ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Μελέτη της Ομιχλώδους Επεξεργασίας (Fog Computing) - Σχεδίαση και Ανάπτυξη Εφαρμογής Κόμβου Ομιχλώδους Επεξεργασίας

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημόπουλος Δημήτριος
Φαμέλης Παναγιώτης

Επιβλέπων : Ιάκωβος Βενιέρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2018

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Μελέτη της Ομιχλώδους Επεξεργασίας (Fog Computing) - Σχεδίαση και Ανάπτυξη Εφαρμογής Κόμβου Ομιχλώδους Επεξεργασίας

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημόπουλος Δημήτριος
Φαμέλης Παναγιώτης

Επιβλέπων : Ιάκωβος Βενιέρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 15η Ιουνίου 2018

.....	.....	.....
Ιάκωβος Βενιέρης	Δήμητρα-Θεοδώρα Κακλαμάνη	Γιώργος Ματσόπουλος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π

Αθήνα, Ιούνιος 2018

.....

Δημόπουλος Δημήτριος

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

.....

Φαμέλης Παναγιώτης

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

Περίληψη

Σήμερα περισσότερο από ποτέ, εισβάλει στην καθημερινότητά μας μια πληθώρα συσκευών με δικτυακές δυνατότητες, ενώ καθημερινά αντικείμενα μετατρέπονται σε «έξυπνα» παρέχοντας καινούργιες, προσαρμοζόμενες στον χρήστη δυνατότητες και λειτουργίες, αξιοποιώντας την σύνδεση στο διαδίκτυο. Οι παραπάνω συσκευές μαζί με αισθητήρες, ενεργοποιητές και συστήματα που συνδέονται στο διαδίκτυο και επικοινωνούν μεταξύ τους αποτελούν το Διαδίκτυο των Πραγμάτων. Οι συσκευές αυτές παράγουν τεράστια ποσά δεδομένων, η επεξεργασία των οποίων είναι αναγκαία προϋπόθεση για να μπορούν να παρέχουν τις εξατομικευμένες αυτές λειτουργίες και να προσαρμόζονται στις αλλαγές του περιβάλλοντός τους.

Η κυρίαρχη μέχρι τώρα προσέγγιση για την επεξεργασία των δεδομένων είναι μέσω της αξιοποίησης υπολογιστικών νεφών για να εξασφαλιστούν οι απαραίτητοι επεξεργαστικοί πόροι, ο αποθηκευτικός χώρος, ο συντονισμός και η επικοινωνία κλπ. Εκτιμάται ότι η περαιτέρω διεύρυνση του Διαδικτύου των πραγμάτων συμβάλει στην εκθετική αύξηση των παραγόμενων δεδομένων, δημιουργώντας ερωτήματα για το αν είναι αποδοτική η μεταφορά του συνόλου των δεδομένων επιβαρύνοντας το δίκτυο και ανοίγει την συζήτηση για εναλλακτικές μεθόδους διαχείρισης των δεδομένων αυτών.

Μια τέτοια πρόταση είναι η λογική της ομιχλώδους επεξεργασίας (Fog Computing). Αυτή αφορά την διεκπεραίωση μέρους της διαχείρισης των παραγόμενων δεδομένων, κοντά στο τόπο που παράγονται, αξιοποιώντας τις υπολογιστικές δυνατότητες των πολυάριθμων συσκευών που υπάρχουν στον εγγύς χώρο, όπως smartphones, laptops, tablets κλπ. Με τον τρόπο αυτό θα μειωθεί ο συνολικός φόρτος του δικτύου καθώς τα δεδομένα μπορούν να υποστούν τουλάχιστον μια προεπεξεργασία, ελαχιστοποιώντας την ποσότητα που προωθείται στο υπολογιστικό νέφος. Η βελτίωση των υπολογιστικών δυνατοτήτων των συσκευών καθημερινής χρήσης καθιστά εφικτή την επεξεργασία τοπικά ενώ αξιοποιεί μέχρι πρότινος μερικώς αναξιοποίητους πόρους των συσκευών αυτών. Τέλος η λειτουργία σε τοπικά δίκτυα μπορεί να πετύχει χρόνους απόκρισης σημαντικά μικρότερους από αυτούς του υπολογιστικού νέφους, παράγοντας που μπορεί να είναι σημαντικός για συγκεκριμένες εφαρμογές, όπως ιατρικές.

Στο πλαίσιο της παρούσας εργασίας μελετάμε τις βασικές αρχές και την λογική της νεφελώδους επεξεργασίας. Βασιζόμενοι σε αυτές, σχεδιάσαμε και υλοποιήσαμε μια εφαρμογή που να μπορεί να λειτουργήσει ως πρωτότυπο λειτουργίας ενός κόμβου νεφελώδους επεξεργασίας.

Λέξεις κλειδιά: Ομιχλώδης επεξεργασία, Επεξεργασία στα άκρα του δικτύου, Υπολογιστικό Νέφος, Διαδίκτυο των Πραγμάτων, MQTT

Abstract

Today, faster than ever, a multitude of networking capable devices invade our daily lives, while more and more everyday objects become “smart”, offering their users new, personalized capabilities and functions by utilizing their internet connectivity. Those aforementioned devices as well as sensors, actuators and systems that connect to the internet and communicate with each other, form what is known as the Internet of Things. All these devices produce enormous amounts of data, whose processing is required to enable their personalized features and their ability to adapt to change in their environment.

Currently, the predominant approach has been processing data utilizing computing clouds to ensure the necessary computing resources, storage, coordination and communication services etc. The rapidly expanding Internet of Things will contribute to an estimated exponential increase of the amount of data being produced, raising questions on the efficiency of transferring the total of all produced data due to the associated network load, and opens the conversation on alternative methods of managing and processing said data.

One such proposal is that of Fog Computing. It proposes part of the required data processing be completed near the point they are produced, utilizing the computing capabilities of numerous nearby devices such as smart phones, laptops, tablets etc. This would reduce total network load as data can at least be pre-processed, minimizing the amount that need to be transferred to the cloud. The growing computing capabilities of common use devices makes local processing a possibility, while such an approach makes use of the aforementioned, largely underutilized devices. Finally, operation and processing in local area networks can achieve significantly lower response times than those of the cloud, an important factor in specific applications, such as in medicine.

In this Diploma Thesis, we study the basic principles and rationale of Fog Computing. Based on them, we design and implement an application that can function as a prototype for the operation of a Fog node.

Keywords: Fog Computing, Edge Computing, Cloud, Internet of Things (IoT), MQTT

Ευχαριστίες

Ευχαριστούμε τον καθηγητή Ιάκωβο Βενιέρη και το Εργαστήριο Ευφυών Επικοινωνιών και Δικτύων Ευρείας Ζώνης για την δυνατότητα που μας δόθηκε να εργαστούμε πάνω στο συγκεκριμένο θέμα. Ιδιαίτερα ευχαριστούμε τον Παναγιώτη Κασνέση και τον Ανδρέα Καψάλη για την βοήθειά τους.

Θα θέλαμε επίσης να ευχαριστήσουμε όλους αυτούς τους ανθρώπους που μας συντρόφεψαν και μας βοήθησαν όλα αυτά τα χρόνια. Ιδιαίτερα θα θέλαμε να ευχαριστήσουμε τους Δ.Φ., Τ.Ο.Π., Κ.Π., Κ.Κ.Ζ., Μ.Μ., Φ.Δ., Ζ.Μ., Π.Η., Π.Ο., Κ.Κ., Ο.Κ., Δ.Γ., Κ.Σ., Λ.Ε., Κ.Ι., Π.Χ., Β.Σ., Πρου καθώς και τους φίλους μας Φρίντα, Νάλα και Κλατς.

Τέλος ευχαριστούμε τους γονείς μας Γιώργο Δημόπουλο και Μαρία Καντζιού, Γιώργο Φαμέλη και Πολύμνια Φαμέλη και τα αδέρφια μας Γιάννη και Μιχάλη, Μαριέττα, Παναγιώτα μαζί με τον Μάρτιν και τον Γιούρι.

“I’ve nothing against farms. Or farmers. You’ve got to have them. It’s just that they used to be a long way off, around the edges.

Now this is the Edge.”

-Cohen the Barbarian

Περιεχόμενα

Ευρετήριο Εικόνων

Ευρετήριο Πινάκων

Κεφάλαιο 1: Εισαγωγή

1.1 Το τοπίο σήμερα

Στην πολλαπλά διασυνδεδεμένη πραγματικότητα του σήμερα, σε κάθε κοινωνική, οικονομική, πολιτική διεργασία μπορούμε να παρατηρήσουμε το αποτύπωμα μιας πληθώρας συσκευών, τεχνικών και μεθόδων που θα μπορούσαμε χονδροειδώς να ονομάσουμε νέες τεχνολογίες. Από τις (επιτυχημένες) προεκλογικές εκστρατείες των δύο τελευταίων προέδρων των ΗΠΑ [1] και τις στρατηγικές της ΕΕ για την διάρθρωση της βιομηχανικής παραγωγής της [2], ως τα κοινωνικά δίκτυα, τους προσωπικούς βοηθούς (όπως η Cortana, η Alexa, Google Assistant κλπ) και τα αυτοοδηγούμενα οχήματα, ένα σημαντικότατο τεχνικό και τεχνολογικό υπόβαθρο που αξιοποιεί τις επεξεργαστικές δυνατότητες των σύγχρονων υπολογιστικών συστημάτων για να παράξει καινοτόμα αποτελέσματα, περνώντας συχνά απαρατήρητο στην καθημερινή ματιά.

Ακριβώς αυτή είναι και η υπόσχεση και το στοίχημα ιδιαίτερα για τις καθημερινές, ευρείας χρήσης εφαρμογές, τις έξυπνες (smart) συσκευές και συστήματα, τηλέφωνα, ρολόγια, τηλεοράσεις αλλά και σπίτια, πόλεις ως το έξυπνο έθνος της Σιγκαπούρης [3]. Η αβίαστη και άκοπη ενσωμάτωσή τους στην ευρύτερη κοινωνική ζωή, για την πλευρά τουλάχιστον των χρηστών, είναι το μέτρο της ευφυίας τους [4]. Οι εξατομικευμένες προτάσεις σε μουσική ταινίες και κάθε είδους διαφήμιση και ενημέρωση από κοινωνικά δίκτυα και προσωπικούς βοηθούς, η έγκαιρη πρόβλεψη και προειδοποίηση για τυχόν κυκλοφοριακή συμφόρηση από τις εφαρμογές πλοήγησης, η προσαρμοσμένη στους ενοίκους πλατφόρμα έξυπνου σπιτιού είναι διαθέσιμα στο ευρύ κοινό, συχνά πριν καν να ζητηθούν από τους χρήστες.

Κάνοντας ένα βήμα πίσω από την κουρτίνα του θαυμαστού αυτού σκηνικού, βλέπουμε πως στον πυρήνα όλων αυτών των διαφορετικών έξυπνων λύσεων βρίσκονται χιλιάδες βαρέλια γεμάτα με το πετρέλαιο του 21ου αιώνα, τα δεδομένα που παράγονται από συσκευές και χρήστες, όπως τα έχει χαρακτηρίσει ο Clive Humbe, Άγγλος μαθηματικός. Είναι αυτός ο τεράστιος όγκος δεδομένων που μέσα από την επεξεργασία τους με κατάλληλες μεθόδους, όπως αλγορίθμους εξόρυξης δεδομένων και μεθόδους μηχανικής μάθησης οδηγεί στην εξαγωγή χρήσιμων ποιοτικών συμπερασμάτων και γνώσης που επιτρέπουν τελικά στις έξυπνες εφαρμογές να προσαρμόζουν συνεχώς τη λειτουργία τους βάσει των πραγματικών συνθηκών και τάσεων.

Προσπαθώντας να δώσουμε μια εικόνα των μεγεθών, τα συστήματα ενός τζετ παράγουν 10 TB δεδομένων κάθε μισή ώρα πτήσης, μία πλατφόρμα πετρελαίου περίπου 500 GB κάθε εβδομάδα [5], ενώ ένας άνθρωπος υπολογίζεται ότι παράγει περίπου 700MB [6]. Αν στα παραπάνω συνυπολογίσουμε την ταχύτατη ανάπτυξη του Διαδικτύου των Πραγμάτων - IoT (Internet of Things) και την σημαντική συμβολή στην παραγωγή δεδομένων από ένα αυξανόμενο πλήθος αισθητήρων, καμερών και άλλων συσκευών, είναι αναμενόμενο να δοκιμασθεί στο άμεσο μέλλον η

αποτελεσματικότητα των μεθόδων επεξεργασίας καθώς τα δεδομένα τα οποία θα την υποστούν ολοένα και αυξάνονται.

Η γενική λογική στην επεξεργασία των παραγόμενων δεδομένων είναι σε πρώτη φάση η συγκέντρωσή τους κεντρικά, ώστε να μπορέσουν να υποστούν συνολική επεξεργασία. Άλλωστε, οι αλγόριθμοι εξόρυξης δεδομένων και η μηχανική μάθηση λειτουργούν βέλτιστα για μεγάλα ποσά δεδομένων από τα οποία εξάγουν ποιοτικά συμπεράσματα. Ο εντοπισμός ανθρώπινων προσώπων σε μια φωτογραφία για παράδειγμα, λειτουργικότητα που διαθέτουν πολλά κοινωνικά δίκτυα, βελτιώνεται συνεχώς αξιοποιώντας τα εκατομμύρια φωτογραφιών που ανεβαίνουν καθημερινά. Η διαδικασία αυτή προϋποθέτει λοιπόν την αποστολή των δεδομένων από το σημείο που παράγονται στους κεντρικούς κόμβους επεξεργασίας, είτε αυτοί είναι εξειδικευμένη ιδιωτική υποδομή, είτε συχνότερα κάποιο cloud (υπολογιστικό νέφος). Ο τεράστιος όγκος των μεταφερόμενων δεδομένων όμως, αυξάνει αναλόγως τον φόρτο εργασίας των δικτύων που αναλαμβάνουν τη μεταφορά τους και η κεντρική συγκέντρωση σημαίνει εκτός τοπικών επιμέρους εκδοχών πως τα δεδομένα αυτά αποστέλλονται μέσω του παγκόσμιου ιστού και απαιτούν αποθηκευτικό χώρο και υπολογιστικούς πόρους από τους κόμβους επεξεργασίας. Με τον ρυθμό που αυξάνεται η παραγωγή δεδομένων της ανθρωπότητας, ανακύπτει εύλογα το ερώτημα όχι μόνο αν ο τρόπος διαχείρισης των δεδομένων είναι αποτελεσματικός, αλλά και κατά πόσον είναι συντηρήσιμος στο άμεσο μέλλον.

1.2 Εναλλακτικές προσεγγίσεις επεξεργασίας στα άκρα του δικτύου

Σε αυτό το έδαφος γεννιέται η συζήτηση για εναλλακτικές προσεγγίσεις στις μεθόδους επεξεργασίας και αξιοποίησης των δεδομένων. Ζητούμενο είναι κατά πόσον μπορεί ένα μέρος της απαραίτητης επεξεργασίας να ολοκληρωθεί πιο κοντά στην πηγή των παραγόμενων δεδομένων. Με τον τρόπο αυτό μπορεί να μειωθεί σημαντικά το ποσό των δεδομένων που ταξιδεύουν στο δίκτυο καθιστώντας την υπάρχουσα δικτυακή υποδομή ικανή να συνεχίσει να υποστηρίζει τις εφαρμογές που αναφέρουμε παραπάνω παρά την συνεχή αύξηση του ποσού των παραγόμενων δεδομένων. Όμως υπάρχουν αρκετοί ακόμα λόγοι για να εμβαθύνουμε σε μια τέτοια προσέγγιση.

Ένα πρώτο άμεσο όφελος είναι πως αν η επεξεργασία γίνεται κοντά στην πηγή των παραγόμενων δεδομένων, εξαλείφεται μέρος των δικτυακών καθυστερήσεων, είτε οι φυσιολογικές, αναμενόμενες μέχρι τα δεδομένα να κινηθούν μέσα στο δίκτυο, όσο και οι επιπρόσθετες όταν το δίκτυο υπερφορτώνεται. Κάτι τέτοιο μπορεί να είναι σημαντικό για εφαρμογές που ο χρόνος απόκρισης του συστήματος είναι σημαντικός όπως για ιατρικές εφαρμογές, αυτοοδηγούμενα οχήματα, εφαρμογές προσωπικού βοηθού που απαντούν σε πραγματικό χρόνο κ.α.

Σε δεύτερη φάση αξίζει κανείς να σκεφτεί το πλήθος των καθημερινών συσκευών με δυνατότητες δικτύωσης και επεξεργασίας που σε μια παραδοσιακή προσέγγιση μένουν αναξιοποίητες. Για παράδειγμα, σε έναν δημόσιο χώρο όπως μία πλατεία,

βρίσκονται ανά πάσα στιγμή δεκάδες άτομα που κατά κανόνα διαθέτουν τουλάχιστον μία έξυπνη συσκευή, με χαρακτηριστικά (επεξεργαστική ισχύ, μνήμη) που είναι συγκρίσιμα με τα αντίστοιχα προσωπικών υπολογιστών. Με κατάλληλη υποδομή, θα μπορούσε μέρος των αναγκαίων επεξεργασιών για τη λειτουργία μίας έξυπνης πόλης να ολοκληρώνεται από αυτές τις συσκευές.

Τέλος, μια προσέγγιση όπως η παραπάνω μειώνει την εξάρτηση από τη σύνδεση με τον παγκόσμιο ιστό για βασικές λειτουργικότητες των συσκευών. Σε ένα αυτοοδηγούμενο όχημα δεν πρέπει η σύνδεση στο διαδίκτυο να είναι προϋπόθεση για την λειτουργία των συστημάτων του αφού μπορεί πολύ συχνά να βρεθεί σε περιοχή χωρίς δικτυακή πρόσβαση.

Η λογική του Fog Computing (Ομιχλώδης Επεξεργασία) είναι μια πρόταση που αξιοποιεί τα παραπάνω στοιχεία και περιγράφει μια δικτυακή λογική συμπληρωματική στη μέχρι τώρα εικόνα. Το Fog προτείνεται σαν ενδιάμεσος, συνδετικός κρίκος μεταξύ των άκρων του δικτύου (edges), δηλαδή συσκευών, αισθητήρων και γενικά πραγμάτων (things), με τους κεντρικούς κόμβους επεξεργασίας στη μορφή του Cloud. Άλλωστε και το όνομά του είναι ο όρος που περιγράφει φαινόμενο που οφείλεται σε χαμηλό σύννεφο, την ομίχλη που είναι η άκρη του σύννεφου.

Το Fog είναι μια δικτυακή αρχιτεκτονική οργανωμένη με επιμέρους κόμβους που επικοινωνούν μεταξύ τους και με το Cloud, εκτελώντας κομμάτι των επεξεργασιών που χρειάζονται στα άκρα του δικτύου, κάνοντας προεπεξεργασία στα δεδομένα που αποστέλλονται στο Cloud και αξιοποιώντας τους διαθέσιμους πόρους, υπό την μορφή καθημερινών συσκευών. Το Fog υπόσχεται να μειώσει τον δικτυακό φόρτο, να αυξήσει την ταχύτητα επεξεργασίας αξιοποιώντας μέχρι πρότινος αναξιοποίητους πόρους, διευρύνοντας έτσι τις ήδη υπάρχουσες δυνατότητες του cloud.

1.3 Αντικείμενο της παρούσας εργασίας

Στην παρούσα εργασία επιχειρούμε να προσεγγίσουμε αρχικά την λογική του Fog Computing. Πρόκειται για σχετικά πρόσφατη τάση και ως εκ τούτου μια εκτενέστερη από το συνηθισμένο βιβλιογραφική αναφορά, τόσο στο Fog όσο και στις συναφείς τεχνολογίες του Cloud και του IoT, θεωρήθηκε απαραίτητη για να χαρτογραφήσουμε κάπως αυτό το νέο πεδίο. Ειδικό βάρος στην έρευνα αυτή έχει μια προσπάθεια τεκμηρίωσης τόσο της πραγματικής δυνατότητας όσο και της χρησιμότητας της προσέγγισης του Fog Computing.

Στη συνέχεια, βασιζόμενοι στην εργασία των Α. Καψάλη, Π. Κασνέση, Ι. Σ. Βενιέρη, Δ. Ι. Κακλάμάνη, Χ. Ζ. Πατρικάκης [7] και συγκεκριμένα την αρχιτεκτονική που περιγράφουν για ένα στοιχειώδες δίκτυο Fog, επιχειρούμε να αναπτύξουμε μια εφαρμογή που θα υλοποιεί την προτεινόμενη αρχιτεκτονική. Επιθυμούμε σε αυτή την προσπάθεια να επιτύχουμε έναν διπλό στόχο.

Αφ’ ενός να έρθουμε αντιμέτωποι με τις προβληματικές που προκύπτουν σε οποιαδήποτε προσπάθεια υλοποίησης ενός πιο αφαιρετικού μοντέλου και άρα με

αυτό τον τρόπο να βελτιώσουμε και να συμπληρώσουμε την αρχική αρχιτεκτονική. Ιδιαίτερο ενδιαφέρον σε αυτή την προσπάθεια έχουν τα στοιχεία της συνάρτησης βαθμολόγησης και της συμμόρφωσης με τις γενικά αποδεκτές αρχές σχεδίασης του Fog.

Αφ' ετέρου θέλουμε να καλύψουμε ένα κενό που υπάρχει σε πρωτότυπα και προσομοιωτές. Καθώς το Fog έχει προταθεί σχετικά πρόσφατα δεν υπάρχουν συγκεκριμένες υλοποιήσεις και εφαρμογές. Με την υλοποίηση της παραπάνω εφαρμογής θα προσπαθήσουμε να παρουσιάσουμε ένα πρωτότυπο που θα μπορεί να εκτελεστεί σε μία σειρά από συσκευές. Έτσι θα μπορούν να γίνουν συγκεκριμένες μετρήσεις και πειράματα σε πραγματικές συνθήκες και να αντληθούν συμπεράσματα για την λειτουργία του Fog.

Τέλος γνώμονα στην εκπόνηση της εργασίας αποτέλεσαν οι ειδικές ανάγκες και απαιτήσεις του Fog. Αυτό σημαίνει ότι σε έναν κόμβο συμμετέχει και συνυπάρχει ένα ευρύ φάσμα διαφορετικών συσκευών και λογισμικών, με συγκεκριμένες δυνατότητες και περιορισμούς. Προσπαθήσαμε στην έρευνα μας, στην σχεδίαση και την υλοποίηση της εφαρμογής να λάβουμε υπόψη τα ειδικά χαρακτηριστικά του Fog.

1.4 Διάρθρωση της Εργασίας

Η παρούσα εργασία διαρθρώνεται συνολικά σε 6 κεφάλαια. Στο κεφάλαιο 2 παρουσιάζουμε το τεχνολογικό υπόστρωμα και τις προβληματικές αυτού, πάνω στο οποίο έρχεται να απαντήσει το Fog, καθώς και τις τεχνολογίες που χρησιμοποιήθηκαν για την εκπόνηση της παρούσας διπλωματικής. Στο τρίτο κεφάλαιο παρουσιάζεται εκτενώς η αρχιτεκτονική στην οποία βασιστήκαμε καθώς και η σχεδίαση της εφαρμογής και των υποσυστημάτων της. Στο τέταρτο κεφάλαιο παρουσιάζεται η εφαρμογή που εν τέλει υλοποιήθηκε και εξηγείται η λειτουργία της. Στο πέμπτο κεφάλαιο παρουσιάζουμε τα αποτελέσματα από ορισμένα πειράματα που εκτελέσαμε και τέλος στο κεφάλαιο 6 υπάρχει μια ανακεφαλαίωση των συμπερασμάτων που αντλήσαμε καθώς και προτάσεις για μελλοντική έρευνα.

Πιο συγκεκριμένα στο δεύτερο κεφάλαιο, εξηγούμε την λογική του Cloud και τα όρια το οποίο αυτό έχει. Στην συνέχεια εξετάζουμε το Διαδίκτυο των Πραγμάτων και γίνεται μια εκτίμηση του κατά πόσο είναι συμβατό με την υπάρχουσα δομή του Fog. Μελετάμε τις διαφορετικές λύσεις που προτείνονται για το παραπάνω πρόβλημα και εστιάζουμε στο Fog, τις βασικές σχεδιαστικές αρχές που έχουν οριστεί για αυτό και κάποια βασικά παραδείγματα χρήσης. Τέλος παρουσιάζουμε τις τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής, παρουσιάζοντας εκτενώς τις λειτουργικότητες και τις απαιτήσεις του MQTT και την βιβλιοθήκη Paho που το υλοποιεί.

Στο κεφάλαιο 3 εξετάζουμε την αρχιτεκτονική της εφαρμογής μας και την σχέση της τις σχεδιαστικές αρχές, όπως αυτές περιγράφθηκαν στο κεφάλαιο 2. Μελετάμε τους ρόλους που εμφανίζονται σε έναν κόμβο Fog και τι λειτουργίες καλύπτουν στο συνολικό δίκτυο του Fog. Τέλος καταπιανόμαστε με την συνάρτηση βαθμολόγησης σε μια προσπάθεια να την επεκτείνουμε καλύπτοντας πραγματικές χρήσεις του Fog

και παρουσιάζουμε τα βασικά σενάρια χρήσης που εμφανίζονται σε ένα τέτοιο δίκτυο.

Το κεφάλαιο 4 αφορά αμιγώς την υλοποίηση της εφαρμογής μας. Παρουσιάζεται ο κώδικας των κλάσεων που την απαρτίζουν μαζί με σχολιασμό για κάθε κλάση και υποσύστημα. Περιγράφουμε τις σχέσεις μεταξύ των κλάσεων με την χρήση διαγραμμάτων και επεξηγούμε την χρήση της εφαρμογής ώστε να μπορεί οποιοσδήποτε ερευνητής να την χρησιμοποιήσει ως προσομοιωτή για τα πειράματά του.

Ορισμένα αποτελέσματα που προκύπτουν από την παραπάνω εφαρμογή παρουσιάζονται στο κεφάλαιο 5 στην μορφή πειραμάτων με σκοπό να δούμε την συμπεριφορά της εφαρμογής μας σε καταστάσεις υπερφόρτωσης και σε σχέση με τους χρόνους του υπολογιστικού νέφους.

Κλείνοντας στο κεφάλαιο 6 γίνεται μια ανακεφαλαίωση και αποτίμηση. Παρουσιάζουμε συνολικά τα αποτελέσματα της παρούσας εργασίας, τα συμπεράσματα που βγάλαμε από την σχεδίαση, ανάπτυξη και χρήση της εφαρμογής. Αναφερόμαστε επίσης σε θέματα που χρήζουν μελλοντικής έρευνας καθώς και ορισμένους προβληματισμούς, όπως αυτά προέκυψαν στην διάρκεια συγγραφής της διπλωματικής εργασίας μας.

Κεφάλαιο 2: Υπόβαθρο

Στο κεφάλαιο αυτό παρουσιάζουμε το αναγκαίο θεωρητικό και τεχνικό υπόστρωμα για να προσεγγίσουμε την έννοια του Fog.

2.1 Υπολογιστικό Νέφος (Cloud)

Ακολουθεί ο ορισμός του National Institute of Standards and technology [8] :

Το Cloud computing είναι ένα μοντέλο που επιτρέπει την πρόσβαση στο δίκτυο σε μια σειρά από κοινόχρηστους ρυθμιζόμενους υπολογιστικούς πόρους (π.χ. δίκτυα, διακομιστές, αποθηκευτικούς χώρους, εφαρμογές και υπηρεσίες) οι οποίοι παρέχονται γρήγορα και απελευθερώνονται με ελάχιστη προσπάθεια διαχείρισης και παρέμβασης από την πλευρά του παρόχου. Το μοντέλο του Cloud αποτελείται από πέντε βασικά χαρακτηριστικά, τρία μοντέλα υπηρεσιών.

2.1.1 Βασικά χαρακτηριστικά

Αυτοεξυπηρέτηση βάσει ζήτησης (on demand). Ένας χρήστης μπορεί να ζητήσει μονομερώς και να αποκτήσει άμεσα υπολογιστικούς πόρους, όπως χρόνο σε κάποιον διακομιστή ή αποθήκευση στο δίκτυο, χωρίς να απαιτείται ανθρώπινη αλληλεπίδραση ή παρέμβαση του παρόχου των υπηρεσιών.

Ευρεία πρόσβαση στο δίκτυο. Οι υπηρεσίες είναι διαθέσιμες μέσω του δικτύου και η πρόσβαση σε αυτές γίνεται μέσω τυποποιημένων μηχανισμών που επιτρέπουν τη χρήση σε ετερογενείς, ειδικές ή γενικές, πλατφόρμες πελατών (π.χ. κινητά τηλέφωνα, tablet, φορητοί υπολογιστές και σταθμοί εργασίας).

Συγκέντρωση πόρων. Οι υπολογιστικοί πόροι του παρόχου συγκεντρώνονται για να εξυπηρετούν πολλούς καταναλωτές χρησιμοποιώντας ένα μοντέλο πολλαπλών μισθώσεων, με διαφορετικούς φυσικούς και εικονικούς πόρους που εκχωρούνται και επανεκχωρούνται δυναμικά ανάλογα με τη ζήτηση των χρηστών. Υπάρχει μια λογική ανεξαρτησίας ως προς την τοποθεσία, με την λογική ότι ο χρήστης γενικά δεν έχει κανέναν έλεγχο ή γνώση σχετικά με την ακριβή τοποθεσία των παρεχόμενων πόρων, αλλά πιθανά να μπορεί να καθορίσει την τοποθεσία σε υψηλότερο επίπεδο αφαίρεσης (π.χ. χώρα, πολιτεία ή κέντρο δεδομένων). Παραδείγματα πόρων είναι η αποθήκευση και επεξεργασία δεδομένων, το εύρος ζώνης δικτύου και η μνήμη.

Ταχεία ελαστικότητα. Οι διαθέσιμοι πόροι μπορούν να δοθούν και να απελευθερωθούν ελαστικά1, σε ορισμένες περιπτώσεις και αυτόματα, και να κλιμακώνονται γρήγορα προς τα πάνω ή προς τα κάτω ανάλογα με τη ζήτηση. Από

1“Ελαστικότητα είναι ο βαθμός στον οποίο ένα σύστημα μπορεί να προσαρμόζεται σε αλλαγές στον φόρτο εργασίας δεσμεύοντας και αποδεσμεύοντας πόρους με αυτόματο τρόπο έτσι ώστε σε κάθε χρονικό σημείο οι διαθέσιμοι πόροι να ταιριάζουν με την παρούσα ζήτηση όσο το δυνατόν πιο πολύ.” [29]

την πλευρά του χρήστη οι διαθέσιμοι πόροι συχνά φαίνονται απεριόριστοι και μπορούν να χρησιμοποιηθούν σε οποιαδήποτε βαθμό ανά πάσα στιγμή.

Μετρούμενη υπηρεσία. Τα συστήματα Cloud ελέγχουν αυτόματα και βελτιστοποιούν τη χρήση των πόρων, αξιοποιώντας τη δυνατότητα μέτρησης σε κάποιο επίπεδο αφαίρεσης κατάλληλο για τον τύπο υπηρεσίας (π.χ. αποθήκευση, επεξεργασία, εύρος ζώνης και ενεργούς λογαριασμούς χρηστών). Η χρήση των πόρων μπορεί να παρακολουθείται, να ελέγχεται και να αναφέρεται, παρέχοντας διαφάνεια τόσο για τον πάροχο όσο και για τον χρήστη της χρησιμοποιούμενης υπηρεσίας.

Υποδομή υπολογιστικού νέφους (cloud infrastructure) αποτελεί το σύνολο υλικού και λογισμικού που έχει τα παραπάνω πέντε χαρακτηριστικά. Η υποδομή μπορούμε να θεωρήσουμε ότι αποτελείται από ένα φυσικό και ένα αφαιρετικό στρώμα. Το φυσικό στρώμα περιλαμβάνει όλη την απαραίτητη υλική υποδομή για τις παρεχόμενες υπηρεσίες Cloud (τυπικά κομμάτια διακομιστή, αποθήκευσης και δικτύου). Τ αφαιρετικό στρώμα αφορά το λογισμικό που τρέχει στο παραπάνω υλικό, μέσω του οποίου υλοποιούνται τα απαραίτητα χαρακτηριστικά του Cloud που αναφέρθηκαν παραπάνω. Το αφαιρετικό επίπεδο βρίσκεται πάνω απο το υλικό σε αυτό το μοντέλο υποδομής.

2.1.2 Μοντέλα υπηρεσιών

Λογισμικό ως Υπηρεσία (Software as a Service(SaaS)). Η δυνατότητα του χρήστη/καταναλωτή να χρησιμοποιεί τις εφαρμογές και υπηρεσίες λογισμικού του παρόχου που εκτελούνται σε μια υποδομή cloud. Οι εφαρμογές είναι προσβάσιμες στις διαφορετικές συσκευές-χρήστες μέσω ειδικών (fat) (συγκεκριμένα προγράμματα) ή γενικών (thin) (περιηγητής ιστού-φυλλομετρητής) διεπαφών. Ο χρήστης δεν έχει δικαιώματα διαχείρισης ή ελέγχου στα στοιχεία υποδομής του cloud, συμπεριλαμβανομένων του δικτύου, των διακομιστών, των λειτουργικών συστημάτων, της αποθήκευσης ή ακόμη και των δυνατοτήτων επιμέρους εφαρμογών. Πιθανή εξαίρεση είναι να μπορεί να επέμβει ο χρήστης κάποιας εφαρμογής σε κάποιες περιορισμένες ρυθμίσεις της.

Πλατφόρμα ως υπηρεσία (Platform as a Service(PaaS)). Η δυνατότητα που παρέχεται στον καταναλωτή να σηκώσει στην υποδομή του cloud εφαρμογές, χρησιμοποιώντας γλώσσες προγραμματισμού, βιβλιοθήκες, υπηρεσίες και εργαλεία που υποστηρίζει ο πάροχος. Και εδώ ο χρήστης δεν έχει δικαιώματα διαχείρισης ή ελέγχου στα στοιχεία υποδομής του cloud, δηλαδή του δικτύου, των διακομιστών, των λειτουργικών συστημάτων και της αποθήκευσης. Έχει όμως έλεγχο πάνω στην εφαρμογή που έχει σηκώσει και πιθανά στις ρυθμίσεις του περιβάλλοντος στο οποίο αυτή τρέχει.

Υποδομή ως υπηρεσία (Infastructure as a Service(IaaS)). Παρέχεται στον καταναλωτή η δυνατότητα να μισθώσει βασικούς υπολογιστικούς πόρους δηλαδή, επεξεργαστική ισχύ, δυνατότητα αποθήκευσης, υποδομή δικτύου κ.α. Σε αυτούς ο καταναλωτής είναι σε θέση να αναπτύξει και να εκτελέσει λογισμικό, το οποίο μπορεί να περιλαμβάνει λειτουργικά συστήματα και άλλες εφαρμογές. Ο καταναλωτής δεν διαχειρίζεται ή ελέγχει την υποδομή του cloud αλλά έχει τον έλεγχο των λειτουργικών

συστημάτων, των αποθηκευτικών χώρων και των εγκατεστημένων εφαρμογών. και ενδεχομένως περιορισμένο έλεγχο επιμέρους/συγκεκριμένων στοιχείων του δικτύου (π.χ. χρήση firewalls).

2.1.3 Τι προσφέρει το Υπολογιστικό Νέφος

Από τον παραπάνω ορισμό φαίνεται ότι το Cloud μπορεί να βοηθήσει με διάφορους τρόπους οργανισμούς και χρήστες. Πρώτα και κύρια δίνει την δυνατότητα εργασίας από παντού εφόσον ο χρήστης δικαιούται πρόσβαση στις υπηρεσίες του cloud. Έτσι μπορεί κανείς να έχει πρόσβαση στην αναγκαία υποδομή για να δουλέψει από τον υπολογιστή του σπιτιού του, το τάμπλετ του ή άλλη φορητή συσκευή και οποιοδήποτε τυχαίο υπολογιστή καθώς η πρόσβασή του στα αναγκαία αρχεία, προγράμματα κλπ εξαρτάται απλά από τα δικαιώματα πρόσβασης που έχει στο cloud και ταυτοποιείται με κάποιον κωδικό. Η ανεξαρτησία από την συσκευή κάνει και την χρήση των υπηρεσιών πιο εύκολη στο μέσο χρήστη καθώς υπάρχει ομοιομορφία ως προς την διεπαφή. Παράλληλα λύνονται και προβλήματα συμβατότητας και μεταφοράς της εργασίας ανάμεσα σε διαφορετικούς χρήστες καθώς όλοι χρησιμοποιούν τις ίδιες εκδόσεις λογισμικού που παρέχονται από το Cloud, κοινό αποθηκευτικό χώρο κλπ με αποτέλεσμα να αυξάνεται η παραγωγικότητα.

Περνώντας σε πιο τεχνικά ζητήματα η υπηρεσίες Cloud αποτελούν εναλλακτική για την υπολογιστική υποδομή που προϋποθέτουν οποιοσδήποτε οργανισμός ή χρήστης για την λειτουργία του. Αυτό αφορά τόσο το κομμάτι του υλικού όπου δεν χρειάζεται να επενδύσει σε ισχυρά μηχανήματα, διακομιστές και αποθηκευτικό χώρο, όσο και στο κομμάτι της τεχνικής διαχείρισης και συντήρησης των παραπάνω, από την εγκατάσταση προγραμμάτων μέχρι σημαντικό κομμάτι των επισκευών. Μέσω της χρήσης του cloud υπάρχει μειωμένος κίνδυνος απώλειας δεδομένων, αρχείων κλπ λόγω αστοχιών του υλικού. Ταυτόχρονα η χρήση του Cloud θεωρείται καλύτερη λύση από πλευράς ασφάλειας των δεδομένων οργανισμών και χρηστών.

Μέσω του Cloud δίνεται πρόσβαση σε υπολογιστικές δυνατότητες που αλλιώς θα ήταν πολύ δύσκολο να έχει κανείς. Η πρόσβαση στο ίντερνετ ισοδυναμεί με πρόσβαση σε τεράστιες υπολογιστικές δυνατότητες ανεξάρτητα από την συσκευή που χρησιμοποιείται για τη σύνδεση στο δίκτυο. Αυτό σημαίνει ότι για παράδειγμα μια ερευνητική ομάδα στην Ανταρκτική μπορεί να επεξεργαστεί μετρήσεις κλπ χωρίς να χρειάζεται να στηθεί τεράστια υποδομή στην τοποθεσία.

2.2 Διαδίκτυο των πραγμάτων και Κυβερνοφυσικά Συστήματα

Το διαδίκτυο των πραγμάτων (IoT) ορίζεται ως το δίκτυο ηλεκτρικών και οικιακών συσκευών, οχημάτων και γενικά αντικειμένων που έχουν ενσωματωμένα ηλεκτρονικά στοιχεία, αισθητήρες, λογισμικό και δυνατότητα διασύνδεσης που τους επιτρέπει να ανταλάσσουν δεδομένα. Κάθε τέτοιο αντικείμενο είναι μοναδικά ταυτοποιήσιμο μέσω των ενσωματωμένων συστημάτων του και μπορεί να συνδέεται στην υπάρχουσα υποδομή του διαδικτύου. Η εισαγωγή αισθητήρων και ενεργοποιητών (actuators) στην λειτουργία δικτύων IoT εντάσσεται στην γενικότερη λογική των κυβερνοφυσικών συστημάτων. Το 2017 ο αριθμός των συσκευών του IoT

ανέρχεται σε 9 δισεκατομμύρια, ενώ οι προβλέψεις για το 2020 είναι 30 δις [9] και μια αγορά γύρω από το IoT με αξία 7,1 τρις δολλάρια [10]

2.2.1 Κυβερνοφυσικα συστηματα

Κυβερνοφυσικό σύστημα είναι ένας μηχανισμός ελέγχου και διαχείρισης όπου υπολογιστικά, εικονικά και φυσικά στοιχεία είναι άρρηκτα και στενά συνδεδεμένα [11]. Τα συστήματα αυτά βρίσκουν ένα μεγάλο εύρος πεδίων εφαρμογής (βιομηχανία, ενέργεια, υγεία, μεταφορές κλπ) βελτιστοποιώντας την λειτουργία τους μέσω αυτοματοποίησης της πρόβλεψης σφαλμάτων και διορθωτικών κινήσεων. Εφαρμογές κυβερνοφυσικών συστημάτων είναι π.χ. ο μηχανισμός πρόβλεψης και αποφυγής συγκρούσεων σε αυτόοδηγούμενα αυτοκίνητα, η ρομποτικά υποβοηθούμενη χειρουργική κα.

Το ίντερνετ των πραγμάτων δίνει τη δυνατότητα απομακρυσμένης πρόσβασης στα διασυνδεδεμένα αντικέιμενα, θολώνοντας την διάκριση φυσικού και ψηφιακού. Ανοίγονται έτσι δυνατότητες για πιο αποτελεσματική, γρήγορη, ακριβή και οικονομική παρέμβαση στη λειτουργεία τους, ενώ μειώνεται η ανάγκη ανθρώπινης παρέμβασης.

2.2.2 Η έννοια του έξυπνου (smart)

Συναφής έννοια με εφαρμογές IoT είναι η έννοια έξυπνο (smart). Ως έξυπνο εννοούμε όχι, κατ' ανάγκη, ένα πιο πολύπλοκο ή σύγχρονο σύστημα διαχείρισης άλλα ότι μέσω της σύμπραξης φυσικού, εικονικού και ψηφιακού η εφαρμογή ενσωματώνεται αβίαστα στην καθημερινότητα και έχει δυνατότητα συνεχούς προσαρμογής στις εκάστοτε πραγματικές συνθήκες. Η υπόσχεση είναι ότι τα συστήματα αυτά ενσωματώνονται σε κάθε περίπτωση σχετικά αβίαστα και οργανικά

Εφαρμογές αυτής της λογικής εμφανίζονται σε πολλούς τομείς, όπως τα έξυπνα πλέγματα ενέργειας (smart grid), όπου αισθητήρες, μετρητές και προσομοιώσεις χρησιμοποιούνται για την αποδοτικότερη λειτουργία και εξυπηρέτηση των πελατών ενός δικτύου ενέργειας [12]. Τα έξυπνα σπίτια (smart homes) όπου οι οικιακές συσκευές μέσω της διασύνδεσης και της ανταλλαγής και επεξεργασίας των δεδομένων τους αυτοματοποιούν τη διαχείριση κλιματισμού, ζεστού νερού χρήσης κτλ. Οι έξυπνες πόλεις (smart cities) με χρήση κυβερνοφυσικών συστημάτων στην τοπική διοίκηση, τις μεταφορές, το οδικό δίκτυο κτλ αποτελούν ένα σημαντικότερο μοντέλο εφαρμογής των τεχνολογιών του IoT και κυβερνοφυσικών συστημάτων, με τα συστήματα αυτά να επικοινωνούν και μεταξύ τους (π.χ. ρύθμιση της κυκλοφορίας και επαναδρομολόγηση δημοσίων μεταφορών βάσει επισκευών στο οδικό δίκτυο). Τέτοια συστήματα υπάρχουν ήδη σε πόλεις στην Σιγκαπούρη [3], στο Αμστερνταμ [13], στο Δουβλίνο [14] κλπ. Ειδική μνεία απαιτεί ίσως το εικονικό εργοστάσιο (smart factory) που αποτελεί και το βασικό μοντέλο που προτείνεται από την ΕΕ στα πλαίσια της 4ης βιομηχανικής επανάστασης σαν μορφή οργάνωσης της παραγωγικής διαδικασίας.

