ΠΑΝΕΠΙΣΤΉΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Επισκόπηση πρωτοκόλλων πρόσβασης στο μέσο για δίκτυο 5G και 6G

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΚΑΛΚΙΤΣΑ ΒΑΣΙΛΕΙΟΥ

(ΑΕΜ: 1261)

Επιβλέπων : **ΒΕΡΓΑΔΟΣ ΔΗΜΗΤΡΙΟΣ**

Καστοριά (12/04/2024)

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Επισκόπηση πρωτοκόλλων πρόσβασης στο μέσο για δίκτυο 5G και 6G

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΚΑΛΚΙΤΣΑ ΒΑΣΙΛΕΙΟΥ

(ΑΕΜ:1261)

Επιβλέπων : ΒΕΡΓΑΔΟΣ ΔΗΜΗΤΡΙΟΣ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 12/04/2024

ΒΕΡΓΑΔΟΣ ΔΗΜΗΤΡΙΟΣ ΝΙΚΟΛΑΟΥ ΣΠΥΡ ΒΑΡΔΑΚΑΣ ΙΩΑΝΝΗΣ

Καστοριά 12/04/2024

Περίληψη

Στα πλαίσια της συγκεκριμένης πτυχιακής, αρχικά πραγματοποιήθηκε παρουσίαση όσο αφορά τα στοιχεία της αρχιτεκτονικής των 5G δικτύων, το πεδίο εφαρμογής τους, καθώς και οι χρησιμοποιούμενες τεχνολογίες τους. Αντίστοιχου τύπου ανάλυση πραγματοποιήθηκε και για την περίπτωση των δικτύων 6G στο επόμενο κεφάλαιο. Τέλος, στο πρακτικό μέρος της πτυχιακής, χρησιμοποιήθηκε ο ns-3 simulator και πιο συγκεκριμένα το module nr, για την πραγματοποίηση της πειραματικής διαδικασίας και την καταγραφή των αντίστοιχων αποτελεσμάτων.

Abstract

In the context of this thesis, a presentation was initially made regarding the elements of the architecture of 5G networks, their scope of application, as well as their technologies used. A similar type of analysis was carried out for the case of 6g networks in the next chapter. Finally, in the practical part of the thesis, the ns-3 simulator and more specifically the NR module were used to carry out the experimental procedure and record the corresponding results.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

ΚΕΦΑΛΑΙΟ 1: 5G ΔΙΚΤΥΑ

Στα πλαίσια του συγκεκριμένου κεφαλαίου θα παρουσιαστούν κάποια εισαγωγικά στοιχεία όσο αφορά την περίπτωση των δικτύων 5G. Πιο συγκεκριμένα θα αναφερθούν τα κύρια χαρακτηριστικά της λειτουργικότητας τους, η αρχιτεκτονική τους, οι τεχνολογίες που χρησιμοποιούνται στην περίπτωσή τους, καθώς και τα πεδία εφαρμογής τους [2],[3].

1.1. Κύρια Χαρακτηριστικά Δικτύων 5G

Στην παρούσα ενότητα, συνοψίζονται τα 3 σημαντικότερα χαρακτηριστικά της λειτουργικότητας των δικτύων 5G:

- **Enhanced Mobile Broadband (EMBB):** Δυνατότητα χρήσης εφαρμογών με επίκεντρο τον τελικό χρήστη που χαρακτηρίζονται από την απαίτηση πρόσβασης σε περιεχόμενο πολυμέσων και απαιτούν υψηλούς ρυθμούς μεταφοράς δεδομένων με μέγιστες ταχύτητες έως 20 Gbps και ταχύτητες που αντιλαμβάνεται ο χρήστης έως και 100 Mbps. Επιπλέον παρέχεται υποστήριξη κινητικότητας έως 500 km/h και φασματική απόδοση τρεις φορές υψηλότερη από εκείνη που παρέχεται από το 4G. Παραδείγματα εφαρμογών EMBB αποτελούν τα βίντεο εξαιρετικά υψηλής ευκρίνειας, καθώς και η εικονική και επαυξημένη πραγματικότητα.

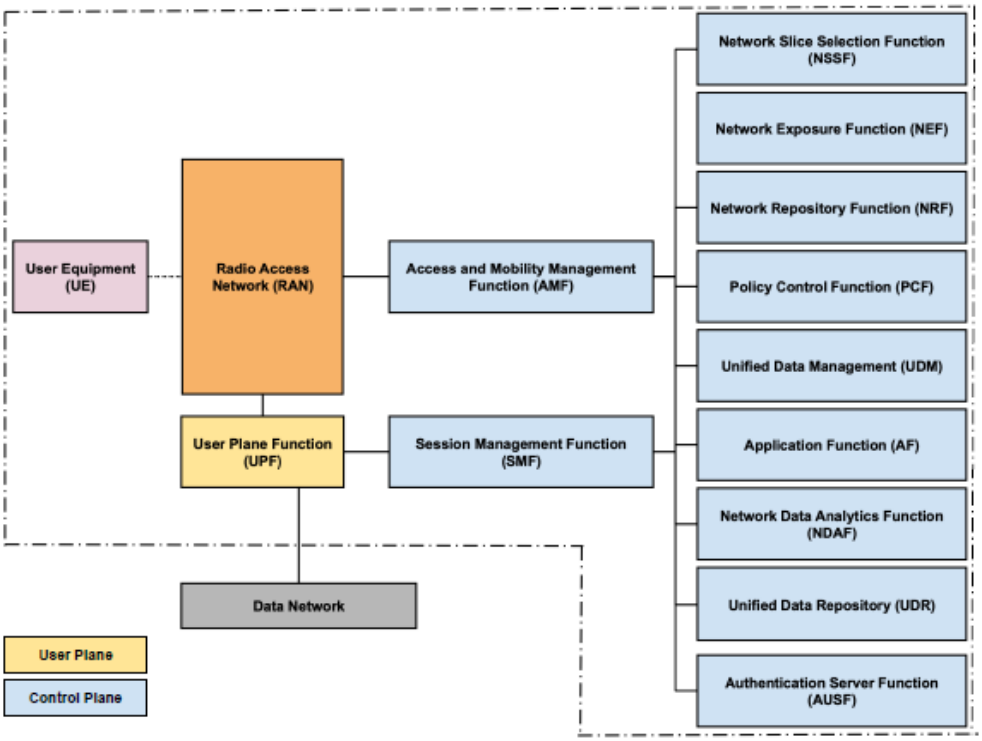
- **Massive Machine – Type Communications (MMTC):** Δυνατότητα χρήσης εφαρμογών που περιλαμβάνουν μεγάλο αριθμό συνδεδεμένων συσκευών, οι οποίες μεταδίδουν χαμηλό όγκο δεδομένων με μικρή ευαισθησία στην καθυστέρηση και των οποίων η κατανάλωση ενέργειας είναι συνήθως χαμηλή. Οι εφαρμογές αυτού του τύπου περιλαμβάνουν συνδεδεμένα οχήματα, φροντίδα κατοικίδιων ζώων / ατόμων και ασύρματα δίκτυα αισθητήρων.
- **Ultra Reliable Low Latency Communications (URLLC):** Δυνατότητα χρήσης εφαρμογών που χαρακτηρίζονται από λειτουργία σε πραγματικό χρόνο, όπως αυτές που σχετίζονται με την ασφάλεια των μεταφορών, τις απομακρυσμένες χειρουργικές επεμβάσεις και την αυτοματοποίηση βιομηχανικών διεργασιών, οι οποίες απαιτούν χαμηλή καθυστέρηση έως 1 millisecond.

1.2. Αρχιτεκτονική Δικτύου 5G

Στην παρακάτω εικόνα, παρατίθεται το διάγραμμα της αρχιτεκτονικής ενός 5G δικτύου, που αποτελείται από 3 βασικά συστατικά:

- Δίκτυο ραδιοπρόσβασης (Radio Access Network – RAN)
- Πυρήνας του δικτύου (Core Network)
- Εξοπλισμός Χρήστη (User Equipment)

Τα στοιχεία αυτά αντιπροσωπεύονται από τις Λειτουργίες Δικτύου (Network Functions - NF) και τις αντίστοιχε διεπαφές.



Εικόνα 1: Διάγραμμα Αρχιτεκτονικής 5G Δικτύου

Επιπλέον, στον παρακάτω πίνακα παρατίθενται οι Λειτουργίες Δικτύου (Network Functions), καθώς και μια σύντομη περιγραφή για αυτές. Για τα ονόματα των ορολογιών, προτιμήθηκε η διεθνής ορολογία και όχι η μετάφραση στην ελληνική γλώσσα, που δεν αποδίδει ακριβώς το νόημα των λέξεων.

Λειτουργία	Περιγραφή
Application	Αλληλοεπιδρά με τον πυρήνα του δικτύου και υποβοηθά στην δρομολόγηση της κίνησης.
Access and mobility management	Περιλαμβάνει τις διαδικασίες της

	εγγραφής, της σύνδεσης , της πρόσβασης και της διαχείρισης κινητικότητας.
Authentication server	Επιβλέπει την διαδικασία αυθεντικοποίησης.
Data Network	Λειτουργία συνδεσιμότητας με κάθε δίκτυο δεδομένων
Network Data Analysis	Ανάλυση δεδομένων των λειτουργιών δικτύου
Network exposure	Εκθέτει τις λειτουργικότητες και τα γεγονότα των λειτουργιών του δικτύου, επιτρέποντας επικοινωνία με εξωτερικές εφαρμογές.
Network repository	Καταχώρηση των λειτουργιών του δικτύου
Network slice selection	Επιλογή των τμημάτων του λογικού δικτύου, που εξυπηρετούν τον εξοπλισμό του χρήστη (UE)
Policy control	Υποστηρίζει τις επιλεγμένες πολιτικές που χρησιμοποιούνται από το επίπεδο ελέγχου (CP) για τον έλεγχο του δικτύου.
Radio access Network	Επιτρέπει την ραδιοπρόσβαση.
Session management	Αιτείται την έναρξη συνεδρίας, την

	τροποποίηση και απελευθέρωση των δεδομένων, καθώς και την ανάθεση IP διευθύνσεων των εξοπλισμών χρήστη (UEs).
Unified data management	Υποστηρίζει την δημιουργία των διαπιστευτηρίων πρόσβασης, τον προσδιορισμό του χρήστη και την διαχείριση συνδρομών.
Unified data repository	Επιτρέπει την εύρεση πληροφοριών που σχετίζονται με συνδρομές και πολιτικές.
User equipment	Επιτρέπει στους χρήστες να χρησιμοποιούν τις υπηρεσίες του δικτύου.
User plane	Διαχειρίζεται την κινητικότητα και την ποιότητα υπηρεσίας (QoS).

Πίνακας 1: Λειτουργίες 5G Δικτύου

Στο σημείο αυτό αξίζει να αναφέρουμε ότι τα δίκτυα 5G ακολουθούν μια αρχιτεκτονική, η οποία βασίζεται στην παροχή υπηρεσιών (Serviced Based Architecture – SBA). Αυτού του τύπου η αρχιτεκτονική παρέχει σημαντικά οφέλη όσο αφορά την δομή του δικτύου.

- **Εύκολη Ανανέωση**: Κάθε υπηρεσία μπορεί να ανανεωθεί με ελάχιστη επίδραση στις υπόλοιπες υπηρεσίες.

- **Επεκτασιμότητα:** Η προσθήκη νέας λειτουργικότητας περιλαμβάνει την προσθήκη νέας υπηρεσίας (συμπεριλαμβανομένων και των αντίστοιχων διεπαφών).
- **Αρθρωτότητα (Modularity):** Ένα σύστημα τεχνολογίας 5G αποτελείται από υπηρεσίες, οι οποίες χαρακτηρίζονται από υψηλό βαθμό λεπτομέρειας και χαμηλό επίπεδο σύζευξης.
- **Επαναχρησιμοποίηση:** Οι υπηρεσίες μπορούν να κληθούν από τυποποιημένες διεπαφές, προωθώντας με τον τρόπο αυτό την έννοια της επαναχρησιμοποίησης των λειτουργικοτήτων.

1.3. Χρησιμοποιούμενες Τεχνολογίες

Στην παρούσα ενότητα θα παρουσιαστούν σημαντικές τεχνολογίες, οι οποίες παίζουν καθοριστικό ρόλο στην λειτουργία των 5G δικτύων:

1.3.1. Τμηματοποίηση

Ένας από τους στόχους των δικτύων 5G βασίζεται στην προσφορά ενός ευρέος φάσματος υπηρεσιών, καθεμία με διαφορετικές απαιτήσεις όσο αφορά την ποιότητα των υπηρεσιών (QoS), δημιουργώντας με τον τρόπο αυτό μια σειρά ευκαιριών για εφαρμογές σε διάφορους τομείς της βιομηχανίας. Για την επίτευξη αυτού του στόχου, τα 5G συστήματα πρέπει να είναι σε θέση να παρέχουν τμήματα του δικτύου εφαρμόζοντας την έννοια της λογικής κατάτμησης, λαμβάνοντας υπόψη τις παρακάτω αρχές:

- αυτοματοποίηση της λειτουργίας του δικτύου

- υψηλή αξιοπιστία
- επεκτασιμότητα
- απομόνωση τμημάτων που ανήκουν σε διαφορετικούς πελάτες
- υποστήριξη για softwarization και virtualization
- ιεραρχική αφαίρεση και προσαρμογή τμήματος
- ελαστικότητα πόρων δικτύου

Η λογική κατάτμηση δικτύου στην περίπτωση του 5G πρέπει να ξεπεράσει με έξυπνο και αποτελεσματικό τρόπο διαφορετικές προκλήσεις, όπως:

- η δημιουργία αλγορίθμων εισαγωγής και σχεδιασμού που στοχεύουν στην κατανομή πόρων στα τμήματα κατά τη δημιουργία και τη λειτουργία τους
- η θέση των λειτουργιών δικτύου στην υποδομή
- ο έλεγχος της κυκλοφορίας εντός του τμήματος
- η διαχείριση από άκρο σε άκρο των τμημάτων
- η διαχείριση της κινητικότητας και η ενορχήστρωση των λειτουργιών δικτύου που απαρτίζουν τα τμήματα
- η ασφάλεια

1.3.2. Τεχνητή Νοημοσύνη

Αρκετοί τομείς της τεχνητής νοημοσύνης, μεταξύ των οποίων ξεχωρίζει η μηχανική μάθηση (Machine Learning - ML), είναι χρήσιμοι για την ανάπτυξη γνωστικών λογικών τμημάτων, τα οποία είναι θεμελιώδη για την επίτευξη αυτοκατευθυνόμενων δικτύων 5G.

Ένα αυτοκατευθυνόμενο δίκτυο πρέπει να αναπτύσσει αυτόνομα τον κύκλο: «αντίληψη-σχέδιο-απόφαση-πράξη», ο οποίος πρέπει να συνοδεύεται από μάθηση και ανατροφοδότηση, τόσο από το δίκτυο στο σύνολό του όσο και από τα επιμέρους συστατικά του τμήματα . Οι τεχνικές μηχανικής μάθησης έχουν πρόσφατα χρησιμοποιηθεί στον τομέα των δικτύων έτσι ώστε να αυτοματοποιηθεί η διαδικασία λήψης αποφάσεων. Χαρακτηριστικά παραδείγματα προς την συγκεκριμένη κατεύθυνση αποτελούν η ενισχυτική μάθηση και η βαθιά ενισχυτική μάθηση.

Στην περίπτωση της ενισχυτικής μάθησης, ένας ευφυής πράκτορας λαμβάνει αποφάσεις περιοδικά έχοντας υπόψη καταστάσεις (για παράδειγμα, ένα σύνολο δρομολογητών που είναι διαθέσιμοι για τον υπολογισμό μιας διαδρομής) και ενέργειες (για παράδειγμα, αποστολή κίνησης μέσω της υπολογιζόμενης διαδρομής). Ο πράκτορας παρατηρεί το αποτέλεσμα της αλληλεπίδρασής του με το περιβάλλον και στην συνέχεια αναπροσαρμόζει την στρατηγική του έτσι ώστε να επιτύχει μια βέλτιστη πολιτική. Ωστόσο, οι τεχνικές μηχανικής μάθησης που χρησιμοποιούνται συγκλίνουν με αργό τρόπο προς την βέλτιστη στρατηγική όταν απαιτείται να ανακαλύψουν και να αποκτήσουν γνώση σχετικά με ένα μεγάλο σύνολο καταστάσεων – ενεργειών, καθιστώντας με τον τρόπο αυτό δύσκολη την εφαρμογή τους στην περίπτωση των δικτύων 5G.

Η βαθιά μάθηση έχει πρόσφατα χρησιμοποιηθεί σαν εργαλείο έτσι ώστε να ξεπεραστούν οι προηγούμενοι περιορισμοί, δημιουργώντας με τον τρόπο αυτό έναν νέο τύπο μάθησης, που ονομάζεται βαθιά ενισχυτική μάθηση

(DRL). Χαρακτηριστικές περιπτώσεις προς την συγκεκριμένη κατεύθυνση αποτελούν οι περιπτώσεις των νευρωνικών δικτύων RNN, FNN και LSTM.

Η ενσωμάτωση της βαθιάς ενισχυτικής μάθησης σε δίκτυα τεχνολογίας 5G μπορεί να επιφέρει θεαματικά αποτελέσματα όσο αφορά τους μηχανισμούς διαχείρισης των λογικών τμημάτων, ειδικά εκείνων που σχετίζονται με τον προγραμματισμό και την κοινή χρήση πόρων. Αυτή η ενσωμάτωση επιτρέπει την μετατροπή τεχνικών που βασίζονται σε μοντέλα, σε τεχνικές χωρίς μοντέλο, οι οποίες μαθαίνουν βαθιά αλληλοεπιδρώντας με το περιβάλλον για να ικανοποιήσουν τόσο τις συμφωνίες επιπέδου εμπειρίας όσο και τις συμφωνίες επιπέδου υπηρεσιών. Οι συμφωνίες αυτές σχετίζονται με δείκτες ποιότητας εμπειρίας (QoE) και ποιότητας υπηρεσίας (QoS) αντίστοιχα.

Λόγω της πολυπλοκότητας των συστημάτων επικοινωνίας στην περίπτωση του 5G, τα ακόλουθα σημεία είναι θεμελιώδη για τη λογική κατάτμηση του δικτύου με βάση την βαθιά ενισχυτική μάθηση:

- Ο ορισμός των συνόλων καταστάσεων και δράσεων πρέπει να καλύπτει τη δυναμική των τμημάτων.
- Η μοντελοποίηση των συνόλων καταστάσεων και δράσεων θα πρέπει να μειώσει το κόστος της μάθησης.
- Η ανταμοιβή πρέπει να είναι όσο το δυνατόν πιο απλή για να διευκολύνει τη διαδικασία μάθησης.
- Η ποσοτικοποίηση της ανταμοιβής θα πρέπει να ενσωματώνει τον χρόνο απόκρισης (η αλληλεπίδραση με το περιβάλλον δεν είναι στιγμιαία).

- Τα αποτελέσματα που παρέχονται από τους πράκτορες βαθιάς μάθησης πρέπει να είναι συμβατά με τη χρονική κλίμακα στην οποία συμβαίνουν τα γεγονότα.

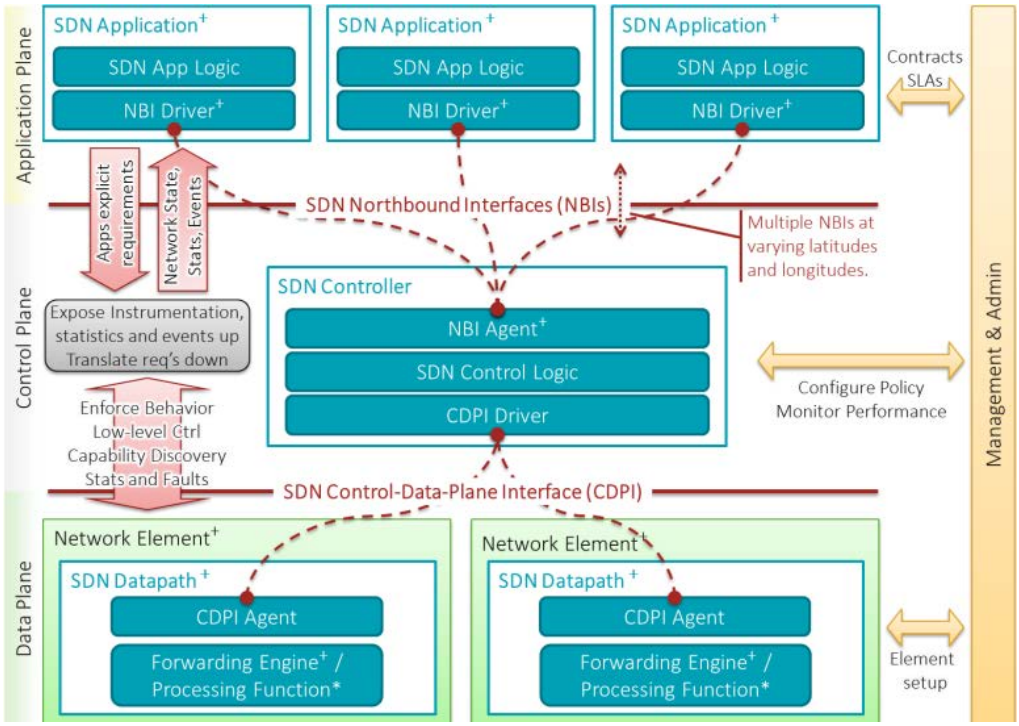
1.3.3. Δίκτυα Καθορισμένα από Λογισμικό (Software Defined Networks)

Μια θεμελιώδης έννοια για την λογισμικοποίηση (softwarization) των δικτύων αποτελεί αυτή των δικτύων που είναι καθορισμένα από λογισμικό (SDNs). Η έννοια αυτή είναι απαραίτητη για την διαχείριση πολλαπλών εικονικών λογικών δικτύων στην περίπτωση της τεχνολογίας 5G. Η αρχιτεκτονική SDN προσφέρει τα παρακάτω πλεονεκτήματα:

- ξεκάθαρος διαχωρισμός του επιπέδου δεδομένων και του επιπέδου ελέγχου
- λογική κεντρικοποίηση της λειτουργίας ελέγχου
- υλοποίηση της λειτουργίας ελέγχου σε λογισμικό

Στην παρακάτω εικόνα, παρατίθεται διάγραμμα αρχιτεκτονικής στην περίπτωση των SDNs [1], στο οποίο διακρίνονται 3 επίπεδα:

- Επίπεδο εφαρμογής
- Επίπεδο ελέγχου
- Επίπεδο δεδομένων



Εικόνα 2: Διάγραμμα Αρχιτεκτονικής SDNs

Σύμφωνα με την παραπάνω αρχιτεκτονική, το επίπεδο δεδομένων αποτελείται από στοιχεία του δικτύου, τα οποία εξειδικεύονται στην διαχείριση πακέτων και επικοινωνεί με το επίπεδο ελέγχου μέσω μίας ή και παραπάνω διεπαφών (χαρακτηριστική είναι η περίπτωση της ζώνης SBI). Όσο αφορά το επίπεδο εφαρμογής, υλοποιεί και οργανώνει την λογική των λειτουργιών του δικτύου. Επιπλέον, το επίπεδο εφαρμογής επικοινωνεί τις απαιτήσεις δικτύου που έχει προς το επίπεδο ελέγχου μέσω μιας ή και παραπάνω διεπαφών (NBI).

Το επίπεδο ελέγχου μεταφράζει τις απαιτήσεις του επιπέδου εφαρμογής και τις εφαρμόζει στα στοιχεία του δικτύου μέσω της ζώνης SBI. Επιπλέον, το επίπεδο ελέγχου καθορίζει εκείνες τις διεπαφές, οι οποίες ενεργοποιούν την

υλοποίηση του ελέγχου που αφορά δίκτυα μεγάλης κλίμακας και ευρείας περιοχής.

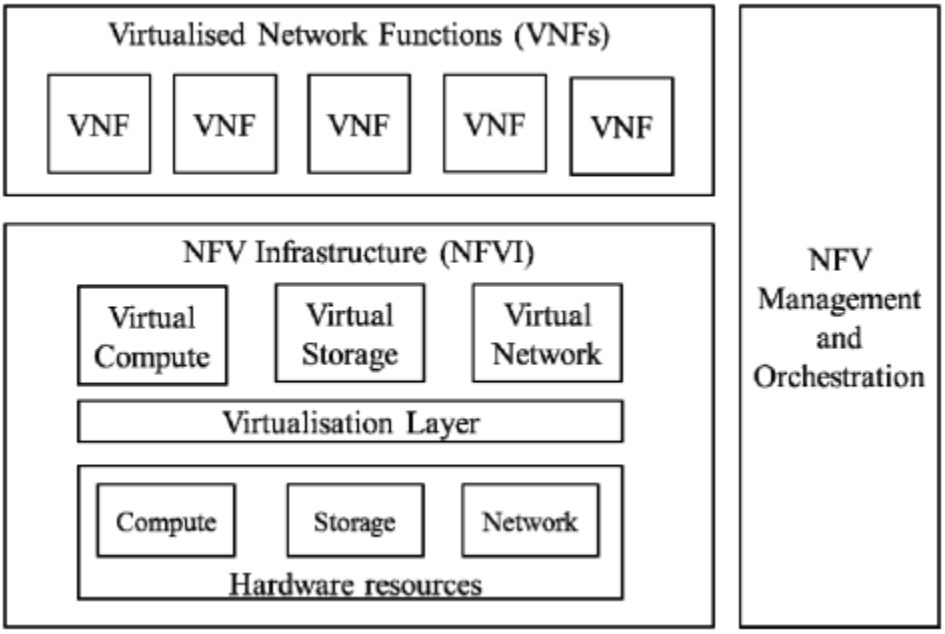
1.3.4. Εικονικοποίηση (Virtualization) των λειτουργιών του δικτύου

Η NFV (Network Function Virtualization) είναι μια τεχνολογία που διευκολύνει την ανάπτυξη πολλαπλών λογικών τμημάτων δικτύου σε μια κοινή υποδομή. Η συγκεκριμένη τεχνολογία αποσυνδέει τις λειτουργίες δικτύου από το υλικό στο οποίο τρέχουν, έτσι να εξαγάγει τους φυσικούς πόρους της υποδομής, να τους διαιρέσει λογικά και να τους αναθέσει στα VFNs που δημιουργούν τα εικονικά λογικά δίκτυα.

