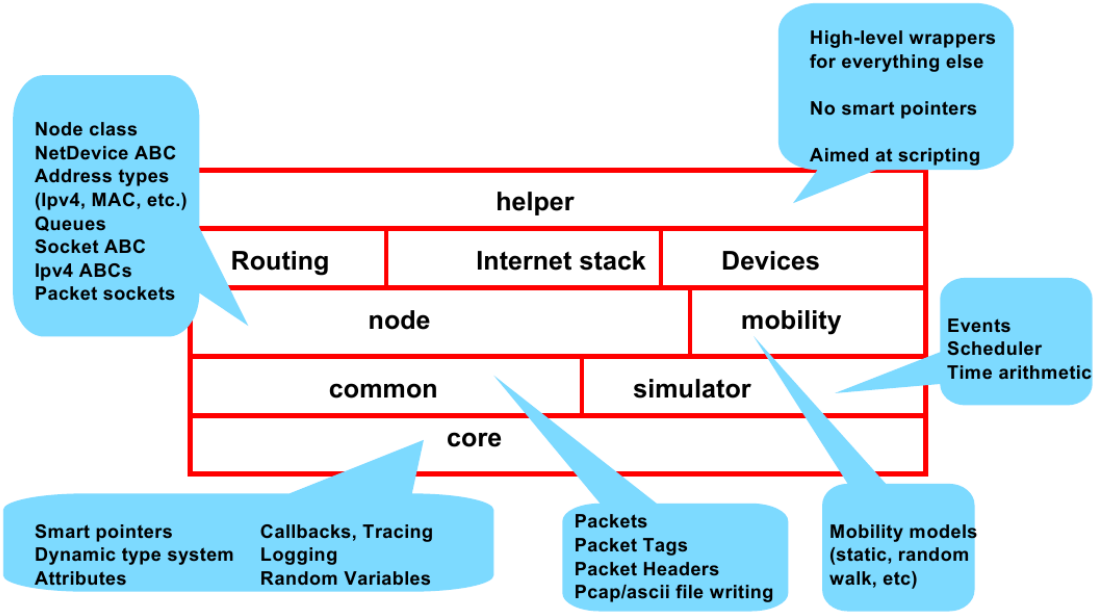
Το ns-3 χρησιμοποιεί τις γλώσσες C++ και Python για την περιγραφή των μοντέλων και της ροής της προσομοίωσης, αντί μιας ειδικού σκοπού γλώσσας μοντελοποίησης. Επιτρέποντας έτσι στους χρήστες να επωφεληθούν από την πλήρη υποστήριξη των παραπάνω γλωσσών.

- **Τα γεγονότα και οι συνδέσεις υλοποιούνται δυναμικά, με Callbacks**

Τα γεγονότα της προσομοίωσης στο ns-3 είναι απλώς και μόνο κλήσεις συναρτήσεων που έχουν προγραμματιστεί για να εκτελεστούν σε προκαθορισμένο χρόνο της προσομοίωσης. Οποιαδήποτε συνάρτηση μπορεί να μετατραπεί σε γεγονός και να χρονοπρογραμματιστεί προς εκτέλεση, με τη χρήση μιας συνάρτησης ανάκλησης. Αυτή η μέθοδός ακολουθείται, σε αντίθεση με τις εξειδικευμένες συναρτήσεις χειριστή γεγονότων (handlers), που συγκεντρώνουν την επεξεργασία των γεγονότων, σε κάθε αντικείμενο της προσομοίωσης. Οι ανακλήσεις επίσης χρησιμοποιούνται στον προσομοιωτή κυρίως για να μειωθούν οι εξαρτήσεις μεταξύ των αντικειμένων της προσομοίωσης κατά τη μεταγλώττιση.

- **Ευέλικτος πυρήνας με στρώμα βοηθών**

Το ns-3 διαθέτει ένα ισχυρό χαμηλού επιπέδου API που παρέχει στους χρήστες, που έχουν την απαραίτητη ικανότητα, μεγάλη ευελιξία στη ρύθμιση των λεπτομερειών στη συμπεριφορά της προσομοίωσης. Ώς ένα στρώμα πάνω από αυτό, υπάρχει ένα σύνολο βοηθητικών API (helpers) που παρέχουν ευκολότερη στη χρήση λειτουργικότητα, υλοποιώντας ικανοποιητικά προεπιλεγμένη συμπεριφορά. Οι χρήστες του ns-3 μπορούν να επιλέξουν κατά περίπτωση μεταξύ των απλούστερων API στο επίπεδο των «βοηθών» και του πλήρους API του χαμηλότερου επιπέδου ή μία μίξη αυτών.



Εικόνα 9 - Η εσωτερική αρχιτεκτονική του ns-3 [2]

- **Έμφαση στην εξομοίωση**

Ο σχεδιασμός της προσομοίωσης είναι προσανατολισμένος στις περιπτώσεις χρήσης που επιτρέπουν στον προσομοιωτή να αλληλεπιδρά με τον πραγματικό κόσμο. Τα αντικείμενα πακέτων του ns-3 αποθηκεύονται εσωτερικά ως packet byte buffers (όπως τα πακέτα στα πραγματικά λειτουργικά συστήματα), έτοιμα να προωθηθούν σε ουρά για να αποσταλούν σε μία πραγματική διασύνδεση δικτύου (network interface). Αρκετές διαφορετικές προσομοιώσεις έχουν διεξαχθεί ενσωματωμένες σε πραγματικά συστήματα (simulation-in-the-loop), έχουν αναπτυχθεί frameworks ολοκλήρωσης σε εικονικές μηχανές και έχουν γίνει πειράματα με ns-3 σε φυσικά συστήματα δοκιμών.

- **Έμφαση στην επαναχρησιμοποίηση λογισμικού**

Παράλληλα με το ns-3, έχει αναπτυχθεί και ένα περιβάλλον απευθείας εκτέλεσης κώδικα που επιτρέπει στους χρήστες να τρέχουν πολλές συμβατές με POSIX εφαρμογές εντός της προσομοίωσης. Το Direct Code Execution (DCE) είναι ένα framework για το ns-3, το οποίο παρέχει την υποδομή για να εκτελούνται, εντός του ns-3, υπάρχουσες υλοποιήσεις πρωτοκόλλων δικτύωσης ή εφαρμογές, χωρίς να απαιτούνται αλλαγές στον πηγαίο κώδικα. Για παράδειγμα, αντί να χρησιμοποιηθεί η υλοποίηση του ns-3 για μια εφαρμογή τύπου ping, μπορεί να χρησιμοποιηθεί η πραγματική υλοποίηση. Επίσης μπορεί να χρησιμοποιηθεί σε προσομοιώσεις το networking stack του Linux.

- **Συμβατότητα με τις πραγματικές διασυνδέσεις**

Οι κόμβοι του ns-3 είναι σχεδιασμένοι σύμφωνα με την αρχιτεκτονική δικτύωσης του Linux, και οι βασικές διεπαφές και αντικείμενα (sockets, net devices) ευθυγραμμίζονται με εκείνες σε έναν υπολογιστή Linux. Αυτό διευκολύνει καλλίτερα την επαναχρησιμοποίηση κώδικα και βελτιώνει το ρεαλισμό των μοντέλων, κάνει επίσης τη ροή εκτέλεσης στον προσομοιωτή ευκολότερη στη σύγκριση με τα πραγματικά συστήματα .

- **Διαχείριση των παραμέτρων**

Ο προσομοιωτής ns-3 διαθέτει ενσωματωμένο σύστημα για τη διαχείριση των προκαθορισμένων ή των κατά περίπτωση οι τιμών για τις παραμέτρους προσομοίωσης, βασισμένο σε ιδιότητες. Όλες οι προεπιλεγμένες τιμές των παραμέτρων που επιδέχονται τροποποίηση διαχειρίζονται από αυτό το σύστημα, που περιλαμβάνει επεξεργασία ορισμάτων της γραμμής εντολών, τεκμηρίωση μέσω του Doxygen, και ένα βασισμένο σε XML και προαιρετικά σε GTK υποσύστημα ρυθμίσεων.

- **Η Απουσία IDE**

Ο ns-3 δεν περιλαμβάνει ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) για τη διαμόρφωση, τη διόρθωση, την εκτέλεση, και την απεικόνιση των προσομοιώσεων σε ένα μόνο παράθυρο εφαρμογής, όπως συμβαίνει σε άλλους προσομοιωτές. Αντί αυτού, η κύρια εργασία γίνεται στη γραμμή εντολών και ενσωματώνονται εργαλεία για τη διαμόρφωση και την απεικόνιση, αναλόγως με τις απαιτήσεις. Ορισμένοι προγραμματιστές έχουν χρησιμοποιήσει το περιβάλλον του Eclipse ως IDE.

2.3 Το βασικό μοντέλο δικτύου

Οι βασικές αφαιρετικές οντότητες που απαρτίζουν ένα μοντέλο δικτύου στο ns-3, είναι οι ακόλουθες:

2.3.1 Κόμβος

Συνήθως μια συσκευή Η/Υ που συνδέεται σε ένα δίκτυο, στην ορολογία του Internet, ονομάζεται host ή μερικές φορές και τερματικό (end system). Λόγω του ότι ο όρος host, συνδέεται στενά με το Internet και τα πρωτοκολλά του, επιλέχτηκε για το ns-3 ο πιο γενικός όρος κόμβος (Node), που προέρχεται από τη θεωρία των γράφων.

Ο κόμβος (Node) είναι η βασική αφαιρετική έννοια που αναπαριστά μία συσκευή Η/Υ που συνδεμένη δίκτυο. Αυτή η έννοια στο ns-3 αναπαρίσταται, σε C++, με την κλάση Node. Η κλάση αυτή παρέχει μεθόδους για την αναπαράσταση των υπολογιστικών συσκευών στην προσομοίωση. Ο κόμβος θεωρείται ως ένας Η/Υ, στον οποίο θα προσθέσουμε λειτουργικότητα, όπως εφαρμογές, στοίβες πρωτοκόλλων και κάρτες επέκτασης με τους σχετικούς οδηγούς.

2.3.2 Εφαρμογή

Τυπικά, το λογισμικό ενός υπολογιστή χωρίζεται σε δύο ευρείες κατηγορίες. Το λογισμικό συστήματος, που διαχειρίζεται τους διάφορους πόρους του υπολογιστή, όπως η μνήμη, οι κύκλοι του επεξεργαστή, ο δίσκος, το δίκτυο, κ.λπ. και το λογισμικό εφαρμογών, το οποίο χρησιμοποιεί τους πόρους αυτούς μέσω του λογισμικού συστήματος για την εκτέλεση κάποια εργασίας που είναι χρήσιμη για το χρήστη. Στο ns-3, δεν υπάρχει ουσιαστικά η έννοια του λειτουργικού συστήματος, υπάρχει μόνο η έννοια της εφαρμογής. Όπως οι εφαρμογές λογισμικού που τρέχουν σε υπολογιστές ώστε να εκτελούν εργασίες στον πραγματικό κόσμο, οι εφαρμογές του ns-3 τρέχουν σε κόμβους για την εκτέλεση των προσομοιώσεων.

Στο ns-3, η βασική αφαίρεση για ένα πρόγραμμα χρήστη που δημιουργεί κάποια κίνηση δεδομένων που πρόκειται να προσομοιωθεί, είναι η εφαρμογή. Αυτή η έννοια αναπαρίσταται σε C++ με την κλάση Application. Η κλάση Application παρέχει μεθόδους για τη διαχείριση στις προσομοιώσεις, των προκαθορισμένων αναπαραστάσεων για τις εφαρμογές επιπέδου χρήστη που ορίστηκαν στο ns-3. Ο

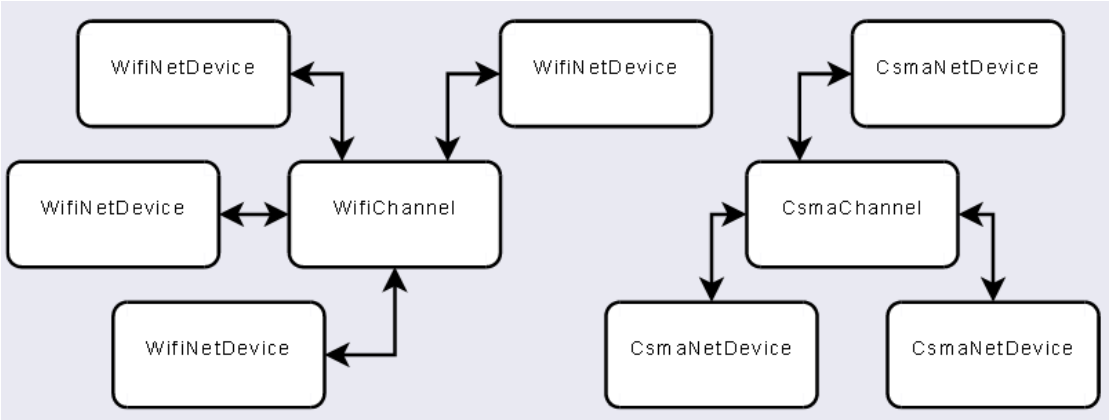
προγραμματιστής απαιτείται να εξειδικεύσει την κλάση Application, σύμφωνα με την αντικειμενοστραφή λογική, για τη δημιουργία νέων εφαρμογών χρήστη.

2.3.3 Κανάλι

Στον πραγματικό κόσμο, μπορεί κανείς να συνδέσει έναν υπολογιστή σε ένα δίκτυο. Συνήθως το φυσικό μέσο επί του οποίου κινούνται τα δεδομένα αυτά, στα δίκτυα, ονομάζεται κανάλι. Στον κόσμο της προσομοίωσης του ns-3, ένας κόμβος συνδέεται σε ένα αντικείμενο που αντιπροσωπεύει το κανάλι επικοινωνίας.

Η βασική αφαίρεση του υποδικτύου επικοινωνίας στο ns-3 ονομάζεται κανάλι και αναπαρίσταται σε C++ με την κλάση Channel. Η κλάση Channel παρέχει μεθόδους για τη διαχείριση των αντικειμένων που αναπαριστούν τα υποδικτύα και τη σύνδεση κόμβων σε αυτά. Τα κανάλια μπορούν επίσης να εξειδικευτούν από τον προγραμματιστή, κατά την αντικειμενοστραφή λογική. Το εξειδικευμένο κανάλι, μπορεί να προσομοιώσει κάτι τόσο απλό όσο ένα χάλκινο σύρμα, μέχρι κάτι τόσο περίπλοκο όπως ένα μεγάλο Ethernet switch ή ένα τρισδιάστατο χώρο γεμάτο με εμπόδια στην περίπτωση των ασύρματων δικτύων.

Κάποιες από τις συνήθως χρησιμοποιούμενες εξειδικεύσεις της κλάσης Channel είναι οι: CsmaChannel, PointToPointChannel και WifiChannel. Η CsmaChannel, για παράδειγμα, μοντελοποιεί ένα υποδίκτυο που λειτουργεί με την μέθοδο CSMA, το οποίο παρέχει λειτουργικότητα παρόμοια με αυτή του Ethernet.