2.2.3 Επεξεργασία των παραγώμενων δεδομένων

Ένα κομμάτι της επεξεργασίας των δεδομένων που παράγονται από ένα δίκτυο IoT έχει πιο μακροπρόθεσμο ορίζοντα και χρησιμοποιείται για την εξαγωγή ποιοτικών συμπερασμάτων από μεγάλο όγκο δεδομένων. Αυτά μπορεί να αφορούν είτε στην ίδια την λειτουργία του συστήματος για την μετέπειτα καλύτερη ρύθμισή του, είτε άλλα χρήσιμα συμπεράσματα (π.χ. εξαγωγή στατιστικών). Αυτή η επεξεργασία περιλαμβάνει τη συγκέντρωσή των δεδομένων σε κάποιο κεντρικό κόμβο, συχνά στο Cloud και την εφαρμογή μεθόδων επεξεργασίας σε αυτά σε δεύτερο χρόνο.

Ένας δεύτερος στόχος της επεξεργασίας των δεδομένων που παράγονται, μπορεί να είναι είτε η άμεση πρόβλεψη και πρόληψη βλαβών είτε η γενικότερη βελτιστοποίηση της λειτουργίας του συστήματος. Συχνά αυτό αφορά την χρήση μεθόδων μηχανικής μάθησης (machine learning) μέσω προσομοιώσεων ή και εκπαίδευσης μέσω των παραγόμενων δεδομένων.

Η χρήση υπολογιστικού νέφους και σε αυτή την περίπτωση μπορεί να καλύψει τις υπολογιστικές ανάγκες, όμως ο εκθετικά αυξανόμενος αριθμός συσκευών, αισθητήρων και γενικά πραγμάτων που συνδέονται στο διαδίκτυο και παράγουν δεδομένα, εγείρει προβλήματα βάσει των περιορισμών του δικτύου [15].

Ενδεικτικά είναι τα συγκεκριμένα μεγέθη. Ένα αυτοδηγούμενο αυτοκίνητο υπολογίζεται ότι θα παράγει περίπου 1 Gigabyte το δευτερόλεπτο. Το έξυπνο δίκτυο ενέργειας των ΗΠΑ εκτιμάται ότι θα παράγει 1000 Petabyte τον χρόνο. Την στιγμή που σήμερα η Google διαχειρίζεται περίπου 1 Petabyte δεδομένα τον μήνα και το δίκτυο της AT&T μετέφερε 200 petabytes το 2010 [16]. Από τα παραπάνω γίνεται προφανές ότι οι νέες συνθήκες που δημιουργεί το IoT είναι πολύ δύσκολα διαχειρίσιμες με τα σημερινά δεδομένα από πλευράς εύρους ζώνης (bandwidth).

Ταυτόχρονα πολλές εφαρμογές IoT απαιτούν μικρούς χρόνους απόκρισης. Για παράδειγμα οι προσωπικοί βοηθοί (Siri, Google assistant κλπ) απαιτείται να απαντούν με φωνητικά μηνύματα σε πραγματικό χρόνο. Ακόμα περισσότερο, σε περιπτώσεις όπως έξυπνα δίκτυα ενέργειας, που βασίζονται σε μοντέλα πρόβλεψης για την κατανάλωση του δικτύου, ή συσκευές για ιατρική χρήση (αισθητήρες που ελέγχουν ασθενείς κλπ), όπου χρειάζεται άμεση ειδοποίηση του ιατρικού προσωπικού, οι μεγάλοι ή απρόβλεπτοι χρόνοι απόκρισης μπορούν να είναι επικίνδυνοι. Ο μέσος χρόνος μετάδοσης μετ’ επιστροφής (round-trip time - RTT) από μία συσκευή στο υπολογιστικό νέφος είναι της τάξης των 100 ms [17]. Ο χρόνος αυτός είναι πολύ κοντά και στον ελάχιστο, δηλαδή στην περίπτωση βέλτιστης λειτουργίας, για αυτές τις συνδέσεις. Τέτοια καθυστέρηση είναι σημαντική για τις εφαρμογές στις οποίες αναφερόμαστε και άρα μη αποδεκτή. Σε πραγματικές συνθήκες δε, έχουμε περιπτώσεις όπου το δίκτυο υπερφορτώνεται ή έχουμε απώλεια πακέτων κλπ, οδηγώντας σε σημαντικά μεγαλύτερους χρόνους καθυστέρησης.

Η εισαγωγή των κυβερνοφυσικών συστημάτων στην ζωή μας υπόσχεται μια ομαλή, αυτορρυθμιζόμενη διαχείριση εργασιών, καθημερινών και μη, χωρίς να απαιτείται συνεχής ανθρώπινη επίβλεψη ή επιμέλεια. Στα κυβερνοφυσικά συστήματα το

κομμάτι των υλικών/φυσικών λειτουργιών έχει μεγαλύτερη βαρύτητα σε σχέση με τα κλασσικά ψηφιακά συστήματα. Επίσης η συνεχιζόμενη λειτουργία δεν είναι πολλές φορές ένα «έξυπνο» χαρακτηριστικό αλλά αναγκαιότητα (π.χ. οι πυρηνικοί αντιδραστήρες έχουν κύκλο λειτουργίας 18 μηνών και κάθε διακοπή έχει κόστος χιλιάδων [18]). Αυτή η απρόσκοπτη λειτουργεία βάζει περιορισμούς στο χρονισμό των αναβαθμίσεων καθώς και των επιθυμητών ενεργειών σε περιπτώσεις μη ομαλής λειτουργίας, όπως βλάβες ή θέματα ασφαλείας.

Ένα ακόμα ζήτημα στη λειτουργία εφαρμογών IoT με βάση το υπολογιστικό νέφος είναι ότι απαιτούν συνεχή πρόσβαση στο διαδίκτυο. Κάτι τέτοιο δεν είναι δεδομένο σε όλες τις περιπτώσεις είτε γιατί υπάρχουν περιοχές που δεν καλύπτονται από το δίκτυο ή σε κατάσταση αστοχίας του δικτύου. Για παράδειγμα ένα αυτοοδηγούμενο αυτοκίνητο που διασχίζει την έρημο θα πρέπει να μπορεί να συνεχίζει απρόσκοπτα τις βασικές του λειτουργίες στην κατάσταση αυτή. Επίσης μια πλατφόρμα εξόρυξης πετρελαίου στην θάλασσα θα πρέπει να συνεχίζει τις λειτουργίες ανάλυσης δεδομένων, ανεξάρτητα από το αν χάνεται η σύνδεση λόγω κακών καιρικών συνθηκών κλπ.

Τέλος ανοίγουν και ζητήματα σε θέματα ασφαλείας. Η κλασική λογική στην ασφάλεια υπολογιστικών συστημάτων είναι η δημιουργία «ζωνών» προστασίας πίσω από τις οποίες τοποθετούνται τα διάφορα συστήματα που θέλουμε να προστατεύσουμε. Για παράδειγμα οι διακομιστές μια εταιρίας προστατεύονται από firewalls και άλλα συστήματα πρόληψης και αποτροπής. Σήμερα κομμάτι των επεξεργαστικά απαιτητικών συστημάτων, όπως τα παραπάνω, μεταφέρεται στο υπολογιστικό νέφος. Για παράδειγμα μεγάλο μέρος της κίνησης του ηλεκτρονικού ταχυδρομείου φιλτράρεται για απειλές στο υπολογιστικό νέφος. Σε περίπτωση που η ασφαλής αυτή περίμετρος παραβιαστεί η κλασική αντιμετώπιση είναι το σύστημα να απενεργοποιείται ώσπου με ανθρώπινη παρέμβαση να εντοπίζεται και να διορθώνεται το πρόβλημα. Αυτή η λογική δεν αντιστοιχεί στο πως η καινούργια κατάσταση του IoT τροποποιεί τα δεδομένα σε ζητήματα ασφαλείας. Αντίθετα ένα πλήθος εφαρμογών απαιτεί πολύ μικρότερη παρεμβατικότητα από τα συστήματα ασφαλείας του. Για παράδειγμα σε περίπτωση που μια μονάδα παραγωγής ηλεκτρικής ενέργειας παραβιαστεί από κακόβουλο λογισμικό που αποσκοπεί στο να κλέψει ενέργεια, είναι περισσότερο ζημιογόνα η διακοπή της λειτουργίας της μονάδας και άρα η αποσταθεροποίηση ολόκληρου του δικτύου ενέργειας.

Συγκεκριμένα ο εκθετικά αυξανόμενος αριθμός των συσκευών που συνδέονται στο διαδίκτυο δυσχεραίνει σε μεγάλο βαθμό την δυνατότητα διαχείρισης τόσο της ταυτοποίησης τους όσο και της επιβεβαίωσης ότι η λειτουργία τους συνεχίζεται υπό ασφαλής, ομαλές συνθήκες. Επίσης καθώς το είδος και η πολυπλοκότητα των επιθέσεων που δέχεται ένα σύστημα και τα αντίστοιχα συστήματα ασφαλείας γίνονται πιο πολύπλοκα και απαιτητικά, συσκευές με περιορισμένους πόρους δεν θα μπορούν να ανταποκριθούν. Για παράδειγμα ένα μέσο αυτοκίνητο έχει κύκλο ζωής γύρω στα 11 χρόνια. Στο διάστημα αυτό το υλικό του θα καταστεί σίγουρα

παρωχημένο και πιθανότατα δεν θα μπορεί να ανταπεξέλθει στις απαιτήσεις του λογισμικού ασφαλείας.

2.2.4 Επεξεργασία στα Άκρα του Δικτύου (Edge Computing)

Ανοίγει έτσι η συζήτηση για μεταφορά κομματιού των λειτουργιών στο άκρο(edge) του δικτύου [19]. Η έννοια άκρο του δικτύου αφορά το λογικό αντίθετο από τους κεντρικούς δικτυακούς κόμβους (core) από τους οποίους περνάει και συγκεντρώνεται μεγάλο κομμάτι της δικτυακής κίνησης [20]. Για να γίνει κατανοητό ως άκρο μπορούμε να θεωρήσουμε ένα κινητό τηλέφωνο, έναν υπολογιστή κλπ. Η επεξεργασία στα άκρα του δικτύου (edge computing) είναι μια μέθοδος βελτιστοποίησης της επεξεργασίας σε υπολογιστικό νέφος εκτελώντας κομμάτι της επεξεργασίας στο άκρο του δικτύου, στην πηγή των δεδομένων, ενώ το υπολογιστικό νέφος χρησιμοποιείται για συντονισμό και αποθήκευση. Με αυτό τον τρόπο μειώνονται τα δεδομένα που μεταφέρονται για να γίνουν οι αναγκαίες επεξεργασίες, απελευθερώνοντας έτσι δικτυακούς πόρους. Για παράδειγμα ένα wearable μπορεί να κάνει μια πρώτη επεξεργασία και να εμφανίζει αποτελέσματα στο κινητό του χρήστη πριν αποθηκευτούν και επεξεργαστούν μακροπρόθεσμα στο υπολογιστικό νέφος.

Παρ’ όλα αυτά, και η επεξεργασία στα άκρα του δικτύου έχει αρκετές ελλείψεις. Καταρχάς οι άκρες τείνουν να είναι συσκευές που περιορίζονται από την μπαταρία ή την μνήμη τους αν και μπορεί να έχουν την αναγκαία υπολογιστική ισχύ. Έτσι μπορεί να δημιουργηθεί πρόβλημα όταν μια συσκευή προσπαθεί να εξυπηρετήσει πολλά συστήματα IoT. Επίσης δεν υπάρχουν επαρκώς δοκιμασμένες πλατφόρμες διαχείρισης και ανάπτυξης γενικών εφαρμογών για επεξεργασία στα άκρα του δικτύου, σαν τις αντίστοιχες για υπολογιστικό νέφος όπως εικονικοποιήση(virtualization), αρχιτεκτονικές με υπηρεσίες (service-based architecture) κα. Οι εφαρμογές για επεξεργασία στα άκρα του δικτύου λοιπόν αναπτύσσονται ειδικά και κατά περίπτωση.

2.3 Ομιχλώδης Επεξεργασία (Fog Networking)

Μια λογική που έρχεται να δώσει μια απάντηση στις παραπάνω προβληματικές είναι αυτή του Fog Computing που πρωτοεισήχθει από τον Flavio Bonomi et al. [21] το 2012 και υιοθετήθηκε από την Cisco ως ένα βοηθητικό στρώμα μεταξύ των συσκευών στα άκρα του δικτύου και του υπολογιστικού νέφους. Με αυτό τον τρόπο επιτυγχάνονται χαμηλότερες και πιο προβλέψιμες καθυστερήσεις. Το στρώμα του Fog δίνει καλύτερη δυνατότητα διαχείρισης των πόρων σαν υπηρεσίες και την δυνατότητα για λειτουργία σύμφωνα με τις αρχές και με παρόμοια λογική του υπολογιστικού νέφους. Στην ουσία οι κόμβοι του Fog λειτουργούν σαν μικρότερα επιμέρους κέντρα δεδομένων που βρίσκονται κοντά στα άκρα.

2.3.1 Γενικη Περιγραφη

Δομική μονάδα του Fog, είναι ο κόμβος (Fog node). Αυτός πρόκειται ουσιαστικά για ένα επιμέρους δίκτυο το οποίο αποτελείται από αισθητήρες, συσκευές IoT, φορητούς υπολογιστές, κινητά και πράγματα(things). Σκοπός κάθε τέτοιου κόμβου είναι να

συντονίζει την ροή των παραγόμενων δεδομένων, ξεδιαλέγοντας ποια από αυτά χρειάζονται να προωθηθούν στο υπολογιστικό νέφος και κατανέμοντας κατάλληλα την επεξεργασία που μπορεί να γίνει τοπικά. Με τον τρόπο αυτό επιτυγχάνεται ταχύτερη και σταθερότερη απόκριση, όταν υπάρχει ευαισθησία στον χρόνο και μειώνεται ο φόρτος ευρύτερα στο δίκτυο.

Η παραπάνω αρχιτεκτονική μπορεί να επιτελέσει λειτουργείες αποθήκευσης και επεξεργασίας δεδομένων με αποτελεσματικό τρόπο καθώς και να ενισχύσει τις δυνατότητες και την διαχείριση τοπικών δικτύων ελαφρύνοντας σημαντικά τον φόρτο συνολικά στο δίκτυο. Κομμάτι της κίνησης των δεδομένων που αφορά συσκευές-μέρη ενός κόμβου Fog μπορεί να πραγματώνεται εσωτερικά αυτού χωρίς την ανάγκη να περνάει από την ραχοκοκαλιά (backbone) του δικτύου επιβαρύνοντάς το.

Η αρχιτεκτονική του Fog δεν είναι ξεκομμένη από το Cloud. Αντιθέτως ο σκοπός είναι να συνδέονται και να αλληλοστηρίζονται. Για παράδειγμά το Fog μπορεί να λειτουργεί σαν τον κόμβο συγκέντρωσης των δεδομένων που θα προωθηθούν στο υπολογιστικό νέφος, ενώ το υπολογιστικό νέφος μπορεί να χρησιμοποιηθεί για την διαχείριση των διάφορων κόμβων του Fog. Προφανώς υπάρχουν λειτουργίες στις οποίες το υπολογιστικό νέφος υπερτερεί σε σχέση με το Fog και το αντίστροφο. Ενώ η υποδομή ενός υπολογιστικού νέφους απαιτεί εξοπλισμό, χώρο, τεράστιες ενεργειακές απαιτήσεις και εξειδικευμένο προσωπικό για να συντηρηθεί και να λειτουργήσει, το Fog προτείνεται ως ένα μοντέλο το οποίο σε πολλές περιπτώσεις θα μπορεί να στηθεί απλά με όποιες συσκευές υπάρχουν στον χώρο. Επίσης το μέγεθός του μπορεί να είναι από πολύ μικρό, όπως οι συσκευές ενός διαμερίσματος ή ένα αυτοκίνητο με τις συσκευές των επιβατών του, έως το σύστημα διαχείρισης μίας μονάδας παραγωγής ηλεκτρισμού ή ενός εργοστασίου. Ακόμα τόσο σε περιπτώσεις κυβερνοφυσικών συστημάτων όσο και σε καταστάσεις που η δικτυακή πρόσβαση δεν είναι ομαλή η εξασφαλισμένη και όπως αναφέρεται παραπάνω το υπολογιστικό νέφος δεν ενδείκνυται για την επεξεργασία και διαχείριση των συστημάτων αυτών, το Fog μπορεί να καλύψει τις όποιες αδυναμίες.

Το Fog μπορεί να πετύχει μικρότερες καθυστερήσεις σε μεταφορά και επεξεργασία από το να στέλνονται τα δεδομένα στο υπολογιστικό νέφος και πίσω. Όμως ένα κομμάτι των παραγωμένων δεδομένων χρειάζεται να προωθείται στο υπολογιστικό νέφος, για μακροπρόθεσμη αποθήκευση, βαθιά εξόρυξη δεδομένων κτλ. Το Fog βρίσκεται πιο κοντά στο άκρο του δικτύου και άρα έχει εικόνα για τις συγκεκριμένες απαιτήσεις και δυνατότητες των συσκευών εκεί, οπότε μπορεί να βελτιστοποιήσει αυτή η διαλογή καθώς και την κατανομή εργασιών στις συσκευές που είναι συνδεδεμένες στο Fog βάσει των συγκεκριμένων τους αναγκών. Ένα τέτοιο δίκτυο μπορεί να χρησιμοποιήσει την πληθώρα συσκευών που βρίσκονται στα άκρα του δικτύου και σήμερα ως επί των πλείστων αξιοποιούνται από ελάχιστα ως καθόλου. Αν σκεφτούμε τις επεξεργαστικές δυνατότητες που έχουν τα σημερινά κινητά τηλέφωνα, τα τάμπλετ και οι υπολογιστές που υπάρχουν σε ένα διαμέρισμα για παράδειγμα, καταλαβαίνουμε ότι η πρόταση του Fog για επεξεργασία στα άκρα του

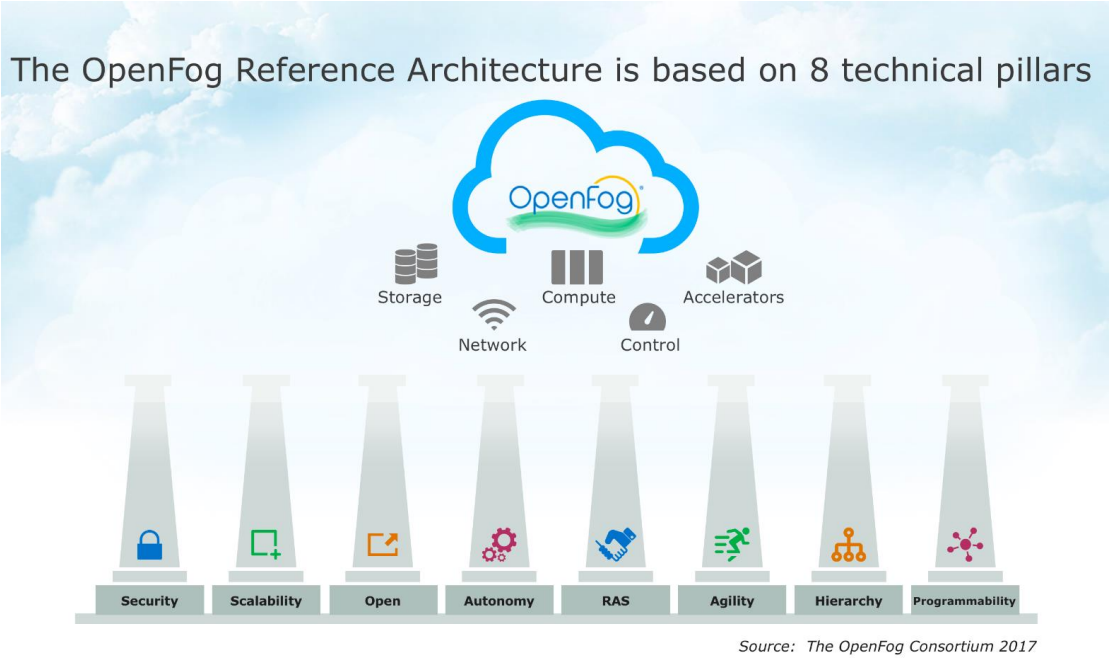
δικτύου είναι εφικτή. Σημαντική είναι η ευελιξία και η μεταβλητότητά που επιδεικνύει ένα τέτοιο δίκτυο καθώς συσκευές με ποικιλόμορφα χαρακτηριστικά και δυνατότητες μπορούν να συνδέονται και να αποσυνδέονται ανά πάσα στιγμή και το δίκτυο να προσαρμόζεται στις αλλαγές αυτές. Σε ένα τέτοιο περιβάλλον οι όποιες αλλαγές δεν εξαρτώνται από τους μεγάλους πάροχους που διαχειρίζονται τα υπολογιστικά νέφη. Αντιθέτως το Fog ενδείκνυται για την χρήση και ανάπτυξη ανοιχτών υπηρεσιών και εφαρμογών ανεξάρτητα από την υιοθέτηση αυτών από το κεντρικό υπολογιστικό νέφος [15].

Προσπαθώντας να αποσαφηνίσουμε τα παραπάνω με ένα παράδειγμα, μπορούμε να σκεφτούμε σαν μια ενδεικτική εφαρμογή Fog, ένα κόμβο που δομείται γύρω από μία στάση λεωφορείου, ή μια πλατεία, σαν μέρος του δικτύου μιας έξυπνης πόλης. Στον κόμβο συγκεντρώνονται δεδομένα από έξυπνες συσκευές, αισθητήρες κλπ που υπάρχουν τοποθετημένοι ή κινούνται στην εμβέλεια του κόμβου. Τέτοιοι μπορεί να είναι μετρητές ταχύτητας ή ροής οχημάτων, βοηθητικά συστήματα διαχείρισης αποκριμάτων, έξυπνα ποτιστικά συστήματα αλλά και στατιστικά για τους διερχόμενους περαστικούς που συλλέγονται μέσω των έξυπνων συσκευών τους. Όλα τα παραπάνω δεδομένα συλλέγονται και υφίστανται προεπεξεργασία από την οποία προκύπτουν χρήσιμα πορίσματα. Μέρος αυτών μπορεί να προωθείται και κεντρικά στο υπολογιστικό νέφος του δήμου αλλά μέρος της επεξεργασίας ολοκληρώνεται τοπικά, μειώνοντας το φόρτο στο δίκτυο και παρέχοντας αποτελέσματα συντομότερα από ότι αν τα δεδομένα αποστέλλονταν μόνο κεντρικά στο υπολογιστικό νέφος. Στην περίπτωση του Fog, η τοπική αυτή επεξεργασία μπορεί να συμβαίνει στις έξυπνες συσκευές των περαστικών, στους υπολογιστές καταστημάτων μέσα στην εμβέλεια του κόμβου κλπ.

2.3.2 Η αρχιτεκτονική του Open Fog Consortium

Καθώς το Fog αποτελεί ένα μοντέλο που έχει προταθεί τα τελευταία χρόνια και δεν έχει γνωρίσει ακόμα ευρύτερη εφαρμογή, είναι λογικό ότι δεν έχει και συγκεκριμένες προτυποποιήσεις με βάση τις οποίες αναπτύσσονται οι όποιες εφαρμογές. Αυτό το κενό προσπαθεί να καλύψει το Open Fog Consortium. Μία κοινοπραξία την οποία ίδρυσαν οι Cisco, Intel, Microsoft, Dell, Arm και το πανεπιστήμιο του Princeton το 2015 με σκοπό την διάδοση του Fog και την τυποποίηση του τρόπου ανάπτυξης, εγκατάστασης και λειτουργίας του σε διάφορα πεδία. Το Open Fog Consortium έχει εκδώσει την τεχνική έκθεση (White Paper) “OpenFog Reference Architecture for Fog Computing” [22]. Σε αυτήν περιγράφονται τα βασικά οφέλη του Fog, το πως εισάγεται και αλληλοεπιδρά με το ήδη υπάρχον τεχνικό πλαίσιο, βασικές αρχές λειτουργίας, εσωτερική διάρθρωση του δικτύου Fog κτλ.

2.3.3 Βασικοί Πυλώνες του OpenFog Reference Architecture



Εικόνα 2.1 [22] - Πυλώνες του OpenFog Reference Architecture

Στην παραπάνω τεχνική έκθεση περιγράφονται 8 βασικοί πυλώνες οι οποίοι αντιπροσωπεύουν τα κομβικά χαρακτηριστικά που πρέπει να πληροί ένα σύστημα Fog. Αυτοί είναι:

I. Ασφάλεια (Security). Επειδή τα δίκτυα Fog μπορεί να είναι εξαιρετικά ανομοιογενή και οι συσκευές που συμμετέχουν σε αυτά ετερόκλητες η ασφάλεια έχει αναβαθμισμένο ρόλο. Τόσο για την προστασία του ίδιου του Fog όσο και γιατί μέτρα ασφαλείας σε επίπεδο Fog μπορούν να αποτελέσουν ένα αποτελεσματικό προστατευτικό πλέγμα γύρω από το υπολογιστικό νέφος. Ακόμα πολλές φορές τα δεδομένα που διακινούνται σε επίπεδο Fog προέρχονται από εφαρμογές IoT και μπορεί να είναι προσωπικά και ευαίσθητα.

II. Κλιμάκωση (Scalability). Ο πυλώνας αυτός αναφέρεται στις ρευστές ανάγκες και μεταβαλλόμενες που έχουν οι τεχνικές και επιχειρησιακές εφαρμογές Fog. Η ελαστικότητα, χαρακτηριστικό του υπολογιστικού νέφους, συνεχίζει να υπάρχει και στο επίπεδο του Fog καθώς με την χρήση νέων ή την αποδέσμευση των υπαρχόντων πόρων(συσκευές κλπ) αυξομειώνονται οι δυνατότητες του Fog σύμφωνα με την ζήτηση. Επιπροσθέτως οι επιμέρους κόμβοι Fog μπορούν να κλιμακώνονται εσωτερικά καθώς σε αυτούς προστίθενται νέο υλικό και λογισμικό. Ακόμα η ζήτηση μπορεί να καλυφθεί με οριζόντια κλιμάκωση μέσω της προσθήκης νέων γειτονικών κόμβων σε περιπτώσεις που υπάρχει μεγάλο υπολογιστικό φορτίο. Τέλος οι δυνατότητες αποθήκευσης, διασύνδεσης και ανάλυσης δεδομένων κλιμακώνονται μαζί με την υπόλοιπη υποδομή του δικτύου Fog.

III. Ανοιχτότητα (Open). Καθώς το Fog αφορά ένα ευρύ φάσμα διαφορετικών εφαρμογών θα πρέπει τα πρωτόκολλα και οι τυποποιήσεις στις οποίες βασίζεται να είναι ανοιχτές. Δεν είναι πρακτική η σύνθεση διαφορετικών κλειστών προτύπων μεταξύ τους για να μπορέσουν να συνθέσουν το δίκτυο Fog που θα στηρίζεται στην συνεργασία πολλών επιμέρους κόμβων. Με αυτό τον τρόπο δίνεται η δυνατότητα σε κάθε συσκευή και χρήστη να συνδέεται στο καταλληλότερο για αυτόν σημείο στην τοπολογία του δικτύου Fog (πιο κοντά ή μακριά από το υπολογιστικό νέφος , ανάγκες για περισσοτέρους ή λιγότερους υπολογιστικούς πόρους/χώρο αποθήκευσης κλπ)

IV. Αυτονομία (Autonomy). Ο πυλώνας αυτός αφορά την δυνατότητα του επιμέρους κόμβου Fog να συνεχίζει τη λειτουργία του σε περιπτώσεις που αστοχούν διάφορες εξωτερικές υπηρεσίες (σύνδεση με το υπολογιστικό νέφος, άλλους κόμβους κλπ). Αυτό σημαίνει ότι ανεξάρτητα με την θέση του κόμβου Fog στην ευρύτερη ιεραρχία του δικτύου, μπορεί να επεξεργάζεται τα παραγόμενα δεδομένα και να παίρνει αποφάσεις ανεξάρτητα από την σύνδεση με το υπολογιστικό νέφος και την πρόσβαση του σε ευρύτερα δεδομένα. Με αυτό τον τρόπο κάθε απόφαση δεν είναι ανάγκη να παίρνεται σε επίπεδο κεντρικού υπολογιστικού νέφους, ενώ ταυτόχρονα είναι δυνατόν να παίρνονται αποφάσεις όλο και πιο κοντά στην άκρη του δικτύου, όπως είναι λογικό από επιχειρησιακή σκοπιά, εκεί δηλαδή που παράγονται τα δεδομένα.

V. Αξιοπιστία – Διαθεσιμότητα – Επισκευασιμότητα (Reliability – Availability – Serviceability RAS). Στον πυλώνα αυτόν συμπεριλαμβάνονται λειτουργείες όπως την ασφάλεια, διαθεσιμότητα και ακεραιότητα των δεδομένων που αποθηκεύονται σε κόμβους Fog. Την χρήση αλγορίθμων μηχανικής μάθησης, εντοπισμού και απομόνωσης βλαβών έτσι ώστε να μειώνεται ο μέσος χρόνος επισκευής (MTTR). Την αυτοματοποίηση της εγκατάστασης, της ενημέρωσης και της επισκευής των συστημάτων Fog ώστε να μπορούν να υλοποιηθούν σε διαφορετικές περιπτώσεις και με δυνατότητα κλιμάκωσης κ.α.

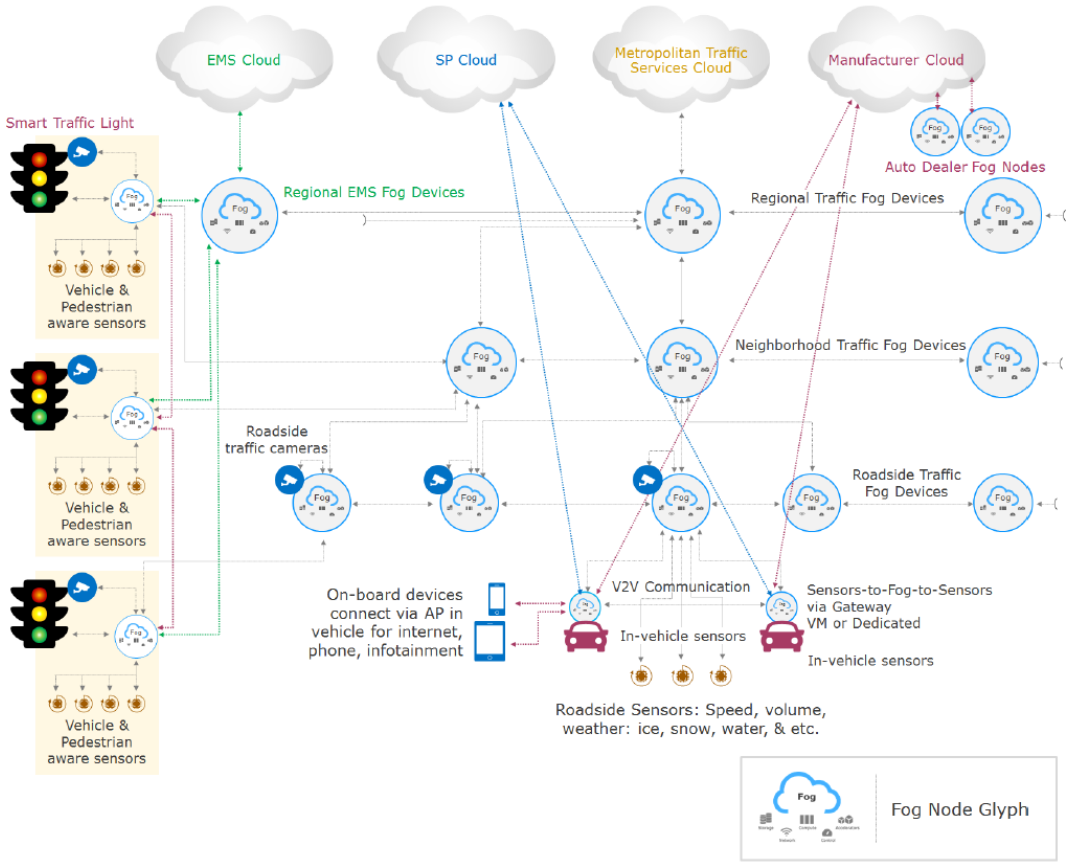
VI. Ευελιξία (Agility). Ο πυλώνας της ευελιξίας αφορά την άμεση εξαγωγή χρήσιμων συμπερασμάτων που υπαγορεύουν διορθωτικές ενέργειες, χρονικά συντομότερα και στην μικρότερη δυνατή απόσταση από τις συσκευές που παράγουν τα δεδομένα. Με τον τρόπο αυτό μειώνεται το βάρος στο δίκτυο, η εξάρτηση από το υπολογιστικό νέφος καθώς και δίνεται η δυνατότητα να παρθούν αποφάσεις και να προσαρμοστεί το δίκτυο σαν άμεση αντίδραση σε αλλαγές των συνθηκών.

VII. Ιεραρχία (Hierarchy). Η ιεραρχία (συστημάτων ή υπολογιστική) δεν είναι απαραίτητο ότι εμφανίζεται σε κάθε εφαρμογή Fog, αλλά υποστηρίζεται. Συμπληρώνει την παραδοσιακή αρχιτεκτονική του υπολογιστικού νέφους, επιτρέποντας έτσι την διασύνδεση και από κοινού λειτουργία Fog με αυτό. Οι πόροι που συμμετέχουν σε ένα ευρύτερο δίκτυο Fog μπορούν να ιδωθούν σαν μια λογική ιεραρχία βάσει των λειτουργικών αναγκών ενός συστήματος IoT, από τα πράγματα (things) μέχρι το ανώτερο επίπεδο επεξεργασίας δεδομένων και λήψης αποφάσεων. Ανάλογα την κλίμακα της εφαρμογής, η

λογική αυτή ιεραρχία μπορεί να πραγματώνεται σαν ένα ενιαίο σύστημα – κόμβος ή περισσότερα σε ιεραρχική σχέση μεταξύ τους.

VIII. Προγραμματισιμότητα (Programmability). Οι κόμβοι του Fog θα πρέπει να είναι στην θέση να προγραμματιστούν σε επίπεδου υλικού και λογισμικού. Δημιουργείται έτσι ένα δίκτυο που είναι προσαρμοστικό και άρα μπορεί να επιτελεί διαφορετικές λειτουργίες. Δυνατότητα που είναι αναγκαία στην περίπτωση του Fog καθώς ένας κόμβος μπορεί να εξυπηρετεί διαφορετικές λειτουργίες ανά ώρα και μέρος.

Όπως είπαμε και παραπάνω δομική μονάδα του Fog είναι ο κόμβος. Ο διαχωρισμός σε κόμβους αφορά γεωγραγραφικούς/χωροταξικούς περιορισμούς αλλά μπορεί να γίνεται και στην βάση διαφορετικών λειτουργιών. Για παράδειγμα στο σχήμα 2.2 οι κόμβοι γύρω από τα φανάρια βασικά συλλέγουν δεδομένα από μία περιοχή που τους αντιστοιχεί, οι κόμβοι στα ανώτερα επίπεδα (πιο μακριά από το άκρο του δικτύου) διακρίνονται βάσει διαφορετικών λειτουργιών , ενώ ένας κόμβος-αυτοκίνητο χαρακτηρίζεται τόσο χωροταξικά όσο και λειτουργικά.

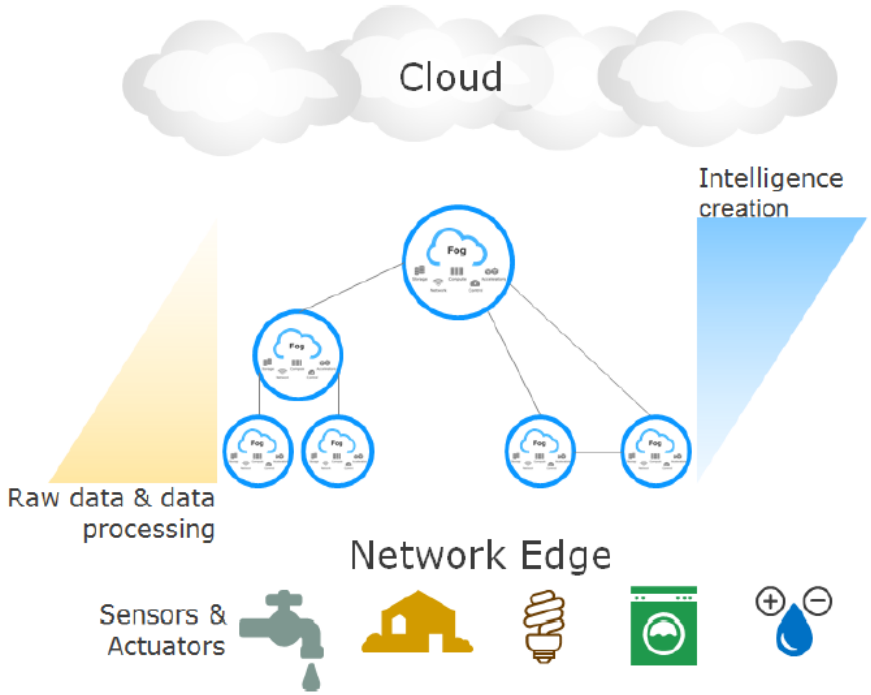


Εικόνα 2.2 [22] - Παράδειγμα Δικτύου Ομιχλώδους Επεξεργασίας για την διαχείριση του κυκλοφοριακού σε μια έξυπνη πόλη

Η δικτυακή δομή που προτείνεται από το Open Fog Consortium περιλαμβάνει περισσότερα από ένα επίπεδα Fog καθένα από τα οποία απαρτίζεται από περισσότερους από έναν κόμβο, όπως φαίνεται στο σχήμα 2.2. Οι κόμβοι δεν είναι απαραίτητο ότι επικοινωνούν μόνο με συγκεκριμένους άλλους κόμβους αλλά ανά

περίπτωση και στόχο που χρειάζεται να πετύχουν επικοινωνούν με τους κατάλληλους ανωτέρων και κατωτέρων επιπέδων, αναζητούν πόρους από κοντινούς κόμβους κλπ. Δεν υπάρχει δηλαδή δεδομένη και σταθερή ιεραρχία και δικτυακή δομή, αλλά όπως προκύπτει και από τους παραπάνω πυλώνες σχεδίασης του Fog, ένας κόμβος μπορεί να προσαρμόζει την λειτουργία του, τους πόρους του και την σχετική του θέση (δικτυακά) με άλλους κόμβους, ανάλογα με τις ανάγκες των εργασιών που αναλαμβάνει.

2.3.4 Η Λογική του Fog N-Επιπέδων (N-tier Fog)



Εικόνα 2.3 [22] - Fog N-Επιπέδων

Παρότι, λοιπόν, δεν υπάρχει μόνιμος καταμερισμός εργασιών μεταξύ των κόμβων του Fog, για κάθε εφαρμογή που αξιοποιεί τεχνολογίες Fog προκρίνεται η ύπαρξη διαφορετικών λειτουργικών επιπέδων, περισσότερα ή λιγότερα ανάλογα με την περίσταση, N-tier Fog (Fog N-επιπέδων). Κόμβοι κοντά στα άκρα του δικτύου τυπικά εστιάζουν στην συλλογή των παραγόμενων από αισθητήρες δεδομένα, στην κανονικοποίηση αυτών και στον έλεγχο αισθητηρών και ενεργοποιητών. Κόμβοι σε ενδιάμεσα επίπεδα εστιάζουν σε ένα πρώτο φιλτράρισμα, κατηγοριοποίηση, ταξινόμηση και συμπίεση των δεδομένων, ενώ μπορεί να αναλαμβάνουν και κομμάτι του ελέγχου. Τέλος τα υψηλότερα επίπεδα πιο κοντά στο υπολογιστικό νέφος εστιάζουν στην εξαγωγή ποιοτικών συμπερασμάτων και την παραγωγή γνώσης από τα παραγόμενα δεδομένα. Αξίζει να σημειώσουμε ότι όσο απομακρυνόμαστε από την άκρη του δικτύου τόσο περισσότερα ποιοτικά συμπεράσματα μπορούμε να εξάγουμε.

2.4 MQTT

Το MQTT (Message Queuing Telemetry Transport) [23] είναι ένα πρωτόκολλο του Data επιπέδου και λειτουργεί πάνω από TCP. Είναι ελαφρύ με μικρό footprint (χώρο που τρώει στην RAM), γι' αυτό είναι κατάλληλο για χρήση σε συσκευές περιορισμένων δυνατοτήτων και δίκτυα χαμηλού bandwidth ή/και υψηλού latency, συνεπώς σε εφαρμογές και δίκτυα IoT.

2.4.1 Μοντέλο Δημοσίευσης-Συνδρομής (Publish-Subscribe)

Το MQTT βασίζεται στο μοντέλο Publish-Subscribe (pub/sub) (Δημοσίευσης-Συνδρομής). Το συγκεκριμένο μοντέλο στηρίζεται στην λογική ότι οι αποστολείς μηνυμάτων (publishers) δεν γνωρίζουν απευθείας τον παραλήπτη του κάθε μηνύματος. Εν αντιθέσει κατηγοριοποιούν τα μηνύματά τους με κάποιον συγκεκριμένο τρόπο και τα κάνουν publish (δημοσιεύουν) με βάση την κατηγορία τους. Αντίστοιχα οι αποδέκτες των μηνυμάτων (subscribers) δηλώνουν το ενδιαφέρον τους για κάποια συγκεκριμένη κατηγόρια και λαμβάνουν τα αντίστοιχα μηνύματα.

Η κατηγοριοποίηση των μηνυμάτων γίνεται βασικά με δύο τρόπους:

1. Βάσει του θέματος (subject-based filtering). Σε αυτό τον τρόπο, κάθε μήνυμα γίνεται published σε κάποιο συγκεκριμένο topic (θέμα) και λαμβάνεται από όλους τους subscribers (συνδρομητές) που έχουν δηλώσει ενδιαφέρον στο topic αυτό. Με τον όρο topic περιγράφετε μια αυθαίρετη κατηγοριοποίηση με βάση την οποία κατευθύνεται η ροή των μηνυμάτων. Στην περίπτωση αυτή, οι publishers είναι αυτοί που είναι υπεύθυνοι για τον ορισμό των κατηγοριοποιήσεων/topics.
2. Βάσει του περιεχομένου (content-based filtering). Σε αυτό τον τρόπο, κάθε μήνυμα φέρει κάποια συγκεκριμένα χαρακτηριστικά και οι subscribers δηλώνουν τις προτιμήσεις τους σε χαρακτηριστικά. Στην περίπτωση αυτή, οι subscribers είναι ουσιαστικά υπεύθυνοι για το φιλτράρισμα των μηνυμάτων.

Φυσικά μπορεί να χρησιμοποιηθεί και ο συνδυασμός των δύο παραπάνω μεθόδων. Δηλαδή μηνύματα που φέρουν χαρακτηριστικά αλλά γίνονται publish και σε topics και αντίστοιχα subscribers που δηλώνουν ενδιαφέρον σε συγκεκριμένα χαρακτηριστικά, εντός συγκεκριμένων topics.

Συνηθισμένη πρακτική όταν χρησιμοποιούνται αντικειμενοστραφείς γλώσσες είναι και το φιλτράρισμα βάσει τύπου. Για παράδειγμα μπορεί ένας subscriber να επιθυμεί να λαμβάνει όλα τα μηνύματα που αφορούν Exceptions.

Κάθε pub/sub σύστημα απαιτεί έναν τρόπο διαχείρισης της κίνησης των διαφόρων μηνυμάτων. Την διαχείριση αυτή την αναλαμβάνει συνήθως ένας ενδιάμεσος κόμβος που ονομάζεται broker (μεσάζοντας), στον οποίο αποστέλλονται τα μηνύματα και ο οποίος έχει γνώση για τα subscriptions που έχουν γίνει. Με αυτό τον τρόπο, ο broker είναι υπεύθυνος να αναμεταδώσει τα μηνύματα που του έρχονται σε όποιον παραλήπτη έχει κάνει subscribe στα αντίστοιχα topics. Η επικοινωνία προϋποθέτει την ύπαρξη ενός broker και άρα κάθε επικοινωνία αποτελείται από την αποστολή του μηνύματος από έναν client-publishser στο broker και από τον broker σε όποιον client-subscriber αφορά το μηνυμα.

