Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Διεργασίες στο παρασκήνιο σε πλατφόρμα Android** **Background Job Processing (Android Platform)**
Ονοματεπώνυμο Φοιτητή	**Ευάγγελος Νικόλης**
Πατρώνυμο	**Θεόδωρος**
Αριθμός Μητρώου	**ΜΠΠΛ/ 16016**
Επιβλέπων	**Ευθύμιος Αλέπης, Επίκουρος καθηγητής**

Ημερομηνία Παράδοσης **Ιανουάριος 2019**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Ευθύμιος Αλέπης Επίκουρος Καθηγητής	Μαρία Βίρβου Καθηγήτρια	Κων/νος Πατσάκης Επίκουρος Καθηγητής

ΠΕΡΙΛΗΨΗ

Τα τελευταία χρόνια, η ραγδαία ανάπτυξη της τεχνολογίας έχει συμβάλει σημαντικά στην εξάπλωση των προϊόντων της στις σύγχρονες κοινωνίες, αφού αυτά έγιναν πιο προσιτά και πιο εύχρηστα. Η αγορά κατακλύστηκε από ηλεκτρονικούς υπολογιστές και φορητές συσκευές(κινητά και ταμπλέτες), αφού αυτά προσέφεραν την δυνατότητα παροχής λύσεων και κάλυψης αναγκών του ανθρώπου.

Αυτές οι συσκευές είναι εξοπλισμένες με ενσωματωμένους αισθητήρες, κάμερα, οθόνη αφής, περισσότερη μνήμη, ισχυρό επεξεργαστή, κινητά δίκτυα 3G και 4G και δυνατότητα Wi-Fi καθώς και αποδοτικούς μηχανισμούς κατανάλωσης ενέργειας. Αυτές οι βελτιώσεις έχουν οδηγήσει τις κινητές συσκευές να είναι σε θέση να εκτελούν εργασίες που συνήθως ήταν προσιτές μόνο για προσωπικούς υπολογιστές. Επιπλέον, λόγω της κυκλοφορίας των εφαρμογών Android OS και iOS για τέτοιες κινητές συσκευές, έχει αυξηθεί και η πολυπλοκότητά τους. Η σύνδεση στο διαδίκτυο έχει γίνει πανταχού παρούσα, καθώς τα δίκτυα Wi-Fi και κινητής τηλεφωνίας είναι διαθέσιμα στις περισσότερες συσκευές. Αυτές οι βελτιώσεις οδηγούν στην επόμενη γενιά υπηρεσιών που μπορούν να παρέχονται όχι μόνο από αποκλειστικούς διακομιστές αλλά και από κινητά τηλέφωνα.

Η διατριβή αυτή ασχολείται με τη σύνδεση πολλών κινητών συσκευών σε ένα κοινό δίκτυο Πελάτη - Εξυπηρετητή (Client - Server) με σκοπό την ταχύτερη εξαγωγή ενός αποτελέσματος, εκμεταλλευόμενη την επεξεργαστική ισχύ όλων των διαθέσιμων κινητών. Σε αυτό συμβάλλει η ταχεία ανάπτυξη επεξεργαστών τελευταίας γενιάς με πολλούς πυρήνες, και η εκμετάλλευση της ισχύς αυτών από την εφαρμογή.

ABSTRACT

In recent years, the rapid development of technology has contributed significantly to the spread of its products to modern societies, as they have become more accessible and easier to use. The market was overwhelmed by computers and portable devices (mobile and tablets), as these offered the ability to provide solutions and meet human needs.
These devices are equipped with built-in sensors, camera, touchscreen, more memory, powerful processor, 3G and 4G mobile networks and Wi-Fi, as well as efficient power consumption mechanisms. These improvements have led mobile devices to be able to perform tasks that were typically only accessible for personal computers. In addition, because of the Android OS and iOS app traffic for such mobile devices, their complexity has also increased. Internet connectivity has been ubiquitous since Wi-Fi and mobile data networks are available in most of the parties. These improvements lead to the next generation of services that can be provided not only by dedicated servers but also by mobile phones.
This dissertation deals with the connection of many mobile devices to a common Peer to Peer (P2P) network in order to expedite a result faster, taking advantage of the processing power of all available mobile devices. This contributes to the rapid development of the latest generation multi-core processors, and the exploitation of their power by application.

ΠΕΡΙΕΧΟΜΕΝΑ

Εισαγωγή

1.1 Στόχος της διατριβής

Στην εργασία αυτή παρουσιάζεται η υλοποίηση ενός δικτύου, το οποίο αποτελείται αποκλειστικά από κινητές συσκευές, και επιτρέπει την εκτέλεση παράλληλων διεργασιών. Ο κώδικας μπορεί να εγκατασταθεί σε συσκευές Android. Οι αλγόριθμοι που εφαρμόζονται μπορούν να εκτελούνται παράλληλα χρησιμοποιώντας τις συνδεδεμένες συσκευές. Οι μελλοντικοί προγραμματιστές μπορούν να προσαρμόσουν τους δικούς τους αλγόριθμους στον κώδικα. Οι στόχοι είναι οι εξής:

• Δημιουργία και συντήρηση δικτύου που χρησιμοποιεί συνδέσεις (WIFI-DIRECT) κινητής συσκευής.

• Ανάπτυξη πλατφόρμας για χρήση κινητών συσκευών σε παράλληλο υπολογιστικό σύστημα.

• Δημιουργία ευέλικτης υποδομής δικτύου που θα επιτρέψει την προσθήκη νέων αλγορίθμων και άλλων εξελίξεων.

• Δοκιμές και συγκρίσεις αποτελεσμάτων από διάφορες κινητές συσκευές που βρίσκονται συνδεδεμένες στο ίδιο δίκτυο.

Αναφέρονται λεπτομερώς οι έρευνες και οι δοκιμές που πραγματοποιήθηκαν ως μέρος της διαδικασίας σύνταξης των προδιαγραφών. Οι μέθοδοι και οι τεχνολογίες που εξετάστηκαν, καθώς και οι περιορισμοί που υπήρχαν στην εφαρμογή.

Περιλαμβάνεται επίσης μια σύνοψη των εννοιών που απαιτούνται για την κατανόηση του κώδικα και την προώθηση αυτού για μελλοντική χρήση. Παρόλο υπάρχει άφθονο υλικό άμεσα διαθέσιμο στο διαδίκτυο, συμπεριλαμβάνονται εξηγήσεις βασικών εννοιών έτσι ώστε αυτό το έγγραφο να είναι, σε κάποιο βαθμό, αυτόνομο.

Το επηρεαζόμενο δίκτυο που κατασκευάστηκε, αποτελείται από μια μόνο συσκευή διακομιστή (Server) και πολλές συσκευές χρηστών (Clients). Ο διακομιστής είναι ο κύριος κόμβος του δικτύου και το έργο του είναι συγκεκριμένο: η αποθήκευση, συντήρηση και διανομή ενός πίνακα ο οποίος καθορίζει τη λειτουργικότητα του δικτύου. Οι συσκευές χρήστη (Clients) επικοινωνούν με τον διακομιστή και λαμβάνουν μια εργασία που πρέπει να εκτελεστεί και εφόσον δεν έχουν κάποια άλλη διεργασία να εκτελέσουν, εκτελούν το έργο που τους δόθηκε.

Ένας από τους στόχους ήταν να δοθεί στο πρόγραμμα μια δομή έτσι ώστε η εφαρμογή, σε κάποια σημεία, να μπορεί εύκολα να αλλάξει ή να επεκταθεί. Με αυτό τον τρόπο, επιτρέπουμε στους προγραμματιστές να τροποποιήσουν τη συμπεριφορά του δικτύου.

Για την μέτρηση της απόδοση του προγράμματος, συνδέθηκαν μέχρι τέσσερις συσκευές στο δίκτυο και έγιναν συγκρίσεις των χρόνων εκτέλεσης δύο αλγορίθμων.

Επίπεδο Δικτύου (Network layer)

2.1 Εισαγωγή (Introduction)

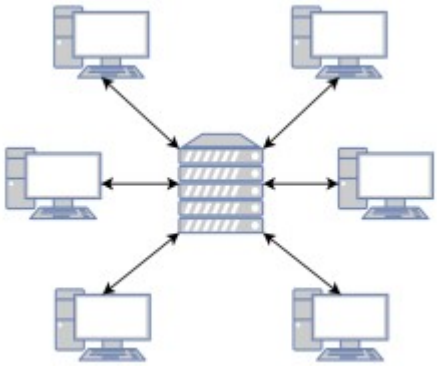
Το ταξίδι της δικτύωσης ξεκινά από την εποχή του Ψυχρού Πολέμου στις αρχές του ’60. Το υπουργείο Εθνικής Άμυνας των Ηνωμένων Πολιτειών ήθελε να αναπτύξει ένα δίκτυο επικοινωνιών και διοίκησης για να ανταπεξέλθει από μία ενδεχόμενη επίθεση με πυρηνικά όπλα από την Σοβιετική Ένωση. Τότε δημιουργήθηκε το APRANET, πρόγονος του Internet. Σκοπός του ήταν η μεταγωγή πακέτων ανάμεσα σε ένα πλήθος ομότιμων υπολογιστών. Έπειτα το 1980 ξεκίνησε η ραγδαία ανάπτυξη του διαδικτύου και η βελτίωση της αρχιτεκτονικής. Αρχικά με την χρήση του μοντέλου client-server και αργότερα με την ομότιμη δικτύωση (P2P). Στις αρχές του 1999 ο Shawn Fanning υλοποίησε μία ιδέα που είχε ώστε αυτός και οι φίλοι του να μπορούν να αναζητήσουν κομμάτια MP3 στο διαδίκτυο. Η Napster ήταν η πρώτη εφαρμογή p2p που μετρούσε από τότε εκατομμύρια χρήστες. Το όνομά της το πήρε από το παρατσούκλι του Fanning λόγο του περίεργου κουρέματός του. Η βασική ιδέα ήταν μία εφαρμογή η οποία θα συνδύαζε μία μηχανή αναζήτησης, ενός προγράμματος ανταλλαγής αρχείων βασισμένης στα πρωτόκολλα διαμοιρασμού αρχείων των Windows και Linux και ενός προγράμματος IRC, ώστε να είναι εφικτή η συζήτηση μεταξύ των χρηστών που ήταν εκείνη την στιγμή OnLine. Έτσι άνοιξε ο δρόμος για την επανάσταση των peer-to-peer εφαρμογών.

2.2 Δικτύωση Πελάτη/Εξυπηρετητή (Client/Server)

Γενικά, το μοντέλο πελάτη-διακομιστή για κατανεμημένα συστήματα, αναφέρεται σε μία βασική αλλαγή στο στυλ των υπολογιστών, την αλλαγή από τα συστήματα που βασίζονται στα μηχανήματα στα συστήματα που βασίζονται στον χρήστη. Ειδικότερα ένα σύστημα πελάτη-διακομιστή είναι ένα σύστημα στο οποίο το δίκτυο ενώνει διάφορους υπολογιστές, ώστε οι πελάτες, να μπορούν να ζητούν υπηρεσίες από έναν διακομιστή, ο οποίος προσφέρει πληροφορίες ή επιπρόσθετη υπολογιστική ισχύ. Με άλλα λόγια στο μοντέλο πελάτη-διακομιστή, ο πελάτης θέτει μία αίτηση (request) και ο διακομιστής επιστρέφει μία ανταπόκριση (response) ή κάνει μία σειρά από ενέργειες. Ο διακομιστής μπορεί να ενεργοποιείται άμεσα για την αίτηση αυτή ή να προσθέτει την αίτηση σε μία ουρά. Η άμεση ενεργοποίηση για την αίτηση μπορεί, για παράδειγμα, να σημαίνει ότι ο διακομιστής υπολογίζει έναν αριθμό και τον επιστρέφει αμέσως στον πελάτη. Η τοποθέτηση της αίτησης σε μία ουρά μπορεί να σημαίνει ότι η αίτηση μπορεί να τεθεί σε αναμονή για να εξυπηρετηθεί από τις υπηρεσίες (services) του διακομιστή. Ένα καλό παράδειγμα για αυτό είναι όταν εκτυπώνουμε ένα κείμενο σε έναν εκτυπωτή δικτύου. Ο διακομιστής τοποθετεί την αίτηση σε μία ουρά μαζί με αιτήσεις εκτυπώσεων και από άλλους πελάτες. Μετά επεξεργάζεται την αίτηση με βάση την σειρά προτεραιότητας, η οποία , σε αυτή την περίπτωση, καθορίζεται από την σειρά με την οποία ο διακομιστής παρέλαβε την αίτηση. Το μοντέλο πελάτη-διακομιστή είναι πολύ σημαντικό διότι επιτυγχάνει τα εξής :

- Αποτελεσματική χρήση της υπολογιστικής ισχύος
- Μείωση του κόστους συντήρησης, δημιουργώντας συστήματα πελάτη-διακομιστή που απαιτούν λιγότερη συντήρηση και κοστίζουν λιγότερο στην αναβάθμιση.
- Αύξηση της παραγωγικότητας, προσφέροντας στους χρήστες πρόσβαση στις αναγκαίες πληροφορίες μέσω σταθερών και εύκολων στη χρήση διασυνδέσεων.
- Αύξηση της ευελιξίας και της δυνατότητας δημιουργίας συστημάτων που υποστηρίζουν πολλά περιβάλλοντα.

Ένα μειονέκτημα που υπάρχει εδώ είναι η επεκτασιμότητα. Οι servers μπορεί να αποτύχουν στις πολλές συνεχόμενες αιτήσεις από τους clients, και οι υπηρεσίες τους (services) μπορεί να γίνουν μη διαθέσιμες. Η μόνη λύση σε αυτό το πρόβλημα είναι να προστεθούν περισσότεροι servers μεταξύ τους έτσι ώστε να αυξηθεί η υπολογιστική ισχύς.



Εικόνα 1. Ένα παράδειγμα δικτύωσης πελάτη/εξυπηρετητή

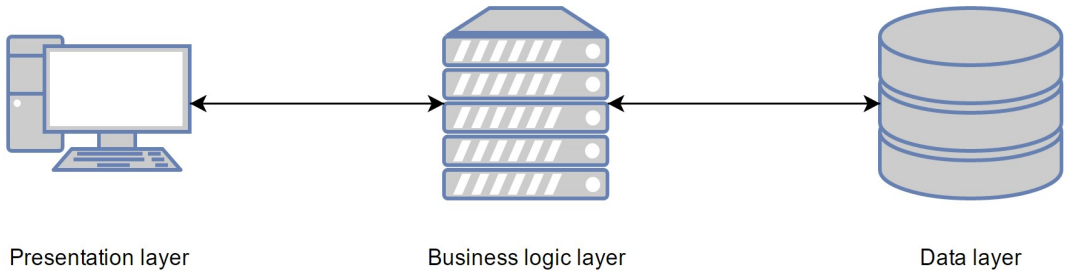
2.2.1 Πολυεπίπεδη Αρχιτεκτονική (n – Tier)

Ένα άλλο κοινό μοντέλο αρχιτεκτονικής για κατανεμημένα συστήματα είναι μια πολύ-επίπεδη αρχιτεκτονική (ή n-tier). Είναι ένας συγκεκριμένος τύπος μοντέλου πελάτη-διακομιστή, όπου διάφορα τμήματα των εφαρμογών χωρίζονται σε πολλά κομμάτια που ονομάζονται στρώματα. Η βασική ιδέα είναι να προσαρμοστεί η εφαρμογή στο σημείο ώστε να γίνει ευκολότερο να αναπτυχθεί, να διατηρηθεί ή ακόμα και να αλλάξει τελείως κάθε στρώμα χωρίς να χρειαστεί η επεξεργασία ολόκληρης της εφαρμογή. Η ύπαρξη χωριστών επιπέδων καθιστά δυνατή την προσαρμογή κάθε επιμέρους βαθμίδας για καλύτερη λειτουργία σε όλο το σύστημα. Είναι δυνατόν, για παράδειγμα, να έχουμε διαφορετικά στρώματα σε ξεχωριστές γεωγραφικές τοποθεσίες και να λειτουργούμε ακόμα ως μία ολόκληρη εφαρμογή.

Ο συνηθέστερος τύπος κατανεμημένων συστημάτων πολλαπλών βαθμίδων είναι τα συστήματα με τρία στρώματα. Αυτά τα συστήματα αποτελούνται από ένα στρώμα **παρουσίασης**, το οποίο είναι υπεύθυνο για άμεση αλληλεπίδραση με το χρήστη, ένα στρώμα **επιχειρησιακής λογικής**, που είναι υπεύθυνο για την επεξεργασία εισερχομένων στοιχείων και την εκτέλεση εργασιών ρουτίνας στα δεδομένα, και ένα επίπεδο **δεδομένων** όπου όλες οι πληροφορίες εφαρμογής αποθηκεύονται σε κάποιο είδος βάσης δεδομένων ή συστήματος αρχείων.

Είναι ευκολότερο να αναπτυχθεί και, αν το υλικό είναι διαθέσιμο, να διανείμει πολλαπλά συστατικά μέσω πολλών μηχανών που παρέχουν αξιοπιστία και αποδοτικότητα στην εφαρμογή. Αυτό είναι το πιο δημοφιλές μοντέλο για τις περισσότερες από τις διαθέσιμες εφαρμογές και υπηρεσίες Web. Για παράδειγμα, ένας χρήστης ζητάει μια ιστοσελίδα σε ένα διακομιστή και παρουσιάζει την σε ένα πρόγραμμα περιήγησης στο Web. Αυτό είναι μέρος του στρώματος παρουσίασης.