Τα κύρια συστατικά στην περίπτωση του NFV είναι:

- Η υποδομή NFV (NFVI)
- Τα VNFs
- MANO (Management and Network Orchestration)

Χαρακτηριστικό είναι το παρακάτω σχήμα [4], το οποίο παρουσιάζει την προαναφερθείσα αρχιτεκτονική:



Εικόνα 3: Virtualization

1.4. Εφαρμογές Τεχνολογίας 5G

Στην παρούσα ενότητα, θα αναφερθούν τα πεδία εφαρμογής της τεχνολογίας 5G και πιο συγκεκριμένα η χρήση της σε:

- Δίκτυα κινητής τηλεφωνίας υψηλής ταχύτητας
- Ψυχαγωγία και πολυμέσα
- Διαδίκτυο των Πραγμάτων (Internet Of Things)
- Υγειονομική περίθαλψη και κρίσιμες εφαρμογές
- Δορυφορικό Διαδίκτυο

1.4.1. Δίκτυο κινητής τηλεφωνίας υψηλής ταχύτητας

Η τεχνολογία 5G αποτελεί μια σημαντική εξέλιξη σε σχέση με όλες τις προηγούμενες τεχνολογίες κινητών επικοινωνιών, καθώς προφέρει πολύ υψηλές ταχύτητες λήψης έως 20 Gbps. Μπορεί κανείς εύκολα να ισχυριστεί ότι ένα 5G ασύρματο δίκτυο λειτουργεί σαν μια σύνδεση στο διαδίκτυο, που βασίζεται στην χρήση οπτικών ινών. Επιπλέον, η τεχνολογία 5G είναι διαφορετική σε σχέση με όλες τις παραδοσιακές τεχνολογίες κινητής μετάδοσης, καθώς προσφέρει με αποτελεσματικό τρόπο δυνατότητα μετάδοσης φωνής αλλά δια - συνδεσιμότητα δεδομένων υψηλής ταχύτητας. Δεν θα πρέπει επίσης να παραβλέπεται το γεγονός ότι προσφέρει επικοινωνίες πολύ χαμηλής καθυστέρησης μικρότερης του ενός millisecond, κατάλληλης για αυτόνομη οδήγηση και εφαρμογές κρίσιμης σημασίας. Η χρήση χιλιοστομετρικών κυμάτων (millimeter waves) για την μετάδοση δεδομένων προσφέρει υψηλότερο εύρος ζώνης και μεγαλύτερο ρυθμό μετάδοσης σε σχέση με τις LTE ζώνες. Για τον λόγο αυτό, η τεχνολογία 5G, επιτρέπει εικονική πρόσβαση σε υψηλή επεξεργαστική ισχύ και ασφαλή πρόσβαση σε υπηρεσίες υπολογιστικού νέφους (cloud computing services). Το μικρό μέγεθος κυψέλης αποτελεί ένα από κρίσιμα χαρακτηριστικά της τεχνολογία 5G, γεγονός που επιφέρει πολλά οφέλη όπως:

- υψηλό ποσοστό κάλυψης
- υψηλός ρυθμός μετάδοσης δεδομένων
- εξοικονόμηση ενέργειας
- εύκολη και γρήγορη πρόσβαση σε cloud υπηρεσίες

1.4.2. Ψυχαγωγία και πολυμέσα

Σε μία έρευνα που πραγματοποιήθηκε, διαπιστώθηκε ότι περισσότερο από το 50% της κίνησης στο διαδίκτυο όσο αφορά τις κινητές συσκευές χρησιμοποιήθηκε για λήψη βίντεο. Αυτή η τάση σίγουρα θα αυξηθεί στο μέλλον, γεγονός που θα καταστήσει την ροή βίντεο περισσότερο διαδεδομένη. Η τεχνολογία 5G προσφέρει ροή υψηλής ταχύτητας για βίντεο ανάλυσης 4K με κρυστάλλινο ήχο και με τον τρόπο αυτό προσφέρει έναν εικονικό κόσμο υψηλής ευκρίνειας στις κινητές συσκευές. Επομένως, θα ωφελήσει τη βιομηχανία ψυχαγωγίας, καθώς προσφέρει την δυνατότητα αναπαραγωγής 120 καρέ ανά δευτερόλεπτο σε βίντεο και επιπλέον δίνεται η δυνατότητα παρακολούθησης καναλιών υψηλής ανάλυσης χωρίς διακοπές, γεγονός που συμβάλλει στην ενίσχυση της θετικής εμπειρίας του χρήστη.

Τα δίκτυα 5G παρέχουν επικοινωνία υψηλής ευκρίνειας και χαμηλής καθυστέρησης, ώστε η επαυξημένη πραγματικότητα (Augmented Reality) και η εικονική πραγματικότητα (Virtual Reality) να εφαρμοστούν με εύκολο και αποδοτικό τρόπο στο μέλλον. Στο σημείο αυτό αξίζει να αναφέρουμε ότι τα παιχνίδια εικονικής πραγματικότητας αποτελούν μια σύγχρονη τάση και πολλές εταιρείες επενδύουν στην δημιουργία τέτοιου τύπου παιχνιδιών. Η τεχνολογία 5G μέσω της απρόσκοπτης πρόσβασης που εξασφαλίζει, συμβάλλει στην περαιτέρω εξάπλωση παιχνιδιών εικονικής πραγματικότητας, ενισχύοντας με τον τρόπο αυτό και τον ψυχαγωγικό της ρόλο.

1.4.3. Διαδίκτυο των Πραγμάτων (Internet of Things)

Τα δίκτυα κινητής τηλεφωνίας 5G διαδραματίζουν σημαντικό ρόλο στην ανάπτυξη του Internet of Things (IoT). Όπως είναι ήδη γνωστό, το IoT παρέχει την δυνατότητα διασύνδεσης διαφορετικών τύπων αντικειμένων και εφαρμογών στο διαδίκτυο. Οι εφαρμογές, οι οποίες χρησιμοποιούνται στην συγκεκριμένη περίπτωση, συλλέγουν δεδομένα από διαφορετικές συσκευές και αισθητήρες. Καθοριστικό ρόλο προς την διευκόλυνση της παραπάνω λειτουργικότητας, αναμένεται να παίξει η τεχνολογία 5G, η οποία παρέχει σύνδεση στο Διαδίκτυο πολύ υψηλής ταχύτητας για τη συλλογή, τη μετάδοση, τον έλεγχο και την επεξεργασία των διακινούμενων δεδομένων. Δεν θα πρέπει να παραβλέπεται το γεγονός ότι το δίκτυο 5G αποτελεί ένα ευέλικτο δίκτυο με αχρησιμοποίητη διαθεσιμότητα φάσματος που προσφέρει πολύ χαμηλού κόστους ανάπτυξη, και για τον λόγο αυτό είναι η πιο αποτελεσματική τεχνολογία στην περίπτωση του Διαδικτύου των Πραγμάτων. Στην συνέχεια αναφέρουμε κάποιες ενδεικτικές περιπτώσεις εφαρμογής:

Έξυπνα σπίτια: Τα προϊόντα και συσκευές που σχετίζονται με την λειτουργία έξυπνων σπιτιών παρουσιάζουν τεράστια ζήτηση την σημερινή εποχή. Η τεχνολογία 5G συμβάλλει στην εύκολη και αποτελεσματική διαχείριση των έξυπνων σπιτιών, καθώς προσφέρει συνδεσιμότητα υψηλής ταχύτητας και δυνατότητα παρακολούθησης των έξυπνων συσκευών που χρησιμοποιούνται. Οι έξυπνες οικιακές συσκευές είναι πλέον εύκολα προσβάσιμες και διαμορφώσιμες από απομακρυσμένες τοποθεσίες χρησιμοποιώντας το δίκτυο 5G, καθώς αυτό προσφέρει πολύ γρήγορη επικοινωνία χαμηλής καθυστέρησης.

Έξυπνες πόλεις: Τα ασύρματα δίκτυα 5G συμβάλλουν επίσης στην ανάπτυξη εφαρμογών έξυπνων πόλεων όπως οι:

- αυτόματη διαχείριση κυκλοφορίας
- εξοικονόμηση ενέργειας
- αποδοτική παροχή ηλεκτρικού ρεύματος
- έξυπνο σύστημα φωτισμού
- διαχείριση των υδάτινων πόρων, η διαχείριση πλήθους
- έλεγχος εκτάκτων αναγκών

Όλες οι παραπάνω εφαρμογές συμβάλλουν στην βελτίωση της καθημερινότητας των πολιτών και υποβοηθούνται από τις δυνατότητες που παρέχονται μέσω της τεχνολογίας 5G.

Βιομηχανικό Διαδίκτυο των Πραγμάτων (Industrial IoT): Η ασύρματη τεχνολογία 5G παρέχει πολλά χαρακτηριστικά για τις βιομηχανίες του μέλλοντος, οι οποίες θα είναι οι βιομηχανίες που βασίζονται στο Διαδίκτυο των Πραγμάτων. Ενδεικτικά αναφέρονται τα παρακάτω:

- ασφάλεια
- παρακολούθηση διαδικασιών
- συσκευασία αντικειμένων
- αποστολή αντικειμένων
- ενεργειακή αποδοτικότητα
- αυτοματοποίηση λειτουργίας του εξοπλισμού
- προγνωστική συντήρηση
- διοικητικές λειτουργίες

Έξυπνη γεωργία: Η τεχνολογία 5G αναμένεται να διαδραματίσει κρίσιμο ρόλο στη γεωργία και την έξυπνη γεωργία. Οι αισθητήρες 5G και η τεχνολογία GPS θα βοηθήσουν τους αγρότες ώστε να παρακολουθούν σε ζωντανό χρόνο επιθέσεις σε καλλιέργειες και να τις διαχειρίζονται με τρόπο γρήγορο και αποτελεσματικό. Αυτοί οι έξυπνοι αισθητήρες μπορούν επίσης να χρησιμοποιηθούν για άρδευση, διαχείριση παρασίτων και έλεγχο της κατανάλωσης ηλεκτρικής ενέργειας.

Αυτόνομη οδήγηση: Το ασύρματο δίκτυο 5G προσφέρει πολύ χαμηλή καθυστέρηση όσο αφορά την πραγματοποίηση των επικοινωνιών, η οποία είναι πολύ σημαντική για αυτόνομη οδήγηση. Αυτό σημαίνει ότι τα αυτοκινούμενα αυτοκίνητα θα εμφανιστούν στην πραγματική ζωή σύντομα με την υποβοήθηση της τεχνολογίας 5G. Στην περίπτωση αυτή, τα αυτόνομα οχήματα θα μπορούν εύκολα να επικοινωνούν με τα έξυπνα σήματα διαχείρισης κυκλοφορίας, αντικείμενα και άλλα οχήματα τα οποία βρίσκονται στον δρόμο. Η πολύ χαμηλή καθυστέρηση, η οποία εξασφαλίζεται μέσω της τεχνολογίας 5G μπορεί να καταστήσει την αυτόνομη οδήγηση περισσότερο εφικτή, καθώς κάθε χιλιοστό του δευτερολέπτου είναι σημαντικό στην περίπτωση των αυτόνομων οχημάτων όσο αφορά την λήψη αποφάσεων που πρέπει να λαμβάνεται.

1.4.4. Υγειονομική περίθαλψη και κρίσιμες εφαρμογές

Η τεχνολογία 5G αναμένεται να φέρει εκσυγχρονισμό στην ιατρική επιστήμη, καθώς οι γιατροί και οι επαγγελματίες στον χώρο της υγεία θα μπορούν να εκτελούν προηγμένες ιατρικές διαδικασίες. Το δίκτυο 5G

παρέχει συνδεσιμότητα μεταξύ όλων των τάξεων, οπότε η παρακολούθηση σεμιναρίων και διαλέξεων θα είναι ευκολότερη. Επιπλέον, μέσω της τεχνολογίας 5G, οι ασθενείς μπορούν να συνδεθούν με τους γιατρούς και να λάβουν τις συμβουλές τους. Δεν θα πρέπει να παραβλέπουμε το γεγονός ότι οι επιστήμονες κατασκευάζουν έξυπνες ιατρικές συσκευές που μπορούν να βοηθήσουν άτομα με χρόνιες ιατρικές παθήσεις.

Επομένως, το δίκτυο 5G αναμένεται ενισχύσει τη βιομηχανία υγειονομικής περίθαλψης με τους παρακάτω τρόπους:

- έξυπνες συσκευές
- Διαδίκτυο Ιατρικών Πραγμάτων (Medical Internet of Things)
- έξυπνοι αισθητήρες για παρακολούθηση ιατρικών δεδομένων
- τεχνολογίες ιατρικής απεικόνισης υψηλής ευκρίνειας
- έξυπνα συστήματα ανάλυσης

Επιπλέον, η τεχνολογία 5G θα βοηθήσει στην πρόσβαση αποθηκευτικών μέσων υπολογιστικού νέφους (cloud storage). Αυτό σημαίνει ότι η πρόσβαση στα δεδομένα υγειονομικής περίθαλψης θα είναι πολύ εύκολη από οποιαδήποτε τοποθεσία παγκοσμίως. Με τον τρόπο αυτό, οι γιατροί και οι νοσηλευτές θα μπορούν εύκολα να αποθηκεύουν και να μοιράζονται αρχεία μεγάλου όγκου, όπως αναφορές μαγνητικής τομογραφίας, μέσα σε δευτερόλεπτα χρησιμοποιώντας τις δυνατότητες που παρέχονται μέσω της τεχνολογίας 5G.

1.4.5. Δορυφορικό Διαδίκτυο

Σε πολλές απομακρυσμένες περιοχές, οι επίγειοι σταθμοί βάσης δεν είναι διαθέσιμοι, επομένως η τεχνολογία 5G θα διαδραματίσει κρίσιμο ρόλο στην παροχή συνδεσιμότητας σε αυτές. Το δίκτυο 5G θα παρέχει συνδεσιμότητα χρησιμοποιώντας δορυφορικά συστήματα και το δορυφορικό σύστημα χρησιμοποιεί έναν αστερισμό πολλαπλών μικρών δορυφόρων έτσι ώστε να παρέχει συνδεσιμότητα σε αστικές και αγροτικές περιοχές σε όλο τον κόσμο.

ΚΕΦΑΛΑΙΟ 2: 6G ΔΙΚΤΥΑ

Στα πλαίσια του συγκεκριμένου κεφαλαίου θα παρουσιαστούν τα κύρια χαρακτηριστικά των δικτύων 6G, οι διαστάσεις σχεδιασμού τους, καθώς και το πεδίο εφαρμογής και οι προκλήσεις τους [5], [6], [7].

2.1. Χαρακτηριστικά Δικτύων 6G

Στα πλαίσια της συγκεκριμένης ενότητας, θα παρουσιαστούν τα κύρια χαρακτηριστικά των δικτύων 6G.

2.1.1. Υψηλός Ρυθμός Μεταφοράς Δεδομένων

Τα δίκτυα 6G θα συνεχίσουν την αυξητική τάση του ρυθμού μεταφοράς δεδομένων, με προβλεπόμενη ταχύτητα λήψης τουλάχιστον 100 Gbps. Η ταχύτητα αυτή είναι δέκα φορές μεγαλύτερη από την (θεωρητική) ταχύτητα λήψης ενός δικτύου 5G και 300 φορές μεγαλύτερη σε σχέση με αυτή των δικτύων 4G.

2.1.2. Υψηλές Συχνότητες

Όσο υψηλότερη είναι η συχνότητα, τόσο περισσότερο είναι το διαθέσιμο εύρος ζώνης. Επομένως, προκειμένου να επιτευχθούν υψηλά εύρη ζώνης, θα πρέπει να εξασφαλιστούν πολύ υψηλές συχνότητες. Για παράδειγμα, τα δίκτυα 4G περιορίζονται σε συχνότητες έως 2,5 GHz, ενώ τα δίκτυα 5G λειτουργούν στις ζώνες 28 και 39 GHz. Όσο αφορά την επόμενη γενιά

δικτύων κινητής τηλεφωνίας, συμπεριλαμβανομένου του 6G, αναμένεται να καταφύγει σε συχνότητες που θα αγγίζουν τα terahertz.

2.1.3. Πολύ μικρές καθυστερήσεις

Η εμπειρία των πελατών κινητής τηλεφωνίας εξαρτάται από το ποσό των δεδομένων που μπορούν να κατεβάσουν. Για πολλές εφαρμογές, η καθυστέρηση του δικτύου είναι εξίσου σημαντικός παράγοντας.

Βεβαίως, η εισαγωγή δικτύων 5G (και η λανθάνουσα κατάσταση τους μικρότερη από 1 χιλιοστό του δευτερολέπτου) θα πρέπει ήδη να δώσει τέλος σε τέτοιους προβληματισμούς. Όμως, η τεχνολογία 6G προτείνει μια ακόμη πιο βελτιωμένη λανθάνουσα κατάσταση λίγων μικροδευτερόλεπτων.

Η εξέλιξη αυτή είναι απαραίτητη για την υποστήριξη του αυξανόμενου αριθμού εφαρμογών Internet of Things (IoT). Ας αναλογιστούμε την περίπτωση των συστημάτων κλειστού βρόχου που ελέγχουν μηχανές και πολύπλοκες βιομηχανικές διεργασίες, οι οποίες βασίζονται σε δεδομένα αισθητήρων σε πραγματικό χρόνο ή ευαίσθητες στο χρόνο ιατρικές εφαρμογές IoT, όπως η επεξεργασία και η ερμηνεία σημάτων ηλεκτροκαρδιογραφήματος (ΗΚΓ) ή ηλεκτροεγκεφαλογραφήματος (EEG).

2.1.4. Τεράστιος όγκος διασυνδεδεμένων συσκευών

Η δύναμη του Διαδικτύου των Πραγμάτων καθορίζεται από τον αριθμό των συνδεδεμένων αισθητήρων και συσκευών. Και εδώ προβλέπεται τεράστια ανάπτυξη. Η εταιρεία έρευνας αγοράς Statista προβλέπει ότι μέχρι το 2025 το IoT θα αποτελείται από σχεδόν 31 δισεκατομμύρια συσκευές, σε σύγκριση με

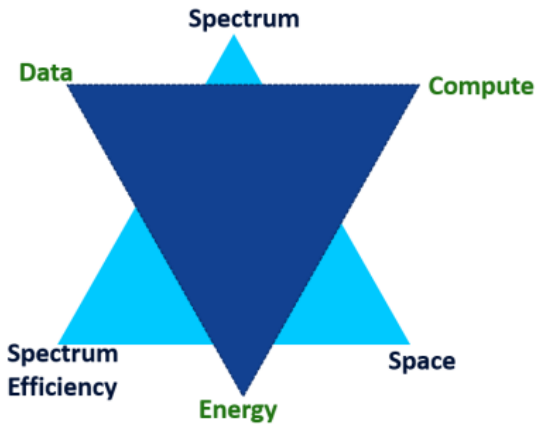
τις 12 έως 13 δισεκατομμύρια συσκευές σήμερα. Με αυτόν τον συνεχώς αυξανόμενο αριθμό συσκευών έρχεται η πρόκληση της σύνδεσης όσο το δυνατόν περισσότερων συσκευών στο Διαδίκτυο (ανά m2 ή km2). Αυτός ο αριθμός αναφέρεται ως πυκνότητα σύνδεσης.

Τα σημερινά δίκτυα 4G επιτυγχάνουν πυκνότητα σύνδεσης περίπου 100.000 συσκευών ανά km2. Το 5G έχει επιφέρει καλύτερα αποτελέσματα, επιτρέποντας τη σύνδεση 1 εκατομμυρίου συσκευών ανά km2. Και με την εισαγωγή δικτύων 6G, ο αριθμός των 10 εκατομμυρίων συνδεδεμένων συσκευών ανά km2 έρχεται πολύ κοντά.

2.2. Διαστάσεις Σχεδίασης του 6G

Σε κάθε γενιά μέχρι το 5G, οι τρεις θεμελιώδεις διαστάσεις: του φάσματος, της φασματικής απόδοσης και της χωρικής επαναχρησιμοποίησης έχουν υπαγορεύσει τον τρόπο με τον οποίο μπορεί να αυξηθεί η χωρητικότητα, όπως αντιπροσωπεύεται από το επόμενο σχήμα. Θα συνεχίσει να ισχύει κάτι τέτοιο και για το 6G. Η τεχνολογία ραδιοσυχνοτήτων θα προχωρήσει σε ισχύ και οικονομικά αποδοτικό φάσμα χρήσης σε ακόμη υψηλότερες ζώνες. Υπάρχει η ευκαιρία τουλάχιστον μιας δεκαπλάσιας αύξησης της ποσότητας του φάσματος με τη μετάβαση σε ζώνες συχνοτήτων terahertz. Η φασματική απόδοση θα βελτιωθεί μέσω της χρήσης MIMO πολλαπλών χρηστών όχι μόνο σε ζώνες κυμάτων εκατοστών (cmWave) αλλά και κυμάτων χιλιοστών (mmWave), καθώς μεταβαίνουμε από αναλογική σε υβριδική/ψηφιακή διαμόρφωση δέσμης. Καθώς το κόστος του μαζικού MIMO πέφτει, ακόμη

μεγαλύτερες συστοιχίες μπορούν να αναπτυχθούν για περαιτέρω αύξηση της φασματικής απόδοσης. Η πυκνότητα του δικτύου θα συνεχίσει αναμφίβολα να αυξάνεται, όχι μόνο για λόγους χωρητικότητας, αλλά για να παρέχει αυξημένη κάλυψη σε ζώνες υψηλότερων συχνοτήτων, σε υψηλότερους ρυθμούς δεδομένων και με υψηλότερη αξιοπιστία. Επιπλέον, θα τεθεί σε εφαρμογή πιο διαδεδομένη πρόσβαση στο φάσμα. Η ανταλλαγή μεταξύ των φορέων εκμετάλλευσης θα επιτρέψει πολύ μεγαλύτερη επαναχρησιμοποίηση του φάσματος. Η αποτελεσματική επαναχρησιμοποίηση του φάσματος είναι ιδιαίτερα σημαντική στις χαμηλότερες ζώνες, καθώς διαθέτουν καλές ιδιότητες διάδοσης χωρίς οπτική επαφή (NLOS) και οι πόροι φάσματος σε αυτές τις ζώνες είναι σπάνιοι.



Εικόνα 4: Διαστάσεις 6G

Υποστηρίζεται ότι το 6G θα διαφέρει θεμελιωδώς από τις προηγούμενες γενιές στο ότι τρεις νέες θεμελιώδεις διαστάσεις θα τεθούν σε λειτουργία εκτός από τις παραπάνω τρεις παραδοσιακές διαστάσεις. Αυτές οι διαστάσεις αντιπροσωπεύουν τους θεμελιώδεις πόρους των δεδομένων, των υπολογιστών και της ενέργειας. Όπως είναι γνωστό, οι τεχνικές AI(Artificial

Intelligence)/ML(Machine Learning) βασίζονται σε δεδομένα και όποιος έχει πρόσβαση σε μεγάλους όγκους δεδομένων συγκεκριμένου τομέα θα είναι επιτυχής στην εφαρμογή αυτών των τεχνικών. Η εφαρμογή του AI / ML στο σχεδιασμό συστημάτων 6G θα είναι θεμελιώδης και παρόμοια με διάφορους άλλους τομείς, και τα δεδομένα δικτύου και αισθητήρων θα γίνουν θεμελιώδεις πόροι που θα αξιοποιηθούν για τη βελτίωση της απόδοσης του συστήματος. Ενώ η υπολογιστική ισχύς ήταν πάντα ένας σημαντικός πόρος για τα κυψελοειδή συστήματα, δύο σημαντικές τάσεις δείχνουν ότι στην περίπτωση της τεχνολογίας 6G δεν θα ισχύει κάτι τέτοιο.

Η πρώτη τάση που παρατηρείται είναι ο αναδυόμενος κορεσμός στον αριθμό των τρανζίστορ που μπορούν να συσκευαστούν σε έναν όγκο μονάδας, ο οποίος περιορίζει την υπολογιστική ισχύ των συσκευών. Η δεύτερη τάση αναφέρει ότι στην περίπτωση της τεχνολογίας 6G θα χρησιμοποιηθούν πολλαπλές συσκευές για ενίσχυση των δυνατοτήτων των ανθρώπων, οι οποίες όμως θα διαθέτουν περιορισμένη υπολογιστική ικανότητα.

2.3. Πεδίο Εφαρμογής 6G

Στα πλαίσια της συγκεκριμένης ενότητας, θα παρουσιαστούν περιπτώσεις στις οποίες θα βρει εφαρμογή η τεχνολογία των δικτύων 6G.

2.3.1. Edge Computing

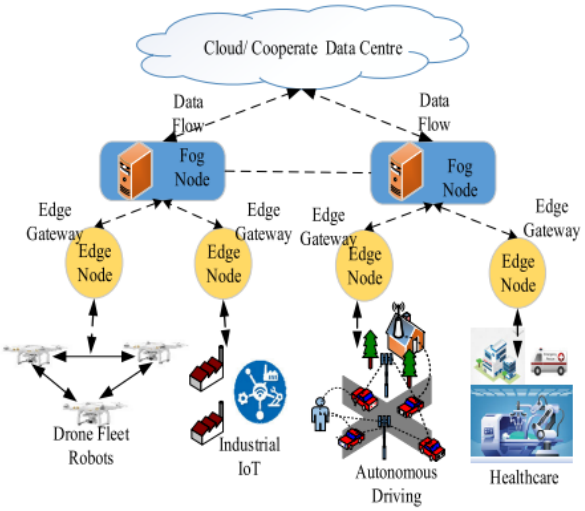
Τα δίκτυα επόμενης γενιάς αναμένεται να υποστηρίζουν πυκνότητα σύνδεσης συσκευών υψηλότερης από 1m/km2 . Η εισαγωγή του τεράστιου αριθμού συσκευών έχει προσθέσει πολλά τελικά σημεία στα υπάρχοντα δίκτυα και καθίσταται περίπλοκη η επεξεργασία τέτοιων τεράστιων δεδομένων με λιγότερο από 1ms χρόνο μετ' επιστροφή (round trip time). Το δίκτυο 6G περιλαμβάνει υπηρεσίες Απτικού Διαδικτύου που απαιτούν εξαιρετικά υψηλό ρυθμό δεδομένων, χαμηλότερη καθυστέρηση και εκτεταμένη αξιοπιστία. Θα επιτρέψει τη φυσική αλληλεπίδραση των ανθρώπων με το αντικείμενο που βρίσκεται στο απομακρυσμένο μέρος. Για την ενεργοποίηση αυτού του τύπου υπηρεσιών, η τεχνολογία Edge Computing (EC) ενσωματώνεται ως η τεχνική για τη μείωση της υπολογιστικής πολυπλοκότητας. Επεξεργάζεται τα περισσότερα από τα δεδομένα στην άκρη του δικτύου αντί να τα μεταφέρει στο cloud δίκτυο, που βρίσκεται πολύ μακριά. Παρέχει μια τοπική πηγή για έναν αριθμό των συσκευών, όπου τα δεδομένα μπορούν να αποθηκευτούν και να υποβληθούν σε επεξεργασία για την ελαχιστοποίηση του χρόνου πρόσβασης και έτσι μειώνει την καθυστέρηση.

Η τεχνική Edge Computing είναι επωφελής σε σχέση με τα παρακάτω:

- Αυξάνει την υπολογιστική ικανότητα του δικτύου.
- Μειώνει το χρόνο υπολογισμού για το χειρισμό των σύνθετων εργασιών σε εφαρμογές σε πραγματικό χρόνο.
- Μειώνει την καθυστέρηση και τον χρόνο πρόσβασης.
- Μειώνει την απαίτηση εύρους ζώνης.
- Βελτιστοποιεί τα δεδομένα για υψηλότερα επίπεδα υπολογισμών.