Όπως αναφέρθηκε παραπάνω, το pub/sub μοντέλο στηρίζεται στην απόζευξη

αποστολέα και παραλήπτη, στην έλλειψη γνώσης δηλαδή του ενός για τον άλλον. Αυτό μπορεί να αφορά:

1. Τοπική απόζευξη (space decoupling): ο παραλήπτης και ο αποστολέας δεν γνωρίζουν ο ένας τον άλλον στην τοπολογία του δικτύου (π.χ. ip, port).
2. Χρονική απόζευξη (time decoupling): ο παραλήπτης και ο αποστολέας δεν χρειάζεται να λειτουργούν την ίδια χρονική περίοδο.
3. Απόζευξη συγχρονισμού (synchronization decoupling): οι λειτουργίες publish και subscribe του παραλήπτη και του αποστολέα δεν συγκρούονται με την κανονική τους λειτουργία.

2.4.2 MQTT

Όσον αφορά το MQTT, χρησιμοποιεί subject-based filtering με χρήση topics και μπορεί να υλοποιήσει και τις 3 παραπάνω περιπτώσεις απόζευξης. Συγκεκριμένα η διαμεσολάβηση ενός broker στην επικοινωνία ανάμεσα στους publishers και τους subscribers επιτυγχάνει την τοπική απόζευξη. Η χρονική απόζευξη μπορεί να υλοποιηθεί με αποθήκευση των μηνυμάτων στον broker για έναν offline client. Η συγκεκριμένη λειτουργεία υλοποιείται από τους περισσοτέρους brokers, όπως και από τον Mosquitto. Τέλος, υπάρχουν πολλές βιβλιοθήκες για υλοποίηση των clients που δίνουν την δυνατότητα απόζευξης συγχρονισμού, όπως και η Paho.
Παρακάτω παρουσιάζονται οι λειτουργικότητες που υλοποιεί το MQTT και το κάνει αρκετά ελκυστικό για την χρήση σε fog computing.

2.4.3 Θέματα (Topics)

Στο MQTT τα topics είναι μια συμβολοσειρά (σε κωδικοποίηση UTF-8) μπορούν να δομούνται σε περισσότερα από ένα επίπεδα που χωρίζονται μεταξύ τους με “/” (forward slash). Η δομή αυτή μοιάζει αρκετά με μια δομή φακέλων (file system) σε ένα λειτουργικό σύστημα. Για να κάνει ένας client publish ή subscribe δεν χρειάζεται να προϋπάρχει το topic. Αντίθετα κάθε topic δημιουργείται αυτόματα με την αποστολή ενός μηνύματος σε αυτό ή με το subscription σε αυτό. Αυτή είναι μια λειτουργικότητα του MQTT που το βοηθάει να είναι ελαφρύ χωρίς να επιβαρύνει τους clients με ελέγχους για το αν υπάρχουν τα topics κλπ. Κάθε συμβολοσειρά για να είναι έγκυρο όνομα θα πρέπει να έχει τουλάχιστον 1 χαρακτήρα, ενώ μπορεί να περιέχει και κενά, αριθμούς και σύμβολα (εκτός των δεσμευμένων /, +, # και δεν μπορεί να ξεκινάει με $). Επίσης διαχωρίζει κεφαλαία με πεζά γράμματα (case sensitive).
Τα σύμβολα +, # είναι δεσμευμένα ως χαρακτήρες μπαλαντέρ επιπέδων. Συγκεκριμένα ο χαρακτήρας # χρησιμοποιείται για πολλαπλά επίπεδα και ο χαρακτήρας + χρησιμοποιείται για ένα επίπεδο. Για παράδειγμα ένας client που έχει κάνει subscribe στο apartment/+/temperature, θα λαμβάνει μηνύματα από τα topics apartment/kitchen/temperature, apartment/livingRoom/temperature, apartment1/bedroom/temperature κλπ. Ενώ ένας client που έχει κάνει subscribe στο apartment/# θα παίρνει μηνύματα από τα topics apartment/kitchen/temperature, apartment/livingRoom/TV, apartment/securityStatus κλπ καθώς και το apartment/. Μπορεί επίσης να γίνει και ο συνδυασμός δύο ή περισσοτέρων μπαλαντέρ από οποιοδήποτε είδος.
Το σύμβολο $ στην αρχή ενός topic υποδηλώνει ένα topic που είναι δεσμευμένο από τον broker για την κατάστασή του π.χ αριθμός συνδεδεμένων clients, χρόνος

λειτουργίας κα.

2.4.4 Ποιότητες Επικοινωνίας (QoS)

Το MQTT υποστηρίζει 3 διαφορετικές ποιότητες επικοινωνίας/υπηρεσίας. Αυτές διαφέρουν στο αν και με ποιο τρόπο ελέγχεται ότι ένα μήνυμα έφτασε στον παραλήπτη. Είναι πολύ σημαντική λειτουργία καθώς προσαρμόζει τη διαδικασία ανταλλαγής μηνυμάτων στις συνθήκες και τις ανάγκες μας ανάλογα με τον client. Έχουμε λοιπόν για την αποστολή μηνυμάτων Quality of Service (QoS) επιπέδου

- 0 (το πολύ μία φορά): Εδώ ο αποστολέας στέλνει (PUBLISH) το μήνυμα απλώς μια φορά χωρίς να περιμένει καμία επιβεβαίωση. Αυτή η ποιότητα ενδείκνυται για σταθερές συνδέσεις με καλή ποιότητα επικοινωνίας (πχ. Ενσύρματες) ή σε περιπτώσεις που το να χαθούν κάποια μηνύματα δεν αποτελεί σοβαρό πρόβλημα (πχ. Πυκνή χρονικά αποστολή δεδομένων για εξαγωγή μέσου όρου).
- 1 (τουλάχιστον μία φορά): Εδώ ο αποστολέας συνεχίζει να στέλνει το μήνυμα (PUBLISH) μέχρι να λάβει επιβεβαίωση από τον παραλήπτη με ένα πακέτο PUBACK πως το μήνυμα έφτασε (διπλή χειραψία, δηλαδή ανταλλαγή 2 μηνυμάτων για την ολοκλήρωση της επικοινωνίας). Τα πακέτα PUBLISH και PUBACK έχουν κοινό packetId ώστε να ξεχωρίζει η συγκριμένη “συνομιλία”. Στην περίπτωση QoS 1 μπορεί να παραδοθεί και να επεξεργαστεί και περισσότερες από μια φορές το μήνυμα (πχ. Αν ο παραλήπτης αργήσει να στείλει PUBACK). Συνεπώς αυτή η ποιότητα ενδείκνυται σε περιπτώσεις που μας ενδιαφέρει να μην χαθεί κάπως το μήνυμα αλλά δεν μας αφορά αν ο παραλήπτης θα το λάβει, και άρα θα το επεξεργαστεί, παραπάνω από μια φορές.
- 2 (ακριβώς μία φορά): Εδώ η επικοινωνία πραγματοποιείται με τετραπλή χειραψία (PUBLISH→PUBREC→PUBREL→PUBCOMP). Ο αποστολέας συνεχίζει όπως και στην προηγούμενη περίπτωση να στέλνει το αρχικό μήνυμα μέχρι να λάβει PUBREC και η επικοινωνία ολοκληρώνεται με άλλα δύο πακέτα, PUBREL και PUBCOMP. Η διαφορά με την περίπτωση QoS 1 είναι πως ο παραλήπτης αφού παραλάβει το πακέτο κρατάει το packetId και απορρίπτει πακέτα με ίδιο packetId. Έτσι, όσες φορές και να φτάσει το συγκεκριμένο πακέτο στον παραλήπτη (επειδή πχ. Άργησε να στείλει PUBREC), αυτός το επεξεργάζεται μόνο μία φορά.

QoS 1 και 2 χρησιμοποιούμε όταν θέλουμε να είμαστε σίγουροι πως ένα μήνυμα φτάνει στον παραλήπτη. Η διαφορά είναι πως, αν ο παραλήπτης δεν μπορεί να διαχειριστεί την περίπτωση να λάβει ένα πακέτο πολλές φορές και κάτι τέτοιο είναι προβληματικό (πχ. Θα βγάλει λάθος αποτελέσματα από την επεξεργασία του) χρησιμεύει το QoS 2. Αλλιώς με QoS 1 πετυχαίνουμε το ίδιο αποτέλεσμα πιο απλά.
Είναι σημαντικό να σημειωθεί ότι το QoS ορίζεται από τους clients, τόσο για κάθε μήνυμα που αποστέλλει ένας από αυτούς στον broker όσο και για κάθε subscription που κάνει. Αυτό έχει σαν αποτέλεσμα ένα μήνυμα να είναι δυνατόν να μεταδοθεί με κάποιο QoS από τον client αποστολέα στον broker και με διαφορετικό από τον broker στον client παραλήπτη. Έτσι αν θέλουμε να εξασφαλίσουμε ότι ο παραλήπτης θα λάβει το μήνυμα ακριβώς μια φορά θα πρέπει το subscription του να έχει QoS 2 και δεν παίζει ρόλο αν το QoS του αρχικού μηνύματος που έλαβε ο broker ήταν 1 ή 2.

2.4.5 Εμμενείς Συνεδρίες (Persistent Sessions)

Όταν ένας client αποσυνδεθεί από τον broker χάνονται οι πληροφορίες για τα topics στα οποία ήταν subscribed και εάν επανασυνδεθεί πρέπει η διαδικασία σύνδεσης και subscriptions να γίνει από την αρχή. Αυτό δεν είναι απαραίτητο, αν κατά την σύνδεση ο client έχει ορίσει την συνεδρία ως persistent.

Σε μία τέτοια συνεδρία αποθηκεύονται στην πλευρά του broker:

- Η ύπαρξη της συνεδρίας.
- Τα subscriptions.
- Όλα τα μηνύματα με QoS 1 ή 2, που δεν έχει επιβεβαιώσει ο client ότι έλαβε.
- Όλα τα νέα μηνύματα με QoS 1 ή 2 που έχει χάσει ο client όσο είναι offline.
- Όλα τα μηνύματα με QoS 2 που έχει λάβει ο broker, αλλά δεν έχει ολοκληρωθεί η τετραπλή χειραψία (δηλαδή ο broker δεν έχει λάβει το PUBREL).

Αντίστοιχα θα πρέπει και ο client να αποθηκεύει στην πλευρά του τα εξής:

- Όλα τα μηνύματα με QoS 1 ή 2, που δεν έχει επιβεβαιώσει ο broker.
- Όλα τα μηνύματα με QoS 2 που έχει λάβει ο client, αλλά δεν έχει ολοκληρωθεί η τετραπλή χειραψία (δηλαδή ο client δεν έχει λάβει το PUBREL).

Οι Persistent συνεδρίες είναι ιδιαίτερα χρήσιμες σε δίκτυα IoT και Fog. Η επανειλημμένη σύνδεση και δήλωση topic ενδιαφέροντος κάθε φορά που ένας client συνδέεται μπορεί να αποτελεί σημαντική επιβάρυνση για clients με περιορισμένους υπολογιστικούς πόρους αλλά και σε δίκτυα χαμηλής ποιότητας και ασταθείς συνδέσεις. Επίσης είναι αναμενόμενη συμπεριφορά για clients σε τέτοια δίκτυα να αποσυνδέονται και να επανασυνδέονται π.χ μετακινούμενοι clients που μπαινοβγαίνουν στην περιοχή που καλύπτει ο broker, χρήστες κινητών που ανοιγοκλείνουν τη σύνδεση κλπ.

Από τεχνικής πλευράς οι persistent συνεδρίες ταυτοποιούνται βάσει του clientid. Ο client ορίζει ότι επιθυμεί να εκκινήσει persistent συνεδρία θέτοντας το πεδίο cleanSession false. Αυτό ουσιαστικά ενημερώνει τον broker να μην ξεκινήσει καινούργια συνεδρία αλλά να συνεχίσει από την προηγούμενη συνεδρία, αν υπάρχει, αποστέλλοντας τα μηνύματα που έχουν αποθηκευτεί κλπ. Εάν το πεδίο cleanSession οριστεί true τότε διαγράφονται ότι στοιχεία έχουν αποθηκευτεί και ξεκινάει μια καινούργια, «καθαρή» συνεδρία η οποία όταν λήξει δεν θα κρατηθεί τίποτα.

2.4.6 Διατηρημένα Μυνήματα (Retained Messages)

Μια ακόμα δυνατότητα του MQTT είναι ο broker να αποθηκεύσει ένα μηνυμα ανά topic, το retained message. Ως retained message ορίζεται το τελευταίο μήνυμα που δημοσιεύεται σε ένα topic και έχει το πεδίο retain true. Όταν ένας client κάνει subscribe σε ένα τέτοιο topic, ο broker του στέλνει το τελευταίο retained message για το topic αυτό. Έτσι ο client παίρνει αμέσως μια ενημέρωση από το topic χωρίς να χρειάζεται να περιμένει να κάνει κάποιος publish εκεί. Το retained message είναι κάτι σαν την τελευταία γνωστή χρήσιμη κατάσταση για το topic, γιατί δεν είναι απαραίτητα η τελευταία ενημέρωση αλλά η τελευταία που ζητήθηκε να κρατηθεί. Παράδειγμα, σε ένα topic που αφορά την κατάσταση μιας συσκευής μπορεί το retained message να είναι offline ή online αντίστοιχα και όταν ένας client κάνει subscribe, να μαθαίνει αυτό αμέσως. Τα retained messages είναι ανεξάρτητα από τα clean ή persistent sessions, καθώς τα πρώτα αφορούν διαφορετικά topics ενώ τα

δεύτερα τις συνδέσεις των clients.

2.4.7 Τελευταία Επιθυμία και Διαθήκη (Last Will and Testament)

Όταν ένας client συνδέεται στον broker μπορεί να ορίσει το λεγόμενο Last Will and Testament. Αυτό περιγράφει ένα μήνυμα και τα χαρακτηριστικά του και αποτελείται από τα παρακάτω πεδία, που προαιρετικά ορίζονται στο πακέτο CONNECT:

- lastWillTopic, ορίζει το topic στο οποίο θα δημοσιευτεί το μήνυμα.
- lastWillQoS, ορίζει το QoS του μηνύματος.
- lastWillMessage, ορίζει το περιεχόμενο του μηνύματος.
- lastWillRetain, ορίζει αν το μήνυμα θα κρατηθεί ως retained.

Ο broker είναι υπεύθυνος για την αποθήκευση αυτού του μηνύματος και σε περίπτωση που ο client αποσυνδεθεί αναπάντεχα, αποστέλλει το συγκεκριμένο μήνυμα με βάση τις παραμέτρους που αναφέρθηκαν παραπάνω.
Ως αναπάντεχη αποσύνδεση θεωρούμε τις παρακάτω περιπτώσεις, χωρίς να περιοριζόμαστε μόνο σε αυτές:

- Ο διακομιστής εντοπίζει κάποιο σφάλμα Εισόδου/Εξόδου ή σφάλμα δικτύου
- Ο client αδυνατεί να απαντήσει στον προκαθορισμένο χρόνο.
- Ο client κλείνει την σύνδεση χωρίς να στείλει το προβλεπόμενο πακέτο DISCONNECT.
- Ο διακομιστής κλείνει την σύνδεση λόγω κάποιου λάθος του πρωτοκόλλου.

Η συνήθης χρησιμότητα του Last Will αφορά την κατάσταση ενός client, αν δηλαδή είναι συνδεδεμένος ή αποσυνδεδεμένος, συχνά ως retained μήνυμα.

2.4.8 Χρόνος Διατήρησης και Κατάληψη Σύνδεσης

Όπως αναφέρθηκε παραπάνω το MQTT επικοινωνεί πάνω από το πρωτόκολλο TCP. Παρότι το TCP σε κανονικές συνθήκες ενημερώνει όταν μια σύνδεση διακόπτεται, διάφορες συνηθισμένες υλοποιήσεις όπως στα κινητά τηλέφωνα για οικονομία παραλλάσσουν το τυπικό πρωτόκολλου TCP με αποτέλεσμα να παραμένουν συνδέσεις φαινομενικά ανοιχτές ενώ έχουν κλείσει από την μία πλευρά, όπως αναφέρει και ο δημιουργός του πρωτοκόλλου Andy Stanford-Clark [24]. Για αυτό τον λόγο το MQTT, καθώς αφορά βασικά τέτοιου είδους συνδέσεις, απαιτεί έναν ακόμα τρόπο ελέγχου. Αυτό επιτυγχάνεται με το Keep Alive (Χρόνος Διατήρησης). Πρόκειται για μια λειτουργικότητα μέσω της οποίας ελέγχεται ότι ο broker και ο client παραμένουν συνδεδεμένοι ο ένας με τον άλλον. Ο client κατά τη σύνδεσή του ορίζει το μέγιστο χρονικό διάστημα μέσα στο οποίο θα πρέπει να επιβεβαιωθεί ότι παραμένει συνδεδεμένος. Συγκεκριμένα στέλνει ένα μήνυμα PINGREQ στο οποίο ο broker πρέπει να απαντήσει με ένα μήνυμα PINGRESP. Σε περίπτωση που ο broker δεν λάβει μήνυμα PINGREQ μέσα στο ορισμένο χρονικό διάστημα, θεωρεί ότι ο client έχει αποσυνδεθεί, κλείνει την σύνδεση από την πλευρά του και στέλνει το last will, αν υπάρχει για τον συγκεκριμένο client. Αξίζει να σημειωθεί ότι ο client είναι υπεύθυνος να ορίσει το χρονικό διάστημα μέσα στο οποίο θα πρέπει να στείλει PINGREQ. Επίσης σε περίπτωση που ένας client ορίσει ως χρόνο keep alive 0, αυτό απενεργοποιεί τον συγκεκριμένο μηχανισμό και ο broker δεν περιμένει μηνύματα PINGREQ από τον συγκεκριμένο client.
Μπορεί ένας client να αποσυνδεθεί και να συνδεθεί ξανά εντός του χρονικού διαστήματος που ο broker περιμένει PINGREQ. Έτσι θα υπήρχαν στον broker 2

συνδέσεις για τον ίδιο client, κατάσταση που είναι μη επιθυμητή. Σε αυτή την περίπτωση ο broker θα κλείσει την προηγούμενη σύνδεση και θα συνεχίσει με την καινούργια. Αυτή η λογική εξασφαλίζει ότι η παλιά και πιθανά λανθασμένη σύνδεση δεν μπαίνει εμπόδιο στην καινούργια. Ταυτόχρονα βέβαια η παραπάνω λειτουργία ανοίγει θέματα security καθώς ένας client με το ίδιο όνομα μπορεί να διακόψει την σύνδεση κάποιου άλλου client. Το MQTT υποστηρίζει βασική διαδικασία ταυτοποίησης μέσω username/password για κάθε client, αν και δεν μεριμνά για την κρυπτογράφηση αυτών.

Τα παραπάνω αφορούν την έκδοση 3.1.1 η οποία αφορά και αυτή την διπλωματική. Στις αρχές του 2018 εκδόθηκε η έκδοση 5 που φέρνει μερικές αλλαγές και διευκολύνσεις και προσπαθεί να καλύψει τα όποια προβλήματα υπήρχαν.

Όπως βλέπουμε το MQTT μπορεί να διαχειριστεί πληθώρα διαφορετικών ειδών συσκευών. Μπορεί να διαχειριστεί σταθερές συσκευές που θεωρητικά θα παραμένουν συνεχώς συνδεδεμένες με τον broker, συσκευές που μπορεί να συνδέονται, να αποσυνδέονται και να επανασυνδέονται χωρίς να χάνουν πληροφορίες (μέσω persistent sessions) καθώς και συσκευές που λειτουργούν σαν παρατηρητές (π.χ. για στατιστικά) χωρίς να επιβαρύνουν ιδιαίτερα τον broker. Τέλος, με τα lastWill πεδία μπορεί να γίνεται καλύτερη διαχείριση και ενημέρωση των συσκευών για την κατάσταση του δικτύου παρά το space decoupling, ενώ και το keepAlive επιτρέπει προσαρμογή στις δυνατότητες του client.

2.4.9 Mosquitto

Στην παρούσα διπλωματική χρησιμοποιήσαμε ως MQTT broker τον Mosquitto. Πρόκειται για μια υλοποίηση ανοιχτού κώδικα (Eclipse Public License) και είναι αρκετά ελαφρύς και μπορεί να χρησιμοποιηθεί σε πληθώρα συσκευών από υπολογιστές single-board(π.χ. raspberry Pi) μέχρι κανονικούς διακομιστές. Βασίζεται στις εκδόσεις 3.1 και 3.1.1 του MQTT και υλοποιεί όλες τις παραπάνω λειτουργικότητες που περιγράφηκαν.

2.4.10 Paho

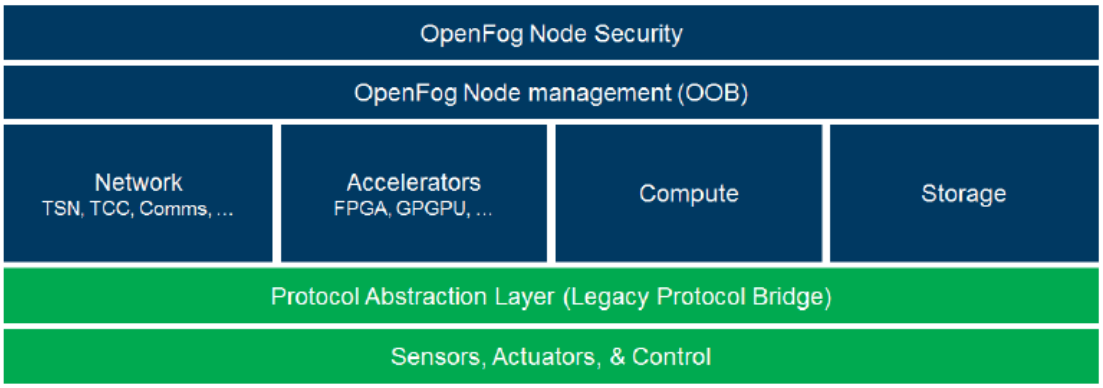
Για την υλοποίηση του MQTT στην εργασία μας χρησιμοποιήθηκε η βιβλιοθήκη Paho [25]. Η βιβλιοθήκη αυτή γεννήθηκε μέσα από ένα project του Eclipse Foundation που προσπαθεί να δημιουργήσει βιβλιοθήκες για διάφορες γλώσσες προγραμματισμού που θα υλοποιούν και θα επιτρέπουν την χρήση πρωτοκόλλων που αφορούν βασικά την επικοινωνία μεταξύ μηχανών (machine to machine communication-M2M). Πρόκειται για μια από τις πρώτες υλοποιήσεις ανοιχτού κώδικα του MQTT και συνεχίζει να αναπτύσσεται από την κοινότητα. Σήμερα η Paho έχει διαφορετικές βιβλιοθήκες για πολλές γλώσσες (C, C++, Python, Go, Java, κλπ). Συγκεκριμένα η βιβλιοθήκη της Java αποτελεί μια πλήρη υλοποίηση του πρωτοκόλλου, καθώς υποστηρίζει όλες τις δυνατότητες που προβλέπονται από αυτό. Θεωρείται ότι είναι αρκετά ευσταθής και έχει χρησιμοποιηθεί σε πολλές εφαρμογές που χρησιμοποιούν το MQTT για την επικοινωνία τους. Αποτελεί μια υλοποίηση με χειρισμό γεγονότων(event-driven) και στηρίζεται στην χρήση callbacks, όπου όταν ένα γεγονός συμβαίνει τότε τρέχει ο αντίστοιχος χειριστής(event-handler) στον client. Τα

γεγονότα που υποστηρίζονται είναι τα connectionLost (όταν χάνεται η σύνδεση με τον broker), deliveryComplete (όταν ολοκληρωθεί η διαδικασία αποστολής κάποιου μυνήματος) και messageArrived (όταν καταυθάνει ένα μύνημα στον client). Τέλος υποστηρίζει τόσο σύγχρονη όσο και ασύγχρονη επικοινωνία μεταξύ των clients.

Κεφάλαιο 3 : Σχεδίαση και Ανάλυση Αρχιτεκτονικής Νεφελώδους Υπολογισμού

3.1 Κόμβος Ομίχλης (Fog Node)

Η βασική μονάδα του Fog είναι ο κόμβος (Fog Node). Το Open Fog Consortium περιγράφει την εσωτερική αρχιτεκτονική ενός κόμβου όπως φαίνεται στο παρακάτω Διάγραμμα.

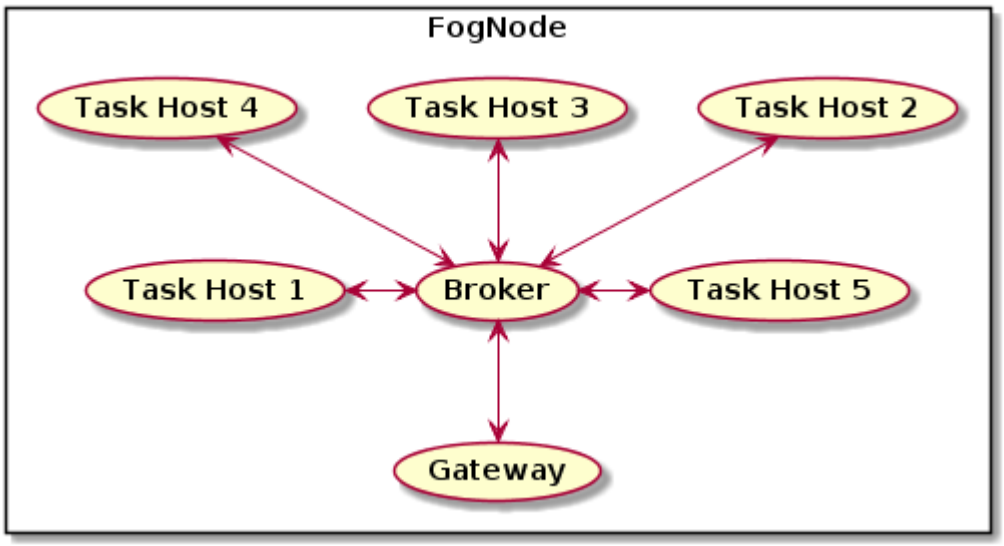


Διάγραμμα 3.1 [22] – Βασικές λειτουργικότητες κόμβου Ομιχλώδους Επεξεργασίας

Όπως φαίνεται στο κατώτερο επίπεδο (πιο κοντά στα άκρα του δικτύου) υπάρχει η σύνδεση με αισθητήρες, ενεργοποιητές και έλεγχος αυτών. Απαιτείται ένα ενδιάμεσο στρώμα προτυποποίησης και ελέγχου πρωτοκόλλων για να επιτυγχάνεται η διεπαφή των παραπάνω πραγμάτων (things) με τις συσκευές του κόμβου. Οι βασικές λειτουργείες που μπορεί να επιτελούνται σε έναν κόμβο είναι υπηρεσίες δικτύου, επεξεργασία, αποθήκευση και επιτάχυνση (π.χ. εισόδου/εξόδου). Αυτές οι λειτουργίες συντονίζονται κεντρικά σε επίπεδο κόμβου. Η δε διεπαφή του κόμβου με εξωτερικά στοιχεία στο δίκτυο προϋποθέτει εξασφαλισμένο επίπεδο ασφαλείας.

Η εσωτερική του διάρθρωση που υλοποιήθηκε στην παρούσα εργασία βασίζεται σε 3 βασικούς ρόλους-λειτουργικότητες που πρέπει να επιτελούνται.

1. Broker (Μεσάζων)
2. Gateway (Πύλη-Διεπαφή)
3. Task Host (Συσκευή Επεξεργασίας)



Διάγραμμα 3.2 – Διάρθρωση κόμβου Ομιχλώδους Επεξεργασίας
(Τα διαγράμματα έγιναν με χρήση του PlantUML)

Η παραπάνω διάκριση συμβαδίζει με την αρχιτεκτονική που προτείνει το Open Fog Consortium καθώς η Gateway επιτελεί τις δύο χαμηλότερες στο διάγραμμα λειτουργίες (πράσινο χρώμα), δηλαδή την συλλογή και προτυποποίηση των παραγόμενων δεδομένων. Η κεντρική διαχείριση και τα θέματα ασφαλείας επαφίονται στην λειτουργία του Broker και οι Task Hosts που συνδέονται επιτελούν τις 4 βασικές λειτουργίες. Στην παρούσα εργασία ασχολούμαστε βασικά με το ζήτημα της αποτελεσματικής ανάθεσης εργασιών στις συνδεδεμένες Task Hosts και την διαχείριση ροής των μηνυμάτων, χωρίς να αγγίζουμε ζητήματα όπως αυτό της ασφάλειας.

3.1.1 Συσκευή Επεξεργασίας (Task Host)

Ως Task Host εννοούμε κάθε συσκευή που συνδέεται στον κόμβο του Fog και δηλώνει πρόθεσή να συμμετέχει στον κόσμο Fog αναλαμβάνοντας να επεξεργαστεί δεδομένα και να περατώσει αιτήματα σύμφωνα με τις οδηγίες του Broker. Στην διάταξη του κόμβου Fog που συζητάμε κάθε συσκευή που δεν είναι Broker ή Gateway, επιτελεί ουσιαστικά τον ρόλο της Task Host.

Ανάλογα τον κόμβο Fog, Task Hosts μπορούν να είναι μια πληθώρα διαφορετικών και ετερόκλητων συσκευών. Για παράδειγμα σε ένα έξυπνο εργοστάσιο, όπου το Fog μπορεί να χρησιμοποιηθεί για την επί τόπου επεξεργασία και ρύθμιση της λειτουργίας του, οι Tasks Hosts θα είναι στην πλειονότητά τους συγκεκριμένες συσκευές που εξυπηρετούν βασικά αυτή την λειτουργία. Συμπληρωματικό ρόλο μπορεί να έχουν για παράδειγμα τα κινητά των εργαζομένων, αλλά δεν είναι κατ' ανάγκην αυτά οι βασικοί φορείς της επεξεργασίας. Από την άλλη σε ένα αυτοοδηγούμενο αυτοκίνητο οι Task Hosts θα είναι συσκευές των επιβατών, άλλες διαθέσιμες συσκευές καθώς το αυτοκίνητο κινείται κλπ.

Ενώ στην περίπτωση σταθερών συσκευών είναι πιο εύκολο να διακρίνουμε ποια συσκευή είναι κατάλληλη για κάποια εργασία, αυτή η επιλογή γίνεται πιο πολύπλοκη για φορητές συσκευές. Στην δεύτερη περίπτωση έχουμε να λάβουμε υπόψη μας και παραμέτρους όπως τη μπαταρία, την ποιότητα της σύνδεση, η ώρα που θα μείνει εντός εμβέλειας του κόμβου κλπ εκτός των κλασσικών όπως η υπολογιστική ισχύς, η μνήμη κλπ. Είναι δε αυτές οι συσκευές που στην σημερινή κατάσταση παραμένουν αναξιοποίητες και παράλληλα έχουν τις απαραίτητες υπολογιστικές δυνατότητες ώστε να αναλάβουν ένα κομμάτι της επεξεργασίας που ήδη επιβαρύνει και θα συνεχίσει πολλαπλάσια το υπολογιστικό νέφος.

3.1.2 Μεσάζων (Broker)

Ο Broker είναι το κέντρο ενός κόμβου Fog. Είναι υπεύθυνος για τον συντονισμό της ροής των μηνυμάτων μέσα στον κόμβο. Στον Broker έρχονται τα διάφορα αιτήματα για επεξεργασία, αποθήκευση κτλ από χαμηλότερα επίπεδα του δικτύου (πιο κοντά στα άκρα) τα οποία καλείται να αναθέσει κατάλληλα στις Task Hosts που συμμετέχουν στον κόμβο ή να τα προωθήσει σε ανώτερο επίπεδο (προς το υπολογιστικό νέφος) καθώς και να επιστρέψει τα αποτελέσματα των επεξεργασιών προς τα χαμηλότερα επίπεδα. Για να μπορέσει να επιτευχθεί κάτι τέτοιο χρειάζονται ορισμένα χαρακτηριστικά των μηνυμάτων όσον αφορά την ευαισθησία της επεξεργασίας προς τον χρόνο, τις απαιτούμενες υπολογιστικές δυνατότητες κτλ.

Βασική λειτουργικότητα που πρέπει ο Broker να έχει για να μπορεί να επιτελεί τον ρόλο του ως συντονιστής είναι η εποπτεία των Task Hosts που είναι διαθέσιμες. Αυτό σημαίνει ο Broker να γνωρίζει κατά πόσο μια συσκευή είναι ενεργή, συνδεδεμένη και έτοιμη να υποδεχθεί ένα αίτημα για επεξεργασία αλλά και να επιλέγει σε ποιες συσκευές ανατίθενται ποιες επεξεργασίες. Σε περιβάλλον Fog οι Task Hosts είναι εν γένει συσκευές των άκρων (edge devices), άρα αρκετά ετερογενείς ως προς τα χαρακτηριστικά και τις δυνατότητές τους. Επίσης συχνά είναι φορητές και άρα το κριτήριο επιλογής εκτός από την επεξεργαστική ισχύ, την μνήμη κτλ, θα πρέπει να αφορά και τα χαρακτηριστικά της σύνδεσής τους, την θέση τους, την μπαταρία τους κλπ. Ταυτόχρονα ο Broker είναι υπεύθυνος για την διαχείριση θεμάτων ασφαλείας. Ως ενδιάμεσος αναμέσα στο υπολογιστικό νέφος και τις συσκευές ο Broker είναι επιφορτισμένος με έναν πρώτο έλεγχο. Ειδικά σε περιβάλλον Fog, όπου οι Task Hosts που συνδέονται μπορεί να είναι άγνωστες και πιθανά κακόβουλες, χρειάζεται ένας μηχανισμός διάκρισης αυτών και απόρριψης των επικίνδυνων.

Για να γίνει πιο κατανοητό ας θεωρήσουμε έναν κόμβο Fog σε μία έξυπνη πόλη. Στον κόμβο αυτόν θα γίνεται κομμάτι επεξεργασίας που παράγεται για παράδειγμα από τους αισθητήρες που καταγράφουν την κυκλοφορία αυτοκινήτων και άλλων οχημάτων και βάση αυτού προσαρμόζουν διαδρομές μέσων μαζικής μεταφοράς, φανάρια κτλ. Σε έναν τέτοιο κόμβο οι Task Hosts μπορούν να αποτελούν τα κινητά των περαστικών, τα τάμπλετς, έξυπνα οχήματα κλπ. Σημασία σε ένα τέτοιο σενάριο αποκτά και η φυσική θέση του Broker ώστε να βρίσκεται κατάλληλα τοποθετημένος σε σχέση με την εμβέλεια των Task Hosts. Για παράδειγμα καλές θέσεις για την τοποθέτηση του Broker μπορεί να είναι μια κεντρική πλατεία, μια στάση λεωφορείου

κλπ. Οι Task Hosts σε αυτή την περίπτωση δεν είναι εξασφαλισμένο ότι παραμένουν συνδεδεμένοι στον Broker καθώς κινούνται ή μπορούν να αποσυνδέονται από το δίκτυο (παράδειγμα, κινούμενο αυτοκίνητο, κινητό τηλέφωνο που απενεργοποιεί την σύνδεση στο δίκτυο ή κλείνει από μπαταρία κλπ). Ο Broker πρέπει να είναι σε θέση να σταθμίσει αυτά με τον καλύτερο δυνατό τρόπο για να επιλέξει τις κατάλληλες Task Hosts. Ένα γρήγορα κινούμενο αυτοκίνητο δεν ενδείκνυται για μια χρονοβόρα επεξεργασία, αφού είναι πολύ πιθανό να βρεθεί εκτός εμβέλειας πρώτου αυτή ολοκληρωθεί. Μια Task Host με χαμηλό επίπεδο μπαταρίας δεν πρέπει να προτιμάτε για την ανάθεση μιας επεξεργασίας. Σε μια Task Host με ασταθή πρόσβαση στο δίκτυο ή χαμηλές επεξεργαστικές δυνατότητες δεν θα πρέπει να ανατίθενται επεξεργασίες που απαιτούν μικρό χρόνο απόκρισης. Για Task Hosts που έχουν εντοπιστεί να μην απαντούν εγκαίρως σε πολλά αιτήματα θα πρέπει να λαμβάνεται υπόψη το γεγονός πριν τους ανατεθεί κάποια επεξεργασία κλπ.

Για να μπορεί να επιτελέσει ο Broker τις παραπάνω λειτουργείες έχει κάποιες ελάχιστες απαιτήσεις τόσο όσον αφορά τις υπολογιστικές του δυνατότητες όσο και την αυτονομία του ενεργειακά και την σταθερότητα του. Ενδείκνυται λοιπόν να είναι ένα σταθερό άκρο με συγκεκριμένη υπολογιστική ισχύ ανάλογα με τον συγκεκριμένο κόμβο Fog.

1Πρέπει να σημειώσουμε εδώ ότι ο Broker (Fog Broker) στον οποίο αναφερόμαστε δεν είναι ο ίδιος με τον Broker που αναφέρθηκε στο κεφάλαιο 2.4.2, ως MQTT Broker. Ο πρώτος (Fog Broker) είναι υπεύθυνος για την διαχείριση του κόμβου Fog και την κατανομή των αιτημάτων στις συσκεύες επεξεργασίας. Ο δεύτερος (MQTT Broker) είναι υπεύθυνος για την ορθή υλοποίηση του πρωτοκόλου MQTT και την επικοινωνία των συμμετεχόντων σε ένα δίκτυο που λειτουργεί με MQTT. Όπως θα δείξουμε και στο κεφάλαιο 5 η σχετική θέση των δύο αυτών οντοτήτων δεν παίζει ιδιαίτερο ρόλο.

3.1.3 Πύλη-Διεπαφή (Gateway)

Η Gateway αποτελεί την διεπαφή του κόμβου Fog με το χαμηλότερο επίπεδο του δικτύου, δηλαδή με έναν επίπεδο πιο κοντά στα άκρα. Ο ρόλος της Gateway στον κόμβο Fog είναι η συλλογή των παραγόμενων δεδομένων, η προτυποίηση και πακετάρισμά τους σε κατάλληλα μηνύματα και η προώθηση αυτών στο δίκτυο (μέσω του Broker) για επεξεργασία ώστε να επιτυγχάνονται οι επιθυμητές λειτουργίες και επεξεργασίες. Επίσης η Gateway αναλαμβάνει να ενημερώσει συσκευές IoT και ενεργοποιητές και γενικά να προωθήσει τα αποτελέσματα προς το χαμηλότερο επίπεδο του δικτύου.

Σε έναν κόμβο Fog ακριβώς πάνω από το άκρο ο ρόλος της Gateway αφορά την διεπαφή με συσκευές περιορισμένων υπολογιστικών και δικτυακών δυνατοτήτων, όπως αισθητήρες IoT, ενεργοποιητές κλπ. Τέτοιες συσκευές μπορεί να αδυνατούν να εκτελέσουν ακόμα και στοιχειώδη προεπεξεργασία, όπως για παράδειγμα συσσωμάτωση πακέτων (packet aggregation), ή να έχουν σημαντικούς περιορισμούς στην δικτυακή τους λειτουργία, όπως για παράδειγμα δυσκολία υποστήριξης ορισμένων πρωτοκόλλων. Τέλος η απουσία ενιαίου δεσμευτικού προτύπου για τον τρόπο με τον οποίο τέτοιες συσκευές κωδικοποιούν και προωθούν τα δεδομένα τους

δημιουργεί ανομοιογένειες στο σετ δεδομένων το οποίο θέλουμε να επεξεργαστούμε. Όπως για παράδειγμα διαφορετικές κωδικοποιήσεις, πακετάρισμα μηνυμάτων με διαφορετικού είδους κεφαλίδες κλπ. Σε αυτή την περίπτωση η Gateway θα πρέπει σε πρώτη φάση να συλλέξει τα διάφορα ετερόκλητα δεδομένα, να τα ομογενοποιήσει κατάλληλα και να τα πακετάρει σε μηνύματα σύμφωνα με τα πρωτόκολλά του κόμβου Fog. Επιτελεί δηλαδή όλη την αναγκαία προετοιμασία για να προωθηθούν και να επεξεργαστούν κατάλληλα τα δεδομένα στον κόμβο περιγράφοντας την επιθυμητή επεξεργασία για αυτά, ενσωματώνοντας τα δεδομένα και παρέχοντας όποια αναγκαία μετα-δεδομένα (απαιτούμενος χρόνος απόκρισης, επεξεργαστική ισχύς που χρειάζεται κλπ) [7]. Αφού ολοκληρωθεί η επεξεργασία η Gateway αναλαμβάνει να προωθήσει με κατάλληλο τρόπο τα αποτελέσματα στο χαμηλότερο επίπεδο, ενημερώνοντας ενεργοποιητές και προσαρμόζοντας γενικά την λειτουργία των συσκευών IoT. Για να μπορεί να γίνει αυτό προϋποτίθεται η Gateway να γνωρίζει τι αιτήματα για επεξεργασία έχει πραγματοποιήσει και να μπορεί να ταιριάζει τα αποτελέσματα τις επεξεργασίας στα αιτήματα αυτά προβαίνοντας στις κατάλληλες ενέργειες. Για τον λόγο αυτό χρειάζεται να ονοματίζει με κατάλληλο τρόπο τα αρχικά αιτήματα για να μπορεί να ταυτοποιήσει τις απαντήσεις.

Σε μία ευρύτερη αρχιτεκτονική δικτύου που αποτελείται από περισσότερα επίπεδα Fog (N-tier Fog) η λειτουργία της Gateway απαιτείται ακόμα, αν και οι συγκεκριμένες απαιτήσεις και προεπεξεργασία που αυτή κάνει αλλάζουν αντίστοιχα με το επίπεδο (λόγω των δυνατοτήτων των συσκευών που συμμετέχουν, της ήδη υπάρχουσας προεπεξεργασίας σε χαμηλότερα επίπεδα κλπ). Τον ρόλο της Gateway για ένα επίπεδο παίζει εδώ ο Broker του προηγούμενου επιπέδου. Αν και η προτυποποίηση πιθανότατα δεν χρειάζεται στα ενδιάμεσα επίπεδα, ακόμα απαιτείται ο έλεγχος στην ροή της πληροφορίας (ποιος ζήτησε τι, πόσο σύντομα το χρειάζεται κλπ) και τον ρόλο αυτόν πραγματώνουν οι ενδιάμεσοι Gateway.