Εάν ο χρήστης υποβάλει μια φόρμα, τα δεδομένα θα σταλούν στον διακομιστή και τα δεδομένα θα διαχειριστούν από το επίπεδο επιχειρηματικής λογικής. Το αποτέλεσμα αυτής της επεξεργασίας θα αποθηκευτεί σε μια βάση δεδομένων, μέρος του στρώματος δεδομένων.



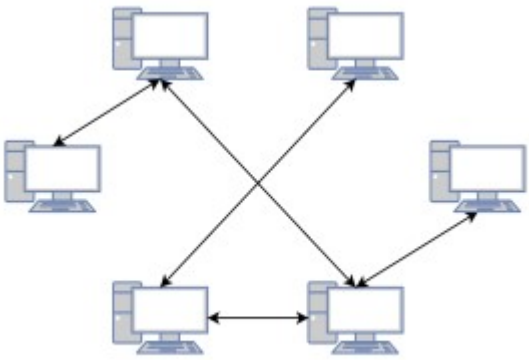
Εικόνα 2. Αρχιτεκτονική πολύ-επίπεδου δικτύου (n – Tier)

2.3 Ομότιμη Δικτύωση (Peer to Peer)

Μία πιο εξελιγμένη μορφή δικτύωσης, που συχνά είναι σε σύγκριση και αντίθεση με την κλασική αρχιτεκτονική client-server, είναι η peer-to-peer ή αλλιώς το ομότιμο δίκτυο, ένα δίκτυο που επιτρέπει μια ή περισσότερες συσκευές να μοιράζονται πόρους ισοδύναμα. Τα συστήματα αυτά έχουν ορισθεί σε πολλές μελέτες. Κάποιοι από τους πιο χαρακτηριστικούς ορισμούς είναι:

- Η κατανεμημένη αρχιτεκτονική δικτύου μπορεί να ονομαστεί σαν ένα ομότιμο δίκτυο, εάν οι συμμετέχοντες μοιράζονται ένα μέρος των ίδιων των πόρων του υλικού τους. Αυτοί οι κοινόχρηστοι πόροι είναι αναγκαίο να παρέχουν υπηρεσία και το περιεχόμενο που προσφέρεται από το δίκτυο.
- Τα p2p συστήματα είναι κατανεμημένα συστήματα που αποτελούνται από διασυνδεδεμένους κόμβους που είναι σε θέση να αυτό-οργανώνονται σε τοπολογίες δικτύων με σκοπό την κατανομή πόρων (όπως το περιεχόμενο, τους κύκλους της CPU, την αποθήκευση και το εύρος ζώνης) και είναι ικανά να προσαρμόζονται στις αποτυχίες και στις μεγάλες μετακινήσεις κόμβων, διατηρώντας αποδεκτή απόδοση και συνδεσιμότητα χωρίς να απαιτείται η διαμεσολάβηση ή η υποστήριξη ενός παγκόσμιου συγκεντρωτικού διακομιστή ή μίας αρχής.

Με απλά λόγια είναι ένα μοντέλο δικτυακής επικοινωνίας στο οποίο δεν υπάρχει κανένας κεντρικός σταθμός και ένας σταθμός μπορεί να γίνει και πελάτης και εξυπηρετητής. Όλοι οι κόμβοι έχουν ίσα δικαιώματα και ευθύνες. Ένα τέτοιο δίκτυο αντιμετωπίζει όλους τους επεξεργαστές εξίσου και χρησιμοποιείται κατά κύριο λόγο σε μικρά δίκτυα με 10 ή λιγότερους χρήστες με άμεση πρόσβαση όπου μπορούν να μοιράζονται περιφερειακές συσκευές χωρίς να περνούν από ξεχωριστό διακομιστή. Αυτό που τα κάνει όμως μοναδικά, δεν είναι μόνο ότι οι κόμβοι είναι ίσοι μεταξύ τους, αλλά ο τύπος και η τοποθεσία των κόμβων. Τα p2p συστήματα έχουν πολλές εφαρμογές, αλλά ο βασικότερος τομέας δραστηριότητάς τους είναι η διανομή περιεχομένου. Άλλοι τομείς είναι τα άμεσα μηνύματα (instant messaging), η εκμετάλλευση υπολογιστικής ισχύος, η τηλεφωνία (VoIP), τα forums, τα Streaming media, οι μηχανές αναζήτησης κ.α. Μερικά γνωστά παραδείγματα είναι το FrostWire, Bit Torrent, Skype.



Εικόνα 3. Ένα παράδειγμα P2P δικτύωσης

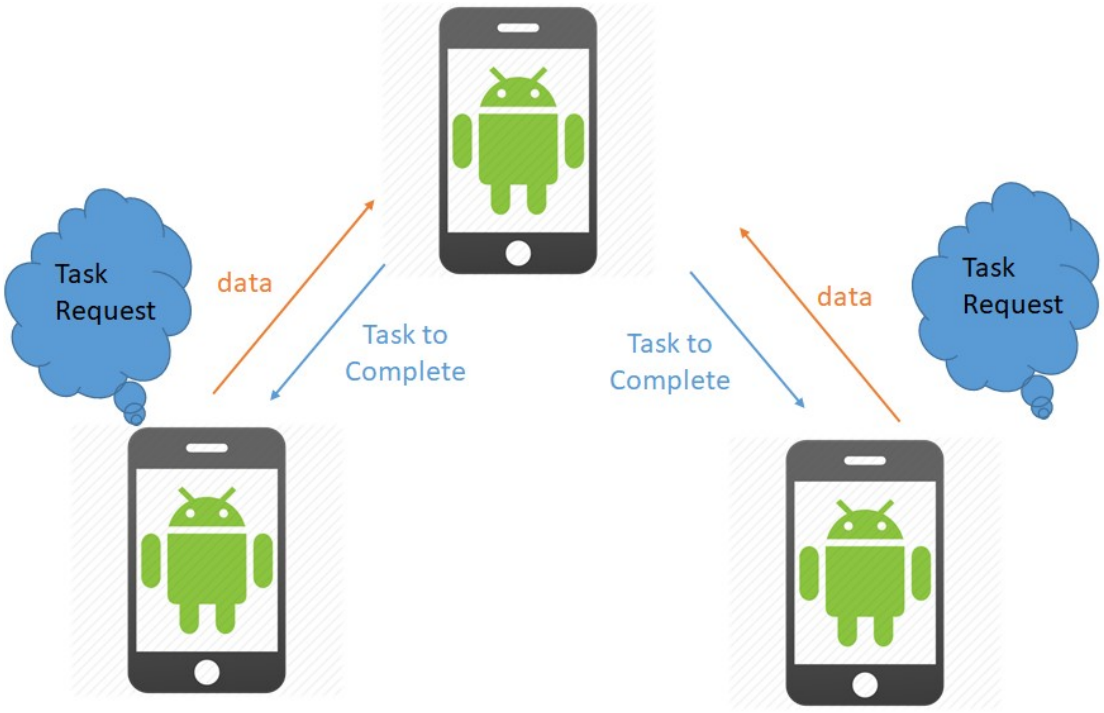
2.3.1 Κατηγοριοποίηση Ομότυμων Δικτύων (P2P)

Οι πολλοί σχεδιασμοί για τα p2p έχουν οδηγήσει σε διάφορες ιδέες για ταξινόμηση. Στην βιβλιογραφία υπάρχει μεγάλο εύρος ως προς την κατηγοριοποίηση και για αυτό το λόγο παρουσιάζονται οι επικρατέστερες προτάσεις. Ανάλογα με την χρονολογία εμφάνισης και τον βαθμό αποκεντροποίησης χωρίζονται σε: Συγκεντρωτικά/Centralized (ή Πρώτης γενιάς), Αποκεντρωτικά/Decentralized/Pure (ή Δεύτερης γενιάς) και μερικής αποκέντρωσης/partial decentralized (ή Τρίτης γενιάς)

- **Συγκεντρωτικά**: Υπάρχει ένας κεντρικός εξυπηρετητής στον οποίο αποθηκεύονται τα περιεχόμενα καταλόγων που μοιράζονται μεταξύ τους οι χρήστες. Οι χρήστες μπορούν να αναζητήσουν στους εξυπηρετητές αυτούς τα αρχεία που ψάχνουν χρησιμοποιώντας ένα κατάλληλο πρόγραμμα πελάτη. Όταν τελικά βρεθεί το στοιχείο ανοίγει μία σύνδεση μεταξύ των δύο χρηστών για να αρχίσει η μεταφορά του αρχείου. Εδώ ανήκει το Napster. Ένα μειονέκτημα σε αυτά τα συστήματα είναι ότι είναι πιο επιρρεπή σε επιθέσεις γιατί υπάρχουν πολλοί ορατοί στόχοι που μπορούν να δεχθούν επίθεση και υπάρχει η εξάρτηση από κάποιους κεντρικούς διακομιστές
- **Αποκεντρωτικά:** Τα δίκτυα αυτά αποτελούν την εξέλιξη των πρώτων εφαρμογών. Ανταποκρίνονται στον ακριβή ορισμό του όρου p2p. Προϋποθέτουν ότι δεν υπάρχει ένας κεντρικός εξυπηρετητής και κάθε σύστημα που συμμετέχει είναι ταυτόχρονα και πελάτης και εξυπηρετητής. Μόλις κάποιος συνδεθεί κάνει γνωστή την παρουσία στο σύνολο υπολογιστών που είναι ήδη συνδεδεμένοι και εκείνοι με την σειρά τους προωθούν την δήλωση παρουσίας σε ένα μεγαλύτερο δίκτυο. Ο χρήστης έχει την δυνατότητα να αναζητήσει αυτό που ψάχνει μεταξύ των συνδεδεμένων υπολογιστών. Σε αυτή την κατηγορία ανήκουν KazaA, Gnutella. Ένα αρνητικό αυτών των συστημάτων είναι η συμφόρηση που προκαλείται από τις περιορισμένες δυνατότητες κάποιων κόμβων.
- **Μερικής αποκέντρωσης:** Είναι δίκτυα αποκεντρωτικού τύπου και η φιλοσοφία τους βασίζεται στον συνεχή διαμοιρασμό των αρχείων και στην κωδικοποίηση τους, ώστε κανείς να μην αποκτήσει κεντρικό έλεγχο των πληροφοριών. Διαθέτουν επίσης χαρακτηριστικά ανωνυμίας. Απαιτούν όμως μία κεντρική οντότητα για την παροχή κάποιων από τις προσφερόμενες υπηρεσίες του δικτύου. Σε αυτά τα συστήματα έχουμε ένα συνδυασμό της p2p αρχιτεκτονικής και αυτής της Client/Server. Τα δίκτυα αυτού του τύπου είναι υπό ανάπτυξη. Στην κατηγορία αυτή ανήκουν: η πλατφόρμα JXTA, FreeNet, I2P, Entropy.

2.4 Το μοντέλο του δικτύου της πτυχιακής

Το μοντέλο του δικτύου που χρησιμοποιήθηκε είναι ένα συγκεντρωτικό ομότιμο δίκτυο (P2P). Στο μοντέλο αυτό, όπως φαίνεται στην εικόνα 4 υπάρχει ένας κεντρικός εξυπηρετητής με τον οποίο συνδέονται οι πελάτες. Ο κάθε πελάτης συνδέεται στον εκάστοτε εξυπηρετητή και ο δεύτερος με την σειρά του αναθέτει στους πελάτες τις διεργασίες που πρόκειται να πραγματοποιήσουν. Οι πελάτες με το που ολοκληρώσουν τις διεργασίες στέλνουν τα αποτελέσματα πίσω στον εξυπηρετητή.



Εικόνα 4. Μοντέλο Υλοποίησης

Επίπεδο σύνδεσης (Link Layer)

3.1 Εισαγωγή

Το επίπεδο σύνδεσης δεδομένων ασχολείται με την τοπική παράδοση πλαισίων μεταξύ συσκευών στο ίδιο τοπικό δίκτυο. Τα πλαίσια του επιπέδου σύνδεσης δεδομένων, όπως καλούνται οι μονάδες δεδομένων αυτού του πρωτοκόλλου, δεν διασχίζουν τα σύνορα ενός τοπικού δικτύου. Η δρομολόγηση ανάμεσα σε δίκτυα και σε παγκόσμιο επίπεδο, είναι δουλειά υψηλότερων επιπέδων, επιτρέποντας στα πρωτόκολλα ζεύξης δεδομένων να εστιάσουν στην τοπική παράδοση, διευθυνσιοδότηση και διαχείριση των μέσων. Από αυτήν την άποψη το επίπεδο ζεύξης δεδομένων, είναι ανάλογο με τον τροχονόμο της γειτονιάς. Πασχίζει να διαιτητεύσει ανάμεσα σε μέλη που αντιδικούν για πρόσβαση στο μέσο. Όταν οι συσκευές προσπαθούν να χρησιμοποιήσουν ταυτόχρονα το μέσο, συμβαίνει σύγκρουση πλαισίων. Τα πρωτόκολλα σύζευξης δεδομένων καθορίζουν πώς οι συσκευές ανιχνεύουν από τέτοιες συγκρούσεις δεδομένων και μπορεί να προμηθεύουν μηχανισμούς για να τις μειώσουν ή να τις προλάβουν.

Στις μέρες μας που η ταχύτητα σύνδεσης και μεταφοράς των δεδομένων είναι από τα πιο σημαντικά θέματα που απασχολούν τους κατασκευαστές κινητών συσκευών, θα αναλύσουμε τους τρείς τρόπους επικοινωνίας μεταξύ τους.

3.2 Τεχνολογίες Επικοινωνίας

3.2.1 Δρομολογητής (Router)

Σε ένα περιβάλλον με υποδομή για ασύρματο σημείο πρόσβασης με τεχνολογία Wi-Fi, η υλοποίηση υπηρεσιών σύνδεσης μπορεί να δημιουργήσει μια σύνδεση μεταξύ συσκευών χρησιμοποιώντας με τη χρήση ασύρματου σημείο πρόσβασης. Να συνδεθούν για παράδειγμα πολλές συσκευές μεταξύ τους με τη χρήση ενός δρομολογητή (Router). Αυτός θα ήταν ένας καλός τρόπος σύνδεσης συσκευών εάν δε υπήρχαν άλλες υποδομές πιο άμεσες και ευέλικτες όπως η τεχνολογία Wi-Fi Direct και το Bluetooth.

3.2.2 Wifi Direct

Το Wi-Fi Direct, που αρχικά ονομάστηκε Wi-Fi P2P, είναι ένα πρότυπο ενεργοποίησης Wi-Fi για εύκολη σύνδεση συσκευών μεταξύ τους χωρίς να απαιτείται ασύρματο σημείο πρόσβασης. Το Wi-Fi Direct επιτρέπει σε δύο συσκευές να δημιουργήσουν απευθείας σύνδεση Wi-Fi χωρίς να απαιτείται ασύρματος δρομολογητής.

Το Wi-Fi γίνεται ένας τρόπος επικοινωνίας ασύρματα, όπως το Bluetooth. Είναι χρήσιμο για όλα, από την περιήγηση στο Internet μέχρι τη μεταφορά αρχείων, και για να επικοινωνούν ταυτόχρονα με μία ή περισσότερες συσκευές σε τυπικές ταχύτητες Wi-Fi. Ένα πλεονέκτημα του Wi-Fi Direct είναι η δυνατότητα σύνδεσης συσκευών, ακόμη και αν προέρχονται από διαφορετικούς κατασκευαστές. Μόνο μία από τις συσκευές Wi-Fi πρέπει να συμμορφώνεται με το Wi-Fi Direct για να δημιουργήσει μια σύνδεση peer-to-peer που μεταφέρει δεδομένα απευθείας μεταξύ τους με άλλες συσκευές με Wi-Fi χωρίς να χρησιμοποιούν ασύρματο σημείο πρόσβασης.
Επί του παρόντος, το Wi-Fi Direct έχει κάποιους περιορισμούς στις συσκευές Android. Σε πρόσφατες δοκιμές, το Wi-Fi Direct μπορούσε να σχηματίσει δίκτυα μόνο με πέντε ή έξι συσκευές κάθε φορά [13]. Παρόλο που, οι εφαρμογές παρουσιάζουν ορισμένους περιορισμούς στη διαμόρφωση του δικτύου, το Wi-Fi -Direct θα μπορούσε να προσφέρει χρήσιμα στοιχεία για τη ρύθμιση της επικοινωνίας μεταξύ των συσκευών.
Το Wi-Fi Direct παρέχει καλύτερες ταχύτητες μεταφοράς από τη Bluetooth (θεωρητικό μέγιστο 250Mbps για Wi-Fi Direct και θεωρητικό μέγιστο 25Mbps), αλλά με μεγαλύτερη κατανάλωση ενέργειας.

3.2.3 BlueTooth

Μια άλλη επιλογή για τη ρύθμιση της επικοινωνίας μεταξύ των συσκευών θα μπορούσε να είναι επιλογή της τεχνολογία Bluetooth.
Το Bluetooth είναι ένα βιομηχανικό πρότυπο για ασύρματα προσωπικά δίκτυα υπολογιστών (Wireless Personal Area Networks, WPAN). Πρόκειται για μια ασύρματη τηλεπικοινωνιακή τεχνολογία μικρών αποστάσεων, η οποία μπορεί να μεταδώσει σήματα μέσω μικροκυμάτων σε ψηφιακές συσκευές. Επομένως το Bluetooth είναι ένα πρωτόκολλο το οποίο παρέχει τυποποιημένη, ασύρματη επικοινωνία ανάμεσα σε PDA, κινητά τηλέφωνα, φορητοί υπολογιστές, προσωπικοί υπολογιστές, εκτυπωτές, καθώς και ψηφιακές φωτογραφικές μηχανές ή ψηφιακές κάμερες, μέσω μιας ασφαλούς, φθηνής και παγκοσμίως διαθέσιμης χωρίς ειδική άδεια ραδιοσυχνότητας μικρής εμβέλειας. Από τεχνικής άποψης το Bluetooth είναι ένα πρωτόκολλο ασύρματης δικτύωσης σε φυσικό επίπεδο, υπό-επίπεδο MAC και, προαιρετικά, υπό-επίπεδο LLC.
Όπως το Wi-Fi Direct, η χρήση Bluetooth δεν απαιτεί την ύπαρξη φυσικής υποδομής (όπως ένα ασύρματο σημείο πρόσβασης).