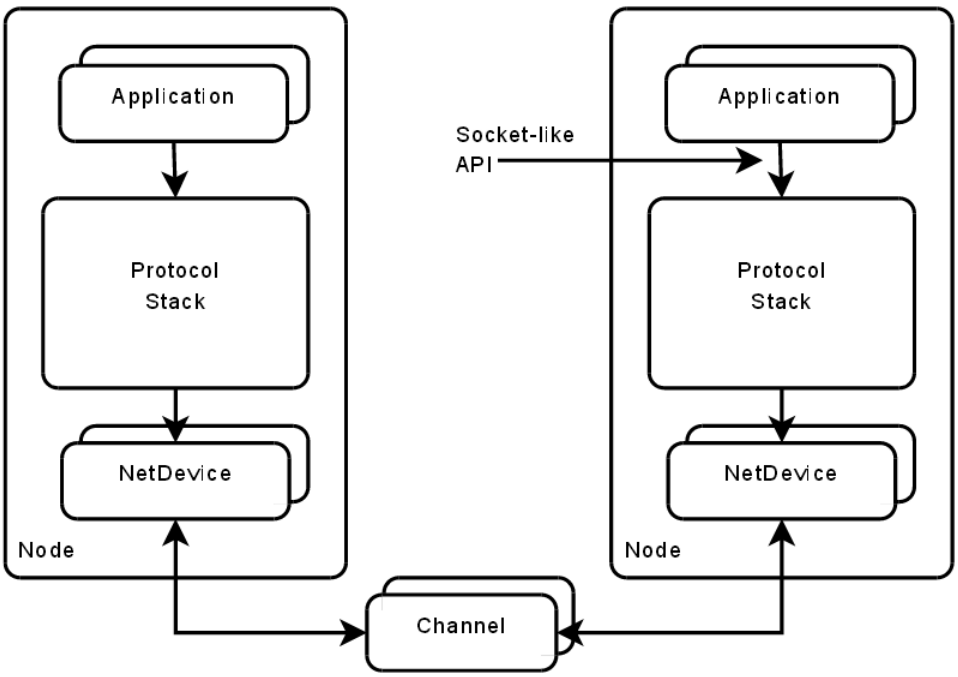
Εικόνα 10 – Το μοντέλο σύνδεσης καναλιού – συσκευών [6]

2.3.4 Συσκευή δικτύου

Για να συνδεθεί ένας υπολογιστής στο δίκτυο απαιτείται να έχει εγκατασταθεί σε αυτόν μια περιφερειακή κάρτα διασύνδεσης δικτύου (Network Interface Card - NIC) ή να έχει ενσωματωμένο αυτό το υλικό διασύνδεσης δικτύου. Μια NIC χρειάζεται για να λειτουργήσει το αντίστοιχο λογισμικό οδήγησης (drivers), που ελέγχει το υλικό. Στα λειτουργικά συστήματα Unix και Linux, ένα τμήμα περιφερειακού υλικού, χαρακτηρίζεται ως μια συσκευή. Οι συσκευές ελέγχονται μέσω προγραμμάτων οδήγησης συσκευών, και οι συσκευές δικτύου που ελέγχονται από οδηγούς συσκευών δικτύου, καλούνται γενικά net devices. Για τις συσκευές αυτές χρησιμοποιούνται ονόματα όπως eth0, eth1, κ.τ.λ..

Στο ns-3, η αφαίρεση της συσκευής δικτύου (Net device) καλύπτει τόσο το λογισμικό οδήγησης, όσο και το υλικό διασύνδεσης που προσομοιώνεται. Οι συσκευές δικτύου «εγκαθίστανται» σε ένα κόμβο, ώστε να μπορεί, κατά την προσομοίωση, να επικοινωνήσει με άλλους κόμβους μέσω των καναλιών, ακριβώς όπως ένας πραγματικός υπολογιστής μπορεί να συνδέεται σε περισσότερα από ένα κανάλια μέσω πολλαπλών συσκευών διασύνδεσης.

Η συσκευή δικτύου, αναπαρίσταται σε C++ με την κλάση NetDevice. Η κλάση NetDevice παρέχει μεθόδους για τη διαχείριση των συνδέσεων με αντικείμενα κόμβου και αντικείμενα καναλιού. Η κλάση αυτή μπορεί να εξειδικευθεί από τον προγραμματιστή σύμφωνα με την αντικειμενοστραφή λογική, για την επίτευξη εξειδικευμένης λειτουργικότητας. Κάποιες από τις συνήθως χρησιμοποιούμενες εξειδικεύσεις της κλάσης NetDevice είναι οι: CsmaNetDevice, PointToPointNetDevice και WifiNetDevice.



Εικόνα 11 - Το βασικό μοντέλο δικτύου του ns-3 [6]

2.4 Οι κλάσεις βοηθοί

Δεδομένου ότι η σύνδεση δικτυακών συσκευών σε κόμβους, δικτυακών συσκευών σε Κανάλια, η απόδοση διευθύνσεων IP, κλπ., είναι ιδιαίτερα συχνές εργασίες στο ns-3, παρέχονται κάποιοι βοηθοί τοπολογίας, για να κάνουν αυτές τις εργασίες όσο το δυνατόν ευκολότερες. Για παράδειγμα, απαιτούνται πολλές διαφορετικές λειτουργίες του πυρήνα του ns-3 για τη δημιουργία μιας δικτυακής συσκευής, τον ορισμό μιας διεύθυνσης MAC, την εγκατάσταση αυτής της συσκευής σε έναν κόμβο, τη ρύθμιση της στοίβας πρωτοκόλλου του κόμβου και στη συνέχεια τη σύνδεση της συσκευής δικτύωσης σε ένα κανάλι. Ακόμα περισσότερες λειτουργίες απαιτούνται για τη σύνδεση πολλών συσκευών σε πολλαπλά κανάλια και στη συνέχεια για τη διασύνδεση μεμονωμένων δικτύων.

Το ns-3 παρέχει αντικείμενα βοηθούς, που συνδυάζουν τις πολλές διαφορετικές εργασίες, σε ένα εύκολο στη χρήση μοντέλο δημιουργίας τοπολογιών δικτύου, χωρίς να απαιτείται απευθείας πρόσβαση στο API του πυρήνα του ns-3.

Κάποιες από τις βασικές κλάσεις βοηθών τοπολογίας, είναι οι: PointToPointHelper, InternetStackHelper, Ipv4AddressHelper, YansWifiChanelHelper, YansWifiPhyHelper και WifiHelper.

Τέτοιες κλάσεις βοηθοί παρέχονται επίσης και για την δημιουργία και τη ρύθμιση των εφαρμογών που θα δημιουργούν την κίνηση δεδομένων στην προσομοίωση (UdpEchoServerHelper, UdpEchoClientHelper, OnOffApplicationHelper), καθώς και για την ρύθμιση μοντέλων δρομολόγησης (Ipv4GlobalRoutingHelper) ή κινητικότητας των σταθμών σε ασύρματα δίκτυα (MobilityHelper).

2.5 Το 802.11 WiFi στο ns-3

Οι κόμβοι του ns-3 μπορεί να περιέχουν μια συλλογή από αντικείμενα NetDevice, όπως ένας πραγματικός υπολογιστής περιέχει ξεχωριστές κάρτες NIC για Ethernet, WiFi, Bluetooth, κ.λπ.. Με την προσθήκη αντικειμένων WifiNetDevice στους κόμβους του ns-3, μπορεί κανείς να δημιουργήσει μοντέλα δίκτυων υποδομής και δίκτυων ad hoc, βασισμένα στο 802.11. Επίσης στο ns-3, οι κόμβοι μπορούν να έχουν εγκατεστημένες πολλαπλές συσκευές WifiNetDevices, σε ξεχωριστά κανάλια, και μια WifiNetDevice μπορεί να συνυπάρχει με άλλους τύπους συσκευών.

2.5.1 Επισκόπηση του μοντέλου

Το ns-3 παρέχει μοντέλα που υποστηρίζουν τις παρακάτω πτυχές του 802.11:

- Βασική λειτουργία IEEE 802.11 DCF με λειτουργίες δικτύου ad hoc και υποδομής.
- Φυσικά επίπεδα για τα 802.11a, 802.11b και 802.11g.
- QoS-EDCA και τις αλλαγές στις ουρές αναμονής του 802.11e.

- Διάφορα μοντέλα απώλειας σήματος (propagation loss) συμπεριλαμβανομένων των Nakagami, Rayleigh, Friis, LogDistance, FixedRss, Random.

- Δύο μοντέλα καθυστέρησης διάδοσης (propagation delay), ένα βασισμένο στην απόσταση και ένα τυχαίο.

- Διάφορους αλγορίθμους ελέγχου του ρυθμού μετάδοσης συμπεριλαμβανομένων των Aarf, Arf, Cara, Onoe, Rraa, ConstantRate, και Minstrel.

- Το IEEE 802.11s (mesh).

Το σύνολο των μοντέλων που παρέχονται στο ns-3, προσπαθεί να παρέχει μια ακριβή υλοποίηση του επιπέδου MAC της προδιαγραφής 802.11 και να παράσχει ένα όχι και τόσο αργό μοντέλο του επιπέδου PHY της προδιαγραφής 802.11a.

Η υλοποίηση είναι τμηματική και παρέχει περίπου τέσσερα επίπεδα μοντέλων:

- Τα μοντέλα φυσικού επιπέδου PHY
- Τα μοντέλα MAC low (υλοποιούν τις DCF και EDCAF)
- Τα μοντέλα MAC high (υλοποιούν τα beacon, probing, και association)
- Τους αλγορίθμους ελέγχου ρυθμού μετάδοσης που χρησιμοποιούνται από τα μοντέλα MAC low.

Υπάρχουν σήμερα τρία μοντέλα MAC low που παρέχουν τα τρία στοιχεία της τοπολογίας του Wi-Fi (μη mesh):

- Σημείο Πρόσβασης (AP) (υλοποιείται στην κλάση ns3::ApWifiMac)

- Σταθμός (non-AP STA) (υλοποιείται στην κλάση ns3::StaWifiMac) και

- Σταθμός (STA) σε ένα Ανεξάρτητο BSS (IBSS - που συχνά αναφέρεται ως δίκτυο ad hoc (υλοποιείται στην κλάση ns3::AdhocWifiMac).

Η απλούστερη από αυτές είναι η ns3::AdhocWifiMac, η οποία υλοποιεί ένα Wi-Fi MAC, όπου δεν εκτελείται κανενός είδους beacon, probing η association. Η κλάση ns3::StaWifiMac υλοποιεί μια μηχανή καταστάσεων ενεργού probing και association, που χειρίζεται το αυτόματο re-association κάθε φορά που χάνονται πάρα πολλά beacons. Επίσης, η κλάση ns3::ApWifiMac υλοποιεί ένα AP που παράγει περιοδικά beacons και αποδέχεται κάθε αίτημα για association.

Αυτά τα τρία μοντέλα του επιπέδου MAC high μοιράζονται ένα κοινό γονέα, την ns3::RegularWifiMac, η οποία παρέχει, μεταξύ άλλων ρυθμίσεων του MAC, ένα χαρακτηριστικό που ονομάζεται QosSupported, το οποίο επιτρέπει την υποστήριξη λειτουργιών QoS του τύπου 802.11e/WMM. Με μοντέλα MAC τα

οποία έχουν δυνατότητες QoS, είναι δυνατόν να διεκπεραιωθεί κίνηση που ανήκει σε τέσσερις διαφορετικές κατηγορίες πρόσβασης (ACs): AC_VO για φωνητική κίνηση, AC_VI για την κυκλοφορία βίντεο, AC_BE για την κυκλοφορία καλύτερης προσπάθειας και AC_BK για την κυκλοφορία παρασκηνίου. Για να καθορίσει το MAC την κατάλληλη AC για ένα MSDU, τα πακέτα που προωθούνται κάτω σε αυτό το επίπεδο του MAC θα πρέπει να σημαίνονται με το ns3::QosTag, προκειμένου να οριστεί ένα TID (αναγνωριστικό κυκλοφορίας) διότι το πακέτο αλλιώς θα θεωρηθεί ότι ανήκει στην AC_BE.

Το επίπεδο MAC low χωρίζεται σε τρία μέρη:

- Την ns3::MacLow που φροντίζει για τις ανταλλαγές RTS/CTS/DATA/ACK.

- Τις ns3::DcfManager και ns3::DcfState οι οποίες υλοποιούν τις λειτουργίες DCF και EDCAF.

- Τις ns3::DcaTxop και ns3::EdcaTxopN που χειρίζονται την ουρά των πακέτων, τον κατακερματισμό των πακέτων, καθώς και την αναμετάδοση πακέτων, εφόσον είναι αναγκαία. Το αντικείμενο ns3::DcaTxop χρησιμοποιείται με μοντέλα MAC high, που δεν έχουν ενεργοποιημένο το QoS, και για τη μετάδοση των πλαισίων (π.χ. πλαισίων διαχείρισης), για τα οποία το πρότυπο απαιτεί ότι θα πρέπει να έχουν πρόσβαση στο μέσο με την DCF. Το ns3::EdcaTxopN χρησιμοποιείται με μοντέλα MAC high, που έχουν ενεργοποιημένο QoS, και εκτελεί τις λειτουργίες QoS, όπως τον τύπου 802.11n αποκατακερματισμό MSDU.