Όπως φαίνεται και στο διάγραμμα 3.2, ο Broker και η Gateway είναι δύο κόμβοι του δικτύου που συνδέονται μεταξύ τους με μία ακμή. Και οι δύο αυτές οντότητες επιτελούν λειτουργίες συντονισμού και αναγκαίας υποδομής για να σχηματιστεί ένας κόμβος Fog. Επιλέγουμε να τους προσεγγίσουμε σας δύο ξεχωριστούς κόμβους διότι οι ρόλοι τους έχουν ποιοτικές διαφορές. Ο Broker αναλαμβάνει τον συντονισμό της κίνησης των διαφόρων μηνυμάτων από και προς τις συνδεδεμένες Task Hosts και την ανάθεση εργασιών σε αυτές ανάλογα με τα χαρακτηριστικά τους. Αυτός ο ρόλος αν και απαιτεί κάποιες προδιαγραφές από την συσκευή που παίζει τον ρόλο του Broker, εφόσον αυτές πληρούνται είναι σχετικά ευέλικτος στο ποια συσκευή τελικά τον επιτελεί. Η Gateway από την άλλη αναλαμβάνει την διεπαφή με ένα πλήθος ετερόκλητων συσκευών των άκρων και την προεπεξεργασία των δεδομένων που αυτές παράγουν ώστε να παριστάνονται με έναν ομοιογενή τρόπο (κοινή κωδικοποίηση, δομή μηνύματος κλπ) και άρα να είναι σαφής ο τρόπος που απαιτείται να επεξεργαστούν. Τόσο η επικοινωνία όσο και η προεπεξεργασία των δεδομένων διαφέρει ανά περίπτωση και είναι συγκεκριμένη για κάθε μία μεμονωμένη συσκευή IoT. Για παράδειγμα το ποια πρότυπα και πρωτόκολλα υποστηρίζει κάποιος συγκεκριμένος αισθητήρας καθώς και η υπολογιστικές του

δυνατότητες καθορίζει την μορφή που θα έχουν τα δεδομένα που φτάνουν στην Gateway. Άρα η συμμόρφωση αυτών με μία γενικά αποδεκτή μορφή απαιτεί συγκεκριμένες για την περίπτωση ενέργειες (μετατροπή από συγκεκριμένη σε συγκεκριμένη κωδικοποίηση, κατάλληλο παραγέμισμα κλπ). Αντίθετα μια συσκευή που πληροί τις προδιαγραφές σε επίπεδο υλικού μπορεί με την εγκατάσταση ελάχιστου λογισμικού να παίξει τον ρόλο του Broker. Για παράδειγμα ένα κινητό τηλέφωνο κατεβάζει και εγκαθιστά μια εφαρμογή για Fog και μπορεί να παίξει τον ρόλο του Broker για χρήστες που έχουν εγκατεστημένη την ίδια εφαρμογή. Αντίθετα το ίδιο κινητό δεν είναι απαραίτητο ότι έχει τις δυνατότητες διασύνδεσης ή μπορεί να τρέξει κατάλληλο λογισμικό για να επικοινωνήσει και να μορφοποίηση κατάλληλα τα δεδομένα που παράγει ένας αισθητήρας IoT. Για ενδιάμεσα επίπεδα Fog η επικοινωνία με τα κατώτερα και ανώτερα επίπεδα γίνεται μέσω των Brokers καθώς τα μηνύματα που ανταλλάσσονται είναι ομογενή. Τέλος σε έναν κόμβο Fog θα μπορούσαν να συμμετέχουν παραπάνω από μια Gateways. Για παράδειγμα σε ένα νοσοκομείο μπορεί να έχουμε μία Gateway ανά δωμάτιο ή ανά όροφο για λόγους ασφαλείας, ταξινόμησης κλπ, οι οποίες όμως όλες συμμετέχουν στο κοινό Fog του νοσοκομείου. Αντίθετα σε έναν κόμβο Fog μπορεί να υπάρχει μόνο ένας Broker καθώς αυτός αναλαμβάνει τον συντονισμό αυτού.

3.2 Συνάρτηση Βαθμολόγησης

3.2.1 Ανάλυση της συνάρτησης βαθμολόγησης

Όπως αναφέραμε παραπάνω σημαντική αρμοδιότητα του Broker είναι η αποδοτική κατανομή των διαφόρων αιτημάτων σε κατάλληλες Task Hosts. Αυτή η λειτουργικότητα επιτυγχάνεται με την χρήση μιας μεθόδου αξιολόγησης των χαρακτηριστικών και της κατάστασης των Task Hosts. Στην αρχιτεκτονική που αναφερόμαστε [7], η προτεινόμενη μέθοδος αξιολόγησης είναι η χρήση μιας scoring function (συνάρτηση βαθμολόγησης). Αυτή αναλαμβάνει να αποδώσει μια βαθμολογία σε κάθε συνδεδεμένη Task Host, λαμβάνοντας υπόψη της κάποια χαρακτηριστικά. Όταν ο Broker λαμβάνει κάποιο αίτημα για επεξεργασία, το αναθέτει στην Task Host με την υψηλότερη βαθμολογία εκείνη την στιγμή.

Πρέπει να αναφέρουμε ότι κριτήριο για την αποτελεσματικότητα της μεθόδου βαθμολόγησης δεν είναι μόνο η ακρίβεια και η επιλογή των καταλληλότερων χαρακτηριστικών. Ρόλο παίζουν επίσης οι απαιτήσεις σε χρόνο και υπολογιστικούς πόρους καθώς και να μην μπαίνει εμπόδιο στην εύρυθμη λειτουργία του Broker. Για παράδειγμα κατά το δυνατόν η βαθμολόγηση να μην καθυστερεί την ανάθεση ενός αιτήματος σε μια Task Host.

Στην αρχική μορφή της Scoring Function τα χαρακτηριστικά που λαμβάνει υπόψιν της είναι:

- Αξιοποίηση πόρων (Utilization)
- Καθυστέρηση δικτύου (Latency)
- Στάθμη μπαταρίας (Battery)

Βασιζόμενοι στην προαναφερθείσα εργασία, επιλέξαμε να παραλλάξουμε την scoring function ως εξής. Η Αξιοποίηση Πόρων, παρότι περιγράφει ένα χρήσιμο, αν μετρηθεί σωστά μέγεθος, στην πράξη είναι δύσκολο να ληφθεί απευθείας από την κατάσταση λειτουργίας μιας Task Host. Το μεγάλο εύρος διακύμανσης στον φόρτο επεξεργασίας ενός πυρήνα της CPU σε πολύ μικρό διάστημα καθιστά την λήψη αυτού του μεγέθους σαν δείκτη αξιοποίησης πόρων αναποτελεσματική. Καθώς μια στιγμιαία μέτρηση μπορεί να είναι τρομέρα αναντίστοιχη με την πραγματική κατάσταση αξιοποίησης των πόρων του συστήματος.

Έτσι επιλέγουμε να προσεγγίσουμε την λογική της αξιοποίησης των διαθέσιμων πόρων μιας Task Host μέσω της διαθέσιμης επεξεργαστικής ισχύος, της διαθέσιμης μνήμης και του αριθμού των εργασιών που της έχουν ήδη ανατεθεί από το σύστημα Fog. Συγκεκριμένα αν η βαθμολογία που αποδίδεται σε μια Task Host είναι ανεξάρτητη του αριθμού των εργασιών που έχει αναλάβει, μπορεί εύκολα να καταλήξουμε σε περιπτώσεις που η διάταξη των συσκευών παραμένει σταθερή. Έτσι όλα τα εισερχόμενα αιτήματα θα ανατίθεται στην Task Host που βρίσκεται πρώτη, αφήνοντας αναξιοποίητες τις υπόλοιπες και πιθανά υπερφορτώντας την. Τα χαρακτηριστικά της Στάθμης Μπαταρίας και της Καθυστέρησης αποτελούν όντως χρήσιμους δείκτες για την Task Host.

Τέλος θεωρήσαμε χρήσιμη την επέκταση των χαρακτηριστικών που λαμβάνει υπόψιν της η Scoring Function, ώστε να καθίσταται πιο αποτελεσματική στην περίπτωση δικτύων IoT, προσθέτοντας τα χαρακτηριστικά Θέση και Βαθμός εμπιστοσύνης. Η επιλογή αυτών των χαρακτηριστικών είναι κατά την γνώμη μας κομβική σε περιπτώσεις κόμβων Fog που συμμετέχουν φορητές Task Hosts και αφορά ζητήματα που θίγει η βιβλιογραφία πάνω στο Fog.

Σε περιπτώσεις όπως η τοποθέτηση Fog Broker σε δημόσιο χώρο σημασία, για την αξιολόγηση και βαθμολόγηση των Task Hosts, μπορεί να έχουν χαρακτηριστικά όπως:

- Επεξεργαστική ισχύς. Ένας απλός τρόπος για να μετρηθεί η συγκεκριμένη ποσότητα είναι ο αριθμός των πυρήνων του επεξεργαστή.
- Διαθέσιμη μνήμη.
- Καθυστέρηση Δικτύου (Latency). Όπως έχουμε πει, παίζει σημαντικό ρόλο σε αιτήματα όπου ο χρόνος απόκρισης παίζει χρειάζεται να είναι μικρός.
- Στάθμη Μπαταρίας. Ειδικά για περιπτώσεις συσκευών όπως κινητά τηλέφωνα καλό είναι να αποφεύγονται συσκευές με χαμηλό επίπεδο μπαταρίας. Όχι μόνο σε ακραίες περιπτώσεις, όπου κινδυνεύουν να απενεργοποιηθούν προτού τελειώσει η επεξεργασία, αλλά και για λόγους ευχρηστίας και άνετης αλληλεπίδρασης με τους χρήστες. Για παράδειγμα σε περιπτώσεις όπως κινητά τηλέφωνα ή ταμπλετ, κριτήριο για την αποτελεσματικότητα της λειτουργίας του Fog είναι να μην δημιουργεί πρόβλημα εξαντλώντας την μπαταρία των χρηστών.
- Θέση στο πεδίο εμβέλειας του Broker. Σε περιπτώσεις κινούμενων συσκευών παίζει ρόλο για το πόση ώρα μια Task Host θα βρίσκεται εντός εμβέλειας του

Broker. Για τον λόγο αυτό χρειάζεται μια εκτίμηση σε σχέση με την θέση και την ταχύτητα της συσκευής, ώστε να αποφεύγονται Task Hosts που πρόκειται πολύ σύντομα να βρεθούν εκτός εμβέλειας και άρα να μην καταφέρουν να επιστρέψουν αποτέλεσμα. Ταυτόχρονα το ζήτημα της χωροταξικής τοποθέτησης του ίδιου του Broker αποτελεί ένα ζήτημα που απασχολεί ερευνητές στο πεδίο του Fog [26] [27]

- Αριθμός Εργασιών. Κριτήριο αποδοτικής κατανομής είναι να μην φορτώνεται υπερβολικά μια μόνο Task Host βάσει των υπολοίπων χαρακτηριστικών της και να υπάρχει σχετική εναλλαγή στο ποιος αναλαμβάνει να διαχειριστεί ένα αίτημα.
- Βαθμός Εμπιστοσύνης. Σε περιπτώσεις όπου μια Task Host έχει υψηλό ποσοστό αποτυχίας στην ολοκλήρωση των επεξεργασιών που της ανατίθενται ενδείκνυται πιθανά να αποφεύγεται να ανατεθεί στην συσκευή αυτή περαιτέρω επεξεργασίες στο μέλλον. Αυτό μπορεί να αφορά Task Hosts με ασταθή σύνδεση, κακόβουλους χρήστες που συμμετέχουν στο Fog κλπ. Το χαρακτηριστικό αυτό μπορεί να αποθηκεύεται σε κάποια βάση δεδομένων στην οποία έχουν πρόσβαση όλοι οι Brokers και να ενημερώνεται το Fog συνολικά και όχι μόνο ο κόμβος στον οποίο συνδέεται κάποιος κακόβουλος χρήστης. Σύμφωνα και με την τεχνική έκθεση του Open Fog Consortium [22] βασικός πυλώνας σχεδίασης ενός συστήματος Fog είναι αυτός της ασφάλειας, ο οποίος περιλαμβάνει και συστήματα αξιολόγησης του βαθμού εμπιστοσύνης υλικού και λογισμικού που συμμετέχει στους κόμβους του Fog.

Στα πλαίσια της παρούσας εργασίας, στην εφαρμογή που αναπτύξαμε, η βαθμολογία μίας Task Host καθορίζεται από τα παραπάνω 7 χαρακτηριστικά.

Όσον αφορά συγκεκριμένα την συνάρτηση βαθμολόγησης αξίζει να υπογραμμιστεί ότι η επιλογή κατάλληλων χρονισμών είναι καθοριστικής σημασίας για την αποδοτική λειτουργία του κόμβου Fog. Συγκεκριμένα στην διαδικασία της βαθμολόγησης η συλλογή των χαρακτηριστικών των συμμετεχόντων Task Hosts μέσω αποστολής μηνυμάτων είναι ο καθοριστικός παράγοντας επιβάρυνσης που επιφέρει η διαδικασία βαθμολόγησης. Ως εκ τούτου, χαρακτηριστικά που παραμένουν αμετάβλητα, καθώς και αυτά που δεν αλλάζουν σημαντικά σε διάστημα δευτερολέπτων, αποτελούν πολύ καλές επιλογές καθώς οι τιμές τους παραμένουν έγκυρες για σχετικά αργή περιοδική ανανέωση. Τέτοια χαρακτηριστικά είναι στην περίπτωσή μας αυτά του αριθμού των πυρήνων, της διαθέσιμης μνήμης, η στάθμη της μπαταρίας και η θέση στο πεδίο εμβέλειας του Broker. Επίσης χαρακτηριστικά που μπορεί ο Broker να υπολογίζει χωρίς την αποστολή μηνυμάτων ενημέρωσης από τις Task Host, αποτελούν καλές περιπτώσεις. Τέτοια είναι στην περίπτωσή μας ο αριθμός εργασιών και ο βαθμός εμπιστοσύνης. Τέλος χαρακτηριστικά που η εγκυρότητα της τιμής τους προϋποθέτει πολύ συχνή ανανέωσή της δεν ενδείκνυται. Αυτός είναι ένας βασικός λόγος που απορρίψαμε την αξιοποίηση πόρων (utilization) ως χαρακτηριστικό βαθμολόγησης. Η καθυστέρηση του δικτύου παρότι απαιτεί σχετικά συχνότερη ενημέρωση από την πρώτη κατηγορία για να είναι έγκυρη, έχει μια σχετική σταθερότητα και άρα πιο αργές περιοδικές μετρήσεις δίνουν σχετικά

αξιόπιστα αποτελέσματα, ενώ πρόκειται για εξαιρετικά σημαντικό δείκτη για να απορριφθεί. Σημασία έχει ακόμα η διαδικασία βαθμολόγησης να έχει σχετική ανεξαρτησία από την διαδικασία συλλογής των χαρακτηριστικών, καθώς η δεύτερη είναι σημαντικά πιο χρονοβόρα.

Επειδή σε έναν κόμβο Fog, σαν αυτούς που προαναφέραμε, η βαθμολόγηση μπορεί να είναι αναγκαίο να γίνεται αρκετά συχνά χρειάζεται μια αρκετά απλή συνάρτηση σκοραρίσματος που να μην επιβαρύνει τον Broker. Θα πρέπει να λάβουμε υπόψη μας ότι ο Broker δεν είναι αναγκαστικά ένα πολύ ισχυρό μηχάνημα, ούτε απαραίτητα η αποκλειστική του λειτουργία αυτή του Broker. Ειδικές παράμετροι και προδιαγραφές γύρω από την βαθμολόγηση όπως κάθε πότε γίνεται η βαθμολόγηση, πόσο ακριβής πρέπει να είναι και η σχετική σημασία των παραπάνω χαρακτηριστικών στην βαθμολογία ή η προσθήκη επιπλέον χαρακτηριστικών στην συνάρτηση βαθμολόγησης δεν μπορούν να συγκεκριμενοποιηθούν στο πλαίσιο αυτής της εργασίας. Αυτό διότι χρειάζεται πειραματισμός και μετρήσεις που να αφορούν συγκεκριμένες συνθήκες. Για παράδειγμα τα σχετικά βάρη των διαφόρων χαρακτηριστικών θα μπορούσαν να προσδιοριστούν επαρκώς μέσω εφαρμογής αλγορίθμων υπολογιστικής μάθησης (machine learning) και αξιοποιώντας την εφαρμογή της παρούσας εργασίας σαν υπόβαθρο για τις δοκιμές. Τα συγκεκριμένα σχετικά βάρη που θα έχουν τα χαρακτηριστικά που απαρτίζουν την βαθμολογία μπορεί να διαφέρουν από εφαρμογή σε εφαρμογή και από περίπτωση σε περίπτωση και θα απαιτούν ειδικό σετ δεδομένων για να προσδιοριστούν πλήρως.

3.2.2 Μελέτη αναδρομικού υπολογισμού βαθμολογίας

Στην περίπτωση της εφαρμογής που αναπτύξαμε η τιμή της βαθμολΚεφάλαιο 3 : Σχεδίαση και Ανάλυση Αρχιτεκτονικής Νεφελώδους Υπολογισμούογίας προκύπτει από την συνάρτηση:

$$Score = \sum_{n=1}^{7} \frac{(x_n - \bar{x}_n)}{s_{\bar{x}_n}^2} \quad (1)$$

Η συνάρτηση αυτή προσθέτει τις κανονικοποιημένες τιμές των 7 χαρακτηριστικών, όπου x_n

Για τον υπολογισμό της κανονικοποιημένης τιμής απαιτείται για κάθε ένα από τα 7 χαρακτηριστικά η μέση τιμή $\bar{x}$ (average) και η τυπική απόκλιση s (standard deviation) για το σύνολο των Task Hosts. Τα δύο αυτά μεγέθη ορίζονται ως εξης:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad (2)$$

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \quad (3)$$

Σε περίπτωση που συνδεθεί ή αποσυνδεθεί μία Task Host τα δύο αυτά μεγέθη μεταβάλλονται και πρέπει να υπολογιστεί η νέα τιμή τους. Κάτι τέτοιο είναι ιδιαίτερα

χρονοβόρο καθώς θα πρέπει να προσπελαστούν όλες οι τιμές και να γίνουν οι αντίστοιχες πράξεις. Αντ' αυτού υλοποιήσαμε τον υπολογισμό της μέσης τιμής και της τυπικής απόκλισης για σύνδεση και αποσύνδεση μίας Task Host αναδρομικά και άρα αποδοτικότερα. Αυτό είναι δυνατό αφού κάθε φορά στο δείγμα απλά προστίθεται ή αφαιρείται ένα καινούργιο στοιχείο, ενώ όλα τα άλλα παραμένουν τα ίδια.

Συγκεκριμένα σε περίπτωση εισόδου νέας Task Host η μέση τιμή και η τυπική απόκλιση σχετίζεται με την προηγούμενη τιμή της, όπως αποδεικνύεται παρακάτω

Για την μέση τιμή έχουμε :

$$\bar{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4}$$

$$\bar{x}_n = \frac{n-1}{n}\left(\sum_{i=1}^{n-1} \frac{x_i}{n-1}\right) + \frac{x_n}{n} \tag{5}$$

Και τελικά έχουμε τον αναδρομικό τύπο :

$$\bar{x}_n = \frac{[\bar{x}_{n-1}(n-1)] + x_n}{n} \tag{6}$$

Για την τυπική απόκλιση, η εύρεση αναδρομικής σχέσης είναι πιο μακροσκελής διαδικασία:

$$s_n^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1} = \frac{Sum_n}{n-1} \tag{7}$$

Ξεκινάμε από το τετράγωνο της τυπικής απόκλισης και συγκεκριμένα τον αριθμητή του, έστω Sum. Είναι :

$$Sum_n = \sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{8}$$

$$Sum_n = \sum_{i=1}^{n}(x_i^2 + \bar{x}^2 - 2\bar{x}x_i) \tag{9}$$

$$Sum_n = \sum_{i=1}^{n}(x_i^2) + n\bar{x}^2 - 2\bar{x}\sum_{i=1}^{n} x_i \tag{10}$$

Και από τον ορισμό της μέσης τιμής

$$Sum_n = \sum_{i=1}^{n}(x_i^2) + n\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right)^2 - 2\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} x_i\right) \tag{11}$$

$$Sum_n = \sum_{i=1}^{n}(x_i^2) + \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2 - 2\frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2 \tag{12}$$

Οπότε τελικά:

$$Sum_n = \left(\sum_{i=1}^{n} (x_i^2) - \frac{1}{n} \left(\sum_{i=1}^{n} x_i \right)^2 \right) \tag{13}$$

Υπολογίζουμε την ποσότητα:

$$Sum_n - Sum_{n-1} = \sum_{i=1}^{n} x_i^2 - \frac{1}{n} \left(\sum_{i=1}^{n} x_i \right)^2 - \sum_{i=1}^{n-1} x_i^2 + \frac{1}{n-1} \left(\sum_{i=1}^{n-1} x_i \right)^2 \tag{14}$$

Αντικαθιστούμε:

$$\sum_{i=1}^{n} x_i = n\bar{x}_n \tag{15}$$

$$\sum_{i=1}^{n-1} x_i = \sum_{i=1}^{n} x_i - x_n = n\bar{x}_n - x_n \tag{16}$$

$$\sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n-1} x_i^2 = x_n^2 \tag{17}$$

Και είναι:

$$\begin{aligned} Sum_n - Sum_{n-1} &= x_n^2 - \frac{1}{n}(n\bar{x}_n)^2 + \frac{1}{n-1}(n\bar{x}_n - x_n)^2 \\ &= x_n^2 - n\bar{x}_n^2 + \frac{1}{n-1}(n^2\bar{x}_n^2 - 2n\bar{x}_n x_n + x_n^2) \\ &= \frac{1}{n-1}(nx_n^2 + n\bar{x}_n^2 - 2n\bar{x}_n x_n) \\ &= \frac{n}{n-1}(\bar{x}_n - x_n)^2 \end{aligned} \tag{18}$$

Και τελικά έχουμε:

$$Sum_n = Sum_{n-1} + \frac{n}{n-1}(\bar{x}_n - x_n)^2 \tag{19}$$

$$(n-1)s_n^2 = (n-2)s_{n-1}^2 + \frac{n}{n-1}(\bar{x}_n - x_n)^2 \tag{20}$$

Και τελικό αναδρομικό τύπο:

$$s_n^2 = \frac{1}{n-1}\left((n-2)s_{n-1}^2 + \frac{n}{n-1}(\bar{x}_n - x_n)^2 \right) \tag{21}$$

Σε περίπτωση εξόδου οι τύποι προσαρμόζονται όπως φαίνεται παρακάτω:

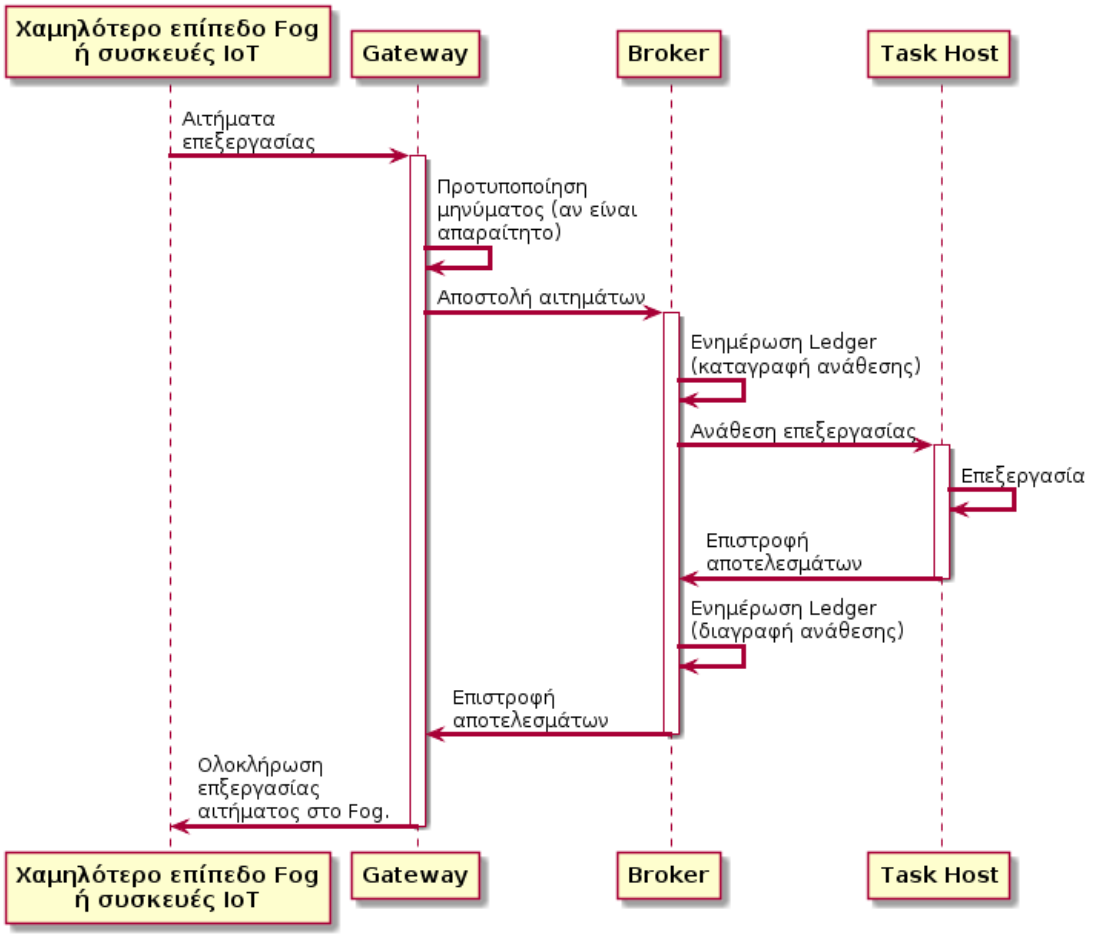
$$\bar{x}_{n-1} = \frac{n\bar{x}_n - x_n}{n-1} \tag{22}$$

$$s_{n-1}^2 = \frac{(n-1)s_n^2 - \frac{n}{n-1}(\bar{x}_n - x_n)^2}{n-2} \quad (23)$$

3.3 Σενάρια

Για να γίνει σαφής πως λειτουργεί συνολικά ένας κόμβος Fog, ο ρόλος των επιμέρους στοιχείων του, οι τρόποι που αυτά αλληλεπιδρούν και τα αναγκαία υποσυστήματα που χρειάζονται για να επιτευχθεί η λειτουργεία, παρατίθενται τα βασικά σενάρια χρήσης.

3.3.1 Κανονική λειτουργία



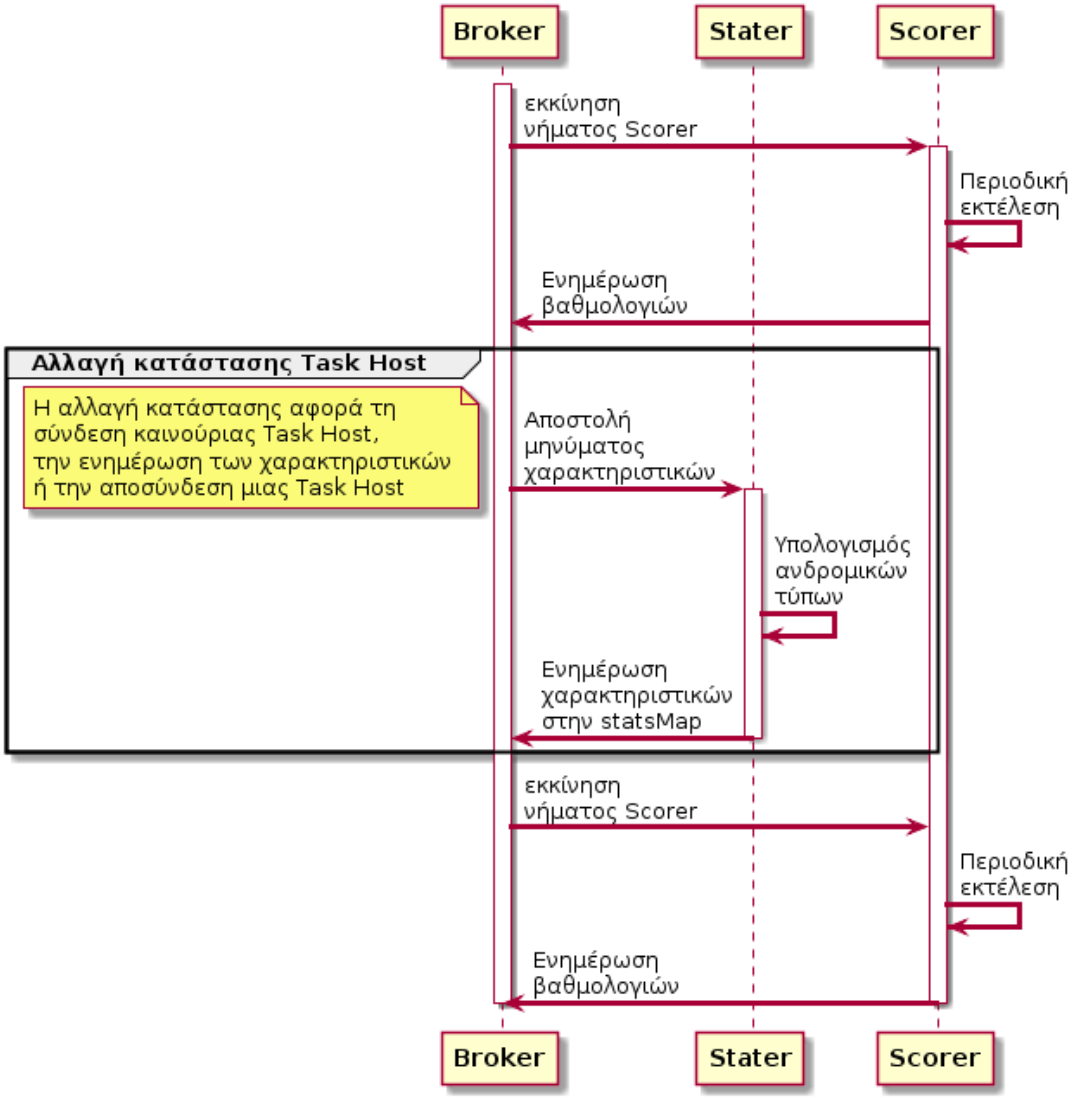
Διάγραμμα 3.3 – Διάγραμμα Αλληλουχίας Κανονικής Λειτουργίας

Το σενάριο αυτό είναι ο βασικός τρόπος λειτουργίας του δικτύου μας χωρίς απρόοπτες συνθήκες. Περιγράφει την κανονική ροή δεδομένων μέσα στον κόμβο Fog. Η Gateway έχει συλλέξει και προτυποποιήσει κατάλληλα τα δεδομένα και στέλνει αίτημα επεξεργασίας με αυτά στον Broker. Το μήνυμα αυτό έχει μοναδικό αριθμό ταυτοποίησης. Αυτός προωθεί σε κατάλληλη Task Host το μήνυμα βάσει την βαθμολογία. Υπάρχει η ανάγκη ο Broker να γνωρίζει ποιο αίτημα έχει ανατεθεί σε ποια Task Host. Για τον σκοπό αυτό χρησιμοποιεί μια δομή δεδομένων το Ledger (Κατάστιχο). Για κάθε αίτημα, ο Broker αποθηκεύει την ανάθεση του συγκεκριμένου μηνύματος στην συγκεκριμένη Task Host. Αφού η Task Host ολοκληρώσει την επεξεργασία δημιουργεί ένα μήνυμα με τον ίδιο αριθμό ταυτοποίησης με το αίτημα

και τα αποτελέσματα και τα αποστέλλει στον Broker. Αυτός ενημερώνει το Ledger για την ολοκλήρωση του συγκεκριμένου αιτήματος και αποστέλλει τα αποτελέσματα στην Gateway. Σε περίπτωση πολλαπλών Gateway, ο Broker χρειάζεται να κρατάει στο Ledger και την Gateway πού απηύθυνε το αίτημα.

3.3.2 Απόδοση Βαθμολογίας

Σημαντική Λειτουργία του Broker είναι η απόδοση βαθμολογίας στις συνδεδεμένες Task Hosts. Καθοριστικό χαρακτηριστικό της λειτουργίας βαθμολόγησης είναι το πότε και υπό ποιες συνθήκες αυτή συμβαίνει. Η πιο απλή συνθήκη είναι η διαδικασία αναβαθμολόγησης να συμβαίνει περιοδικά. Εκτός από αυτό, σημαντικά συμβάντα που τροποποιούν τις συνθήκες όπου είναι χρήσιμη η αναβαθμολόγηση είναι η αποσύνδεση ή σύνδεση μιας Task Host, η άφιξη ενός νέου αιτήματος, η ολοκλήρωση ενός αιτήματος ή η εκ νέου αποστολή χαρακτηριστικών από την Task Host.

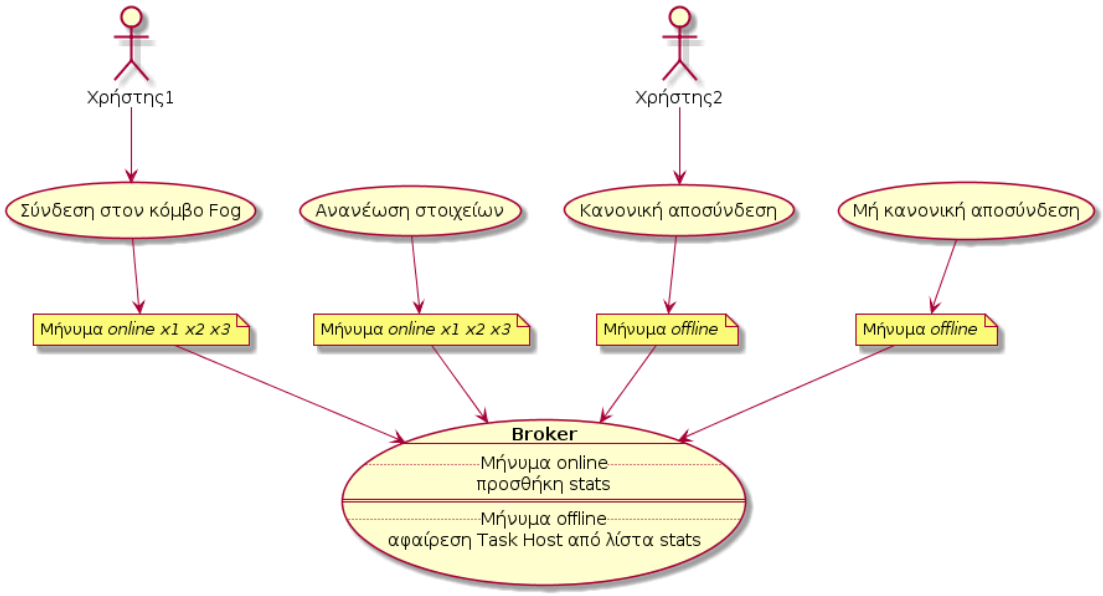


Διάγραμμα 3.4 – Διάγραμμα Αλληλουχίας Απόδοσης Βαθμολογίας

Η διαδικασία βαθμολόγησης με μεγαλύτερη ακρίβεια είναι αυτή που κάθε φορά που ο Broker χρειάζεται να αναθέσει κάποιο αίτημα σε κάποια Task Host θα ζητά τα χαρακτηριστικά των συνδεδεμένων στο δίκτυο και θα βαθμολογεί εκείνη την στιγμή.

Όμως η προσέγγιση αυτή προκαλεί καθυστερήσεις καθώς από την άφιξη ενός αιτήματος στον Broker μέχρι την ανάθεση του σε μια Task Host μεσολαβεί αναγκαία ένα χρονικό διάστημα μέχρι οι Task Hosts να αποστείλουν τα στοιχεία τους και ο Broker να τις βαθμολογήσει. Όπως αναφέρουμε και στο προηγούμενο κεφάλαιο, μπορεί ο αποδοτικότερος τρόπος αξιοποίησης της λογικής της βαθμολόγησης των Task Hosts να μην απαιτεί τόσο μεγάλη αξιοπιστία όσο ταχύτητα. Άρα η ενδεχόμενη βαθμολόγηση στις παραπάνω συνθήκες αφορά την προσπάθεια να αποφύγουμε την πληρέστερη αλλά και πιο χρονοβόρα εκδοχή του να βαθμολογούνται ξανά οι Task Hosts με κάθε καινούργιο αίτημα. Η σημασία των παραπάνω συνθηκών μας επιτρέπει να έχουμε μια σχετικά έγκυρη εικόνα, και άρα βαθμολόγηση, για την κατάσταση του συστήματος. Στην παρούσα εργασία η βαθμολόγηση των Task Hosts συμβαίνει τόσο περιοδικά όσο και κατ' εντολή. Όμως η τελική απόφαση για το πότε και πως είναι πιο αποδοτικό να γίνεται η διαδικασία βαθμολόγησης χρήζει περαιτέρω διερεύνησης με μεθόδους ανάλογες της βελτιστοποίησης της ίδιας της συνάρτησης βαθμολόγησης, κάτι που ξεφεύγει από το εύρος της παρούσας διπλωματικής και αποτελεί πεδίο μελλοντικής έρευνας.

3.3.3 Μια Συσκευή Επεξεργασίας αλλάζει κατάσταση

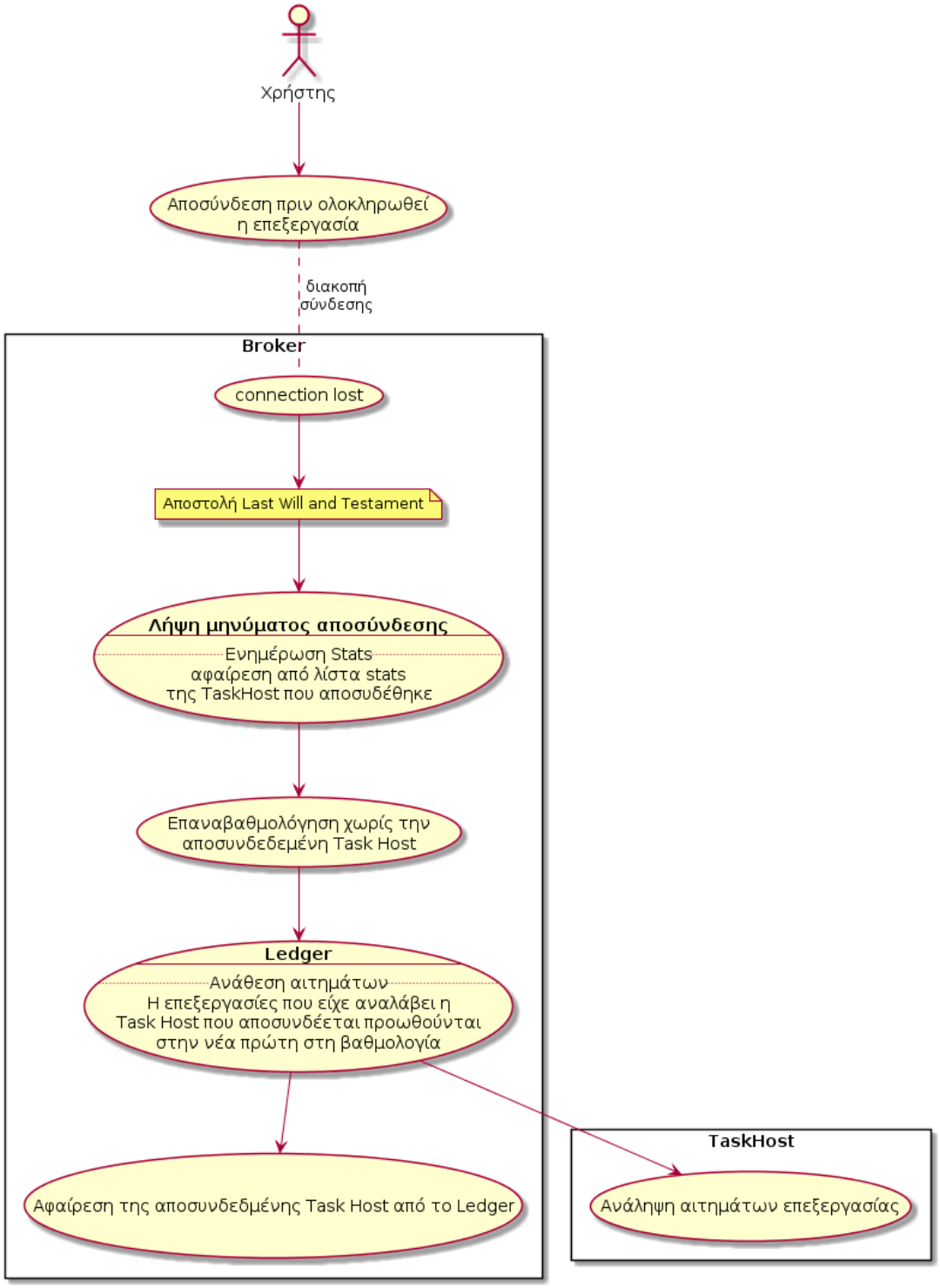


Διάγραμμα 3.5 – Διάγραμμα περιπτώσεων χρήσης: Κανονική Κατάσταση Λειτουργίας

Ο Broker για να μπορεί να βαθμολογήσει της Task Hosts, χρειάζεται να ξέρει την κατάσταση και τα χαρακτηριστικά τους. Για αυτό τον λόγο όταν μία Task Host συνδέεται αποστέλλει τα χαρακτηριστικά της σε μήνυμα της μορφής «online x1-x2-x3…». Όπου x1-x2 κλπ είναι τα χαρακτηριστικά με συγκεκριμένη σειρά {Αριθμός πυρήνων επεξεργαστή, Διαθέσιμη Μνήμη, Καθυστέρηση Δικτύου, Στάθμη Μπαταρίας, Θέση }. Ο Broker συμπληρώνει τα χαρακτηριστικά αυτά με αυτά που ο ίδιος ξέρει {Αριθμός Εργασιών, Βαθμός Εμπιστοσύνης} και τα αποθηκεύει ώστε να μπορέσει να βαθμολογήσει την Task Host όταν χρειαστεί. Οι Task Hosts περιοδικά στέλνουν ενημέρωση των χαρακτηριστικών τους και ότι παραμένουν συνδεδεμένοι (online) στην ίδια μορφή. Τέλος όταν μία Task Host αποσυνδεθεί από το δίκτυο

αποστέλλει μήνυμα «offline», ώστε ο Broker να γνωρίζει ότι δεν μπορεί να της αναθέσει κάποιο αίτημα και να την αφαιρέσει από την λίστα βαθμολογιών. Σε περίπτωση που μία Task Host αποσυνδεθεί αναπάντεχα, ο μηχανισμός του MQTT Last Will and Testament εξασφαλίζει ότι στον Broker αποστέλλεται επίσης μήνυμα που τον ενημερώνει ότι η συσκευή αποσυνδέεται. Σε αυτή την περίπτωση, αν στην Task Host είχε ανατεθεί κάποιο αίτημα, τότε ο Broker θα πρέπει να κάνει τις απαραίτητες ενέργειες όπως φαίνεται στο επόμενο σενάριο.

3.3.4 Αποσύνδεση Συσκευής Επεξεργασίας ενώ εχει μηνύματα



Διάγραμμα 3.6 – Διάγραμμα περιπτώσεων χρήσης: Αποσύνδεση πριν ολοκληρωθεί η επεξεργασία

Ένα ενδεχόμενο με μεγάλη σημασία για την καλή λειτουργία ενός δικτύου Fog είναι να χαθεί η επικοινωνία με μία Task Host ενώ αυτή έχει αναλάβει να επεξεργαστεί ένα αίτημα αλλά δεν έχει ολοκληρώσει την επεξεργασία της αποστέλλοντας στον Broker τα αποτελέσματα. Αυτή η περίπτωση αφορά ακόμα περισσότερο κόμβους Fog στα οποία συμμετέχουν φορητές συσκευές, συσκευές με ασταθή σύνδεση και γενικά Task

Hosts που είναι πιθανό να αποσυνδέονται αναπάντεχα για διάφορους λόγους. Τέτοιοι κόμβοι μπορεί να υπάρχουν για παράδειγμα σε δημόσιους χώρους με τον ρόλο των Task Hosts να καλύπτεται από τις συσκευές των περαστικών, οι οποίες λόγω απομάκρυνσης εκτός εμβελείας, ασταθούς σύνδεσης ή και απενεργοποίησης των λειτουργιών διασύνδεσης της συσκευής από τον χρήστη να υπάγονται στην παραπάνω περίπτωση.