3.3 Τεχνολογίες Επικοινωνίας που χρησιμοποιήθηκαν στην πτυχιακή

Στην υλοποίηση της εφαρμογής σε αυτή τη διπλωματική εργασία χρησιμοποιήθηκε το πρωτόκολλο σύνδεσης Wi-Fi Direct. Η σύνδεση υλοποιήθηκε με την κλάση της Java WifiP2pManager, η οποία συμπεριλαμβάνεται στο Networking API.

- Σε πρώτη φάση χρησιμοποιείται η κλάση αυτή για να ληφθούν όλες οι διαθέσιμες συσκευές που είναι στο δίκτυο. Όλα τα δεδομένα μπαίνουν σε μια λίστα για να είναι εμφανή και στο UI της συσκευής. Εάν δεν υπάρχουν διαθέσιμες συσκευές στο δίκτυο, γίνονται κάποιες αρχικοποιήσεις.
  Στην πορεία της υλοποίησης διαπιστώθηκε πως στην περίπτωση αποσύνδεσης μιας συσκευής από το δίκτυο wifiP2p, το API της Java κάνει μέχρι και ένα λεπτό να το αντιληφθεί. Για τον λόγο αυτό υλοποιήθηκε και η μέθοδος deletePeersOnDisconnection(), η οποία όταν αντιληφθεί ότι υπάρχουν συσκευές που έχουν βγει από το δίκτυο τις διαγράφει και σαν μεταβλητές από την μνήμη του κινητού.

```
public WifiP2pManager.PeerListListener peerListListener = new WifiP2pManager.PeerListListener() {
    @Override
    public void onPeersAvailable(WifiP2pDeviceList peerList) {
        if (!peerList.getDeviceList().equals(peers)) {
            peers.clear();
            peers.addAll(peerList.getDeviceList());

            deviceNameArray = new String[peerList.getDeviceList().size()];
            Constants.deviceArray = new WifiP2pDevice[peerList.getDeviceList().size()];
            int index = 0;

            for (WifiP2pDevice device : peerList.getDeviceList()) {
                deviceNameArray[index] = device.deviceName + " - " + device.deviceAddress;
            Constants.deviceArray[index] = device;
                index++;
            }
            ArrayAdapter<String> adapter = new ArrayAdapter<>(getApplicationContext(),
android.R.layout.simple_list_item_1, deviceNameArray);
            listView.setAdapter(adapter);
        }

        if (peers.size() == 0) {
            Toast.makeText(getApplicationContext(), "No Device Found", Toast.LENGTH_SHORT).show();
            mapClients.clear();
            readWrites.clear();
            connectionStatus.setText("");
            connectionStatus2.setText("");
            Constants.information = "Connected Devices:";
            Constants.clientCounter = 0;
            Constants.result = 0;
            Constants.hasSendMacAddress = false;
            toolbar.setVisibility(View.GONE);
            return;
        }
        deletePeersOnDisconnection();
    }
};
```

- Στη συνέχεια εφ' όσον έχουν εντοπιστεί όλες οι διαθέσιμες συσκευές, πατώντας επάνω στην συσκευή γίνεται η σύνδεση μεταξύ τους.

```
listView.setOnItemClickListener((parent, view, position, id) -> {
    final WifiP2pDevice device = Constants.deviceArray[position];
    WifiP2pConfig config = new WifiP2pConfig();
    config.deviceAddress = device.deviceAddress;

    mManager.connect(mChannel, config, new WifiP2pManager.ActionListener() {
        @Override
        public void onSuccess() {
            Toast.makeText(getApplicationContext(), "Connected to " + device.deviceName,
Toast.LENGTH_SHORT).show();
        }

        @Override
        public void onFailure(int reason) {
            Toast.makeText(getApplicationContext(), device.deviceName + " disconnected",
Toast.LENGTH_SHORT).show();
        }
    });
});
```

- Τέλος εφόσον έχει υπάρξει επιτυχής σύνδεση, γίνεται ένας έλεγχος για το ποιος είναι ο εξυπηρετητής (Server) και ποιος ο πελάτης (Client), και αντίστοιχα καλούνται οι κλάσεις που περιγράφονται στο 4ο κεφάλαιο και οι οποίες ανοίγουν έναν δίαυλο επικοινωνίας μεταξύ πελάτη και εξυπηρετητή.

```
public WifiP2pManager.ConnectionInfoListener connectionInfoListener = new
WifiP2pManager.ConnectionInfoListener() {
    @SuppressLint("SetTextI18n")
    @Override
    public void onConnectionInfoAvailable(WifiP2pInfo wifiP2pInfo) {

        final InetAddress groupOwnerAddress = wifiP2pInfo.groupOwnerAddress;

        if (wifiP2pInfo.groupFormed && wifiP2pInfo.isGroupOwner) {
            dataMining.setVisibility(View.VISIBLE);
            connectionStatus.setText("I am the Server");
            toolbar.setVisibility(View.VISIBLE);
            new Thread((new Server(MainActivity.this, SERVER_PORT))).start();

        } else if (wifiP2pInfo.groupFormed) {
            connectionStatus.setText("I am the Client");
            toolbar.setVisibility(View.GONE);
            new Thread(new Client(groupOwnerAddress, SERVER_PORT, MainActivity.this)).start();
        }
    }
};
```

Επίπεδο μεταφοράς (Transport Layer)

4.1 Εισαγωγή

Το επίπεδο μεταφοράς (Transport layer) διεκπεραιώνει τη μεταφορά των δεδομένων από χρήστη σε χρήστη, απαλλάσσοντας έτσι τα ανώτερα επίπεδα από κάθε φροντίδα να προσφέρουν αξιόπιστη μεταφορά δεδομένων από το ένα άκρο της επικοινωνίας στο άλλο. Το

επίπεδο μεταφοράς ελέγχει την αξιοπιστία ενός χρησιμοποιούμενου καναλιού με έλεγχο ροής (flow control), κατάτμηση και από-τμηματοποίηση, καθώς και έλεγχο σφαλμάτων (error control). Ορισμένα πρωτόκολλα καταγράφουν καταστάσεις και συνδέσεις, οπότε κρατούν λογαριασμό των πακέτων και στέλνουν πάλι αυτά που δεν παρελήφθησαν σωστά. Τα διάφορα πρωτόκολλα μορφοποιούν διαφορετικά τα εκπεμπόμενα πακέτα πληροφοριών, αλλά τα προς αποστολή δεδομένα παραλαμβάνονται αρχικά από τα ανώτερα επίπεδα.

Το συνηθέστερο παράδειγμα πρωτοκόλλου μεταφοράς είναι το TCP (Transmission Control Protocol, πρωτόκολλο ελέγχου μετάδοσης). Άλλα πρωτόκολλα μεταφοράς είναι τα UDP (User Datagram Protocol, πρωτόκολλο για ασύνδετη αποστολή δεδομένων, SCTP (Stream Control Transmission Protocol, πρωτόκολλο ελέγχου της ροής μετάδοσης), κλπ.

4.2 Υποδοχές (sockets)

Socket είναι το ένα άκρο, από έναν επικοινωνιακό δίαυλο διπλής κατεύθυνσης, μεταξύ δύο προγραμμάτων που εκτελούνται στο δίκτυο. Περιλαμβάνει το πρωτόκολλο, την διεύθυνση (IP) και τον αριθμό θύρας (Port Number) του άκρου.

4.2.1 Ορισμοί

Η έκδοση 4.1cBSD του Unix (1982), για τους υπολογιστές VAX, απ' το πανεπιστήμιο του Berkley, πρώτο-εισήγαγε το socket interface σαν μια μέθοδο επικοινωνίας απομακρυσμένων διεργασιών. Είχαν αρχικά υλοποιηθεί στην γλώσσα C του Unix, αλλά η απήχηση που γνώρισαν, επέβαλε την μεταφορά τους τόσο σε άλλα λειτουργικά συστήματα (π.χ. Winsock library για τα Microsoft Windows), όσο και σε άλλες γλώσσες προγραμματισμού (π.χ. Java).

Κάθε πρόγραμμα διαβάζει από και γράφει σε ένα socket, με τρόπο παρόμοιο της εγγραφής και ανάγνωσης αρχείων του file system. Υπάρχουν δύο είδη sockets:

- Το πρώτο ονομάζεται TCP (Transmission Control Protocol) socket και είναι μια υπηρεσία προσανατολισμένη στην σύνδεση (connection-oriented service). Μπορούμε να το θεωρήσουμε ανάλογο της τηλεφωνικής υπηρεσίας, στην οποία, μετά την εγκαθίδρυση μιας σύνδεσης μεταξύ δύο συνομιλητών, αυτή χρησιμοποιείται μέχρι το πέρας της συζητήσεως τους.
- Το άλλο είδος ονομάζεται UDP (Unreliable Datagram Protocol) socket και είναι μια υπηρεσία χωρίς σύνδεση (connectionless service). Κάτι ανάλογο, σε αυτήν την περίπτωση, είναι το Video Streaming : μπορούμε να στείλουμε πολλά πακέτα στον ίδιο παραλήπτη, αλλά δεν είναι σίγουρο ότι όλα θα ακολουθήσουν την ίδια διαδρομή (σύνδεση) για να φτάσουν στον προορισμό τους.

Μία ακόμη σημαντική διαφορά, μεταξύ των παραπάνω δύο ειδών, είναι ότι τα TCP sockets εξασφαλίζουν μια αξιόπιστη μεταφορά της πληροφορίας : ότι αποστέλλεται από το ένα άκρο είναι σίγουρο ότι θα φτάσει στο άλλο. Στο UDP socket όμως δεν συμβαίνει αυτό. Είναι στην ευθύνη του αποστολέα να ελέγξει ότι αυτό που έστειλε, το έλαβε τελικά ο παραλήπτης και δεν χάθηκε στον δρόμο. Από την άλλη, η σύνδεση με TCP socket απαιτεί την ανταλλαγή τριών "πακέτων χειραψίας" (handshake packets) και είναι πιο χρονοβόρα στην αρχικοποίησή της από την αντίστοιχη με UDP datagrams. Οι προηγούμενες δύο διαφορές καθορίζουν τελικά και την χρήση των δύο αυτών ειδών.

Για την αποφυγή σύγχυσης, να σημειώσουμε ότι ειδικά στην Java, ο όρος Socket χρησιμοποιείται για τα TCP sockets, στην ονοματολογία των κλάσεων και των μεθόδων, ενώ για την δήλωση των UDP sockets, χρησιμοποιείται ο όρος Datagram.

4.3 Προγραμματισμός για TCP – Sockets

Τα sockets είναι μια έννοια που αρχικά πρωτοεμφανίστηκε στα λειτουργικά συστήματα UNIX. Το σύστημα εισόδου εξόδου (input/output) των socket στηρίζεται στις ακόλουθες διεργασίες:

- Open: Άνοιγμα αρχείου
- Read/Write: διάβασμα από ή εγγραφή στο αρχείο

- Close: κλείσιμο αρχείου

Αυτές οι έννοιες επεκτάθηκαν τα τελευταία χρόνια τόσο στην επικοινωνία με χρήση φορητών συσκευών, όσο και στην επικοινωνία μεταξύ διεργασιών (inter-process communication). Τα sockets δηλαδή είναι μέρος του διαύλου επικοινωνίας (communication channel) μεταξύ 2 διεργασιών.

Το στρώμα μεταφοράς αντιπροσωπεύεται στο Internet από τα πρωτόκολλα TCP (connection oriented) και UDP (connectionless, δηλ. με datagrams). Βάση λοιπόν κάθε δικτυακής εφαρμογής είναι το πώς χειριζόμαστε τα δύο αυτά πρωτόκολλα προγραμματιστικά μέσα στα τερματικά συστήματα. Η απάντηση εγκλείεται στη έννοια των sockets. Ένα socket (υποδοχή, 'πρίζα') είναι μία οντότητα (αντικείμενο στην περίπτωση της Java) μέσω της οποίας το δικό μας πρόγραμμα ανταλλάσσει δεδομένα με τον δίαυλο επικοινωνίας. Προγραμματιστικά δεν έχουμε παρά να διαβάζουμε από εκεί την ροή εισόδου στο σύστημά μας και αντίστροφα να γράφουμε εκεί την ροή εξόδου από το σύστημά μας. Ιδιαίτερα στην Java, υπάρχει πρόνοια οι δύο αυτές διαδικασίες να μην διαφέρουν από το γράψιμο / διάβασμα σε / από ένα αρχείο. Στην περίπτωση πολλών συνδέσεων TCP (πάντα αμφίδρομων), πρέπει σε κάθε μία να αντιστοιχείται στο δικό της socket. Για κάθε νέα επικοινωνιακή (TCP) σύνδεση δημιουργείται ένα socket. Η εξαφάνιση της σύνδεσης συνεπάγεται την εξαφάνιση του socket και το αντίστροφο.

Όλα τα sockets τα οποία χειρίζεται μία συγκεκριμένη διαδικασία (η δική μας, ή μια από τις γνωστές εφαρμογές στο διαδίκτυο) έχουν βεβαίως το δικό τους όνομα για να ξεχωρίζονται προγραμματιστικά, αλλά είναι και το καθένα συνδεδεμένο στην 'άκρη' μίας σύνδεσης προς την εφαρμογή που εξυπηρετείται. Η άκρη αυτή αναγνωρίζεται με έναν απλό αριθμό, το port no ή τον αριθμό θύρας.

Αρχίζοντας από το δικό μας socket (συγκεκριμένο αντικείμενο), μπορούμε νοητικά (πρακτικά γίνεται αυτόματα) να βρούμε τον αριθμό της αντίστοιχης σύνδεσης TCP (τον οποίο φέρουν όλα τα πακέτα του πρωτοκόλλου αυτού), να καταλήξομε στο άκρο του συνομιλητή μας και να βρούμε μέσα σε αυτόν το δικό του socket (συγκεκριμένο αντικείμενο), καθώς και το port no που αυτό εξυπηρετεί. Έτσι κατορθώνεται να συνεννοούνται το δικό μας με το δικό του πρόγραμμα.

Μηχανισμός / Κλάση	***Περιγραφή***
Socket	TCP άκρο – πελάτης (Client)
ServerSocket	TCP άκρο - εξυπηρετητής (Server)
DatagramSocket	UDP άκρο (client & server)
DatagramPacket	UDP πακέτο
InetAddress	Διεύθυνση Internet Protocol (IP)
URL	Uniform Resource Locator
URLConnection	Σύνδεση με αντικείμενο του web

Πίνακας 1. Κλάσεις δικτύου

4.4 Το μοντέλο Client-Server

Το ευρύτερα διαδεδομένο μοντέλο ανάπτυξης δικτυακών εφαρμογών είναι το μοντέλο του πελάτη-εξυπηρετητή (client- server). Ο εξυπηρετητής είναι μια διεργασία, η οποία εκτελείται σε έναν υπολογιστή και αναμένει να συνδεθεί σε αυτήν κάποιο πρόγραμμα - ο πελάτης, όπως ονομάζεται -, για να του παράσχει υπηρεσίες. Ένα τυπικό σενάριο που ακολουθείται συνήθως, είναι το εξής :

1. Η διεργασία - *εξυπηρετητής* αρχίζει να εκτελείται σε κάποιον υπολογιστή. Μετά την αρχικοποίησή της, πέφτει σε "λήθαργο", αναμένοντας μία διεργασία - πελάτη να επικοινωνήσει μαζί της και να της ζητήσει κάποια υπηρεσία.

2. Μία διεργασία - *πελάτης* αρχίζει να εκτελείται, είτε στο ίδιο σύστημα, είτε σε κάποιο απομακρυσμένο, το οποίο συνδέεται με τον υπολογιστή στον οποίο “τρέχει” ο εξυπηρετητής μέσω δικτύου. Η διεργασία πελάτης στέλνει μια αίτηση, μέσω του δικτύου, στον εξυπηρετητή, ζητώντας του κάποιου είδους υπηρεσία (π.χ. μεταφορά αρχείου, απομακρυσμένη εκτύπωση, ανάγνωση και αποστολή mail και άλλες). Όταν ο εξυπηρετητής αποδεχτεί την αίτηση (request) του πελάτη, ο πελάτης δημιουργεί μια υποδοχή (Socket) για την επικοινωνία του με τον εξυπηρετητή.
3. Ταυτόχρονα με την εξυπηρέτηση κάποιου πελάτη, ο server έχει την δυνατότητα να δέχεται και αιτήσεις άλλων πελατών προς εξυπηρέτηση. Όταν ο εξυπηρετητής τελειώσει με όλους τους πελάτες, τότε ξαναπέφτει σε “λήθαργο”, περιμένοντας για μια καινούργια αίτηση και η διαδικασία ξαναρχίζει από την αρχή.