Υπάρχουν επίσης αρκετοί αλγόριθμοι ελέγχου του ρυθμού μετάδοσης που μπορεί να χρησιμοποιηθούν από το επίπεδο MAC low, όπως οι:

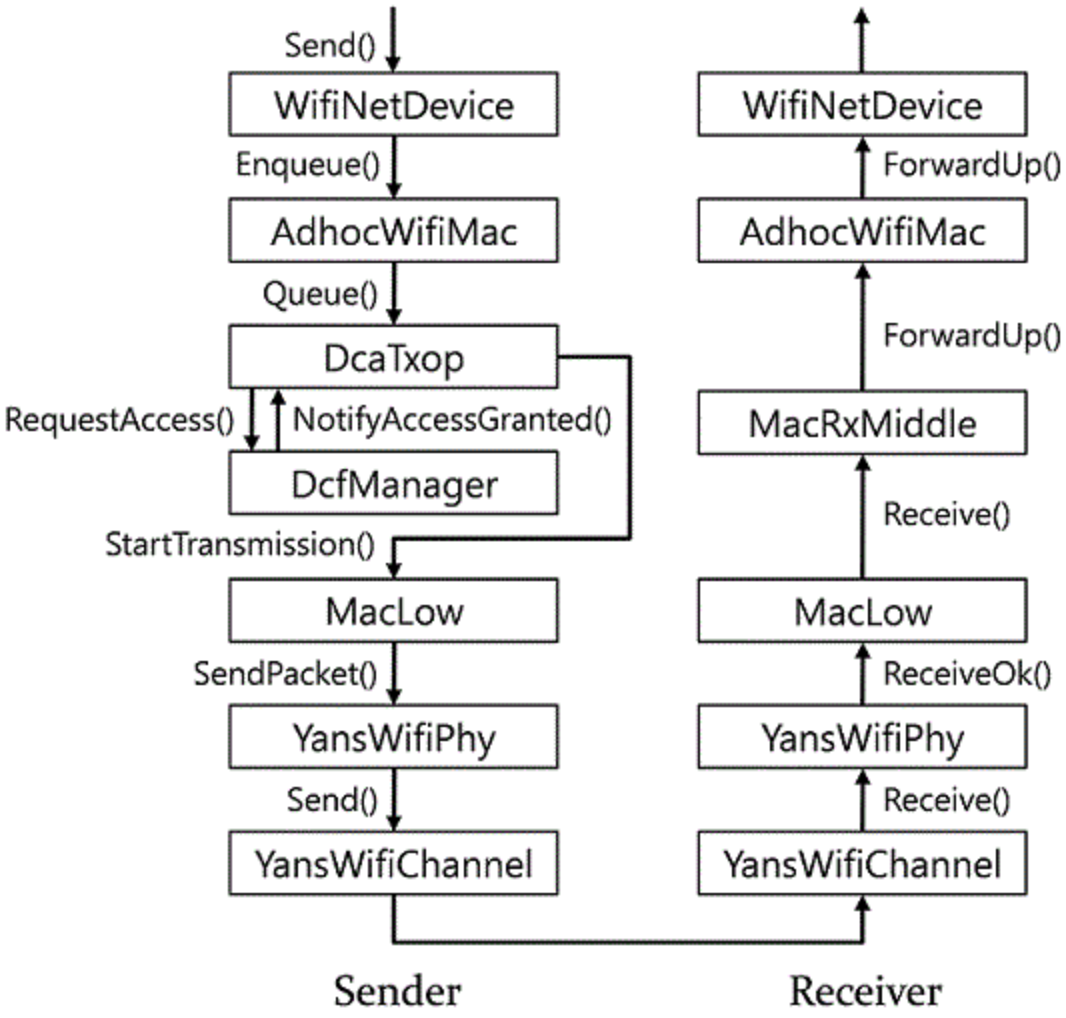
- ns3::ArfMacStations
- ns3::AArfMacStations
- ns3::IdealMacStations
- ns3::CrMacStations
- ns3::OnoeMacStations
- ns3::AmrrMacStations

Το επίπεδο PHY υλοποιεί ένα ενιαίο μοντέλο στην κλάση ns3::YansWifiPhy. Το μοντέλο φυσικού επιπέδου που υλοποιείται, περιγράφεται πλήρως στο paper με τίτλο “Yet Another Network Simulator”. [12]

2.5.2 Η διαδικασία εκπομπής και λήψης πλαισίων

Η ενότητα του ns-3 για το 802.11 αποτελείται από περίπου 50 αρχεία C++ . Η υλοποίηση της βασικής διαδικασίας μετάδοσης φαίνεται σε 6 από τα αρχεία αυτά. Πέντε κλάσεις είναι οι πιο σημαντικές για τη κατανόηση της βασικής διαδικασία μετάδοσης και λήψης, οι: DcaTxop, DcfManager, MacLow, YansWifiPhy, και YansWifiChannel .

Ένα πακέτο δεδομένων , από το ανώτερο επίπεδο, πρέπει να εισαχθεί στην ουρά. Όταν ένα πακέτο δεδομένων εισάγεται στην ουρά, οι DCA και DCF ελέγχουν αν μια άλλη μετάδοση ή λήψη πακέτου είναι σε επεξεργασία. Αν δεν υπάρχει άλλο πακέτο ή όταν η μετάδοση πακέτων έχει τελειώσει, ένα πακέτο δεδομένων εξάγεται από την ουρά και διαβιβάζεται στην κλάση MacLow. Η MacLow ελέγχει αν αυτό το πακέτο δεδομένων είναι multicast ή απαιτεί RTS ή κατακερματισμό. Μετά την εκτίμηση της διάρκειας της μετάδοσης, η MacLow προωθεί το πακέτο δεδομένων στην YansWifiPhy, η οποία το προωθεί στην YansWifiChannel. Η διαδικασία υποχώρησης (backoff), απαιτείται μόνο αν το κανάλι ήταν απασχολημένο λίγο πριν. Αυτό σημαίνει ότι αν δεν υπήρχε μετάδοση ή λήψη (δηλαδή το κανάλι είναι αδρανές), για ένα συγκεκριμένο χρονικό διάστημα (δηλαδή, κατά τη διάρκεια DIFS), το πρωτόκολλο MAC στέλνει ένα πακέτο δεδομένων αμέσως μόλις πάρει το πακέτο δεδομένων από ένα ανώτερο επίπεδο.



Εικόνα 12 - Η διαδικασία εκπομπής/λήψης πακέτων [3]

Οι βασικές διαδικασίες εκπομπής και λήψης ενός πλαισίου, στην απλούστερη περίπτωση του ad-hoc δικτύου WiFi, περιγράφονται στου δύο πίνακες που ακολουθούν.

Η βασική διαδικασία εκπομπής είναι η ακόλουθη:

Upper Layer → WifiNetDevice::Send()	# Μετέτρεψε την διεύθυνση IP σε διεύθυνση MAC address και πρόσθεσε κεφαλίδα LLC.

→ AdhocWifiMac::Enqueue()	# Καθόρισε κεφαλίδα Wifi MAC και εκτίμησε λειτουργίες του PHY (π.χ. data rate) που υποστηρίζονται από τον κόμβο προορισμού.
→ DcaTxop::Queue()	# Εισήγαγε το νέο πακέτο στην ουρά μέσω της WifiMacQueue::Enqueue().
→ DcaTxop::StartAccessIfNeeded()	# Αιτήσου πρόσβαση στο διαχειριστή DCF, αν δεν υπάρχει πακέτο σε αναμονή (π.χ. άν m_accessRequested == true).
→ DcfManager::RequestAccess()	# Εάν υπάρχει ένα άλλο πακέτο που να εκκρεμεί, σηματοδοτούν μια σύγκρουση μέσω της DcaTxop::NotifyCollision(), η οποία θα επιλέξει ένα τυχαίο αριθμό backoff και θα προσπαθήσει εκ' νέου να αποκτήσει πρόσβαση.
→ DcfManager::DoGrantAccess()	# Ελέγξτε για εσωτερική σύγκρουση ανάμεσα στις τέσσερις διαφορετικές ουρές (για την EDCA του 802.11e).
→ DcfState::NotifyAccessGranted()	# Όρισε την m_accessRequested = false και ενημέρωσε ότι χορηγήθηκε πρόσβαση στο κανάλι.
→ DcaTxop::NotifyAccessGranted()	# Εξήγαγε ένα πακέτο από την ουρά, όρισε νέο αριθμό ακολουθίας (sequence number), έλεγξε εάν αυτό το πακέτο είναι μήνυμα multicast ή αν απαιτείται κατακερματισμός ή RTS.

→ MacLow::StartTransmission()	# Αν απαιτείται RTS, καλέστε την SendRtsForPacket(). Αν όχι, καλέστε την SendDataPacket().
→ MacLow::SendDataPacket()	# Εκτίμησε τη συνολική διάρκεια ανταλλαγής, δηλ. το άθροισμα της μετάδοσης, της λήψης του ACK και/ή της επόμενης μετάδοσης (προκειμένου για Block ACK ή κατακερματισμό).
→ MacLow::ForwardDown()	# Προώθησε το πακέτο στο επίπεδο PHY.
→ YansWifiPhy::SendPacket()	# Άλλαξε την τρέχουσα κατάσταση του PHY σε TX (αυτή η εντολή θα ενημερώσει τελικά την DCF ότι αρχίζει η μετάδοση). Προώθησε το πακέτο στο κανάλι.
→ YansWifiChannel::Send()	# Εκτίμησε την ισχύ λήψης. Όρισε την καθυστέρηση (propagation delay), έτσι ώστε να προκληθεί εκτέλεση των συναρτήσεων YansWifiChannel::Receive() των γειτονικών σταθμών, μετά από ένα σύντομο χρονικό διάστημα.

Πίνακας 3 - Η βασική διαδικασία εκπομπής [2]

Η βασική διαδικασία λήψης είναι η ακόλουθη:

→ YansWifiChannel::Receive()	# Ενεργοποιείται μετά από την παρέλευση της καθυστέρησης διάδοσης (που καθορίζεται από τον αποστολέα μέσω της YansWifiChannel::

	Receive()).
→ YansWifiPhy::StartReceivePacket()	# Υπολόγισε το τρέχον επίπεδο παρεμβολών. Απέρριψε το ληφθέν πακέτο, αν η κατάσταση του PHY δεν είναι η αδράνεια ή η ισχύς λήψεως είναι κάτω από ένα όριο. Ρυθμίσε την καθυστέρηση ώστε να ενεργοποιήσει την YansWifiPhy::EndSync() μετά τη διαβίβαση του πακέτου.
→ YansWifiPhy::EndSync()	# Υπολογίστε το ποσοστό σφάλματος πακέτου (packet error rate ή PER) από το λόγο σήματος προς θόρυβο (SNR) (δηλ., σήμα ισχύος / (θόρυβος + ισχύς παρεμβολής)). Αν μια τυχαία τιμή είναι μικρότερη από την τιμή PER, το πακέτο απορρίπτεται.
→ MacLow::ReceiveOk()	# Αν το πακέτο είναι το πακέτο RTS, χρονοπρογραμμάτισε την εκτέλεση της MacLow::SendCtsAfterRts(). Αν είναι CTS, χρονοπρογραμμάτισε την ετέλεση της MacLow::SendDataAfterCts (). Αν είναι DATA, διαβίβασε το στο ανώτερο επίπεδο. Αν ο προορισμός είμαι εγώ, χρονοπρογραμμάτισε την εκτέλεση της MacLow: :SendAckAfterData().
→ MacRxMiddle::Receive()	# Ελέγξτε αν το ληφθέν πακέτο είναι διπλότυπο με τον αριθμό ακολουθίας.

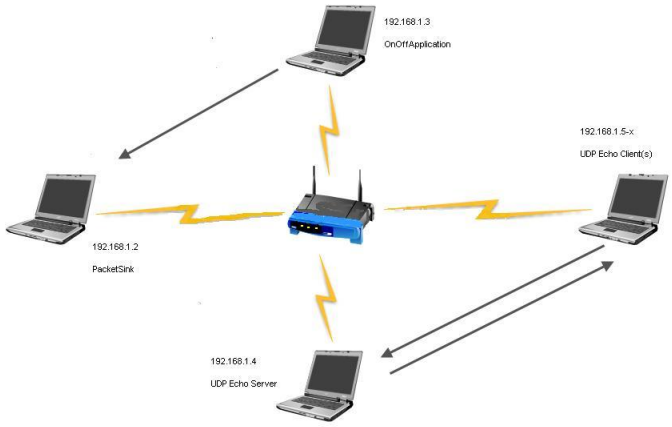
→ AdhocWifiMac::ForwardUp()	# Απλώς προώθησε το πακέτο που ελήφθη στο ανώτερο.
→ WifiNetDevice::ForwardUp() _ Upper Layer	# Εκτός και εάν ο προορισμός είναι άλλος κόμβος, προώθησε το ληφθέν πακέτο προς το ανώτερο επίπεδο.

Πίνακας 4 - Η βασική διαδικασία λήψης

ΚΕΦΑΛΑΙΟ 3 Η ΠΡΟΣΟΜΟΙΩΣΗ

3.1 Το δίκτυο που προσομοιώνεται

Για την πραγματοποίηση της προσομοίωσης θεωρήθηκε ένα δίκτυο που αποτελείται από ένα BSS, όπως αυτό του σχήματος 3.1, αποτελούμενο από ένα AP και 4 έως 35 σταθμούς, χωρίς να υπάρχει DS. Ο λόγος θεώρησης δικτύου αυτού του είδους, είναι ότι η προσομοίωση εστιάζεται στην λειτουργία του επιπέδου MAC.



Εικόνα 13 - Το βασικό δίκτυο της προσομοίωσης

Η έκδοση του ns-3 που χρησιμοποιείται για την προσομοίωση αυτή είναι η ns-3.17, εγκατεστημένη στο λειτουργικό σύστημα Linux (openSuSe 12.2)

Το μοντέλο του ns-3 για το επίπεδο MAC (MAC layer model) ρυθμίστηκε να είναι του τύπου EDCA (QoS).

Το μοντέλο κινητικότητας των σταθμών ορίστηκε να είναι το μοντέλο «Random walk 2» του ns-3, ενώ το μοντέλο κινητικότητας του Σημείου Πρόσβασης (Access Point ή AP) ορίστηκε να είναι το μοντέλο «Constant position» του ns-3 (Σταθερή θέση).

Στους σταθμούς και στο AP, δόθηκαν ιδιωτικές διευθύνσεις από υποδίκτυο 192.168.1.0/255.255.255.0, ξεκινώντας από το AP.

Δημιουργήθηκε σταθερού ρυθμού (Constant Bit Rate) κίνηση πακέτων UDP, με προτεραιότητα φωνής (AC_VO) από τον σταθμό 1 (192.168.1.3) προς τον σταθμό 0 (192.168.1.2), χρησιμοποιώντας τις κλάσεις ns3::OnOffApplication και

ns3::PacketSink. Η ρύθμιση του tag AC_VO, έγινε με τροποποίηση του κώδικά της κλάσης ns3::OnOffApplication (hard coding).

Επίσης δημιουργήθηκε κίνηση echo, χωρίς QoS tag (AC_BE) μεταξύ του σταθμού 2 (192.168.1.4), ως echo server και των σταθμών 3 έως 35 (192.168.1.5-192.168.1.x) ως echo clients, χρησιμοποιώντας τις κλάσεις ns3::UdpEchoServer και ns3::UdpEchoClient, του ns-3.

3.2 Δομή του σεναρίου προσομοίωσης (ns-3 script)

Το σενάριο προσομοίωσης, είναι ένα σενάριο ns-3, το οποίο είναι ένα απλό πρόγραμμα γραμμένο στην γλώσσα C++, από το οποίο καλούνται οι αναγκαίες κλάσεις του προσομοιωτή για την δημιουργία της τοπολογίας του δικτύου και των διάφορων λειτουργιών του.

Αρχικά δηλώνονται οι οδηγίες προς τον μεταγλωττιστή (include statements), για την εισαγωγή των απαραίτητων αρχείων κεφαλίδων (header files), των αρχείων βιβλιοθήκης του ns-3 και της C++, που θα χρησιμοποιηθούν.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/internet-module.h"
#include "ns3/qos-tag.h"
#include "math.h"
```

Ακολουθεί, σε μορφή σχολίων, μια απλή σχηματική αναπαράσταση της τοπολογίας δικτύου που προσομοιώνεται, με χαρακτήρες ASCII.