Την περίπτωση αυτή διαχειρίζεται ο Broker με τη βοήθεια της δομής δεομένων Ledger που αναφέρθηκε και παραπάνω. Όταν μια Task Host αποσυνδέεται, ο Broker λαμβάνει ένα μήνυμα αποσύνδεσης. Με την λήψη ενός τέτοιου μηνύματος, ο Broker αφαιρεί από την λίστα των βαθμολογιών την Task Host, όπως αναφέρουμε και παραπάνω και μετά ελέγχει αν υπάρχουν αναθέσεις για επεξεργασία στην συγκεκριμένη Task Host, ελέγχοντας αν υπάρχει εγγραφή για αυτή στο ledger. Αν υπάρχει, δηλαδή αν η Task Host έχει αναλάβει αιτήματα που δεν έχει ολοκληρώσει, ο Broker αναθέτει αυτά σε νέες Task Host και μετά αφαιρεί την Task Host που βγήκε offline από το ledger.

Το παραπάνω σενάριο επιφέρει σημαντικές καθυστερήσεις, καθώς υπάρχει ένα ασυμπίεστο χρονικό διάστημα μέχρι ο Broker να καταλάβει ότι η Task Host έχει αποσυνδεθεί. Διάστημα στο οποίο δεν γίνεται καμία πρόοδος στην επεξεργασία του αιτήματος, αφού αυτό θα πρέπει τελικά να ανατεθεί σε νέα Task Host. Για τον λόγο αυτό εισήχθει στην συνάρτηση βαθμολόγησης τόσο η θέση όσο και πολύ περισσότερο ο βαθμός εμπιστοσύνης ώστε να προλαμβάνονται κατά το δυνατόν τέτοια σενάρια.

Κεφάλαιο 4: Υλοποίηση

4.1 Δομή μηνυμάτων

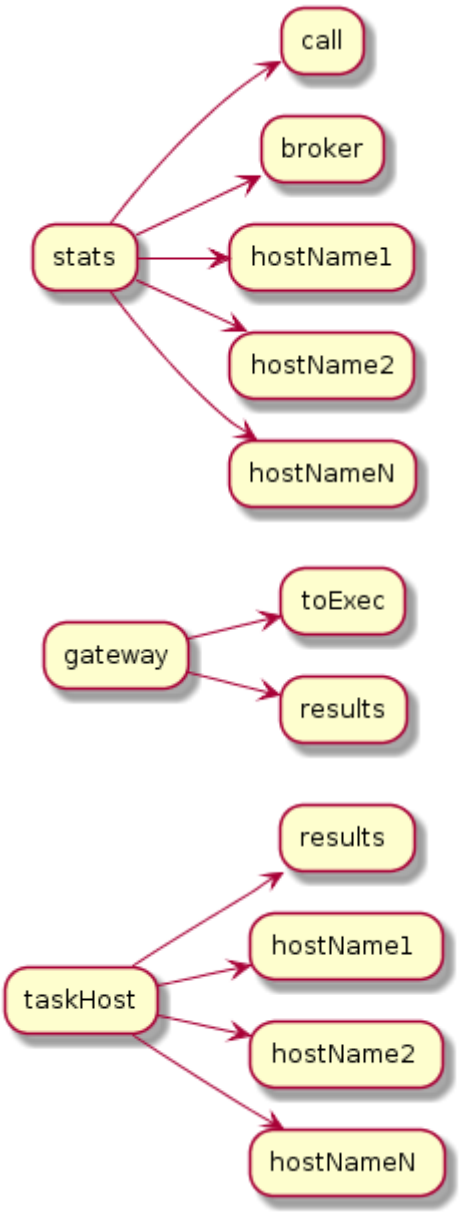
Τα αιτήματα επεξεργασίας που προωθεί η Gateway στον κόμβο Fog είναι μορφοποιημένα σύμφωνα με το πρότυπο JSON. Η μορφοποίηση βασίζεται στο παράδειγμα μηνύματος της εργασίας του Ανδρέα Καψάλη [1] με μικρές αλλαγές και προσθήκες. Σε αυτό υπάρχουν 3 βασικά πεδία και ακόμα ένα που το προσθέτει η Gateway. Αυτά είναι data όπου περιέχονται τα δεδομένα προς επεξεργασία, execution script που περιλαμβάνει τον κώδικα που πρέπει να εκτελεστεί για τα παραπάνω δεδομένα, τα metadata που χαρακτηρίζουν το αίτημα ως προς την χρονική του ευαισθησία ή οτιδήποτε άλλο χρειαστεί και το id που προσθέτει η Gateway για να μπορεί να ταυτοποιούνται τα μηνύματα.

Σε σχέση με την αρχική μορφή του μηνύματος όπως παρουσιάζεται στο [1], δεν περιλαμβάνονται metadata που δεν πρόκειται να χρησιμοποιηθούν στα πλαίσια της παρούσας εργασίας, περιλαμβάνεται κώδικας τόσο στην αρχική Python, όσο και σε Javascript, και προστέθηκε ο αύξων αριθμός ταυτοποίησης.

Η συγκεκριμένη εφαρμογή γράφτηκε σε Java και άρα εκτελείται σε JVM (Java Virtual Machine), το οποίο μπορεί να εκτελέσει κώδικα Javascript. Με αυτόν τον τρόπο δεν χρειάζεται να γίνει κλήση συστήματος για να εκτελεστεί ο κώδικας, όπως στην περίπτωση της Python. Επίσης, δεν απαιτείται από την συσκευή που εκτελείται η TaskHost να έχει εγκατεστημένη την Python. Στην περίπτωση του android η Java εκτελείται σε ART (Android Runtime) [2] και όχι στην JVM, η οποία δεν υποστηρίζει εκ των προτέρων εκτέλεση Javascript, αλλά με την προσθήκη της βιβλιοθήκης LiquidCore [3] αυτό καθίσταται δυνατό.

4.2 Δομή Θεμάτων (Topics)

Όπως αναφέρθηκε παραπάνω το MQTT χρησιμοποιεί topics για την επικοινωνία των clients. Παρακάτω παρουσιάζεται η δομή των topics.



Διαγράμμα 4.1 – Δομή Θεμάτων (topics)

Όπου hostName1, hostName2, hostNameN είναι τα ονόματα των Task Host που είναι συνδεδεμένες. Πρακτικά κάθε Task Host που συνδέεται δημιουργεί το αντίστοιχο topic με το όνομά της.

Τα topics gateway/ αφορούν την επικοινωνία του Broker με την Gateway. Τα taskHost/ αφορούν την επικοινωνία του Broker με τις Task Host. Ενώ τα stats/ την επικοινωνία που αφορά τα χαρακτηριστικά των Task Hosts και του Broker.

Συγκεκριμένα:

- Ο Broker κάνει subscribe στα topics gateway/toExec, taskhost/results, stats/#. Και κάνει publish στα gateway/results, stats/call, stats/broker και σε όλα τα taskHost/"hostNameN".
- Η Gateway κάνει subscribe στα topics taskhost/results.
  Και κάνει publish στα gateway/toExec.
- Η Task Host κάνει subscribe στα topics stats/call, stats/broker και στο taskHost/"hostName", όπου "hostName" είναι το όνομά της.
  Και κάνει publish στα taskHost/results, stats/"hostName".

Παρακάτω παρουσιάζουμε την υλοποίηση των κλάσεων καθώς και βασικών υποσυστημάτων αυτών.

4.3 Υλοποίηση Κλάσεων

Στο υποκεφάλαιο αυτό περιγράφονται οι κλάσεις της εφαρμογής καθώς και ο τρόπος λειτουργίας τους. Για τον τρόπο λειτουργίας υπάρχουν αναφορές σε σχόλια μέσα στον πηγαίο κώδικα. Ο πηγαίος κώδικας Επισυνάπτεται στο τέλος στο Παράρτημα Α

4.3.1 Πακέτο Broker

4.3.1.1 Broker

Ο Broker αναλαμβάνει τον συντονισμό του κόμβου Fog. Αυτό σημαίνει ότι θα πρέπει να λαμβάνει όλα τα μηνύματα που τον αφορούν. Δηλαδή τα μηνύματα προς επεξεργασία (αιτήματα) από την Gateway, τις απαντήσεις που προκύπτουν από την επεξεργασία των αιτημάτων από τις Task Host και τα μηνύματα με την κατάσταση των Task Hosts (online/offline και χαρακτηριστικά). Επιπλέον θα πρέπει να έχει τρόπο για να αποδίδει βαθμολογία στις Task Hosts (ανά τακτά χρονικά διαστήματα ή όταν απαιτείται) και να κατανέμει κατάλληλα τα νέα αιτήματα. Τέλος θα πρέπει, όπως φάνηκε και από τα σενάρια στο κεφάλαιο 4, να κρατάει ποια Task Host έχει αναλάβει ποιο αίτημα. Ακολουθεί ο κώδικας και επεξήγηση αυτού.

Για την υλοποίηση των παραπάνω ο Broker χρησιμοποιεί τις παρακάτω δομές δεδομένων.

- statsMap, τύπου Hashtable<String, Attributes > στο οποίο αποθηκεύονται τούπλες με το όνομα κάθε Task Host (String) ως κλειδί και μία κλάση Attributes στην οποία κρατιούνται τα τρέχοντα χαρακτηριστικά που αντιστοιχούν στην συγκεκριμένη Task Host. Καθώς στην statsMap διαβάζουν και γράφουν, όπως περιγράφεται παρακάτω τόσο το νήμα Scorer, όσο και το νήμα Stater, επιλέχθηκε η δομή Hashtable που είναι synchronized.
- trustMap, τύπου Hashtable <String, Integer> στο οποίο αποθηκεύονται τούπλες με το όνομα κάθε Task Host (String) ως κλειδί και τον βαθμό εμπιστοσύνης που της αντιστοιχεί σε Integer. Αν και η υλοποίησή μας αξιοποιεί στην βαθμολόγηση τον βαθμού εμπιστοσύνης, η απόδοση του βαθμού εμπιστοσύνης γίνεται με τυχαίο τρόπο. Ανάλογα με την χρήση και την

συγκεκριμένη εφαρμογή, η trustMap μπορεί να δημιουργείται από κάποιο τοπικό αρχείο ή να ζητείται από κάποια κεντρική βάση δεδομένων.

- scoreSet, τύπου TreeSet<Coo> στο οποίο αποθηκεύονται αντικείμενα Coo. Όπως περιγράφεται και παρακάτω ένα αντικείμενο Coo φυλάει το όνομα μιας Task Host και την βαθμολογία που της έχει αποδοθεί και υλοποιεί την διεπαφή (interface) Comparable. Με αυτό τον τρόπο η scoreSet είναι πάντοτε ταξινομημένη με φθίνουσα σειρά.
- scoreSetLatency, τύπου TreeSet<Coo>. Ακριβώς όπως και η scoreSet, αλλά κρατάει την βαθμολογία με την δεύτερη συνάρτηση βαθμολόγησης.
- Ledger, τύπου Hashtable<String, Hashtable<String, MqttMessage>>. Πρόκειται ουσιαστικά για ένα Hashtable με κλειδί το όνομα μιας Task Host στην οποία αντιστοιχίζεται ένα άλλο Hashtable που απαρτίζεται από ζευγάρια <id, MqttMessage>. Χρησιμοποιείται για να γνωρίζουμε ποια Task Host έχει αναλάβει ποιο αίτημα και να μπορούμε να ανασύρουμε το αίτημα σε περίπτωση που χρειαστεί να το προωθήσουμε σε άλλη Task Host (περιπτώσεις αποσύνδεσης πριν ολοκληρωθεί η επεξεργασία κλπ). Επίσης ένα από τα χαρακτηριστικά με βάση το οποίο αποδίδεται βαθμολογία είναι ο αριθμός εργασιών, χαρακτηριστικό που ο Broker γνωρίζει από το μέγεθος του παραπάνω Hashtable για κάθε Task Host.
- Πίνακες (Arrays) average και sigma στους οποίους αποθηκεύονται οι μέσοι όροι και οι αποκλίσεις για κάθε ένα χαρακτηριστικό με την σειρά που αναφέρονται και στο υποκεφάλαιο της Συνάρτησης Βαθμολόγησης

Για τις παραπάνω δομές δεδομένων έχουν οριστεί και οι αντίστοιχες μέθοδοι που χρειάζονται για να γίνουν αλλαγές σε αυτές, να ελεγχθεί αν έχουν κάποιο στοιχείο, να διαβαστεί κάποιο στοιχείο κλπ.

Κατά την εκκίνησή του, ο Broker συνδέεται στον MQTT Broker, με clientId: broker (σχόλιο 1). Στο Paho, ορίζεται ένα interface(διεπαφή), το MqttCallback, το οποίο περιγράφει μεθόδους απαραίτητες για την υλοποίηση ενός MQTT client. Ο Broker ορίζει ως Callback τον εαυτό του (σχόλιο 2) και παρακάτω υλοποιεί τις μεθόδους του Callback (σχόλιο 5). Κάνει subscribe στα topics που χρειάζεται (σχόλιο 3) και ξεκινάει την εκτέλεση δύο Threads, ενός Updater και ενός Scorer, οι οποίοι εκτελούνται περιοδικά (σχόλιο 4) και περιγράφονται παρακάτω.

Στην περίπτωση εισερχόμενου αιτήματος ενεργοποιείται η αντίστοιχη μέθοδος messageArrived του Paho. Εάν το μήνυμα αφορά αίτημα επεξεργασίας από την Gateway (σχόλιο 6), ο Broker διαβάζει από αυτό τον αύξοντα αριθμό του μηνύματος, id (σχόλιο 8) και ελέγχει αν το αίτημα είναι υψηλής ευαισθησίας ως προς την καθυστέρηση του δικτύου (σχόλια 9,10) για να επιλέξει την κατάλληλη συνάρτηση βαθμολόγησης (σχόλιο 11). Στην συνέχεια αναθέτει την επεξεργασία στην Task Host με την υψηλότερη βαθμολογία (σχόλιο 12) και καταγράφει την ανάθεση στο ledger (σχόλιο 13).

Όταν η επεξεργασία ενός αιτήματος ολοκληρωθεί, η Task Host επιστρέφει τα αποτελέσματα στο αντίστοιχο topic (σχόλιο 14). Ο Broker διαβάζει το μήνυμα (σχόλιο

15) και ενημερώνει κατάλληλα το ledger, ελέγχοντας και για περιπτώσεις όπου δεν έχει αποθηκεύμενο τέτοιο μύνημα (σχόλια 16, 17) και προωθεί τα αποτελέσματα στην Gateway (σχόλιο 18).

Κάθε φορά που μια Task Host στέλνει ενημέρωση για την κατάστασή της (online/offline) και τα χαρακτηριστικά της (σχόλιο 19) ο Broker καλεί έναν Stater, ο οποίο περιγράφεται παρακάτω, για να ενημερώσει κατάλληλα τις τιμές των χαρακτηριστικών (σχόλιο 20). Σε περίπτωση που η Task Host έχει αλλάξει κατάσταση από online σε offline καλείται ένας Scorer για να ενημερωθούν οι βαθμολογίες και να αφαιρεθεί η συγκεκριμένη Task Host (σχόλιο 21). Στην περίπτωση αυτή πρέπει να ελέγξει αν η συγκεκριμένη Task Host είχε ολοκληρώσει την επεξεργασία όλων των αιτημάτων που είχε αναλάβει. Αν έχουν απομείνει μη ολοκληρωμένα αιτήματα, τα αναθέτει σε άλλη Task Host, κάνοντας τις κατάλληλες ενέργειες στο ledger. Δηλαδή αφαιρώντας τις εγγραφές που αφορούν την παλιά Task Host Και προσθέτοντας εγγραφές για την νέα (σχόλιο 22, 23).

4.3.1.2 Stater

Ο Stater είναι ένα νήμα που αναλαμβάνει την διαχείριση των στατιστικών κάθε Task Host που είναι συνδεδεμένη στον κόμβο Fog. Συγκεκριμένα, ο Stater ενεργοποιείται κάθε φορά που έρχεται στον Broker ένα μήνυμα που αφορά σύνδεση νέας Task Host ή ενημέρωση των στατιστικών μιας ήδη συνδεδεμένης, όπως και όταν μια Task Host αποσυνδέεται (μήνυμα σε topic που περιέχει stats, σχόλιο 19 του Broker). Πέρα από την ενημέρωση του statsMap, δηλαδή της δομής στην οποία κρατά ο Broker τα στατιστικά ο Stater διαχειρίζεται τον αναδρομικό υπολογισμό του μέσου όρου (average) και της τυπικής απόκλισης (sigma) για κάθε στατιστικό. Η τυπική απόκλιση, αξιοποιείται στον υπολογισμό της βαθμολογίας των Task Host.

Για την υλοποίηση των παραπάνω ο Stater ενημερώνει τις ακόλουθες δομές δεδομένων του Broker, avg[], sigma[], statsMap<>.

Κατά την εκκίνησή του ο Stater διαβάζει το μήνυμα που του έχει περάσει ο Broker και παίρνει το όνομα της Task Host και το περιεχόμενό του. Αν το περιεχόμενο του μηνύματος ξεκινά με την λέξη online, δηλαδή πρόκειται για νέα Task Host ή ενημέρωση των στατιστικών, ο Stater δημιουργεί ένα καινούριο αντικείμενο Attributes (δες παρακάτω) με τα στατιστικά του μηνύματος και ενημερώνει την statsMap (σχόλιο 12). Αν η Task Host υπάρχει ήδη στο statsMap, πρέπει να αφαιρεθεί από αυτή και να υπολογιστούν ο μέσος όρος και η τυπική απόκλιση πριν προστεθεί με τα νέα στατιστικά και υπολογιστούν εκ νέου μέση τιμή και τυπική απόκλιση (σχόλιο 13, επόμενη παράγραφος αναλυτικά). Σε περίπτωση αποσύνδεσης Task Host, το περιεχόμενο του μηνύματος είναι offline και ο Stater αναλαμβάνει να την αφαιρέσει από την statsMap, να υπολογίσει και να ενημερώσει τους average[] και sigma[] (σχόλιο 14).

Πιο συγκεκριμένα, όταν μια Task Host συνδέεται ο Stater, στην μέθοδο hostOnline, ελέγχει αρχικά αν είναι η μοναδική που υπάρχει αυτή τη στιγμή, αν δηλαδή είναι άδεια η statsMap (σχόλιο 1). Σε αυτή την περίπτωση ορίζει κατάλληλα τη μέση τιμή

και την τυπική απόκλιση (σχόλια 2,3) και αυξάνει τον μετρητή του αριθμού των Task Host (σχόλιο 4). Αν υπάρχουν και άλλες Task Host, αυξάνεται ο μετρητής και υπολογίζονται οι μέσες τιμές και οι τυπικές αποκλίσεις για κάθε ένα από τα 7 στατιστικά (σχόλια 5,6,7). Όταν αποσυνδέεται μια Task Host υπολογίζονται οι μέσες τιμές και οι τυπικές αποκλίσεις για κάθε ένα από τα 7 στατιστικά και μειώνεται ο μετρητής των Task Host (μέθοδος hostOffline, σχόλια 8,9,10)

4.3.1.3 Scorer

Ο Scorer είναι το νήμα που αναλαμβάνει την βαθμολόγηση των Task Host. Χρησιμοποιεί τις statsMap, average και sigma του Broker για νας ενημερώσει δύο TreeSet το scoreSet και το scoreSetLatency, στα οποία αποθηκεύονται οι βαθμολογίες των Task Host σε φθίνουσα σειρά . Στο πρώτο βρίσκονται οι γενικές βαθμολογίες ενώ στο δεύτερο οι βαθμολογίες για time sensitive tasks.

Κατά την εκκίνηση ο Stater παίρνει από τον Broker τα στατιστικά των Task Hosts, τις μέσες τιμές και τις τυπικές αποκλίσεις (σχόλιο 1), χρησιμοποιόντας τις αντίστοιχες μεθόδους του Broker, getStatsMap(), getSigma(), getAverage() και αδειάζει τα scoreSet και scoreSetLatency από τις παλιές βαθμολογίες (σχόλιο 2). Στη συνέχεια διατρέχει τη statsMap για να αποδώσει βαθμολογία σε κάθε Task Host (σχόλιο 3) λαμβάνοντας υπόψη και την περίπτωση μηδενικής τυπικής απόκλισης (σχόλιο 4) και ενημερώνει με τις νέες βαθμολογίες τα scoreSet και scoreSetLatency, καλώντας τις μεθόδους setScoreSet() και setScoreLatency() του Broker.

4.3.1.4 Updater

Ο Updater είναι ένα νήμα που εκτελείται περιοδικά και αναλαμβάνει να ζητά από τις συνδεδεμένες Task Hosts τα στατιστικά τους.

Το παραπάνω το επιτυγχάνει καλώντας κάθε φορά που τρέχει την μέθοδο callForStats() του Broker, με την οποία στέλνεται στο topic stats/call ένα μήνυμα που καλεί της συνδεδεμένες Task Hosts να αποστείλουν τα χαρακτηριστικά τους

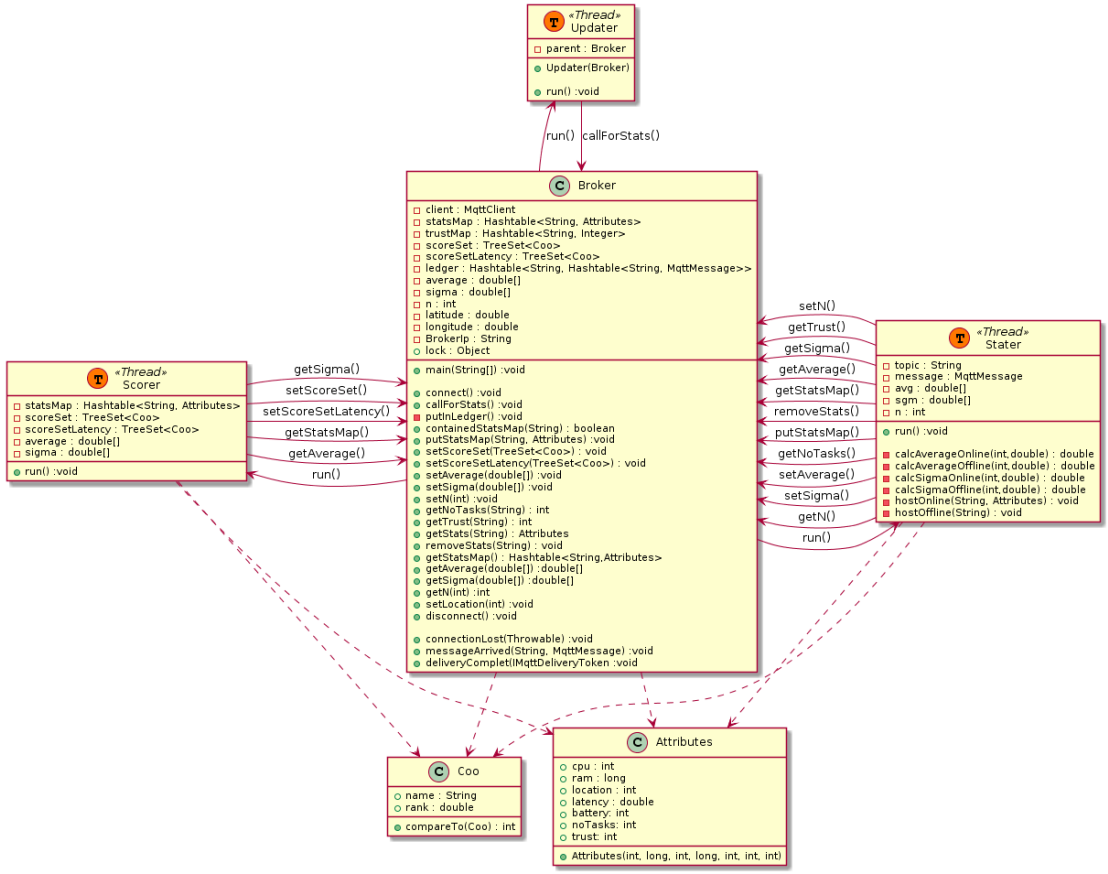
4.3.1.5 Οι κλάσεις Coo και Attributes

Οι κλάσεις Coo και Attributes είναι βοηθητικές.

Η πρώτη είναι η μορφή στην οποία αποθηκεύονται οι βαθμολογίες, μία τούπλα (String, Double) με το όνομα και την βαθμολογία μιας Task Host και υλοποιεί το interface comparable για να μπορούν να συγκριθούν οι βαθμολογίες.

Η Attributes είναι η κλάση που περιγράφει τα στατιστικά των Task Hosts σε 7 μεταβλητές (cpu, ram, location, latency, abattery, noTasks, trust). Για τα στατιστικά latency και noTasks που είναι καλύτερα όσο μικρότερα είναι, κάνει τον απαραίτητο έλεγχο διαίρεσης με το 0 και τα αποθηκεύει ως 1/v.

Συνολικά το όλο πακέτο του Broker περιγράφεται από το παρακάτω διάγραμμα:



Διάγραμμα 4.2 - Διάγραμμα κλάσεων: Πακέτο Broker

4.3.2 Gateway

Η Gateway είναι υπεύθυνη για την αποστολή αιτημάτων επεξεργασίας που λαμβάνει από συσκευές IoT ή άλλους κόμβους σε τυποποιημένη μορφή. Είναι επίσης αυτή που ελέγχει τον συνολικό χρόνο που χρειάστηκε για την ολοκλήρωση της επεξεργασίας ενός μηνύματος.

Για τον υπολογισμό του χρόνου που χρειάζεται ένα μήνυμα από την στιγμή που στέλνεται στον Broker μέχρι να ολοκληρωθεί η επεξεργασία του και η gateway να λάβει απάντηση, χρησιμοποιείται ένα Hashtable<Integer, Long>, το ledger. Σε αυτό κρατάει για κάθε id μηνύματος (Integer), τον χρόνο αποστολής του (Long).

Κατά την εκτέλεσή της η Gateway αρχικά συνδέεται στον Broker (σχόλιο 1) και κάνει subscribe στο topic fromBroker/results, εκεί δηλαδή που ο Broker επιστρέφει τα αποτελέσματα των αιτημάτων επεξεργασίας (σχόλιο 2). Στη συνέχεια περιμένει να διαβάσει μηνύματα JSON που θέλει να δώσει ο χρήστης προς επεξεργασία μέχρι να της δοθεί εντολή να τερματίσει τη λειτουργία της στέλνοντας exit (σχόλια 3,8). Μετατρέπει τα μηνύματα που λαμβάνει σε String (σχόλιο 4) και στέλνει το αίτημα επεξεργασίας στον Broker (σχόλιο 5). Για κάθε αίτημα που στέλνει σημειώνει τον χρόνο αποστολής τον οποίο αποθηκεύει σε ένα ledger (σχόλιο 6) και αυξάνει τον

μετρητή των μηνυμάτων (σχόλιο 7) που λειτουργεί σαν αύξων αριθμός ταυτοποίησής τους σε όλη την κίνησή τους στον κόμβο Fog.

Στην Gateway υλοποιούμε κατάλληλα την μέθοδο messageArrived του interface MqttCallback για να βρούμε τον συνολικό χρόνο για την ολοκλήρωση της επεξεργασίας ενός μηνύματος (σχόλιο 9). Έτσι όταν έρχεται ένα νέο μήνυμα στο fromBroker/results, η Gateway το τυπώνει και υπολογίζει τον συνολικό χρόνο επεξεργασίας (σχόλιο 10,11). Αν το μήνυμα δεν υπάρχει στον ledger που κρατιούνται τα μηνύματα που είχε αυτή στείλει, τυπώνει μήνυμα λάθους (σχόλιο 12)

4.3.3 Πακέτο TaskHost

4.3.3.1 TaskHost

Η κλάση TaskHost υλοποιεί τον ρόλο της συσκευής επεξεργασίας, είναι δηλαδή υπεύθυνη τόσο για την ενημέρωση του Broker για τα στατιστικά της όταν αυτά ζητηθούν, όσο και για την εκτέλεση των αιτημάτων που στέλνονται στον κόμβο Fog μέσω της Gateway από άλλα μέρη του δικτύου. Μια ειδική υπολειτουργία που σχετίζεται με την TaskHost είναι αυτή του προσδιορισμού της θέσης της Task Host ώστε να βαθμολογηθεί κατάλληλα. Αυτό στην περίπτωση του προσομοίωσης αναλαμβάνει η κλάση Geolocator που καλεί η TaskHost όπως θα δούμε παρακάτω.

Κατά την εκτέλεσή της η TaskHost αφού δημιουργήσει ένα νήμα Geolocator (σχόλιο 1), βρίκσκει το όνομα της συγκεκριμένης Task Host, την διεύθυνση MAC της, στην περίπτωση αυτή (σχόλιο 2), ελέγχοντας ότι αυτή είναι πραγματική (σχόλιο 3). Πριν συνδεθεί στον Broker με όνομα host την MAC που βρήκε παραπάνω (σχόλιο 5), πρέπει να οριστεί το μήνυμα last will and testament που αποστέλλεται σε περίπτωση που η Task Host βγει αιφνίδια offline (σχόλιο 4) και αφού γίνει η σύνδεση να κάνει subscribe στα topics που ζητά ο broker τα στατιστικά (stats/call) και στέλνει τα μηνύματα προς επεξεργασία (fromBroker/hostname –σχόλιο 6). Στη συνέχεια δημοσιεύει πρώτη φορά τα στατιστικά της στο topic stats/hostname (σχόλιο 7) τα οποία βρίσκει με τη μέθοδο getStat (σχόλιο 8). Στην περίπτωση taskHost που δεν εκτελείται σε φορητή συσκευή και περιβάλλον android (περίπτωση που περιγράφεται παρακάτω), τα μοναδικά χαρακτηριστικά που είναι πραγματικά είναι οι διαθέσιμοι πυρήνες cpu, η μνήμη και η καθυστέρηση (latency) της σύνδεσης με τον Broker. Η στάθμη της μπαταρίας αποδίδεται με τυχαίο τρόπο στην αρχή της εκτέλεσης του Task Host, ενώ η ζώνη και η κατεύθυνση καθορίζονται από την Geolocator, όπως περιγράφεται παρακάτω. Η καθυστέρηση του δικτύου (latency) γίνεται γνωστή με την βοήθεια του Latencer, όπως περιγράφεται παρακάτω. Ο Latencer καλείται παρειοδικά από την TaskHost (σχόλιο 2).

Όπως και στις παραπάνω κλάσεις η TaskHost υλοποιεί την μέθοδο messageArrived του MqttCallback (σχόλιο 11). Όταν λάβει μήνυμα από τον Broker για να αποστείλει χαρακτηριστικά, λαμβάνει από την συσκευή τα τρέχοντα χαρακτηριστικά και τα αποστέλλει στο κατάλληλο topic (σχόλιο 12), με τον τρόπο που περιγράφηκε και στο πρώτο μήνυμα.

Σε περίπτωση που έρθει αίτημα για επεξεργασία (σχόλιο 13), σημειώνουμε την στιγμή της αρχής της εκτέλεσης (σχόλιο 14). Στην συνέχεια διαβάζει και αναλύει (parse) το JSON μήνυμα στα επιμέρους πεδία του (σχόλιο 15). Λαμβάνει το πεδίο id (σχόλιο 16) και τον κώδικα που αντιστοιχεί στην Python (σχόλιο 17). Γράφει σε ένα αρχείο event.json όλο το json μήνυμα (σχόλιο 18) και σε ένα αρχείο script.py τον κώδικα της python (σχόλιο 19). Στην συνέχεια ζητάει από το λειτουργικό να τρέξει τον κώδικα (σχόλιο 20) και διαβάζει τα αποτελέσματα (σχόλιο 21). Σε περίπτωση που αποτύχει (σχόλιο 22) επαναλαμβάνει την διαδικασία για την javascript (σχόλιο 23). Δηλαδή διαβάζει τον κώδικα javascript (σχόλιο 24), τον τρέχει (σχόλιο 25) και διαβάζει το αποτέλεσμα (σχόλιο 26). Αυτό το αποστέλλει μαζί με το όνομα της συγκεκριμένης Task Host και το id του μηνύματος (σχόλιο 27) και σημειώνει τον χρόνο που χρειάστηκε για αυτή την επεξεργασία (σχόλιο 28).

4.3.3.2 Geolocator

Η κλάση Geolocator ουσιαστικά πρόκειται για μια εξομοίωση κίνησης σε geofences όπως αυτή περιγράφεται αναλυτικά παρακάτω, στην υλοποίηση της εφαρμογής android. Με αυτό τον τρόπο μπορεί μια σταθερή Task Host (π.χ. ένας σταθερός υπολογιστής) να λειτουργήσει σαν κινούμενη όσον αφορά τα χαρακτηριστικά της. Ουσιαστικά ορίζονται 10 ζώνες με κέντρο τον Broker και αποδίδεται μια αρχική θέση και κατεύθυνση στον Task Host. Στην συνέχεια εξομοιώνεται μια ευθύγραμμη κίνηση με βάση τις αρχικές συνθήκες. Οι ζώνη μπορεί να πάρει τιμές στο εύρος (1,10) και η κατεύθυνση ορίζεται ως -1 προς τα έξω (απομάκρυνση από τον Broker) και 1 προς τα μέσα (προς τον Broker). Έτσι δημιουργείται μια αύξουσα κλίμακα στο εύρος (-9,9) πολλαπλασιάζοντας την ζώνη στην οποία βρίσκεται η Task Host με την κατεύθυνσή της.

Κατά την εκκίνηση αποδίδονται τιμές στα zone και direction (σχόλιο 1) και θέτε αντίστοιχα το location, της TaskHost που αντιστοιχεί στην συγκεκριμένη Geolocator, ως zone*direction (σχόλιο 2). Στην συνέχεια εξομοιώνεται η κίνηση μέχρι να βγεί εκτός ορίων (εκτός της 10ης ζώνης) (σχόλιο 3), όπου εάν βρεθεί στην ζώνη 0 αλλάζει η κατεύθυνσή του, καθώς πια θα κινείται προς τα έξω (σχόλιο 4). Κάθε φορά που αλλάζει το zone ή το direction καλείται η μέθοδος setLocation() της Task Host, όπου θέτονται στην Task Host οι σωστές τιμές. Τέλος όταν βρεθεί εκτός των ορίων, ζητά από την TaskHost να αποσυνδεθεί με την μέθοδο disconnect().

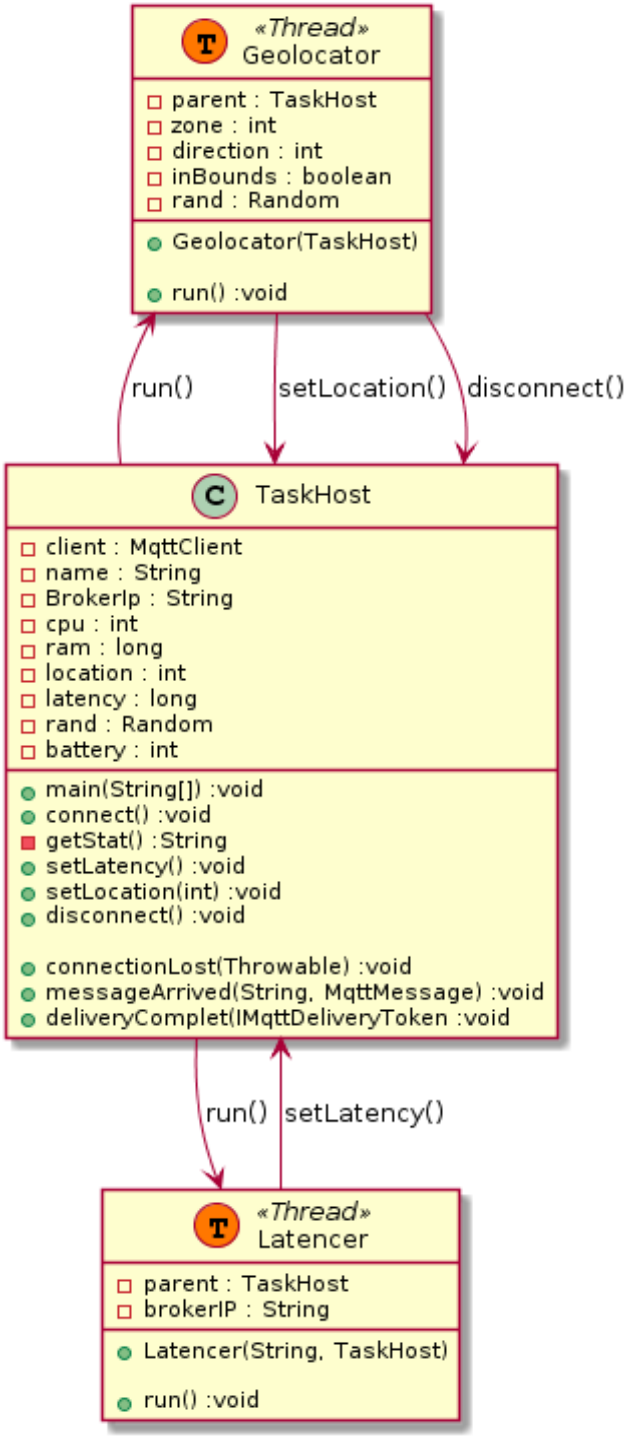
Η κλάση Geolocator παρέχεται για να μπορεί να εξομοιωθεί στοιχειωδώς η κίνηση της Task Host. Σε περιπτώσεις σταθερών Task Host, είναι δυνατή η απενεργοποίησή της. Επίσης σε περιπτώσεις που χρησιμοποιείται μια διαφορετική μέθοδος για την αξιολόγηση κινούμενων Task Hosts, είναι εύκολη η αντικατάσταση της κλάσης Geolocator από μια διαφορετική, αρκεί τα αποτελέσματα εξόδους να έχουν την μορφή.

4.3.3.3 Latencer

Για να μπορέσει να μετρήσει η Task Host την καθυστέρηση του δικτύου χρησιμοποιείται η κλάση Latencer. Ο Latencer είναι ένα νήμα (thread) που εκτελείται

περιοδικά και υπολογίζει τον χρόνο σύνδεσης με τον Broker και άρα την καθυστέρηση του δικτύου. Καθώς ο Latencer αναμένει απαντήσεις απαντήσεις από τον Broker για να υπολογίσει την καθυστέρηση του δικτύου, είναι αναγκαίο να λειτουργεί ανεξάρτητα από την Task Host, ώστε η λειτουργία της να μην διακόπτεται λόγω αυτής της αναμονής. Προφανώς με τον τρόπο αυτό η καθυστέρηση του δικτύου υπολογίζεται, όχι την στιγμή που ζητούνται τα χαρακτηριστικά της Task Host, αλλά αποστέλλεται η τελευταία που είχε υπολογίσει ο Latencer κατά την εκτέλεσή του.

Συνολικά το όλο πακέτο της TaskHost περιγράφεται από το παρακάτω διάγραμμα:



Διάγραμμα 4.3 - Διάγραμμα κλάσεων: Πακέτο TaskHost

4.3.4 Εφαρμογη Android

Ο ρόλος της Task Host υλοποιήθηκε και σε περιβάλλον Android, έτσι ώστε να μπορεί να γίνει εξομοίωση και με πραγματικές συσκευές με λογισμικό Android. Η γενική λογική παρέμεινε η ίδια, αν και χρειάστηκαν να υλοποιηθούν μία διεπαφή με τον χρήστη, πραγματικά δεδομένα θέσης και μπαταρίας, καθώς και να γίνουν ορισμένες αλλαγές για να συμμορφώνεται με τα πλαίσια που θέτει το Android. Για τα πραγματικά δεδομένα θέσης χρησιμοποιήθηκε η διεπαφή του Google Play Services με την οποία γίνεται εύκολος ο ορισμός των Geofences και η αναγνώριση του περάσματος από ένα geofence σε άλλο, για να χρησιμοποιηθεί στην συνάρτηση βαθμολόγησης. Με τον όρο geofence περιγράφουμε εικονικά οριοθετημένες περιοχές που αξιοποιούνται από συστήματα εντοπισμού θέσης, όπως GPS κλπ, για να ελέγχεται η κίνηση και η θέση συσκευών και ανθρώπων μέσα σε αυτές. Η χρήση των geofences είναι ιδιαίτερα βολική στην περίπτωσή μας όπου μας ενδιαφέρει να εκτιμήσουμε έγκαιρα πότε μία συσκευή ενδέχεται να βγει εκτός εμβέλειας, γεγονός που αποτυπώνεται στην βαθμολογία της.

Η μεγαλύτερη διαφορά ήταν η δυσκολία δυνατότητας κλήσης άλλης εφαρμογής για την εκτέλεση του κώδικα σε Python. Για αυτόν τον λόγο υλοποιήθηκε μόνο η δυνατότητα εκτέλεσης Javascript μέσω της βιβλιοθήκης LiquidCore. Παρακάτω παρατίθενται οι βασικές κλάσεις που αποτελούν την εφαρμογή (application).

4.3.4.1 MainActivity

Πρόκειται για την κλάση η οποία τρέχει όταν εκκινά η εφαρμογή μας. Αυτή είναι υπεύθυνη για εμφάνιση της διεπαφής προς τον χρήστη και την ανάγνωση των δεδομένων που εισάγει ο χρήστης. Ταυτόχρονα αρχικοποιεί όλες τις υπηρεσίες και κλάσεις που θα χρειαστούν για την εφαρμογή μας, όπως την επικοινωνία με τα google play services για την κίνηση, τα χαρακτηριστικά της σύνδεσης mqtt κλπ.

Κατ’ αρχάς η MainActivity εμφανίζει την οθόνη όπως αυτή ορίζεται στο layout (σχόλιο 2). Όταν ο χρήστης εισάγει την IP του MQTT Broker στον οποίο θέλει να συνδεθεί και πατήσει το κουμπί “OK”, η MainActivity διαβάζει την IP (σχόλιο 4) και εκκινά έναν MqttHelper (όπως περιγράφεται παρακάτω) για να συνδεθεί. Για την σύνδεση αυτή απαιτούνται το όνομα του κινητού (σχόλιο 7), για το οποίο χρησιμοποιήθηκε το Device ID και η λίστα με τα subscriptions που θα πρέπει να γίνουν (σχόλιο 8).

Κατά την εκκίνηση δημιουργείται επίσης ένα GoogleApiClient που προσπαθεί να συνδεθεί στα google play services για να έχει πρόσβαση στην θέση του κινητού (σχόλιο 1). Εάν η σύνδεση είναι επιτυχημένη βρίσκει την τελευταία γνωστή τοποθεσία που βρισκόταν το κινητό (σχόλιο 9).