Ορίζουμε ως σύνδεση, τον επικοινωνιακό δίαυλο μεταξύ δύο διεργασιών. Την σύνδεση μπορούμε να την θεωρήσουμε ως μία πεντάδα, που περιγράφεται ως εξής :
{ πρωτόκολλο, τοπική-διεύθυνση, τοπική-διεργασία, απομακρυσμένη - διεύθυνση, απομακρυσμένη-διεργασία }

Το πρωτόκολλο αναφέρεται στο σύνολο των κανόνων που διέπουν την επικοινωνία. Η τοπική-διεύθυνση και απομακρυσμένη-διεύθυνση, προσδιορίζουν την ταυτότητα των υπό-δικτύων και των υπολογιστών, στους οποίους εκτελούνται οι επικοινωνούσες διεργασίες. Η τοπική-διεργασία και απομακρυσμένη-διεργασία, προσδιορίζουν την ταυτότητα των διεργασιών που θα επικοινωνούν, καθώς σε έναν υπολογιστή, μπορούν να εκτελούνται περισσότερες της μιας διεργασίες. Κάθε μία από αυτές τις διεργασίες που εκτελούνται στον ίδιο host και που χρειάζονται επικοινωνία μέσω δικτύου, λαμβάνει έναν 16-bit ακέραιο αριθμό, ο οποίος αναπαριστά την θύρα (port number) της διεργασίας και κατ’ επέκταση, της υπηρεσίας.
Ορίζουμε, επίσης ως μισή σύνδεση (half association), είτε το σύνολο { πρωτόκολλο, τοπική-διεύθυνση, τοπική-διεργασία }, είτε το σύνολο { πρωτόκολλο, απομακρυσμένη-διεύθυνση, απομακρυσμένη-διεργασία }. Η μισή σύνδεση ονομάζεται αλλιώς και socket.

4.5 Το μοντέλο δημιουργίας επικοινωνιακού διαύλου της πτυχιακής

Συνέχεια του πρωτοκόλλου σύνδεσης και εφ’ όσον έχουν γίνει οι απαραίτητες ρυθμίσεις από την κάρτα δικτύου, δηλαδή, από την κλάση Wi-Fi Direct η οποία στην περίπτωσή μας είναι η υπεύθυνη για την ρύθμιση του πρωτόκολλου του δικτύου, θα πρέπει να γίνει και η σύνδεση των συσκευών που βρίσκονται στο ίδιο δίκτυο μεταξύ τους. Αυτό επιτυγχάνεται καλώντας τις δύο κλάσεις που κατασκευαστήκαν για αυτό τον σκοπό. Οι κλάσεις αυτές χρησιμοποιούν το Networking API της Java για να ανοίξουν και να κλείσουν ένα κανάλι επικοινωνίας χρησιμοποιώντας το πρωτόκολλο μεταφοράς TCP.

Την κλάση **Server** η οποία δημιουργεί μια υποδοχή(socket), η οποία διανέμεται σε μια συγκεκριμένη θύρα (port), και στη συνέχεια περιμένει να ακούσει κάποια αίτηση σύνδεσης (request) από οποιονδήποτε πελάτη.

```
public class Server implements Runnable {
    private ServerSocket serverSocket;
    private int SERVER_PORT;
    private MainActivity context;

    public Server(MainActivity context, int port) {
        this.SERVER_PORT = port;
        this.context = context;
    }

    @Override
    public void run() {
        try {
            serverSocket = new ServerSocket();
```

```
            serverSocket.setReuseAddress(true);
            serverSocket.bind(new InetSocketAddress(SERVER_PORT));
        } catch (BindException be) {

        } catch (IOException e) {
            e.printStackTrace();
        }
        Socket clientSocket;
        try {
            while (!Thread.currentThread().isInterrupted()) {
                clientSocket = this.serverSocket.accept();
                clientSocket.setTcpNoDelay(true);

                ReadWrite readWrite = new ReadWrite(clientSocket, context);
                new Thread(readWrite).start();
                readWrites.add(readWrite);

                while (true) {
                    if (hasSendMacAddress) {
                        initializeClients(clientSocket);
                        hasSendMacAddress = false;
                        break;
                    }
                }
            }
        } catch (NullPointerException ne) {
            ne.printStackTrace();
        } catch (IOException e) {
            try {
                if (serverSocket != null) {
                    serverSocket.close();
                }
            } catch (IOException io) {
                e.printStackTrace();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void initializeClients(Socket clientSocket) {
        ClientModel clientModel = new ClientModel();
        clientModel.setServerIp(clientSocket);
        clientModel.setClientPort(clientSocket);
        clientModel.setDeviceMacAddress(deviceArray, deviceMacAddresses);
        clientModel.setDeviceName(deviceArray, deviceMacAddresses);
        clientModel.setServerPort(clientSocket);
        clientModel.setClientIp(clientSocket);
        int clientPort = clientSocket.getPort();
        String[] clientIpAddress = clientSocket.getInetAddress().toString().split("/");
        mapClients.put(clientIpAddress[1], clientModel); //putting all the clients into a hash list
        clientCounter++;
        if (clientCounter == 1) {
            information += "\n" + "Connected IP : " + clientIpAddress[1] + ":" + clientPort + "\n";
        } else if (clientCounter > 1) {
            information += "Connected IP : " + clientIpAddress[1] + ":" + clientPort + "\n";
        }
        context.runOnUiThread(() -> updateTextView(information));
    }

    private void updateTextView(String message) {
        TextView txtView = context.findViewById(R.id.connectionStatus2);
        txtView.setVisibility(View.VISIBLE);
```

```
        txtView.setText(message);
    }
}
```

Από την άλλη πλευρά, υπάρχει και η κλάση **Client (πελάτης)** η οποία δημιουργεί μια υποδοχή για την επικοινωνία με τον Server. Ο πελάτης (client) γνωρίζοντας πλέον τόσο την διεύθυνση όσο και την θύρα του Server στον οποίο θέλει να έχει πρόσβαση, στέλνει μια αίτηση (request).

```
public class Client implements Runnable {
    private Socket clientSocket;
    private String hostAdd;
    private int serverPort;
    private MainActivity context;

    public Client(InetAddress hostAddress, int serverPort, MainActivity context) {
        this.context = context;
        hostAdd = hostAddress.getHostAddress();
        clientSocket = new Socket();
        try {
            clientSocket.setTcpNoDelay(true);
        } catch (SocketException e) {
            e.printStackTrace();
        }
        this.serverPort = serverPort;
    }

    @Override
    public void run() {
        try {//connection with the server
            clientSocket.connect(new InetSocketAddress(hostAdd, serverPort), 500);

            ReadWrite readWrite = new ReadWrite(clientSocket, context);
            new Thread(readWrite).start();
            readWrites.add(readWrite); //holding the list of server device to send data

            //sends the device names back to the server after connection established
            for (ReadWrite j : readWrites) {
                j.writeObject(SEND_DEVICE_MAC_TO_SERVER, deviceMacAddresses);
            }
        } catch (IOException e) {
            if (clientSocket != null) {
                try {
                    clientSocket.close();
                } catch (IOException io) {
                    io.printStackTrace();
                }
            }
            e.printStackTrace();
        }
    }
}
```

Η επόμενη κλάση που υλοποιήθηκε είναι η **ReadWrite** με την οποία μπορούμε να γράψουμε και να διαβάσουμε τα δεδομένα μας μέσα στο/από το κανάλι επικοινωνίας. Αυτό επιτυγχάνεται χρησιμοποιώντας το API της Java IO(Input Output). Χρησιμοποιώντας το πακέτο αυτό μπορούμε να γράψουμε και να διαβάσουμε μέσα στα κανάλια δικτύου. Για να μπορέσει να γίνει αυτό θα πρέπει να έχει ανοίξει ένα κανάλι επικοινωνίας όπως περιγράφεται παραπάνω.

```
public class ReadWrite implements Runnable {
    public Socket clientSocket;
```

```
    private ObjectOutputStream objectOutputStream = null;
    private ObjectInputStream objectInputStream = null;
    private MainActivity context;

    ReadWrite(Socket skt, MainActivity context) {
        this.context = context;
        this.clientSocket = skt;
        try {
            clientSocket.setTcpNoDelay(true);
        } catch (SocketException e) {
            e.printStackTrace();
        }
        try {
            objectOutputStream = new ObjectOutputStream(clientSocket.getOutputStream());
            objectInputStream = new ObjectInputStream(clientSocket.getInputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @SuppressWarnings("unchecked")
    @Override
    public void run() {
        try {
            while (!Thread.currentThread().isInterrupted()) {
                byte messageType = objectInputStream.readByte();
                switch (messageType) {
                    case CHAT_MESSAGE: {
                        String message = (String) objectInputStream.readObject();
                        context.runOnUiThread(() -> updateTextView(message));
                        break;
                    }
                    case SEND_DEVICE_MAC_TO_SERVER: {
                        List<String> list = (List<String>) objectInputStream.readObject();
                        deviceMacAddresses.clear();
                        deviceMacAddresses.addAll(list);
                        hasSendMacAddress = true;
                        break;
                    }
                    case SEND_THE_ARRAY_TO_CLIENTS: {
                        String clientIp = clientSocket.getLocalAddress().getHostName();
                        String kindOfAlgorithm = objectInputStream.readUTF();
                        switch (kindOfAlgorithm) {
                            case "Primes": {
                                int arraySize = objectInputStream.readInt();
                                int[] intArray = new int[arraySize];
                                for (int k = 0; k < arraySize; k++) {
                                    intArray[k] = objectInputStream.readInt();
                                }
                                String kindOfCalculation = objectInputStream.readUTF();
                                MessageReceiverFromService newReceiver = new
MessageReceiverFromService(new Message(clientIp, readWrites, context));
                                MappingPrimes.arrayToCalculate = intArray;
                                Intent intent = new Intent(context, MappingPrimes.class);
                                intent.putExtra("receiver", newReceiver);
                                intent.putExtra("kindOfCalculation", kindOfCalculation);
                                context.startService(intent);
                                break;
                            }
                            case "Doubles": {
                                int arraySize = objectInputStream.readInt();
                                double[] doubleArray = new double[arraySize];
                                for (int k = 0; k < arraySize; k++) {
```

```
                        doubleArray[k] = objectInputStream.readDouble();
                    }
                    String kindOfCalculation = objectInputStream.readUTF();
                    MessageReceiverFromService newReceiver = new
MessageReceiverFromService(new Message(clientIp, readWrites, context));
                    MappingSum.arrayToCalculate = doubleArray;
                    Intent intent = new Intent(context, MappingSum.class);
                    intent.putExtra("receiver", newReceiver);
                    intent.putExtra("kindOfCalculation", kindOfCalculation);
                    context.startService(intent);
                    break;
                }
            }
            break;
        }
        case SEND_RESULT_BACK_TO_SERVER: {
         COUNTER_PEERS++;
            String clientIp = objectInputStream.readUTF();
            int some = objectInputStream.readInt();
            switch (some) {
                case PRIME: {
                    long primeResult = objectInputStream.readLong();
                    Map<String, ClientModel> map = finalSetOfMapForPrimes(mapClients, clientIp,
primeResult);
                    if (COUNTER_PEERS == mapClients.size()) {
                        finalPrimeMessage(map);
                        COUNTER_PEERS = 0;
                    }
                    break;
                }
                case SUM: {
                    double doubleResult = objectInputStream.readDouble();
                    Map<String, ClientModel> map = finalSetOfMapForDoubles(mapClients, clientIp,
doubleResult);
                    if (COUNTER_PEERS == mapClients.size()) {
                        finalDoubleMessage(map);
                        COUNTER_PEERS = 0;
                    }
                    break;
                }
            }
            break;
        }
    }
}
} catch (NullPointerException np) {
    np.printStackTrace();
} catch (IOException | ClassNotFoundException e) {
    try {
        if (objectInputStream != null)
            objectInputStream.close();
    } catch (IOException io) {
        io.printStackTrace();
    }
    try {
        if (objectOutputStream != null)
            objectOutputStream.close();
    } catch (IOException io) {
        io.printStackTrace();
    }
    try {
        if (clientSocket != null) {
            clientSocket.close();
```

```
            }
        } catch (IOException io) {
            io.printStackTrace();
        }
    }
}
```

Διεργασίες και Νήματα (Processes and Threads)

5.1 Εισαγωγή

Οι χρήστες υπολογιστών θεωρούμε δεδομένο ότι τα συστήματά μας μπορούν να κάνουν πολλά πράγματα ταυτόχρονα. Για παράδειγμα να μπορούμε χρησιμοποιούμε τον επεξεργαστή κειμένου ενώ ταυτόχρονα άλλες εφαρμογές "κατεβάζουν" αρχεία, εκτυπώνουν, παίζουν μουσική κλπ. Ακόμη και από μία μόνο εφαρμογή προσδοκούμε να κάνει πολλά πράγματα ταυτόχρονα. Από έναν επεξεργαστή κειμένου προσδοκούμε να μπορεί να ανταποκριθεί στην είσοδο από το πληκτρολόγιο ή το ποντίκι ακόμη και αν εκείνη την ώρα ανανεώνει την οθόνη ή μορφοποιεί το κείμενο. Ταυτόχρονος προγραμματισμός (concurrent programming) ονομάζεται η μεθοδολογία προγραμματισμού που μας επιτρέπει να υλοποιήσουμε την ταυτόχρονη εκτέλεση διαφόρων εργασιών στην ίδια υπολογιστική συσκευή.

Η Java είναι σχεδιασμένη να υποστηρίζει ταυτόχρονο προγραμματισμό μέσω της ίδιας της γλώσσας και των βιβλιοθηκών της. Από την έκδοση 5 το API της Java διαθέτει υψηλού επιπέδου υποστήριξη για ταυτόχρονο προγραμματισμό.

Εδώ θα δούμε τις βασικές δυνατότητες που διαθέτει η Java για να υποστηρίξει ταυτόχρονο προγραμματισμό.

5.2 Ορισμοί και βασικές έννοιες

Ένα **πρόγραμμα (Program)** με απλούς όρους μπορεί να ερμηνευτεί ως οποιοδήποτε εκτελέσιμο αρχείο. Βασικά, περιέχει ένα συγκεκριμένο σύνολο οδηγιών που γράφονται με σκοπό τη διεξαγωγή συγκεκριμένης ενέργειας. Βρίσκεται στη μνήμη και είναι μια παθητική οντότητα που δεν πάει μακριά όταν το σύστημα κάνει επανεκκίνηση.

Οποιαδήποτε κατάσταση λειτουργίας ενός προγράμματος καλείται ως **διεργασία (Process)** ή μπορεί επίσης να ερμηνευτεί ως ένα πρόγραμμα υπό εκτέλεση. 1 πρόγραμμα μπορεί να έχει Ν διεργασίες. Η διαδικασία βρίσκεται στην κύρια μνήμη και συνεπώς εξαφανίζεται κάθε φορά που γίνεται επανεκκίνηση του μηχανήματος. Πολλές διεργασίες μπορούν να εκτελούνται παράλληλα σε ένα σύστημα πολλαπλών επεξεργαστών.

Ένα **νήμα (Thread)** περιγράφεται συνήθως ως μια ελαφριά διεργασία. 1 διεργασία μπορεί να έχει Ν νήματα. Όλα τα θέματα που σχετίζονται με μια κοινή διεργασία μοιράζονται την ίδια μνήμη με τη διεργασία αυτή. Η ουσιαστική διαφορά μεταξύ ενός νήματος και μιας διεργασίας είναι το έργο που κάθε ένας χρησιμοποιείται για να ολοκληρώσει. Τα νήματα χρησιμοποιούνται για μικρές και συμπαγείς εργασίες, ενώ οι διεργασίες χρησιμοποιούνται για πιο βαριές εργασίες.

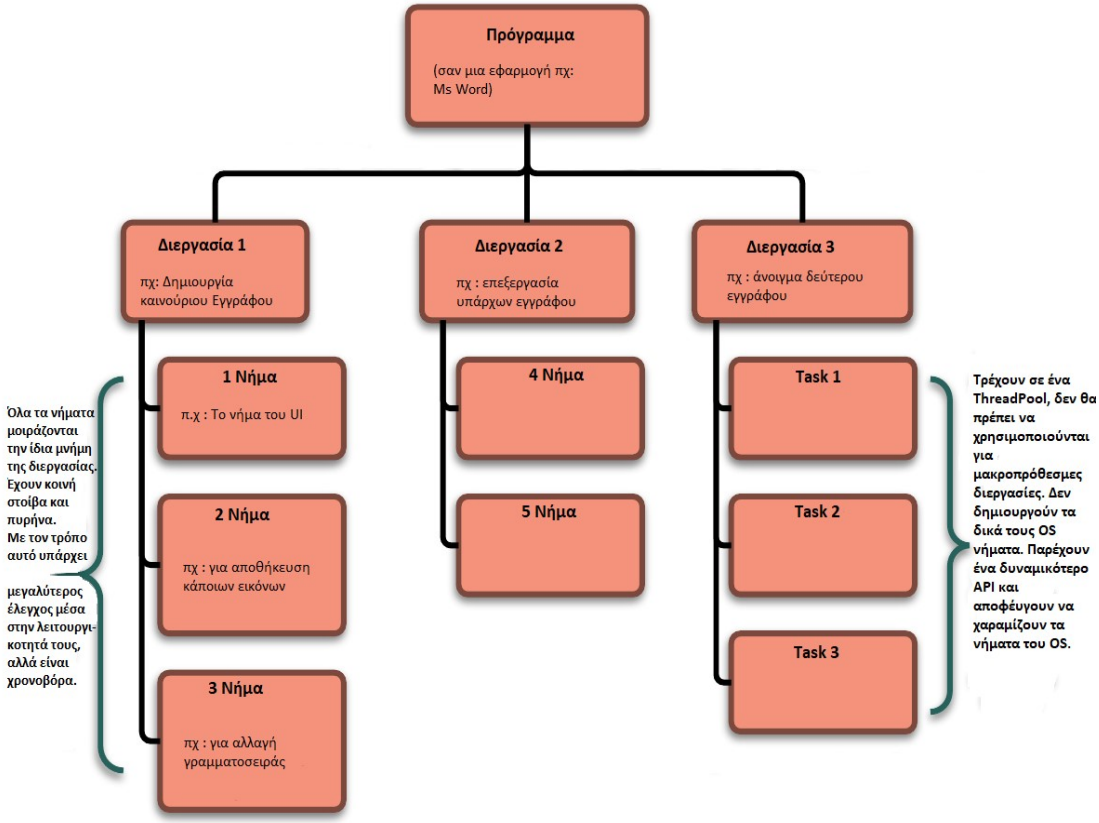
Μια μεγάλη διαφορά μεταξύ ενός νήματος και μιας διεργασίας είναι ότι τα νήματα στην ίδια διεργασία καταναλώνουν τον ίδιο χώρο διευθύνσεων, ενώ διαφορετικές διεργασίες δεν το κάνουν. Αυτό επιτρέπει στα νήματα να διαβάζουν και να γράφουν στις κοινές δομές δεδομένων και μεταβλητών και επίσης αυξάνει την ευκολία επικοινωνίας μεταξύ των νημάτων. Η επικοινωνία μεταξύ δύο ή περισσοτέρων διεργασιών - επίσης γνωστή ως Inter-Process Communication (IPC) - είναι αρκετά δύσκολη και χρησιμοποιεί αρκετούς πόρους.