- Λειτουργεί ως συμπλήρωμα στον υπολογισμό ομίχλης

Χαρακτηριστική είναι η παρακάτω εικόνα:



Εικόνα 5: Edge Computing

2.3.2. Κβαντικοί Υπολογισμοί

Στο 6G, τα συμβατικά υπολογιστικά συστήματα πρόκειται να αντικατασταθούν από κβαντικούς υπολογιστές για την προώθηση του παραλληλισμού, που είναι εγγενής ιδιότητα της κβαντικής μηχανικής. Ο κβαντικός υπολογισμός θα επιτρέψει την παράλληλη επεξεργασία πολυδιάστατων και δεδομένων μεγάλου μεγέθους. Θα ενισχύσει τους αλγόριθμους και τις τεχνικές των ML και AI που ασχολούνται με ένα τεράστιο ποσό δεδομένων για εκπαίδευση. Η κβαντική επικοινωνία μπορεί να είναι χρήσιμη για την παροχή υψηλής ασφάλειας σε δίκτυα 6G. Μπορεί να θεωρηθεί ως βασικός καταλύτης της τεχνολογίας 6g. Ενισχύει την αποτελεσματικότητα του δικτύου για τη διατήρηση της γνώσης σε πραγματικό χρόνο και παρέχει ταχεία ανταπόκριση στη ζήτηση των

χρηστών. Τα κβαντικά υπολογιστικά δίκτυα μπορούν να υποστηρίξουν αποτελεσματικά πολλές εφαρμογές σε μεγάλη απόσταση. Η κβαντική επεξεργασία και ο υπολογισμός ενισχύουν τους ρυθμούς δεδομένων και μειώνουν την καθυστέρηση. Βοηθά στην επίτευξη της εξαιρετικά χαμηλής καθυστέρησης των σύγχρονων τεχνολογιών. Ο κβαντικός υπολογισμός χρησιμοποιεί το κβαντομηχανικό φαινόμενο για χειρισμό κβαντικών πληροφοριών (σε qubits). Γενικά χαρακτηριστικά του κβαντικού υπολογισμού είναι:

- Παρέχει ταχύτερη διαχείριση μεγάλου αριθμού δεδομένων.
- Βασίζεται στο qubit (quantum bit), το οποίο είναι η μονάδα κβαντικών πληροφοριών.
- Προωθεί την παράλληλη επεξεργασία για την υποστήριξη εκατομμυρίων συσκευών ταυτόχρονα.
- Βοηθά στην επεξεργασία δεδομένων εκπαίδευσης για μοντέλα μηχανικής μάθησης και βαθιάς μάθησης

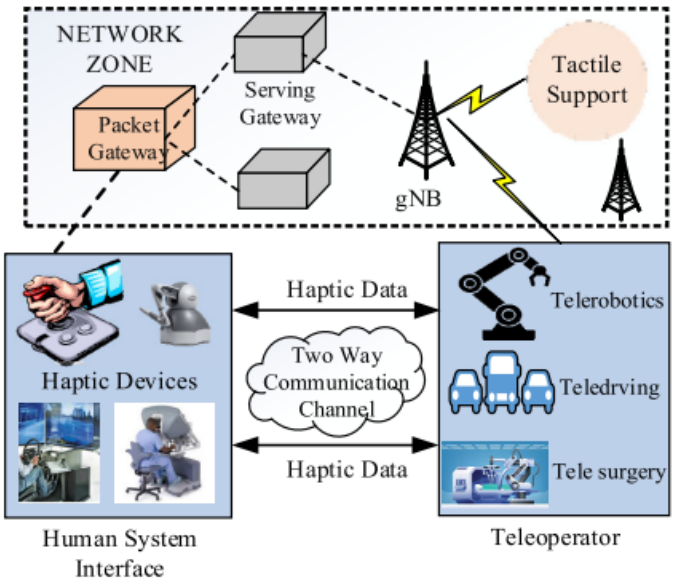
2.3.3. Απτικό Διαδίκτυο

Το Απτικό Διαδίκτυο (Tactile Internet) παρέχει μια πλατφόρμα για την ενεργοποίηση εφαρμογών σε πραγματικό χρόνο, οι οποίες κυμαίνονται από την ενεργοποίηση της ρομποτικής έως τις απτικές επικοινωνίες, συμπεριλαμβανομένης της τηλεχειρουργικής, της αυτόνομης οδήγησης, της AR (Augmented Reality), της VR (Virtual Reality) και της XR (Extended Reality). Στο δίκτυο με δυνατότητα αφής, η χαμηλότερη καθυστέρηση συνδυάζεται με βελτιωμένη αξιοπιστία και ασφάλεια. Θεωρείται ως ο

βασικός παράγοντας της απτικής επικοινωνίας, η οποία ενσωματώνει την πραγματική εμβάπτιση του ανθρώπου στον εικονικό κόσμο με την αίσθηση της αφής και της φυσικής αλληλεπίδρασης από ένα απομακρυσμένο μέρος. Ενσωματώνει τις αισθήσεις (αφή) του ανθρώπου με μια έξυπνη μηχανή και έχει φέρει επανάσταση στα δίκτυα IoT.

Το συμβατικό δίκτυο δεν υποστηρίζει τη μετάδοση απτικών δεδομένων με παραδοσιακά συστήματα. Έτσι, το Απτικό Διαδίκτυο λειτουργεί ως η πλατφόρμα που παρέχει υποστήριξη για τη μεταφορά απτικών δεδομένων. Περιλαμβάνει τη μεταφορά συναισθημάτων, αφής και κίνησης μέσω του δικτύου.

Το επόμενο σχήμα δείχνει την απτική αρχιτεκτονική που βασίζεται στο Διαδίκτυο ως ενεργοποιητή της απτικής επικοινωνίας:



Εικόνα 6: Απτική Επικοινωνία

2.3.4. Οπτική Ασύρματη Επικονωνία

Στα μελλοντικά ασύρματα δίκτυα, οι επικοινωνίες ραδιοσυχνοτήτων αντικαθίστανται από την OWC (Optical Wireless Communication). Περιλαμβάνει τις ζώνες συχνοτήτων υπέρυθρης, ορατού φωτός και υπεριώδους ακτινοβολίας. Μεταξύ αυτών των τριών, η OWC λειτουργεί γενικά στη ζώνη ορατού φωτός για τη μεταφορά δεδομένων. Ο τύπος επικοινωνίας μέσω του φάσματος ορατού φωτός ονομάζεται VLC (Visible Light Communication). Στο VLC, το ορατό φως χρησιμοποιείται για να προσφέρει φωτισμό και επικοινωνία, μαζί με τη συλλογή ενέργειας. Παρέχει μια εναλλακτική λύση για την παροχή ταχύτερων ρυθμών δεδομένων στον εσωτερικό χρήστη με τη βοήθεια LED. Χρησιμοποιεί ορατό φως που είναι ζώνες Terahertz (THz) για τη μεταφορά πληροφοριών που δεν έχουν επιβλαβείς επιπτώσεις στην ανθρώπινη υγεία και έτσι προωθεί την πράσινη επικοινωνία .

Οι αναβαθμισμένες εφαρμογές του δικτύου 6G θα απαιτούν εξαιρετικά υψηλούς ρυθμούς δεδομένων (Tb / s). Μια τέτοια τεράστια ζήτηση για ρυθμούς δεδομένων μπορεί να εκπληρωθεί με την επέκταση του εύρους συχνοτήτων των ασύρματων δικτύων. Επί του παρόντος, σε 5G NR, αναμένεται να χρησιμοποιηθούν τα mmWaves (έως 100GHz). Αυτές οι υψηλές συχνότητες συνδέονται με διάφορες απώλειες που αντισταθμίζονται με τη χρήση της τεχνικής της στενής διαμόρφωσης δέσμης. Οι συχνότητες άνω των 100 GHz αναμένεται να χρησιμοποιηθούν σε δίκτυα 6G. Το φάσμα συχνοτήτων μεταξύ 100 – 300 GHz ορίζεται ως η ζώνη υπο-THz και 300GHz έως 3 THz ως οι ζώνη THz. Ωστόσο, η οπτική ζώνη ορατού φωτός κυμαίνεται από 430-730 THz. Οι επικοινωνίες THz είναι εξαιρετικά ασφαλείς επειδή

αυτές οι επικοινωνίες χρησιμοποιούν στενή δέσμη και μικρή διάρκεια για τη μεταφορά δεδομένων που μειώνει την πιθανότητα επίθεσης. Οι συχνότητες THz είναι μη ιονίζουσες και έχουν λιγότερους θερμικούς κινδύνους από τις ακτινοβολίες RF (Radio Frequency). Είναι αδιαμφισβήτητο ότι αυτές οι συχνότητες έχουν πολύ λιγότερο επιβλαβείς επιπτώσεις στην υγεία των ανθρώπων.

2.3.5. Τεχνικές δυναμικής πολλαπλής πρόσβασης

Τα δίκτυα 6G θα παρέχουν μια πλατφόρμα για εφαρμογές που βασίζονται σε AI και συνεπώς απαιτούν ένα εντελώς νέο πλαίσιο με πρωτόκολλα που βασίζονται σε AI και προσαρμόσιμες τεχνικές πολλαπλής πρόσβασης. Όπως έχει ήδη αναφερθεί, το 6G θα ενσωματώσει όλους τους τομείς του δικτύου, συμπεριλαμβανομένου του επίγειου δικτύου, του αερομεταφερόμενου δικτύου, του διαστημικού δικτύου και του υποβρύχιου δικτύου. Η συνεργασία αυτών των δικτύων θα απαιτήσει ένα εξαιρετικά ευέλικτο και προσαρμόσιμο σύστημα. Ένα νέο πλαίσιο πρωτοκόλλων είναι υποχρεωτικό, το οποίο θα μπορεί να προσαρμοστεί ανάλογα με τις συνθήκες του καναλιού. Απαιτεί αρκετούς δυναμικούς μηχανισμούς για την παροχή ομαλής παράδοσης σε διαφορετικά πρότυπα κινητικότητας του δικτύου 6G. Έτσι, καθίσταται απαραίτητο να αναπτυχθούν νέα πρωτόκολλα που βασίζονται στην τεχνητή νοημοσύνη για την παροχή πρόσβασης, παράδοσης και εξακρίβωσης της ταυτότητας.

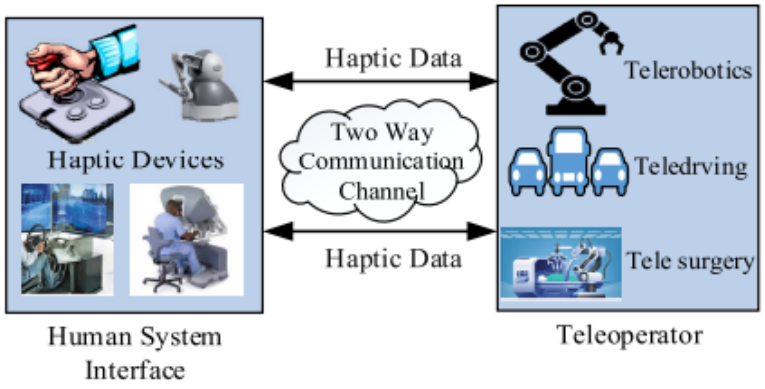
2.3.6. Απτικές Επικοινωνίες

Η απτική ασχολείται με την ανθρώπινη αίσθηση της αφής. Η απτική επικοινωνία περιλαμβάνει γενικά τον έλεγχο και τη διαχείριση του αντικειμένου από το μακρινό μέρος χρησιμοποιώντας την αίσθηση της αφής. Η απτική επικοινωνία περιλαμβάνει επίσης οπτικοακουστικές επικοινωνίες. Παρέχει ένα περιβάλλον όπου ένας χρήστης μπορεί να βιώσει την πραγματική εμβάπτιση του ψηφιακού κόσμου με τον πραγματικό κόσμο για την εκτέλεση διαφόρων εργασιών. Αυτές οι επικοινωνίες θα επιτρέψουν τη φυσική αλληλεπίδραση του χρήστη με ψηφιακές συσκευές που τοποθετούνται εξ αποστάσεως. Η τηλε-παρουσία και η τηλε-δράση θεωρούνται ως ο κύριος στόχος της απτικής επικοινωνίας. Οι δύο έννοιες αυτές περιλαμβάνουν την παρουσία ενός χρήστη στο απομακρυσμένο περιβάλλον χωρίς να είναι εκεί. Το σύστημα που μπορεί να επιτρέψει αυτές τις επικοινωνίες αναφέρεται γενικά ως τηλεαπτικό συστήματα. Αυτά τα συστήματα περιλαμβάνουν διάφορες τεχνολογίες που είναι ικανές να παρέχουν ένα σενάριο στον χρήστη, το οποίο ασχολείται με την ικανότητα ενός ατόμου να αισθάνεται κάτι από ένα απομακρυσμένο μέρος.

Τα τηλεαπτικά συστήματα αποτελούνται από δύο κύρια συστατικά στοιχεία, τη διεπαφή ανθρώπινου συστήματος (HSI) και έναν τηλεχειριστή (γενικά ψηφιακή συσκευή/μηχανή). Μεταξύ αυτών των δύο συνιστωσών, υπάρχει ένα αμφίδρομο κανάλι επικοινωνίας. Στην πλευρά του χρήστη, υπάρχει μια απτική διεπαφή που περιέχει αισθητήρες και τον χειριστή/άνθρωπο που θα αλληλοεπιδράσει με αυτούς τους αισθητήρες. Κάθε κίνηση (από την άποψη της ταχύτητας και της θέσης) του χειριστή ανιχνεύεται από τους αισθητήρες

και μεταδίδεται ως δεδομένα κίνησης στον τηλε-χειριστή μέσω του καναλιού επικοινωνίας. Ως αποτέλεσμα, μπορεί να γίνει οποιαδήποτε ενέργεια στην απομακρυσμένη ψηφιακή συσκευή.

Οι απτικές συνδέσεις απαιτούν πολύ χαμηλή καθυστέρηση μικρότερη από 1 ms με τη μικρότερη απώλεια πακέτων που είναι απαραίτητη για την αξιοπιστία του συστήματος. Η σταθερότητα του δικτύου και του συνδέσμου επικοινωνίας προκαλεί σημαντική ανησυχία στην απτική επικοινωνία. Ακόμη και μικρή καθυστέρηση ή απώλεια δεδομένων στο δίκτυο μπορεί να παρεμποδίσει την ποιότητα της εργασίας ή της υπηρεσίας. Το Απτικό Διαδίκτυο είναι ο βασικός παράγοντας της απτικής επικοινωνίας. Το επόμενο σχήμα δείχνει την αλληλεπίδραση των ανθρώπων με τηλε-ρομπότ, τηλε-οδήγηση και τηλεχειρουργική. Η απτική επικοινωνία αντιμετωπίζει αρκετές προκλήσεις που σχετίζονται με την καθυστέρηση, την αξιοπιστία, τη σταθερότητα, το υλικό και την ασφάλεια.



Εικόνα 7: Απτικές Επικοινωνίες

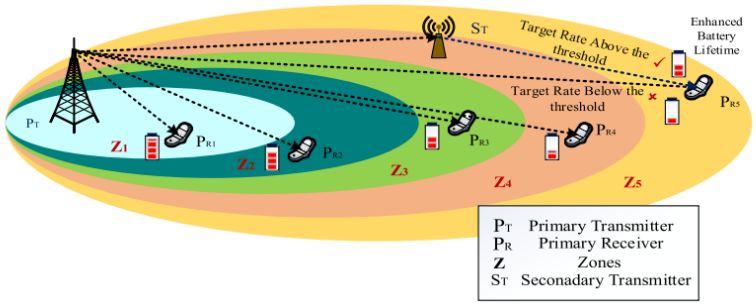
2.3.7. Χαμηλή Κατανάλωση Ενέργειας

Τα δίκτυα 6G αναμένεται να παρέχουν υψηλή απόδοση με χαμηλή κατανάλωση ενέργειας και καλύτερη απόδοση. Τα επόμενα χρόνια, η επισκεψιμότητα των χρηστών θα είναι πολύ υψηλή με τον τεράστιο αριθμό συνδεδεμένων χρηστών. Οι νέες εφαρμογές θα καταναλώνουν υψηλή ισχύ για τη λειτουργία τους. Έτσι, η βελτιστοποίηση ισχύος είναι πρωτίστως απαραίτητη. Απαιτείται η εφαρμογή διαφόρων νέων τεχνικών που μπορούν να βελτιστοποιήσουν το επίπεδο κατανάλωσης ενέργειας των συσκευών και να ενισχύσουν την ενεργειακή απόδοση του δικτύου. Τεχνικές όπως η E2ARC θα βοηθήσουν στη βελτιστοποίηση του επιπέδου κατανάλωσης ενέργειας των συσκευών.

Στο E2ARC, τα μπλοκ πόρων κατανέμονται προσαρμοστικά στον χρήστη με βάση τη ζήτηση και έτσι βελτιστοποιούν το επίπεδο κατανάλωσης ενέργειας. Προηγουμένως, στα δίκτυα 5G, η κοινή χρήση φάσματος έχει προωθηθεί για την αποτελεσματική χρήση του διαθέσιμου φάσματος και την αύξηση της απόδοσης του φάσματος. Το D2D και η αναμετάδοση βοήθησαν στη βελτιστοποίηση της κατανάλωσης ενέργειας με την εκφόρτωση της πυκνής ζήτησης δικτύου. Στο 6G, πρέπει να υιοθετηθούν αρκετές προσαρμοστικές τεχνικές για την εξασφάλιση χαμηλής κατανάλωσης ενέργειας. Απαιτούνται διάφορα συστήματα και αλγόριθμοι με βάση την τεχνητή νοημοσύνη που θα είναι χαμηλού κόστους και θα καταναλώνουν λιγότερη ενέργεια.

2.3.8. Υψηλή Ζωή Μπαταρίας

Η περιορισμένη διάρκεια ζωής της μπαταρίας των συσκευών θέτει ένα εμπόδιο στην υιοθέτηση σύγχρονων εφαρμογών. Τα δίκτυα 6G αναμένεται να διευκολύνουν μια ποικιλία νέων εφαρμογών. Έτσι, οι ερευνητές πρέπει να επικεντρωθούν στην ενίσχυση της διάρκειας ζωής της μπαταρίας των συσκευών, έτσι ώστε με μία φορά φόρτιση, η συσκευή να μπορεί να διατηρήσει έως και μία εβδομάδα. Η βελτιωμένη διάρκεια ζωής της μπαταρίας θα κάνει επίσης το δίκτυο ενεργειακά αποδοτικό. Μια τεχνική όπως το ZBGreen βοηθά στην παράταση της διάρκειας ζωής της μπαταρίας της συσκευής με την ανάπτυξη ενός δευτερεύοντος πομπού για μια απομακρυσμένη συσκευή. Στο ZBGreen, ένα δίκτυο που βασίζεται στην κοινή χρήση φάσματος λειτουργεί με έναν πρωτεύοντα πομπό που μπορεί να σχηματίσει πολλαπλά ζεύγη D2D με διαφορετικούς δέκτες που κατανέμονται γύρω από αυτόν σε διαφορετικές ζώνες. Αυτές οι ζώνες σχηματίζονται με βάση την απόσταση μεταξύ των δεκτών και του πομπού όπως φαίνεται στο επόμενο σχήμα. Διάφοροι κύριοι δέκτες (PR1, PR2, PR3, PR4 και PR5) κατανέμονται σε διαφορετικές ζώνες (Z1, Z2, Z3, Z4 και Z5) γύρω από έναν κύριο πομπό (PT). Εξετάζονται επίσης δευτερεύοντες πομποί (ST) (που λειτουργούν ως ρελέ). Αυτά τα STs θα μεταδώσουν σε συνεργασία με το PT για την εκπλήρωση των εκάστοτε στόχων.



Εικόνα 8: Primary and Secondary Transmitter

2.3.9. Επαυξημένη, Εικονική και Μικτή Πραγματικότητα

Η εικονική, επαυξημένη και μικτή πραγματικότητα είναι οι κρίσιμες εφαρμογές των συστημάτων επόμενης γενιάς. Η εικονική πραγματικότητα περιλαμβάνει πλήρη εμβάπτιση του χρήστη στον τεχνητό κόσμο ή στον ψηφιακό κόσμο. Η επαυξημένη πραγματικότητα ασχολείται με την απασχόληση ψηφιακών αντικειμένων σε σενάρια πραγματικού κόσμου. Για παράδειγμα, το Google Glass είναι ένα ακουστικό βασισμένο σε επαυξημένη πραγματικότητα που θα εμφανίζει ψηφιακό περιεχόμενο σε μια μικρή οθόνη. Αυτές οι τεχνολογίες θα απαιτούν υψηλούς ρυθμούς δεδομένων για αδιάλειπτες υπηρεσίες, οι οποίες μπορούν να επιτευχθούν με τη βοήθεια συχνοτήτων Terahertz ή ζωνών ορατού φωτός. Στην περίπτωση του 6G οι εφαρμογές AR, VR και MR θα αποτελέσουν κρίσιμο παράγοντα της απτικής τεχνολογίας. Η εικονική πραγματικότητα με απτική τεχνολογία θα βοηθήσει στη δημιουργία εικονικών περιβαλλόντων του απομακρυσμένου χώρου. Θα παράγει ένα συναρπαστικό περιβάλλον, όπου ένα άτομο μπορεί να αισθανθεί τα αντικείμενα με ρεαλιστικό τρόπο. Επιπλέον, η επαυξημένη

πραγματικότητα σε συνδυασμό με την απτική επικοινωνία θα βοηθήσει στην εκτέλεση διαφόρων λειτουργιών σε μακρινά μέρη.

2.3.10. Διαδίκτυο των πάντων (Internet of Everything)

Το IoE (Internet of Everything) είναι μια επέκταση του IoT, το οποίο θα συνδέει έξυπνα τα άτομα, τα φυσικά αντικείμενα, τις διαδικασίες και τα δεδομένα. Το IoE θα είναι πιο περίπλοκο με μεγάλο αριθμό συσκευών και ανθρώπων συνδεδεμένων μεταξύ τους. Βασίζεται στην ιδέα ότι όλα όσα μας περιβάλλουν είναι έξυπνα. Είναι ένας αυτόνομος και απρόσκοπτος συντονισμός μεγάλου αριθμού υπολογιστικών συσκευών. Το IoE περιλαμβάνει διάφορες γνωστές εφαρμογές όπως το Internet of Vehicles (IoV), το Unmanned Aerial Vehicle (UAV) και το Internet of Robots (IoR).

2.4. Μελλοντικές Προκλήσεις 6G

Πρόσφατα, η THz επικοινωνία αναδεικνύεται ως η πλέον δελεαστική τεχνολογία του δικτύου 6G, που θα επεκτείνει την ικανότητα του συστήματος παρέχοντας περισσότερες σειρές φάσματος. Ωστόσο, αυτές οι συχνότητες είναι ευαίσθητες σε μπλοκαρίσματα και υψηλές απώλειες απορρόφησης, οι οποίες μπορεί να περιορίσουν την ανάπτυξή τους μόνο σε επικοινωνίες μικρής εμβέλειας. Ως εκ τούτου, οι ερευνητές θα πρέπει να επικεντρωθούν στη διερεύνηση των ικανοτήτων των συσκευών, ώστε να λειτουργούν σε τέτοιες υψηλές συχνότητες και να τις τροποποιούν ανάλογα, ώστε να μπορεί να επιτευχθεί κέρδος υψηλής απόδοσης. Επιπλέον, τα

παρόντα μοντέλα για τις εκτιμήσεις καναλιών δεν επαρκούν για τον προσδιορισμό της αβέβαιης και εξαιρετικά ποικίλης φύσης των υψηλότερων συχνοτήτων. Έτσι, είναι πρωτίστως απαραίτητο να αναπτυχθούν νέα μοντέλα καναλιών και διάδοσης για την εκτίμηση της συμπεριφοράς τέτοιων πολύπλοκων περιβαλλόντων. Το 6G υποτίθεται ότι συνεργάζεται με δίκτυα διαστήματος, αέρα, εδάφους και νερού. Ως εκ τούτου, απαιτούνται ευέλικτα σχήματα και δυναμικά πρωτόκολλα που μπορούν να προσαρμοστούν ανάλογα με το περιβάλλον για την παροχή απρόσκοπτης συνδεσιμότητας και διαλειτουργικότητας δικτύου.

Επιπροσθέτως, το 6G θα ενσωματώσει διαφορετικά δίκτυα, όπως επίγεια και δορυφορικά δίκτυα για την επίτευξη εκτεταμένης κάλυψης και υψηλής κινητικότητας. Ωστόσο, η πραγματοποίηση της δορυφορικής επικοινωνίας με την επίγεια επικοινωνία είναι αρκετά δύσκολη όσον αφορά τη μεγάλη καθυστέρηση, την προφανή μετατόπιση Doppler και τις δια-δορυφορικές συνδέσεις. Αυτές οι προκλήσεις μπορούν να επηρεάσουν τις διαδικασίες συγχρονισμού, τυχαίας πρόσβασης, ανίχνευσης σήματος, απόδοσης λήψης και ούτω καθεξής. Έτσι, για την επιτυχή απόκτηση των πλεονεκτημάτων ενός ολοκληρωμένου συστήματος επικοινωνίας, πρέπει να παρέχονται προηγμένες τεχνικές που μπορούν να ελαχιστοποιήσουν τις προκλήσεις που σχετίζονται με αυτές.

Όσο αφορά την τεχνική edge computing, μπορεί να αυξήσει δραστικά την υπολογιστική ικανότητα του δικτύου. Αλλά με την περιορισμένη διαθεσιμότητα πόρων και αποθήκευσης, είναι πολύ δύσκολο να λειτουργούν πολύ περίπλοκοι αλγόριθμοι που βασίζονται σε τεχνητή νοημοσύνη και

απαιτούν τεράστια συλλογή δεδομένων. Έτσι, απαιτούνται έρευνες για τον σχεδιασμό λεπτών και νέων αλγορίθμων τεχνητής νοημοσύνης προς την συγκεκριμένη κατεύθυνση. Επιπλέον, απαιτείται η ανάπτυξη αποτελεσματικών τεχνικών προγραμματισμού για την ενίσχυση της απόδοσης του συστήματος.

Το 6G θα βασίζεται σε AI, το οποίο απαιτεί υψηλή ισχύ για τον υπολογισμό και τον χειρισμό σύνθετων δεδομένων. Έτσι, η ενεργειακή απόδοση και η βελτιστοποίηση ισχύος θα είναι οι σημαντικές πτυχές του δικτύου 6G. Διάφορες εφαρμογές όπως η ολογραφική απεικόνιση, η απτική επικοινωνία, το AR, το VR και το MR είναι εφαρμογές οι οποίες έχουν υψηλές απαιτήσεις ενέργειας. Επομένως, πρέπει να αναπτυχθούν τεχνικές βελτιστοποίησης ισχύος και ενεργειακής απόδοσης για τα δίκτυα επόμενης γενιάς.

Τέλος, η ασφάλεια των δεδομένων και το απόρρητο είναι τα κυριότερα ζητήματα των ασύρματων δικτύων. Οι εξέχουσες τεχνολογίες του δικτύου 6G απαιτούν τεράστια συλλογή και μετάδοση δεδομένων. Προκειμένου να διασφαλιστεί η ασφάλεια και το απόρρητο, απαιτούνται διάφορες τεχνικές ασφαλείας και αλγόριθμοι κρυπτογράφησης. Τα δίκτυα 6G υποτίθεται ότι παρέχουν ασυλία έναντι νέων απειλών. Μελλοντικά, η οικονομία και η κοινωνία θα εξαρτώνται πλήρως από τον ψηφιακό κόσμο. Έτσι, μια αξιόπιστη αρχιτεκτονική με την απαιτούμενη μυστικότητα και ιδιωτικότητα είναι υποχρεωτική.

ΚΕΦΑΛΑΙΟ 3: ΠΕΙΡΑΜΑΤΙΚΗ ΔΙΑΔΙΚΑΣΙΑ

Στα πλαίσια του πρακτικού μέρους της πτυχιακής εργασίας, χρησιμοποιήθηκε ο ns-3 simulator και πιο συγκεκριμένα το NR module [8], το οποίο χρησιμοποιείται για εξομοίωση 3GPP NSA κυψελωτών δικτύων. Ουσιαστικά, εκτελέστηκαν έτοιμα παραδείγματα τα οποία ήταν διαθέσιμα [9], για τα οποία στην συνέχεια παρατίθεται ο κώδικάς που χρησιμοποιήθηκε με σχόλια, καθώς και τα αντίστοιχα πειραματικά αποτελέσματα που προέκυψαν.

Στο σημείο αυτό αξίζει να αναφερθεί ότι η πειραματική διαδικασία πραγματοποιήθηκε σε λειτουργικό σύστημα Ubuntu 20.04.