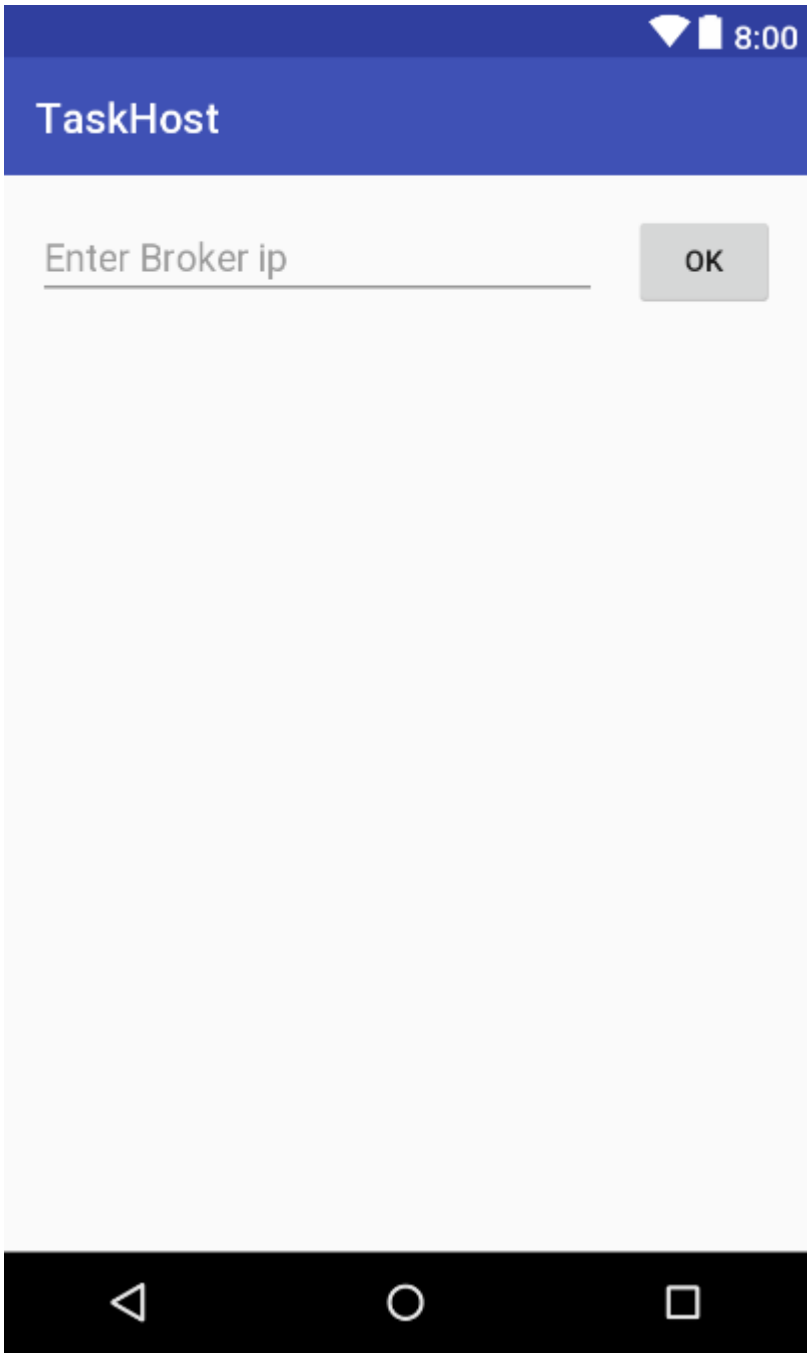
Όπως περιγράφεται στην κλάση MqttHelper παρακάτω, όταν γίνει η σύνδεση με τον Broker, διαβάζεται στο topic stats/broker η θέση του. Τότε, αφού δηλαδή έχουμε τη θέση του Broker, μπορούμε να φτιάξουμε τα Geofences (σχόλιο 10). Τα Geofences είναι 10 συνολικά με κέντρο τον Broker τα οποία έχουν ακτίνα τα ακέραια πολλαπλάσια του 50 (σχόλιο 6). Ουσιαστικά δηλαδή το πρώτο Geofence έχει ακτίνα 50 μέτρα, το δεύτερο 100, το τρίτο 150 κοκ. Έπειτα ορίζεται η κλάση

GeofenceTransitionsIntentService, ως αυτή που θα διαβάζει τα events που αφορούν την αλλαγή από geofence σε geofence και θα κάνει τις απαραίτητες αλλαγές (σχόλιο 11).

Τέλος υπάρχει η συνάρτηση setText (σχόλιο 12), η οποία εμφανίζει στην οθόνη το μήνυμα που θα τις δώσουμε σαν όρισμα.

4.3.4.2 Layout

Παρακάτω παρατίθεται η αρχική οθόνη της εφαρμογής μας. Το xml που αντιστοιχεί σε αυτή επισυνάπτεται μαζί με τον πηγαίο κώδικα στο Παράρτημα Α.



Εικόνα 4.4 – Γραφικό περιβάλλον εφαρμογής Android

4.3.4.3 GeofenceTransitionsIntentService

Πρόκειται για μια βοηθητική κλάση η οποία χρησιμεύει στο να διαβάζει τις αλλαγές που γίνονται όσον αφορά τα Geofences και την σχετική θέση της συσκευής με αυτά.

Στην ουσία κάθε φορά που η συσκευή μπαίνει ή βγαίνει από ένα geofence αυτό μεταφράζεται σε ένα event, η GeofenceTransitionsIntentService διαβάζει το συγκεκριμένο event και θέτει ως ζώνη (zone) του Task Host το αντίστοιχο geofence (σχόλιο 1). Επίσης ελέγχει αν πρόκειται για event εισόδου (enter) ή εξόδου (exit) και θέτει την κατεύθυνση (direction) ανάλογα (1 για είσοδο, -1 για έξοδο) (σχόλιο 2). Για την τρέχουσα τοποθεσία που αντλείται με τον παραπάνω τρόπο ενημερώνεται η MqttHelper με τις συναρτήσεις setZone() και setDirection().

4.3.4.4 MqttHelper

Η συγκεκριμένη κλάση είναι αυτή που αντιστοιχεί πιο καθαρά στην κλάση TaskHost όπως περιγράφηκε παραπάνω. Αναλαμβάνει την σύνδεση με τον Broker, υλοποιεί την interface(διεπαφή) του Paho, MqttCallback καθ είναι υπεύθυνη για την σωστή αποστολή των στατιστικών της συσκευής επεξεργασίας και την εκτέλεση των αιτημάτων που της ανατίθενται.

Όταν η MainActivity δημιουργεί ένα αντικείμενο της mqttHelper, της δίνει τα ορίσματα που αντιστοιχούν στο IP του Broker, στο όνομα της συσκευής και την λίστα με τα subscriptions που πρέπει να γίνουν (σχόλιο 1). Στην συνέχεια η MqttHelper συνδέεται στον Broker (σχόλιο 2) και στέλνει το πρώτο μήνυμα με τα στατιστικά της (σχόλιο 3). Επίσης καλεί για κάθε topic την subscribeToTopic (σχόλιο 4) για να κάνει subscribe σε κάθε topic. Όταν λάβει την θέση του broker στο stats/broker δημιουργεί τα κατάλληλα geofences, όπως περιγράφηκε και στην MainActivity καλώντας τις μεθόδους setGeoFence() και επίσης αρχικοποιεί την ζώνη και την κατεύθυνσή της συσκευής. Η αρχικοποίηση γίνεται θέτοντας την κατεύθυνση -1 (την χειρότερη περίπτωση δηλαδή) και την ζώνη ως την απόσταση της συσκευής από τον Broker διά 50 (σχόλιο 5).

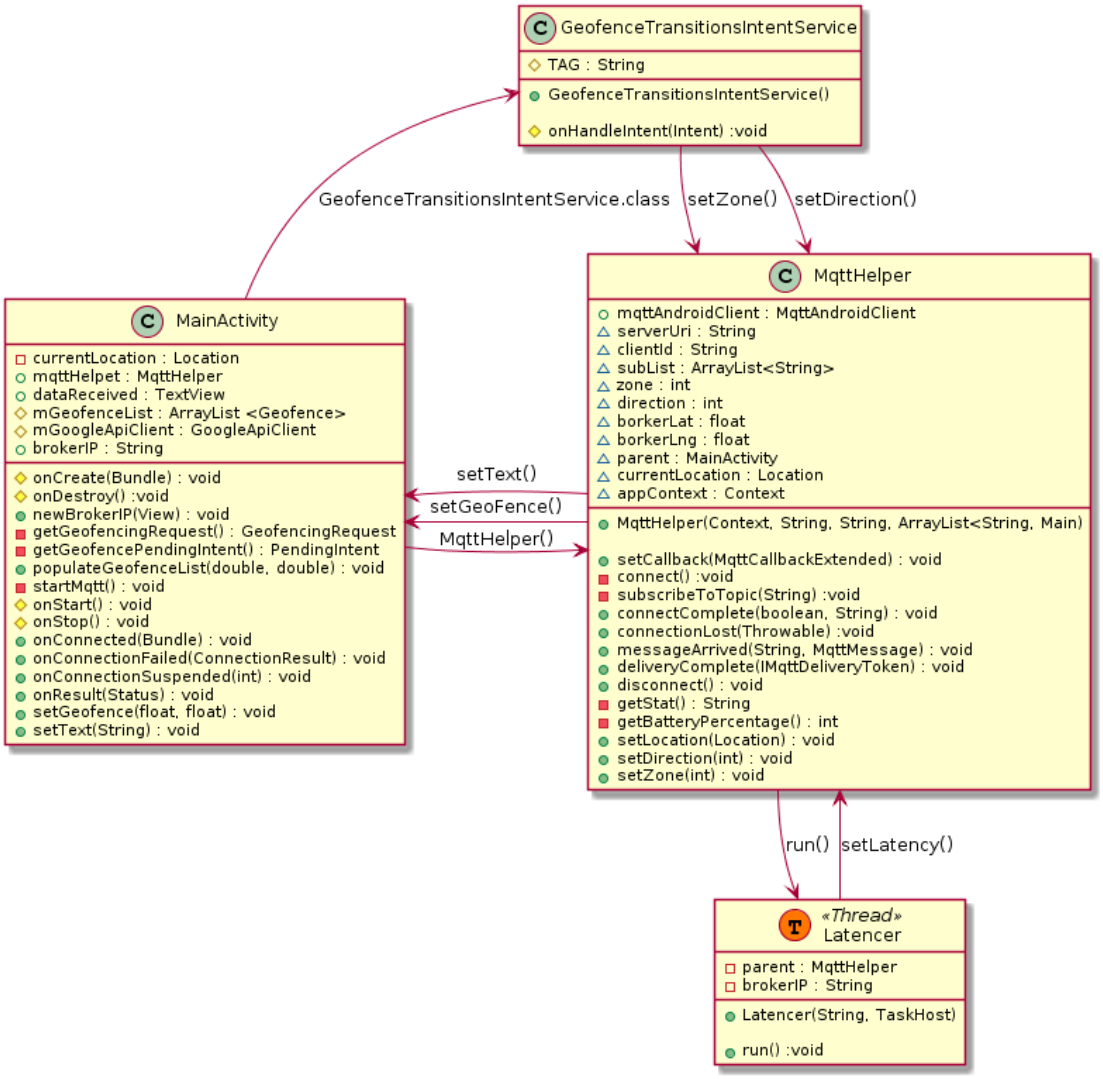
Σε περίπτωση που έρθει αίτημα για επεξεργασία (σχόλιο 6), σημειώνουμε τον χρόνο της αρχής της εκτέλεσης (σχόλιο 7). Στην συνέχεια διαβάζει και αναλύει (parse) το JSON μήνυμα στα επιμέρους πεδία του (σχόλιο 8). Λαμβάνει το πεδίο id (σχόλιο 9) και τον κώδικα που αντιστοιχεί στην Javascript (σχόλιο 10). Αφού εκτελέσει τον κώδικα (σχόλιο 11) αποστέλλει το αποτέλεσμα στον Broker (σχόλιο 12) και τυπώνει τον χρόνο που χρειάστηκε για να αναλύσει και να ολοκληρώσει το αίτημα.

Όταν λάβει μήνυμα από τον Broker για να αποστείλει χαρακτηριστικά, λαμβάνει από την συσκευή τα τρέχοντα χαρακτηριστικά και τα αποστέλλει στο κατάλληλο topic (σχόλιο 13). Για να γίνει αυτό καλεί την getStat() η οποία λαμβάνει από το σύστημα τους διαθέσιμους πυρήνες και μνήμη. Επίσης λαμβάνει την τρέχουσα στάθμη της μπαταρίας (σχόλιο 14), δημιουργώντας ένα Intent για την υπηρεσία (service) της μπαταρίας, καθώς και την καθυστέρηση στην σύνδεση με τον Broker (latency) καλώντας το πρόγραμμα ping από το λειτουργικό του android για το IP του Broker (σχόλιο 15).

4.3.4.5 Latencer

Επιτελεί την ίδια λειτουργία με την αντίστοιχη κλάση TaskHost στην κύρια εφαρμογή. Λόγω των περιορισμών στο περιβάλλον του android η υλοποίηση είναι ελαφρά διαφορετική, καθώς στην περίπτωση του android καλείται η εντολή ping και δεν χρησιμοποιείται η μέθοδος isReachable της κλάσης InetAddress.

Συνολικά η εφαρμογή του Android περιγράφεται από το παρακάτω διάγραμμα:



Διάγραμμα 4.5 - Διάγραμμα κλάσεων: Εφαρμογή Android

Κεφάλαιο 5: Πειράματα

Στα πλαίσια της ανάπτυξης της εφαρμογής κόμβου Fog εκτελέσαμε διαφορετικά πειράματα για να ελέγξουμε την λειτουργία και τις επιδόσεις της σε διαφορετικές συνθήκες.

Παρότι έχει ήδη γίνει αναφορά, αξίζει να υπενθυμίσουμε στο σημείο αυτό πως στην εφαρμογή μας λειτουργούν δύο οντότητες που ονομάζονται Broker : Ο MQTT Broker (Mosquitto) και η κλάση Broker, ο Fog Broker. Οι δύο αυτές οντότητες έχουν διαφορετικό ρόλο και λειτουργία. Ο πρώτος είναι η εφαρμογή που πραγματώνει την επικοινωνία με MQTT των συμμετεχόντων στον κόμβο. Ο δεύτερος (Fog Broker) είναι η κλάση που σχεδιάσαμε και υλοποιήσαμε και είναι υπεύθυνος για τον συντονισμό του κόμβου Fog, όπως αυτή περιγράφηκε παραπάνω.

Συγκεκριμένα έγιναν 3 πειράματα :

1. Μέτρηση του χρόνου απόκρισης του συστήματος για διαφορετικούς ρυθμούς αποστολής μηνυμάτων. Ο στόχος είναι να αποκτήσουμε αίσθηση για τις δυνατότητες και τα όρια του συστήματος που σχεδιάσαμε και υλοποιήσαμε. Επίσης, οι μετρήσεις αυτές παίζουν και τον ρόλο κατάστασης αναφοράς για τα υπόλοιπα πειράματα.
2. Σύγκριση των χρόνων απόκρισης ανάλογα με την συσκευή στην οποία εκτελούνται οι διάφορες κλάσεις και ο MQTT Broker. Επίσης, σύγκριση της λειτουργίας του κόμβου με τοπικά εγκατεστημένο MQTT Broker σε σχέση με χρήση MQTT Broker μέσω διαδικτύου.
3. Σύγκριση των χρόνων απόκρισης του δικτύου με την συμμετοχή περισσότερων από μια Task Host. Συγκεκριμένα, δοκιμάζουμε την συμπεριφορά του δικτύου σε οριακές καταστάσεις και πως αυτή βελτιώνεται με την παρουσία περισσότερων συσκευών επεξεργασίας ώστε να μοιράζεται ο φόρτος εργασίας.

Οι συσκευές που χρησιμοποιήθηκαν για τα πειράματα και τα χαρακτηριστικά τους αναφέρονται στο Παράρτημα Β. Στο παρών κεφάλαιο θα τις αναφέρουμε με τα ομόματα που αναγράφονται εκει.

5.1 Πείραμα 1ο

Για το πείραμα αυτό χρησιμοποιήσαμε την παρακάτω διάταξη στον κόμβο:

Διάγραμμα 5.1 – Διάταξη πρώτου Πειράματος

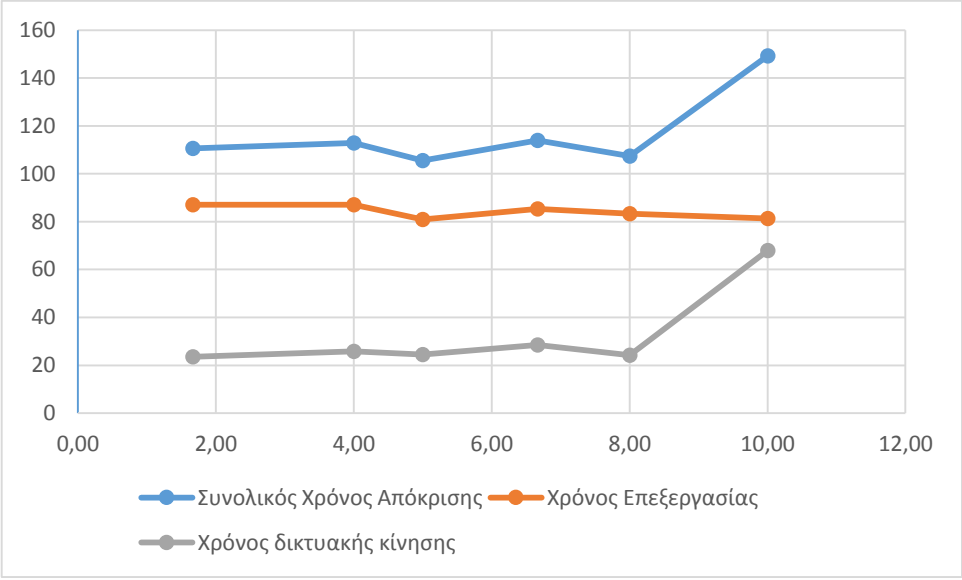
Για να προσομοιωθεί η επεξεργασία περισσότερων δεδομένων ή/και αλγορίθμου μεγαλύτερης πολυπλοκότητας, στην λειτουργία της Task Host προστέθηκε καθυστέρηση 50ms με μια εντολή sleep πριν την επιστροφή των αποτελεσμάτων. Η μέθοδος που ακολουθήσαμε είναι η εξής :

Η gateway στέλνει συνολικά 300 μηνύματα προς επεξεργασία. Ο ρυθμός αποστολής μεταβάλλεται για να ελέγξουμε την απόκριση του δικτύου. Καταγράφουμε τον συνολικό χρόνο που έκανε κάθε μήνυμα από τη στιγμή που το προωθεί η Gateway στον Broker καθώς και τον χρόνο που παραμένει στην Task Host, ως χρόνο επεξεργασίας. Αφαιρώντας από την πρώτη μέτρηση την δεύτερη βρίσκουμε το χρονικό διάστημα που αφορά δικτυακή κίνηση/καθυστέρηση και όχι επεξεργασία του μηνύματος. Υπολογίζουμε τις μέσες τιμές για τα τρία αυτά μεγέθη και έτσι αποκτούμε μια εικόνα για την λειτουργία του κόμβου. Παρακάτω παραθέτουμε τα ποσοτικά αποτελέσματα.

Περίοδος αποστολής μηνυμάτων (ms)	Συνολικός μέσος χρόνος απόκρισης (ms)	Μέσος χρόνος επεξεργασίας (ms)	Μέσος χρόνος δικτυακής κίνησης (ms)
600	110,653	87,087	23,567
250	112,893	87,033	25,86
200	105,48	80,953	24,527
150	113,926	85,406	28,52
125	107,456	83,29	24,167
100	149,254	81,321	67,93

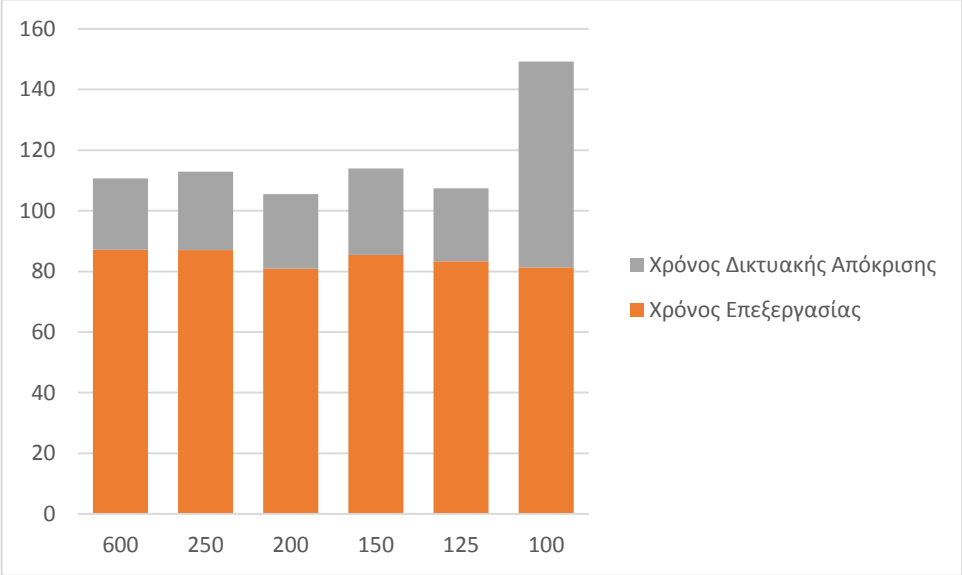
Πίνακας 5.1 – Αποτελέσματα πρώτου πειράματος

Και σε μορφή διαγράμματος χρόνου (ms) ανά ρυθμό αποστολής μηνυμάτων (Hz).



Διάγραμμα 5.2 – Διάγραμμα χρόνων πρώτου πειράματος

Καθώς και διάγραμμα κατανομής χρόνου ανά περίοδο αποστολής.



Διάγραμμα 5.3 – Διάγραμμα μερών συνολικής χρονικής απόκρισης

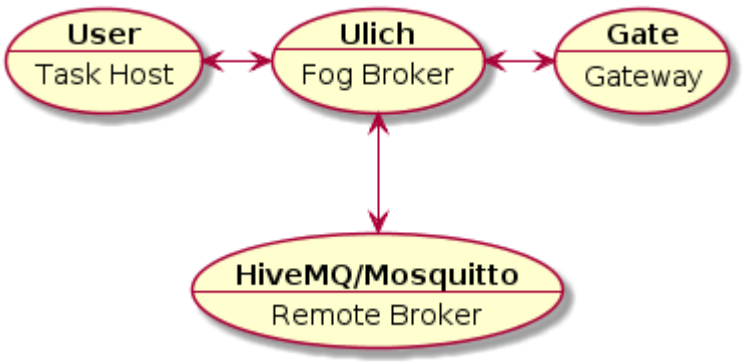
Παρατηρούμε πως ο χρόνος δικτυακής κίνησης κυμαίνεται γενικά κάτω από τα 30ms για περίοδο αποστολής μηνυμάτων μέχρι και 125 ms, μέγεθος αποδεκτό καθώς ένας μέσος χρόνος απόκρισης υπολογιστικού νέφους είναι 100 ms. Προφανώς όταν μειωθεί η περίοδος κάτω από τα 100 ms και πλησιάσει στον ελάχιστο συνολικό χρόνο απάντησης κάθε καθυστέρηση δεν γίνεται να εξισορροπηθεί και μεταφέρεται στις επόμενες μετρήσεις αφού το επόμενο μήνυμα περιμένει ήδη στην ουρά. Επίσης και για περίοδο γύρω στα 100 ms ενώ η μέση τιμή είναι σημαντικά υψηλότερη, παρατηρώντας τις πραγματικές μετρήσεις βλέπουμε πως ενώ εμφανίζονται καθυστερήσεις σε συνεχόμενα μηνύματα, που λογικά οφείλονται σε μια αρχική καθυστέρηση, το σύστημα καταφέρνει σύντομα να ισορροπήσει ξανά στους κανονικούς χρόνους απόκρισης. Αυτή η παρατήρηση ισχύει και γενικά για μεγάλους ρυθμούς αποστολής μηνυμάτων όπως είναι φυσικό, αφού αν ένα μήνυμα καθυστερήσει σημαντικά, αυτό θα επηρεάσει και τον χρόνο των επόμενων μηνυμάτων, που θα βρίσκονται στην ουρά. Τέλος κατά την εκκίνηση του συστήματος, την πρώτη φορά που ανατίθεται στην Task Host ένα αίτημα, ο χρόνος απόκρισης είναι σημαντικά μεγαλύτερος. Σε κατάσταση κανονικής λειτουργίας, το σύστημα επανέρχεται σύντομα.

5.2 Πείραμα 2ο

Για το πείραμα αυτό συγκρίνουμε πως επηρεάζει την λειτουργία του συστήματος η εκτέλεση του MQTT Broker σε διαφορετικές συσκευές του τοπικού δικτύου, καθώς και εκτός αυτού. Συγκεκριμένα εκτελέσαμε τα παρακάτω πειράματα :

- Ο MQTT Broker (Mosquitto) εκτελείται στην ίδια συσκευή με την Gateway
- Ο MQTT Broker (HiveMQ) εκτελείται σε δημόσιο MQTT server (HiveMQ Public Broker – `broker.hivemq.com`) στην Φρανκφούρτη

- Ο MQTT Broker (Mosquitto) εκτελείται σε δημόσιο MQTT server (Eclipse Public Broker – `iot.eclipse.org`) στην Οττάβα του Καναδά



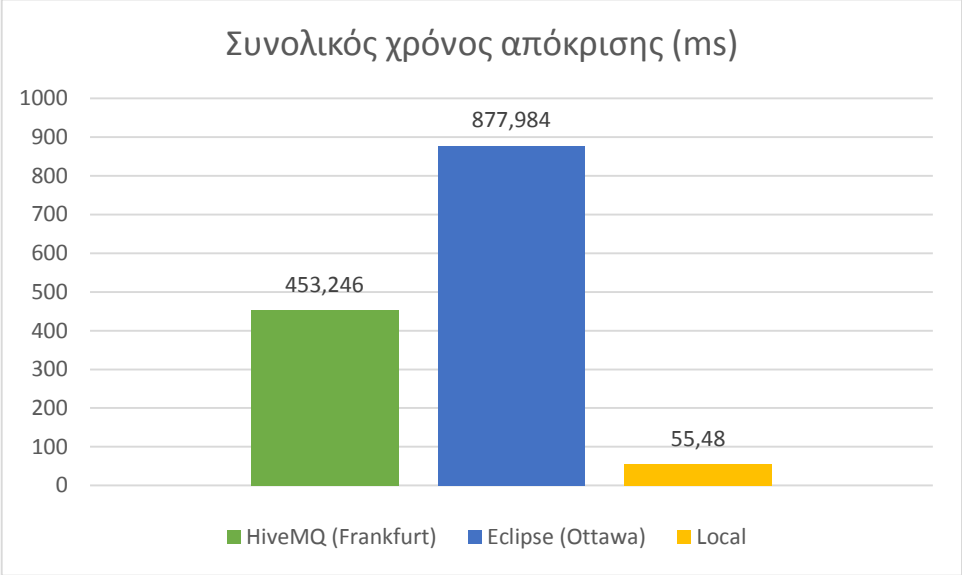
Διάγραμμα 5.4 – Διάταξη δεύτερου Πειράματος

Οι μετρήσεις έδειξαν πως η θέση του MQTT Broker στο τοπικό δίκτυο δεν επηρεάζει σημαντικά την λειτουργία του κόμβου. Δεν παίζει δηλαδή ρόλο ο MQQT Broker να εκτελείται στην ίδια συσκευή με κάποια συγκεκριμένη κλάση (Gateway, Broker). Φαίνεται όμως να έχει σημασία η ποιότητα της συσκευής στην οποία εκτελείται ο MQTT Broker καθώς ο κόμβος εμφάνισε σχετική αστάθεια όταν ο αυτός εκτελέστηκε στον φορητό υπολογιστή.

Όσον αφορά τους δημόσιους MQTT Brokers, οι επιδόσεις τους ήταν αναμενόμενα πολύ χειρότερες από αυτές του τοπικά εγκατεστημένου. Συγκεκριμένα, ο server στην Φρανκφούρτη είχε μέση χρόνο δικτυακής κίνησης 453,246 ms, με αποτέλεσμα να μην μπορεί να υποστηρίξει περίοδο αποστολής κάτω από 500 ms, ενώ αυτός στην Οττάβα ακόμα χειρότερα, χωρίς να μπορεί να υποστηρίξει περίοδο κάτω των 700 ms.

Συμπερασματικά, οι MQTT Broker που ελέγχθηκαν εκτός του τοπικά εγκατεστημένου Mosquitto δεν έχουν επιδόσεις που να τους καθιστούν χρήσιμους για κάτι παραπάνω από δοκιμές ενώ η προσέγγιση του Fog απαιτεί την ύπαρξη του MQTT Broker τοπικά.

Παρακάτω παρουσιάζονται σε διάγραμμα τα αποτελέσματα:



Διάγραμμα 5.5 – Συγκριτικό διάγραμμα δεύτερου πειράματος

5.3 Πείραμα 3ο

Για το πείραμα αυτό ελέγχουμε την λειτουργία του συστήματος με περισσότερες από μία συνδεδεμένες Task Hosts. Αυτή η περίπτωση ανταποκρίνεται σε μία κατάσταση όπου ο κόμβος Fog έχει να διαχειριστεί πολύπλοκα αιτήματα επεξεργασίας που απαιτούν από κάθε συσκευή περισσότερο χρόνο για την ολοκλήρωση των υπολογισμών. Σε αντίθετη περίπτωση, για αιτήματα που η επεξεργασία τους διαρκεί πολύ λίγο, είναι αναμενόμενο ότι σημασία έχει βασικά η δικτυακή καθυστέρηση και δεν παίζει ιδιαίτερο ρόλο η παρουσία περισσότερων συσκευών υπολογισμού. Συγκεκριμένα ακολουθούμε την παρακάτω μεθοδολογία.

Προσεγγίζουμε την περίοδο αποστολής μηνυμάτων για την οποία με μία Task Host, παρουσιάζεται αδυναμία εξυπηρέτησης των αιτημάτων. Αυτή είναι η περίοδος για την οποία το δίκτυο δεν καταφέρνει να ισορροπήσει από τις όποιες διακυμάνσεις, με αποτέλεσμα όλο και περισσότερα μηνύματα να φτάνουν στην Task Host, ενώ αυτή δεν έχει τελειώσει την επεξεργασία των προηγούμενων. Όπως είναι αναμενόμενο η περίοδος αυτή είναι περίπου ίση με τον χρόνο που χρειάζεται η Task Host να ολοκληρώσει την επεξεργασία του μηνύματος συν τις ελάχιστες δικτυακές καθυστερήσεις. Η δικτυακή διάταξη είναι η παρακάτω.

Διάγραμμα 5.6 - Διάταξη τρίτου πειραματος. Περίπτωση μίας Task Host

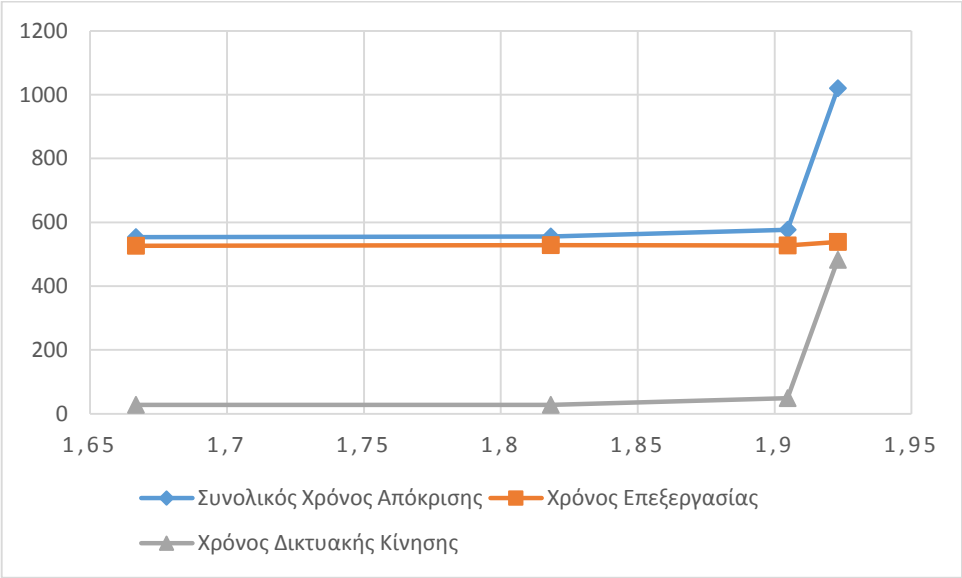
Στην περίπτωσή μας για να εξομοιώσουμε ένα υπολογιστικά απαιτητικό αίτημα αυξήσαμε τον χρόνο αναμονής των Task Host πριν επιστρέψουν την απάντηση σε 500

ms. Έτσι ο χρόνος επεξεργασίας κινούταν γύρω στα 526 ms. Όπως φαίνεται και από τις παρακάτω μετρήσεις, η περίοδος κορεσμού για την διάταξη με μία Task Host ήταν κοντά στα 525 ms. Στην περίπτωση αυτή, το σύστημα αργούσε σημαντικά να επανέλθει από μία πιθανή καθυστέρηση στην επιστροφή μιας απάντησης, λόγω δικτυακής ή επεξεργαστικής αστοχίας. Παρακάτω παρουσιάζουμε τα ως τώρα αποτελέσματα

Περίοδος αποστολής μηνυμάτων (ms)	Συνολικός μέσος χρόνος απόκρισης (ms)	Μέσος χρόνος επεξεργασίας (ms)	Μέσος χρόνος δικτυακής κίνησης (ms)
600	553,779	526,2742	27,505
550	555,793	528,314	27,479
525	576,41	527,713	48,823
520	1020,037	538,121	481,916

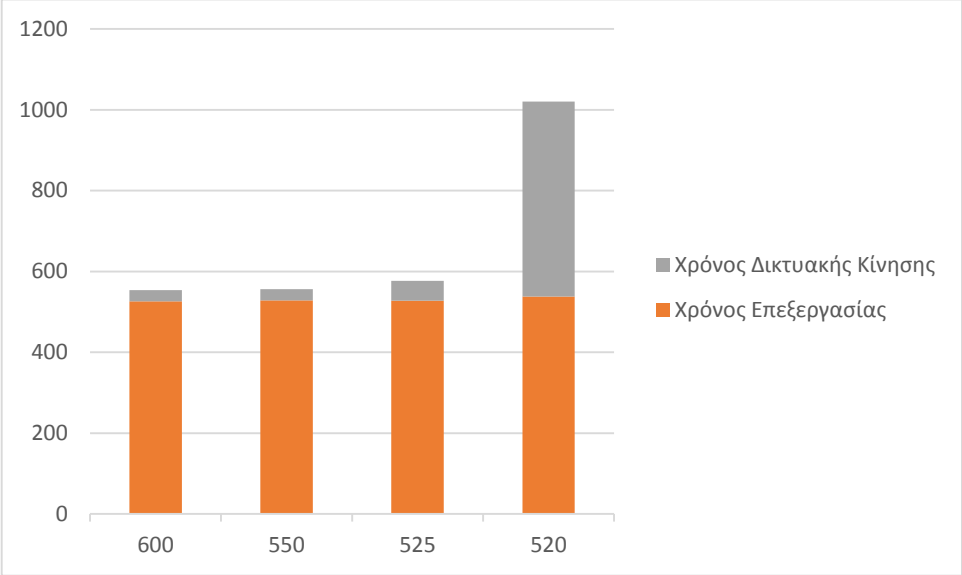
Πίνακας 5.2 – Αποτελέσματα τρίτου πειράματος. Περίπτωση μίας Task Host

Και σε μορφή διαγράμματος.



Διάγραμμα 5.7 – Διάγραμμα χρόνων τρίτου πειράματος. Περίπτωση μίας Task Host

Καθώς και διάγραμμα κατανομής χρόνου ανά περίοδο αποστολής.



Διάγραμμα 5.8 – Διάγραμμα μερών συνολικής χρονικής απόκρισης

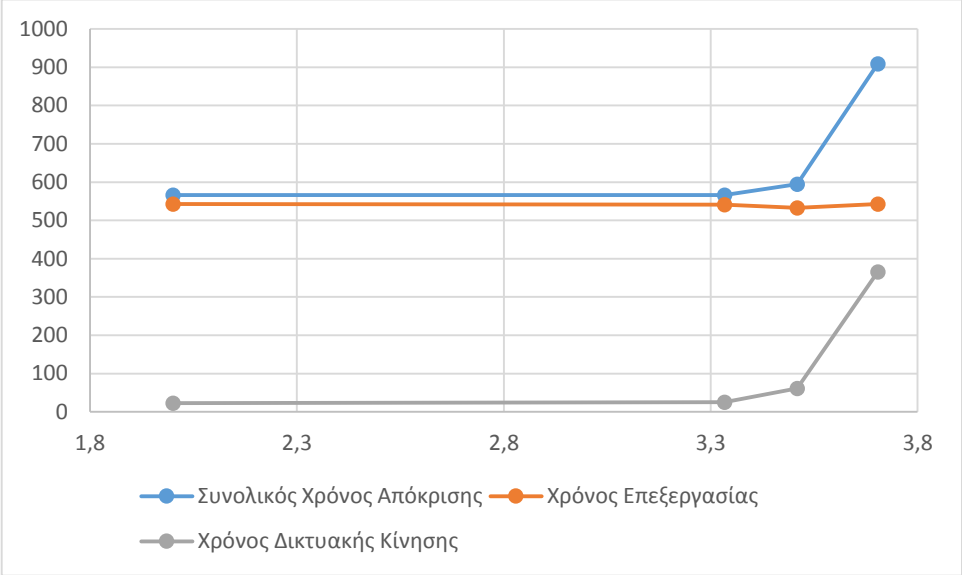
Διάγραμμα 5.9 - Διάταξη τρίτου πειραματος. Περίπτωση δύο Task Host

Στην συνέχεια προσθέσαμε στον κόμβο άλλη μία Task Host όπως φαίνεται στο διάγραμμα 5.8 της διάταξης, τις οποίας οι επιδόσεις είναι ελαφρώς χειρότερες από την πρώτη. Για να εξασφαλίσουμε ότι και οι δύο Task Hosts θα αναλαμβάνουν να επεξεργαστούν αιτήματα, αυξήσαμε το βάρος του Αριθμού Εργασιών στην βαθμολόγηση. Για διπλασιασμό του αριθμού των Task Hosts αναμένουμε η περίοδος κορεσμού να είναι χαμηλότερη και κοντά στο μισό της προηγούμενης. Αυτό συμβαίνει διότι για δύο πανομοιότυπες συσκευές με μισή περίοδο, κάθε μία θα είχε περίπου το ίδιο χρονικό παράθυρο με την περίπτωση μίας συσκευής σε ολόκληρη περίοδο, αν τα αιτήματα ανατίθενται εναλλάξ. Πράγματι τα πειραματικά δεδομένα επιβεβαιώνουν τον ισχυρισμό μας.

Περίοδος αποστολής μηνυμάτων (ms)	Συνολικός μέσος χρόνος απόκρισης (ms)	Μέσος χρόνος επεξεργασίας (ms)	Μέσος χρόνος δικτυακής κίνησης (ms)
500	566,227	543,107	23,12
300	566,45	540,764	25,686
285	594,477	533,07	61,407
270	908,583	542,87	365,713

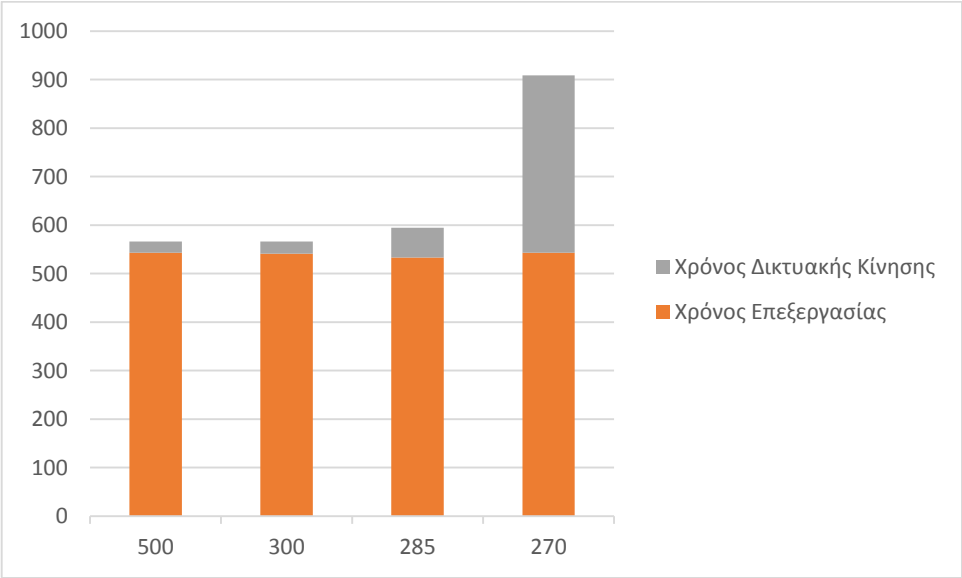
Πίνακας 5.3 – Αποτελέσματα τρίτου πειράματος. Περίπτωση δύο Task Host

Και σε μορφή διαγράμματος



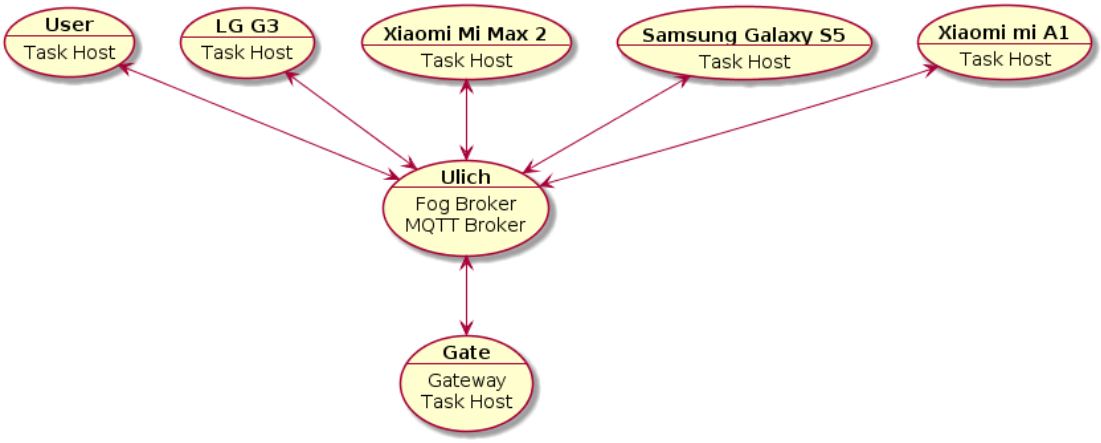
Διάγραμμα 5.10– Διάγραμμα χρόνων τρίτου πειράματος. Περίπτωση δύο Task Host

Καθώς και το διάγραμμα κατανομής χρόνου ανά περίοδο αποστολής.



Διάγραμμα 5.11 – Διάγραμμα μερών συνολικής χρονικής απόκρισης

Τέλος, δοκιμάσαμε την λειτουργία του συστήματος με έξι συνδεδεμένες Task Host, όπως φαίνεται στο διάγραμμα 5.11.