Οι **εργασίες** (**Tasks)** (Εικόνα 5). είναι πολύ παρόμοιες με τα νήματα, η διαφορά είναι ότι γενικά δεν αλληλοεπιδρούν άμεσα με το λειτουργικό σύστημα. Όπως ένα Thread Pool, μια εργασία δεν

δημιουργεί το δικό της νήμα στο λειτουργικό σύστημα (OS). Μια εργασία μπορεί να έχει ένα ή περισσότερα νήματα εσωτερικά.

- Μια *εργασία* είναι απλούστερη στη χρήση και πιο αποτελεσματική στη δημιουργία των δικών μας νημάτων. Αλλά μερικές φορές, χρειαζόμαστε περισσότερο έλεγχο από αυτά που προσφέρει η εργασία. Σε αυτές τις περιπτώσεις, είναι πιο λογικό να χρησιμοποιήσουμε άμεσα ένα νήμα.

Το τελικό συμπέρασμα είναι ότι η εργασία είναι σχεδόν πάντα η καλύτερη επιλογή. παρέχει ένα πολύ ισχυρότερο API και αποφεύγει τη σπατάλη των νημάτων του λειτουργικού συστήματος. Οι μόνοι λόγοι για να δημιουργήσουμε ρητά τα δικά μας νήματα στον σύγχρονο κώδικα είναι να ρυθμίσουμε τις επιλογές ανά-νήμα ή να διατηρήσουμε ένα μόνιμο νήμα που πρέπει να διατηρήσει τη δική του ταυτότητα.



Εικόνα 5. Διάγραμμα διεργασιών - νημάτων

5.3 Διεργασίες (processes) και νήματα (thread)

Διεργασία (Process) είναι μια συγκεκριμένη εκτέλεση κάποιου προγράμματος για λογαριασμό κάποιου χρήστη. Στην πληροφορική, ένα **νήμα** εκτέλεσης (Thread) είναι η μικρότερη ακολουθία προγραμματισμένων εντολών που μπορεί να υποστεί διαχείριση ανεξάρτητα από το λειτουργικό σύστημα. Ένα νήμα είναι μια ελαφριά διεργασία. Η υλοποίηση των νημάτων και των διεργασιών διαφέρει από το ένα λειτουργικό σύστημα στο άλλο. Στις περισσότερες όμως περιπτώσεις ένα νήμα εμπεριέχεται σε μια διεργασία. Μπορούν να υπάρχουν πολλαπλά νήματα μέσα στην ίδια διεργασία, τα οποία μπορούν να μοιράζονται πόρους από το σύστημα, όπως μνήμη.

Διαφορετικές διεργασίες δεν μπορούν να μοιράζονται τους ίδιους πόρους. Συγκεκριμένα, τα νήματα μιας διεργασίας περιέχουν τις εντολές προς την εκτελούμενη διεργασία (δηλαδή τον κώδικα της) και το εννοιολογικό της πλαίσιο (οι τιμές των μεταβλητών της σε οποιαδήποτε χρονική στιγμή.

Σε έναν απλό επεξεργαστή, η **πολυνημάτωση** (multithreading) πραγματοποιείται με τη μέθοδο της πολυπλεξίας με διαίρεση (καταμερισμό) χρόνου (όπως στην πολυεπεξεργασία): ο επεξεργαστής μεταπηδάει μεταξύ των διάφορων νημάτων. Αυτή η εναλλαγή μεταξύ των διεργασιών ονομάζεται Μεταγωγή περιβάλλοντος (context switch), και συμβαίνει σε πολύ τακτά χρονικά διαστήματα, τέτοια έτσι ώστε ο χρήστης έχει την εντύπωση ότι τα νήματα εκτελούνται την ίδια στιγμή. Μόνο σε έναν επεξεργαστή με πολλούς επεξεργαστικούς πυρήνες, τα νήματα εκτελούνται πραγματικά ταυτόχρονα και κάθε πυρήνας εκτελεί ένα συγκεκριμένο νήμα ή εργασία.

Πολλά σύγχρονα λειτουργικά συστήματα υποστηρίζουν τόσο νήματα καταμερισμού χρόνου, όσο και νήματα ταυτόχρονης πολυεπεξεργασίας, στον χρονοδρομολογητή τους. Ο πυρήνας ενός λειτουργικού συστήματος επιτρέπει στους προγραμματιστές να χειρίζονται τα νήματα μέσω της διεπαφής κλήσεων συστήματος. Ορισμένες υλοποιήσεις λέγονται νήμα πυρήνα, ενώ ελαφρά διεργασία είναι ο ειδικός τύπος νήματος πυρήνα που μοιράζεται την ίδια κατάσταση και πληροφορίες. Στα συστήματα Unix υπάρχει το POSIX στάνταρντ για δημιουργία POSIX νημάτων.

Ένα νήμα διαφέρει από μια διεργασία ενός πολυεπεξεργαστικού λειτουργικού συστήματος στα εξής:

- οι διεργασίες είναι τυπικώς ανεξάρτητες, ενώ τα νήματα αποτελούν υποσύνολα μιας διεργασίας.
- οι διεργασίες περιέχουν σημαντικά περισσότερες πληροφορίες κατάστασης από τα νήματα, ενώ πολλαπλά νήματα μιας διεργασίας μοιράζονται την κατάσταση της διεργασίας όπως επίσης μνήμη και άλλους πόρους.
- οι διεργασίες έχουν ξεχωριστούς χώρους διευθυνσιοδότησης, ενώ τα νήματα μοιράζονται το σύνολο του χώρου διευθύνσεων που τους παραχωρείται.
- η εναλλαγή ανάμεσα στα νήματα μιας διεργασίας είναι πολύ γρηγορότερη από την εναλλαγή ανάμεσα σε διαφορετικές διεργασίες.

Ένα πρόγραμμα που μεταγλωττίζεται δημιουργείται ένα εκτελέσιμο αρχείο. Αυτό όταν εκτελείται δημιουργείται μια νέα διεργασία μέσα στην οποία τρέχει ο εκτελέσιμος κώδικας. Το λειτουργικό σύστημα φροντίζει να δημιουργήσει χώρο μνήμης-διευθύνσεων για την διεργασία. Στο Unix κάθε εκτέλεση του εκτελέσιμου αρχείου δημιουργεί μια νέα ξεχωριστή διεργασία η οποία τρέχει παράλληλα (ή ψευδοπαράλληλα με την τεχνική διαίρεσης χρόνου - όταν έχουμε ένα επεξεργαστή). Το νήμα είναι μια πολύ πιο "ελαφριά" μορφή διεργασίας όπου η δημιουργία είναι εκατοντάδες φορές πιο γρήγορη από την δημιουργία μιας διεργασίας. Τα νήματα τρέχουν παράλληλα όπως και οι διεργασίες. Η ιδέα είναι ότι δημιουργούμε μια διεργασία και μέσα από αυτή δημιουργούμε νήματα τα οποία έχουν πρόσβαση στο χώρο μνήμης της διεργασίας.

5.4 Πολυνημάτωση και Πολυδιεργασία (Multithreading and Multitasking)

Η πολυνημάτωση (multithreading) αποτελεί ένα ευρέως διαδεδομένο μοντέλο προγραμματισμού και εκτέλεσης διεργασιών το οποίο επιτρέπει την ύπαρξη πολλών νημάτων μέσα στα πλαίσια μιας και μόνο διεργασίας. Τα νήματα αυτά μοιράζονται τους πόρους της διεργασίας και μπορούν να εκτελούνται ανεξάρτητα. Η **πολυνημάτωση** δεν πρέπει να μπερδεύεται με την **πολυδιεργασία** (multitasking) όπου έχει να κάνει με την εκτέλεση

προγραμμάτων δηλαδή διεργασιών παράλληλα σε ένα υπολογιστικό σύστημα με πολλούς επεξεργαστές. Το πλεονέκτημα ενός πολυνηματικού προγράμματος είναι ότι του επιτρέπει να εκτελείται γρηγορότερα σε υπολογιστικά συστήματα που έχουν πολλούς επεξεργαστές, επεξεργαστές με πολλούς πυρήνες, ή κατά μήκος μιας συστοιχίας υπολογιστών. Για να μπορούν να χειραγωγηθούν σωστά τα δεδομένα, τα νήματα θα πρέπει ορισμένες φορές να συγκεντρωθούν σε ένα ορισμένο χρονικό διάστημα έτσι ώστε να επεξεργασθούν τα δεδομένα στη σωστή σειρά. Ένα ακόμα πλεονέκτημα της πολυδιεργασίας (και κατ' επέκταση της πολυνημάτωσης) ακόμα και στους απλούς επεξεργαστές (με έναν πυρήνα επεξεργασίας) είναι η δυνατότητα για μια εφαρμογή να ανταποκρίνεται άμεσα. Έχοντας ένα πρόγραμμα που εκτελείται μόνο σε ένα νήμα, αυτό θα φαίνεται ότι κολλάει ή ότι παγώνει, στις περιπτώσεις που εκτελείται κάποια μεγάλη διεργασία που απαιτεί πολύ χρόνο. Αντίθετα σε ένα πολυνηματικό σύστημα, οι χρονοβόρες διεργασίες μπορούν να εκτελούνται παράλληλα με άλλα νήματα που φέρουν άλλες εντολές για το ίδιο πρόγραμμα, καθιστώντας το άμεσα ανταποκρίσιμο.

Τα λειτουργικά συστήματα προγραμματίζουν τα νήματα με έναν από τους δύο παρακάτω τρόπους:

- Πολυδιεργασία προτίμησης (Preemptive Multitasking): γενικώς θεωρείται η καλύτερη προσέγγιση. Σύμφωνα με αυτή, το λειτουργικό σύστημα καθορίζει πότε θα γίνεται μια εναλλαγή από το ένα νήμα στο άλλο. Μειονέκτημα αυτής της μεθόδου είναι ότι το σύστημα μπορεί να πραγματοποιήσει μια εναλλαγή σε ακατάλληλη στιγμή με αρνητικές επιπτώσεις για τις εφαρμογές που εκτελούνται.
- Πολυνημάτωση συνεργασίας (Cooperative Multithreading): Σύμφωνα με αυτή τη προσέγγιση, τα ίδια τα νήματα αναλαμβάνουν τον έλεγχο. Έτσι όταν ένα νήμα τελειώσει, παραδίδει τους πόρους εκτέλεσης σε άλλο νήμα. Μειονέκτημα αυτής της προσέγγισης είναι ότι ένα νήμα μπορεί να περιμένει επ' αόριστόν να ελευθερωθούν πόροι συστήματος.

Μέχρι και τα τέλη της δεκαετίας του 1990, οι πιο δημοφιλείς επεξεργαστές των προσωπικών υπολογιστών δεν υποστήριζαν πολλαπλά νήματα εκτέλεσης - κάποια λειτουργικά συστήματα υποστήριζαν τη χρήση νημάτων στο λογισμικό, που δεν εκτελούνταν πραγματικά παράλληλα αλλά από τον χρονοπρογραμματιστή του συστήματος, και είχαν εφαρμογές κυρίως στην επεξεργασία δεδομένων. Ωστόσο, επεξεργαστές σε άλλα ενσωματωμένα συστήματα που είχαν υψηλές απαιτήσεις για ανταπόκριση σε πραγματικό χρόνο (όπως π.χ. φανάρια σηματοδότησης, ελεγκτές εργοστασιακών μηχανών κ.α.) υποστήριζαν μια μορφή πολυνημάτωσης. Με το πέρας της δεκαετίας του 1990, η ιδέα της ταυτόχρονης εκτέλεσης εντολών σε πολλαπλά νήματα, γνωστή ως ταυτόχρονη πολυνημάτωση, υλοποιήθηκε και στους προσωπικούς υπολογιστές με το λανσάρισμα των επεξεργαστών Pentium 4 της Intel, με την ονομασία Υπερνημάτωση. Η απόγονη αρχιτεκτονική πού ακολούθησε τους Pentium 4 (εμπορικά προϊόντα Core και Core 2) δεν την ενσωμάτωνε. Από το 2008 και με την έλευση της αρχιτεκτονικής Nehalem και μέχρι σήμερα, η υπερνημάτωση επανεμφανίστηκε με βελτιώσεις στις σειρές επεξεργαστών Intel Core i5 και Core i7.

5.5 Συγχρονισμένες (Concurrent) & Παράλληλες (Parallel) εργασίες (tasks)

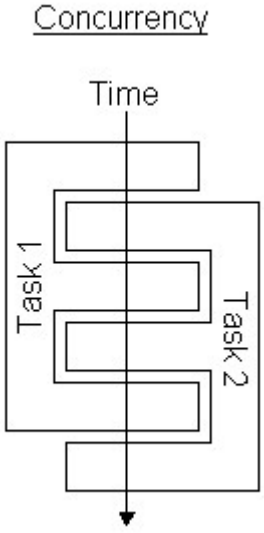
Η παράλληλη υλοποίηση και η ταυτόχρονη υλοποίηση πολλών εργασιών δεν είναι καθόλου το ίδιο πράγμα στον προγραμματισμό.

Συγχρονισμός (Concurrency) σημαίνει ότι πολλαπλές εργασίες ξεκινάνε εκτελούνται και ολοκληρώνονται σε επικαλυπτόμενες χρονικές περιόδους χωρίς να έχουν μια συγκεκριμένη σειρά. Είναι η συγκρότηση από ξεχωριστές εκτελέσιμες διεργασίες.

Παραλληλισμός (Parallelism) είναι όταν πολλαπλές εργασίες ή πολλαπλά νήματα μιας διεργασίας τρέχουν ακριβώς την ίδια στιγμή. Για παράδειγμα, στην περίπτωση που έχουμε έναν επεξεργαστή με πολλούς πυρήνες. Είναι η ταυτόχρονη εκτέλεση διεργασιών.

5.5.1 Συγχρονισμός (Concurrency)

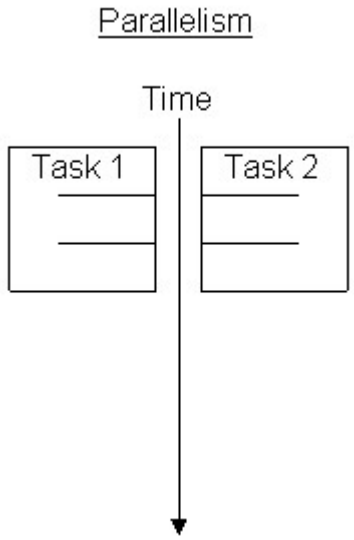
Συγχρονισμός σημαίνει ότι μια εφαρμογή τρέχει με περισσότερες από μια εργασίες στην ίδια περίοδο του χρόνου (συγχρονισμένα). Όταν ο υπολογιστής έχει μόνο έναν επεξεργαστή η εφαρμογή δεν θα τρέξει περισσότερες από μία εργασίες ακριβώς στον ίδιο χρόνο, αλλά θα εκτελεστεί πρώτα η μια εργασία θα τερματίσει και στην συνέχεια θα εκτελεστεί η δεύτερη εργασία και θα τερματίσει με τη σειρά της. Ο επεξεργαστής δηλαδή μοιράζει τον χρόνο (time-slicing) σε περίπου ίσα μέρη όπου η κάθε εργασία τρέχει τερματίζει και περιμένει να εκτελεστεί η επόμενη. Έτσι φαίνεται σαν τελικό αποτέλεσμα ότι οι εργασίες εκτελούνται στον ίδιο χρόνο ακριβώς αλλά στην ουσία δεν συμβαίνει αυτό.



Εικόνα 6. Συγχρονισμένες εργασίες

5.5.2 Παραλληλότητα (Parallelism)

Παραλληλότητα σημαίνει ότι μια εφαρμογή χωρίζει σε μικρότερα κομμάτια τις εργασίες τις οι οποίες μπορούν να τρέξουν παράλληλα. Για παράδειγμα, στην περίπτωση που έχουμε πολλούς επεξεργαστές ή επεξεργαστές με πολλούς πυρήνες (Multithreaded), πολλές εργασίες μπορούν να εκτελεστούν ακριβώς την ίδια ώρα.



Εικόνα 7. Παράλληλες εργασίες

5.6 Διεργασίες και νήματα στην πτυχιακή

Αρχικά όταν δημιουργούμε ένα καινούριο κανάλι επικοινωνίας μεταξύ πελάτη-εξυπηρετητή από το κυρίως νήμα (Main Thread), θα πρέπει για κάθε περίπτωση αυτό να τρέξει υποχρεωτικά σε ένα ξεχωριστό thread.