3.1. Παράδειγμα 1

Παρατίθεται ο κώδικας, που χρησιμοποιήθηκε:

```
/**
 * \ingroup examples
 * \file cttc-3gpp-channel-simple-ran.cc
 * \brief Simple RAN
 *
 * This example describes how to setup a simulation using the 3GPP channel model
 * from TR 38.900. This example consists of a simple topology of 1 UE and 1 gNb,
 * and only NR RAN part is simulated. One Bandwidth part and one CC are defined.
 * A packet is created and directly sent to gNb device by SendPacket function.
 * Then several functions are connected to PDCP and RLC traces and the delay is
 * printed.
 */

#include "ns3/antenna-module.h"
#include "ns3/config-store.h"
#include "ns3/core-module.h"
#include "ns3/eps-bearer-tag.h"
#include "ns3/grid-scenario-helper.h"
#include "ns3/internet-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/log.h"
#include "ns3/mobility-module.h"
#include "ns3/network-module.h"
```

```
#include "ns3/nr-helper.h"
#include "ns3/nr-module.h"
#include "ns3/nr-point-to-point-epc-helper.h"

using namespace ns3;

/*
 * Enable the logs of the file by enabling the component "Cttc3gppChannelSimpleRan",
 * in this way:
 * $ export NS_LOG="Cttc3gppChannelSimpleRan=level_info|prefix_func|prefix_time"
 */
NS_LOG_COMPONENT_DEFINE("Cttc3gppChannelSimpleRan");

static bool g_rxPdcpCallbackCalled = false;
static bool g_rxRxRlcPDUCallbackCalled = false;

/**
 * Function creates a single packet and directly calls the function send
 * of a device to send the packet to the destination address.
 * @param device Device that will send the packet to the destination address.
 * @param addr Destination address for a packet.
 * @param packetSize The packet size.
 */
static void
SendPacket(Ptr<NetDevice> device, Address& addr, uint32_t packetSize)
{
   Ptr<Packet> pkt = Create<Packet>(packetSize);
   Ipv4Header ipv4Header;
   ipv4Header.SetProtocol(UdpL4Protocol::PROT_NUMBER);
   pkt->AddHeader(ipv4Header);
   EpsBearerTag tag(1, 1);
   pkt->AddPacketTag(tag);
   device->Send(pkt, addr, Ipv4L3Protocol::PROT_NUMBER);
}

/**
 * Function that prints out PDCP delay. This function is designed as a callback
 * for PDCP trace source.
 * @param path The path that matches the trace source
 * @param rnti RNTI of UE
 * @param lcid logical channel id
 * @param bytes PDCP PDU size in bytes
 * @param pdcpDelay PDCP delay
 */
void
RxPdcpPDU(std::string path, uint16_t rnti, uint8_t lcid, uint32_t bytes, uint64_t pdcpDelay)
{
   std::cout << "\n Packet PDCP delay:" << pdcpDelay << "\n";
   g_rxPdcpCallbackCalled = true;
}

/**
 * Function that prints out RLC statistics, such as RNTI, lcId, RLC PDU size,
 * delay. This function is designed as a callback
 * for RLC trace source.
 * @param path The path that matches the trace source
 * @param rnti RNTI of UE
 * @param lcid logical channel id
 * @param bytes RLC PDU size in bytes
 * @param rlcDelay RLC PDU delay
 */
void
RxRlcPDU(std::string path, uint16_t rnti, uint8_t lcid, uint32_t bytes, uint64_t rlcDelay)
{
```

```
    std::cout << "\n\n Data received at RLC layer at:" << Simulator::Now() << std::endl;
    std::cout << "\n rnti:" << rnti << std::endl;
    std::cout << "\n lcid:" << (unsigned)lcid << std::endl;
    std::cout << "\n bytes :" << bytes << std::endl;
    std::cout << "\n delay :" << rlcDelay << std::endl;
    g_rxRxRlcPDUCallbackCalled = true;
}

/**
 * Function that connects PDCP and RLC traces to the corresponding trace sources.
 */
void
ConnectPdcpRlcTraces()
{
    Config::Connect("/NodeList/*/DeviceList/*/LteUeRrc/DataRadioBearerMap/1/LtePdcp/RxPDU",
                    MakeCallback(&RxPdcpPDU));

    Config::Connect("/NodeList/*/DeviceList/*/LteUeRrc/DataRadioBearerMap/1/LteRlc/RxPDU",
                    MakeCallback(&RxRlcPDU));
}

/**
 * Function that connects UL PDCP and RLC traces to the corresponding trace sources.
 */
void
ConnectUlPdcpRlcTraces()
{
    Config::Connect("/NodeList/*/DeviceList/*/LteEnbRrc/UeMap/*/DataRadioBearerMap/*/LtePdcp/RxPDU",
                    MakeCallback(&RxPdcpPDU));

    Config::Connect("/NodeList/*/DeviceList/*/LteEnbRrc/UeMap/*/DataRadioBearerMap/*/LteRlc/RxPDU",
                    MakeCallback(&RxRlcPDU));
}

int
main(int argc, char* argv[])
{
    uint16_t numerologyBwp1 = 0;
    uint32_t udpPacketSize = 1000;
    double centralFrequencyBand1 = 28e9;
    double bandwidthBand1 = 400e6;
    uint16_t gNbNum = 1;
    uint16_t ueNumPergNb = 1;
    bool enableUl = false;

    Time sendPacketTime = Seconds(0.4);

    CommandLine cmd(__FILE__);
    cmd.AddValue("numerologyBwp1", "The numerology to be used in bandwidth part 1", numerologyBwp1);
    cmd.AddValue("centralFrequencyBand1",
                 "The system frequency to be used in band 1",
                 centralFrequencyBand1);
    cmd.AddValue("bandwidthBand1", "The system bandwidth to be used in band 1", bandwidthBand1);
    cmd.AddValue("packetSize", "packet size in bytes", udpPacketSize);
    cmd.AddValue("enableUl", "Enable Uplink", enableUl);
    cmd.Parse(argc, argv);

    int64_t randomStream = 1;
    // Create the scenario
    GridScenarioHelper gridScenario;
    gridScenario.SetRows(1);
    gridScenario.SetColumns(gNbNum);
    gridScenario.SetHorizontalBsDistance(5.0);
    gridScenario.SetBsHeight(10.0);
```

```
gridScenario.SetUtHeight(1.5);
// must be set before BS number
gridScenario.SetSectorization(GridScenarioHelper::SINGLE);
gridScenario.SetBsNumber(gNbNum);
gridScenario.SetUtNumber(ueNumPergNb * gNbNum);
gridScenario.SetScenarioHeight(3); // Create a 3x3 scenario where the UE will
gridScenario.SetScenarioLength(3); // be distribuited.
randomStream += gridScenario.AssignStreams(randomStream);
gridScenario.CreateScenario();

Ptr<NrPointToPointEpcHelper> epcHelper = CreateObject<NrPointToPointEpcHelper>();
Ptr<IdealBeamformingHelper> idealBeamformingHelper = CreateObject<IdealBeamformingHelper>();
Ptr<NrHelper> nrHelper = CreateObject<NrHelper>();

nrHelper->SetBeamformingHelper(idealBeamformingHelper);
nrHelper->SetEpcHelper(epcHelper);

// Create one operational band containing one CC with one bandwidth part
BandwidthPartInfoPtrVector allBwps;
CcBwpCreator ccBwpCreator;
const uint8_t numCcPerBand = 1;

// Create the configuration for the CcBwpHelper
CcBwpCreator::SimpleOperationBandConf bandConf1(centralFrequencyBand1,
                                    bandwidthBand1,
                                    numCcPerBand,
                                    BandwidthPartInfo::UMi_StreetCanyon_LoS);

// By using the configuration created, it is time to make the operation band
OperationBandInfo band1 = ccBwpCreator.CreateOperationBandContiguousCc(bandConf1);

Config::SetDefault("ns3::ThreeGppChannelModel::UpdatePeriod", TimeValue(MilliSeconds(0)));
nrHelper->SetSchedulerAttribute("FixedMcsDl", BooleanValue(true));
nrHelper->SetSchedulerAttribute("StartingMcsDl", UintegerValue(28));
nrHelper->SetChannelConditionModelAttribute("UpdatePeriod", TimeValue(MilliSeconds(0)));
nrHelper->SetPathlossAttribute("ShadowingEnabled", BooleanValue(false));

nrHelper->InitializeOperationBand(&band1);
allBwps = CcBwpCreator::GetAllBwps({band1});

// Beamforming method
idealBeamformingHelper->SetAttribute("BeamformingMethod",
                          TypeIdValue(DirectPathBeamforming::GetTypeId()));

// Antennas for all the UEs
nrHelper->SetUeAntennaAttribute("NumRows", UintegerValue(2));
nrHelper->SetUeAntennaAttribute("NumColumns", UintegerValue(4));
nrHelper->SetUeAntennaAttribute("AntennaElement",
                        PointerValue(CreateObject<IsotropicAntennaModel>()));

// Antennas for all the gNbs
nrHelper->SetGnbAntennaAttribute("NumRows", UintegerValue(4));
nrHelper->SetGnbAntennaAttribute("NumColumns", UintegerValue(8));
nrHelper->SetGnbAntennaAttribute("AntennaElement",
                         PointerValue(CreateObject<IsotropicAntennaModel>()));

// Install and get the pointers to the NetDevices
NetDeviceContainer enbNetDev =
   nrHelper->InstallGnbDevice(gridScenario.GetBaseStations(), allBwps);
NetDeviceContainer ueNetDev =
   nrHelper->InstallUeDevice(gridScenario.GetUserTerminals(), allBwps);

randomStream += nrHelper->AssignStreams(enbNetDev, randomStream);
randomStream += nrHelper->AssignStreams(ueNetDev, randomStream);
```

```
// Set the attribute of the netdevice (enbNetDev.Get (0)) and bandwidth part (0)
nrHelper->GetGnbPhy(enbNetDev.Get(0), 0)
   ->SetAttribute("Numerology", UintegerValue(numerologyBwp1));

for (auto it = enbNetDev.Begin(); it != enbNetDev.End(); ++it)
{
   DynamicCast<NrGnbNetDevice>(*it)->UpdateConfig();
}

for (auto it = ueNetDev.Begin(); it != ueNetDev.End(); ++it)
{
   DynamicCast<NrUeNetDevice>(*it)->UpdateConfig();
}

InternetStackHelper internet;
internet.Install(gridScenario.GetUserTerminals());
Ipv4InterfaceContainer ueIpIface;
ueIpIface = epcHelper->AssignUeIpv4Address(NetDeviceContainer(ueNetDev));

if (enableUl)
{
   Simulator::Schedule(sendPacketTime,
                       &SendPacket,
                       ueNetDev.Get(0),
                       enbNetDev.Get(0)->GetAddress(),
                       udpPacketSize);
}
else
{
   Simulator::Schedule(sendPacketTime,
                       &SendPacket,
                       enbNetDev.Get(0),
                       ueNetDev.Get(0)->GetAddress(),
                       udpPacketSize);
}

// attach UEs to the closest eNB
nrHelper->AttachToClosestEnb(ueNetDev, enbNetDev);

if (enableUl)
{
   std::cout << "\n Sending data in uplink." << std::endl;
   Simulator::Schedule(Seconds(0.2), &ConnectUlPdcpRlcTraces);
}
else
{
   std::cout << "\n Sending data in downlink." << std::endl;
   Simulator::Schedule(Seconds(0.2), &ConnectPdcpRlcTraces);
}

nrHelper->EnableTraces();

Simulator::Stop(Seconds(1));
Simulator::Run();
Simulator::Destroy();

if (g_rxPdcpCallbackCalled && g_rxRxRlcPDUCallbackCalled)
{
   return EXIT_SUCCESS;
}
else
{
   return EXIT_FAILURE;
```

```
  }
}
```

Και τα αντίστοιχα πειραματικά αποτελέσματα:

```
test@ubuntu:~/Desktop/ns-3-dev$ ./ns3 run cttc-3gpp-channel-simple-ran
 Sending data in downlink.

 Data received at RLC layer at:+4.02243e+08ns
 rnti:1
 lcid:3
 bytes :1024
 delay :2242855
 Packet PDCP delay:2242855
```

Εικόνα 9: Πειραματικά Αποτελέσματα Παραδείγματος 1

3.2. Παράδειγμα 2

Παρατίθεται ο κώδικας, που χρησιμοποιήθηκε:

```
/**
 * \file cttc-3gpp-channel-nums.cc
 * \ingroup examples
 * \brief Simple topology numerologies example.
 *
 * This example allows users to configure the numerology and test the end-to-end
 * performance for different numerologies. In the following figure we illustrate
 * the simulation setup.
 *
 * For example, UDP packet generation rate can be configured by setting
 * "--lambda=1000". The numerology can be toggled by the argument,
 * e.g. "--numerology=1". Additionally, in this example two arguments
 * are added "bandwidth" and "frequency", both in Hz units. The modulation
 * scheme of this example is in test mode, and it is fixed to 28.
 *
 * By default, the program uses the 3GPP channel model, without shadowing and with
 * line of sight ('l') option. The program runs for 0.4 seconds and one single
 * packet is to be transmitted. The packet size can be configured by using the
 * following parameter: "--packetSize=1000".
 *
 * This simulation prints the output to the terminal and also to the file which
 * is named by default "cttc-3gpp-channel-nums-fdm-output" and which is by
 * default placed in the root directory of the project.
```

```
 *
 * To run the simulation with the default configuration one shall run the
 * following in the command line:
 *
 * ./ns3 run cttc-3gpp-channel-nums
 *
 */

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("3gppChannelNumerologiesExample");

int
main(int argc, char* argv[])
{
   // enable logging or not
   bool logging = false;
   if (logging)
   {
      LogComponentEnable("UdpClient", LOG_LEVEL_INFO);
      LogComponentEnable("UdpServer", LOG_LEVEL_INFO);
      LogComponentEnable("LtePdcp", LOG_LEVEL_INFO);
   }

   // set simulation time and mobility
   double simTime = 1;          // seconds
   double udpAppStartTime = 0.4; // seconds

   // other simulation parameters default values
   uint16_t numerology = 0;

   uint16_t gNbNum = 1;
   uint16_t ueNumPergNb = 1;

   double centralFrequency = 7e9;
   double bandwidth = 100e6;
   double txPower = 14;
   double lambda = 1000;
   uint32_t udpPacketSize = 1000;
   bool udpFullBuffer = true;
   uint8_t fixedMcs = 28;
   bool useFixedMcs = true;
   bool singleUeTopology = true;
   // Where we will store the output files.
   std::string simTag = "default";
   std::string outputDir = "./";

   CommandLine cmd(__FILE__);
   cmd.AddValue("gNbNum", "The number of gNbs in multiple-ue topology", gNbNum);
   cmd.AddValue("ueNumPergNb", "The number of UE per gNb in multiple-ue topology", ueNumPergNb);
   cmd.AddValue("numerology", "The numerology to be used.", numerology);
   cmd.AddValue("txPower", "Tx power to be configured to gNB", txPower);
   cmd.AddValue("simTag",
                "tag to be appended to output filenames to distinguish simulation campaigns",
                simTag);
   cmd.AddValue("outputDir", "directory where to store simulation results", outputDir);
   cmd.AddValue("frequency", "The system frequency", centralFrequency);
   cmd.AddValue("bandwidth", "The system bandwidth", bandwidth);
   cmd.AddValue("udpPacketSize", "UDP packet size in bytes", udpPacketSize);
   cmd.AddValue("lambda", "Number of UDP packets per second", lambda);
   cmd.AddValue("udpFullBuffer",
                "Whether to set the full buffer traffic; if this parameter is set then the "
                "udpInterval parameter"
                "will be neglected",
```

```
        udpFullBuffer);
cmd.AddValue(
   "fixedMcs",
   "The fixed MCS that will be used in this example if useFixedMcs is configured to true (1).",
   fixedMcs);
cmd.AddValue("useFixedMcs",
        "Whether to use fixed mcs, normally used for testing purposes",
        useFixedMcs);
cmd.AddValue("singleUeTopology",
        "If true, the example uses a predefined topology with one UE and one gNB; "
        "if false, the example creates a grid of gNBs with a number of UEs attached",
        singleUeTopology);

cmd.Parse(argc, argv);

NS_ASSERT(ueNumPergNb > 0);

// setup the nr simulation
Ptr<NrHelper> nrHelper = CreateObject<NrHelper>();

/*
 * Spectrum division. We create one operation band with one component carrier
 * (CC) which occupies the whole operation band bandwidth. The CC contains a
 * single Bandwidth Part (BWP). This BWP occupies the whole CC band.
 * Both operational bands will use the StreetCanyon channel modeling.
 */
CcBwpCreator ccBwpCreator;
const uint8_t numCcPerBand = 1; // in this example, both bands have a single CC
BandwidthPartInfo::Scenario scenario = BandwidthPartInfo::RMa_LoS;
if (ueNumPergNb > 1)
{
   scenario = BandwidthPartInfo::InH_OfficeOpen;
}

// Create the configuration for the CcBwpHelper. SimpleOperationBandConf creates
// a single BWP per CC
CcBwpCreator::SimpleOperationBandConf bandConf(centralFrequency,
                                       bandwidth,
                                       numCcPerBand,
                                       scenario);

// By using the configuration created, it is time to make the operation bands
OperationBandInfo band = ccBwpCreator.CreateOperationBandContiguousCc(bandConf);

/*
 * Initialize channel and pathloss, plus other things inside band1. If needed,
 * the band configuration can be done manually, but we leave it for more
 * sophisticated examples. For the moment, this method will take care
 * of all the spectrum initialization needs.
 */
nrHelper->InitializeOperationBand(&band);

BandwidthPartInfoPtrVector allBwps = CcBwpCreator::GetAllBwps({band});

/*
 * Continue setting the parameters which are common to all the nodes, like the
 * gNB transmit power or numerology.
 */
nrHelper->SetGnbPhyAttribute("TxPower", DoubleValue(txPower));
nrHelper->SetGnbPhyAttribute("Numerology", UintegerValue(numerology));

// Scheduler
nrHelper->SetSchedulerTypeId(TypeId::LookupByName("ns3::NrMacSchedulerTdmaRR"));
nrHelper->SetSchedulerAttribute("FixedMcsDl", BooleanValue(useFixedMcs));
```

```
nrHelper->SetSchedulerAttribute("FixedMcsUl", BooleanValue(useFixedMcs));

if (useFixedMcs == true)
{
   nrHelper->SetSchedulerAttribute("StartingMcsDl", UintegerValue(fixedMcs));
   nrHelper->SetSchedulerAttribute("StartingMcsUl", UintegerValue(fixedMcs));
}

Config::SetDefault("ns3::LteRlcUm::MaxTxBufferSize", UintegerValue(999999999));

// Antennas for all the UEs
nrHelper->SetUeAntennaAttribute("NumRows", UintegerValue(2));
nrHelper->SetUeAntennaAttribute("NumColumns", UintegerValue(4));
nrHelper->SetUeAntennaAttribute("AntennaElement",
                        PointerValue(CreateObject<IsotropicAntennaModel>()));

// Antennas for all the gNbs
nrHelper->SetGnbAntennaAttribute("NumRows", UintegerValue(4));
nrHelper->SetGnbAntennaAttribute("NumColumns", UintegerValue(8));
nrHelper->SetGnbAntennaAttribute("AntennaElement",
                         PointerValue(CreateObject<ThreeGppAntennaModel>()));

// Beamforming method
Ptr<IdealBeamformingHelper> idealBeamformingHelper = CreateObject<IdealBeamformingHelper>();
idealBeamformingHelper->SetAttribute("BeamformingMethod",
                           TypeIdValue(DirectPathBeamforming::GetTypeId()));
nrHelper->SetBeamformingHelper(idealBeamformingHelper);

Config::SetDefault("ns3::ThreeGppChannelModel::UpdatePeriod", TimeValue(MilliSeconds(0)));
//  nrHelper->SetChannelConditionModelAttribute ("UpdatePeriod", TimeValue (MilliSeconds (0)));
nrHelper->SetPathlossAttribute("ShadowingEnabled", BooleanValue(false));

// Error Model: UE and GNB with same spectrum error model.
nrHelper->SetUlErrorModel("ns3::NrEesmIrT1");
nrHelper->SetDlErrorModel("ns3::NrEesmIrT1");

// Both DL and UL AMC will have the same model behind.
nrHelper->SetGnbDlAmcAttribute(
   "AmcModel",
   EnumValue(NrAmc::ErrorModel)); // NrAmc::ShannonModel or NrAmc::ErrorModel
nrHelper->SetGnbUlAmcAttribute(
   "AmcModel",
   EnumValue(NrAmc::ErrorModel)); // NrAmc::ShannonModel or NrAmc::ErrorModel

// Create EPC helper
Ptr<NrPointToPointEpcHelper> epcHelper = CreateObject<NrPointToPointEpcHelper>();
nrHelper->SetEpcHelper(epcHelper);
// Core latency
epcHelper->SetAttribute("S1uLinkDelay", TimeValue(MilliSeconds(0)));

// gNb routing between Bearer and bandwidh part
uint32_t bwpIdForBearer = 0;
nrHelper->SetGnbBwpManagerAlgorithmAttribute("GBR_CONV_VOICE", UintegerValue(bwpIdForBearer));

// Initialize nrHelper
nrHelper->Initialize();

/*
 *  Create the gNB and UE nodes according to the network topology
 */
NodeContainer gNbNodes;
NodeContainer ueNodes;
MobilityHelper mobility;
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
```

```
Ptr<ListPositionAllocator> bsPositionAlloc = CreateObject<ListPositionAllocator>();
Ptr<ListPositionAllocator> utPositionAlloc = CreateObject<ListPositionAllocator>();

const double gNbHeight = 10;
const double ueHeight = 1.5;

if (singleUeTopology)
{
  gNbNodes.Create(1);
  ueNodes.Create(1);
  gNbNum = 1;
  ueNumPergNb = 1;

  mobility.Install(gNbNodes);
  mobility.Install(ueNodes);
  bsPositionAlloc->Add(Vector(0.0, 0.0, gNbHeight));
  utPositionAlloc->Add(Vector(0.0, 30.0, ueHeight));
}
else
{
  gNbNodes.Create(gNbNum);
  ueNodes.Create(ueNumPergNb * gNbNum);

  int32_t yValue = 0.0;
  for (uint32_t i = 1; i <= gNbNodes.GetN(); ++i)
  {
    // 2.0, -2.0, 6.0, -6.0, 10.0, -10.0, ....
    if (i % 2 != 0)
    {
      yValue = static_cast<int>(i) * 30;
    }
    else
    {
      yValue = -yValue;
    }

    bsPositionAlloc->Add(Vector(0.0, yValue, gNbHeight));

    // 1.0, -1.0, 3.0, -3.0, 5.0, -5.0, ...
    double xValue = 0.0;
    for (uint16_t j = 1; j <= ueNumPergNb; ++j)
    {
      if (j % 2 != 0)
      {
        xValue = j;
      }
      else
      {
        xValue = -xValue;
      }

      if (yValue > 0)
      {
        utPositionAlloc->Add(Vector(xValue, 1, ueHeight));
      }
      else
      {
        utPositionAlloc->Add(Vector(xValue, -1, ueHeight));
      }
    }
  }
}
mobility.SetPositionAllocator(bsPositionAlloc);
mobility.Install(gNbNodes);
```

```
mobility.SetPositionAllocator(utPositionAlloc);
mobility.Install(ueNodes);

// Install nr net devices
NetDeviceContainer gNbNetDev = nrHelper->InstallGnbDevice(gNbNodes, allBwps);

NetDeviceContainer ueNetDev = nrHelper->InstallUeDevice(ueNodes, allBwps);

int64_t randomStream = 1;
randomStream += nrHelper->AssignStreams(gNbNetDev, randomStream);
randomStream += nrHelper->AssignStreams(ueNetDev, randomStream);

// When all the configuration is done, explicitly call UpdateConfig ()

for (auto it = gNbNetDev.Begin(); it != gNbNetDev.End(); ++it)
{
   DynamicCast<NrGnbNetDevice>(*it)->UpdateConfig();
}

for (auto it = ueNetDev.Begin(); it != ueNetDev.End(); ++it)
{
   DynamicCast<NrUeNetDevice>(*it)->UpdateConfig();
}

// create the internet and install the IP stack on the UEs
// get SGW/PGW and create a single RemoteHost
Ptr<Node> pgw = epcHelper->GetPgwNode();
NodeContainer remoteHostContainer;
remoteHostContainer.Create(1);
Ptr<Node> remoteHost = remoteHostContainer.Get(0);
InternetStackHelper internet;
internet.Install(remoteHostContainer);

// connect a remoteHost to pgw. Setup routing too
PointToPointHelper p2ph;
p2ph.SetDeviceAttribute("DataRate", DataRateValue(DataRate("100Gb/s")));
p2ph.SetDeviceAttribute("Mtu", UintegerValue(2500));
p2ph.SetChannelAttribute("Delay", TimeValue(Seconds(0.000)));
NetDeviceContainer internetDevices = p2ph.Install(pgw, remoteHost);
Ipv4AddressHelper ipv4h;
ipv4h.SetBase("1.0.0.0", "255.0.0.0");
Ipv4InterfaceContainer internetIpIfaces = ipv4h.Assign(internetDevices);
Ipv4StaticRoutingHelper ipv4RoutingHelper;
Ptr<Ipv4StaticRouting> remoteHostStaticRouting =
   ipv4RoutingHelper.GetStaticRouting(remoteHost->GetObject<Ipv4>());
remoteHostStaticRouting->AddNetworkRouteTo(Ipv4Address("7.0.0.0"), Ipv4Mask("255.0.0.0"), 1);
internet.Install(ueNodes);

Ipv4InterfaceContainer ueIpIface = epcHelper->AssignUeIpv4Address(NetDeviceContainer(ueNetDev));

// Set the default gateway for the UEs
for (uint32_t j = 0; j < ueNodes.GetN(); ++j)
{
   Ptr<Ipv4StaticRouting> ueStaticRouting =
      ipv4RoutingHelper.GetStaticRouting(ueNodes.Get(j)->GetObject<Ipv4>());
   ueStaticRouting->SetDefaultRoute(epcHelper->GetUeDefaultGatewayAddress(), 1);
}

// attach UEs to the closest eNB
nrHelper->AttachToClosestEnb(ueNetDev, gNbNetDev);

// assign IP address to UEs, and install UDP downlink applications
uint16_t dlPort = 1234;
```

```
ApplicationContainer serverApps;

// The sink will always listen to the specified ports
UdpServerHelper dlPacketSinkHelper(dlPort);
serverApps.Add(dlPacketSinkHelper.Install(ueNodes.Get(0)));

UdpClientHelper dlClient;
dlClient.SetAttribute("RemotePort", UintegerValue(dlPort));
dlClient.SetAttribute("PacketSize", UintegerValue(udpPacketSize));
dlClient.SetAttribute("MaxPackets", UintegerValue(0xFFFFFFFF));
if (udpFullBuffer)
{
   double bitRate = 75000000; // 75 Mbps will saturate the NR system of 20 MHz with the
                              // NrEesmIrT1 error model
   bitRate /= ueNumPergNb;    // Divide the cell capacity among UEs
   if (bandwidth > 20e6)
   {
      bitRate *= bandwidth / 20e6;
   }
   lambda = bitRate / static_cast<double>(udpPacketSize * 8);
}
dlClient.SetAttribute("Interval", TimeValue(Seconds(1.0 / lambda)));

// The bearer that will carry low latency traffic
EpsBearer bearer(EpsBearer::GBR_CONV_VOICE);

Ptr<EpcTft> tft = Create<EpcTft>();
EpcTft::PacketFilter dlpf;
dlpf.localPortStart = dlPort;
dlpf.localPortEnd = dlPort;
tft->Add(dlpf);

/*
 * Let's install the applications!
 */
ApplicationContainer clientApps;

for (uint32_t i = 0; i < ueNodes.GetN(); ++i)
{
   Ptr<Node> ue = ueNodes.Get(i);
   Ptr<NetDevice> ueDevice = ueNetDev.Get(i);
   Address ueAddress = ueIpIface.GetAddress(i);

   // The client, who is transmitting, is installed in the remote host,
   // with destination address set to the address of the UE
   dlClient.SetAttribute("RemoteAddress", AddressValue(ueAddress));
   clientApps.Add(dlClient.Install(remoteHost));

   // Activate a dedicated bearer for the traffic type
   nrHelper->ActivateDedicatedEpsBearer(ueDevice, bearer, tft);
}

// start server and client apps
serverApps.Start(Seconds(udpAppStartTime));
clientApps.Start(Seconds(udpAppStartTime));
serverApps.Stop(Seconds(simTime));
clientApps.Stop(Seconds(simTime));

// enable the traces provided by the nr module
// nrHelper->EnableTraces();

FlowMonitorHelper flowmonHelper;
NodeContainer endpointNodes;
```

```
endpointNodes.Add(remoteHost);
endpointNodes.Add(ueNodes);

Ptr<ns3::FlowMonitor> monitor = flowmonHelper.Install(endpointNodes);
monitor->SetAttribute("DelayBinWidth", DoubleValue(0.001));
monitor->SetAttribute("JitterBinWidth", DoubleValue(0.001));
monitor->SetAttribute("PacketSizeBinWidth", DoubleValue(20));

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

// Print per-flow statistics
monitor->CheckForLostPackets();
Ptr<Ipv4FlowClassifier> classifier =
   DynamicCast<Ipv4FlowClassifier>(flowmonHelper.GetClassifier());
FlowMonitor::FlowStatsContainer stats = monitor->GetFlowStats();

double averageFlowThroughput = 0.0;
double averageFlowDelay = 0.0;

std::ofstream outFile;
std::string filename = outputDir + "/" + simTag;
outFile.open(filename.c_str(), std::ofstream::out | std::ofstream::trunc);
if (!outFile.is_open())
{
   NS_LOG_ERROR("Can't open file " << filename);
   return 1;
}
outFile.setf(std::ios_base::fixed);

for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin();
    i != stats.end();
    ++i)
{
   Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(i->first);
   std::stringstream protoStream;
   protoStream << (uint16_t)t.protocol;
   if (t.protocol == 6)
   {
      protoStream.str("TCP");
   }
   if (t.protocol == 17)
   {
      protoStream.str("UDP");
   }
   outFile << "Flow " << i->first << " (" << t.sourceAddress << ":" << t.sourcePort << " -> "
         << t.destinationAddress << ":" << t.destinationPort << ") proto "
         << protoStream.str() << "\n";
   outFile << "  Tx Packets: " << i->second.txPackets << "\n";
   outFile << "  Tx Bytes:   " << i->second.txBytes << "\n";
   outFile << "  TxOffered:  "
         << i->second.txBytes * 8.0 / (simTime - udpAppStartTime) / 1000 / 1000 << " Mbps\n";
   outFile << "  Rx Bytes:   " << i->second.rxBytes << "\n";
   if (i->second.rxPackets > 0)
   {
      // Measure the duration of the flow from receiver's perspective
      double rxDuration =
         i->second.timeLastRxPacket.GetSeconds() - i->second.timeFirstTxPacket.GetSeconds();

      averageFlowThroughput += i->second.rxBytes * 8.0 / rxDuration / 1000 / 1000;
      averageFlowDelay += 1000 * i->second.delaySum.GetSeconds() / i->second.rxPackets;

      outFile << "  Throughput: " << i->second.rxBytes * 8.0 / rxDuration / 1000 / 1000
            << " Mbps\n";
```

```
            outFile << "  Mean delay:  "
                    << 1000 * i->second.delaySum.GetSeconds() / i->second.rxPackets << " ms\n";
            // outFile << "  Mean upt:  " << i->second.uptSum / i->second.rxPackets / 1000/1000 << "
            // Mbps \n";
            outFile << "  Mean jitter:  "
                    << 1000 * i->second.jitterSum.GetSeconds() / i->second.rxPackets << " ms\n";
        }
        else
        {
            outFile << "  Throughput:  0 Mbps\n";
            outFile << "  Mean delay:  0 ms\n";
            outFile << "  Mean upt:  0  Mbps \n";
            outFile << "  Mean jitter: 0 ms\n";
        }
        outFile << "  Rx Packets: " << i->second.rxPackets << "\n";
    }

    double meanFlowThroughput = averageFlowThroughput / stats.size();
    double meanFlowDelay = averageFlowDelay / stats.size();
    Ptr<UdpServer> serverApp = serverApps.Get(0)->GetObject<UdpServer>();
    double totalUdpThroughput =
        ((serverApp->GetReceived() * udpPacketSize * 8) / (simTime - udpAppStartTime)) * 1e-6;

    outFile << "\n\n  Mean flow throughput: " << meanFlowThroughput << "\n";
    outFile << "  Mean flow delay: " << meanFlowDelay << "\n";
    outFile << "\n UDP throughput (bps) for UE with node ID 0:" << totalUdpThroughput << std::endl;

    outFile.close();

    std::ifstream f(filename.c_str());

    if (f.is_open())
    {
        std::cout << f.rdbuf();
    }

    Simulator::Destroy();

    double toleranceMeanFlowThroughput = 383.557857 * 0.0001;
    double toleranceMeanFlowDelay = 3.533664 * 0.0001;
    double toleranceUdpThroughput = 372.5066667 * 0.0001;

    // called from examples-to-run.py with all default parameters
    if (argc == 0 && (meanFlowThroughput < 383.557857 - toleranceMeanFlowThroughput ||
                      meanFlowThroughput > 383.557857 + toleranceMeanFlowThroughput ||
                      meanFlowDelay < 3.533664 - toleranceMeanFlowDelay ||
                      meanFlowDelay > 3.533664 + toleranceMeanFlowDelay ||
                      totalUdpThroughput < 372.5066667 - toleranceUdpThroughput ||
                      totalUdpThroughput > 372.5066667 + toleranceUdpThroughput))
    {
        return EXIT_FAILURE;
    }
    else
    {
        return EXIT_SUCCESS;
    }
}
```