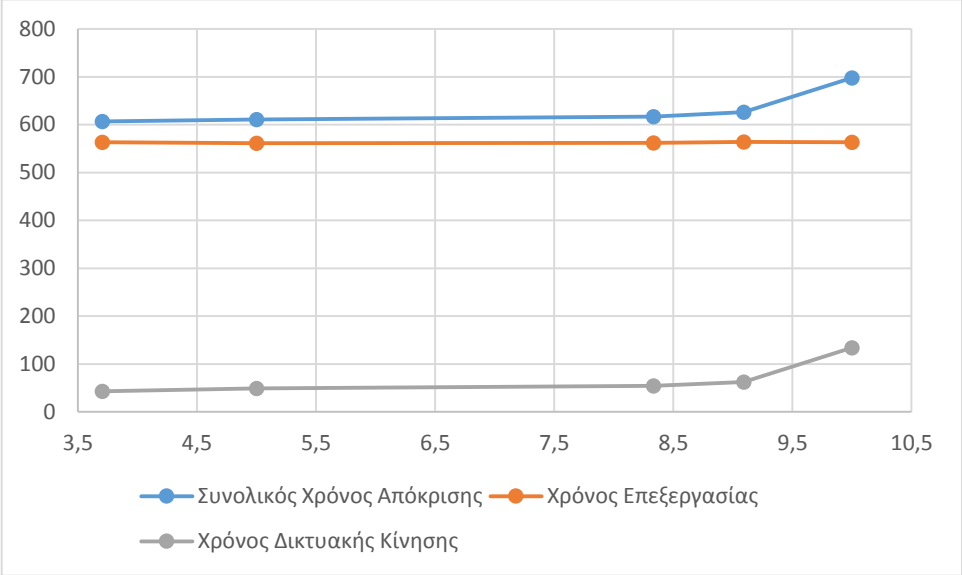
Διάγραμμα 5.12 - Διάταξη τρίτου πειραματος. Περίπτωση δύο Task Host

Τα αποτελέσματα του πειράματος έχουν διπλή σημασία, πέρα από να παραμείνει σε αποδεκτά επίπεδα ο χρόνος απόκριση για μικρότερες περιόδους αποστολής μηνυμάτων, το συγκεκριμένο πείραμα ελέγχει και την ευστάθεια του συστήματος για έναν σεβαστό αριθμό Task Hosts καθώς και όταν σε αρκετές Task Hosts εκτελείται η εφαρμογή Android. Το σύστημα ανταποκρίθηκε αποτελεσματικά, καθώς λειτούργησε ικανοποιητικά αρκετά κοντά στο θεωρητικό σημείο κορεσμού του. Ένα χρήσιμο συμπέρασμα είναι ότι όσο αυξάνεται ο αριθμός των Task Hosts, η συμπεριφορά κοντά στο σημείο κορεσμού είναι ομαλότερη, καθώς μία καθυστέρηση σε μια επιμέρους Task Host έχει περισσότερο περιθώριο να απορροφηθεί από το σύστημα, αφού η πιθανότητα να αστοχήσουν ταυτόχρονα όλες οι συσκευές είναι μικρότερη. Παρακάτω παρουσιάζονται τα αποτελέσματα. Να σημειώσουμε ότι οι τέσσερεις παραπάνω συσκευές από το προηγούμενο πείραμα ήταν όλες κινητά τηλέφωνα στις οποίες έτρεχε η εφαρμογή Android. Οι συσκευές αυτές βαθμολογήθηκαν υψηλά με αποτέλεσμα να αναλάβουν σημαντικό κομμάτι της επεξεργασίας, ανά περίπτωση μεγαλύτερο και του ενός έκτου των μηνυμάτων. Αυτό αποτυπώνεται στα αποτελέσματα καθώς παρότι η θεωρητική βαθμολογία των συσκευών είναι ανάλογη με των υπολογιστών, στην πράξη οι επεξεργαστικές και κυρίως δικτυακές του επιδόσεις ήταν με συνέπεια λίγο χειρότερες.

Περίοδος αποστολής μηνυμάτων (ms)	Συνολικός μέσος χρόνος απόκρισης (ms)	Μέσος χρόνος επεξεργασίας (ms)	Μέσος χρόνος δικτυακής κίνησης (ms)
270	606,595	563,429	43,166
200	610,591	561,489	49,102
120	616,503	562,217	54,287
110	626,42	564,265	62,155
100	697,602	563,321	134,281

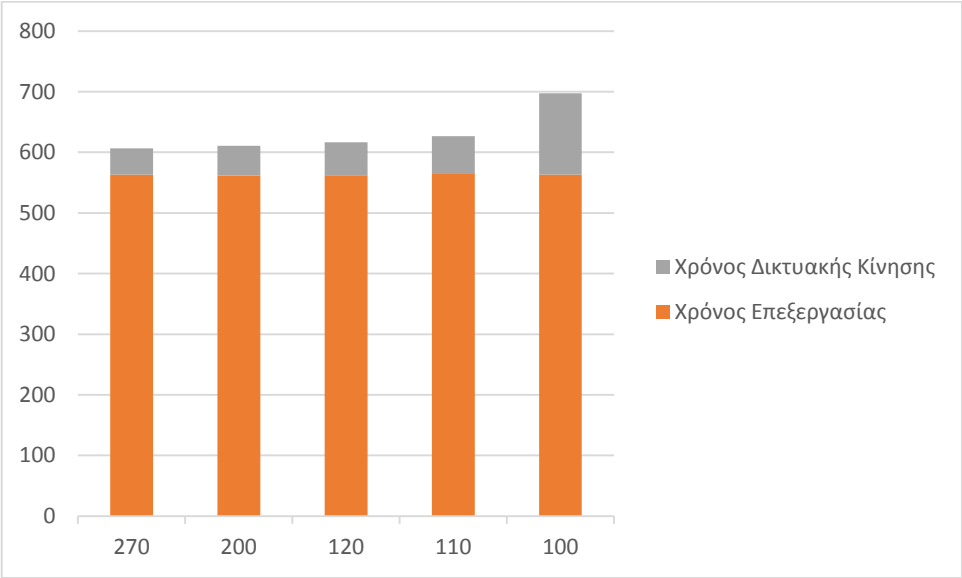
Πίνακας 5.4 – Αποτελέσματα τρίτου πειράματος. Περίπτωση έξι Task Host

Και σε μορφή διαγράμματος



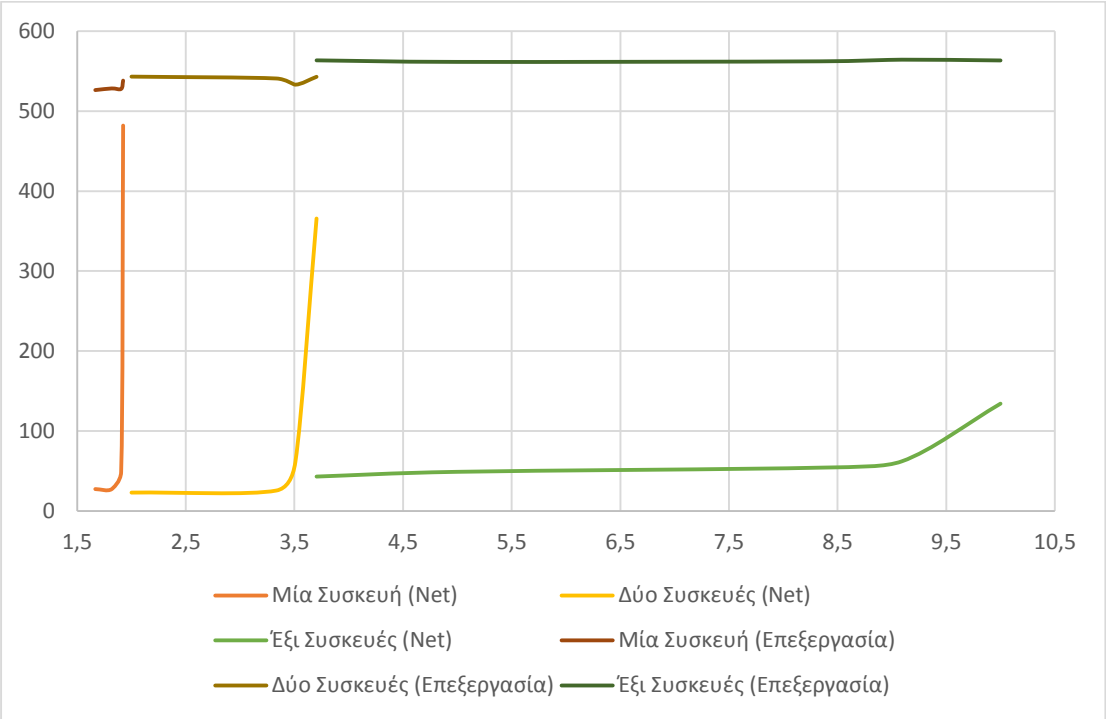
Διάγραμμα 5.13– Διάγραμμα χρόνων τρίτου πειράματος. Περίπτωση έξι Task Host

Καθώς και το διάγραμμα **κατανομής χρόνου ανά περίοδο αποστολής.**



Διάγραμμα 5.14 – Διάγραμμα μερών συνολικής χρονικής απόκρισης

Στο παρακάτω διάγραμμα εμφανίζονται συγκεντρωτικά τα αποτελέσματα για το τρίτο πείραμα.



Διάγραμμα 5.15 – Συγκεντρωτικό διάγραμμα τρίτου πειράματος

Κεφάλαιο 6: Επίλογος

Κλείνοντας αυτή την εργασία θα προσπαθήσουμε να ανακεφαλαιώσουμε και να συνοψίσουμε τα αποτελέσματά της. Σκοπός είναι να εξάγουμε συγκεκριμένα συμπεράσματα ως προς του στόχους που τέθηκαν στην εισαγωγή και να σημειώσουμε συγκεκριμένες μελλοντικές προτάσεις για έρευνα που προέκυψαν στην διάρκεια της συγγραφής της παρούσας εργασίας. Καθώς και να θέσουμε κάποιους πιο γενικούς προβληματισμούς που προέκυψαν από την ενασχόλησή μας με το Fog και τις συναφείς τενχολογίες.

6.1 Συμπεράσματα

Στο εισαγωγικό κεφάλαιο θέσαμε ορισμένους στόχους ως προς την κατεύθυνση της παρούσας εργασίας.

Ολοκληρώνοντάς την, βλέπουμε ότι το πρόβλημα της διαχείρισης των εκθετικά αυξανόμενων δεδομένων απασχολεί πολλούς ερευνητές του κλάδου. Σαν λύση εμφανίζονται τόσο το Fog, όσο και άλλες αρχιτεκτονικές όπως αυτή του edge computing κλπ. Στην παρούσα εργασία καταφέραμε να περιγράψουμε συνολικά το πρόβλημα της επεξεργασίας των δεδομένων, σήμερα και μελλοντικά, καθώς και να εξηγήσουμε τα προτερήματα του Fog που το καθιστούν πραγματική λύση για το πρόβλημα αυτό. Για να επιτευχθεί αυτό κάναμε μια εκτενή παρουσίαση του Fog και των αρχών σχεδίασής του.

Βασιζόμενοι στην παραπάνω έρευνα καταφέραμε να εναρμονίσουμε την εργασία [7] με τις γενικές αρχές του Fog, όπως αυτές περιγράφονται στο Fog Consortium. Ορίσαμε τον κόμβο και τα στοιχεία που τον απαρτίζουν με τρόπο που περιγράφει την επικοινωνία με άλλους κόμβους, το Cloud και τις συσκευές IoT, ώστε αυτός να μπορεί να συμμορφώνεται με την λογική του N-tier Fog και να αποτελέσει κόμβο σε κάθε επίπεδο. Ταυτόχρονα επεκτείναμε την Συνάρτηση Βαθμολόγησης ώστε να παίρνει υπόψιν της τα μεγέθη της θέσης και της αξιοπιστίας, μεγέθη που σύμφωνα με την περιγραφή του Fog έχουν ιδιαίτερη σημασία. Προσθέσαμε επίσης αναγκαίες λειτουργικότητες στους ρόλους που περιγράφονται, που προέκυψαν μέσα από την διαδικασία υλοποίησης του πρωτοτύπου-προσομοιωτή, όπως αυτές περιγράφονται από τα σενάρια στο κεφάλαιο 4.

Με την ολοκλήρωση του πρωτοτύπου-προσομοιωτή έχουμε στα χέρια μας ένα λειτουργικό σύστημα που μπορεί να εγκατασταθεί και να εκτελεστεί σε μια σειρά από συσκευές, αρκεί αυτές να εκτελούν Java. Με αυτό τον τρόπο πετύχαμε τον στόχο μας να δημιουργήσουμε έναν προσομοιωτή στον οποίο μπορεί κάποιος να πάρει συγκεκριμένες μετρήσεις και να βγάλει συμπεράσματα για το Fog συνολικά. Δεν παραμείναμε όμως εκεί. Υλοποιήσαμε την εφαρμογή στο Android, εμπλουτίζοντας το εύρος των διαφορετικών συσκευών στις οποίες μπορεί να εκτελεστεί η εφαρμογή, με τις συσκευές που βασίζονται στο πολύ δημοφιλές λειτουργικό. Τέτοιες συσκευές θα απαρτίζουν μεγάλο κομμάτι των συσκευών επεξεργασίας σε έναν κόμβο Fog, όπως φάνηκε από την βιβλιογραφική έρευνα.

Τέλος ελέγξαμε την συμπεριφορά της εφαρμογής μας σε ποικιλία συνθηκών μέσω πειραμάτων, στο κεφάλαιο 5. Σαν γενική αποτίμηση αξίζει να αναδειχθεί ότι η επεξεργασία στον κόμβο Fog είχε σημαντικά μικρότερες δικτυακές καθυστερήσεις από το Cloud, ενώ ο κόμβος μπορεί να διαχειριστεί και απαιτητική επεξεργασία, όταν σε αυτόν συμμετέχουν αρκετές Task Hosts. Παράλληλα φαίνεται πως ο MQTT Broker είναι με διαφορά το πιο απαιτητικό μέρος από πλευράς πόρων στην λειτουργία ενός κόμβου Fog αποκλείοντας πιο αδύναμες συσκευές από το να επιτελούν αυτόν τον ρόλο. Τέλος η εκτέλεση πειραμάτων με συσκευές λογισμικού Android σε ρόλο Task Host αναδεικνύουν ότι μία μη αποδοτική συνάρτηση βαθμολόγησης επηρεάζει σημαντικά την συνολική συμπεριφορά του κόμβου και είναι αναγκαία η εξειδικευμένη μελέτη για τον προσδιορισμό των κατάλληλων βαρών.

6.2 Μελλοντική Έρευνα

Στην διάρκεια εκπόνησης της παρούσας εργασίας αναδείχθηκε πληθώρα πεδίων στα οποία θα μπορούσε κανείς να εμβαθύνει, που άπτονται της παρούσας εργασίας αλλά ξεφεύγουν από τον σκοπό και την κλίμακά της. Παρακάτω παρουσιάζουμε συγκεκριμένα και ενδιαφέροντα, κατά την γνώμη μας, θέματα για περεταίρω έρευνα και εμβάθυνση.

Όσον αφορά την συνολικότερη λογική του Fog Networking, to Open Fog Consortium περιγράφει κάποιες γενικές κατευθύνσεις για την εξέλιξη του Fog καθώς και γενικές σχεδιαστικές αρχές, που είναι ζήτημα της ερευνητικής δουλειάς στο πεδίο τα επόμενα χρόνια του πως θα συγκεκριμενοποιηθούν. Ένα πρώτο έλλειμα είναι η απουσία γενικών κανόνων, πρωτοκόλλων και προτύπων γύρω από την σχεδίαση και υλοποίηση δικτυώσεων Fog που είναι αναγκαίες για την ευρύτερη εφαρμογή οποιουδήποτε μοντέλου.

Ένα συγκεκριμένο ειδικό έλλειμα, που προέκυψε και μέσα από την δουλειά μας, είναι ποια ακριβώς θα είναι η μορφή των μηνυμάτων στα οποία θα ενσωματώνονται τα δεδομένα που συλλέγονται από χαμηλότερα επίπεδα και οι μέθοδοι επεξεργασίας αυτών. Καθώς και ποιος θα είναι ο τρόπος επικοινωνίας με τα χαμηλά επίπεδα, ιδιαίτερα με το στρώμα αισθητήρων και ενεργοποιητών που βρίσκονται στα άκρα του δικτύου. Το ζήτημα αυτό απαιτεί την συγκεκριμένη έρευνα σε πραγματικές συσκευές και την συνεργασία των κατασκευαστών των συσκευών αυτών. Το συγκεκριμένο ζήτημα δεν περιορίζεται μόνο στην τυπική μορφή που έχει η επικοινωνία με τις συσκευές αυτές, αλλά μπορεί να επηρεάζει και θέματα σχεδίασης και ευρύτερης αρχιτεκτονικής ενός κόμβου και γενικά του Fog. Για παράδειγμα, αισθητήρες με περιορισμένες δυνατότητες σύνδεσης θα απαιτούν αντίστοιχη διεπαφή όπως αυτή υλοποιείται από την Gateway, είτε όσον αφορά συγκεκριμένες προδιαγραφές του υλικού που έχουν να κάνουν με την επικοινωνία, είτε ειδική διαχείριση ώστε να επιτυγχάνεται ομοιομορφία των συλλεγόμενων δεδομένων. Τέλος στην επικοινωνία των συσκευών με την Gateway πιθανά υπισέρχονται ζητήματα πατέντας, που απαιτούν ειδική μεταχείριση ή και ειδικό υλικό.

Ζήτημα με μεγάλη σημασία που απασχολεί έντονα και την διεθνή επιστημονική κοινότητα είναι η ασφάλεια. Σε δίκτυα Fog μέρος της επεξεργασίας εξαρτάται από την συμμετοχή έξυπνων συσκευών καθημερινής χρήσης. Τα σημερινά επίπεδα ασφάλειας σε μεγάλο αριθμό συσκευών IoT είναι εξαιρετικά χαμηλά [28]. Για να έχει λοιπόν κίνητρο ένας ιδιώτης να συμμετέχει με τις συσκευές του στον κόμβο Fog αλλά και να τηρείται η ιδιωτικότητα και η ασφάλεια των δεδομένων είναι αναγκαία μία συγκεκριμένη αναβάθμιση των επιπέδων ασφαλείας και επέκταση των ήδη υπαρχόντων συστημάτων.

Στην εφαρμογή που υλοποιήσαμε πάρθηκαν αναγκαστικά κάποιες επιλογές σχετικά με τεχνικά ζητήματα όπως τα πρωτόκολλα επικοινωνίας, οι αλγόριθμοι που υποστηρίζονται και οι γλώσσες οι οποίες χρησιμοποιήθηκαν. Παρότι οι συγκεκριμένες επιλογές θεμελιώνονται με την σημερινή αντίληψη και κατανόησή μας για το Fog, απαιτείται περεταίρω εμβάθυνση. Για παράδειγμα θα ήταν χρήσιμη μια συγκριτική μελέτη διαφορετικών πρωτοκόλλων επικοινωνίας στον κόμβο Fog, αντί του MQTT.

Στην υλοποίησή μας η κατανομή των αιτημάτων στις Task Hosts γίνεται βάση της συνάρτησης βαθμολόγησης. Η συνάρτηση αυτή προϋποθέτει ότι τα χαρακτηριστικά να σταθμίζονται ανάλογα με τις ανάγκες μιας συγκεκριμένης εφαρμογής ή η ίδια η συνάρτηση να εμπλουτίζεται με άλλα και να απαλείφονται χαρακτηριστικά που πιθανά δεν είναι χρήσιμα ανά περίπτωση. Μια τέτοια μελέτη μπορεί να είναι ιδιαίτερα ενδιαφέρουσα και να αξιοποιήσει μεθόδους μηχανικής μάθησης για την έυρεση κατάλληλων βαρών. Κάτι τέτοιο προϋποθέτει εξειδίκευση και συγκεκριμένα δεδομένα εκπαίδευσης του συστήματος για κάθε περίπτωση χρήσης. Η συνάρτηση βαθμολόγησης που υλοποιήσαμε στην παρούσα διπλωματική μπορεί να ειδωθεί ως η αρχική κατάσταση η οποία θα προσαρμοστεί στις ανάγκες του κάθε συστήματος. Η εφαρμογή που αναπτύξαμε μπορεί να λειτουργήσει σαν προσομοιωτής και βοηθητικό εργαλείο στην παραγωγή των αναγκαίων δεδομένων για την μηχανική μάθηση. Τέλος ενδιαφέρον παρουσιάζει η αναζήτηση εναλλακτικών λογικών άλλων από την χρήση μιας συνάρτησης βαθμολόγησης ως μέθοδο κατανομής των αιτημάτων.

Τέλος μεγάλη σημασία για εμάς έχει και η κοινωνική διάσταση όλων των παραπάνω τεχνικών και τεχνολογικών εξελίξεων. Η ομιχλώδης επεξεργασία έχει σαν στόχο την αποκέντρωση μέρους ή και ολόκληρης της διαδικασίας διαχείρισης και επεξεργασίας των δεδομένων. Με αυτόν τον τρόπο τα δεδομένα που παράγονται για παράδειγμα σε μια γειτονιά, διακινούνται και επεξεργάζονται, πρώτα από όλα, στις συσκευές των κατοίκων της γειτονιάς, σε ένα μοντέλο έξυπνης διαχείρισης μιας πόλης. Αυτό όμως δεν αντιστοιχεί και σε αλλαγή στην εποπτεία των κατοίκων της γειτονιάς, αφού η όλη διαδικασία συμβαίνει χωρίς αυτοί να μπορούν να την παρακολουθήσουν και να την επηρεάσουν. Η επεξεργασία στα άκρα δίνει την δυνατότητα να παίξουν υπό συνθήκες πιο ενεργητικό ρόλο και να έχουν μεγαλύτερη εποπτεία στην διαδικασία διαχείρισης των δεδομένων που παράγουν.

Παράλληλα ο τρόπος δικτύωσης που προτείνεται από την ομιχλώδη επεξεργασία ευνοεί την οριζόντια επικοινωνία μεταξύ των συμμετεχόντων στους κόμβους μέσω μιας εναλλακτικής και πιθανά γρηγορότερης οδού, από τις παραδοσιακές. Οι χρήστες θα μπορούν να επικοινωνούν άμεσα σε τοπικό επίπεδο ή με την μεταφορά από κόμβο σε κόμβο χωρίς να υπάρχει ανάγκη για την ύπαρξη κεντρικών διαύλων. Ένα τέτοιο μοντέλο θα μπορούσε να αποδυναμώσει το μονοπώλιο του ελέγχου και της εποπτείας από τους μεγάλους τηλεπικοινωνιακούς παρόχους, που αποτελούν και τους κεντρικούς κόμβους στην μέχρι τώρα διάρθρωση του διαδικτύου. Έχει λοιπόν αξία η έρευνα πάνω σε άλλα πρωτόκολλα δρομολόγησης και αρχιτεκτονικές δικτύων, όπως αυτό του Mesh Networking.

Όσο οι τεχνολογικές τομές και οι εξελίξεις όπως οι έξυπνες συσκευές και συστήματα, το Διαδίκτυο των Πραγμάτων κλπ διαπλέκονται εντονότερα και με πιο πολύπλοκους τρόπους με την κοινωνική πραγματικότητα και την καθημερινή ζωή, τόσο ένας μηχανικός δεν γίνεται να σχεδιάζει και να υλοποιεί εν κενώ. Αλλά θα πρέπει η διαδικασία σχεδίασης και υλοποίησης να λαμβάνει συνεχώς υπόψιν της και να αλληλοεπιδρά οργανικά με την κοινωνική πραγματικότητα και πως αυτή επηρεάζεται και καθορίζεται από τις τεχνικές και επιστημονικές εξελίξεις.

Παραρτημα Α: Κώδικας εφαρμογής

A.1 Broker package

A.1.1 Broker package

```
1.  package broker;
2.
3.  import java.util.Hashtable;
4.  import java.util.Iterator;
5.  import java.util.Random;
6.  import java.util.Set;
7.  import java.util.TreeSet;
8.  import java.util.concurrent.Executors;
9.  import java.util.concurrent.ScheduledExecutorService;
10. import java.util.concurrent.TimeUnit;
11.
12. import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
13. import org.eclipse.paho.client.mqttv3.MqttCallback;
14. import org.eclipse.paho.client.mqttv3.MqttClient;
15. import org.eclipse.paho.client.mqttv3.MqttException;
16. import org.eclipse.paho.client.mqttv3.MqttMessage;
17. import org.eclipse.paho.client.mqttv3.MqttTopic;
18.
19. import org.json.simple.*;
20. import org.json.simple.parser.JSONParser;
21.
22. public class Broker implements MqttCallback {
23.
24.     MqttClient client;
25.     static private Hashtable<String, Attributes> statsMap =
26.                                     new Hashtable<String, Attributes>();
27.     static private Hashtable<String, Integer> trustMap =
28.                                     new Hashtable<String, Integer>();
29.     static private TreeSet<Coo> scoreSet = new TreeSet<Coo>();
30.     static private TreeSet<Coo> scoreSetLatency = new TreeSet<Coo>();
31.     static private Hashtable<String, Hashtable<String, MqttMessage>> ledger
                    = new Hashtable<String, Hashtable<String, MqttMessage>>();
32.
33.     static private double[] average = new double[7];
34.     static private double[] sigma = new double[7];
35.     static private int n = 0;
36.
37.     private double latitude = 37.950076;
38.     private double longitude = 23.711669;
39.
40.     static private String BrokerIp;
41.
42.     static final Object lock = new Object();
43.
44.     public static synchronized boolean containedStatsMap(String name){
45.         return statsMap.containsKey(name);
46.     }
47.
48.     public static synchronized void putStatsMap(String name,
49.                                                 Attributes stats){
50.         statsMap.put(name, stats);
51.     }
52.
53.     public static synchronized void setScoreSet(TreeSet<Coo> tree){
54.         scoreSet.clear();
55.         scoreSet.addAll(tree);
56.     }
```

```
57.
58.     public static synchronized void setScoreSetLatency(TreeSet<Coo> tree){
59.         scoreSetLatency.clear();
60.         scoreSetLatency.addAll(tree);
61.     }
62.
63.     public static void setAverage(double[] avg){
64.         for (int i=0; i<7; i++) {
65.             average[i] = avg[i];
66.         }
67.     }
68.
69.     public static void setSigma(double[] sgm){
70.         for (int i=0; i<7; i++) {
71.             sigma[i] = sgm[i];
72.         }
73.     }
74.
75.     public static void setN(int number){
76.         n = number;
77.     }
78.
79.     public static synchronized int getNoTasks(String key) {
80.         if (ledger.containsKey(key)) {
81.             return 1 + ledger.get(key).size();
82.         }
83.         else {
84.             return 1;
85.         }
86.     }
87.
88.     public static synchronized int getTrust(String key) {
89.         if (!trustMap.containsKey(key)) {
90.             Random rand = new Random();
91.             trustMap.put(key, rand.nextInt(100));
92.             System.out.println(trustMap.get(key));
93.         }
94.         return trustMap.get(key);
95.     }
96.
97.     public static synchronized Attributes getStats(String key){
98.         return statsMap.get(key);
99.     }
100.
101.         public static synchronized void removeStats(String key) {
102.             statsMap.remove(key);
103.         }
104.
105.         public static synchronized Hashtable<String, Attributes>
106.                                                         getStatsMap(){
107.             return statsMap;
108.         }
109.
110.         public static double[] getAverage(){
111.             return average;
112.         }
113.
114.         public static double[] getSigma(){
115.             return sigma;
116.         }
117.
118.         public static int getN(){
119.             return n;
120.         }
121.
122.         private void putInLedger(String name, String id,
```

```
                                                        MqttMessage message) {
            Hashtable<String, MqttMessage> tmpTasks;

            if (ledger.containsKey(name)) {
                tmpTasks = ledger.get(name);
            }
            else {
                tmpTasks = new Hashtable<String, MqttMessage>();
            }
            tmpTasks.put(id, message);

            ledger.put(name, tmpTasks);
        }

        public void callForStats() {
            MqttMessage msg = new MqttMessage();
            msg.setPayload("Call for stats".getBytes());
            msg.setRetained(false);
            msg.setQos(0);

            try {
                client.publish("stats/call", msg);
            } catch (MqttException e) {
                e.printStackTrace();
            }

        }

        public static void main(String[] args) {
            BrokerIp = args[0];
            new Broker().connect();
        }

        public void connect() {
            try {

                //1 Connect to MQTT Broker
                client = new MqttClient("tcp://"+BrokerIp+":1883",
                                                              "broker");
                client.connect();
                //2 Set Callback
                client.setCallback(this);
                //3 Topics to Subscribe
                client.subscribe("gateway/toExec");
                client.subscribe("taskHost/results");
                client.subscribe("stats/#");
                client.subscribe("orfeas/sagapo");
                client.subscribe("orfeas/kaigo");

                MqttMessage msg = new MqttMessage();
                msg.setPayload((latitude +"~" + longitude).getBytes());

                try {
                    client.publish("stats/broker", msg);
                } catch (MqttException e) {
                    e.printStackTrace();
                }


                //4 Start Updater and Scorer and run them periodically
                ScheduledExecutorService updaterExecutor =
                          Executors.newSingleThreadScheduledExecutor();
                updaterExecutor.scheduleAtFixedRate(new Updater(this),
                                                    0, 10, TimeUnit.SECONDS);

                ScheduledExecutorService scorerExecutor =
```

```
                                Executors.newSingleThreadScheduledExecutor();
                    scorerExecutor.scheduleAtFixedRate(new Scorer(), 1, 20,
                                                        TimeUnit.MILLISECONDS);

                } catch (MqttException e) {
                    e.printStackTrace();
                }
            }

            @Override
            public void connectionLost(Throwable cause) {
                System.out.println("Connection Lost!!!");
            }

            //5 Implementation of messageArrived() method
            @Override
            public void messageArrived(String topic, MqttMessage message)
                                                                throws Exception {

                if (topic.equals("gateway/toExec")) {
                    //6 Case of incoming message from Gateway

                    //If you want to score before choosing host:
                    /*  Thread t2 = new Scorer();
                        t2.start();
                        t2.join();
                    */

                    //7 Parsing JSON message
                    String jsonMsg = new String(message.getPayload());

                    JSONObject obj = new JSONObject();
                    JSONParser parser = new JSONParser();
                    obj = (JSONObject) parser.parse(jsonMsg);
                    //8 Read message id
                    String id = (String) obj.get("id");

                    //9 Read metadata
                    JSONArray metadata = new JSONArray();
                    metadata = (JSONArray) obj.get("metadata");
                    String flag = new String();


                    for (int i=0; i <metadata.size();i++ ) {
                        System.out.println(metadata.get(i));
                        JSONObject json = (JSONObject) metadata.get(i);
                        System.out.println(json);
                        //10 Read latency_sensitive field
                        if (json.containsKey("latency_sensitive")) {
                            flag = (String) json.get("latency_sensitive");
                            break;
                        }
                    }


                    String bestName = new String();
                    //11 Select Task Host based on score depending on
                                                        latency_sensitive field
                    if (flag.equals("yes")) {
                        bestName = scoreSetLatency.first().name;
                    }
                    else {
                        bestName = scoreSet.first().name;
                    }

                    //12 Publish message to selected Task Host
```

```
                    MqttTopic pubTopic = client.getTopic("taskHost/" +
                                                                    bestName);
                    pubTopic.publish(message);

                    //13 Log message - Task Host pair to ledger
                    putInLedger(bestName, id, message);
                }
                if (topic.contains("taskHost/results")) {
                    //14 Case of returning results

                    //15 Parse message to get message id and Task Host name
                    String payload = message.toString();
                    String[] subl =payload.split("~");

                    Hashtable<String, MqttMessage> tmpTasks;

                    //16 Remove message - Task Host pair from ledger
                    if (ledger.containsKey(subl[0])) {
                        tmpTasks = ledger.get(subl[0]);
                        if (tmpTasks.containsKey(subl[1])) {
                            tmpTasks.remove(subl[1]);
                            if (tmpTasks.isEmpty()) {
       //17 if Task Host has no other pending tasks, remove Task Host from
                                                                    the ledger
                                ledger.remove(subl[0]);
                            }
                            else {
                                ledger.put(subl[0], tmpTasks);
                            }

                          //18 Forward results and message id to the Gateway

                            message.setPayload((subl[0]+"~"+subl[1]+"~"+
                                                    subl[2]).getBytes());
                            message.setRetained(false);

                            client.publish("gateway/results", message);
                        }
                        else {
                            System.out.println("Message not on Ledger,
                              maybe Task Host disconnected previously");
                        }
                    }
                    else {
                        System.out.println("Task Host not on Ledger,
                                maybe Task Host disconnected previously");
                    }
                }

                if (topic.contains("stats") && !topic.contains("call")
                                            && !topic.contains("broker")) {
                    //19 Case of stat updates

                    //20 Call the Stater
                    Thread t1 = new Stater(topic, message);

                    t1.start();
                    System.out.println("Started thread");
                    t1.join();


                    if ((message.toString()).startsWith("offline")) {

                        //21 If Task Host goes offline, remove it from the
                                                Score Set by scoring again
                        Thread t2 = new Scorer();
```

```
                        t2.start();
                        t2.join();

                        String hostName = topic.substring(6);

                        if (ledger.containsKey(hostName)) {
                            Hashtable<String, MqttMessage> tmpTasks;
                            tmpTasks = ledger.get(hostName);
                            System.out.println(hostName);



                            Set<String> keys = tmpTasks.keySet();
                            Iterator<String> itr = keys.iterator();

                            String key;


                            //22 If Task Host had pending tasks, forward
                                                        them to new Task Host
                            while (itr.hasNext()) {
                                key = itr.next();


                                String bestName = scoreSet.first().name;
                                MqttMessage tmpMsg = new MqttMessage();
                                tmpMsg = tmpTasks.get(key);

                                MqttTopic pubTopic = client.getTopic(
                                                        "taskHost/" + bestName);
                                pubTopic.publish(tmpMsg);

                                putInLedger(bestName, key, tmpMsg);

                            }
                            //23 Remove offline Task Host from the ledger.
                            ledger.get(hostName).clear();
                            ledger.remove(hostName);
                        }
                    }

                }
            @Override
            public void deliveryComplete(IMqttDeliveryToken token) {

            }

        }
```

A.1.2 Stater

```
1.  package broker;
2.
3.  import java.lang.Math;
4.  import org.eclipse.paho.client.mqttv3.MqttMessage;
5.
6.  public class Stater extends Thread {
7.      private String topic;
8.      private MqttMessage message;
9.
10.     private double[] avg = new double[7];
11.     private double[] sgm = new double[7];
12.     private int n;
13.
14.     public Stater(String t, MqttMessage m) {
15.         topic = t;
16.         message = m;
17.     }
18.
19.     private double calcAverageOnline(int i, double x) {
20.         return ((n-1) * avg[i] + x)/n;
21.     }
22.
23.     private double calcSigmaOnline(int i, double x) {
24.         double sigma = ((n-2) * sgm[i]*sgm[i] +
25.                                 (n * (avg[i] - x)*(avg[i] - x))/(n-1))/(n-1);
26.         if (Double.isNaN(Math.sqrt(sigma))) {
27.             return 0;
28.         }
29.         else {
30.             return Math.sqrt(sigma);
31.         }
32.     }
33.
34.     private double calcAverageOffline(int i, double x) {
35.         return (n * avg[i] - x) / (n - 1);
36.     }
37.
38.     private double calcSigmaOffline(int i, double x) {
39.         double sigma = ((n-1)*sgm[i]*sgm[i] -
40.                                     (n*(avg[i]-x)*(avg[i]-x))/(n-1)) / (n-2);
41.         if (Double.isNaN(Math.sqrt(sigma))) {
42.             return 0;
43.         }
44.         else {
45.             return Math.sqrt(sigma);
46.         }
47.     }
48.
49.     private void hostOnline(String hostName, Attributes stats) {
50.         if (Broker.getStatsMap().isEmpty()) {
51.             //1 if statsMap is empty (1st Host)
52.
53.             Broker.putStatsMap(hostName, stats);
54.
55.             //2 set average to be the same as the only value
56.             avg[0] = stats.cpu;
57.             avg[1] = stats.ram;
58.             avg[2] = stats.location;
59.             avg[3] = stats.latency;
60.             avg[4] = stats.battery;
61.             avg[5] = stats.noTasks;
62.             avg[6] = stats.trust;
63.
```

```
64.             Broker.setAverage(avg);
65.
66.             //3 set sigma to be 0, as per its definition
67.             sgm[0] = 0;
68.             sgm[1] = 0;
69.             sgm[2] = 0;
70.             sgm[3] = 0;
71.             sgm[4] = 0;
72.             sgm[5] = 0;
73.             sgm[6] = 0;
74.
75.             Broker.setSigma(sgm);
76.
77.             //4 Increase number of Hosts
78.             n=1;
79.             Broker.setN(n);
80.         }
81.         else {
82.             //2 if statsMap is not empty
83.             Broker.putStatsMap(hostName, stats);
84.
85.             //5 Increase number of Hosts
86.             n = Broker.getN();
87.             n = n + 1;
88.             Broker.setN(n);
89.
90.
91.             //6 Calculate new average for all 7 of the attributes
92.             avg = Broker.getAverage();
93.
94.             avg[0] = calcAverageOnline(0, stats.cpu);
95.             avg[1] = calcAverageOnline(1, stats.ram);
96.             avg[2] = calcAverageOnline(2, stats.location);
97.             avg[3] = calcAverageOnline(3, stats.latency);
98.             avg[4] = calcAverageOnline(4, stats.battery);
99.      avg[5] = calcAverageOnline(5, stats.noTasks);
100.            avg[6] = calcAverageOnline(6, stats.trust);
101.
102.
103.            Broker.setAverage(avg);
104.
105.            //7 Calculate new average for all 7 of the attributes
106.            sgm = Broker.getSigma();
107.
108.            sgm[0] = calcSigmaOnline(0, stats.cpu);
109.            sgm[1] = calcSigmaOnline(1, stats.ram);
110.            sgm[2] = calcSigmaOnline(2, stats.location);
111.            sgm[3] = calcSigmaOnline(3, stats.latency);
112.            sgm[4] = calcSigmaOnline(4, stats.battery);
113.            sgm[5] = calcSigmaOnline(5, stats.noTasks);
114.            sgm[6] = calcSigmaOnline(6, stats.trust);
115.
116.            Broker.setSigma(sgm);
117.        }
118.       }
119.
120.        private void hostOffline(String hostName) {
121.            Attributes toDel = Broker.getStats(hostName);
122.
123.            n = Broker.getN();
124.
125.            sgm = Broker.getSigma();
126.
127.            //8 Calculate new sigma for all 7 of the attributes
128.            avg = Broker.getAverage();
129.
```

```
            sgm[0] = calcSigmaOffline(0, toDel.cpu);
            sgm[1] = calcSigmaOffline(1, toDel.ram);
            sgm[2] = calcSigmaOffline(2, toDel.location);
            sgm[3] = calcSigmaOffline(3, toDel.latency);
            sgm[4] = calcSigmaOffline(4, toDel.battery);
            sgm[5] = calcSigmaOffline(5, toDel.noTasks);
            sgm[6] = calcSigmaOffline(6, toDel.trust);

            Broker.setSigma(sgm);

            //9 Calculate new average for all 7 of the attributes
            avg[0] = calcAverageOffline(0, toDel.cpu);
            avg[1] = calcAverageOffline(1, toDel.ram);
            avg[2] = calcAverageOffline(2, toDel.location);
            avg[3] = calcAverageOffline(3, toDel.latency);
            avg[4] = calcAverageOffline(4, toDel.battery);
            avg[5] = calcAverageOffline(5, toDel.noTasks);
            avg[6] = calcAverageOffline(6, toDel.trust);

            Broker.setAverage(avg);

            //10 Decrease number of Hosts
            n = n - 1;
            Broker.setN(n);

            Broker.removeStats(hostName);
        }



        @Override
        public void run() {
            synchronized (Broker.lock) {
            //11 Parse the hostname and the message
            String hostName = topic.substring(6);
            String payload = message.toString();
            String[] subl =payload.split("~");

            Attributes stats;

            if (subl[0].equals("online")) {
         //12 if online create new Attributes Object and update the statsMap
                stats = new Attributes(Integer.parseInt(subl[1]),
                       Long.parseLong(subl[2]), Integer.parseInt(subl[3]),
                                    (long) Float.parseFloat(subl[4]),
                                           Integer.parseInt(subl[5]),
                                           Broker.getNoTasks(hostName),
                                           Broker.getTrust(hostName));
                if (Broker.containedStatsMap(hostName)) {
        //13 To calculate new average and sigma, we must first remove and
                      calculate the average and sigma without the old value
                       hostOffline(hostName);
                       hostOnline(hostName, stats);
                }
                else {
                       hostOnline(hostName, stats);
                }
            }
            else if (subl[0].equals("offline") &&
                                         Broker.containedStatsMap(hostName)) {
       //14 if offline remove the Task Host and calculate new average and
                                                                        sigma
                         hostOffline(hostName);
                    }
                }
            }
```

```
196.       }
```

A.1.3 Scorer

```
1.  package broker;
2.
3.  import java.util.Hashtable;
4.  import java.util.Iterator;
5.  import java.util.Set;
6.  import java.util.TreeSet;
7.
8.  public class Scorer extends Thread{
9.
10.     private Hashtable<String, Attributes> statsMap = new Hashtable<String, A
    ttributes>();
11.     private TreeSet<Coo> scoreSet = new TreeSet<Coo>();
12.     private TreeSet<Coo> scoreSetLatency = new TreeSet<Coo>();
13.
14.
15.     private double[] sigma = new double[7];
16.     private double[] average = new double[7];
17.
18.
19.     @Override
20.     public void run() {
21.         synchronized (Broker.lock) {
22.             //1 Get statsMap and average and sigma arrays
23.             statsMap = Broker.getStatsMap();
24.             sigma = Broker.getSigma();
25.             average = Broker.getAverage();
26.
27.             Set<String> keys = statsMap.keySet();
28.             Iterator<String> itr = keys.iterator();
29.
30.             String key;
31.             Attributes stats;
32.
33.             //2 clear the old score sets
34.             scoreSet.clear();
35.             scoreSetLatency.clear();
36.
37.             while (itr.hasNext()) {
38.                 //3 iterate through every Task Host to score it
39.                 Coo score = new Coo();
40.                 Coo scoreLatency = new Coo();
41.                 key = itr.next();
42.
43.                 stats = statsMap.get(key);
44.
45.                 score.name = key;
46.                 scoreLatency.name = key;
47.
48.                 score.rank = 0;
49.                 scoreLatency.rank = 0;
50.                 //4 check if sigma is zero (divby0)
51.                 if (sigma[0] != 0) {
52.                    score.rank += (stats.cpu - average[0]) / sigma[0];
53.                    scoreLatency.rank += (stats.cpu - average[0]) / sigma[0];

54.                 }
55.                 else {
56.                    score.rank += (stats.cpu - average[0]);
57.                    scoreLatency.rank += (stats.cpu - average[0]);
58.                 }
```

```
59.                 if (sigma[1] != 0) {
60.                    score.rank += (stats.ram - average[1]) / sigma[1];
61.                    scoreLatency.rank += (stats.ram - average[1]) / sigma[1];
62.                 }
63.                 else {
64.                    score.rank += (stats.ram - average[1]);
65.                    scoreLatency.rank += (stats.ram - average[1]);
66.                 }
67.                 if (sigma[2] != 0) {
68.                    score.rank += (stats.location - average[2]) / sigma[2];
69.                    scoreLatency.rank += (stats.location - average[2]) /
70.                                                                  sigma[2];
71.                 }
72.                 else {
73.                    score.rank += (stats.location - average[2]);
74.                    scoreLatency.rank += (stats.location - average[2]);
75.                 }
76.                 if (sigma[3] != 0) {
77.                    score.rank += (stats.latency - average[3]) / sigma[3];
78.                    scoreLatency.rank += 2 * (stats.latency - average[3]) /
79.                                                                  sigma[3];
80.                 }
81.                 else {
82.                    score.rank += (stats.latency - average[3]);
83.                    scoreLatency.rank += 2 * (stats.latency - average[3]);
84.                 }
85.                 if (sigma[4] != 0) {
86.                    score.rank += (stats.battery - average[4]) / sigma[4];
87.                    scoreLatency.rank += (stats.battery - average[4]) /
88.                                                                  sigma[4];
89.                 }
90.                 else {
91.                    score.rank += (stats.battery - average[4]);
92.                    scoreLatency.rank += (stats.battery - average[4]);
93.                 }
94.                 if (sigma[5] != 0) {
95.                    score.rank += 1000 * (stats.noTasks - average[5]) /
96.                                                                  sigma[5];
97.                    scoreLatency.rank += 1000 * (stats.noTasks - average[5])
98.                                                                 / sigma[5];
99.                 }
100.                       else {
101.                          score.rank += 1000 * (stats.noTasks - average[5]);
102.                          scoreLatency.rank += 1000 * (stats.noTasks –
103.                                                          average[5]);
104.                }
105.                if (sigma[6] != 0) {
106.                   score.rank += (stats.trust - average[6]) / sigma[6];
107.                   scoreLatency.rank += (stats.trust - average[6]) /
108.                                                                 sigma[6];
109.                }
110.                else {
111.                   score.rank += (stats.trust - average[6]);
112.                   scoreLatency.rank += (stats.trust - average[6]);
113.                }
114.
115.                scoreSet.add(score);
116.                scoreSetLatency.add(scoreLatency);
117.          }
118.
119.          Broker.setScoreSet(scoreSet);
120.          Broker.setScoreSetLatency(scoreSetLatency);
121.       }
122.       }
```

```
123.        }
```