Έτσι στη MainActivity.class δημιουργούμε 1 thread για να ανοίξουμε το κανάλι επικοινωνίας

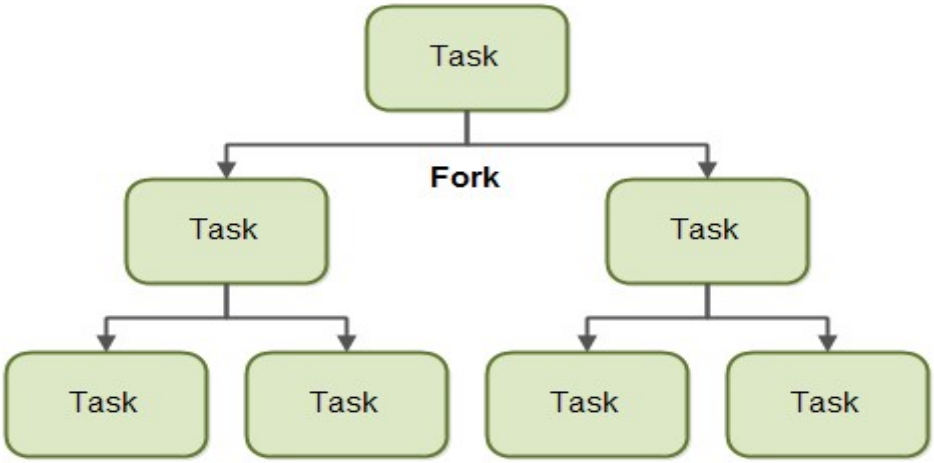
```
new Thread((new Server(MainActivity.this, SERVER_PORT))).start();
new Thread(new Client(groupOwnerAddress, SERVER_PORT, MainActivity.this)).start();
```

5.6.1 Χρήση του πακέτου της Java ForkJoinPool

Άλλη μια χρήση των νημάτων στην πτυχιακή είναι η κλήση της κλάσης ForkJoinPool της Java, η οποία βρίσκεται στο πακέτο της java.util.concurrent. Η κλάση ForkJoinPool κάνει ευκολότερη τη διαδικασία για τις εργασίες(Tasks) να διαιρέσουν τη δουλειά τους σε μικρότερες εργασίες(subTasks), οι οποίες υποβάλλονται στη ForkJoinPool επίσης. Οι εργασίες συνεχίζουν να διαιρούν την δουλειά τους σε μικρότερες εργασίες όσο αυτό χρειάζεται.

5.6.2 Διακλάδωση (Fork)

Μια εργασία (Task) που χρησιμοποιεί την αρχή της διακλάδωσης (fork) και της ένωσης (join) μπορεί να χωριστεί σε μικρότερες εργασίες(subtasks) που μπορούν να εκτελεστούν ταυτόχρονα (concurrently). Αυτό απεικονίζεται στο παρακάτω διάγραμμα:

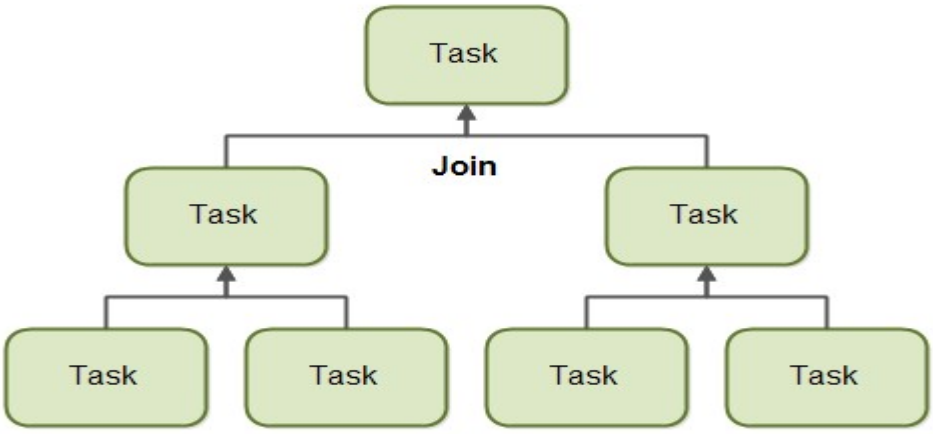


Εικόνα 6. Διάγραμμα διακλάδωσης

Διαιρώντας μια εργασία τον εαυτό της σε υπό – εργασίες, κάθε υπό-εργασία μπορεί να εκτελεστεί παράλληλα και σε διαφορετικούς επεξεργαστές (εάν υπάρχουν), ή σε διαφορετικά νήματα στον ίδιο επεξεργαστή.
Μια εργασία διαιρεί τον εαυτό της, εάν και εφ' όσον η δουλειά που της ανατέθηκε να κάνει είναι τόσο μεγάλη που δεν μπορεί να ανταπεξέλθει μόνη της σε αυτό. Ωστόσο υπάρχει μια επιβράδυνση χωρίζοντας μια εργασία σε επιμέρους εργασίες εάν η αρχική εργασία που έχει ανατεθεί είναι μικρού φόρτου. Αυτό συμβαίνει γιατί η διαίρεση της εργασίας θα πάρει μεγαλύτερο χρόνο από το να εκτελεστεί η ίδια σειριακά.

5.6.3 Συνένωση (Join)

Όταν μια εργασία έχει χωριστεί σε μικρότερα κομμάτια, περιμένει τις υπό-εργασίες να τελειώσουν να εκτελούνται. Όταν και αυτές με την σειρά τους σταματήσουν να εκτελούνται, η εργασία συνενώνει (αθροίζει) όλα τα αποτελέσματα σε ένα τελικό αποτέλεσμα. Αυτό φαίνεται στο παρακάτω διάγραμμα.



Εικόνα 7. Διάγραμμα συνένωσης

Βεβαίως, μπορεί να μην επιστρέφουν αποτέλεσμα όλοι οι τύποι εργασιών. Εάν μια εργασία δεν επιστρέφει αποτέλεσμα, τότε περιμένει τις υπό μέρους εργασίες να ολοκληρωθούν. Κανένα αποτέλεσμα δεν επιστρέφεται.

5.6.4 Δημιουργία της ForkJoinPool

Εδώ δημιουργείται ένα αντικείμενο τύπου ForkJoinPool

```
private static ForkJoinPool FORK_JOIN_POOL = new
ForkJoinPool(Runtime.getRuntime().availableProcessors());
```

και δημιουργούμε ένα αντικείμενο τύπου ForkJoinPool με παραμέτρους τον αριθμό των διαθέσιμων πύρινων του επεξεργαστή που χρειάζεται να δουλέψουν.

- Περίπτωση ταυτόχρονης επεξεργασίας πρώτων αριθμών

Στη συνέχεια ορίζεται η κλάση ConcurrentArrayCountPrimes η οποία κάνει extend την υπό κλάση RecursiveTask. Η κλάση αυτή κάνει όλη τη δουλειά και διαιρεί την αρχική εργασία σε υπό-μέρους εργασίες οι οποίες εκτελούνται ταυτόχρονα (Concurrent) μέχρι να ληφθεί το επιθυμητό αποτέλεσμα.

```
public class ConcurrentArrayCountPrimes extends RecursiveTask<Long> {

    private static final int PARALLEL_CUTOFF = 1000;
    private int[] numbers;
    private int lowerIndex, upperIndex;

    public ConcurrentArrayCountPrimes(int[] numbers, int lowerIndex, int upperIndex) {
        this.numbers = numbers;
        this.lowerIndex = lowerIndex;
        this.upperIndex = upperIndex;
    }


    @Override
    protected Long compute() {
        int range = upperIndex - lowerIndex;
        if (range <= PARALLEL_CUTOFF) {
            return (PrimeUtils.countArrayPrimes(numbers, lowerIndex, upperIndex));
        } else {
            List<ConcurrentArrayCountPrimes> subTasks = new ArrayList<>(createSubTasks());
            for (ConcurrentArrayCountPrimes subTask : subTasks) {
                subTask.fork();
            }
            long result = 0;
            for (ConcurrentArrayCountPrimes subTask : subTasks) {
                result += subTask.join();
            }

            return result;
        }
    }

    private List<ConcurrentArrayCountPrimes> createSubTasks() {
        int range = upperIndex - lowerIndex;
        int middleIndex = lowerIndex + (range / 2);
```

```
        List<ConcurrentArrayCountPrimes> subTasks = new ArrayList<>();

        ConcurrentArrayCountPrimes subTask1 = new ConcurrentArrayCountPrimes(numbers, lowerIndex,
middleIndex);
        ConcurrentArrayCountPrimes subTask2 = new ConcurrentArrayCountPrimes(numbers, middleIndex
+ 1, upperIndex);

        subTasks.add(subTask1);
        subTasks.add(subTask2);

        return subTasks;
    }
}
```

Στην περίπτωσή μας, παίρνει ένα πίνακα ακεραίων N στοιχείων, διαιρεί τον πίνακα αυτόν κάθε φορά στη μέση(fork), έτσι δημιουργούνται παράλληλες εργασίες, μέχρις ότου τα στοιχεία του κάθε πίνακα που θα προκύψει να γίνουν μικρότερα από 1000. Όταν θα συμβεί αυτό θα αρχίσει να υπολογίζει σειριακά τους πρώτους αριθμούς(prime numbers) που θα υπάρχουν σε κάθε πίνακα που δημιούργησε, και στο τέλος θα αθροίσει το αποτέλεσμα (join) που θα είναι ένας μεγάλος ακέραιος αριθμός που θα επιστραφεί σαν αποτέλεσμα. Όλες οι διεργασίες εκτελούνται ταυτόχρονα.
Η υλοποίηση αυτή μοιάζει πολύ με τον αλγόριθμο Merge-Sort ο οποίος βασίζεται στην τεχνική διαίρει και βασίλευε.

Τέλος στην κλάση ExecutePrimes και με την μέθοδο arrayOfPrimesSumConcurrent() εκτελούμε την ForkJoinPool.

```
public static long arrayOfPrimesSumConcurrent(int[] numbers) {
   return (FORK_JOIN_POOL.invoke(new ConcurrentArrayCountPrimes(numbers, 0,
numbers.length - 1)));
}
```

- Περίπτωση παράλληλης επεξεργασίας πρώτων αριθμών

Στη συνέχεια ορίζεται η κλάση ParallelArrayCountPrimes η οποία κάνει extend την κλάση RecursiveTask. Η κλάση αυτή κάνει όλη τη δουλειά και διαιρεί την αρχική εργασία σε υπό-μέρους εργασίες οι οποίες εκτελούνται παράλληλα (Parallel) μέχρι να ληφθεί το επιθυμητό αποτέλεσμα.

```
public class ParallelArrayCountPrimes extends RecursiveTask<Long> {

   private static final int PARALLEL_CUTOFF = 1000;
   private int[] numbers;
   private int lowerIndex, upperIndex;

   public ParallelArrayCountPrimes(int[] numbers, int lowerIndex, int upperIndex){
      this.numbers = numbers;
      this.lowerIndex = lowerIndex;
      this.upperIndex = upperIndex;
   }


   @Override
   protected Long compute() {
      int range = upperIndex - lowerIndex;
      if (range <= PARALLEL_CUTOFF) {
         return(PrimeUtils.countArrayPrimes(numbers, lowerIndex, upperIndex));
      } else {
         int middleIndex = lowerIndex + (range/2);
         ParallelArrayCountPrimes leftSummer = new ParallelArrayCountPrimes(numbers,
lowerIndex, middleIndex);
         ParallelArrayCountPrimes rightSummer = new ParallelArrayCountPrimes(numbers,
middleIndex+1, upperIndex);
         leftSummer.fork();
         Long rightSum = rightSummer.compute();
         Long leftSum = leftSummer.join();
         return(leftSum + rightSum);
      }
   }
}
```

Η υλοποίηση είναι ίδια με την προηγούμενη μόνο που στην περίπτωση αυτή οι εργασίες εκτελούνται παράλληλα και όχι ταυτόχρονα.

Αλγόριθμοι Υλοποίησης

Οι αλγόριθμοι που υλοποιήθηκαν στην παρούσα πτυχιακή είναι δύο. Ο πρώτος αλγόριθμος είναι αυτός της εύρεσης πρώτων αριθμών από έναν πίνακα Ν στοιχείων. Ο δεύτερος αλγόριθμος είναι αυτός του αθροίσματος δεκαδικών αριθμών από ένα πίνακα Ν στοιχείων.

6.1 Εύρεση πρώτων αριθμών ενός πίνακα

Στα μαθηματικά πρώτος αριθμός (ή απλά πρώτος) είναι ένας φυσικός αριθμός μεγαλύτερος της μονάδας με την ιδιότητα οι μόνοι φυσικοί διαιρέτες του να είναι η μονάδα και ο εαυτός του. Ένας φυσικός αριθμός μεγαλύτερος της μονάδας , ο οποίος δεν είναι πρώτος αριθμός ονομάζεται σύνθετος αριθμός. Για παράδειγμα, ο αριθμός 5 είναι πρώτος, επειδή μόνο οι αριθμοί 1 και 5 τον διαιρούν εξίσου, ενώ ο 6 είναι σύνθετος επειδή έχει διαιρέτες τους 2 και 3 εκτός των 1 και 6. Το μηδέν και το ένα δεν είναι πρώτοι αριθμοί. Το μηδέν συχνά δεν θεωρείται καν φυσικός αριθμός.
Για να επιτευχθεί η υλοποίηση του αλγόριθμου, τοποθετούνται τυχαίοι αριθμοί σε έναν πίνακα Ν στοιχείων και στην συνέχεια ελέγχεται εάν το κάθε στοιχείο του πίνακα είναι πρώτος αριθμός. Στο τέλος αθροίζεται το αποτέλεσμα για να βρεθούν πόσα από τα στοιχεία του πίνακα είναι πρώτοι αριθμοί.

```
private static boolean isPrime(long n) {
   if (n <= 1) return false;
   if (n == 2) return true;
   if (n %2 == 0) return false;
   for (int i = 3; i <= Math.sqrt(n); i += 2) {
      if (n %i == 0) return false;
   }
   return true;
}


public static long countArrayPrimes(int[] numbers, int lowerIndex, int upperIndex) {
   long sum = 0;
   for (int i = lowerIndex; i <= upperIndex; i++) {
      sum += (isPrime(numbers[i])) ? 1 : 0;
   }
   return (sum);
}
```

Με την πρώτη μέθοδο ελέγχεται εάν ένας από τους αριθμούς είναι πρώτος, και στη συνέχεια γίνεται μια καταμέτρηση αυτών των αριθμών.

6.2 Άθροισμα ακεραίων αριθμών ενός πίνακα

Η διαδικασία είναι απλή, αθροίζουμε τα στοιχεία ενός πίνακα δεκαδικών για να βρούμε το τελικό αποτέλεσμα. Η μέθοδος υλοποίησης του αθροίσματος είναι η παρακάτω.

```
public static double arraySum(double[] numbers, int lowerIndex, int upperIndex) {
   double sum = 0;
   for (int i = lowerIndex; i <= upperIndex; i++) {
      sum += numbers[i];
   }
   return (sum);
}
```

Το μοντέλο δόμησης των δεδομένων

Τα δεδομένα που μεταφέρονται από συσκευή σε συσκευή έχουν κάποια συγκεκριμένη δομή. Η κάθε συσκευή που συνδέεται στο δίκτυο έχει μια ip διεύθυνση η οποία την κάνει να ξεχωρίζει από τις υπόλοιπες συσκευές. Επίσης έχει και μια μοναδική mac address, και το όνομα της. Όλα αυτά τα δεδομένα μαζί με τα υπόλοιπα που περιγράφονται στον πίνακα 2, συνθέτουν το μοντέλο δόμησης των δεδομένων που μεταφέρονται από τον εξυπηρετητή στους πελάτες και αντίστροφα.

serverIp	Η διεύθυνση ip του Server
deviceName	Το όνομα της συσκεύης
deviceMacAddress	Η μοναδική mac address της συσκεύης
clientIp	Η ip του πελάτη
serverPort	Η θύρα (Port) που ακούει ο εξυπηρετητής
clientPort	Η θύρα (Port) που ακούει ο πελάτης
resultFromSumArray	Το αποτέλεσμα αθροίσματος των δεκαδικών
hasCompleteTheJob	Εάν η δουλειά του πελάτη έχει τελειώσει
startTime	Ο χρόνος που γίνεται η ανάθεση της δουλειάς στον πελάτη από τον εξυπηρετητή
endTime	Ο χρόνος που λαμβάνει πίσω το αποτέλεσμα ο εξυπηρετητής(Ο συνολικός χρόνος)
elapsedSeconds	Ο συνολικός χρόνος που χρειάστηκε για να γίνει ο υπολογισμός και να επιστραφεί το αποτέλεσμα πίσω στον εξυπηρετητή
primeChunkedArray	Ένας χωρισμένος πίνακας με στοιχεία τους πρώτους αριθμούς για υπολογισμό για κάθε συσκευή ξεχωριστά
primeResultFromSumArray	Ένας πίνακας με τα αποτελέσματα αθροίσματος πρώτων αριθμών
chunkedArray	Ένας χωρισμένος πίνακας με στοιχεία τους δεκαδικούς αριθμούς για υπολογισμό για κάθε συσκευή ξεχωριστά