Και τα αντίστοιχα πειραματικά αποτελέσματα:

```
test@ubuntu:~/Desktop/ns-3-dev$ ./ns3 run  cttc-3gpp-channel-nums

 Total UDP throughput (bps):3.72507e+08
test@ubuntu:~/Desktop/ns-3-dev$
test@ubuntu:~/Desktop/ns-3-dev$
test@ubuntu:~/Desktop/ns-3-dev$
test@ubuntu:~/Desktop/ns-3-dev$
test@ubuntu:~/Desktop/ns-3-dev$
test@ubuntu:~/Desktop/ns-3-dev$
test@ubuntu:~/Desktop/ns-3-dev$
```

Εικόνα 10: Πειραματικά Αποτελέσματα Παραδείγματος 2

3.3. Παράδειγμα 3

Παρατίθεται ο κώδικας, που χρησιμοποιήθηκε:

```
/**
 *
 * \file cttc-3gpp-channel-simple-fdm.cc
 * \ingroup examples
 * \brief Simple frequency division multiplexing example.
 *
 * This example describes how to setup a simple simulation with the frequency
 * division multiplexing. Simulation example allows configuration of the two
 * bandwidth parts where each is dedicated to different traffic type.
 * The topology is a simple topology that consists of 1 UE and 1 eNB. There
 * is one data bearer active and it will be multiplexed over a one of
 * the two bandwidth parts depending on whether the traffic is configured to
 * be low latency or not. By default the traffic is low latency. So,
 * the example can be run from the command line in the following way:
 *
 * ./ns3 run cttc-3gpp-channel-simple-fdm
 *
 * or to configure flow as not ultra low latency:
 *
 * ./ns3 run 'cttc-3gpp-channel-simple-fdm --isUll=0'
 *
 * Variables that are accessible through the command line (e.g. numerology of
 * BWP 1 can be configured by using --numerologyBwp1=4, so if the user would
 * like to specify this parameter the program can be run in the following way:
 *
 * ./ns3 run "cttc-3gpp-channel-simple-fdm --numerologyBwp1=4"
 *
 *
 *
 * The configured spectrum division is as follows:
 *
 * ------------------------------Band 1---------------------------------
 * ------------------------------CC1-----------------------------------
 * ------------BWP1---------------|--------------BWP2------------------
 *
 */

#include "ns3/antenna-module.h"
#include "ns3/config-store.h"
```

```
#include "ns3/core-module.h"
#include "ns3/eps-bearer-tag.h"
#include "ns3/internet-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/log.h"
#include "ns3/mobility-module.h"
#include "ns3/network-module.h"
#include "ns3/nr-helper.h"
#include "ns3/nr-module.h"
#include "ns3/nr-point-to-point-epc-helper.h"
#include "ns3/three-gpp-spectrum-propagation-loss-model.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("Cttc3gppChannelSimpleFdm");

static int g_rlcTraceCallbackCalled =
    false; //!< Global variable used to check if the callback function for RLC is called and thus to
           //!< determine if the example is run correctly or not
static int g_pdcpTraceCallbackCalled =
    false; //!< Global variable used to check if the callback function for PDCP is called and thus
           //!< to determine if the example is run correctly or not

/**
 * Function creates a single packet and directly calls the function send
 * of a device to send the packet to the destination address.
 * @param device Device that will send the packet to the destination address.
 * @param addr Destination address for a packet.
 * @param packetSize The packet size.
 */
static void
SendPacket(Ptr<NetDevice> device, Address& addr, uint32_t packetSize)
{
    Ptr<Packet> pkt = Create<Packet>(packetSize);
    // Adding empty IPV4 header after adding the IPV6 support for NR module.
    // NrNetDevice::Receive need to peek the header to know the IP protocol.
    // Since, there are no apps install in this test, this packet will be
    // dropped in Ipv4L3Protocol::Receive method upon not finding the route.
    Ipv4Header ipHeader;
    pkt->AddHeader(ipHeader);

    // the dedicated bearer that we activate in the simulation
    // will have bearerId = 2
    EpsBearerTag tag(1, 2);
    pkt->AddPacketTag(tag);
    device->Send(pkt, addr, Ipv4L3Protocol::PROT_NUMBER);
}

/**
 * Function that prints out PDCP delay. This function is designed as a callback
 * for PDCP trace source.
 * @param path The path that matches the trace source
 * @param rnti RNTI of UE
 * @param lcid logical channel id
 * @param bytes PDCP PDU size in bytes
 * @param pdcpDelay PDCP delay
 */
void
RxPdcpPDU(std::string path, uint16_t rnti, uint8_t lcid, uint32_t bytes, uint64_t pdcpDelay)
{
    std::cout << "\n Packet PDCP delay:" << pdcpDelay << "\n";
    g_pdcpTraceCallbackCalled = true;
}
```

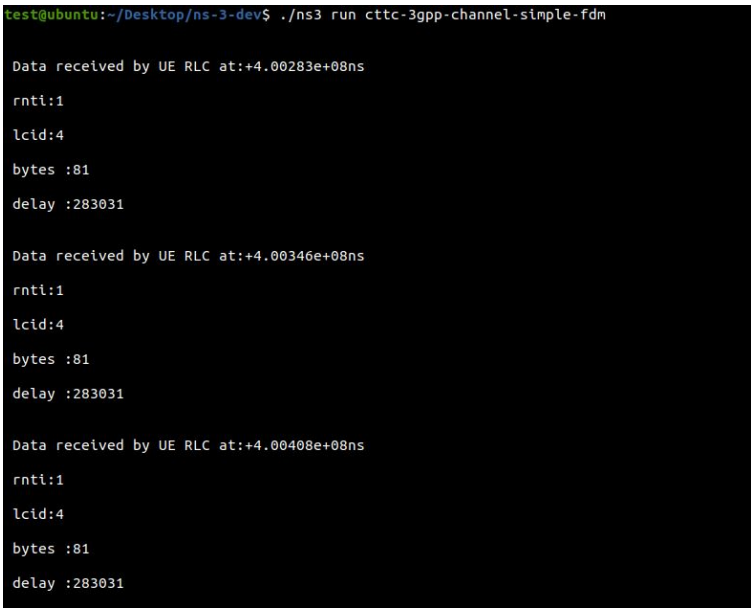
```
/**
 * Function that prints out RLC statistics, such as RNTI, lcId, RLC PDU size,
 * delay. This function is designed as a callback
 * for RLC trace source.
 * @param path The path that matches the trace source
 * @param rnti RNTI of UE
 * @param lcid logical channel id
 * @param bytes RLC PDU size in bytes
 * @param rlcDelay RLC PDU delay
 */
void
RxRlcPDU(std::string path, uint16_t rnti, uint8_t lcid, uint32_t bytes, uint64_t rlcDelay)
{
   std::cout << "\n\n Data received by UE RLC at:" << Simulator::Now() << std::endl;
   std::cout << "\n rnti:" << rnti << std::endl;
   std::cout << "\n lcid:" << (unsigned)lcid << std::endl;
   std::cout << "\n bytes :" << bytes << std::endl;
   std::cout << "\n delay :" << rlcDelay << std::endl;
   g_rlcTraceCallbackCalled = true;
}

/**
 * Function that connects PDCP and RLC traces to the corresponding trace sources.
 */
void
ConnectPdcpRlcTraces()
{
   // after recent changes in the EPC UE node ID has changed to 3
   // dedicated bearer that we have activated has bearer id 2
   Config::Connect("/NodeList/*/DeviceList/*/LteUeRrc/DataRadioBearerMap/*/LtePdcp/RxPDU",
                   MakeCallback(&RxPdcpPDU));
   // after recent changes in the EPC UE node ID has changed to 3
   // dedicated bearer that we have activated has bearer id 2
   Config::Connect("/NodeList/*/DeviceList/*/LteUeRrc/DataRadioBearerMap/*/LteRlc/RxPDU",
                   MakeCallback(&RxRlcPDU));
}

int
main(int argc, char* argv[])
{
   uint16_t gNbNum = 1;
   uint16_t ueNumPergNb = 1;
   uint16_t numerologyBwp1 = 4;
   uint16_t numerologyBwp2 = 2;
   double centralFrequencyBand = 28.1e9;
   double bandwidthBand = 200e6;
   double txPowerPerBwp = 4;
   uint32_t packetSize = 1000;
   bool isUll = true; // Whether the flow is a low latency type of traffic.

   Time sendPacketTime = Seconds(0.4);

   CommandLine cmd(__FILE__);
   cmd.AddValue("gNbNum", "The number of gNbs in multiple-ue topology", gNbNum);
   cmd.AddValue("ueNumPergNb", "The number of UE per gNb in multiple-ue topology", ueNumPergNb);
   cmd.AddValue("numerologyBwp1", "The numerology to be used in bandwidth part 1", numerologyBwp1);
   cmd.AddValue("numerologyBwp2", "The numerology to be used in bandwidth part 2", numerologyBwp2);
   cmd.AddValue("frequency", "The system frequency", centralFrequencyBand);
   cmd.AddValue("bandwidthBand", "The system bandwidth", bandwidthBand);
   cmd.AddValue("packetSize", "packet size in bytes", packetSize);
   cmd.AddValue("isUll", "Enable Uplink", isUll);
   cmd.Parse(argc, argv);

   int64_t randomStream = 1;
```

```
// Create the scenario
GridScenarioHelper gridScenario;
gridScenario.SetRows(1);
gridScenario.SetColumns(gNbNum);
gridScenario.SetHorizontalBsDistance(5.0);
gridScenario.SetBsHeight(10.0);
gridScenario.SetUtHeight(1.5);
// must be set before BS number
gridScenario.SetSectorization(GridScenarioHelper::SINGLE);
gridScenario.SetBsNumber(gNbNum);
gridScenario.SetUtNumber(ueNumPergNb * gNbNum);
gridScenario.SetScenarioHeight(3); // Create a 3x3 scenario where the UE will
gridScenario.SetScenarioLength(3); // be distributed.
randomStream += gridScenario.AssignStreams(randomStream);
gridScenario.CreateScenario();

Config::SetDefault("ns3::EpsBearer::Release", UintegerValue(15));

Ptr<NrPointToPointEpcHelper> epcHelper = CreateObject<NrPointToPointEpcHelper>();
Ptr<IdealBeamformingHelper> idealBeamformingHelper = CreateObject<IdealBeamformingHelper>();
Ptr<NrHelper> nrHelper = CreateObject<NrHelper>();

nrHelper->SetBeamformingHelper(idealBeamformingHelper);
nrHelper->SetEpcHelper(epcHelper);

// Create one operational band containing one CC with 2 bandwidth parts
BandwidthPartInfoPtrVector allBwps;
CcBwpCreator ccBwpCreator;
const uint8_t numCcPerBand = 1; // one CC per Band

// Create the configuration for the CcBwpHelper
CcBwpCreator::SimpleOperationBandConf bandConf(centralFrequencyBand,
                                    bandwidthBand,
                                    numCcPerBand,
                                    BandwidthPartInfo::UMi_StreetCanyon_LoS);
bandConf.m_numBwp = 2; // two BWPs per CC

// By using the configuration created, it is time to make the operation band
OperationBandInfo band = ccBwpCreator.CreateOperationBandContiguousCc(bandConf);

nrHelper->SetPathlossAttribute("ShadowingEnabled", BooleanValue(false));

nrHelper->InitializeOperationBand(&band);
allBwps = CcBwpCreator::GetAllBwps({band});

// Beamforming method
idealBeamformingHelper->SetAttribute("BeamformingMethod",
                            TypeIdValue(DirectPathBeamforming::GetTypeId()));

// Antennas for all the UEs
nrHelper->SetUeAntennaAttribute("NumRows", UintegerValue(2));
nrHelper->SetUeAntennaAttribute("NumColumns", UintegerValue(4));
nrHelper->SetUeAntennaAttribute("AntennaElement",
                          PointerValue(CreateObject<IsotropicAntennaModel>()));

// Antennas for all the gNbs
nrHelper->SetGnbAntennaAttribute("NumRows", UintegerValue(4));
nrHelper->SetGnbAntennaAttribute("NumColumns", UintegerValue(8));
nrHelper->SetGnbAntennaAttribute("AntennaElement",
                           PointerValue(CreateObject<IsotropicAntennaModel>()));

uint32_t bwpIdForLowLat = 0;
uint32_t bwpIdForVoice = 1;
```

```
// gNb routing between Bearer and bandwidh part
nrHelper->SetGnbBwpManagerAlgorithmAttribute("NGBR_LOW_LAT_EMBB",
                                UintegerValue(bwpIdForLowLat));
nrHelper->SetGnbBwpManagerAlgorithmAttribute("GBR_CONV_VOICE", UintegerValue(bwpIdForVoice));

// Ue routing between Bearer and bandwidth part
nrHelper->SetUeBwpManagerAlgorithmAttribute("NGBR_LOW_LAT_EMBB", UintegerValue(bwpIdForLowLat));
nrHelper->SetUeBwpManagerAlgorithmAttribute("GBR_CONV_VOICE", UintegerValue(bwpIdForVoice));

// Install and get the pointers to the NetDevices
NetDeviceContainer enbNetDev =
   nrHelper->InstallGnbDevice(gridScenario.GetBaseStations(), allBwps);
NetDeviceContainer ueNetDev =
   nrHelper->InstallUeDevice(gridScenario.GetUserTerminals(), allBwps);

randomStream += nrHelper->AssignStreams(enbNetDev, randomStream);
randomStream += nrHelper->AssignStreams(ueNetDev, randomStream);

// Set the attribute of the netdevice (enbNetDev.Get (0)) and bandwidth part (0)/(1)
nrHelper->GetGnbPhy(enbNetDev.Get(0), 0)
   ->SetAttribute("Numerology", UintegerValue(numerologyBwp1));
nrHelper->GetGnbPhy(enbNetDev.Get(0), 1)
   ->SetAttribute("Numerology", UintegerValue(numerologyBwp2));
nrHelper->GetGnbPhy(enbNetDev.Get(0), 0)->SetTxPower(txPowerPerBwp);
nrHelper->GetGnbPhy(enbNetDev.Get(0), 1)->SetTxPower(txPowerPerBwp);

for (auto it = enbNetDev.Begin(); it != enbNetDev.End(); ++it)
{
   DynamicCast<NrGnbNetDevice>(*it)->UpdateConfig();
}

for (auto it = ueNetDev.Begin(); it != ueNetDev.End(); ++it)
{
   DynamicCast<NrUeNetDevice>(*it)->UpdateConfig();
}

InternetStackHelper internet;
internet.Install(gridScenario.GetUserTerminals());
Ipv4InterfaceContainer ueIpIface;
ueIpIface = epcHelper->AssignUeIpv4Address(NetDeviceContainer(ueNetDev));

Simulator::Schedule(sendPacketTime,
                &SendPacket,
                enbNetDev.Get(0),
                ueNetDev.Get(0)->GetAddress(),
                packetSize);

// attach UEs to the closest eNB
nrHelper->AttachToClosestEnb(ueNetDev, enbNetDev);

Ptr<EpcTft> tft = Create<EpcTft>();
EpcTft::PacketFilter dlpf;
dlpf.localPortStart = 1234;
dlpf.localPortEnd = 1235;
tft->Add(dlpf);
enum EpsBearer::Qci q;

if (isUll)
{
   q = EpsBearer::NGBR_LOW_LAT_EMBB;
}
else
{
   q = EpsBearer::GBR_CONV_VOICE;
```

```
  }

  EpsBearer bearer(q);
  nrHelper->ActivateDedicatedEpsBearer(ueNetDev, bearer, tft);

  Simulator::Schedule(Seconds(0.2), &ConnectPdcpRlcTraces);

  nrHelper->EnableTraces();

  Simulator::Stop(Seconds(1));
  Simulator::Run();
  Simulator::Destroy();

  if (g_rlcTraceCallbackCalled && g_pdcpTraceCallbackCalled)
  {
    return EXIT_SUCCESS;
  }
  else
  {
    return EXIT_FAILURE;
  }
}
```

Και τα αντίστοιχα πειραματικά αποτελέσματα:



Εικόνα 11: Πειραματικά Αποτελέσματα Παραδείγματος 3 (1)

Εικόνα 12: Πειραματικά Αποτελέσματα Παραδείγματος 3(2)