```
// Network Topology
//
//   Wifi 192.168.1.0 / 255.255.255.0
//
//   AP
```

```
// *   *   *   *  * ..... *
// |   |   |   |  |         |
// 0   0   1   2  3 ..... 35
//
```

Κατόπιν δηλώνονται οι χώροι ονομάτων που θα χρησιμοποιηθούν στο σενάριο.

```
using namespace ns3;
using namespace std;
```

Ακολουθούν οι δηλώσεις των καθολικών μεταβλητών (Global Variables) του σεναρίου.

```
double firstRxTime = -1.0;
double lastRxTime;
uint32_t bytesTotal = 0;
double firstTxTime = -1;
double lastTxTime;
uint32_t sentPackets = 0;
uint32_t receivedPackets = 0;
double sumDelay = 0;
double delayValues [10000];
uint32_t valueCounter = 0;
```

Στη συνέχεια ορίζονται οι δύο συναρτήσεις (functions) του σεναρίου προσομοίωσης, οι οποίες χρησιμοποιώντας την λειτουργικότητα tracing του ns-3, θα μας δώσουν:

Η πρώτη (TxTrace) τους χρόνους αποστολής του πρώτου και του τελευταίου πακέτου, καθώς και τον αριθμό των απεσταλμένων πακέτων,

Και η δεύτερη (SinkRxTrace) τους χρόνους λήψης του πρώτου και του τελευταίου πακέτου, τον αριθμό των ληφθέντων πακέτων, τον συνολικό όγκο των ληφθέντων πακέτων, καθώς και υπολογισμό του αθροίσματος των καθυστερήσεων (sumDelay) και του αριθμού των επιμέρους τιμών καθυστερήσεων των πακέτων.

```
void TxTrace (Ptr<const Packet> pkt)
{
 if (firstTxTime < 0)
   firstTxTime = Simulator::Now().GetSeconds();

 lastTxTime = Simulator::Now().GetSeconds();
 sentPackets++;
}

void SinkRxTrace (Ptr<const Packet> pkt, const Address &addr)
{
 if (firstRxTime < 0)
   firstRxTime = Simulator::Now().GetSeconds();

 lastRxTime = Simulator::Now().GetSeconds();
 bytesTotal += pkt->GetSize();
 receivedPackets++;

 double delay = lastRxTime - lastTxTime;
 sumDelay = sumDelay + ( delay );

delayValues[valueCounter] = delay;

 valueCounter = valueCounter + 1;
}
```

Στη συνέχεια ακολουθεί το κυρίως σενάριο της προσομοίωσης, στο οποίο δημιουργείται η τοπολογία του δικτύου, και η κίνηση δεδομένων. Πρόκειται για μία συνάρτηση main της γλώσσας C++, στην οποία δηλώνονται οι μεταβλητές που καθορίζουν τον χρόνο προσομοίωσης (simTime), το βαθμό αναλυτικότητας των μηνυμάτων εξόδου (verbose) και τον αριθμό των σταθμών του δικτύου (nSta) και ζητούνται από τον πίνακα των ορισμάτων εισόδου (argv[]) τιμές για τις δύο τελευταίες.

```
int main (int argc, char *argv[])
```

```
{
  bool verbose = false;
  double simTime = 60.0;
  uint32_t nSta = 4;

 CommandLine cmd;
 cmd.AddValue ("nSta", "Number of wifi STA devices", nSta);
 cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);

 cmd.Parse (argc,argv);


 if (verbose)
  {
   LogComponentEnable ("OnOffApplication", LOG_LEVEL_INFO);
   LogComponentEnable ("PacketSink", LOG_LEVEL_ALL);
  }
```

Κατόπιν Δημιουργείται ένας περιέκτης (Container) για AP και ένα AP μέσα σε αυτόν, καθώς και ένα Container για τους σταθμούς, καθώς και nSta σταθμοί μέσα σε αυτόν.

```
 NodeContainer wifiStaNodes;
 wifiStaNodes.Create (nSta);
 NodeContainer wifiApNode;
 wifiApNode.Create(1);
```

Με τη χρήση των κλάσεων βοηθών ns3::YansWifiChannelHelper και ns3::YansWifiPhyHelper, ρυθμίζονται οι συσκευές του φυσικού επιπέδου και το φυσικό μέσο (κανάλι) που θα τις συνδέει, όπως ορίζονται από τα μοντέλα YansWifiChannel και YansWifiPhy του ns-3 .

```
 YansWifiChannelHelper channel = YansWifiChannelHelper::Default();
 YansWifiPhyHelper phy = YansWifiPhyHelper::Default();
 phy.SetChannel (channel.Create());
```

Αφότου έχει ρυθμιστεί το φυσικό επίπεδο (PHY layer), δημιουργείται το επίπεδο MAC, με τη χρήση της κλάσης ns3::WifiHelper, καθώς και της ns3:: QosWifiMacHelper, προκειμένου το επίπεδο MAC να είναι τύπου QoS (802.11e - EDCA).

```
WifiHelper wifi = WifiHelper::Default();
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

QosWifiMacHelper mac = QosWifiMacHelper::Default ();

Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
"Ssid", SsidValue (ssid),
"ActiveProbing", BooleanValue (false));
```

Μετά την ρύθμιση των παραμέτρων των σταθμών, τόσο για το φυσικό, όσο και για το επίπεδο MAC, δημιουργούνται συσκευές με τα παραπάνω χαρακτηριστικά για κάθε σταθμό και συνδέονται στο κοινό φυσικό μέσο.

```
NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);
```

Παρομοίως ρυθμίζεται το επίπεδο MAC για το AP και δημιουργείται το AP, συνδεδεμένο ίδιο φυσικό μέσο.

```
mac.SetType ("ns3::ApWifiMac",
        "Ssid", SsidValue (ssid),
        "BeaconGeneration", BooleanValue (true),
       "BeaconInterval", TimeValue (Seconds (2.5)));

NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, wifiApNode);
```

Στη συνέχεια ορίζεται το απαραίτητα μοντέλα κινητικότητας των σταθμών και του AP του δικτύου, ώστε το AP να παραμένει ακίνητο και οι σταθμοί να

κινούνται με τυχαίο τρόπο μέσα σε χώρο, διαστάσεων που ορίζονται από τις παραμέτρους.

```
MobilityHelper mobility;

mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                               "MinX", DoubleValue (0.0),
                               "MinY", DoubleValue (0.0),
                               "DeltaX", DoubleValue (5.0),
                               "DeltaY", DoubleValue (10.0),
                               "GridWidth", UintegerValue (3),
                               "LayoutType", StringValue ("RowFirst"));

mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
                           "Bounds", RectangleValue (Rectangle (-200, 200, -200, 200)));
mobility.Install (wifiStaNodes);

mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (wifiApNode);
```

Αφού ήδη έχουν δημιουργηθεί οι κόμβοι, οι συσκευές και το φυσικό μέσο του δικτύου, εγκαθίσταται σε αυτούς με το παρακάτω κομμάτι κώδικα η λειτουργικότητα των ανωτέρων επιπέδων του OSI (Internet Protocol Stack) με την κλάση βοηθό ns3::InternetStackHelper. Κατόπιν με την ns3::Ipv4AddressHelper ανατίθενται διευθύνσεις IPv4, από το υποδίκτυο 192.168.1.0/255.255.255.0, στο AP και σταθμούς του δικτύου.

```
InternetStackHelper stack;
stack.Install (wifiApNode);
stack.Install (wifiStaNodes);

Ipv4AddressHelper address;

address.SetBase ("192.168.1.0", "255.255.255.0");
```

```
Ipv4InterfaceContainer staInterfaces;
Ipv4InterfaceContainer apInterfaces;

apInterfaces = address.Assign (apDevices);
staInterfaces = address.Assign (staDevices);
```

Μετά την δημιουργία της τοπολογίας του δικτύου και της ρύθμισης των μοντέλων του ns-3 που χρησιμοποιούνται στο δίκτυο, ακολουθεί η δημιουργία κίνησης πακέτων μεταξύ των σταθμών.

Πρώτα δημιουργείται κίνηση πακέτων UDP, με την χρήση της κλάσης ns3::OnOffApplication η οποία προσομοιώνει μία εφαρμογή παραγωγής πακέτων και της κλάσης ns3::PacketSink η οποία προσομοιώνει μια εφαρμογή αποδέκτη πακέτων. Για την δημιουργία και την ρύθμιση των παραπάνω εφαρμογών της προσομοίωσης, χρησιμοποιούνται οι αντίστοιχες κλάσεις βοηθοί, ns3::OnOffApplication και ns3::PacketSinkHelper. Μέσω των βοηθών αυτών, δίνονται οι απαραίτητες παράμετροι διευθύνσεων IP και θύρας UDP, ώστε η κίνηση να κατευθύνεται από τον σταθμό 1 προς τον σταθμό 0. Οι εφαρμογές αυτές ανατίθενται στους περιέκτες (containers) τύπου ApplicationContainer, sourceApps και sinkApps, αντίστοιχα και ορίζεται ως χρόνος έναρξης τους το 1^{ο} δευτερόλεπτο της προσομοίωσης και λήξης το τελευταίο δευτερόλεπτο (simTime).

```
NS_LOG_INFO("Create Applications");

uint16_t Port = 9;

Address remoteAddress(InetSocketAddress(staInterfaces.GetAddress(0),
Port));

OnOffHelper onOffHelper ("ns3::UdpSocketFactory", remoteAddress);
ApplicationContainer sourceApps=onOffHelper.Install(wifiStaNodes.Get(1));

sourceApps.Start (Seconds (1.0));
sourceApps.Stop (Seconds (simTime));

Address sinkAddress(InetSocketAddress(staInterfaces.GetAddress(0),Port));
```

```
PacketSinkHelper packetSinkHelper("ns3::UdpSocketFactory",sinkAddress);
ApplicationContainer sinkApps = packetSinkHelper.Install (wifiStaNodes.Get(0));

sinkApps.Start (Seconds (1.0));
sinkApps.Stop (Seconds (simTime));
```

Ακολούθως δημιουργείται κίνηση πακέτων UDP, με την χρήση της κλάσης ns3::UdpEchoClient η οποία προσομοιώνει μία εφαρμογή παραγωγής πακέτων και της κλάσης ns3::UdpEchoServer η οποία προσομοιώνει μια εφαρμογή αποδοχής και επανεκπομπής των πακέτων. Για την δημιουργία και την ρύθμιση των παραπάνω εφαρμογών της προσομοίωσης, χρησιμοποιούνται οι αντίστοιχες κλάσεις βοηθοί, ns3:: UdpEchoClient και ns3:: UdpEchoServer.

Μέσω των βοηθών αυτών, δίνονται οι απαραίτητες παράμετροι διευθύνσεων IP και θύρας UDP, ώστε η κίνηση να κατευθύνεται από τον σταθμό/ούς 3 έως nSta (echo clients), προς τον σταθμό 2 (echo server). Οι εφαρμογές αυτές ανατίθενται στους περιέκτες (containers) τύπου ApplicationContainer, serverApps και clientApps, αντίστοιχα και ορίζεται ως χρόνος έναρξης τους το 1^{ο} δευτερόλεπτο της προσομοίωσης και λήξης το τελευταίο δευτερόλεπτο (simTime).

```
uint16_t echoPort = 7;

UdpEchoServerHelper echoServer (echoPort);

ApplicationContainer serverApps=echoServer.Install(wifiStaNodes.Get(2));

serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (simTime));

UdpEchoClientHelper echoClient (staInterfaces.GetAddress(2), echoPort);
echoClient.SetAttribute ("MaxPackets", UintegerValue (10));//1
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
```

```
for(uint32_t i = 3; i < nSta; i++){
ApplicationContainer clientApps = echoClient.Install(wifiStaNodes.Get(i));

clientApps.Start (Seconds (1.0));
clientApps.Stop (Seconds (simTime));
```

Ο ορισμός του χρόνου προσομοίωσης, σε δευτερόλεπτα, δίνεται με την παράμετρο simTime, στην συνάρτηση Stop() του προσομοιωτή και ακολουθεί η ενεργοποίηση της λειτουργίας καταγραφής (sniffing) των πλαισίων (frames) στο interface του AP και της αποθήκευσης σε αρχεία τύπου ".pcap" με ονομασία που θα αρχίζει από "QoSSim".

```
Simulator::Stop (Seconds (simTime));

phy.EnablePcap ("QoSSim", apDevices.Get (0));
```

Ακολούθως και πριν την κλήση της μεθόδου έναρξης της προσομοίωσης, Run(), δηλώνονται δύο δυναμικές κλήσεις των μεθόδων SinkRxTrace() και TxTrace(), κάθε φορά που λαμβάνει χώρα γεγονός αποστολής ή λήψης πακέτου. Οι μέθοδοι αυτοί ορίστηκαν στην αρχή του σεναρίου.

```
Config::ConnectWithoutContext("/NodeList/*/ApplicationList/*/
$ns3::PacketSink/Rx",MakeCallback(&SinkRxTrace));

Config::ConnectWithoutContext("/NodeList/*/ApplicationList/*/
$ns3::OnOffApplication/Tx",MakeCallback(&TxTrace));

Simulator::Run ();
```

Μετά την έναρξη της προσομοίωσης ακολουθούν οι υπολογισμοί των μετρικών: Μέση Καθυστέρηση (Mean Delay), Διακύμανση Καθυστέρησης (Jitter), Μέση Απόδοση (Average Troughput) και Απωλεσθέντα Πακέτα (Lost Packets) και η εμφάνιση του αποτελέσματος στην προκαθορισμένη έξοδο (Standard Output).

```
double meanDelay = sumDelay/receivedPackets;
```

```
double jitterSum = 0;
for (uint32_t i = 0; i < receivedPackets; i++){
  jitterSum = jitterSum + pow( ( delayValues[i] - meanDelay ), 2.0);
}
double jitter = sqrt(jitterSum / receivedPackets);

cout << "Num_Stations= " << nSta;
cout << " " << " Avg_Troughput= " << bytesTotal*8/(lastRxTime - firstRxTime) / 1024 << " kbits/sec";
cout << " " << " Lost_Packets= " << sentPackets - receivedPackets;
cout << " " << " Mean_Delay= " << meanDelay << " sec";
cout << " " << " Jitter= " << jitter << endl;
```

Τέλος ακολουθεί η κλήση της μεθόδου Destroy(), του προσομοιωτή, προκειμένου να ελευθερωθεί η μνήμη από τα αντικείμενα και τις δομές δεδομένων που δημιουργήθηκαν κατά την προσομοίωση και η έξοδος-επιστροφή από την main.