Πίνακας 2. Μοντέλο δεδομένων

7.1 Λίστα Hash Map από δεδομένα

Μια λίστα Hash στην JAVA είναι η λίστα όπου κάθε γραμμή της περιλαμβάνει ένα κλειδί(Key) και μια δομή από δεδομένα(values). Το κλειδί αυτό πρέπει να είναι μοναδικό για κάθε γραμμή της λίστας. Με άλλα λόγια η κάθε γραμμή του πίνακα θα πρέπει να είναι μοναδική. Η πτυχιακή υλοποιήθηκε με βάση αυτό το μοντέλο. Το κλειδί για την κάθε γραμμή αντιστοιχεί στην ip της κάθε συσκευής (πελάτη) και άρα είναι μοναδικό. Κάθε κλειδί τώρα έχει και κάποιες τιμές(Values). Οι τιμές αυτές αντιστοιχούν στις τιμές του πίνακα 1.
Η δομή της λίστας είναι η εξής:

```
public static Map<String, ClientModel> mapClients = new HashMap<>();
```

Όπου το Generic String αφορά στο κλειδί του πίνακα δηλαδή την ip, και το μοντέλο ClientModel αφορά στις τιμές που θα έχει αυτό το μοναδικό key. Η δομή του ClientModel είναι η ακόλουθη:

```
package com.example.vangelis.connectingdevices.model;

import android.net.wifi.p2p.WifiP2pDevice;
```

```
import java.io.Serializable;
import java.net.Socket;
import java.util.List;

public class ClientModel implements Serializable {

    private String serverIp, deviceName, deviceMacAddress, clientIp;
    private int serverPort, clientPort;
    private double resultFromSumArray;
    private double[] chunkedArray;
    private boolean hasCompleteTheJob;
    private long startTime, endTime;
    private static final double ONE_BILLION = 1_000_000_000;
    private double elapsedSeconds;
    private int[] primeChunkedArray;
    private long primeResultFromSumArray;

    public ClientModel() {
    }

    public double getElapsedSeconds() {
        return this.elapsedSeconds;
    }

    public void setElapsedSeconds(double elapsedSeconds) {
        this.elapsedSeconds = elapsedSeconds;
    }

    public double getStartTime() {
        return startTime;
    }

    public void setStartTime(long startTime) {
        this.startTime = startTime;
    }

    public double getEndTime() {
        return endTime;
    }

    public void setEndTime(long endTime) {
        this.endTime = endTime;
        this.elapsedSeconds = (endTime - this.startTime) / ONE_BILLION;
    }

    public String getClientIp() {
        return clientIp;
    }

    public void setClientIp(Socket socket) {
        String[] clientIpAddress =
socket.getInetAddress().toString().split("/");
        this.clientIp = clientIpAddress[1];
    }
```

```
    public boolean getHasCompleteTheJob() {
        return hasCompleteTheJob;
    }

    public void setHasCompleteTheJob(boolean hasCompleteTheJob) {
        this.hasCompleteTheJob = hasCompleteTheJob;
    }


    public String getServerIp() {
        return serverIp;
    }

    public void setServerIp(Socket socket) {
        this.serverIp = socket.getLocalAddress().getHostName();
    }

    public String getDeviceName() {
        return deviceName;
    }

    public void setDeviceName(WifiP2pDevice[] wifiP2pDevice, List<String>
deviceAddress) {
        for (WifiP2pDevice j : wifiP2pDevice) {
            for (String i : deviceAddress) {
                if (j.deviceAddress.equals(i)) {
                    this.deviceName = j.deviceName;
                    break;
                }
            }
        }
    }

    public String getDeviceMacAddress() {
        return deviceMacAddress;
    }

    public void setDeviceMacAddress(WifiP2pDevice[] wifiP2pDevice,
List<String> deviceAddress) {
        for (WifiP2pDevice j : wifiP2pDevice) {
            for (String i : deviceAddress) {
                if (j.deviceAddress.equals(i)) {
                    this.deviceMacAddress = j.deviceAddress;
                    //Log.e("Mac Address", this.deviceMacAddress);
                    break;
                }
            }
        }
    }

    public int getServerPort() {
        return serverPort;
    }

    public void setServerPort(Socket socket) {
        this.serverPort = socket.getLocalPort();
```

```
    }

    public int getClientPort() {
        return clientPort;
    }

    public void setClientPort(Socket socket) {
        this.clientPort = socket.getPort();
    }

    public double getResultFromSumArray() {
        return resultFromSumArray;
    }

    public void setResultFromSumArray(double resultFromSumArray) {
        this.resultFromSumArray = resultFromSumArray;
    }

    public double[] getChunkedArray() {
        return chunkedArray;
    }

    public void setChunkedArray(double[] chunkedArray) {
        this.chunkedArray = chunkedArray;
    }

    public int[] getPrimeChunkedArray() {
        return primeChunkedArray;
    }

    public void setPrimeChunkedArray(int[] primeChunkedArray) {
        this.primeChunkedArray = primeChunkedArray;
    }

    public long getPrimeResultFromSumArray() {
        return primeResultFromSumArray;
    }

    public void setPrimeResultFromSumArray(long primeResultFromSumArray) {
        this.primeResultFromSumArray = primeResultFromSumArray;
    }
}
```

Η Εφαρμογή

Είναι τόσο διαδεδομένη η χρήση φορητών συσκευών στην καθημερινή ζωή μας στην εποχή που ζούμε, που κάθε άτομο σχεδόν έχει στην κατοχή του μία. Αυτόν μας δίνει ένα πολύ μεγάλο κίνητρο αφού θα μπορούσαμε να χρησιμοποιήσουμε τους διαθέσιμους πόρους μιας συσκευής για έναν υπολογισμό και αν αυτή συσκευή βρισκόταν σε ένα δίκτυο θα αυτό θα μπορούσε να γίνει για όλες τις κινητές συσκευές του δικτύου.

8.1 Το πακέτο των εφαρμογών

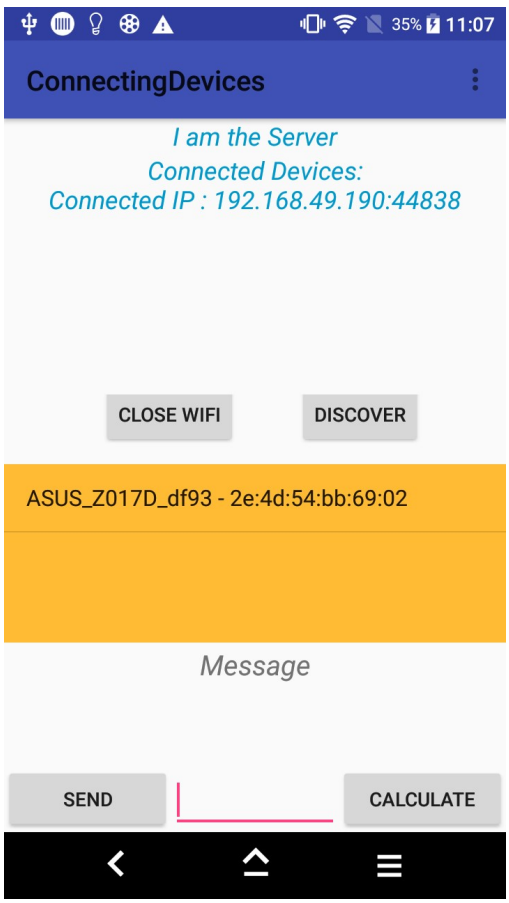
Οι απαιτήσεις της εργασίας αυτής επέβαλαν για την ανάπτυξη των ζητουμένων την δημιουργία μιας εφαρμογής η οποία θα λειτουργεί αποκλειστικά και μόνο για κινητές συσκευές και ταμπλέτες με λογισμικό android. Έτσι αναπτύχθηκε η εφαρμογή η οποία είναι συμβατή για συσκευές με έκδοση λογισμικού android από το 5.1 (Android Lollipop) και πάνω. Το λογισμικό που χρησιμοποιήθηκε για την ανάπτυξη της εφαρμογής είναι το Android Studio 3.1 με την γλώσσα JAVA στην έκδοση 9.

8.2 Το Interface

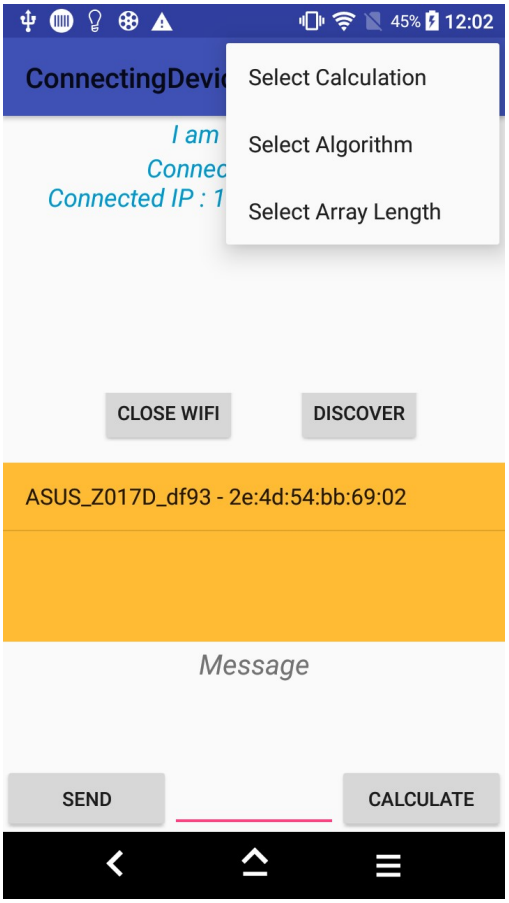
Αξίζει να σημειωθεί ότι αναπτύχθηκε μια εφαρμογή η οποία υποστηρίζει και αναγνωρίζει αυτόματα ποιος είναι ο εξυπηρετητής και ποιος ο πελάτης. Διαμορφώνει δηλαδή το UI και την λειτουργικότητα της ανάλογα την περίπτωση. Εφ' όσον η συσκευή στο δίκτυο αναγνωριστεί ως εξυπηρετητής, αυτόματα δημιουργείται ένα action Bar στο επάνω μέρος της εφαρμογής, το οποίο έχει κάποιες λειτουργικότητες.

8.2.1 Εξυπηρετητής (Server)

Ο εξυπηρετητής είναι υπεύθυνος στο να κάνει ορισμένες αρχικοποιήσεις **Εικόνα[9]**. Ο χρήστης θα πρέπει να επιλέξει τον τύπο του υπολογισμού (Σειριακός, Παράλληλος). Στην συνέχεια θα πρέπει να επιλέξει το είδος του αλγορίθμου (Πρώτοι αριθμοί, άθροισμα δεκαδικών), και τέλος θα πρέπει να δοθεί και το μήκος που θα έχει ο πίνακας προς υπολογισμό. Αν επιλεγούν όλα αυτά τα στοιχεία τότε ο χρήστης θα μπορέσει να προχωρήσει στον υπολογισμό πατώντας το κουμπί CALCULATE. Από κάτω ακριβώς υπάρχουν κάποια μηνύματα. Ενημερώνουν τον χρήστη εάν η συσκευή του είναι ο εξυπηρετητής (Server), και ποιοι πελάτες(Clients) έχουν συνδεθεί σε αυτόν δείχνοντας σε μια λίστα την IP και την θύρα της συνδεδεμένης συσκευής. Στη συνέχεια υπάρχουν δύο κουμπιά το ένα είναι για να ανοίγει και να κλείνει το δίκτυο WI-FI και το δεύτερο για να ψάχνει και να ανακαλύπτει διαθέσιμες συσκευές προς σύνδεση. Οι διαθέσιμες συσκευές (peers) εμφανίζονται στο κίτρινο πλαίσιο η μια κάτω από την άλλη κάνοντας αναφορά στο όνομα της συσκευής και στην μοναδική διεύθυνση Mac που διαθέτει. Τέλος στο τελευταίο πλαίσιο της εφαρμογής μπορούμε να στείλουμε ένα μήνυμα από τον εξυπηρετητή σε όλες τις συνδεδεμένες συσκευές του δικτύου και να λάβουμε και μια απάντηση.



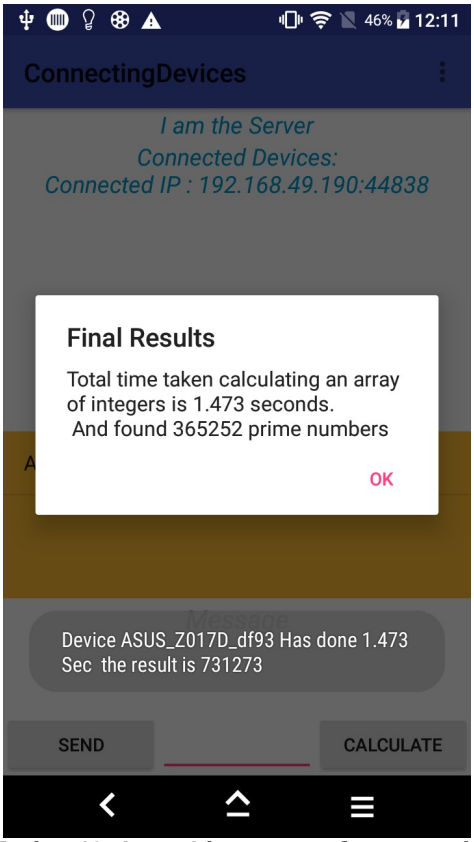
Εικόνα 8. UI σε συσκευή εξυπηρετητή



Εικόνα 9. UI σε συσκευή εξυπηρετητή

Διαδικασία υπολογισμού

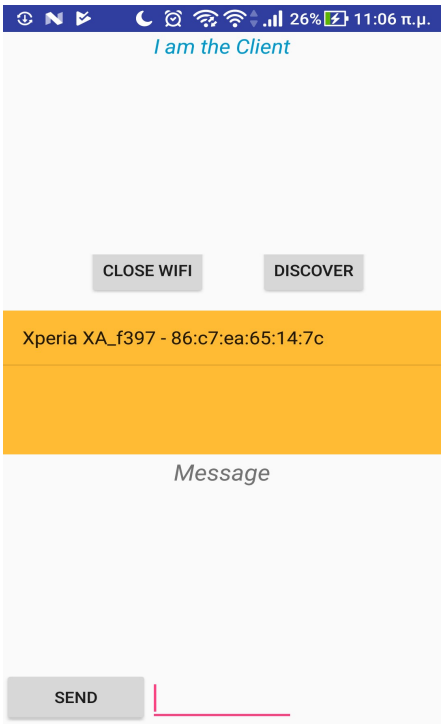
Όταν πατηθεί το κουμπί CALCULATE υπάρχει ένας χρόνος αναμονής από την αποστολή του αιτήματος από τον εξυπηρετητή σε όλους τους πελάτες έως την επιστροφή του αποτελέσματος. Οι πελάτες επεξεργάζονται το αίτημα και επιστρέφουν ο κάθε ένας πίσω το αποτέλεσμα. Όταν αυτό ολοκληρωθεί παίρνουμε τα αποτελέσματα στην οθόνη του εξυπηρετητή σε ένα Alert Dialog Box μαζί με ένα Toast Message **Εικόνα[10]**. Το μήνυμα Alert μας ενημερώνει πως ο συνολικός χρόνος ήταν 1,4 δευτερόλεπτα και το αποτέλεσμα του υπολογισμού ήταν να βρεθούν 365.252 πρώτοι αριθμοί από τον πίνακα ακεραίων που επιλέξαμε. Ο συνολικός χρόνος ξεκινάει να μετράει από την αποστολή του αιτήματος προς τους πελάτες μέχρι και την επιστροφή του τελευταίου αποτελέσματος από κάποιον πελάτη.
Το σύντομο μήνυμα Toast αναφέρει την συσκευή που έκανε τον υπολογισμό και πόσο χρόνο ο υπολογισμός χρειάστηκε να γίνει από την στιγμή που έφυγαν τα δεδομένα από τον εξυπηρετητή μέχρις ότου επεστράφησαν πίσω. Τον ολικό χρόνο δηλαδή και όχι τον χρόνο υπολογισμού.



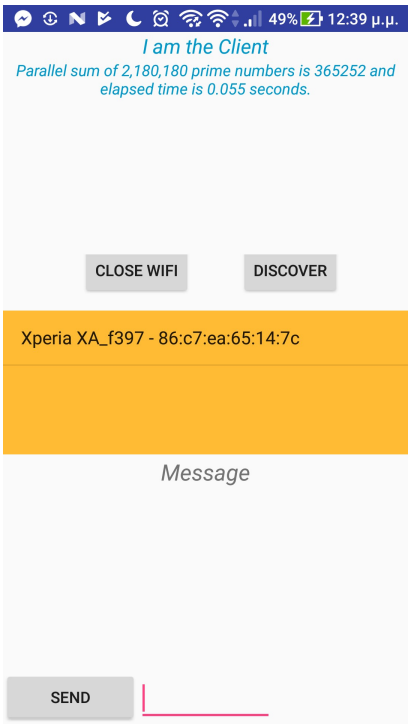
Εικόνα 10. Αποτελέσματα σε εξυπηρετητή

8.2.2 Πελάτες (Clients)

Από την πλευρά της η συσκευή του πελάτη σε αυτή τη φάση ενημερώνει απλά τον χρήστη ότι είναι πελάτης (Client). Εξακολουθεί να έχει τις λειτουργικότητές του ανοίγματος δικτύου όπως επίσης και τις εμφάνισης διαθέσιμων συσκευών (peers), και του μηνύματος προς τον εξυπηρετητή **Εικόνα [11]**.



Εικόνα 11. UI σε συσκευή πελάτη



Εικόνα 12. Αποτελέσματα σε πελάτη

Διαδικασία υπολογισμού

Με το που ολοκληρωθεί ο υπολογισμός στην συσκευή του πελάτη εμφανίζεται ένα μήνυμα που τον ενημερώνει ποια διαδικασία ακολουθήθηκε πόσοι αριθμοί υπολογίστηκαν ποιο είναι το αποτέλεσμα και τον χρόνο υπολογισμού **Εικόνα [12]**.

Μετρήσεις

Οι μετρήσεις πραγματοποιήθηκαν σε τέσσερις διαφορετικές συσκευές.