3.4. Παράδειγμα 4

Παρατίθεται ο κώδικας, που χρησιμοποιήθηκε:

```
/**
 *
 * \file cttc-3gpp-channel-nums-fdm.cc
 * \ingroup examples
 * \brief Frequency division multiplexing example, with TDD and FDD
 *
 * The example is showing how to configure multiple bandwidth parts, in which
 * some of them form a FDD configuration, while others uses TDD. The user
 * can configure the bandwidth and the frequency of these BWPs. Three types
 * of traffic are available: two are DL (video and voice) while one is
 * UL (gaming). Each traffic will be routed to different BWP. Voice will go
 * in the TDD BWP, while video will go in the FDD-DL one, and gaming in the
 * FDD-UL one.
```

```
 *
 * The configured spectrum division is the following:
\verbatim
   |------------BandTdd--------------|--------------BandFdd---------------|
   |------------CC0------------------|--------------CC1-------------------|
   |------------BWP0-----------------|------BWP1-------|-------BWP2-------|
\endverbatim
 * We will configure BWP0 as TDD, BWP1 as FDD-DL, BWP2 as FDD-UL.
 */

#include "ns3/antenna-module.h"
#include "ns3/applications-module.h"
#include "ns3/config-store-module.h"
#include "ns3/config-store.h"
#include "ns3/core-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/internet-apps-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/network-module.h"
#include "ns3/nr-module.h"
#include "ns3/point-to-point-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("3gppChannelNumsFdm");

int
main(int argc, char* argv[])
{
   uint16_t gNbNum = 4;
   uint16_t ueNum = 4;

   uint32_t udpPacketSizeVideo = 100;
   uint32_t udpPacketSizeVoice = 1252;
   uint32_t udpPacketSizeGaming = 500;
   uint32_t lambdaVideo = 50;
   uint32_t lambdaVoice = 100;
   uint32_t lambdaGaming = 250;

   uint32_t simTimeMs = 1400;
   uint32_t udpAppStartTimeMs = 400;

   double centralFrequencyBand1 = 28e9;
   double bandwidthBand1 = 100e6;
   double centralFrequencyBand2 = 28.2e9;
   double bandwidthBand2 = 100e6;
   double totalTxPower = 4;
   std::string simTag = "default";
   std::string outputDir = "./";
   bool enableVideo = true;
   bool enableVoice = true;
   bool enableGaming = true;

   CommandLine cmd(__FILE__);

   cmd.AddValue("packetSizeVideo",
                "packet size in bytes to be used by video traffic",
                udpPacketSizeVideo);
   cmd.AddValue("packetSizeVoice",
                "packet size in bytes to be used by voice traffic",
                udpPacketSizeVoice);
   cmd.AddValue("packetSizeGaming",
                "packet size in bytes to be used by gaming traffic",
```

```
        udpPacketSizeGaming);
cmd.AddValue("lambdaVideo",
        "Number of UDP packets in one second for video traffic",
        lambdaVideo);
cmd.AddValue("lambdaVoice",
        "Number of UDP packets in one second for voice traffic",
        lambdaVoice);
cmd.AddValue("lambdaGaming",
        "Number of UDP packets in one second for gaming traffic",
        lambdaGaming);
cmd.AddValue("enableVideo", "If true, enables video traffic transmission (DL)", enableVideo);
cmd.AddValue("enableVoice", "If true, enables voice traffic transmission (DL)", enableVoice);
cmd.AddValue("enableGaming", "If true, enables gaming traffic transmission (UL)", enableGaming);
cmd.AddValue("simTimeMs", "Simulation time", simTimeMs);
cmd.AddValue("centralFrequencyBand1",
        "The system frequency to be used in band 1",
        centralFrequencyBand1);
cmd.AddValue("bandwidthBand1", "The system bandwidth to be used in band 1", bandwidthBand1);
cmd.AddValue("centralFrequencyBand2",
        "The system frequency to be used in band 2",
        centralFrequencyBand2);
cmd.AddValue("bandwidthBand2", "The system bandwidth to be used in band 2", bandwidthBand2);
cmd.AddValue("totalTxPower",
        "total tx power that will be proportionally assigned to"
        " bands, CCs and bandwidth parts depending on each BWP bandwidth ",
        totalTxPower);
cmd.AddValue("simTag",
        "tag to be appended to output filenames to distinguish simulation campaigns",
        simTag);
cmd.AddValue("outputDir", "directory where to store simulation results", outputDir);

cmd.Parse(argc, argv);

NS_ABORT_IF(centralFrequencyBand1 > 100e9);
NS_ABORT_IF(centralFrequencyBand2 > 100e9);

Config::SetDefault("ns3::LteRlcUm::MaxTxBufferSize", UintegerValue(999999999));

int64_t randomStream = 1;

GridScenarioHelper gridScenario;
gridScenario.SetRows(gNbNum / 2);
gridScenario.SetColumns(gNbNum);
gridScenario.SetHorizontalBsDistance(5.0);
gridScenario.SetBsHeight(10.0);
gridScenario.SetUtHeight(1.5);
// must be set before BS number
gridScenario.SetSectorization(GridScenarioHelper::SINGLE);
gridScenario.SetBsNumber(gNbNum);
gridScenario.SetUtNumber(ueNum);
gridScenario.SetScenarioHeight(3); // Create a 3x3 scenario where the UE will
gridScenario.SetScenarioLength(3); // be distribuited.
randomStream += gridScenario.AssignStreams(randomStream);
gridScenario.CreateScenario();

/*
 * TODO: Add a print, or a plot, that shows the scenario.
 */

Ptr<NrPointToPointEpcHelper> epcHelper = CreateObject<NrPointToPointEpcHelper>();
Ptr<IdealBeamformingHelper> idealBeamformingHelper = CreateObject<IdealBeamformingHelper>();
Ptr<NrHelper> nrHelper = CreateObject<NrHelper>();

// Put the pointers inside nrHelper
```

```
nrHelper->SetBeamformingHelper(idealBeamformingHelper);
nrHelper->SetEpcHelper(epcHelper);

BandwidthPartInfoPtrVector allBwps;
CcBwpCreator ccBwpCreator;
const uint8_t numCcPerBand = 1; // in this example, both bands have a single CC

CcBwpCreator::SimpleOperationBandConf bandConfTdd(centralFrequencyBand1,
                                    bandwidthBand1,
                                    numCcPerBand,
                                    BandwidthPartInfo::UMi_StreetCanyon);
CcBwpCreator::SimpleOperationBandConf bandConfFdd(centralFrequencyBand2,
                                    bandwidthBand2,
                                    numCcPerBand,
                                    BandwidthPartInfo::UMi_StreetCanyon);

bandConfFdd.m_numBwp = 2; // Here, bandFdd will have 2 BWPs

// By using the configuration created, it is time to make the operation bands
OperationBandInfo bandTdd = ccBwpCreator.CreateOperationBandContiguousCc(bandConfTdd);
OperationBandInfo bandFdd = ccBwpCreator.CreateOperationBandContiguousCc(bandConfFdd);

/*
 * The configured spectrum division is:
 * |------------BandTdd--------------|--------------BandFdd--------------|
 * |------------CC0------------------|--------------CC1------------------|
 * |------------BWP0-----------------|------BWP1-------|-------BWP2-------|
 *
 * We will configure BWP0 as TDD, BWP1 as FDD-DL, BWP2 as FDD-UL.
 */

/*
 * Attributes of ThreeGppChannelModel still cannot be set in our way.
 * TODO: Coordinate with Tommaso
 */
Config::SetDefault("ns3::ThreeGppChannelModel::UpdatePeriod", TimeValue(MilliSeconds(0)));
nrHelper->SetChannelConditionModelAttribute("UpdatePeriod", TimeValue(MilliSeconds(0)));
nrHelper->SetPathlossAttribute("ShadowingEnabled", BooleanValue(false));

nrHelper->InitializeOperationBand(&bandTdd);
nrHelper->InitializeOperationBand(&bandFdd);
allBwps = CcBwpCreator::GetAllBwps({bandTdd, bandFdd});

// Beamforming method
idealBeamformingHelper->SetAttribute("BeamformingMethod",
                          TypeIdValue(DirectPathBeamforming::GetTypeId()));

// Core latency
epcHelper->SetAttribute("S1uLinkDelay", TimeValue(MilliSeconds(0)));

// Antennas for all the UEs
nrHelper->SetUeAntennaAttribute("NumRows", UintegerValue(2));
nrHelper->SetUeAntennaAttribute("NumColumns", UintegerValue(4));
nrHelper->SetUeAntennaAttribute("AntennaElement",
                        PointerValue(CreateObject<IsotropicAntennaModel>()));

// Antennas for all the gNbs
nrHelper->SetGnbAntennaAttribute("NumRows", UintegerValue(4));
nrHelper->SetGnbAntennaAttribute("NumColumns", UintegerValue(8));
nrHelper->SetGnbAntennaAttribute("AntennaElement",
                         PointerValue(CreateObject<IsotropicAntennaModel>()));

nrHelper->SetGnbPhyAttribute("TxPower", DoubleValue(4.0));
```

```
uint32_t bwpIdForVoice = 0;
uint32_t bwpIdForVideo = 1;
uint32_t bwpIdForGaming = 2;

nrHelper->SetGnbBwpManagerAlgorithmAttribute("GBR_CONV_VOICE", UintegerValue(bwpIdForVoice));
nrHelper->SetGnbBwpManagerAlgorithmAttribute("GBR_CONV_VIDEO", UintegerValue(bwpIdForVideo));
nrHelper->SetGnbBwpManagerAlgorithmAttribute("GBR_GAMING", UintegerValue(bwpIdForGaming));

nrHelper->SetUeBwpManagerAlgorithmAttribute("GBR_CONV_VOICE", UintegerValue(bwpIdForVoice));
nrHelper->SetUeBwpManagerAlgorithmAttribute("GBR_CONV_VIDEO", UintegerValue(bwpIdForVideo));
nrHelper->SetUeBwpManagerAlgorithmAttribute("GBR_GAMING", UintegerValue(bwpIdForGaming));

NetDeviceContainer enbNetDev =
   nrHelper->InstallGnbDevice(gridScenario.GetBaseStations(), allBwps);
NetDeviceContainer ueNetDev =
   nrHelper->InstallUeDevice(gridScenario.GetUserTerminals(), allBwps);

randomStream += nrHelper->AssignStreams(enbNetDev, randomStream);
randomStream += nrHelper->AssignStreams(ueNetDev, randomStream);

NS_ASSERT(enbNetDev.GetN() == 4);

// ---------- First GNB:

// BWP0, the TDD one
nrHelper->GetGnbPhy(enbNetDev.Get(0), 0)->SetAttribute("Numerology", UintegerValue(0));
nrHelper->GetGnbPhy(enbNetDev.Get(0), 0)
   ->SetAttribute("Pattern", StringValue("F|F|F|F|F|F|F|F|F|F|"));
nrHelper->GetGnbPhy(enbNetDev.Get(0), 0)->SetAttribute("TxPower", DoubleValue(4.0));

// BWP1, FDD-DL
nrHelper->GetGnbPhy(enbNetDev.Get(0), 1)->SetAttribute("Numerology", UintegerValue(0));
nrHelper->GetGnbPhy(enbNetDev.Get(0), 1)
   ->SetAttribute("Pattern", StringValue("DL|DL|DL|DL|DL|DL|DL|DL|DL|DL|"));
nrHelper->GetGnbPhy(enbNetDev.Get(0), 1)->SetAttribute("TxPower", DoubleValue(4.0));

// BWP2, FDD-UL
nrHelper->GetGnbPhy(enbNetDev.Get(0), 2)->SetAttribute("Numerology", UintegerValue(0));
nrHelper->GetGnbPhy(enbNetDev.Get(0), 2)
   ->SetAttribute("Pattern", StringValue("UL|UL|UL|UL|UL|UL|UL|UL|UL|UL|"));
nrHelper->GetGnbPhy(enbNetDev.Get(0), 2)->SetAttribute("TxPower", DoubleValue(0.0));

// Link the two FDD BWP:
nrHelper->GetBwpManagerGnb(enbNetDev.Get(0))->SetOutputLink(2, 1);

// ---------- Second GNB:

// BWP0, the TDD one
nrHelper->GetGnbPhy(enbNetDev.Get(1), 0)->SetAttribute("Numerology", UintegerValue(1));
nrHelper->GetGnbPhy(enbNetDev.Get(1), 0)
   ->SetAttribute("Pattern", StringValue("F|F|F|F|F|F|F|F|F|F|"));
nrHelper->GetGnbPhy(enbNetDev.Get(1), 0)->SetAttribute("TxPower", DoubleValue(4.0));

// BWP1, FDD-DL
nrHelper->GetGnbPhy(enbNetDev.Get(1), 1)->SetAttribute("Numerology", UintegerValue(1));
nrHelper->GetGnbPhy(enbNetDev.Get(1), 1)
   ->SetAttribute("Pattern", StringValue("DL|DL|DL|DL|DL|DL|DL|DL|DL|DL|"));
nrHelper->GetGnbPhy(enbNetDev.Get(1), 1)->SetAttribute("TxPower", DoubleValue(4.0));

// BWP2, FDD-UL
nrHelper->GetGnbPhy(enbNetDev.Get(1), 2)->SetAttribute("Numerology", UintegerValue(1));
nrHelper->GetGnbPhy(enbNetDev.Get(1), 2)
   ->SetAttribute("Pattern", StringValue("UL|UL|UL|UL|UL|UL|UL|UL|UL|UL|"));
nrHelper->GetGnbPhy(enbNetDev.Get(1), 2)->SetAttribute("TxPower", DoubleValue(0.0));
```

```
// Link the two FDD BWP:
nrHelper->GetBwpManagerGnb(enbNetDev.Get(1))->SetOutputLink(2, 1);

// ---------- Third GNB:

// BWP0, the TDD one
nrHelper->GetGnbPhy(enbNetDev.Get(2), 0)->SetAttribute("Numerology", UintegerValue(2));
nrHelper->GetGnbPhy(enbNetDev.Get(2), 0)
   ->SetAttribute("Pattern", StringValue("F|F|F|F|F|F|F|F|F|F|"));
nrHelper->GetGnbPhy(enbNetDev.Get(2), 0)->SetAttribute("TxPower", DoubleValue(4.0));

// BWP1, FDD-DL
nrHelper->GetGnbPhy(enbNetDev.Get(2), 1)->SetAttribute("Numerology", UintegerValue(2));
nrHelper->GetGnbPhy(enbNetDev.Get(2), 1)
   ->SetAttribute("Pattern", StringValue("DL|DL|DL|DL|DL|DL|DL|DL|DL|DL|"));
nrHelper->GetGnbPhy(enbNetDev.Get(2), 1)->SetAttribute("TxPower", DoubleValue(4.0));

// BWP2, FDD-UL
nrHelper->GetGnbPhy(enbNetDev.Get(2), 2)->SetAttribute("Numerology", UintegerValue(2));
nrHelper->GetGnbPhy(enbNetDev.Get(2), 2)
   ->SetAttribute("Pattern", StringValue("UL|UL|UL|UL|UL|UL|UL|UL|UL|UL|"));
nrHelper->GetGnbPhy(enbNetDev.Get(2), 2)->SetAttribute("TxPower", DoubleValue(0.0));

// Link the two FDD BWP:
nrHelper->GetBwpManagerGnb(enbNetDev.Get(2))->SetOutputLink(2, 1);

// ---------- Fourth GNB:

// BWP0, the TDD one
nrHelper->GetGnbPhy(enbNetDev.Get(3), 0)->SetAttribute("Numerology", UintegerValue(3));
nrHelper->GetGnbPhy(enbNetDev.Get(3), 0)
   ->SetAttribute("Pattern", StringValue("F|F|F|F|F|F|F|F|F|F|"));
nrHelper->GetGnbPhy(enbNetDev.Get(3), 0)->SetAttribute("TxPower", DoubleValue(4.0));

// BWP1, FDD-DL
nrHelper->GetGnbPhy(enbNetDev.Get(3), 1)->SetAttribute("Numerology", UintegerValue(3));
nrHelper->GetGnbPhy(enbNetDev.Get(3), 1)
   ->SetAttribute("Pattern", StringValue("DL|DL|DL|DL|DL|DL|DL|DL|DL|DL|"));
nrHelper->GetGnbPhy(enbNetDev.Get(3), 1)->SetAttribute("TxPower", DoubleValue(4.0));

// BWP2, FDD-UL
nrHelper->GetGnbPhy(enbNetDev.Get(3), 2)->SetAttribute("Numerology", UintegerValue(3));
nrHelper->GetGnbPhy(enbNetDev.Get(3), 2)
   ->SetAttribute("Pattern", StringValue("UL|UL|UL|UL|UL|UL|UL|UL|UL|UL|"));
nrHelper->GetGnbPhy(enbNetDev.Get(3), 2)->SetAttribute("TxPower", DoubleValue(0.0));

// Link the two FDD BWP:
nrHelper->GetBwpManagerGnb(enbNetDev.Get(3))->SetOutputLink(2, 1);

// Set the UE routing:

for (uint32_t i = 0; i < ueNetDev.GetN(); i++)
{
   nrHelper->GetBwpManagerUe(ueNetDev.Get(i))->SetOutputLink(1, 2);
}

// When all the configuration is done, explicitly call UpdateConfig ()

for (auto it = enbNetDev.Begin(); it != enbNetDev.End(); ++it)
{
   DynamicCast<NrGnbNetDevice>(*it)->UpdateConfig();
}
```

```
for (auto it = ueNetDev.Begin(); it != ueNetDev.End(); ++it)
{
  DynamicCast<NrUeNetDevice>(*it)->UpdateConfig();
}

// From here, it is standard NS3. In the future, we will create helpers
// for this part as well.

// create the internet and install the IP stack on the UEs
// get SGW/PGW and create a single RemoteHost
Ptr<Node> pgw = epcHelper->GetPgwNode();
NodeContainer remoteHostContainer;
remoteHostContainer.Create(1);
Ptr<Node> remoteHost = remoteHostContainer.Get(0);
InternetStackHelper internet;
internet.Install(remoteHostContainer);

// connect a remoteHost to pgw. Setup routing too
PointToPointHelper p2ph;
p2ph.SetDeviceAttribute("DataRate", DataRateValue(DataRate("100Gb/s")));
p2ph.SetDeviceAttribute("Mtu", UintegerValue(2500));
p2ph.SetChannelAttribute("Delay", TimeValue(Seconds(0.000)));
NetDeviceContainer internetDevices = p2ph.Install(pgw, remoteHost);
Ipv4AddressHelper ipv4h;
Ipv4StaticRoutingHelper ipv4RoutingHelper;
ipv4h.SetBase("1.0.0.0", "255.0.0.0");
Ipv4InterfaceContainer internetIpIfaces = ipv4h.Assign(internetDevices);
Ptr<Ipv4StaticRouting> remoteHostStaticRouting =
  ipv4RoutingHelper.GetStaticRouting(remoteHost->GetObject<Ipv4>());
remoteHostStaticRouting->AddNetworkRouteTo(Ipv4Address("7.0.0.0"), Ipv4Mask("255.0.0.0"), 1);
internet.Install(gridScenario.GetUserTerminals());

Ipv4InterfaceContainer ueIpIface = epcHelper->AssignUeIpv4Address(NetDeviceContainer(ueNetDev));

// Set the default gateway for the UEs
for (uint32_t j = 0; j < gridScenario.GetUserTerminals().GetN(); ++j)
{
  Ptr<Ipv4StaticRouting> ueStaticRouting = ipv4RoutingHelper.GetStaticRouting(
    gridScenario.GetUserTerminals().Get(j)->GetObject<Ipv4>());
  ueStaticRouting->SetDefaultRoute(epcHelper->GetUeDefaultGatewayAddress(), 1);
}

// Fix the attachment of the UEs: UE_i attached to GNB_i
for (uint32_t i = 0; i < ueNetDev.GetN(); ++i)
{
  auto enbDev = DynamicCast<NrGnbNetDevice>(enbNetDev.Get(i));
  auto ueDev = DynamicCast<NrUeNetDevice>(ueNetDev.Get(i));
  NS_ASSERT(enbDev != nullptr);
  NS_ASSERT(ueDev != nullptr);
  nrHelper->AttachToEnb(ueDev, enbDev);
}

/*
 * Traffic part. Install two kind of traffic: low-latency and voice, each
 * identified by a particular source port.
 */
uint16_t dlPortVideo = 1234;
uint16_t dlPortVoice = 1235;
uint16_t ulPortGaming = 1236;

ApplicationContainer serverApps;

// The sink will always listen to the specified ports
UdpServerHelper dlPacketSinkVideo(dlPortVideo);
```

```
UdpServerHelper dlPacketSinkVoice(dlPortVoice);
UdpServerHelper ulPacketSinkVoice(ulPortGaming);

// The server, that is the application which is listening, is installed in the UE
// for the DL traffic, and in the remote host for the UL traffic
serverApps.Add(dlPacketSinkVideo.Install(gridScenario.GetUserTerminals()));
serverApps.Add(dlPacketSinkVoice.Install(gridScenario.GetUserTerminals()));
serverApps.Add(ulPacketSinkVoice.Install(remoteHost));

/*
 * Configure attributes for the different generators, using user-provided
 * parameters for generating a CBR traffic
 *
 * Low-Latency configuration and object creation:
 */
UdpClientHelper dlClientVideo;
dlClientVideo.SetAttribute("RemotePort", UintegerValue(dlPortVideo));
dlClientVideo.SetAttribute("MaxPackets", UintegerValue(0xFFFFFFFF));
dlClientVideo.SetAttribute("PacketSize", UintegerValue(udpPacketSizeVideo));
dlClientVideo.SetAttribute("Interval", TimeValue(Seconds(1.0 / lambdaVideo)));

// The bearer that will carry low latency traffic
EpsBearer videoBearer(EpsBearer::GBR_CONV_VIDEO);

// The filter for the low-latency traffic
Ptr<EpcTft> videoTft = Create<EpcTft>();
EpcTft::PacketFilter dlpfVideo;
dlpfVideo.localPortStart = dlPortVideo;
dlpfVideo.localPortEnd = dlPortVideo;
videoTft->Add(dlpfVideo);

// Voice configuration and object creation:
UdpClientHelper dlClientVoice;
dlClientVoice.SetAttribute("RemotePort", UintegerValue(dlPortVoice));
dlClientVoice.SetAttribute("MaxPackets", UintegerValue(0xFFFFFFFF));
dlClientVoice.SetAttribute("PacketSize", UintegerValue(udpPacketSizeVoice));
dlClientVoice.SetAttribute("Interval", TimeValue(Seconds(1.0 / lambdaVoice)));

// The bearer that will carry voice traffic
EpsBearer voiceBearer(EpsBearer::GBR_CONV_VOICE);

// The filter for the voice traffic
Ptr<EpcTft> voiceTft = Create<EpcTft>();
EpcTft::PacketFilter dlpfVoice;
dlpfVoice.localPortStart = dlPortVoice;
dlpfVoice.localPortEnd = dlPortVoice;
voiceTft->Add(dlpfVoice);

// Gaming configuration and object creation:
UdpClientHelper ulClientGaming;
ulClientGaming.SetAttribute("RemotePort", UintegerValue(ulPortGaming));
ulClientGaming.SetAttribute("MaxPackets", UintegerValue(0xFFFFFFFF));
ulClientGaming.SetAttribute("PacketSize", UintegerValue(udpPacketSizeGaming));
ulClientGaming.SetAttribute("Interval", TimeValue(Seconds(1.0 / lambdaGaming)));

// The bearer that will carry gaming traffic
EpsBearer gamingBearer(EpsBearer::GBR_GAMING);

// The filter for the gaming traffic
Ptr<EpcTft> gamingTft = Create<EpcTft>();
EpcTft::PacketFilter ulpfGaming;
ulpfGaming.remotePortStart = ulPortGaming;
ulpfGaming.remotePortEnd = ulPortGaming;
ulpfGaming.direction = EpcTft::UPLINK;
```

```
  gamingTft->Add(ulpfGaming);

  /*
   * Let's install the applications!
   */
  ApplicationContainer clientApps;

  for (uint32_t i = 0; i < gridScenario.GetUserTerminals().GetN(); ++i)
  {
     Ptr<Node> ue = gridScenario.GetUserTerminals().Get(i);
     Ptr<NetDevice> ueDevice = ueNetDev.Get(i);
     Address ueAddress = ueIpIface.GetAddress(i);

     // The client, who is transmitting, is installed in the remote host,
     // with destination address set to the address of the UE
     if (enableVoice)
     {
        dlClientVoice.SetAttribute("RemoteAddress", AddressValue(ueAddress));
        clientApps.Add(dlClientVoice.Install(remoteHost));

        nrHelper->ActivateDedicatedEpsBearer(ueDevice, voiceBearer, voiceTft);
     }

     if (enableVideo)
     {
        dlClientVideo.SetAttribute("RemoteAddress", AddressValue(ueAddress));
        clientApps.Add(dlClientVideo.Install(remoteHost));

        nrHelper->ActivateDedicatedEpsBearer(ueDevice, videoBearer, videoTft);
     }

     // For the uplink, the installation happens in the UE, and the remote address
     // is the one of the remote host

     if (enableGaming)
     {
        ulClientGaming.SetAttribute("RemoteAddress",
                                    AddressValue(internetIpIfaces.GetAddress(1)));
        clientApps.Add(ulClientGaming.Install(ue));

        nrHelper->ActivateDedicatedEpsBearer(ueDevice, gamingBearer, gamingTft);
     }
  }

  // start UDP server and client apps
  serverApps.Start(MilliSeconds(udpAppStartTimeMs));
  clientApps.Start(MilliSeconds(udpAppStartTimeMs));
  serverApps.Stop(MilliSeconds(simTimeMs));
  clientApps.Stop(MilliSeconds(simTimeMs));

  // enable the traces provided by the nr module
  // nrHelper->EnableTraces();

  FlowMonitorHelper flowmonHelper;
  NodeContainer endpointNodes;
  endpointNodes.Add(remoteHost);
  endpointNodes.Add(gridScenario.GetUserTerminals());

  Ptr<ns3::FlowMonitor> monitor = flowmonHelper.Install(endpointNodes);
  monitor->SetAttribute("DelayBinWidth", DoubleValue(0.001));
  monitor->SetAttribute("JitterBinWidth", DoubleValue(0.001));
  monitor->SetAttribute("PacketSizeBinWidth", DoubleValue(20));

  Simulator::Stop(MilliSeconds(simTimeMs));
```

```
Simulator::Run();

/*
 * To check what was installed in the memory, i.e., BWPs of eNb Device, and its configuration.
 * Example is: Node 1 -> Device 0 -> BandwidthPartMap -> {0,1} BWPs -> NrGnbPhy -> Numerology,
GtkConfigStore config;
config.ConfigureAttributes ();
*/

// Print per-flow statistics
monitor->CheckForLostPackets();
Ptr<Ipv4FlowClassifier> classifier =
   DynamicCast<Ipv4FlowClassifier>(flowmonHelper.GetClassifier());
FlowMonitor::FlowStatsContainer stats = monitor->GetFlowStats();

double averageFlowThroughput = 0.0;
double averageFlowDelay = 0.0;

std::ofstream outFile;
std::string filename = outputDir + "/" + simTag;
outFile.open(filename.c_str(), std::ofstream::out | std::ofstream::trunc);
if (!outFile.is_open())
{
   std::cerr << "Can't open file " << filename << std::endl;
   return 1;
}

outFile.setf(std::ios_base::fixed);

for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin();
    i != stats.end();
    ++i)
{
   Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(i->first);
   std::stringstream protoStream;
   protoStream << (uint16_t)t.protocol;
   if (t.protocol == 6)
   {
      protoStream.str("TCP");
   }
   if (t.protocol == 17)
   {
      protoStream.str("UDP");
   }
   outFile << "Flow " << i->first << " (" << t.sourceAddress << ":" << t.sourcePort << " -> "
         << t.destinationAddress << ":" << t.destinationPort << ") proto "
         << protoStream.str() << "\n";
   outFile << "  Tx Packets: " << i->second.txPackets << "\n";
   outFile << "  Tx Bytes:   " << i->second.txBytes << "\n";
   outFile << "  TxOffered:  "
         << i->second.txBytes * 8.0 / ((simTimeMs - udpAppStartTimeMs) / 1000.0) / 1000.0 /
              1000.0
         << " Mbps\n";
   outFile << "  Rx Bytes:   " << i->second.rxBytes << "\n";
   if (i->second.rxPackets > 0)
   {
      // Measure the duration of the flow from receiver's perspective
      // double rxDuration = i->second.timeLastRxPacket.GetSeconds () -
      // i->second.timeFirstTxPacket.GetSeconds ();
      double rxDuration = (simTimeMs - udpAppStartTimeMs) / 1000.0;

      averageFlowThroughput += i->second.rxBytes * 8.0 / rxDuration / 1000 / 1000;
      averageFlowDelay += 1000 * i->second.delaySum.GetSeconds() / i->second.rxPackets;
```

```
            outFile << "  Throughput: " << i->second.rxBytes * 8.0 / rxDuration / 1000 / 1000
                    << " Mbps\n";
            outFile << "  Mean delay:  "
                    << 1000 * i->second.delaySum.GetSeconds() / i->second.rxPackets << " ms\n";
            // outFile << "  Mean upt:  " << i->second.uptSum / i->second.rxPackets / 1000/1000 << "
            // Mbps \n";
            outFile << "  Mean jitter:  "
                    << 1000 * i->second.jitterSum.GetSeconds() / i->second.rxPackets << " ms\n";
        }
        else
        {
            outFile << "  Throughput:  0 Mbps\n";
            outFile << "  Mean delay:  0 ms\n";
            outFile << "  Mean jitter: 0 ms\n";
        }
        outFile << "  Rx Packets: " << i->second.rxPackets << "\n";
    }

    double meanFlowThroughput = averageFlowThroughput / stats.size();
    double meanFlowDelay = averageFlowDelay / stats.size();
    double throughputTolerance = meanFlowThroughput * 0.001;

    outFile << "\n\n  Mean flow throughput: " << meanFlowThroughput << "\n";
    outFile << "  Mean flow delay: " << meanFlowDelay << "\n";

    outFile.close();

    std::ifstream f(filename.c_str());

    if (f.is_open())
    {
        std::cout << f.rdbuf();
    }

    Simulator::Destroy();

    // called from examples-to-run.py with all default parameters
    if (argc == 0 && (meanFlowThroughput < 0.709696 - throughputTolerance ||
                      meanFlowThroughput > 0.709696 + throughputTolerance))
    {
        return EXIT_FAILURE;
    }
    else
    {
        return EXIT_SUCCESS;
    }
}
```

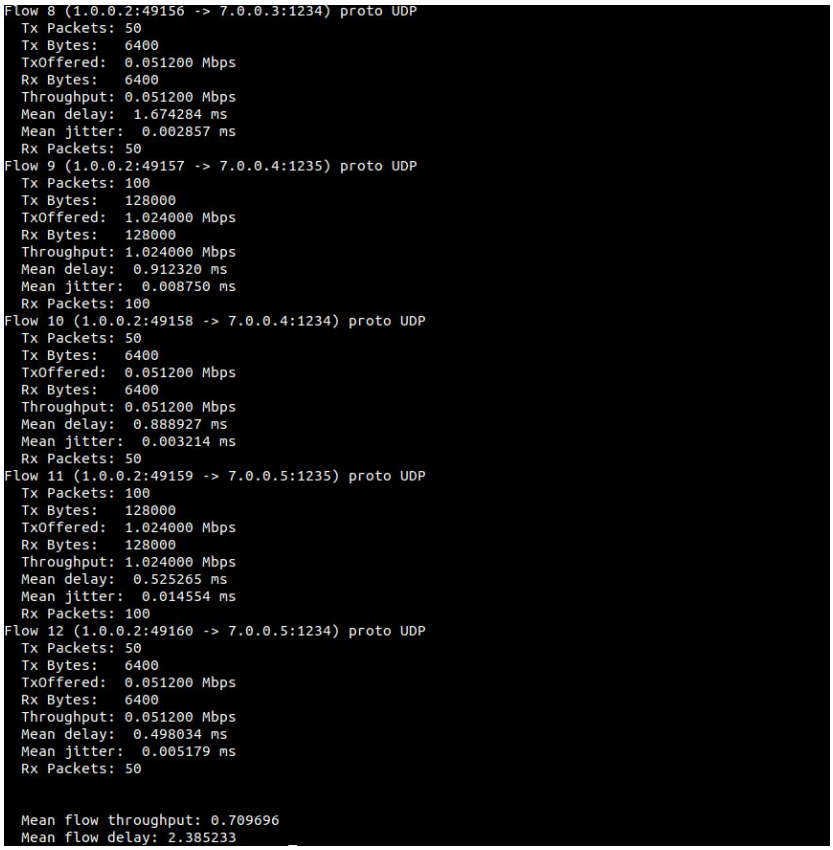
Και τα αντίστοιχα πειραματικά αποτελέσματα:

```
Flow 1 (7.0.0.2:49153 -> 1.0.0.2:1236) proto UDP
  Tx Packets: 250
  Tx Bytes:   132000
  TxOffered:  1.056000 Mbps
  Rx Bytes:   131472
  Throughput: 1.051776 Mbps
  Mean delay:  6.214487 ms
  Mean jitter:  3.967415 ms
  Rx Packets: 249
Flow 2 (7.0.0.3:49153 -> 1.0.0.2:1236) proto UDP
  Tx Packets: 250
  Tx Bytes:   132000
  TxOffered:  1.056000 Mbps
  Rx Bytes:   131472
  Throughput: 1.051776 Mbps
  Mean delay:  4.083309 ms
  Mean jitter:  0.026101 ms
  Rx Packets: 249
Flow 3 (7.0.0.4:49153 -> 1.0.0.2:1236) proto UDP
  Tx Packets: 250
  Tx Bytes:   132000
  TxOffered:  1.056000 Mbps
  Rx Bytes:   132000
  Throughput: 1.056000 Mbps
  Mean delay:  3.780094 ms
  Mean jitter:  0.607000 ms
  Rx Packets: 250
Flow 4 (7.0.0.5:49153 -> 1.0.0.2:1236) proto UDP
  Tx Packets: 250
  Tx Bytes:   132000
  TxOffered:  1.056000 Mbps
  Rx Bytes:   132000
  Throughput: 1.056000 Mbps
  Mean delay:  1.867517 ms
  Mean jitter:  0.050500 ms
  Rx Packets: 250
```

Εικόνα 13: Πειραματικά Αποτελέσματα Παραδείγματος 4(1)

```
Flow 5 (1.0.0.2:49153 -> 7.0.0.2:1235) proto UDP
  Tx Packets: 100
  Tx Bytes:   128000
  TxOffered:  1.024000 Mbps
  Rx Bytes:   128000
  Throughput: 1.024000 Mbps
  Mean delay:  3.252855 ms
  Mean jitter:  0.010000 ms
  Rx Packets: 100
Flow 6 (1.0.0.2:49154 -> 7.0.0.2:1234) proto UDP
  Tx Packets: 50
  Tx Bytes:   6400
  TxOffered:  0.051200 Mbps
  Rx Bytes:   6400
  Throughput: 0.051200 Mbps
  Mean delay:  3.245712 ms
  Mean jitter:  0.002857 ms
  Rx Packets: 50
Flow 7 (1.0.0.2:49155 -> 7.0.0.3:1235) proto UDP
  Tx Packets: 100
  Tx Bytes:   128000
  TxOffered:  1.024000 Mbps
  Rx Bytes:   128000
  Throughput: 1.024000 Mbps
  Mean delay:  1.679998 ms
  Mean jitter:  0.008571 ms
  Rx Packets: 100
```

Εικόνα 14: Πειραματικά Αποτελέσματα Παραδείγματος 4(2)



Εικόνα 15: Πειραματικά Αποτελέσματα Παραδείγματος 4(3)