```
Simulator::Destroy ();
return 0;
```

3.3 Τροποποίηση της κλάσης ns3::OnOffApplication.

Προκειμένου τα πλαίσια (frames) των πακέτων που θα αποστέλλονται από την εφαρμογή OnOffApplication, να είναι προτεραιότητας (Access Category) AC_VO (Voice), απαιτείται το TID να έχει την τιμή 6 αντί της προκαθορισμένης τιμής 0, η οποία αντιστοιχεί σε προτεραιότητα AC_BE (Best Efort).

Με την προσθήκη του παρακάτω κομματιού κώδικα, στον κώδικα τη OnOffApplication, αμέσως πριν από την αποστολή του πακέτου, καθορίζεται η τιμή του TID. Με αυτό τον τρόπο, ενεργοποιώντας ή απενεργοποιώντας σε σχόλιο την ανάθεση τιμής της TID, επιλέγεται μεταξύ AC_VO και AC_BE, κατά περίπτωση.

```
QosTag qostag;
```

Qostag.SetTid(6); *// AC_VO*

//Qostag.SetTid(0); // AC_BE

Packet->AddPacketTag(qosTag);

Επίσης έγιναν οι αναγκαίες τροποποιήσεις στο αρχείο header OnOffApplication.h, οι οποίες συνίστανται στην προσθήκη:

- της οδηγίας ***#include "ns3/qos-tag.h"*** *και*
- *της δήλωσης* ***QosTag qosTag;*** στο χώρο δήλωσης των μεταβλητών

καθώς και η προσθήκη του wifi module ns-3 ως εξάρτηση στο αρχείο WSCRIPT, που βρίσκεται στον φάκελο των μοντέλων εφαρμογών του ns-3 (src/applications):

module = bld.create_ns3_module('applications', ['internet', 'config-store', 'tools', ***'wifi'****])*

3.4 Εκτέλεση της προσομοίωσης

Το σενάριο προσομοίωσης εκτελείται, διαδοχικά για κάθε τιμή του αριθμού των σταθμών (nSta), από 4 μέχρι 35, μέσω του παρακάτω σεναρίου φλοιού (bash shell script), ονομασία “run-all.sh” και με τη χρήση του τελεστή ανακατεύθυνσης “ > ”, τα αποτελέσματα καταγράφονται σε αρχείο με κατάληξη “.dat”.