- ✓ Τηλέφωνο Asus Zen Phone 3 (8 cores) (Android Nougat 7.1)
- ✓ Τηλέφωνο Sony XPeria XA F3111 (8 cores) (Android Nougat 7.1)
- ✓ Ταμπλέτα BitMore Tab1022H (4 cores) (Android 6.0 Marshmallow)
- ✓ Τηλέφωνο Xiaomi RedmiNote 5 (8 cores) (Android Nougat 7.1)

Καθ' όλη τη διάρκεια των μετρήσεων το τηλέφωνο Sony XPeria έπαιζε τον ρόλο του εξυπηρετητή (Server), και οι υπόλοιπες συσκευές ήταν οι Πελάτες(Clients) .Όπως βλέπουμε στον πίνακα[3], υπάρχουν πέντε στήλες. Το μέγεθος του πίνακα υποδηλώνει από πόσα στοιχεία υπολογισμού (elements) θα αποτελείται ο πίνακας προς υπολογισμό. Ο τύπος εργασίας είναι ο αλγόριθμος που χρησιμοποιήθηκε για την εκτέλεση της εργασίας. Ο συνολικός χρόνος είναι χρόνος που χρειάστηκε από την στιγμή που ξεκινάει η αποστολή δεδομένων από τον εξυπηρετητή μέχρι την στιγμή που ο εξυπηρετητής θα παραλάβει και το τελευταίο αποτέλεσμα από τις συσκευές πίσω. Ο αριθμός συνδεδεμένων συσκευών δείχνει πόσοι είναι οι πελάτες στο υπάρχον δίκτυο. Και τέλος με τον αύξων αριθμό της εργασίας γίνεται μια αντιστοιχία στον πίνακα 4.

Εργασίες

Μέγεθος Πίνακα	Τύπος Εργασίας	Συνολικός Χρόνος (Δευτ)	Αριθμός Συνδεδεμένων Συσκευών (Clients)	Α / Α Εργασίας
1.000.000	Άθροισμα Δεκαδικών	1,80	1	**1**
2.000.000	Άθροισμα Δεκαδικών	3,59	1	**2**
5.000.000	Άθροισμα Δεκαδικών	9,2	1	**3**
10.000.000	Άθροισμα Δεκαδικών	15,0	1	**4**
1.000.000	Άθροισμα Δεκαδικών	1,9	1	**5**
2.000.000	Άθροισμα Δεκαδικών	3,4	1	**6**
5.000.000	Άθροισμα Δεκαδικών	7,95	1	**7**
10.000.000	Άθροισμα Δεκαδικών	14,8	1	**8**
10.000.000	Άθροισμα Δεκαδικών	14,8	1	**9**
1.000.000	Άθροισμα Πρώτων	1,7	1	**10**
2.000.000	Άθροισμα Πρώτων	2,2	1	**11**
5.000.000	Άθροισμα Πρώτων	5,6	1	**12**
10.000.000	Άθροισμα Πρώτων	8,8	1	**13**
1.000.000	Άθροισμα Πρώτων	1,36	1	**14**
2.000.000	Άθροισμα	2,0	1	**15**

	Πρώτων			
5.000.000	Άθροισμα Πρώτων	5,0	1	**16**
10.000.000	Άθροισμα Πρώτων	7,7	1	**17**
10.000.000	Άθροισμα Πρώτων	7,7	1	**18**
1.000.000	Άθροισμα Δεκαδικών	2.085	2	**19**
2.000.000	Άθροισμα Δεκαδικών	3,164	2	**20**
5.000.000	Άθροισμα Δεκαδικών	7,709	2	**21**
10.000.000	Άθροισμα Δεκαδικών	15,0	2	**22**
1.000.000	Άθροισμα Δεκαδικών	1,893	2	**23**
2.000.000	Άθροισμα Δεκαδικών	3,2	2	**24**
5.000.000	Άθροισμα Δεκαδικών	7,7	2	**25**
10.000.000	Άθροισμα Δεκαδικών	14,9	2	**26**
1.000.000	Άθροισμα Πρώτων	1,52	2	**27**
2.000.000	Άθροισμα Πρώτων	2,731	2	**28**
5.000.000	Άθροισμα Πρώτων	6,75	2	**29**
10.000.000	Άθροισμα Πρώτων	12,5	2	**30**
1.000.000	Άθροισμα Πρώτων	1,489	2	**31**
2.000.000	Άθροισμα Πρώτων	2,3	2	**32**
5.000.000	Άθροισμα Πρώτων	5,9	2	**33**
10.000.000	Άθροισμα Πρώτων	10,7	2	**34**
10.000.000	Άθροισμα Πρώτων	10,5	2	**35**
1.000.000	Άθροισμα Δεκαδικών	1,59	2	**36**
2.000.000	Άθροισμα Δεκαδικών	2,75	2	**37**
5.000.000	Άθροισμα Δεκαδικών	6,39	2	**38**
10.000.000	Άθροισμα Δεκαδικών	12,48	2	**39**
1.000.000	Άθροισμα Δεκαδικών	2,24	2	**40**
2.000.000	Άθροισμα Δεκαδικών	2,90	2	**41**
5.000.000	Άθροισμα Δεκαδικών	6,39	2	**42**

10.000.000	Άθροισμα Δεκαδικών	12,43	2	**43**
1.000.000	Άθροισμα Πρώτων	1,331	2	**44**
2.000.000	Άθροισμα Πρώτων	4,075	2	**45**
5.000.000	Άθροισμα Πρώτων	4,866	2	**46**
10.000.000	Άθροισμα Πρώτων	6,748	2	**47**
1.000.000	Άθροισμα Πρώτων	0,764	2	**48**
22.000.000	Άθροισμα Πρώτων	7,225	2	**49**
5.000.000	Άθροισμα Πρώτων	5,372	2	**50**
10.000.000	Άθροισμα Πρώτων	6,366	2	**51**

Πίνακας 3. Συσκευές και συνολικοί χρόνοι υλοποίησης εργασιών

Ο πίνακας[4] αποτελείται από πέντε στήλες. Ο αύξων αριθμός εργασίας κάνει την αντιστοίχιση της εργασίας του πίνακα [3] με τον πίνακα [4]. Για παράδειγμα για την εργασία με αύξων αριθμό 19 του πίνακα [3], στον πίνακα [4] αντιστοιχούν 2 γραμμές γιατί υπάρχουν 2 συνδεδεμένοι πλάτες στο δίκτυο . Η πρώτη γραμμή μας δείχνει ποια είναι η συσκευή, πόσο χρόνο (εσωτερικός χρόνος συσκευής) έκανε αυτή η συσκευή στον σειριακό υπολογισμό, πόσο χρόνο στον παράλληλο υπολογισμό, και τέλος τον χρόνο στον παράλληλο υπολογισμό 2.

Συσκευές και εσωτερικοί χρόνοι

Α / Α Εργασίας	**Μοντέλο Συσκευής**	**Διαδοχικός (Serial)Χρόνος Συσκευής(Δευτ)**	**Παράλληλος(Parallel) Χρόνος Συσκευής (Δευτ)**	**Παράλληλος 2(Parallel 2) Χρόνος Συσκευής (Δευτ)**
1	Asus	0,032		
2	Asus	0,012		
3	Asus	0,031		
4	Asus	0,056		
5	Asus		0,026	
6	Asus		0,023	
7	Asus		0,026	
8	Asus		0,034	
9	Asus			0,046
10	Asus	0,174		
11	Asus	0,143		
12	Asus	0,366		
13	Asus	0,729		
14	Asus		0,038	
15	Asus		0,050	
16	Asus		0,088	
17	Asus		0,133	
18	Asus			0,172
19	Asus	0,107		
19	Bit More	0.036		
20	Asus	0,021		

20	Bit More	0,046		
21	Asus	0,017		
21	Bit More	0,119		
22	Asus	0,032		
22	Bit More	0,234		
23	Asus		0,105	
23	Bit More		0,032	
24	Asus		0,047	
24	Bit More		0,017	
25	Asus		0,015	
25	Bit More		0,085	
26	Asus		0,037	
26	Bit More		0,081	
27	Asus	0,098		
27	Bit More	0,409		
28	Asus	0,106		
28	Bit More	0,787		
29	Asus	0,216		
29	Bit More	1,926		
30	Asus	0,363		
30	Bit More	3,765		
31	Asus		0,040	
31	Bit More		0,222	
32	Asus		0,040	
32	Bit More		0,244	
33	Asus		0,073	
33	Bit More		0,550	
34	Asus		0,089	
34	Bit More		1,131	
35	Asus			0,089
35	Bit More			1,182
36	Asus	0,005		
36	Xiaomi	0,005		
37	Asus	0,010		
37	Xiaomi	0,009		
38	Asus	0,017		
38	Xiaomi	0,022		
39	Asus	0,031		
39	Xiaomi	0,039		
40	Asus	0,023		
40	Xiaomi	0,043		
41	Asus	0,030		
41	Xiaomi	0,010		
42	Asus	0,018		
42	Xiaomi	0,015		
43	Asus	0,023		
43	Xiaomi	0,023		
44	Asus		0,056	
44	Xiaomi		0,070	
45	Asus		0,091	
45	Xiaomi		0,094	
46	Asus		0,193	
46	Xiaomi		0,217	
47	Asus		0,0375	

47	Xiaomi		0,398	
48	Asus		0,030	
48	Xiaomi		0,028	
49	Asus		0,029	
49	Xiaomi		0,028	
50	Asus		0,067	
50	Xiaomi		0,052	
51	Asus		0,076	
51	Xiaomi		0,080	

Πίνακας 3. Συσκευές, είδος αλγορίθμου, και εσωτερικοί χρόνοι υλοποίησης εργασιών

9.1 Συμπεράσματα

Το πρώτο πρόβλημα που εντοπίστηκε στην πτυχιακή αφορά το πρωτόκολλο WIFI-DIRECT. Συσκευές που υποστηρίζουν το συγκεκριμένο πρωτόκολλο, θα πρέπει να έχον λογισμικό Android 4.0 και πάνω. Όσες συσκευές έχουν χαμηλότερο λογισμικό απλά δεν μπορούν να υποστηρίξουν το συγκεκριμένο πρωτόκολλο [12]. Όμως ακόμα και αν η συσκευή διαθέτει λογισμικό πάνω από το προκαθορισμένο δεν είναι σίγουρο ότι θα υποστηρίζει πολλούς πελάτες συνδεδεμένους σε έναν εξυπηρετητή (one to many) [7]. Το σίγουρο είναι ότι θα υποστηρίζει σύνδεση (one to one) δηλαδή έναν πελάτη και έναν εξυπηρετητή. Όσον αφορά τον αριθμό των πελατών έχουν γίνει μελέτες που δείχνουν ότι το πρωτόκολλο μπορεί να υποστηρίξει μέχρι και 4 συσκευές ταυτόχρονα χωρίς προβλήματα [14].
Αυτό όπως καταλαβαίνουμε περιορίζει σημαντικά όσον αφορά το τελικό αποτέλεσμα των μετρήσεων. Γιατί άλλο είναι να έχεις πολλές συσκευές στο δίκτυο που θα κάνουν υπολογισμούς και άλλο να έχεις μόνο έναν αριθμό συσκευών. Στην παρούσα πτυχιακή δοκιμάστηκαν 3 συνδεδεμένες συσκευές σε δίκτυο WIFI-DIRECT, η μια εκ των οποίον ήταν ο εξυπηρετητής(Server) και οι υπόλοιπες 2 οι πελάτες(Clients).

Ένα δεύτερο πρόβλημα επίσης σημαντικό εντοπίστηκε στην ταχύτερη αποστολή των δεδομένων από τους πελάτες. Όταν μια συσκευή η οποία είναι συνδεδεμένη στο δίκτυο και το υλικό(hardware) του WIFI-DIRECT που έχει, είναι κακής ποιότητας δηλαδή αργό, αργεί και αυτή με τη σειρά της να στείλει και να λάβει αποτελέσματα με συνέπεια να καθυστερεί να έρθει το τελικό αποτέλεσμα πίσω στον παραλήπτη (Server). Αυτό φαίνεται και από τον πίνακα αποτελεσμάτων. Συνδέοντας την συσκευή BitMore στο δίκτυο, σαν πελάτη, φαίνεται να έχουμε μια καθυστέρηση στο τελικό αποτέλεσμα και αυτό λόγω του φτηνού υλικού στα περιφερειακά της εξαρτήματα. Το ίδιο ισχύει και όταν η συσκευή – πελάτης κάνει κάποιες εργασίες μεγαλύτερης προτεραιότητας και αργεί να στείλει πίσω το αποτέλεσμα στον εξυπηρετητή με αποτέλεσμα να υπάρχει και σε αυτήν την περίπτωση μια καθυστέρηση.

Τα παραπάνω συμπεράσματα αφορούν το τελικό αποτέλεσμα και τη χρονική διάρκειά του. Υπάρχει όμως και ο ενδιάμεσος χρόνος, ο χρόνος που κάνει δηλαδή η κάθε συσκευή εσωτερικά για την εξαγωγή του αποτελέσματος. Ο χρόνος αυτός εξαρτάται κάθε φορά από τον τρόπο υλοποίησης. Ο τρόπος αυτός ενδέχεται να είναι η σειριακή υλοποίηση, ή η παράλληλη υλοποίηση με την χρήση της βιβλιοθήκης της Java Fork Join Pool.
Η σειριακή υλοποίηση είναι όπως την γνωρίζουμε. Όλες οι πράξεις γίνονται σειριακά και χωρίς κάποια νήματα (Threads) να τρέχουν παράλληλα. Όπως δείχνουν τα αποτελέσματα, μας συμφέρει να χρησιμοποιούμε σειριακή υλοποίηση στην περίπτωση που έχουμε μικρό όγκο δεδομένων. Αυτό συμβάλει στην ταχύτερη εξαγωγή του αποτελέσματος. Στην περίπτωση που έχουμε μεγάλο όγκο από δεδομένα είναι σίγουρο πως θα πρέπει να χρησιμοποιήσουμε παράλληλες διεργασίες για να επεξεργαστούμε τα δεδομένα μας, και αυτό μας στο προσφέρει η χρήση της βιβλιοθήκης Fork Join. Στον πίνακα [2] βλέπουμε ότι όσο μεγαλώνει ο όγκος των δεδομένων μας τόσο μειώνεται ο χρόνος της παράλληλης επεξεργασίας αισθητά σε σχέση με την σειριακή. Αυτό συμβαίνει διότι εσωτερικά η βιβλιοθήκη FORK JOIN κόβει την αρχική

εργασία σε επιμέρους διεργασίες οι οποίες πραγματοποιούνται ταυτόχρονα με σκοπό να βγάλουν το τελικό αποτέλεσμα.

Η χρήση του πρωτοκόλλου WIFI DIRECT ακόμα δεν έχει φτάσει στο επίπεδο που θα επιθυμούσαμε. Θα πρέπει να γίνει ταχύτερο εκτός από την υλοποίηση του κώδικα, αλλά και σε επίπεδο υλικού. Βέβαια είναι ταχύτερο από άλλα πρωτόκολλά μεταφοράς όπως το Blue Tooth, αυτό όμως δεν το καθιστά το ταχύτερο.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Java Tutorial, “Java” http://tutorials.jenkov.com, 2015. [accessed 2018-05-14].

[2] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,”
Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

[3] L. Svobodova, Client/Server Model of Distributed Processing, pp. 485–498. Berlin, Heidelberg:
Springer Berlin Heidelberg, 1985.

[4] M. D. Network, “Tiered distribution.” https://msdn.microsoft.com/en-us/library/ff647195.aspx. [accessed 2018-06-14].

[5] Java Tutorial, “Java” http://www.baeldung.com, 2015. [accessed 2018-05-14].

[6] Allen Holub, “Java Network.” https://www.javaworld.com/article/2076774/java-concurrency/programming-java-threads-in-the-real-world--part-1.html. [accessed 2018-07-22].

[7] Wifi Alliance, “wi-fi.org https://www.wi-fi.org/knowledge-center/faq/how-many-devices-can-connect [accessed 2018-06-15].

[8] U. of California, “Seti@home.” http://setiathome.berkeley.edu/sah_status.html. [accessed 2018-06-15].

[9] L. S. de Oliveira, “P3: Parallel peer-to-peer,” informatics, Faculdade de Ciências da Universidade
do Porto, June 2003.

[10] J. P. F. de Magalhães, “Um sistema de ficheiros distribuído para uma arquitectura peerto-
peer,” informatics, Faculdade de Ciências da Universidade do Porto, April 2004.

[11] L. Oliveira, L. Lopes, and F. Silva, “p3: Parallel peer to peer an internet parallel programming
environment,” in International Conference on Research in Networking, pp. 274–288, Springer, 2002.

[12] Developers, “Android.com”
https://developer.android.com/guide/topics/connectivity/wifip2p. [accessed 2018-06-15]

[13] Marty Hall, Parallel Programming with Fork/Join, Publishing,
http://courses.coreservlets.com/Course-Materials/java.html [accessed 2018-05-11].

[14] Stack Overflow, “Wifi-Direct”
https://developer.android.com/guide/topics/connectivity/wifip2p. [accessed 2018-06-15]

[15] Helder Vasconcelos, “Asynchronous-Android-Programming”
PACKT Publishing, 2016.

[16] Anders Goransson, “Efficient Android Threading, Asynchronous Processing Techniques for Android Applications”
Oreilly, 2014.

[17] J. Rodrigues, J. Silva, R. Martins, L. Lopes, U. Drolia, P. Narasimhan, and F. Silva, Benchmarking
Wireless Protocols for Feasibility in Supporting Crowdsourced Mobile Computing,
pp. 96–108. Cham: Springer International Publishing, 2016.