3.5. Παράδειγμα 5

Παρατίθεται ο κώδικας, που χρησιμοποιήθηκε:

```
/**
 * \file cttc-error-model.cc
 * \ingroup examples
 * \brief Error model example with fixed MCS: 1 gNB and 1 UE, multiple packets with varying fading
 * conditions.
 *
 * This example allows the user to test the end-to-end performance with the new
 * NR PHY abstraction model for error modeling by using a fixed MCS. It allows the user to set the
 * MCS, the gNB-UE distance, the MCS table, the error model type, and the HARQ method.
 *
 * The NR error model can be set as "--errorModel=ns3::NrEesmCcT1", for HARQ-CC and MCS Table1,
 * while "--errorModel=ns3::NrLteMiErrorModel" configures the LTE error model.
 * For NR, you can choose between different types of error model, which use
 * different tables and different methods to process the HARQ history, e.g.,
 * "--errorModel=ns3::NrEesmIrT1", for HARQ-IR and MCS Table2.
 * You can fix also the MCS index to use with "--mcs=7" (7 in this case), which refers
```

```
 * to the configured MCS table.
 *
 * The scenario consists of a single gNB and a single UE, placed at positions (0.0, 0.0, 10), and
 * (0.0, ueY, 1.5), respectively. ueY can be configured by the user, e.g. "ueY=20", and defaults
 * to 30 m.
 *
 * By default, the program uses the 3GPP channel model, Urban Micro scenario, without shadowing and
 * with probabilistic line of sight / non-line of sight option. The program runs for 50 seconds and
 * one packet is transmitted every 200 ms from gNB to UE (donwlink direction). The packet size can
 * be configured by using the following parameter: "--packetSize=1000". The channel update period is
 * 150 ms, so that every packet encounters a different fading condition.
 *
 * This simulation prints the output to the terminal. The output statistics are
 * averaged among all the transmitted packets.
 *
 * To run the simulation with the default configuration one shall run the
 * following in the command line:
 *
 * ./ns3 run cttc-error-model
 *
 */

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("CttcErrorModelExample");

static Ptr<ListPositionAllocator>
GetGnbPositions(double gNbHeight = 10.0)
{
   Ptr<ListPositionAllocator> pos = CreateObject<ListPositionAllocator>();
   pos->Add(Vector(0.0, 0.0, gNbHeight));

   return pos;
}

static Ptr<ListPositionAllocator>
GetUePositions(double ueY, double ueHeight = 1.5)
{
   Ptr<ListPositionAllocator> pos = CreateObject<ListPositionAllocator>();
   pos->Add(Vector(0.0, ueY, ueHeight));

   return pos;
}

static std::vector<uint64_t> packetsTime;

static void
PrintRxPkt([[maybe_unused]] std::string context, Ptr<const Packet> pkt)
{
   // ASSUMING ONE UE
   SeqTsHeader seqTs;
   pkt->PeekHeader(seqTs);
   packetsTime.push_back((Simulator::Now() - seqTs.GetTs()).GetMicroSeconds());
}

int
main(int argc, char* argv[])
{
   uint32_t mcs = 13;
   const uint8_t gNbNum = 1;
   const uint8_t ueNum = 1;
   double totalTxPower = 4;
   uint16_t numerologyBwp = 4;
   double centralFrequencyBand = 28e9;
```

```
double bandwidthBand = 100e6;
double ueY = 30.0;

double simTime = 10.0; // 50 seconds: to take statistics
uint32_t pktSize = 500;
Time udpAppStartTime = MilliSeconds(1000);
Time packetInterval = MilliSeconds(200);
Time updateChannelInterval = MilliSeconds(150);
bool isUl = false;

std::string errorModel = "ns3::NrEesmCcT1";

CommandLine cmd(__FILE__);

cmd.AddValue("simTime", "Simulation time", simTime);
cmd.AddValue("mcs", "The MCS that will be used in this example", mcs);
cmd.AddValue("errorModelType",
        "Error model type: ns3::NrEesmCcT1, ns3::NrEesmCcT2, ns3::NrEesmIrT1, "
        "ns3::NrEesmIrT2, ns3::NrLteMiErrorModel",
        errorModel);
cmd.AddValue("ueY", "Y position of any UE", ueY);
cmd.AddValue("pktSize", "Packet Size", pktSize);
cmd.AddValue("isUl", "Is this an UL transmission?", isUl);

cmd.Parse(argc, argv);

uint32_t packets = (simTime - udpAppStartTime.GetSeconds()) / packetInterval.GetSeconds();
NS_ABORT_IF(packets == 0);

/*
 * Default values for the simulation. We are progressively removing all
 * the instances of SetDefault, but we need it for legacy code (LTE)
 */
Config::SetDefault("ns3::LteRlcUm::MaxTxBufferSize", UintegerValue(999999999));

Config::SetDefault("ns3::NrAmc::ErrorModelType", TypeIdValue(TypeId::LookupByName(errorModel)));
Config::SetDefault("ns3::NrAmc::AmcModel",
            EnumValue(NrAmc::ShannonModel)); // NOT USED in this example. MCS is fixed.

// create base stations and mobile terminals
NodeContainer gNbNodes;
NodeContainer ueNodes;
MobilityHelper mobility;

double gNbHeight = 10.0;
double ueHeight = 1.5;

gNbNodes.Create(gNbNum);
ueNodes.Create(ueNum);

Ptr<ListPositionAllocator> gNbPositionAlloc = GetGnbPositions(gNbHeight);
Ptr<ListPositionAllocator> uePositionAlloc = GetUePositions(ueY, ueHeight);

mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.SetPositionAllocator(gNbPositionAlloc);
mobility.Install(gNbNodes);

mobility.SetPositionAllocator(uePositionAlloc);
mobility.Install(ueNodes);

/*
 * Setup the NR module. We create the various helpers needed for the
 * NR simulation:
 * - EpcHelper, which will setup the core network
```

```
 * - IdealBeamformingHelper, which takes care of the beamforming part
 * - NrHelper, which takes care of creating and connecting the various
 * part of the NR stack
 */
Ptr<NrPointToPointEpcHelper> epcHelper = CreateObject<NrPointToPointEpcHelper>();
Ptr<IdealBeamformingHelper> idealBeamformingHelper = CreateObject<IdealBeamformingHelper>();
Ptr<NrHelper> nrHelper = CreateObject<NrHelper>();

// Put the pointers inside nrHelper
nrHelper->SetBeamformingHelper(idealBeamformingHelper);
nrHelper->SetEpcHelper(epcHelper);

/*
 * Spectrum division. We create one operational band, with one CC, and the CC with a single
 * bandwidth part.
 */
BandwidthPartInfoPtrVector allBwps;
CcBwpCreator ccBwpCreator;
const uint8_t numCcPerBand = 1;

CcBwpCreator::SimpleOperationBandConf bandConf(centralFrequencyBand,
                                       bandwidthBand,
                                       numCcPerBand,
                                       BandwidthPartInfo::UMi_StreetCanyon);
OperationBandInfo band = ccBwpCreator.CreateOperationBandContiguousCc(bandConf);

/*
 * Attributes of ThreeGppChannelModel still cannot be set in our way.
 * TODO: Coordinate with Tommaso
 */
Config::SetDefault("ns3::ThreeGppChannelModel::UpdatePeriod", TimeValue(updateChannelInterval));
nrHelper->SetChannelConditionModelAttribute("UpdatePeriod", TimeValue(MilliSeconds(0)));
nrHelper->SetPathlossAttribute("ShadowingEnabled", BooleanValue(false));

/*
 * Initialize channel and pathloss, plus other things inside band.
 */
nrHelper->InitializeOperationBand(&band);
allBwps = CcBwpCreator::GetAllBwps({band});

Packet::EnableChecking();
Packet::EnablePrinting();

/*
 *  Case (i): Attributes valid for all the nodes
 */
// Beamforming method
idealBeamformingHelper->SetAttribute("BeamformingMethod",
                             TypeIdValue(DirectPathBeamforming::GetTypeId()));

// Core latency
epcHelper->SetAttribute("S1uLinkDelay", TimeValue(MilliSeconds(0)));

// Antennas for all the UEs
nrHelper->SetUeAntennaAttribute("NumRows", UintegerValue(2));
nrHelper->SetUeAntennaAttribute("NumColumns", UintegerValue(4));
nrHelper->SetUeAntennaAttribute("AntennaElement",
                          PointerValue(CreateObject<IsotropicAntennaModel>()));

// Antennas for all the gNbs
nrHelper->SetGnbAntennaAttribute("NumRows", UintegerValue(4));
nrHelper->SetGnbAntennaAttribute("NumColumns", UintegerValue(8));
nrHelper->SetGnbAntennaAttribute("AntennaElement",
                           PointerValue(CreateObject<IsotropicAntennaModel>()));
```

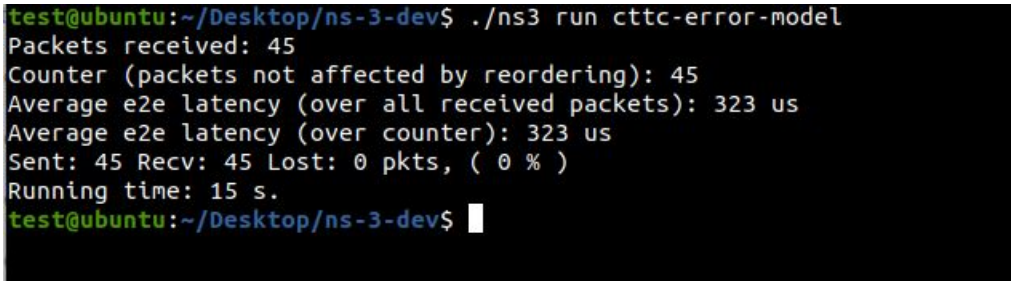
```
// Scheduler
nrHelper->SetSchedulerAttribute("FixedMcsDl", BooleanValue(true));
nrHelper->SetSchedulerAttribute("FixedMcsUl", BooleanValue(true));
nrHelper->SetSchedulerAttribute("StartingMcsDl", UintegerValue(mcs));
nrHelper->SetSchedulerAttribute("StartingMcsUl", UintegerValue(mcs));

// Error Model: UE and GNB with same spectrum error model.
nrHelper->SetUlErrorModel(errorModel);
nrHelper->SetDlErrorModel(errorModel);

// Both DL and UL AMC will have the same model behind.
// Note: NOT USED in this example. MCS is fixed.
nrHelper->SetGnbDlAmcAttribute("AmcModel", EnumValue(NrAmc::ShannonModel));
nrHelper->SetGnbUlAmcAttribute("AmcModel", EnumValue(NrAmc::ShannonModel));

nrHelper->SetUePhyAttribute("TxPower", DoubleValue(totalTxPower));

uint32_t bwpId = 0;

// gNb routing between Bearer and bandwidh part
nrHelper->SetGnbBwpManagerAlgorithmAttribute("NGBR_LOW_LAT_EMBB", UintegerValue(bwpId));

// Ue routing between Bearer and bandwidth part
nrHelper->SetUeBwpManagerAlgorithmAttribute("NGBR_LOW_LAT_EMBB", UintegerValue(bwpId));

NetDeviceContainer gnbNetDev = nrHelper->InstallGnbDevice(gNbNodes, allBwps);
NetDeviceContainer ueNetDev = nrHelper->InstallUeDevice(ueNodes, allBwps);

int64_t randomStream = 1;
randomStream += nrHelper->AssignStreams(gnbNetDev, randomStream);
randomStream += nrHelper->AssignStreams(ueNetDev, randomStream);

/*
 * Case (iii): Go node for node and change the attributes we have to setup
 * per-node.
 */

// Get the first netdevice (enbNetDev.Get (0)) and the first bandwidth part (0)
// and set the attribute.
nrHelper->GetGnbPhy(gnbNetDev.Get(0), 0)
   ->SetAttribute("Numerology", UintegerValue(numerologyBwp));
nrHelper->GetGnbPhy(gnbNetDev.Get(0), 0)->SetAttribute("TxPower", DoubleValue(totalTxPower));

// When all the configuration is done, explicitly call UpdateConfig ()

for (auto it = gnbNetDev.Begin(); it != gnbNetDev.End(); ++it)
{
   DynamicCast<NrGnbNetDevice>(*it)->UpdateConfig();
}

for (auto it = ueNetDev.Begin(); it != ueNetDev.End(); ++it)
{
   DynamicCast<NrUeNetDevice>(*it)->UpdateConfig();
}

// create the internet and install the IP stack on the UEs
// get SGW/PGW and create a single RemoteHost
Ptr<Node> pgw = epcHelper->GetPgwNode();
NodeContainer remoteHostContainer;
remoteHostContainer.Create(1);
Ptr<Node> remoteHost = remoteHostContainer.Get(0);
InternetStackHelper internet;
internet.Install(remoteHostContainer);
```

```
// connect a remoteHost to pgw. Setup routing too
PointToPointHelper p2ph;
p2ph.SetDeviceAttribute("DataRate", DataRateValue(DataRate("100Gb/s")));
p2ph.SetDeviceAttribute("Mtu", UintegerValue(2500));
p2ph.SetChannelAttribute("Delay", TimeValue(Seconds(0.000)));
NetDeviceContainer internetDevices = p2ph.Install(pgw, remoteHost);
Ipv4AddressHelper ipv4h;
Ipv4StaticRoutingHelper ipv4RoutingHelper;
ipv4h.SetBase("1.0.0.0", "255.0.0.0");
Ipv4InterfaceContainer internetIpIfaces = ipv4h.Assign(internetDevices);
Ptr<Ipv4StaticRouting> remoteHostStaticRouting =
   ipv4RoutingHelper.GetStaticRouting(remoteHost->GetObject<Ipv4>());
remoteHostStaticRouting->AddNetworkRouteTo(Ipv4Address("7.0.0.0"), Ipv4Mask("255.0.0.0"), 1);
internet.Install(ueNodes);
Ipv4InterfaceContainer ueIpIface;
ueIpIface = epcHelper->AssignUeIpv4Address(NetDeviceContainer(ueNetDev));

// Set the default gateway for the UEs
for (uint32_t j = 0; j < ueNodes.GetN(); ++j)
{
   Ptr<Ipv4StaticRouting> ueStaticRouting =
      ipv4RoutingHelper.GetStaticRouting(ueNodes.Get(j)->GetObject<Ipv4>());
   ueStaticRouting->SetDefaultRoute(epcHelper->GetUeDefaultGatewayAddress(), 1);
}

// assign IP address to UEs, and install UDP downlink applications
uint16_t dlPort = 1234;
UdpServerHelper dlPacketSinkHelper(dlPort);
ApplicationContainer txApps;
ApplicationContainer sinkApps;
NodeContainer txNodes;
NodeContainer sinkNodes;
Ipv4InterfaceContainer sinkIps;

if (isUl)
{
   sinkIps.Add(internetIpIfaces.Get(1));
   sinkNodes = remoteHostContainer;
   txNodes = ueNodes;
}
else
{
   sinkIps = ueIpIface;
   sinkNodes = ueNodes;
   txNodes = remoteHostContainer;
}

// configure here UDP traffic
for (uint32_t i = 0; i < txNodes.GetN(); ++i)
{
   for (uint32_t j = 0; j < sinkNodes.GetN(); ++j)
   {
      UdpClientHelper dlClient(sinkIps.GetAddress(j), dlPort);
      dlClient.SetAttribute("MaxPackets", UintegerValue(packets));
      dlClient.SetAttribute("PacketSize", UintegerValue(pktSize));
      dlClient.SetAttribute("Interval", TimeValue(packetInterval));

      txApps.Add(dlClient.Install(txNodes.Get(i)));
   }
}

sinkApps.Add(dlPacketSinkHelper.Install(sinkNodes));
for (uint32_t j = 0; j < sinkApps.GetN(); ++j)
```

```
  {
    Ptr<UdpServer> client = DynamicCast<UdpServer>(sinkApps.Get(j));
    NS_ASSERT(client != nullptr);
    std::stringstream ss;
    ss << j;
    client->TraceConnect("Rx", ss.str(), MakeCallback(&PrintRxPkt));
  }

  // start UDP server and client apps
  sinkApps.Start(udpAppStartTime);
  txApps.Start(udpAppStartTime);
  sinkApps.Stop(Seconds(simTime));
  txApps.Stop(Seconds(simTime));

  // attach UEs to the closest eNB
  nrHelper->AttachToClosestEnb(ueNetDev, gnbNetDev);

  // enable the traces provided by the nr module
  // nrHelper->EnableTraces();

  Simulator::Stop(Seconds(simTime));

  auto start = std::chrono::steady_clock::now();

  Simulator::Run();

  auto end = std::chrono::steady_clock::now();

  uint64_t sum = 0;
  uint32_t cont = 0;
  for (auto& v : packetsTime)
  {
    if (v < 100000)
    {
      sum += v;
      cont++;
    }
  }
  std::cout << "Packets received: " << packetsTime.size() << std::endl;
  std::cout << "Counter (packets not affected by reordering): " << +cont << std::endl;

  if (packetsTime.size() > 0 && cont > 0)
  {
    std::cout << "Average e2e latency (over all received packets): " << sum / packetsTime.size()
              << " us" << std::endl;
    std::cout << "Average e2e latency (over counter): " << sum / cont << " us" << std::endl;
  }
  else
  {
    std::cout << "Average e2e latency: Not Available" << std::endl;
  }

  for (auto it = sinkApps.Begin(); it != sinkApps.End(); ++it)
  {
    uint64_t recv = DynamicCast<UdpServer>(*it)->GetReceived();
    std::cout << "Sent: " << packets << " Recv: " << recv << " Lost: " << packets - recv
              << " pkts, ( " << (static_cast<double>(packets - recv) / packets) * 100.0
              << " % )" << std::endl;
  }

  Simulator::Destroy();

  std::cout << "Running time: "
            << std::chrono::duration_cast<std::chrono::seconds>(end - start).count() << " s."
```

```
        << std::endl;
  return 0;
```

Και τα αντίστοιχα πειραματικά αποτελέσματα:



Εικόνα 16: Πειραματικά Αποτελέσματα Παραδείγματος 5

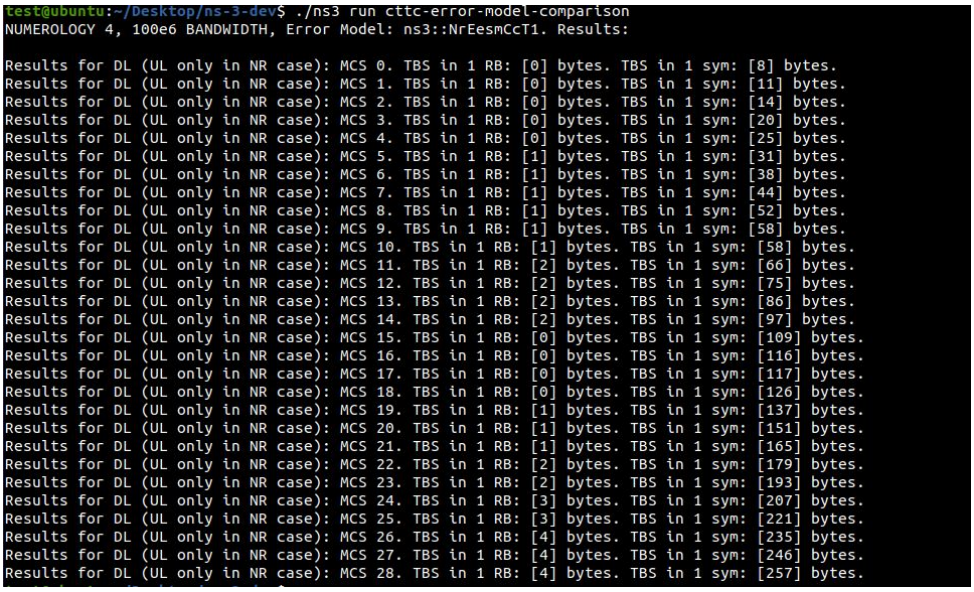
3.6. Παράδειγμα 6

Παρατίθεται ο κώδικας, που χρησιμοποιήθηκε:

```
/**
 * \file cttc-error-model-comparison.cc
 * \ingroup examples
 * \brief Error model example comparison: TBS for all MCSs.
 *
 * This example allows the user to compare the Transport Block Size that is obtained
 * for each MCS index under different error models (NR and LTE) and different MCS Tables.
 *
 * The NR error model can be set as "--errorModel=ns3::NrEesmCcT1", for HARQ-CC and MCS Table1,
 * while "--errorModel=ns3::NrLteMiErrorModel" configures the LTE error model.
 * For NR, you can choose between different types of error model, which use
 * different tables and different methods to process the HARQ history, e.g.,
 * "--errorModel=ns3::NrEesmIrT1", for HARQ-IR and MCS Table2.
 *
 * There is no deployment scenario configured, the example directly computes the TBS
 * for all MCSs of the configured error model and MCS Table, assuming numerology 4
 * and 100 MHz of channel bandwidth.
 *
 * This simulation prints the output to the terminal, showing for each MCS: 1)
 * the TBS that fits in 1 OFDM symbol (whole bandwidth) and 2) the TBS that fits
 * in 1 OFDM symbol and a single RB.
 *
 * To run the simulation with the default configuration one shall run the
 * following in the command line:
 *
 * ./ns3 run cttc-error-model-comparison
 *
 */
```

```
using namespace ns3;

NS_LOG_COMPONENT_DEFINE("CttcErrorModelComparisonExample");

int
main(int argc, char* argv[])
{
   std::string errorModel = "ns3::NrEesmCcT1";

   CommandLine cmd(__FILE__);

   cmd.AddValue("errorModelType",
                "Error model type: ns3::NrEesmCcT1, ns3::NrEesmCcT2, ns3::NrEesmIrT1, "
                "ns3::NrEesmIrT2, ns3::NrLteMiErrorModel",
                errorModel);

   cmd.Parse(argc, argv);

   /*
    * TODO: remove all the instances of SetDefault, NrEesmErrorModel, NrAmc
    */
   Config::SetDefault("ns3::NrAmc::ErrorModelType", TypeIdValue(TypeId::LookupByName(errorModel)));
   Config::SetDefault("ns3::NrAmc::AmcModel", EnumValue(NrAmc::ShannonModel));

   // Compute number of RBs that fit in 100 MHz channel bandwidth with numerology 4 (240 kHz SCS)
   const uint8_t numerology = 4;
   const uint32_t bandwidth = 100e6;
   const uint32_t numRbsInBandwidth = bandwidth / (15e3 * std::pow(2, numerology) * 12);

   Ptr<NrAmc> amc = CreateObject<NrAmc>();
   amc->SetDlMode();

   std::string tbs;
   for (uint32_t mcs = 0; mcs <= amc->GetMaxMcs(); ++mcs)
   {
      std::stringstream ss;
      ss << "\nResults for DL (UL only in NR case): MCS " << mcs << ". TBS in 1 RB: ["
         << amc->CalculateTbSize(mcs, 1) << "] bytes. TBS in 1 sym: ["
         << amc->CalculateTbSize(mcs, numRbsInBandwidth) << "] bytes.";
      tbs += ss.str();
   }

   std::cout << "NUMEROLOGY 4, 100e6 BANDWIDTH, Error Model: ";
   std::cout << errorModel << ". Results: " << std::endl;
   std::cout << tbs << std::endl;

   return 0;
}
```

Και τα αντίστοιχα πειραματικά αποτελέσματα:



Εικόνα 17: Πειραματικά Αποτελέσματα Παραδείγματος 6

3.7. Παράδειγμα 7

Παρατίθεται ο κώδικας, που χρησιμοποιήθηκε:

```
/**
 * \ingroup examples
 * \file cttc-nr-demo.cc
 * \brief A cozy, simple, NR demo (in a tutorial style)
 *
 * This example describes how to setup a simulation using the 3GPP channel model
 * from TR 38.900. This example consists of a simple grid topology, in which you
 * can choose the number of gNbs and UEs. Have a look at the possible parameters
 * to know what you can configure through the command line.
 *
 * With the default configuration, the example will create two flows that will
 * go through two different subband numerologies (or bandwidth parts). For that,
 * specifically, two bands are created, each with a single CC, and each CC containing
 * one bandwidth part.
 *
 * The example will print on-screen the end-to-end result of one (or two) flows,
 * as well as writing them on a file.
 *
 * \code{.unparsed}
$ ./ns3 run "cttc-nr-demo --PrintHelp"
   \endcode
 *
 */

/*
 * Include part. Often, you will have to include the headers for an entire module;
```

```
 * do that by including the name of the module you need with the suffix "-module.h".
 */

#include "ns3/antenna-module.h"
#include "ns3/applications-module.h"
#include "ns3/buildings-module.h"
#include "ns3/config-store-module.h"
#include "ns3/core-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/internet-apps-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/network-module.h"
#include "ns3/nr-module.h"
#include "ns3/point-to-point-module.h"

/*
 * Use, always, the namespace ns3. All the NR classes are inside such namespace.
 */
using namespace ns3;

/*
 * With this line, we will be able to see the logs of the file by enabling the
 * component "CttcNrDemo"
 */
NS_LOG_COMPONENT_DEFINE("CttcNrDemo");

int
main(int argc, char* argv[])
{
    /*
     * Variables that represent the parameters we will accept as input by the
     * command line. Each of them is initialized with a default value, and
     * possibly overridden below when command-line arguments are parsed.
     */
    // Scenario parameters (that we will use inside this script):
    uint16_t gNbNum = 1;
    uint16_t ueNumPergNb = 2;
    bool logging = false;
    bool doubleOperationalBand = true;

    // Traffic parameters (that we will use inside this script):
    uint32_t udpPacketSizeULL = 100;
    uint32_t udpPacketSizeBe = 1252;
    uint32_t lambdaULL = 10000;
    uint32_t lambdaBe = 10000;

    // Simulation parameters. Please don't use double to indicate seconds; use
    // ns-3 Time values which use integers to avoid portability issues.
    Time simTime = MilliSeconds(1000);
    Time udpAppStartTime = MilliSeconds(400);

    // NR parameters. We will take the input from the command line, and then we
    // will pass them inside the NR module.
    uint16_t numerologyBwp1 = 4;
    double centralFrequencyBand1 = 28e9;
    double bandwidthBand1 = 100e6;
    uint16_t numerologyBwp2 = 2;
    double centralFrequencyBand2 = 28.2e9;
    double bandwidthBand2 = 100e6;
    double totalTxPower = 4;

    // Where we will store the output files.
    std::string simTag = "default";
```

```
std::string outputDir = "./";

/*
 * From here, we instruct the ns3::CommandLine class of all the input parameters
 * that we may accept as input, as well as their description, and the storage
 * variable.
 */
CommandLine cmd(__FILE__);

cmd.AddValue("gNbNum", "The number of gNbs in multiple-ue topology", gNbNum);
cmd.AddValue("ueNumPergNb", "The number of UE per gNb in multiple-ue topology", ueNumPergNb);
cmd.AddValue("logging", "Enable logging", logging);
cmd.AddValue("doubleOperationalBand",
        "If true, simulate two operational bands with one CC for each band,"
        "and each CC will have 1 BWP that spans the entire CC.",
        doubleOperationalBand);
cmd.AddValue("packetSizeUll",
        "packet size in bytes to be used by ultra low latency traffic",
        udpPacketSizeULL);
cmd.AddValue("packetSizeBe",
        "packet size in bytes to be used by best effort traffic",
        udpPacketSizeBe);
cmd.AddValue("lambdaUll",
        "Number of UDP packets in one second for ultra low latency traffic",
        lambdaULL);
cmd.AddValue("lambdaBe",
        "Number of UDP packets in one second for best effor traffic",
        lambdaBe);
cmd.AddValue("simTime", "Simulation time", simTime);
cmd.AddValue("numerologyBwp1", "The numerology to be used in bandwidth part 1", numerologyBwp1);
cmd.AddValue("centralFrequencyBand1",
        "The system frequency to be used in band 1",
        centralFrequencyBand1);
cmd.AddValue("bandwidthBand1", "The system bandwidth to be used in band 1", bandwidthBand1);
cmd.AddValue("numerologyBwp2", "The numerology to be used in bandwidth part 2", numerologyBwp2);
cmd.AddValue("centralFrequencyBand2",
        "The system frequency to be used in band 2",
        centralFrequencyBand2);
cmd.AddValue("bandwidthBand2", "The system bandwidth to be used in band 2", bandwidthBand2);
cmd.AddValue("totalTxPower",
        "total tx power that will be proportionally assigned to"
        " bands, CCs and bandwidth parts depending on each BWP bandwidth ",
        totalTxPower);
cmd.AddValue("simTag",
        "tag to be appended to output filenames to distinguish simulation campaigns",
        simTag);
cmd.AddValue("outputDir", "directory where to store simulation results", outputDir);

// Parse the command line
cmd.Parse(argc, argv);

/*
 * Check if the frequency is in the allowed range.
 * If you need to add other checks, here is the best position to put them.
 */
NS_ABORT_IF(centralFrequencyBand1 > 100e9);
NS_ABORT_IF(centralFrequencyBand2 > 100e9);

/*
 * If the logging variable is set to true, enable the log of some components
 * through the code. The same effect can be obtained through the use
 * of the NS_LOG environment variable:
 *
 * export NS_LOG="UdpClient=level_info|prefix_time|prefix_func|prefix_node:UdpServer=..."
```

```
 *
 * Usually, the environment variable way is preferred, as it is more customizable,
 * and more expressive.
 */
if (logging)
{
   LogComponentEnable("UdpClient", LOG_LEVEL_INFO);
   LogComponentEnable("UdpServer", LOG_LEVEL_INFO);
   LogComponentEnable("LtePdcp", LOG_LEVEL_INFO);
}

/*
 * Default values for the simulation. We are progressively removing all
 * the instances of SetDefault, but we need it for legacy code (LTE)
 */
Config::SetDefault("ns3::LteRlcUm::MaxTxBufferSize", UintegerValue(999999999));

/*
 * Create the scenario. In our examples, we heavily use helpers that setup
 * the gnbs and ue following a pre-defined pattern. Please have a look at the
 * GridScenarioHelper documentation to see how the nodes will be distributed.
 */
int64_t randomStream = 1;
GridScenarioHelper gridScenario;
gridScenario.SetRows(1);
gridScenario.SetColumns(gNbNum);
gridScenario.SetHorizontalBsDistance(5.0);
gridScenario.SetVerticalBsDistance(5.0);
gridScenario.SetBsHeight(1.5);
gridScenario.SetUtHeight(1.5);
// must be set before BS number
gridScenario.SetSectorization(GridScenarioHelper::SINGLE);
gridScenario.SetBsNumber(gNbNum);
gridScenario.SetUtNumber(ueNumPergNb * gNbNum);
gridScenario.SetScenarioHeight(3); // Create a 3x3 scenario where the UE will
gridScenario.SetScenarioLength(3); // be distribuited.
randomStream += gridScenario.AssignStreams(randomStream);
gridScenario.CreateScenario();

/*
 * Create two different NodeContainer for the different traffic type.
 * In ueLowLat we will put the UEs that will receive low-latency traffic,
 * while in ueVoice we will put the UEs that will receive the voice traffic.
 */
NodeContainer ueLowLatContainer;
NodeContainer ueVoiceContainer;

for (uint32_t j = 0; j < gridScenario.GetUserTerminals().GetN(); ++j)
{
   Ptr<Node> ue = gridScenario.GetUserTerminals().Get(j);
   if (j % 2 == 0)
   {
      ueLowLatContainer.Add(ue);
   }
   else
   {
      ueVoiceContainer.Add(ue);
   }
}

/*
 * TODO: Add a print, or a plot, that shows the scenario.
 */
```

```
/*
 * Setup the NR module. We create the various helpers needed for the
 * NR simulation:
 * - EpcHelper, which will setup the core network
 * - IdealBeamformingHelper, which takes care of the beamforming part
 * - NrHelper, which takes care of creating and connecting the various
 * part of the NR stack
 */
Ptr<NrPointToPointEpcHelper> epcHelper = CreateObject<NrPointToPointEpcHelper>();
Ptr<IdealBeamformingHelper> idealBeamformingHelper = CreateObject<IdealBeamformingHelper>();
Ptr<NrHelper> nrHelper = CreateObject<NrHelper>();

// Put the pointers inside nrHelper
nrHelper->SetBeamformingHelper(idealBeamformingHelper);
nrHelper->SetEpcHelper(epcHelper);

/*
 * Spectrum division. We create two operational bands, each of them containing
 * one component carrier, and each CC containing a single bandwidth part
 * centered at the frequency specified by the input parameters.
 * Each spectrum part length is, as well, specified by the input parameters.
 * Both operational bands will use the StreetCanyon channel modeling.
 */
BandwidthPartInfoPtrVector allBwps;
CcBwpCreator ccBwpCreator;
const uint8_t numCcPerBand = 1; // in this example, both bands have a single CC

// Create the configuration for the CcBwpHelper. SimpleOperationBandConf creates
// a single BWP per CC
CcBwpCreator::SimpleOperationBandConf bandConf1(centralFrequencyBand1,
                                    bandwidthBand1,
                                    numCcPerBand,
                                    BandwidthPartInfo::UMi_StreetCanyon);
CcBwpCreator::SimpleOperationBandConf bandConf2(centralFrequencyBand2,
                                    bandwidthBand2,
                                    numCcPerBand,
                                    BandwidthPartInfo::UMi_StreetCanyon);

// By using the configuration created, it is time to make the operation bands
OperationBandInfo band1 = ccBwpCreator.CreateOperationBandContiguousCc(bandConf1);
OperationBandInfo band2 = ccBwpCreator.CreateOperationBandContiguousCc(bandConf2);

/*
 * The configured spectrum division is:
 * ------------Band1--------------|--------------Band2-----------------
 * ------------CC1----------------|--------------CC2-------------------
 * ------------BWP1---------------|--------------BWP2------------------
 */

/*
 * Attributes of ThreeGppChannelModel still cannot be set in our way.
 * TODO: Coordinate with Tommaso
 */
Config::SetDefault("ns3::ThreeGppChannelModel::UpdatePeriod", TimeValue(MilliSeconds(0)));
nrHelper->SetChannelConditionModelAttribute("UpdatePeriod", TimeValue(MilliSeconds(0)));
nrHelper->SetPathlossAttribute("ShadowingEnabled", BooleanValue(false));

/*
 * Initialize channel and pathloss, plus other things inside band1. If needed,
 * the band configuration can be done manually, but we leave it for more
 * sophisticated examples. For the moment, this method will take care
 * of all the spectrum initialization needs.
 */
nrHelper->InitializeOperationBand(&band1);
```

```
/*
 * Start to account for the bandwidth used by the example, as well as
 * the total power that has to be divided among the BWPs.
 */
double x = pow(10, totalTxPower / 10);
double totalBandwidth = bandwidthBand1;

/*
 * if not single band simulation, initialize and setup power in the second band
 */
if (doubleOperationalBand)
{
   // Initialize channel and pathloss, plus other things inside band2
   nrHelper->InitializeOperationBand(&band2);
   totalBandwidth += bandwidthBand2;
   allBwps = CcBwpCreator::GetAllBwps({band1, band2});
}
else
{
   allBwps = CcBwpCreator::GetAllBwps({band1});
}

/*
 * allBwps contains all the spectrum configuration needed for the nrHelper.
 *
 * Now, we can setup the attributes. We can have three kind of attributes:
 * (i) parameters that are valid for all the bandwidth parts and applies to
 * all nodes, (ii) parameters that are valid for all the bandwidth parts
 * and applies to some node only, and (iii) parameters that are different for
 * every bandwidth parts. The approach is:
 *
 * - for (i): Configure the attribute through the helper, and then install;
 * - for (ii): Configure the attribute through the helper, and then install
 * for the first set of nodes. Then, change the attribute through the helper,
 * and install again;
 * - for (iii): Install, and then configure the attributes by retrieving
 * the pointer needed, and calling "SetAttribute" on top of such pointer.
 *
 */

Packet::EnableChecking();
Packet::EnablePrinting();

/*
 *  Case (i): Attributes valid for all the nodes
 */
// Beamforming method
idealBeamformingHelper->SetAttribute("BeamformingMethod",
                            TypeIdValue(DirectPathBeamforming::GetTypeId()));

// Core latency
epcHelper->SetAttribute("S1uLinkDelay", TimeValue(MilliSeconds(0)));

// Antennas for all the UEs
nrHelper->SetUeAntennaAttribute("NumRows", UintegerValue(2));
nrHelper->SetUeAntennaAttribute("NumColumns", UintegerValue(4));
nrHelper->SetUeAntennaAttribute("AntennaElement",
                          PointerValue(CreateObject<IsotropicAntennaModel>()));

// Antennas for all the gNbs
nrHelper->SetGnbAntennaAttribute("NumRows", UintegerValue(4));
nrHelper->SetGnbAntennaAttribute("NumColumns", UintegerValue(8));
nrHelper->SetGnbAntennaAttribute("AntennaElement",
```

```
                        PointerValue(CreateObject<IsotropicAntennaModel>()));

uint32_t bwpIdForLowLat = 0;
uint32_t bwpIdForVoice = 0;
if (doubleOperationalBand)
{
   bwpIdForVoice = 1;
   bwpIdForLowLat = 0;
}

// gNb routing between Bearer and bandwidh part
nrHelper->SetGnbBwpManagerAlgorithmAttribute("NGBR_LOW_LAT_EMBB",
                                UintegerValue(bwpIdForLowLat));
nrHelper->SetGnbBwpManagerAlgorithmAttribute("GBR_CONV_VOICE", UintegerValue(bwpIdForVoice));

// Ue routing between Bearer and bandwidth part
nrHelper->SetUeBwpManagerAlgorithmAttribute("NGBR_LOW_LAT_EMBB", UintegerValue(bwpIdForLowLat));
nrHelper->SetUeBwpManagerAlgorithmAttribute("GBR_CONV_VOICE", UintegerValue(bwpIdForVoice));

/*
 * We miss many other parameters. By default, not configuring them is equivalent
 * to use the default values. Please, have a look at the documentation to see
 * what are the default values for all the attributes you are not seeing here.
 */

/*
 * Case (ii): Attributes valid for a subset of the nodes
 */

// NOT PRESENT IN THIS SIMPLE EXAMPLE

/*
 * We have configured the attributes we needed. Now, install and get the pointers
 * to the NetDevices, which contains all the NR stack:
 */

NetDeviceContainer enbNetDev =
   nrHelper->InstallGnbDevice(gridScenario.GetBaseStations(), allBwps);
NetDeviceContainer ueLowLatNetDev = nrHelper->InstallUeDevice(ueLowLatContainer, allBwps);
NetDeviceContainer ueVoiceNetDev = nrHelper->InstallUeDevice(ueVoiceContainer, allBwps);

randomStream += nrHelper->AssignStreams(enbNetDev, randomStream);
randomStream += nrHelper->AssignStreams(ueLowLatNetDev, randomStream);
randomStream += nrHelper->AssignStreams(ueVoiceNetDev, randomStream);
/*
 * Case (iii): Go node for node and change the attributes we have to setup
 * per-node.
 */

// Get the first netdevice (enbNetDev.Get (0)) and the first bandwidth part (0)
// and set the attribute.
nrHelper->GetGnbPhy(enbNetDev.Get(0), 0)
   ->SetAttribute("Numerology", UintegerValue(numerologyBwp1));
nrHelper->GetGnbPhy(enbNetDev.Get(0), 0)
   ->SetAttribute("TxPower", DoubleValue(10 * log10((bandwidthBand1 / totalBandwidth) * x)));

if (doubleOperationalBand)
{
   // Get the first netdevice (enbNetDev.Get (0)) and the second bandwidth part (1)
   // and set the attribute.
   nrHelper->GetGnbPhy(enbNetDev.Get(0), 1)
      ->SetAttribute("Numerology", UintegerValue(numerologyBwp2));
   nrHelper->GetGnbPhy(enbNetDev.Get(0), 1)
      ->SetTxPower(10 * log10((bandwidthBand2 / totalBandwidth) * x));
```

```
}

// When all the configuration is done, explicitly call UpdateConfig ()

for (auto it = enbNetDev.Begin(); it != enbNetDev.End(); ++it)
{
  DynamicCast<NrGnbNetDevice>(*it)->UpdateConfig();
}

for (auto it = ueLowLatNetDev.Begin(); it != ueLowLatNetDev.End(); ++it)
{
  DynamicCast<NrUeNetDevice>(*it)->UpdateConfig();
}

for (auto it = ueVoiceNetDev.Begin(); it != ueVoiceNetDev.End(); ++it)
{
  DynamicCast<NrUeNetDevice>(*it)->UpdateConfig();
}

// From here, it is standard NS3. In the future, we will create helpers
// for this part as well.

// create the internet and install the IP stack on the UEs
// get SGW/PGW and create a single RemoteHost
Ptr<Node> pgw = epcHelper->GetPgwNode();
NodeContainer remoteHostContainer;
remoteHostContainer.Create(1);
Ptr<Node> remoteHost = remoteHostContainer.Get(0);
InternetStackHelper internet;
internet.Install(remoteHostContainer);

// connect a remoteHost to pgw. Setup routing too
PointToPointHelper p2ph;
p2ph.SetDeviceAttribute("DataRate", DataRateValue(DataRate("100Gb/s")));
p2ph.SetDeviceAttribute("Mtu", UintegerValue(2500));
p2ph.SetChannelAttribute("Delay", TimeValue(Seconds(0.000)));
NetDeviceContainer internetDevices = p2ph.Install(pgw, remoteHost);
Ipv4AddressHelper ipv4h;
Ipv4StaticRoutingHelper ipv4RoutingHelper;
ipv4h.SetBase("1.0.0.0", "255.0.0.0");
Ipv4InterfaceContainer internetIpIfaces = ipv4h.Assign(internetDevices);
Ptr<Ipv4StaticRouting> remoteHostStaticRouting =
  ipv4RoutingHelper.GetStaticRouting(remoteHost->GetObject<Ipv4>());
remoteHostStaticRouting->AddNetworkRouteTo(Ipv4Address("7.0.0.0"), Ipv4Mask("255.0.0.0"), 1);
internet.Install(gridScenario.GetUserTerminals());

Ipv4InterfaceContainer ueLowLatIpIface =
  epcHelper->AssignUeIpv4Address(NetDeviceContainer(ueLowLatNetDev));
Ipv4InterfaceContainer ueVoiceIpIface =
  epcHelper->AssignUeIpv4Address(NetDeviceContainer(ueVoiceNetDev));

// Set the default gateway for the UEs
for (uint32_t j = 0; j < gridScenario.GetUserTerminals().GetN(); ++j)
{
  Ptr<Ipv4StaticRouting> ueStaticRouting = ipv4RoutingHelper.GetStaticRouting(
    gridScenario.GetUserTerminals().Get(j)->GetObject<Ipv4>());
  ueStaticRouting->SetDefaultRoute(epcHelper->GetUeDefaultGatewayAddress(), 1);
}

// attach UEs to the closest eNB
nrHelper->AttachToClosestEnb(ueLowLatNetDev, enbNetDev);
nrHelper->AttachToClosestEnb(ueVoiceNetDev, enbNetDev);

/*
```

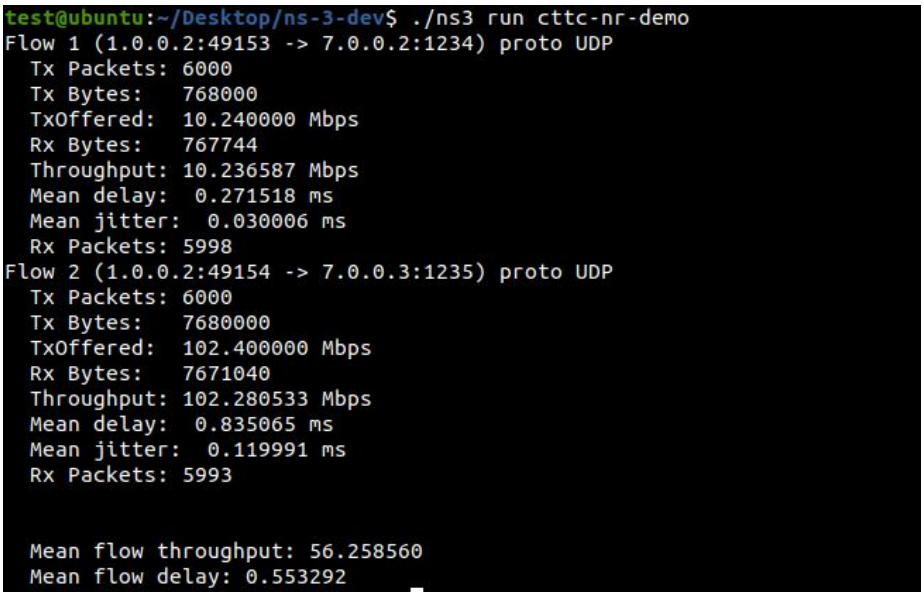
```
 * Traffic part. Install two kind of traffic: low-latency and voice, each
 * identified by a particular source port.
 */
uint16_t dlPortLowLat = 1234;
uint16_t dlPortVoice = 1235;

ApplicationContainer serverApps;

// The sink will always listen to the specified ports
UdpServerHelper dlPacketSinkLowLat(dlPortLowLat);
UdpServerHelper dlPacketSinkVoice(dlPortVoice);

// The server, that is the application which is listening, is installed in the UE
serverApps.Add(dlPacketSinkLowLat.Install(ueLowLatContainer));
serverApps.Add(dlPacketSinkVoice.Install(ueVoiceContainer));

/*
 * Configure attributes for the different generators, using user-provided
 * parameters for generating a CBR traffic
 *
 * Low-Latency configuration and object creation:
 */
UdpClientHelper dlClientLowLat;
dlClientLowLat.SetAttribute("RemotePort", UintegerValue(dlPortLowLat));
dlClientLowLat.SetAttribute("MaxPackets", UintegerValue(0xFFFFFFFF));
dlClientLowLat.SetAttribute("PacketSize", UintegerValue(udpPacketSizeULL));
dlClientLowLat.SetAttribute("Interval", TimeValue(Seconds(1.0 / lambdaULL)));

// The bearer that will carry low latency traffic
EpsBearer lowLatBearer(EpsBearer::NGBR_LOW_LAT_EMBB);

// The filter for the low-latency traffic
Ptr<EpcTft> lowLatTft = Create<EpcTft>();
EpcTft::PacketFilter dlpfLowLat;
dlpfLowLat.localPortStart = dlPortLowLat;
dlpfLowLat.localPortEnd = dlPortLowLat;
lowLatTft->Add(dlpfLowLat);

// Voice configuration and object creation:
UdpClientHelper dlClientVoice;
dlClientVoice.SetAttribute("RemotePort", UintegerValue(dlPortVoice));
dlClientVoice.SetAttribute("MaxPackets", UintegerValue(0xFFFFFFFF));
dlClientVoice.SetAttribute("PacketSize", UintegerValue(udpPacketSizeBe));
dlClientVoice.SetAttribute("Interval", TimeValue(Seconds(1.0 / lambdaBe)));

// The bearer that will carry voice traffic
EpsBearer voiceBearer(EpsBearer::GBR_CONV_VOICE);

// The filter for the voice traffic
Ptr<EpcTft> voiceTft = Create<EpcTft>();
EpcTft::PacketFilter dlpfVoice;
dlpfVoice.localPortStart = dlPortVoice;
dlpfVoice.localPortEnd = dlPortVoice;
voiceTft->Add(dlpfVoice);

/*
 * Let's install the applications!
 */
ApplicationContainer clientApps;

for (uint32_t i = 0; i < ueLowLatContainer.GetN(); ++i)
{
   Ptr<Node> ue = ueLowLatContainer.Get(i);
   Ptr<NetDevice> ueDevice = ueLowLatNetDev.Get(i);
```

```
    Address ueAddress = ueLowLatIpIface.GetAddress(i);

    // The client, who is transmitting, is installed in the remote host,
    // with destination address set to the address of the UE
    dlClientLowLat.SetAttribute("RemoteAddress", AddressValue(ueAddress));
    clientApps.Add(dlClientLowLat.Install(remoteHost));

    // Activate a dedicated bearer for the traffic type
    nrHelper->ActivateDedicatedEpsBearer(ueDevice, lowLatBearer, lowLatTft);
  }

 for (uint32_t i = 0; i < ueVoiceContainer.GetN(); ++i)
  {
    Ptr<Node> ue = ueVoiceContainer.Get(i);
    Ptr<NetDevice> ueDevice = ueVoiceNetDev.Get(i);
    Address ueAddress = ueVoiceIpIface.GetAddress(i);

    // The client, who is transmitting, is installed in the remote host,
    // with destination address set to the address of the UE
    dlClientVoice.SetAttribute("RemoteAddress", AddressValue(ueAddress));
    clientApps.Add(dlClientVoice.Install(remoteHost));

    // Activate a dedicated bearer for the traffic type
    nrHelper->ActivateDedicatedEpsBearer(ueDevice, voiceBearer, voiceTft);
  }

 // start UDP server and client apps
 serverApps.Start(udpAppStartTime);
 clientApps.Start(udpAppStartTime);
 serverApps.Stop(simTime);
 clientApps.Stop(simTime);

 // enable the traces provided by the nr module
 // nrHelper->EnableTraces();

 FlowMonitorHelper flowmonHelper;
 NodeContainer endpointNodes;
 endpointNodes.Add(remoteHost);
 endpointNodes.Add(gridScenario.GetUserTerminals());

 Ptr<ns3::FlowMonitor> monitor = flowmonHelper.Install(endpointNodes);
 monitor->SetAttribute("DelayBinWidth", DoubleValue(0.001));
 monitor->SetAttribute("JitterBinWidth", DoubleValue(0.001));
 monitor->SetAttribute("PacketSizeBinWidth", DoubleValue(20));

 Simulator::Stop(simTime);
 Simulator::Run();

 /*
  * To check what was installed in the memory, i.e., BWPs of eNb Device, and its configuration.
  * Example is: Node 1 -> Device 0 -> BandwidthPartMap -> {0,1} BWPs -> NrGnbPhy -> Numerology,
 GtkConfigStore config;
 config.ConfigureAttributes ();
 */

 // Print per-flow statistics
 monitor->CheckForLostPackets();
 Ptr<Ipv4FlowClassifier> classifier =
    DynamicCast<Ipv4FlowClassifier>(flowmonHelper.GetClassifier());
 FlowMonitor::FlowStatsContainer stats = monitor->GetFlowStats();

 double averageFlowThroughput = 0.0;
 double averageFlowDelay = 0.0;
```

```
  std::ofstream outFile;
  std::string filename = outputDir + "/" + simTag;
  outFile.open(filename.c_str(), std::ofstream::out | std::ofstream::trunc);
  if (!outFile.is_open())
  {
    std::cerr << "Can't open file " << filename << std::endl;
    return 1;
  }

  outFile.setf(std::ios_base::fixed);

  double flowDuration = (simTime - udpAppStartTime).GetSeconds();
  for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin();
     i != stats.end();
     ++i)
  {
    Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(i->first);
    std::stringstream protoStream;
    protoStream << (uint16_t)t.protocol;
    if (t.protocol == 6)
    {
      protoStream.str("TCP");
    }
    if (t.protocol == 17)
    {
      protoStream.str("UDP");
    }
    outFile << "Flow " << i->first << " (" << t.sourceAddress << ":" << t.sourcePort << " -> "
        << t.destinationAddress << ":" << t.destinationPort << ") proto "
        << protoStream.str() << "\n";
    outFile << "  Tx Packets: " << i->second.txPackets << "\n";
    outFile << "  Tx Bytes:   " << i->second.txBytes << "\n";
    outFile << "  TxOffered:  " << i->second.txBytes * 8.0 / flowDuration / 1000.0 / 1000.0
        << " Mbps\n";
    outFile << "  Rx Bytes:   " << i->second.rxBytes << "\n";
    if (i->second.rxPackets > 0)
    {
      // Measure the duration of the flow from receiver's perspective
      averageFlowThroughput += i->second.rxBytes * 8.0 / flowDuration / 1000 / 1000;
      averageFlowDelay += 1000 * i->second.delaySum.GetSeconds() / i->second.rxPackets;

      outFile << "  Throughput: " << i->second.rxBytes * 8.0 / flowDuration / 1000 / 1000
          << " Mbps\n";
      outFile << "  Mean delay:  "
          << 1000 * i->second.delaySum.GetSeconds() / i->second.rxPackets << " ms\n";
      // outFile << "  Mean upt:  " << i->second.uptSum / i->second.rxPackets / 1000/1000 << "
      // Mbps \n";
      outFile << "  Mean jitter:  "
          << 1000 * i->second.jitterSum.GetSeconds() / i->second.rxPackets << " ms\n";
    }
    else
    {
      outFile << "  Throughput:  0 Mbps\n";
      outFile << "  Mean delay:  0 ms\n";
      outFile << "  Mean jitter: 0 ms\n";
    }
    outFile << "  Rx Packets: " << i->second.rxPackets << "\n";
  }

  double meanFlowThroughput = averageFlowThroughput / stats.size();
  double meanFlowDelay = averageFlowDelay / stats.size();

  outFile << "\n\n  Mean flow throughput: " << meanFlowThroughput << "\n";
  outFile << "  Mean flow delay: " << meanFlowDelay << "\n";
```

```
  outFile.close();

  std::ifstream f(filename.c_str());

  if (f.is_open())
  {
    std::cout << f.rdbuf();
  }

  Simulator::Destroy();

  if (argc == 0)
  {
    double toleranceMeanFlowThroughput = 0.0001 * 56.258560;
    double toleranceMeanFlowDelay = 0.0001 * 0.553292;

    if (meanFlowThroughput >= 56.258560 - toleranceMeanFlowThroughput &&
      meanFlowThroughput <= 56.258560 + toleranceMeanFlowThroughput &&
      meanFlowDelay >= 0.553292 - toleranceMeanFlowDelay &&
      meanFlowDelay <= 0.553292 + toleranceMeanFlowDelay)
    {
      return EXIT_SUCCESS;
    }
    else
    {
      return EXIT_FAILURE;
    }
  }
  else if (argc == 1 and ueNumPergNb == 9) // called from examples-to-run.py with these parameters
  {
    double toleranceMeanFlowThroughput = 0.0001 * 47.858536;
    double toleranceMeanFlowDelay = 0.0001 * 10.504189;

    if (meanFlowThroughput >= 47.858536 - toleranceMeanFlowThroughput &&
      meanFlowThroughput <= 47.858536 + toleranceMeanFlowThroughput &&
      meanFlowDelay >= 10.504189 - toleranceMeanFlowDelay &&
      meanFlowDelay <= 10.504189 + toleranceMeanFlowDelay)
    {
      return EXIT_SUCCESS;
    }
    else
    {
      return EXIT_FAILURE;
    }
  }
  else
  {
    return EXIT_SUCCESS; // we dont check other parameters configurations at the moment
  }
}
```

Και τα αντίστοιχα πειραματικά αποτελέσματα:



Εικόνα 18: Πειραματικά Αποτελέσματα Παραδείγματος 7

ΒΙΒΛΙΟΓΡΑΦΙΑ - ΠΗΓΕΣ

[1]. Bailey, S., Bansal, D., Dunbar, L., Hood, D., Kis, Z. L., MackCrane, B., ... & Varma, E. (2013). Sdn architecture overview. Open Networking Foundation, Ver, 1.

[2]. Dangi, R., Lalwani, P., Choudhary, G., You, I., & Pau, G. (2022). Study and investigation on 5G technology: A systematic review. Sensors, 22(1), 26.

[3]. Dogra, A., Jha, R. K., & Jain, S. (2020). A survey on beyond 5G network with the advent of 6G: Architecture and emerging technologies. IEEE Access, 9, 67512-67547.

[4]. ETSI, G. (2014). Network Functions Virtualization (NFV); Architectural Framework. ETsI Gs NFV, 2, V1.

[5]. Gutierrez, C. A., Caicedo, O., & Campos-Delgado, D. U. (2021). 5G and Beyond: Past, present and future of the mobile communications. IEEE Latin America Transactions, 19(10), 1702-1736.

[6]. Saad, W., Bennis, M., & Chen, M. (2019). A vision of 6G wireless systems: Applications, trends, technologies, and open research problems. IEEE network, 34(3), 134-142.

[7]. Viswanathan, H., & Mogensen, P. E. (2020). Communications in the 6G era. IEEE Access, 8, 57063-57074.

[8]. https://gitlab.com/cttc-lena/nr/-/blob/master/README.md

[9]. https://gitlab.com/cttc-lena/nr/-/tree/master/examples