```
#!/bin/bash
for ( ( i=4; i<36; i+=1 ) ); do
        ./waf --run "QoSSim --nSta=$i --verbose=0 --RngRun=9"
    done
```

Το σενάριο εκτελείται μία φορά με TID=6 (AC_VO) στον κώδικα της OnOffApplication, για την προσομοίωση κίνησης με κατηγορία προτεραιότητας φωνής (Voice) και ακολούθως μια δεύτερη φορά με TID=0 (AC_BE), για την προσομοίωση κίνησης με την προκαθορισμένη κατηγορία προτεραιότητας (Best Effort). Αποτέλεσμα των δύο εκτελέσεων, είναι δύο αρχεία, “Results-VO.dat” “Results-BE.dat”

~/tarballs/ns-allinone-3.17/ns-3.17> *Bash run-all.sh > Results-VO.dat*

και

~/tarballs/ns-allinone-3.17/ns-3.17> *Bash run-all.sh > Results-BE.dat*

ΚΕΦΑΛΑΙΟ 4 ΑΠΟΤΕΛΕΣΜΑΤΑ ΤΗΣ ΠΡΟΣΟΜΟΙΩΣΗΣ

Σε αυτό το κεφάλαιο τα αποτελέσματα δίνονται σε μορφή γραφικών παραστάσεων και προέκυψαν μετά από εκτέλεση του σεναρίου προσομοίωσης, και μετά από 9 επαναλήψεις για την κάθε επιλογή, έτσι ώστε να διασφαλιστεί η όσο το δυνατόν μεγαλύτερη ακρίβεια των αποτελεσμάτων.

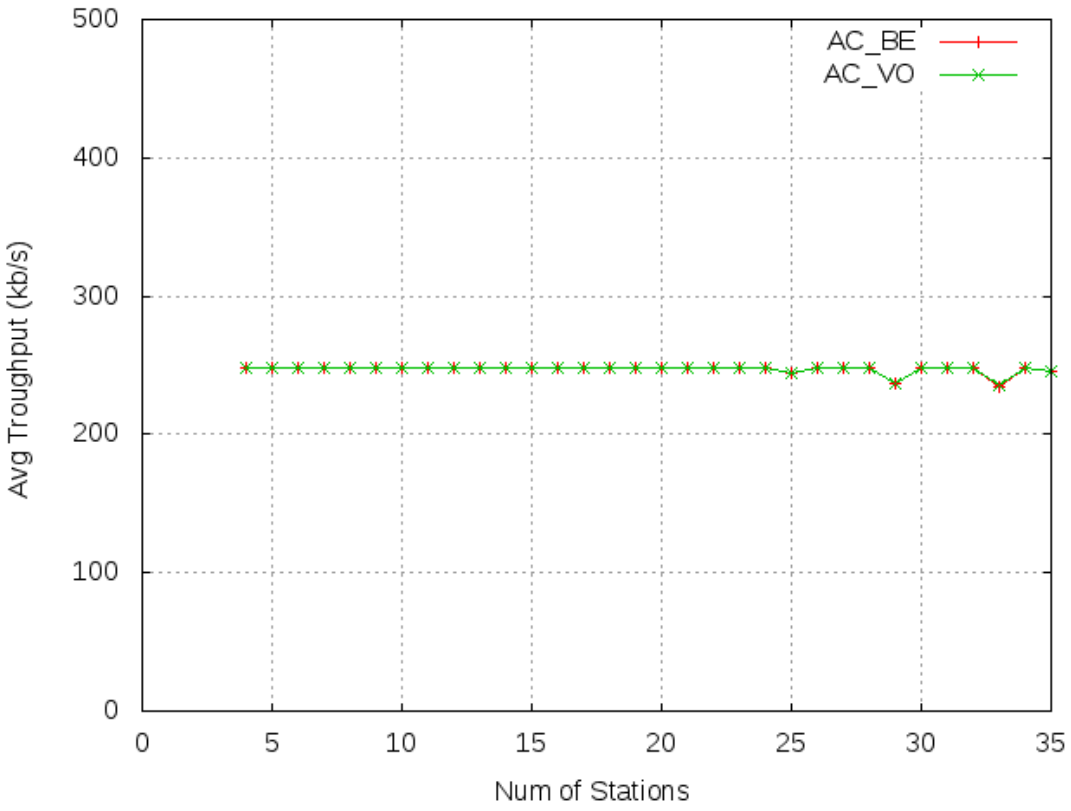
Από την εκτέλεση του σεναρίου, παρήχθησαν τα αρχεία “Results-VO.dat” και “Results-BE.dat”, για κίνηση πακέτων UDP με κατηγορία προτεραιότητας AC_VO και AC_BE αντίστοιχα.

4.1 Αποτελέσματα για τη Μέση Απόδοση (Average Throughput)

Αυξάνοντας σταδιακά τον αριθμό των σταθμών που υπάρχουν στο δίκτυο και συγκεκριμένα αυτών που ανταγωνίζονται την υπό μέτρηση κίνηση, παρατηρείται πως η ποσότητα Throughput μένει ουσιαστικά αμετάβλητη και δεν διαφοροποιείται από την χρήση προτεραιότητας AC_VO έναντι της AC_BE.

Η συμπεριφορά αυτή οφείλεται στον μικρό σχετικά αριθμό σταθμών στο δίκτυο και στο χαμηλό ρυθμό μετάδοσης που χρησιμοποιείται από το προκαθορισμένο μοντέλο του ns-3.

Τα αποτελέσματα αυτά παρουσιάζονται στο διάγραμμα που ακολουθεί.



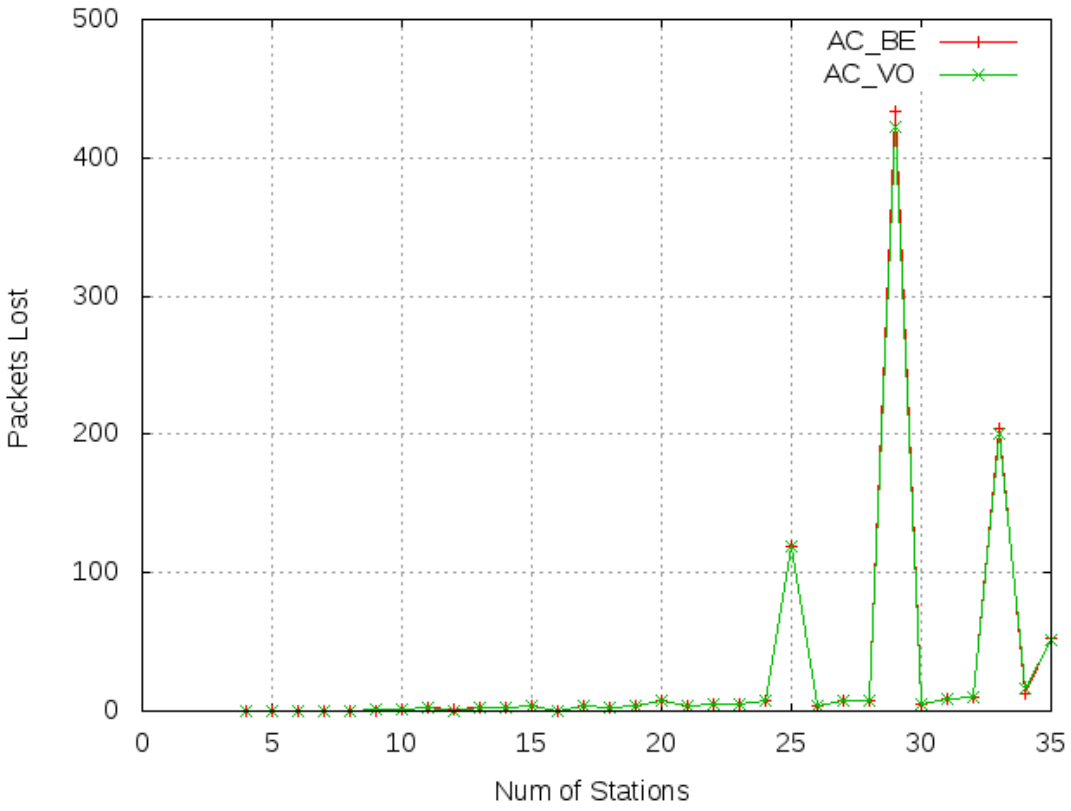
Εικόνα 14 – Διάγραμμα Μέσης Απόδοσης (Avg Throughput)

4.2 Αποτελέσματα για την Απώλεια Πακέτων (Packets Lost)

Αυξάνοντας σταδιακά τον αριθμό των σταθμών που υπάρχουν στο δίκτυο και συγκεκριμένα αυτών που ανταγωνίζονται την υπό μέτρηση κίνηση, παρατηρείται πως ο αριθμός των πακέτων που χάνονται (Packets Lost) αυξάνεται σταδιακά και παράλληλα δεν παρατηρείται διαφοροποίηση από τη χρήση προτεραιότητας AC_VO έναντι της AC_BE.

Η συμπεριφορά αυτή είναι αναμενόμενη, διότι η χρήση κατηγορίας πρόσβασης μεγαλύτερης προτεραιότητας (AC_VO) επηρεάζει μόνο την προτεραιότητα αποστολής αυτών των πλαισίων (frames) και όχι την αξιοπιστία της διαδικασίας αποστολής και λήψης.

Τα αποτελέσματα αυτά παρουσιάζονται στο διάγραμμα που ακολουθεί.



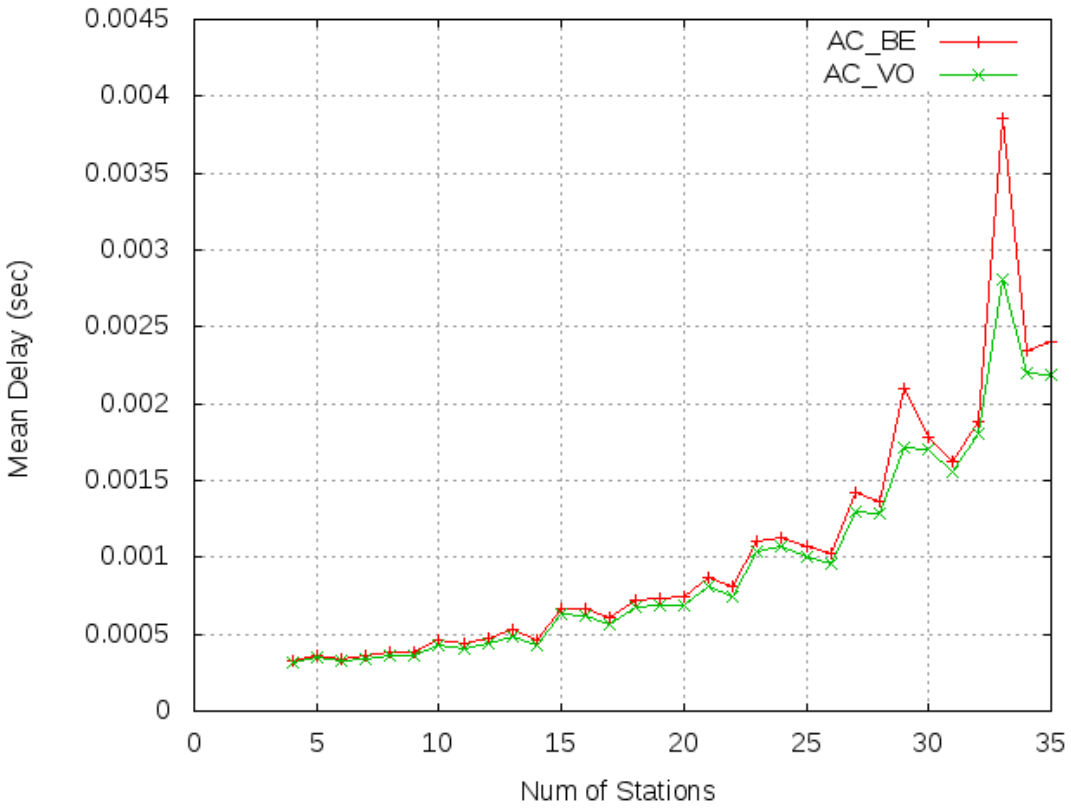
Εικόνα 15 - Διάγραμμα Απολεσθέντων Πακέτων (Packets Lost)

4.3 Αποτελέσματα για τη Μέση Καθυστέρηση (Mean Delay)

Αυξάνοντας σταδιακά τον αριθμό των σταθμών που υπάρχουν στο δίκτυο και συγκεκριμένα αυτών που ανταγωνίζονται την υπό μέτρηση κίνηση, παρατηρείται πως η ποσότητα Mean Delay αυξάνεται με ρυθμό αυξανόμενο. Παράλληλα η εν' λόγω μετρική παρουσιάζει βελτίωση όταν γίνεται χρήση προτεραιότητας AC_VO,έναντι της AC_BE.

Η συμπεριφορά αυτή δείχνει ότι εκπληρώνεται ο σκοπός της χρήσης προτεραιότητας AC_VO, που είναι η χρήση της για εφαρμογές μετάδοσης φωνής και στην οποία η καθυστέρηση επηρεάζει την ποιότητα της υπηρεσίας.

Τα αποτελέσματα αυτά παρουσιάζονται στο διάγραμμα που ακολουθεί.



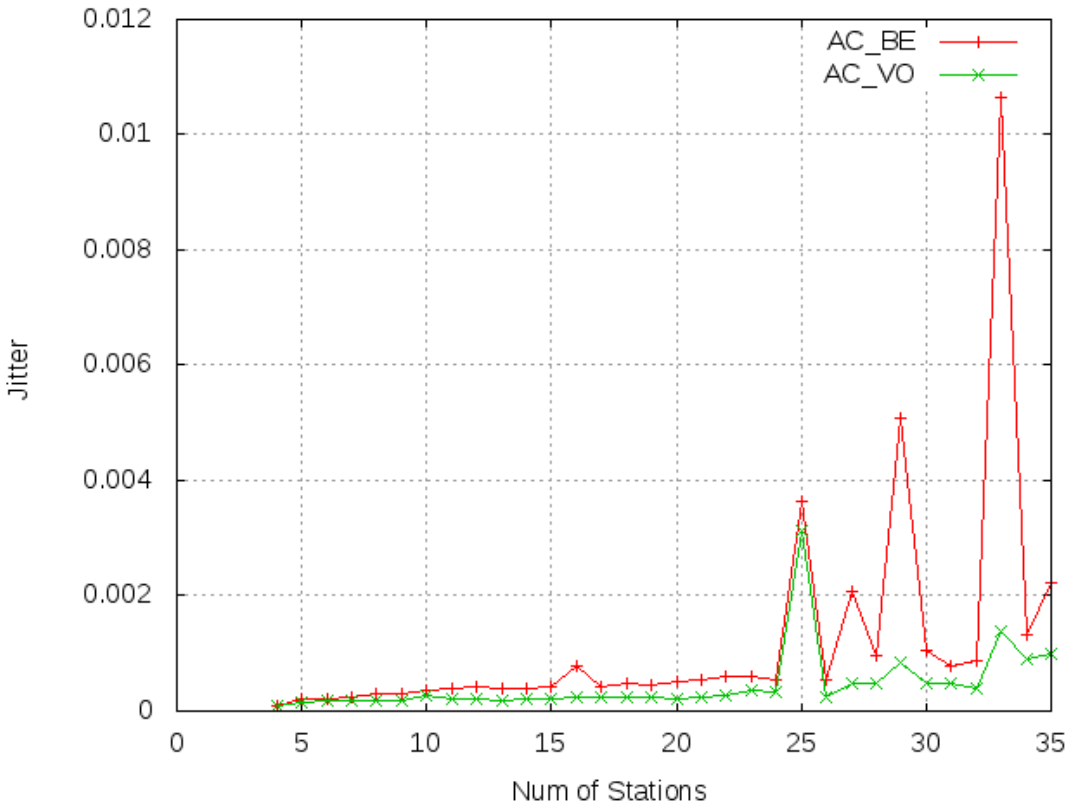
Εικόνα 16 - Διάγραμμα Μέσης Καθυστέρησης (Mean Delay)

4.4 Αποτελέσματα για τη Διακύμανση της καθυστέρησης (Jitter)

Αυξάνοντας σταδιακά τον αριθμό των σταθμών που υπάρχουν στο δίκτυο και συγκεκριμένα αυτών που ανταγωνίζονται την υπό μέτρηση κίνηση, παρατηρείται πως η διακύμανση της καθυστέρησης (Jitter) αυξάνεται σταδιακά, ως αναμενόταν. Παράλληλα η εν' λόγω μετρική παρουσιάζει σημαντική βελτίωση όταν γίνεται χρήση προτεραιότητας AC_VO,έναντι της AC_BE.

Η συμπεριφορά αυτή επίσης δείχνει την βελτίωση της ποιότητας υπηρεσίας, σε εφαρμογές μετάδοσης φωνής, με τη χρήση προτεραιότητας AC_VO, καθόσον οι αυξημένες τιμές στη διακύμανση της καθυστέρησης επηρεάζουν αρνητικά την ποιότητα του ήχου λόγω άφιξης πακέτων με λανθασμένη σειρά.

Τα αποτελέσματα αυτά παρουσιάζονται στο διάγραμμα που ακολουθεί.



Εικόνα 17 - Διάγραμμα Διακύμανσης της Καθυστέρησης (Jitter)

ΚΕΦΑΛΑΙΟ 5 ΣΥΜΠΕΡΑΣΜΑΤΑ

Τα αποτελέσματα της προσομοίωσης δείχνουν την ξεκάθαρη τάση για μείωση της μέσης καθυστέρησης (mean delay) καθώς και της διακύμανσης της καθυστέρησης αυτής (jitter), όταν χρησιμοποιείται προτεραιότητα δεδομένων φωνής (AC_VO) αντί της προκαθορισμένης προτεραιότητας (AC_BE).

Η καθυστέρηση και κυρίως η διακύμανση της, επηρεάζουν αρνητικά την ποιότητα των υπηρεσιών ζωντανής μετάδοσης φωνής ή εικόνας, κυρίως λόγω της άφιξης πακέτων εκτός σειράς, τα οποία επηρεάζουν αρνητικά σε σημαντικό βαθμό.

Συνεπώς η παροχή QoS της αναθεώρησης IEEE 802.11e, με τη δυνατότητα επιλογής μεγαλύτερης προτεραιότητας για την κυκλοφορία των ευαίσθητων εφαρμογών που υποστηρίζει, διαπιστώνεται ότι είναι αποτελεσματική στη βελτίωση της ποιότητας των υπηρεσιών ζωντανής μετάδοσης φωνής ή άλλης ομοίων απαιτήσεων εφαρμογής.

ΒΙΒΛΙΟΓΡΑΦΙΑ

ns-3 (Network Simulator 3), http://www.nsnam.org (τελευταία προσπέλαση Απρίλιος 2014)

[1] Eldad Perahia & Robert Stacey, "**Next Generation Wireless LANs**", Cambridge University Press, 2008.

[2] Juseok Kim "**WIFI on ns-3**", Απρίλιος 2011, http://www2.engr.arizona.edu/~junseok/ns3_wifi.htm

[3] ns-3 Project, "**ns-3 tutorial**", Release 3.17, Μαϊος 2013, http://www.nsnam.org/docs/release/3.17/tutorial/ns-3-tutorial.pdf

[4] ns-3 Project, "**ns-3 online manual**", Release 3.10, Μαϊος 2013, http://www.nsnam.org/docs/release/3.10/manual/singlehtml/index.html#document-wifi

[5] Mathieu Lacage, "**An ns-3 tutorial**",

http://www.nsnam.org/tutorials/ns-3-tutorial-tunis-apr09.pdf

[6] Marcus Burton, "**802.11 Arbitration**", Σεπτέμβριος 2009 http://www.cwnp.com/cmsAdmin/uploads/802-11_arbitration.pdf

[7] Montana State University, "**Introduction to 802.11 MAC**", Οκτώβριος 2005, http://www.coe.montana.edu/ee/rwolff/EE543-05/Lectures%20fall05/class%201%20MAC%2080211.pdf

[8] Andrew S.Tanenbaum & David J.Wetherall, "**Computer Networks**", 5η Έκδοση, **Κεφ.4.4**, Prentice Hall Press, 2011

[9] Intelligent Wireless Network Group, "**ns-3 Tutorial (Part IV) Wireless & Tracing System and Visualizing Results**", Μάιος 2011.

[10] **ns-3 user's forum**, https://groups.google.com/forum/#!topic/ns-3-users/fCSyT-yFs-U (τελευταία προσπέλαση Μάρτιος 2014)

[11] Mathieu Lacage, "**Yet Another Network Simulator**", 2004, http://cutebugs.net/files/wns2-yans.pdf

ΠΑΡΑΡΤΗΜΑ 1 – ΚΩΔΙΚΑΣ

Α) Το σενάριο προσομοίωσης – QoSSim.cc

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/internet-module.h"
#include "ns3/qos-tag.h"
#include "math.h"


// Default Network Topology
//
//   Wifi 192.168.1.0
//
//   AP
//   *    *    *    *   *  .......  *
//   |    |    |    |  |           |
//   0    0    1    2   3  .......  64
//

using namespace ns3;
using namespace std;
```

```
NS_LOG_COMPONENT_DEFINE ("QoSSim");

double firstRxTime = -1.0;
double lastRxTime;
uint32_t bytesTotal = 0;
double firstTxTime = -1;
double lastTxTime;
uint32_t sentPackets = 0;
uint32_t receivedPackets = 0;
double sumDelay = 0;
double delayValues [10000];
uint32_t valueCounter = 0;

void TxTrace (Ptr<const Packet> pkt)
{
  if (firstTxTime < 0)
    firstTxTime = Simulator::Now().GetSeconds();

  lastTxTime = Simulator::Now().GetSeconds();
  sentPackets++;
 }

void SinkRxTrace (Ptr<const Packet> pkt, const Address &addr)
{
  if (firstRxTime < 0)
    firstRxTime = Simulator::Now().GetSeconds();

  lastRxTime = Simulator::Now().GetSeconds();
```

```
  bytesTotal += pkt->GetSize();
  receivedPackets++;

  double delay = lastRxTime - lastTxTime;
  sumDelay = sumDelay + ( delay );


  delayValues[valueCounter] = delay;
  valueCounter = valueCounter + 1;
 }

//======== MAIN SCRIPT ==========

int main (int argc, char *argv[])
{
    bool verbose = false; //true;
    double simTime = 60.0;
    uint32_t nSta = 4;//max 64;

  CommandLine cmd;
  cmd.AddValue ("nSta", "Number of wifi STA devices", nSta);
  cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);

  cmd.Parse (argc,argv);


  if (verbose)
    {
      LogComponentEnable ("OnOffApplication", LOG_LEVEL_INFO);
```

```
    LogComponentEnable ("PacketSink", LOG_LEVEL_ALL);//INFO);
  }


  NodeContainer wifiStaNodes;
  wifiStaNodes.Create (nSta);
  NodeContainer wifiApNode;
  wifiApNode.Create(1);

  YansWifiChannelHelper channel = YansWifiChannelHelper::Default();
  YansWifiPhyHelper phy = YansWifiPhyHelper::Default();
  phy.SetChannel (channel.Create());

  WifiHelper wifi = WifiHelper::Default();
  wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

// QoS MAC
  QosWifiMacHelper mac = QosWifiMacHelper::Default ();

  Ssid ssid = Ssid ("ns-3-ssid");
  mac.SetType ("ns3::StaWifiMac",
               "Ssid", SsidValue (ssid),
               "ActiveProbing", BooleanValue (false));

  NetDeviceContainer staDevices;
  staDevices = wifi.Install (phy, mac, wifiStaNodes);

  mac.SetType ("ns3::ApWifiMac",
               "Ssid", SsidValue (ssid),
```

```
            "BeaconGeneration", BooleanValue (true),
               "BeaconInterval", TimeValue (Seconds (2.5)));

NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, wifiApNode);

MobilityHelper mobility;

mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                               "MinX", DoubleValue (0.0),
                               "MinY", DoubleValue (0.0),
                               "DeltaX", DoubleValue (5.0),
                               "DeltaY", DoubleValue (10.0),
                               "GridWidth", UintegerValue (3),
                               "LayoutType", StringValue ("RowFirst"));

mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
                           "Bounds", RectangleValue (Rectangle (-200, 200, -200, 200)));
mobility.Install (wifiStaNodes);

mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (wifiApNode);

InternetStackHelper stack;
stack.Install (wifiApNode);
stack.Install (wifiStaNodes);

Ipv4AddressHelper address;
```

```
address.SetBase ("192.168.1.0", "255.255.255.0");

Ipv4InterfaceContainer staInterfaces;
Ipv4InterfaceContainer apInterfaces;

apInterfaces = address.Assign (apDevices);
staInterfaces = address.Assign (staDevices);

// ΔΗΜΙΟΥΡΓΙΑ ΚΙΝΗΣΗΣ ΠΑΚΕΤΩΝ UDP ΜΕ ΤΗΝ ΚΛΑΣΗ OnOffApplication, ΑΠΟ ΤΗ ΕΦΑΡΜΟΓΗ sourceApp ΣΤΗΝ sinkApp.
NS_LOG_INFO("Create Applications");

uint16_t Port = 9;

Address remoteAddress(InetSocketAddress(staInterfaces.GetAddress(0),Port));

OnOffHelper onOffHelper ("ns3::UdpSocketFactory", remoteAddress);
ApplicationContainer sourceApps = onOffHelper.Install(wifiStaNodes.Get(1));

sourceApps.Start (Seconds (1.0));
sourceApps.Stop (Seconds (simTime));

Address sinkAddress(InetSocketAddress(staInterfaces.GetAddress(0),Port));

PacketSinkHelper packetSinkHelper("ns3::UdpSocketFactory",sinkAddress);
ApplicationContainer sinkApps = packetSinkHelper.Install (wifiStaNodes.Get(0));

sinkApps.Start (Seconds (1.0));
sinkApps.Stop (Seconds (simTime));
```

```
// non qos traffic
uint16_t echoPort = 7;

UdpEchoServerHelper echoServer (echoPort);

ApplicationContainer serverApps = echoServer.Install(wifiStaNodes.Get(2));

serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (simTime));

UdpEchoClientHelper echoClient (staInterfaces.GetAddress (2), echoPort);
echoClient.SetAttribute ("MaxPackets", UintegerValue (10));//1
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));


ApplicationContainer clientApps;
for(uint32_t i = 3; i < nSta; i++){
ApplicationContainer clientApps = echoClient.Install(wifiStaNodes.Get(i));
}

clientApps.Start (Seconds (1.0));
clientApps.Stop (Seconds (simTime));

Simulator::Stop (Seconds (simTime));

phy.EnablePcap ("QoSSim", apDevices.Get (0));
```

```
Config::ConnectWithoutContext("/NodeList/*/ApplicationList/*/$ns3::PacketSink/Rx",Make
Callback(&SinkRxTrace));



Config::ConnectWithoutContext("/NodeList/*/ApplicationList/*/$ns3::OnOffApplication/Tx",
MakeCallback(&TxTrace));


  Simulator::Run ();


  double meanDelay = sumDelay/receivedPackets;


  double jitterSum = 0;

  for (uint32_t i = 0; i < receivedPackets; i++){

    jitterSum = jitterSum + pow( ( delayValues[i] - meanDelay ), 2.0);

  }

  double jitter = sqrt(jitterSum / receivedPackets);


  cout << "Num_Stations= " << nSta;

  cout << " " << " Avg_Troughput= " << bytesTotal*8/(lastRxTime - firstRxTime) / 1024 << "
kbits/sec";

  cout << " " << " Lost_Packets= " << sentPackets - receivedPackets;

  cout << " " << " Mean_Delay= " << meanDelay << " sec";

  cout << " " << " Jitter= " << jitter << endl;


  Simulator::Destroy ();

  return 0;

}
```

Β) Η κλάση OnOffApplication – OnOffApplication.cc

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
//
// ---------- ΤΡΟΠΟΠΟΙΗΘΗΚΕ ----------
//
#include "ns3/log.h"
#include "ns3/address.h"
#include "ns3/inet-socket-address.h"
#include "ns3/inet6-socket-address.h"
#include "ns3/packet-socket-address.h"
#include "ns3/node.h"
#include "ns3/nstime.h"
#include "ns3/data-rate.h"
#include "ns3/random-variable-stream.h"
#include "ns3/socket.h"
#include "ns3/simulator.h"
#include "ns3/socket-factory.h"
#include "ns3/packet.h"
#include "ns3/uinteger.h"
#include "ns3/trace-source-accessor.h"
#include "onoff-application.h"
#include "ns3/udp-socket-factory.h"
#include "ns3/string.h"
#include "ns3/pointer.h"

NS_LOG_COMPONENT_DEFINE ("OnOffApplication");
```

```
namespace ns3 {


NS_OBJECT_ENSURE_REGISTERED (OnOffApplication);


TypeId
OnOffApplication::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::OnOffApplication")
    .SetParent<Application> ()
    .AddConstructor<OnOffApplication> ()
    .AddAttribute ("DataRate", "The data rate in on state.",
                   DataRateValue (DataRate ("500kb/s")),
                   MakeDataRateAccessor (&OnOffApplication::m_cbrRate),
                   MakeDataRateChecker ())
    .AddAttribute ("PacketSize", "The size of packets sent in on state",
                   UintegerValue (512),
                   MakeUintegerAccessor (&OnOffApplication::m_pktSize),
                   MakeUintegerChecker<uint32_t> (1))
    .AddAttribute ("Remote", "The address of the destination",
                   AddressValue (),
                   MakeAddressAccessor (&OnOffApplication::m_peer),
                   MakeAddressChecker ())
    .AddAttribute ("OnTime", "A RandomVariableStream used to pick the duration of the
'On' state.",
                   StringValue ("ns3::ConstantRandomVariable[Constant=1.0]"),
                   MakePointerAccessor (&OnOffApplication::m_onTime),
                   MakePointerChecker <RandomVariableStream>())
    .AddAttribute ("OffTime", "A RandomVariableStream used to pick the duration of the
'Off' state.",
```

```
                StringValue ("ns3::ConstantRandomVariable[Constant=1.0]"),
                MakePointerAccessor (&OnOffApplication::m_offTime),
                MakePointerChecker <RandomVariableStream>())
    .AddAttribute ("MaxBytes",
                "The total number of bytes to send. Once these bytes are sent, "
                "no packet is sent again, even in on state. The value zero means "
                "that there is no limit.",
                UintegerValue (0),
                MakeUintegerAccessor (&OnOffApplication::m_maxBytes),
                MakeUintegerChecker<uint32_t> ())
    .AddAttribute ("Protocol", "The type of protocol to use.",
                TypeIdValue (UdpSocketFactory::GetTypeId ()),
                MakeTypeIdAccessor (&OnOffApplication::m_tid),
                MakeTypeIdChecker ())
    .AddTraceSource ("Tx", "A new packet is created and is sent",
                  MakeTraceSourceAccessor (&OnOffApplication::m_txTrace))
  ;
  return tid;
}



OnOffApplication::OnOffApplication ()
  : m_socket (0),
    m_connected (false),
    m_residualBits (0),
    m_lastStartTime (Seconds (0)),
    m_totBytes (0)
{
  NS_LOG_FUNCTION (this);
```

```
}

OnOffApplication::~OnOffApplication()
{
  NS_LOG_FUNCTION (this);
}

void
OnOffApplication::SetMaxBytes (uint32_t maxBytes)
{
  NS_LOG_FUNCTION (this << maxBytes);
  m_maxBytes = maxBytes;
}

Ptr<Socket>
OnOffApplication::GetSocket (void) const
{
  NS_LOG_FUNCTION (this);
  return m_socket;
}

int64_t
OnOffApplication::AssignStreams (int64_t stream)
{
  NS_LOG_FUNCTION (this << stream);
  m_onTime->SetStream (stream);
  m_offTime->SetStream (stream + 1);
  return 2;
}
```

```
void
OnOffApplication::DoDispose (void)
{
  NS_LOG_FUNCTION (this);

  m_socket = 0;
  // chain up
  Application::DoDispose ();
}

// Application Methods
void OnOffApplication::StartApplication () // Called at time specified by Start
{
  NS_LOG_FUNCTION (this);

  // Create the socket if not already
  if (!m_socket)
    {
      m_socket = Socket::CreateSocket (GetNode (), m_tid);
      if (Inet6SocketAddress::IsMatchingType (m_peer))
        {
          m_socket->Bind6 ();
        }
      else if (InetSocketAddress::IsMatchingType (m_peer) ||
               PacketSocketAddress::IsMatchingType (m_peer))
        {
          m_socket->Bind ();
        }
```

```
      m_socket->Connect (m_peer);
      m_socket->SetAllowBroadcast (true);
      m_socket->ShutdownRecv ();

      m_socket->SetConnectCallback (
        MakeCallback (&OnOffApplication::ConnectionSucceeded, this),
        MakeCallback (&OnOffApplication::ConnectionFailed, this));
    }
  // Insure no pending event
  CancelEvents ();
  // If we are not yet connected, there is nothing to do here
  // The ConnectionComplete upcall will start timers at that time
  //if (!m_connected) return;
  ScheduleStartEvent ();
}

void OnOffApplication::StopApplication () // Called at time specified by Stop
{
  NS_LOG_FUNCTION (this);

  CancelEvents ();
  if(m_socket != 0)
    {
      m_socket->Close ();
    }
  else
    {
      NS_LOG_WARN ("OnOffApplication found null socket to close in StopApplication");
    }
```

```
}

void OnOffApplication::CancelEvents ()
{
  NS_LOG_FUNCTION (this);

  if (m_sendEvent.IsRunning ())
    { // Cancel the pending send packet event
      // Calculate residual bits since last packet sent
      Time delta (Simulator::Now () - m_lastStartTime);
      int64x64_t bits = delta.To (Time::S) * m_cbrRate.GetBitRate ();
      m_residualBits += bits.GetHigh ();
    }
  Simulator::Cancel (m_sendEvent);
  Simulator::Cancel (m_startStopEvent);
}

// Event handlers
void OnOffApplication::StartSending ()
{
  NS_LOG_FUNCTION (this);
  m_lastStartTime = Simulator::Now ();
  ScheduleNextTx ();  // Schedule the send packet event
  ScheduleStopEvent ();
}

void OnOffApplication::StopSending ()
{
  NS_LOG_FUNCTION (this);
```

```
  CancelEvents ();

  ScheduleStartEvent ();
}

// Private helpers
void OnOffApplication::ScheduleNextTx ()
{
  NS_LOG_FUNCTION (this);

  if (m_maxBytes == 0 || m_totBytes < m_maxBytes)
    {
      uint32_t bits = m_pktSize * 8 - m_residualBits;
      NS_LOG_LOGIC ("bits = " << bits);
      Time nextTime (Seconds (bits /
                              static_cast<double>(m_cbrRate.GetBitRate ()))); // Time till next packet
      NS_LOG_LOGIC ("nextTime = " << nextTime);
      m_sendEvent = Simulator::Schedule (nextTime,
                                         &OnOffApplication::SendPacket, this);
    }
  else
    { // All done, cancel any pending events
      StopApplication ();
    }
}

void OnOffApplication::ScheduleStartEvent ()
{  // Schedules the event to start sending data (switch to the "On" state)
  NS_LOG_FUNCTION (this);
```

```
  Time offInterval = Seconds (m_offTime->GetValue ());
  NS_LOG_LOGIC ("start at " << offInterval);
  m_startStopEvent = Simulator::Schedule (offInterval, &OnOffApplication::StartSending,
this);
}

void OnOffApplication::ScheduleStopEvent ()
{  // Schedules the event to stop sending data (switch to "Off" state)
  NS_LOG_FUNCTION (this);

  Time onInterval = Seconds (m_onTime->GetValue ());
  NS_LOG_LOGIC ("stop at " << onInterval);
  m_startStopEvent = Simulator::Schedule (onInterval, &OnOffApplication::StopSending,
this);
}

void OnOffApplication::SendPacket ()
{
  NS_LOG_FUNCTION (this);

  NS_ASSERT (m_sendEvent.IsExpired ());
  Ptr<Packet> packet = Create<Packet> (m_pktSize);

  //---------- ADDED FOR AC_VO ----------
  QosTag qosTag;
  qosTag.SetTid(6); // AC_VO (default TID=0, AC_BE)
  packet->AddPacketTag(qosTag);
```

```
//---------------------------------------------

m_txTrace (packet);
m_socket->Send (packet);
m_totBytes += m_pktSize;
if (InetSocketAddress::IsMatchingType (m_peer))
 {
   NS_LOG_INFO ("At time " << Simulator::Now ().GetSeconds ()
                << "s on-off application sent "
                <<  packet->GetSize () << " bytes to "
                << InetSocketAddress::ConvertFrom(m_peer).GetIpv4 ()
                << " port " << InetSocketAddress::ConvertFrom (m_peer).GetPort ()
                << " total Tx " << m_totBytes << " bytes");
 }
else if (Inet6SocketAddress::IsMatchingType (m_peer))
 {
   NS_LOG_INFO ("At time " << Simulator::Now ().GetSeconds ()
                << "s on-off application sent "
                <<  packet->GetSize () << " bytes to "
                << Inet6SocketAddress::ConvertFrom(m_peer).GetIpv6 ()
                << " port " << Inet6SocketAddress::ConvertFrom (m_peer).GetPort ()
                << " total Tx " << m_totBytes << " bytes");
 }
m_lastStartTime = Simulator::Now ();
m_residualBits = 0;
ScheduleNextTx ();
}
```

```
void OnOffApplication::ConnectionSucceeded (Ptr<Socket> socket)
{
  NS_LOG_FUNCTION (this << socket);
  m_connected = true;
}

void OnOffApplication::ConnectionFailed (Ptr<Socket> socket)
{
  NS_LOG_FUNCTION (this << socket);
}
} // Namespace ns3
```

Γ) Το αρχείο header – OnOffApplication.h

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
//
// Copyright (c) 2006 Georgia Tech Research Corporation
//
// This program is free software; you can redistribute it and/or modify
// it under the terms of the GNU General Public License version 2 as
// published by the Free Software Foundation;
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
//
// Author: George F. Riley<riley@ece.gatech.edu>
//

// ns3 - On/Off Data Source Application class
// George F. Riley, Georgia Tech, Spring 2007
// Adapted from ApplicationOnOff in GTNetS.

#ifndef ONOFF_APPLICATION_H
#define ONOFF_APPLICATION_H

#include "ns3/address.h"
#include "ns3/application.h"
#include "ns3/event-id.h"
#include "ns3/ptr.h"
#include "ns3/data-rate.h"
#include "ns3/traced-callback.h"
#include "ns3/qos-tag.h"

namespace ns3 {

class Address;
class RandomVariableStream;
class Socket;

/**
 * \ingroup applications
 * \defgroup onoff OnOffApplication
 *
 * This traffic generator follows an On/Off pattern: after
 * Application::StartApplication
 * is called, "On" and "Off" states alternate. The duration of each of
 * these states is determined with the onTime and the offTime random
 * variables. During the "Off" state, no traffic is generated.
```

```
 * During the "On" state, cbr traffic is generated. This cbr traffic is
 * characterized by the specified "data rate" and "packet size".
 */
/**
* \ingroup onoff
*
* \brief Generate traffic to a single destination according to an
*        OnOff pattern.
*
* This traffic generator follows an On/Off pattern: after
* Application::StartApplication
* is called, "On" and "Off" states alternate. The duration of each of
* these states is determined with the onTime and the offTime random
* variables. During the "Off" state, no traffic is generated.
* During the "On" state, cbr traffic is generated. This cbr traffic is
* characterized by the specified "data rate" and "packet size".
*
* Note:  When an application is started, the first packet transmission
* occurs _after_ a delay equal to (packet size/bit rate).  Note also,
* when an application transitions into an off state in between packet
* transmissions, the remaining time until when the next transmission
* would have occurred is cached and is used when the application starts
* up again.  Example:  packet size = 1000 bits, bit rate = 500 bits/sec.
* If the application is started at time 3 seconds, the first packet
* transmission will be scheduled for time 5 seconds (3 + 1000/500)
* and subsequent transmissions at 2 second intervals.  If the above
* application were instead stopped at time 4 seconds, and restarted at
* time 5.5 seconds, then the first packet would be sent at time 6.5 seconds,
* because when it was stopped at 4 seconds, there was only 1 second remaining
* until the originally scheduled transmission, and this time remaining
* information is cached and used to schedule the next transmission
* upon restarting.
*
* If the underlying socket type supports broadcast, this application
* will automatically enable the SetAllowBroadcast(true) socket option.
*/
class OnOffApplication : public Application
{
public:
  static TypeId GetTypeId (void);

  OnOffApplication ();

  virtual ~OnOffApplication();

  /**
   * \param maxBytes the total number of bytes to send
   *
   * Set the total number of bytes to send. Once these bytes are sent, no packet
   * is sent again, even in on state. The value zero means that there is no
   * limit.
   */
  void SetMaxBytes (uint32_t maxBytes);

  /**
```

```
   * \return pointer to associated socket
   */
  Ptr<Socket> GetSocket (void) const;

 /**
  * Assign a fixed random variable stream number to the random variables
  * used by this model.  Return the number of streams (possibly zero) that
  * have been assigned.
  *
  * \param stream first stream index to use
  * \return the number of stream indices assigned by this model
  */
  int64_t AssignStreams (int64_t stream);

protected:
  virtual void DoDispose (void);
private:
  // inherited from Application base class.
  virtual void StartApplication (void);    // Called at time specified by Start
  virtual void StopApplication (void);     // Called at time specified by Stop

  //helpers
  void CancelEvents ();

  void Construct (Ptr<Node> n,
                  const Address &remote,
                  std::string tid,
                  const RandomVariable& ontime,
                  const RandomVariable& offtime,
                  uint32_t size);


  // Event handlers
  void StartSending ();
  void StopSending ();
  void SendPacket ();

  Ptr<Socket>     m_socket;       // Associated socket
  Address         m_peer;         // Peer address
  bool            m_connected;    // True if connected
  Ptr<RandomVariableStream>  m_onTime;       // rng for On Time
  Ptr<RandomVariableStream>  m_offTime;      // rng for Off Time
  DataRate        m_cbrRate;      // Rate that data is generated
  uint32_t        m_pktSize;      // Size of packets
  uint32_t        m_residualBits; // Number of generated, but not sent, bits
  Time            m_lastStartTime; // Time last packet sent
  uint32_t        m_maxBytes;     // Limit total number of bytes sent
  uint32_t        m_totBytes;     // Total bytes sent so far
  EventId         m_startStopEvent;     // Event id for next start or stop event
  EventId         m_sendEvent;    // Eventid of pending "send packet" event
  bool            m_sending;      // True if currently in sending state
  TypeId          m_tid;
  TracedCallback<Ptr<const Packet> > m_txTrace;
  QosTag          qosTag;         // ADDED FOR QOS
```

```
private:
  void ScheduleNextTx ();
  void ScheduleStartEvent ();
  void ScheduleStopEvent ();
  void ConnectionSucceeded (Ptr<Socket> socket);
  void ConnectionFailed (Ptr<Socket> socket);
};

} // namespace ns3

#endif /* ONOFF_APPLICATION_H */
```

Δ) Το αρχείο WSCRIPT

```
## -*- Mode: python; py-indent-offset: 4; indent-tabs-mode: nil; coding: utf-8; -*-

def build(bld):
    module = bld.create_ns3_module('applications', ['internet', 'config-store', 'tools', 'wifi'])
    module.source = [
        'model/bulk-send-application.cc',
        'model/onoff-application.cc',
        'model/packet-sink.cc',
        'model/ping6.cc',
        'model/radvd.cc',
        'model/radvd-interface.cc',
        'model/radvd-prefix.cc',
        'model/udp-client.cc',
        'model/udp-server.cc',
        'model/seq-ts-header.cc',
        'model/udp-trace-client.cc',
        'model/packet-loss-counter.cc',
        'model/udp-echo-client.cc',
        'model/udp-echo-server.cc',
        'model/v4ping.cc',
        'helper/bulk-send-helper.cc',
        'helper/on-off-helper.cc',
        'helper/packet-sink-helper.cc',
        'helper/ping6-helper.cc',
        'helper/udp-client-server-helper.cc',
        'helper/udp-echo-helper.cc',
        'helper/v4ping-helper.cc',
        ]

    applications_test = bld.create_ns3_module_test_library('applications')
    applications_test.source = [
        'test/udp-client-server-test.cc',
        ]

    headers = bld(features='ns3header')
    headers.module = 'applications'
    headers.source = [
        'model/bulk-send-application.h',
        'model/onoff-application.h',
        'model/packet-sink.h',
        'model/ping6.h',
        'model/radvd.h',
        'model/radvd-interface.h',
        'model/radvd-prefix.h',
        'model/udp-client.h',
        'model/udp-server.h',
        'model/seq-ts-header.h',
        'model/udp-trace-client.h',
        'model/packet-loss-counter.h',
        'model/udp-echo-client.h',
```

```
    'model/udp-echo-server.h',
    'model/v4ping.h',
    'helper/bulk-send-helper.h',
    'helper/on-off-helper.h',
    'helper/packet-sink-helper.h',
    'helper/ping6-helper.h',
    'helper/udp-client-server-helper.h',
    'helper/udp-echo-helper.h',
    'helper/v4ping-helper.h',
    ]

bld.ns3_python_bindings()
```

ΠΑΡΑΡΤΗΜΑ 2 – ΑΡΧΕΙΑ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

Α) Περιεχόμενα αρχείου RESULTS-BE-35.dat

```
Num_Stations= 4  Avg_Troughput= 248.468 kbits/sec  Lost_Packets= 0
Mean_Delay= 0.000328674 sec  Jitter= 9.47518e-05
Num_Stations= 5  Avg_Troughput= 248.488 kbits/sec  Lost_Packets= 0
Mean_Delay= 0.000363441 sec  Jitter= 0.000224468
Num_Stations= 6  Avg_Troughput= 248.551 kbits/sec  Lost_Packets= 0
Mean_Delay= 0.000338523 sec  Jitter= 0.000214032
Num_Stations= 7  Avg_Troughput= 248.537 kbits/sec  Lost_Packets= 0
Mean_Delay= 0.000360995 sec  Jitter= 0.000244058
Num_Stations= 8  Avg_Troughput= 248.558 kbits/sec  Lost_Packets= 0
Mean_Delay= 0.000389056 sec  Jitter= 0.000293423
Num_Stations= 9  Avg_Troughput= 248.595 kbits/sec  Lost_Packets= 0
Mean_Delay= 0.00037886 sec  Jitter= 0.000303313
Num_Stations= 10  Avg_Troughput= 248.558 kbits/sec  Lost_Packets= 1
Mean_Delay= 0.000460358 sec  Jitter= 0.000357828
Num_Stations= 11  Avg_Troughput= 248.505 kbits/sec  Lost_Packets= 2
Mean_Delay= 0.00043846 sec  Jitter= 0.000397841
Num_Stations= 12  Avg_Troughput= 248.524 kbits/sec  Lost_Packets= 1
Mean_Delay= 0.000477227 sec  Jitter= 0.000423986
Num_Stations= 13  Avg_Troughput= 248.536 kbits/sec  Lost_Packets= 2
Mean_Delay= 0.000526124 sec  Jitter= 0.000390249
Num_Stations= 14  Avg_Troughput= 248.491 kbits/sec  Lost_Packets= 3
Mean_Delay= 0.00046148 sec  Jitter= 0.00038952
Num_Stations= 15  Avg_Troughput= 248.453 kbits/sec  Lost_Packets= 4
Mean_Delay= 0.000668215 sec  Jitter= 0.000428021
Num_Stations= 16  Avg_Troughput= 248.585 kbits/sec  Lost_Packets= 0
Mean_Delay= 0.000670017 sec  Jitter= 0.000773659
Num_Stations= 17  Avg_Troughput= 248.506 kbits/sec  Lost_Packets= 4
Mean_Delay= 0.000606545 sec  Jitter= 0.000433517
Num_Stations= 18  Avg_Troughput= 248.551 kbits/sec  Lost_Packets= 2
Mean_Delay= 0.000723109 sec  Jitter= 0.000468519
Num_Stations= 19  Avg_Troughput= 248.426 kbits/sec  Lost_Packets= 4
Mean_Delay= 0.00072954 sec  Jitter= 0.000450652
Num_Stations= 20  Avg_Troughput= 248.34 kbits/sec  Lost_Packets= 8
Mean_Delay= 0.000746672 sec  Jitter= 0.000522075
Num_Stations= 21  Avg_Troughput= 248.491 kbits/sec  Lost_Packets= 4
Mean_Delay= 0.000871305 sec  Jitter= 0.000536569
Num_Stations= 22  Avg_Troughput= 248.495 kbits/sec  Lost_Packets= 5
Mean_Delay= 0.000810709 sec  Jitter= 0.000607282
Num_Stations= 23  Avg_Troughput= 248.475 kbits/sec  Lost_Packets= 5
Mean_Delay= 0.00110282 sec  Jitter= 0.000592029
Num_Stations= 24  Avg_Troughput= 248.388 kbits/sec  Lost_Packets= 7
Mean_Delay= 0.00112637 sec  Jitter= 0.000528203
Num_Stations= 25  Avg_Troughput= 244.883 kbits/sec  Lost_Packets= 119
Mean_Delay= 0.00107313 sec  Jitter= 0.00365254
Num_Stations= 26  Avg_Troughput= 248.52 kbits/sec  Lost_Packets= 4
Mean_Delay= 0.00102215 sec  Jitter= 0.000549788
Num_Stations= 27  Avg_Troughput= 248.467 kbits/sec  Lost_Packets= 8
Mean_Delay= 0.00142421 sec  Jitter= 0.00206688
Num_Stations= 28  Avg_Troughput= 248.447 kbits/sec  Lost_Packets= 7
Mean_Delay= 0.00136898 sec  Jitter= 0.000976153
```

```
Num_Stations= 29  Avg_Troughput= 237.275 kbits/sec  Lost_Packets= 434
Mean_Delay= 0.00209286 sec  Jitter= 0.00507562
Num_Stations= 30  Avg_Troughput= 248.424 kbits/sec  Lost_Packets= 5
Mean_Delay= 0.00178665 sec  Jitter= 0.00104364
Num_Stations= 31  Avg_Troughput= 248.418 kbits/sec  Lost_Packets= 9
Mean_Delay= 0.00162751 sec  Jitter= 0.000794086
Num_Stations= 32  Avg_Troughput= 248.382 kbits/sec  Lost_Packets= 10
Mean_Delay= 0.00188334 sec  Jitter= 0.000878175
Num_Stations= 33  Avg_Troughput= 234.062 kbits/sec  Lost_Packets= 204
Mean_Delay= 0.0038538 sec  Jitter= 0.010635
Num_Stations= 34  Avg_Troughput= 248.208 kbits/sec  Lost_Packets= 13
Mean_Delay= 0.00234775 sec  Jitter= 0.00133291
Num_Stations= 35  Avg_Troughput= 245.384 kbits/sec  Lost_Packets= 53
Mean_Delay= 0.00239683 sec  Jitter= 0.00223682
```

Β) Περιεχόμενα αρχείου RESULTS-VO-35.dat

```
Num_Stations= 4  Avg_Troughput= 248.468 kbits/sec  Lost_Packets= 0
Mean_Delay= 0.000320226 sec  Jitter= 9.8361e-05
Num_Stations= 5  Avg_Troughput= 248.488 kbits/sec  Lost_Packets= 0
Mean_Delay= 0.000345099 sec  Jitter= 0.000160489
Num_Stations= 6  Avg_Troughput= 248.55 kbits/sec  Lost_Packets= 0
Mean_Delay= 0.000325051 sec  Jitter= 0.000167503
Num_Stations= 7  Avg_Troughput= 248.537 kbits/sec  Lost_Packets= 0
Mean_Delay= 0.000340839 sec  Jitter= 0.000168487
Num_Stations= 8  Avg_Troughput= 248.558 kbits/sec  Lost_Packets= 0
Mean_Delay= 0.000363115 sec  Jitter= 0.000193299
Num_Stations= 9  Avg_Troughput= 248.479 kbits/sec  Lost_Packets= 1
Mean_Delay= 0.000356152 sec  Jitter= 0.000184842
Num_Stations= 10  Avg_Troughput= 248.519 kbits/sec  Lost_Packets= 1
Mean_Delay= 0.000430488 sec  Jitter= 0.000263816
Num_Stations= 11  Avg_Troughput= 248.494 kbits/sec  Lost_Packets= 2
Mean_Delay= 0.000400883 sec  Jitter= 0.000222534
Num_Stations= 12  Avg_Troughput= 248.549 kbits/sec  Lost_Packets= 0
Mean_Delay= 0.000441886 sec  Jitter= 0.000209786
Num_Stations= 13  Avg_Troughput= 248.497 kbits/sec  Lost_Packets= 2
Mean_Delay= 0.000487368 sec  Jitter= 0.000188985
Num_Stations= 14  Avg_Troughput= 248.463 kbits/sec  Lost_Packets= 3
Mean_Delay= 0.0004307 sec  Jitter= 0.000195795
Num_Stations= 15  Avg_Troughput= 248.401 kbits/sec  Lost_Packets= 4
Mean_Delay= 0.000630114 sec  Jitter= 0.000217984
Num_Stations= 16  Avg_Troughput= 248.559 kbits/sec  Lost_Packets= 0
Mean_Delay= 0.000623125 sec  Jitter= 0.000252317
Num_Stations= 17  Avg_Troughput= 248.409 kbits/sec  Lost_Packets= 4
Mean_Delay= 0.000564177 sec  Jitter= 0.000239069
Num_Stations= 18  Avg_Troughput= 248.491 kbits/sec  Lost_Packets= 2
Mean_Delay= 0.000681285 sec  Jitter= 0.000249218
Num_Stations= 19  Avg_Troughput= 248.412 kbits/sec  Lost_Packets= 4
Mean_Delay= 0.00069213 sec  Jitter= 0.000226127
Num_Stations= 20  Avg_Troughput= 248.283 kbits/sec  Lost_Packets= 8
Mean_Delay= 0.000689825 sec  Jitter= 0.000217689
Num_Stations= 21  Avg_Troughput= 248.406 kbits/sec  Lost_Packets= 4
Mean_Delay= 0.000814473 sec  Jitter= 0.000253896
Num_Stations= 22  Avg_Troughput= 248.395 kbits/sec  Lost_Packets= 5
Mean_Delay= 0.000744731 sec  Jitter= 0.000281591
```

```
Num_Stations= 23  Avg_Troughput= 248.383 kbits/sec  Lost_Packets= 5
Mean_Delay= 0.00104153 sec  Jitter= 0.000369052
Num_Stations= 24  Avg_Troughput= 248.305 kbits/sec  Lost_Packets= 7
Mean_Delay= 0.00107194 sec  Jitter= 0.00032876
Num_Stations= 25  Avg_Troughput= 244.81 kbits/sec  Lost_Packets= 119
Mean_Delay= 0.00100705 sec  Jitter= 0.00313732
Num_Stations= 26  Avg_Troughput= 248.425 kbits/sec  Lost_Packets= 4
Mean_Delay= 0.000961094 sec  Jitter= 0.000237694
Num_Stations= 27  Avg_Troughput= 248.284 kbits/sec  Lost_Packets= 8
Mean_Delay= 0.00130064 sec  Jitter= 0.000473388
Num_Stations= 28  Avg_Troughput= 248.311 kbits/sec  Lost_Packets= 7
Mean_Delay= 0.00128294 sec  Jitter= 0.000469279
Num_Stations= 29  Avg_Troughput= 236.906 kbits/sec  Lost_Packets= 422
Mean_Delay= 0.00171226 sec  Jitter= 0.000853618
Num_Stations= 30  Avg_Troughput= 248.39 kbits/sec  Lost_Packets= 5
Mean_Delay= 0.00170595 sec  Jitter= 0.000496233
Num_Stations= 31  Avg_Troughput= 248.223 kbits/sec  Lost_Packets= 9
Mean_Delay= 0.00155822 sec  Jitter= 0.000469919
Num_Stations= 32  Avg_Troughput= 248.195 kbits/sec  Lost_Packets= 10
Mean_Delay= 0.00179895 sec  Jitter= 0.000396905
Num_Stations= 33  Avg_Troughput= 235.946 kbits/sec  Lost_Packets= 201
Mean_Delay= 0.0028106 sec  Jitter= 0.00138096
Num_Stations= 34  Avg_Troughput= 247.691 kbits/sec  Lost_Packets= 16
Mean_Delay= 0.00220136 sec  Jitter= 0.000892924
Num_Stations= 35  Avg_Troughput= 245.291 kbits/sec  Lost_Packets= 51
Mean_Delay= 0.00218704 sec  Jitter= 0.000987326
```