A.1.4 Updater

```
1.  package broker;
2.
3.  public class Updater extends Thread {
4.
5.      private Broker parent;
6.
7.      public Updater (Broker p){
8.          parent = p;
9.      }
10.
11.     @Override
12.     public void run() {
13.         parent.callForStats();
14.     }
15.
16. }
```

A.1.5 Coo

```
1.  package broker;
2.
3.  public class Coo implements Comparable<Coo> {
4.
5.      String name;
6.      double rank;
7.
8.      @Override
9.      public int compareTo(Coo o) {
10.         if (rank < o.rank){
11.             return 1;
12.         }
13.         else {
14.             return -1;
15.         }
16.     }
17.
18. }
```

A.1.6 Attributes

```
1.  package broker;
2.
3.  public class Attributes {
4.
5.      public int cpu;
6.      public long ram;
7.      public int location;
8.      public double latency;
9.      public int battery;
10.     public int noTasks;
11.     public int trust;
12.
13.     public Attributes(int c, long r, int geolocation, long l, int b,
14.                                                         int n, int t) {
15.         cpu = c;
16.         ram = r;
17.         location = geolocation;
18.         if(l!=0) {
```

```
            latency = 1/(float)l;
        }
        else {
             latency = 420420;
        }
        battery = b;
        if(n!=0) {
            noTasks = 1/n;
        }
        else {
             noTasks = 420420;
        }
        trust = t;
    }
}
```

A.2 Gateway Package

A.2.1 Gateway

```
1.  package gateway;
2.
3.  import java.io.BufferedReader;
4.  import java.io.FileReader;
5.  import java.util.Hashtable;
6.  import java.util.Scanner;
7.
8.  import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
9.  import org.eclipse.paho.client.mqttv3.MqttCallback;
10. import org.eclipse.paho.client.mqttv3.MqttClient;
11. import org.eclipse.paho.client.mqttv3.MqttException;
12. import org.eclipse.paho.client.mqttv3.MqttMessage;
13. import org.eclipse.paho.client.mqttv3.MqttPersistenceException;
14. import org.eclipse.paho.client.mqttv3.MqttTopic;
15.
16. public class Gateway implements MqttCallback {
17.
18.     static Hashtable<Integer, Long> ledger = new Hashtable<Integer, Long>();

19.
20.     MqttClient client;
21.
22.     static String BrokerIp;
23.     long startTime;
24.
25.     static int n = 0;
26.
27.     public static void main(String[] args) {
28.         BrokerIp = args[0];
29.         n = Integer.parseInt(args[0]);
30.         new Gateway().connect();
31.
32.     }
33.
34.     public void send(StringBuilder msg, int id) {
35.
36.         MqttMessage message = new MqttMessage();
37.         message.setPayload((msg.toString()).getBytes());
38.
39.         //4 Publish the JSON message
40.         MqttTopic pubTopic = client.getTopic("gateway/toExec");
41.         try {
42.             pubTopic.publish(message);
43.         } catch (MqttPersistenceException e) {
44.             e.printStackTrace();
45.         } catch (MqttException e) {
46.             e.printStackTrace();
47.         }
48.         startTime = System.currentTimeMillis();
49.
50.         //5 Note time to ledger
51.         ledger.put(id, startTime);
52.     }
53.
54.     public void connect() {
55.         try {
56.             //1 connect to MQTT Broker
57.             client = new MqttClient("tcp://"+BrokerIp+":1883", "gateway");
58.             client.connect();
59.             client.setCallback(this);
```

```
60.             //3 Subscribe to topics
61.             client.subscribe("gateway/results");
62.         } catch (MqttException e) {
63.             e.printStackTrace();
64.         }
65.
66.         Sender s = new Sender(n, this);
67.         s.start();
68.
69.         //2 Read user input
70.         Scanner scanner = new Scanner(System.in);
71.         System.out.print("Enter next file to send or \"exit\" to exit #> ");

72.         String file = scanner.nextLine();
73.         int id = 1;
74.         for (int i = 0; i < n; i++) {
75.             while (!file.equals("exit")) {
76.                 try {
77.                     //3 read the message file (JSON) to a string
78.                     BufferedReader buffer = new BufferedReader(new
79.                                             FileReader("msg/JStestMessage.json"));
80.
81.                     StringBuilder msg = new StringBuilder();
82.                     String line = buffer.readLine();
83.
84.                     while (line != null) {
85.                         msg.append(line);
86.                         msg.append("\n");
87.                         line = buffer.readLine();
88.                     }
89.                     msg = msg.deleteCharAt(msg.lastIndexOf("}"));
90.                     msg.append(",\"id\":\""+id+"\"}\n");
91.                     //System.out.print(msg);
92.
93.
94.                     buffer.close();
95.
96.                     MqttMessage message = new MqttMessage();
97.                     message.setPayload((msg.toString()).getBytes());
98.
99.                     //4 Publish the JSON message
100.                            MqttTopic pubTopic = client.getTopic(
101.                                                              "gateway/toExec");
102.                            pubTopic.publish(message);
103.                            startTime = System.currentTimeMillis();
104.
105.                            //5 Note time to ledger
106.                            ledger.put(id, startTime);
107.
108.                        } catch (Exception e) {
109.                            e.printStackTrace();
110.                        }
111.                        System.out.print("Enter next file to send or
112.                                                    \"exit\" to exit #> ");
113.                        file = scanner.nextLine();
114.
115.                        //6 advance id to the next integer
116.                        id++;
117.                    }
118.                    if (file.equals("exit")) {
119.                        //7 in case of input = "exit" leave
120.                        System.exit(0);
121.                    }
122.                }
123.            }
124.
```

```
125.            @Override
126.            public void connectionLost(Throwable cause) {
127.                System.out.println("Connection Lost!!!");
128.            }
129.
130.            //8 Implementation of messageArrived method
131.            @Override
132.            public void messageArrived(String topic, MqttMessage message)
133.                                                        throws Exception {
134.
135.                if (topic.equals("gateway/results")) {
136.
137.                    //9 in case of response, print the response
138.                    System.out.println(message);
139.                    String payload = message.toString();
140.
141.                    //10 get id of response and calculate round trip time
142.                    String[] subl =payload.split("~");
143.                    if (ledger.containsKey(Integer.parseInt(subl[1]))) {
144.                        long elapsedTime = System.currentTimeMillis() -
145.                                    ledger.get(Integer.parseInt(subl[1]));
146.                        System.out.println(subl[2] + "Response time: " +
147.                                                                elapsedTime);
148.                    }
149.                    else {
150.                        //11 if no id is found print error message
151.                        System.out.println("Error: Not recognized id");
152.                    }
153.                }
154.            }
155.
156.            @Override
157.            public void deliveryComplete(IMqttDeliveryToken token) {
158.
159.            }
160.
161.        }
```

A.3 TaskHost Package

A.3.1 TaskHost

```
1.  package taskHost;
2.
3.  import java.io.BufferedReader;
4.  import java.io.InputStreamReader;
5.  import java.io.PrintWriter;
6.  import java.io.StringWriter;
7.  import java.net.InetAddress;
8.  import java.net.NetworkInterface;
9.  import java.util.Enumeration;
10. import java.util.Iterator;
11. import java.util.Random;
12. import java.util.concurrent.Executors;
13. import java.util.concurrent.ScheduledExecutorService;
14. import java.util.concurrent.TimeUnit;
15.
16. import javax.script.*;
17.
18. import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
19. import org.eclipse.paho.client.mqttv3.MqttCallback;
20. import org.eclipse.paho.client.mqttv3.MqttClient;
21. import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
22. import org.eclipse.paho.client.mqttv3.MqttException;
```

```
23. import org.eclipse.paho.client.mqttv3.MqttMessage;
24. import org.eclipse.paho.client.mqttv3.MqttTopic;
25.
26. import org.json.simple.*;
27. import org.json.simple.parser.JSONParser;
28.
29.
30. public class TaskHost implements MqttCallback {
31.
32.     private MqttClient client;
33.     private String name;
34.
35.     static private String BrokerIp;
36.
37.     private int cpu;
38.     private long ram;
39.     private int location;
40.     private long latency;
41.     private Random rand = new Random();
42.     private int battery =rand.nextInt(100);
43.
44.     public void setlocation(int l){
45.         location = l;
46.     }
47.
48.     public static void main(String[] args) {
49.         BrokerIp = args[0];
50.         new TaskHost().connect();
51.     }
52.
53.     public void connect() {
54.         try {
55.
56.             //1 Create Geolocator thread
57.             Thread t1 = new Geolocator(this);
58.             t1.start();
59.
60.             //2 Call Latencer every x seconds
61.             ScheduledExecutorService scorerExecutor =
62.                                     Executors.newSingleThreadScheduledExecutor();
63.             scorerExecutor.scheduleAtFixedRate(new Latencer(BrokerIp, this),
                                                        0, 5, TimeUnit.SECONDS);
64.
65.             //3 Find the name of the Task Host, in this case its Mac Address

66.             boolean flag = true;
67.             InetAddress ip = null;
68.             Enumeration<NetworkInterface> networkInterfaces =
69.                                         NetworkInterface.getNetworkInterfaces();
70.             while (networkInterfaces.hasMoreElements() && flag) {
71.                 NetworkInterface ni =
72.                       (NetworkInterface) networkInterfaces.nextElement();
73.                 Enumeration<InetAddress> nias = ni.getInetAddresses();
74.                 while(nias.hasMoreElements() && flag) {
75.                     ip= (InetAddress) nias.nextElement();
76.                    //4 check that mac address is real, not loopback or other

77.                     if (!ip.isLinkLocalAddress() && !ip.isLoopbackAddress()
78.                                                 && ip instanceof InetAddress) {
79.                             flag = false;
80.                         }
81.                     }
82.
83.             }
84.
85.
```

```
86.             //get interface and mac
87.             //InetAddress ip = InetAddress.getLocalHost();
88.             NetworkInterface interf = NetworkInterface.getByInetAddress(ip);

89.             byte[] mac =  interf.getHardwareAddress();
90.
91.             //convert byte array to string (mac)
92.             StringBuilder sb = new StringBuilder(18);
93.             for (byte b : mac) {
94.                 if (sb.length() > 0) {
95.                     sb.append(':');
96.                 }
97.                 sb.append(String.format("%02x", b));
98.             }
99.             name = sb.toString();
100.
101.                    //5 set the last will to "offline"
102.                    MqttConnectOptions options = new MqttConnectOptions();
103.                    options.setWill("stats/" + name, "offline".getBytes(),
104.                                                                    0, true);
105.
106.                    //6 connect to MQTT broker with mac address as the name
107.                    client = new MqttClient("tcp://"+BrokerIp+":1883", name);

108.                    client.connect(options);
109.                    client.setCallback(this);
110.                    //7 subscribe to topics
111.                    client.subscribe("taskHost/" + name);
112.                    client.subscribe("stats/call");
113.
114.                    MqttMessage stats = new MqttMessage();
115.                    stats.setPayload(getStat().getBytes());
116.                    stats.setRetained(true);
117.
118.                    MqttTopic pubTopic = client.getTopic("stats/" + name);
119.
120.                    //8 publish first stats message
121.
122.                    pubTopic.publish(stats);
123.
124.                } catch (Exception e) {
125.                    e.printStackTrace();
126.                }
127.            }
128.
129.            private String getStat() {
130.
131.                //9 get Stats
132.                cpu = Runtime.getRuntime().availableProcessors();
133.                ram = Runtime.getRuntime().totalMemory();
134.
135.                return "online~"+cpu+"~"+ram+"~"+location+"~"+latency+
136.                                                              "~"+battery;
137.            }
138.
139.            public void setLatency(long x) {
140.                latency = x;
141.
142.            }
143.
144.
145.            //10 Disconnect from the MQTT Broker
146.            public void disconnect() {
147.                try {
148.                    client.disconnect();
149.                } catch (MqttException e) {
```

```
                e.printStackTrace();
            }

        }

        @Override
        public void connectionLost(Throwable cause) {
            System.out.println("Connection Lost!!!");

        }

        //11 Implementation of messageArrived method
        @Override
        public void messageArrived(String topic, MqttMessage message)
                                                            throws Exception {

            if (topic.equals("stats/call")) {

                //12 if the message is a call for stats, get stats and
                                                    publish to personal topic
                MqttMessage stats = new MqttMessage();
                stats.setPayload(getStat().getBytes());
                stats.setRetained(false);

                MqttTopic pubTopic = client.getTopic("stats/" + name);

                pubTopic.publish(stats);
            }

            else if (topic.equals("taskHost/" + name)) {
                //13 if message is a task to execute

                //14 Note start time of execution
                long startTime = System.currentTimeMillis();

                String jsonMsg = new String(message.getPayload());

                //15 Parse JSON file
                JSONObject obj = new JSONObject();
                JSONParser parser = new JSONParser();
                obj = (JSONObject) parser.parse(jsonMsg);


                //16 Get id
                String id = (String) obj.get("id");

                JSONObject execution_script = new JSONObject();
                execution_script = (JSONObject)
                                                obj.get("execution_script");
                JSONArray code = new JSONArray();

                String ret = new String(" ");
                //Python Alternative
                boolean python = false;
                if (python) {
                    try {
                        //17 Get python code
                       code = (JSONArray) execution_script.get("codePy");


                        //18 Write JSON to event.json file
                        PrintWriter writer = new
                                   PrintWriter("event.json", "US-ASCII");
                        writer.println(jsonMsg);
```

```
                        writer.close();

                        //19 write python script to script.py file
                        writer = new
                                    PrintWriter("script.py", "US-ASCII");

                        Iterator<String> it = code.iterator();
                        while (it.hasNext()) {
                            writer.println(it.next());
                        }
                        writer.close();

                        //20 create python process
                        ProcessBuilder pb = new
                                    ProcessBuilder("python","script.py");
                        Process p = pb.start();

                        BufferedReader in = new BufferedReader(
                              new InputStreamReader(p.getInputStream()));

                        StringBuilder retBuilder = new StringBuilder();
                        String line = in.readLine();

                        //21 read result
                        while (line != null) {
                            retBuilder.append(line);
                            retBuilder.append("\n");
                            line = in.readLine();
                        }

                        in.close();
                        ret = retBuilder.toString();
                    }
                    catch (Exception e) {
                        //22 in case python failed
                        python = false;
                    }
                }

                //23 if python failed try javascript
                if (!python) {
                    //Build Script String

                    //24 get javascript code
                    code = (JSONArray) execution_script.get("codeJs");
                    StringBuilder msg = new StringBuilder();
                    Iterator<String> it = code.iterator();
                    while (it.hasNext()) {
                        msg.append(it.next());
                        msg.append("\n");
                    }
                    msg.append("data = " + jsonMsg);
                    msg.append("print(escr(data));");

                    //25 Create Javascript Engine
                    ScriptEngineManager manager = new
                                                    ScriptEngineManager();
                    ScriptEngine engine = manager.getEngineByName("js");


                    StringWriter sw = new StringWriter();
                    PrintWriter pw = new PrintWriter(sw);
                    engine.getContext().setWriter(pw);
                    try {
                        engine.eval(msg.toString());
                    }
```

```
                        catch (Exception e) {
                            System.out.println(e);
                        }

                        //26 Get result
                        ret = sw.getBuffer().toString();
                    }


                    MqttMessage answer = new MqttMessage();
                    answer.setRetained(false);
                    String tmp = new String();
                    //27 Publish self, name, id of the task executed
                                                                and its result
                    tmp = name + "~" + id + "~" + ret;
                    answer.setPayload(tmp.getBytes());

                    MqttTopic pubTopic = client.getTopic("taskHost/results");

                    pubTopic.publish(answer);

                    //28 Note end time of execution and print
                    long endTime = System.currentTimeMillis();

                    System.out.println(endTime - startTime);
                }
            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken token) {

            }

        }
```

A.3.2 Geolocator

```
package taskHost;

import java.util.Random;

public class Geolocator extends Thread {
    private int zone;
    private int direction; //-1 outwards, 1 inwards
    private boolean inBounds = true;
    private TaskHost parent;

    private Random rand = new Random();

    public Geolocator (TaskHost p){
        parent = p;
    }

    @Override
    public void run() {
        //1 starting place
        zone = rand.nextInt(10) + 1;
        direction = rand.nextInt(2);
        if (direction == 0) {
            direction = -1;
        }

```

```
        //2 set starting place on TaskHost
        parent.setlocation(zone*direction);

        //3 move
        while (inBounds) {
            try {
                Thread.sleep(rand.nextInt(1000));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            if (direction == 1) {
                if (zone > 1) {
                    zone--;
                }
                //4 if it reached the lowest zone, change direction
                                                    (going outwards)
                else{
                    zone = 2;
                    direction = -1;
                }
            } else {
                if (zone < 10) {
                    zone++;
                }
                else{
                    inBounds = false;
                }
            }
            parent.setlocation(zone*direction);

        }
        //5 if out of bounds, disconnect
        parent.disconnect();


    }

}
```

A.3.3 Latencer

```
package taskHost;

import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;

public class Latencer extends Thread {

    private String brokerIP = new String();
    private TaskHost parent;

    public Latencer(String ip,TaskHost p) {
        brokerIP = ip;
        parent = p;
    }

    @Override
    public void run() {
        long start = System.currentTimeMillis();
        boolean isPinged = false;
        try {
```

```
22.             isPinged = InetAddress.getByName(brokerIP).isReachable(2000);
23.         } catch (UnknownHostException e1) {
24.             e1.printStackTrace();
25.         } catch (IOException e1) {
26.             e1.printStackTrace();
27.         }
28.         long stop = System.currentTimeMillis();
29.
30.         if (isPinged) {
31.             parent.setLatency(stop-start);
32.         }
33.         else {
34.             parent.setLatency(420420);
35.         }
36.     }
37. }
```

A.4 Εφαρμογη Android

A.4.1 MainActivity

```
1.  package com.kereji.fog.taskhost;
2.
3.  import android.Manifest;
4.  import android.app.PendingIntent;
5.  import android.content.Intent;
6.  import android.content.pm.PackageManager;
7.  import android.location.Location;
8.  import android.os.Bundle;
9.  import android.provider.Settings;
10. import android.support.v4.content.ContextCompat;
11. import android.support.v7.app.AppCompatActivity;
12. import android.util.Log;
13. import android.widget.TextView;
14. import android.widget.EditText;
15. import android.view.View;
16.
17. import com.google.android.gms.common.ConnectionResult;
18. import com.google.android.gms.common.api.GoogleApiClient;
19. import com.google.android.gms.common.api.ResultCallback;
20. import com.google.android.gms.common.api.Status;
21. import com.google.android.gms.location.Geofence;
22. import com.google.android.gms.location.GeofencingRequest;
23. import com.google.android.gms.location.LocationServices;
24.
25. import org.eclipse.paho.client.mqttv3.MqttException;
26.
27. import java.util.ArrayList;
28.
29. import helpers.GeofenceTransitionsIntentService;
30. import helpers.MqttHelper;
31.
32. public class MainActivity extends AppCompatActivity
33.         implements
34.         GoogleApiClient.ConnectionCallbacks,
35.         GoogleApiClient.OnConnectionFailedListener,
36.         ResultCallback<Status>{
37.     private Location currentLocation;
38.     MqttHelper mqttHelper;
39.
40.     TextView dataReceived;
41.
42.     protected ArrayList<Geofence> mGeofenceList;
43.     protected GoogleApiClient mGoogleApiClient;
44.
45.     String brokerIP;
46.
47.     @Override
48.     protected void onCreate(Bundle savedInstanceState) {
49.         super.onCreate(savedInstanceState);
50.
51.         //1 Create google api client to connect to google play services
52.         mGoogleApiClient = new GoogleApiClient.Builder(this)
53.                 .addConnectionCallbacks(this)
54.                 .addOnConnectionFailedListener(this)
55.                 .addApi(LocationServices.API)
56.                 .build();
57.         mGoogleApiClient.connect();
58.
59.         //2 Show the UI layout to the screen
60.         setContentView(R.layout.activity_main);
```

```
61.
62.         dataReceived = (TextView) findViewById(R.id.dataReceived);
63.
64.
65.         mGeofenceList = new ArrayList<Geofence>();
66.     }
67.
68.     @Override
69.     protected void onDestroy() {
70.         super.onDestroy();
71.         //3 on destroy disconnect from MQTT Broker
72.         try {
73.             mqttHelper.disconnect();
74.         } catch (MqttException e) {
75.             e.printStackTrace();
76.         }
77.
78.
79.     }
80.
81.     //4 Read Broker IP from UI
82.     public void newBrokerIP(View view) {
83.         Log.w("Debug","onclick");
84.         EditText editText = (EditText) findViewById(R.id.brokerIP);
85.         brokerIP = editText.getText().toString();
86.
87.
88.         dataReceived.setText("Connecting to " + brokerIP);
89.         startMqtt();
90.     }
91.
92.     //5 Build the Geofences
93.     private GeofencingRequest getGeofencingRequest() {
94.         GeofencingRequest.Builder builder = new GeofencingRequest.Builder();

95.         builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);

96.         builder.addGeofences(mGeofenceList);
97.         return builder.build();
98.     }
99.
100.            private PendingIntent getGeofencePendingIntent() {
101.                Intent intent = new Intent(this,
102.                                           GeofenceTransitionsIntentService.class);
103.                return PendingIntent.getService(this, 0, intent,
104.                                                PendingIntent.FLAG_UPDATE_CURRENT);
105.            }
106.
107.            //6 create geofence list centered on broker ip
108.            public void populateGeofenceList(double a, double b) {
109.                for (int i =1; i<=10;i++) {
110.                    mGeofenceList.add(new Geofence.Builder()
111.                            .setRequestId(""+i)
112.                            .setCircularRegion(
113.                                    a,
114.                                    b,
115.                                    i*50
116.                            )
117.                            .setExpirationDuration(0)
118.                            .setTransitionTypes(
119.                                    Geofence.GEOFENCE_TRANSITION_ENTER |
120.                                        Geofence.GEOFENCE_TRANSITION_EXIT)
121.                                                                    .build());
122.                }
123.            }
124.
```

```
125.            private void startMqtt() {
126.
127.                //7 get device ID to be used as client name
128.                String deviceId = Settings.Secure.getString(
129.                     this.getContentResolver(), Settings.Secure.ANDROID_ID);
130.                Log.w("Debug", "THIS IS THE DEVICE ID!!!! " + deviceId);
131.
132.                //8 define subscription topics
133.                ArrayList<String> subList = new ArrayList<String>();
134.
135.                subList.add("taskHost/" + deviceId);
136.                subList.add("stats/call");
137.                subList.add("stats/broker");
138.
139.                mqttHelper = new MqttHelper(getApplicationContext(),
140.                                            brokerIP, deviceId, subList, this);
141.            }
142.
143.            @Override
144.            protected void onStart() {
145.                super.onStart();
146.                if (!mGoogleApiClient.isConnecting()
147.                                        || !mGoogleApiClient.isConnected()) {
148.                    mGoogleApiClient.connect();
149.                }
150.            }
151.
152.            @Override
153.            protected void onStop() {
154.                super.onStop();
155.                if (mGoogleApiClient.isConnecting()
156.                                        || mGoogleApiClient.isConnected()) {
157.                    mGoogleApiClient.disconnect();
158.                }
159.            }
160.
161.            @Override
162.            public void onConnected(Bundle connectionHint) {
163.                //9 when connected to google play services,
164.                       find current location and set task host's initial
165.                                                location on the geofencing grid
166.                Log.w("Location", "Inside");
167.                if (ContextCompat.checkSelfPermission(this,
168.                                Manifest.permission.ACCESS_COARSE_LOCATION)
169.                                        == PackageManager.PERMISSION_GRANTED) {
170.                    Location lastLocation =
171.        LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);
172.
173.                    double lat = lastLocation.getLatitude(),
174.                                        lon = lastLocation.getLongitude();
175.                    Log.w("Location", "Current location is:
176.                                                    "+ lat + "," + lon);
177.
178.                    currentLocation = lastLocation;
179.                    Log.w("Location", "Current location is:
180.                                                "+ currentLocation.toString());
181.
182.                    MqttHelper.setLocation(currentLocation);
183.
184.                }
185.            }
186.
187.            @Override
188.            public void onConnectionFailed(ConnectionResult result) {
189.            }
```

```
190.
191.            @Override
192.            public void onConnectionSuspended(int cause) {
193.                mGoogleApiClient.connect();
194.            }
195.
196.            @Override
197.            public void onResult(Status status) {
198.            }
199.
200.            //10 Called when we have broker position
201.            public void setGeoFence(float a, float b) {
202.                // BROKER LAT~LNG
203.                populateGeofenceList(a,b);
204.
205.                //11 Define GeofenceTransitionsIntentService as
206.                                                              geofencing intent
207.                try {
208.                    LocationServices.GeofencingApi.addGeofences(
209.                            mGoogleApiClient,
210.                            getGeofencingRequest(),
211.                            getGeofencePendingIntent()
212.                    ).setResultCallback(this);
213.                } catch (SecurityException securityException) {
214.                }
215.
216.                Log.w("newBrokerIP", "GEOFENCES SET");
217.
218.                for (Geofence i : mGeofenceList) {
219.                    Log.w("newBrokerIP", i.getRequestId());
220.                }
221.
222.            }
223.
224.            //12 Set the text on UI to s
225.            public void setText(String s) {
226.                dataReceived.setText(s);
227.            }
228.        }
```

A.4.2 Layout

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.a
    ndroid.com/apk/res/android"
3.      xmlns:app="http://schemas.android.com/apk/res-auto"
4.      xmlns:tools="http://schemas.android.com/tools"
5.      android:layout_width="match_parent"
6.      android:layout_height="match_parent"
7.      tools:context=".MainActivity">
8.
9.      <TextView
10.         android:id="@+id/dataReceived"
11.         android:layout_width="wrap_content"
12.         android:layout_height="wrap_content"
13.         app:layout_constraintBottom_toBottomOf="parent"
14.         app:layout_constraintLeft_toLeftOf="parent"
15.         app:layout_constraintRight_toRightOf="parent"
16.         app:layout_constraintTop_toTopOf="parent" />
17.
18.     <EditText
19.         android:id="@+id/brokerIP"
20.         android:layout_width="0dp"
21.         android:layout_height="wrap_content"
```

```
        android:layout_marginLeft="16dp"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp"
        android:ems="10"
        android:hint="@string/edit_message"
        android:inputType="textPersonName"
        app:layout_constraintEnd_toStartOf="@+id/button2"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button2"
        android:layout_width="68dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        android:layout_marginStart="16dp"
        android:onClick="newBrokerIP"
        android:text="@string/button_ok"
        app:layout_constraintBaseline_toBaselineOf="@+id/brokerIP"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toEndOf="@+id/brokerIP" />

</android.support.constraint.ConstraintLayout>
```

A.4.3 GeofenceTransitionsIntentService

```
package helpers;

import android.app.IntentService;
import android.content.Intent;
import android.util.Log;

import com.google.android.gms.location.Geofence;
import com.google.android.gms.location.GeofencingEvent;

public class GeofenceTransitionsIntentService extends IntentService {
    protected static final String TAG = "GeofenceTransitionsIS";

    public GeofenceTransitionsIntentService() {
        super(TAG);  // use TAG to name the IntentService worker thread
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        GeofencingEvent event = GeofencingEvent.fromIntent(intent);
        if (event.hasError()) {
            Log.e(TAG, "GeofencingEvent Error: " + event.getErrorCode());
            return;
        }

        //1 Set task host's zone to equal the id of the geofence (1 to 10)
        for (Geofence geofence : event.getTriggeringGeofences()) {
            MqttHelper.setZone(Integer.parseInt(geofence.getRequestId()));
        }

        // Get the transition type.
        int geofenceTransition = event.getGeofenceTransition();

```

```
        //2 Set task host's direction depending on event exit or enter
        if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER ) {
            MqttHelper.setDirection(1);
        }else if(geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT){
            MqttHelper.setDirection(-1);
        }else{
            Log.w("GEOFENCE", "Event not handled"+geofenceTransition);
        }
    }
}
```

A.4.4 MqttHelper

```
package helpers;

import android.content.Context;
import android.location.Location;
import android.util.Log;

import java.util.ArrayList;
import android.content.IntentFilter;
import android.content.Intent;
import android.os.BatteryManager;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Iterator;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.client.mqttv3.DisconnectedBufferOptions;
import org.eclipse.paho.client.mqttv3.IMqttActionListener;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.IMqttToken;
import org.eclipse.paho.client.mqttv3.MqttCallbackExtended;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.liquidplayer.javascript.JSContext;
import org.liquidplayer.javascript.JSValue;

import com.kereji.fog.taskhost.MainActivity;

public class MqttHelper implements MqttCallbackExtended {
    public MqttAndroidClient mqttAndroidClient;

    String serverUri = new String();

    String clientId = new String();
    ArrayList<String> subList = new ArrayList<String>();

    static int zone = 1;
    static int direction = 1;

    float brokerLat;
    float brokerLng;
    MainActivity parent;

    static Location currentLocation = new Location("host");

```

```
52.     static Context appContext;
53.
54.     float latency = 50;
55.
56.
57.     public MqttHelper(Context context, String brokerIP, String name,
58.                                   ArrayList<String> sList, MainActivity p) {
59.
60.         ScheduledExecutorService scorerExecutor =
61.                                 Executors.newSingleThreadScheduledExecutor();
62.         scorerExecutor.scheduleAtFixedRate(
63.                   new Latencer(serverUri, this), 0, 10, TimeUnit.SECONDS);
64.
65.         //1 Get brokerIP, name and subscription list from MainActivity
66.         parent = p;
67.         appContext = context;
68.
69.         serverUri = brokerIP ;
70.         clientId = name;
71.         subList.clear();
72.         subList.addAll(sList);
73.
74.         mqttAndroidClient = new MqttAndroidClient(context,
75.                                   "tcp://" + serverUri +":1883", clientId);
76.         mqttAndroidClient.setCallback(this);
77.
78.         connect();
79.     }
80.
81.     public void setCallback(MqttCallbackExtended callback) {
82.         mqttAndroidClient.setCallback(callback);
83.     }
84.
85.     private void connect() {
86.         MqttConnectOptions mqttConnectOptions = new MqttConnectOptions();
87.         mqttConnectOptions.setAutomaticReconnect(true);
88.         mqttConnectOptions.setCleanSession(false);
89.
90.
91.         try {
92.
93.             if (mqttAndroidClient.isConnected()){
94.                 mqttAndroidClient.disconnect();
95.             }
96.
97.             //2 Attempt to connect to MQTT Broker
98.             mqttAndroidClient.connect(mqttConnectOptions,
99.                                             null, new IMqttActionListener() {
100.                         @Override
101.                         public void onSuccess(IMqttToken asyncActionToken) {
102.
103.                             Log.w("Mqtt", "Connected to: "
104.                                             + "tcp://" + serverUri +":1883");
105.
106.                             DisconnectedBufferOptions
107.                                             disconnectedBufferOptions =
108.                                             new DisconnectedBufferOptions();
109.                             disconnectedBufferOptions.setBufferEnabled(true);
110.                             disconnectedBufferOptions.setBufferSize(100);
111.                            disconnectedBufferOptions.setPersistBuffer(false);
112.                             disconnectedBufferOptions.
113.                                             setDeleteOldestMessages(false);
114.                             mqttAndroidClient.
```

```
                                setBufferOpts(disconnectedBufferOptions);

                            for (String i : subList) {
                                subscribeToTopic(i);
                            }

                            //3 Send first message to broker
                            MqttMessage message = new MqttMessage();
                            String tmp = getStat();
                            message.setPayload(tmp.getBytes());

                            try {
                                mqttAndroidClient.publish("stats/"
                                                    + clientId, message);
                            } catch (MqttException e) {
                                e.printStackTrace();
                            }


                        }

                        @Override
                        public void onFailure(IMqttToken asyncActionToken,
                                                        Throwable exception) {
                            Log.w("Mqtt", "Failed to connect to: " + "tcp://"
                                + serverUri +":1883" + exception.toString());
                        }
                    });


                } catch (MqttException ex) {
                    ex.printStackTrace();
                }
            }

            //4 method to call when we want to subscribe to a topic
            private void subscribeToTopic(final String subTopic) {
                try {
                    mqttAndroidClient.subscribe(subTopic, 0, null,
                                            new IMqttActionListener() {
                        @Override
                        public void onSuccess(IMqttToken asyncActionToken) {

                            Log.w("Mqtt", "Subscribed! to " + subTopic);
                        }

                        @Override
                        public void onFailure(IMqttToken asyncActionToken,
                                                    Throwable exception) {
                            Log.w("Mqtt", "Subscribed fail! to " + subTopic);

                        }
                    });

                } catch (MqttException ex) {
                    System.err.println("Exception whilst subscribing");
                    ex.printStackTrace();
                }
            }


            @Override
            public void connectComplete(boolean b, String s) {
                Log.w("mqtt", s);
                parent.setText(s);
            }
```

```
179.
180.            @Override
181.            public void connectionLost(Throwable throwable) {
182.
183.            }
184.
185.            @Override
186.            public void messageArrived(String topic,
187.                                       MqttMessage mqttMessage) throws Exception {
188.                Log.w("Mqtt message: ",
189.                                       mqttMessage.toString() + " from " + topic);
190.
191.
192.                if (topic.equals("stats/broker")) {
193.                    //5 Get broker location and create geofences
194.                    Log.w("Mqtt message: ", "Broker stats received");
195.                    String[] subl = (mqttMessage.toString()).split("~");
196.                    brokerLat = Float.parseFloat(subl[0]);
197.                    brokerLng = Float.parseFloat(subl[1]);
198.                    parent.setGeoFence(brokerLat, brokerLng);
199.
200.                    Location brokerLocation = new Location("broker");
201.                    setZone((int) currentLocation.distanceTo
202.                                                        (brokerLocation) /50);
203.                    //Set at the worst place until it moves in geofence
204.                              area (in case distance from broker > 50 * 10)
205.                    setDirection(-1);
206.
207.                    Log.w("Location", "distance is " +
208.                                currentLocation.distanceTo(brokerLocation));
209.
210.
211.                }
212.                if (topic.equals("taskHost/" + clientId)) {
213.                    //6 if message is a task to execute
214.
215.                    //7 Note start time of execution
216.                    long startTime = System.currentTimeMillis();
217.
218.                    String jsonMsg = new String(mqttMessage.getPayload());
219.
220.                    //8 Parse JSON file
221.                    JSONObject obj = new JSONObject();
222.                    JSONParser parser = new JSONParser();
223.                    obj = (JSONObject) parser.parse(jsonMsg);
224.
225.                    //9 Get id
226.                    String id = (String) obj.get("id");
227.                    Log.w("JSONparse", id);
228.
229.
230.                    //10 get Javascript code
231.                    JSONObject execution_script = new JSONObject();
232.                    execution_script = (JSONObject)
233.                                                obj.get("execution_script");
234.                    JSONArray code = new JSONArray();
235.                    code = (JSONArray) execution_script.get("codeJs");
236.
237.                    StringBuilder msg = new StringBuilder();
238.                    Iterator<String> it = code.iterator();
239.                    while (it.hasNext()) {
240.                        msg.append(it.next());
241.                        msg.append("\n");
242.                    }
243.                    msg.append("data = " + jsonMsg);
244.                    msg.append("a = escr(data);");
```

```
                    Log.w("JSONparse", msg.toString());

                    //11 run javascript code and get return value
                    JSContext context = new JSContext();
                    context.evaluateScript(msg.toString());
                    JSValue newAValue = context.property("a");
                    Log.w("JS", "a is "+ newAValue.toString());

                    String ret = newAValue.toString();
                    MqttMessage answer = new MqttMessage();
                    answer.setRetained(false);
                    String tmp = new String();
                    tmp = clientId + "~" + id + "~" + ret.toString();
                    answer.setPayload(tmp.getBytes());

                    //12 Publish self-name, id of the task executed
                                                            and its result
                    try {
                        Log.w("MQTTpub", "forwarding results");
                       mqttAndroidClient.publish("taskHost/results", answer);

                    } catch (MqttException e) {
                        e.printStackTrace();
                    }

                    long endTime = System.currentTimeMillis();

                    parent.setText(ret.toString() + "\n~~~~~~~~~~\n" +
                                        (endTime-startTime) + "\n~~~~~~~~~~");

                    System.out.println(endTime - startTime);
                }

                if (topic.equals("stats/call")) {
                    //13 if the message is a call for stats get stats
                                             and publish to personal topic
                    Log.w("Mqtt message: ", "called for stats");

                    String tmp = getStat();
                    Log.w("Mqtt message: ", "cpu + ram + location +
                                            latency + battery: " +tmp);

                    parent.setText("cpu + ram + location + latency +
                                                            battery " +tmp);

                    MqttMessage message = new MqttMessage();
                    message.setPayload(tmp.getBytes());

                    try {
                      mqttAndroidClient.publish("stats/" +clientId, message);

                    } catch (MqttException e) {
                        e.printStackTrace();
                    }
                }
            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken
                                                    iMqttDeliveryToken) {
            }

            public void disconnect() throws MqttException {
                if (mqttAndroidClient.isConnected()){
                    mqttAndroidClient.disconnect();
                }
```

```
309.            }
310.
311.            private String getStat() {
312.                int cpu = Runtime.getRuntime().availableProcessors();
313.                long ram = Runtime.getRuntime().totalMemory();
314.
315.                int battery = getBatteryPercentage();
316.
317.                return "online~" + cpu + "~" + ram + "~" +
318.                           zone*direction + "~" + latency + "~" + battery ;
319.
320.            }
321.
322.            private int getBatteryPercentage() {
323.
324.                //14 We create an Intent to query for the battey status
325.                IntentFilter iFilter = new IntentFilter(
326.                                                Intent.ACTION_BATTERY_CHANGED);
327.                Intent batteryStatus = appContext.registerReceiver(null,
328.                                                                   iFilter);
329.
330.                int level = batteryStatus != null ?
331.                  batteryStatus.getIntExtra(BatteryManager.EXTRA_LEVEL, -1)
332.                                                                        : -1;
333.
334.                return level;
335.            }
336.
337.            public void setLatency(long x) {
338.                latency = x;
339.            }
340.
341.
342.            public static void setLocation(Location l) {
343.                currentLocation = l;
344.            }
345.
346.            public static void setDirection(int x) {
347.                direction = x;
348.            }
349.
350.            public static void setZone(int x) {
351.                zone = x;
352.            }
353.        }
```

A.4.5 Latencer

```
1.  package helpers;
2.
3.  import android.util.Log;
4.
5.  import java.io.BufferedReader;
6.  import java.io.InputStreamReader;
7.
8.  public class Latencer extends Thread {
9.      private String brokerIP = new String();
10.     private MqttHelper parent;
11.
12.     public Latencer(String ip,MqttHelper p) {
13.         brokerIP = ip;
14.         parent = p;
15.     }
16.
```

```
17.     @Override
18.     public void run() {
19.         String ret = new String();
20.         Log.w("ping: ", brokerIP);
21.         try {
22.             Process p = Runtime.getRuntime().exec("ping -w 5
23.                                                    c 3 " + brokerIP);
24.             p.waitFor();
25.
26.
27.
28.             BufferedReader in = new BufferedReader(new
29.                                     InputStreamReader(p.getInputStream()));
30.
31.             String line = in.readLine();
32.
33.             while (line != null) {
34.                 Log.w("ping: ", line);
35.                 if (line.contains("time=")) {
36.                     ret = line;
37.                     break;
38.                 }
39.                 line = in.readLine();
40.             }
41.             in.close();
42.
43.             Log.w("ping: ", ret);
44.
45.             ret = ret.substring(ret.lastIndexOf("=")+1,
46.                                                   ret.lastIndexOf("m")-1);
47.
48.         }
49.         catch (Exception e) {
50.             //
51.         }
52.         if (ret.isEmpty()) {
53.             parent.setLatency(1000);
54.         }else {
55.             parent.setLatency(Long.parseLong(ret));
56.         }
57.     }
```

Παράρτημα Β

User-Desktop PC (χρήση σαν Task Host)

OS	Windows 7 Professional
Chipset	Intel® 100 Series/C230
CPU	Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz, 4 Core(s), 4 Logical Processor(s)
RAM	8 GB

Gate-Laptop (χρήση σαν Gateway)

OS	Windows 10 Home
Chipset	Intel® 6 Series/C200
CPU	Intel(R) Core(TM) i5-2430M CPU @ 2.4 GHz, 2 Core(s), 4 Logical Processor(s)
RAM	6 GB

Ulich-Desktop PC (χρήση σαν MQTT Broker and the Fog Broker)

OS	Windows 7 Professional
Chipset	Intel® 6 Series/C200
CPU	Intel(R) Core(TM) i5-2500K CPU @ 3.30GHz, 4 Core(s), 4 Logical Processor(s)
RAM	8 GB

LG G3

OS	Andorid 5
Chipset	Qualcomm MSM8974AC Snapdragon 801
CPU	Quad-core 2.5 GHz Krait 400
RAM	3 GB

Xiaomi Mi Max 2

OS	Android 7.1.1
Chipset	Qualcomm MSM8953 Snapdragon 625
CPU	Octa-core 2.0 GHz Cortex-A53
RAM	4 GB

Xiaomi Mi A1

OS	Andorid 7.1.2
Chipset	Qualcomm MSM8953 Snapdragon 625
CPU	Octa-core 2.0 GHz Cortex-A53
RAM	4 GB

Samsung Galaxy S5

OS	Andorid 6.0.1
Chipset	Qualcomm MSM8974AC Snapdragon 801
CPU	Quad-core 2.5 GHz Krait 400
RAM	2 GB

Βιβλιογραφία

[1] Ed Pilkington, Amanda Michel, "The Guardian", [Online]. Available: https://www.theguardian.com/world/2012/feb/17/obama-digital-data-machine-facebook-election. [Accessed 09 06 2018].

[2] European Parliament, "Industry 4.0 Digitalisation for productivity and growth", September 2015.

[3] Smart Nation Singapore, "https://www.smartnation.sg/", [Online].

[4] J. Schaefer, "Leverege", 14 01 2017. [Online]. Available: https://www.leverege.com/blogpost/smart-devices-what-makes-them-smart. [Accessed 09 06 2018].

[5] Cisco, "Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are".

[6] Cisco, "https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html", [Online]. [Accessed 09 06 2018].

[7] A. Kapsalis, P.Kasnesis, I. S. Venieris, D. I. Kaklamani, C. Z. Patrikakis, "A Cooperative Fog Approach for Effective Workload Balancing", *IEEE Cloud Computing,* vol. 4, no. 2, pp. 36-45, 2017.

[8] National Institute of Standards and Technology, "The NIST Definition of Cloud", 2011.

[9] A. Nordum, "IEEE Spectrum", 18 08 2016. [Online]. Available: https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated. [Accessed 09 06 2018].

[10] Chin-Lung Hsu, Judy Chuan-Chuan Lin, "An empirical examination of consumer adoption of Internet of Things services: Network externalities and concern for information privacy perspectives", *Computers in Human Behavior,* vol. 62, pp. 516-527, 2016.

[11] Sanjit A. Seshia, Shiyan Hu, Wenchao Li, Qi Zhu, "Design Automation of Cyber-Physical Systems: Challenges, Advances, and Opportunities", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 36, no. 9, pp. 1421-1434, 2017.

[12] U.S. Department of Energy, "SmartGrid.gov", [Online]. Available: https://www.smartgrid.gov/the_smart_grid/smart_grid.html. [Accessed 09 06 2018].

[13] Amsterdam Smart City, "Amsterdam Smart City", [Online]. Available: https://amsterdamsmartcity.com/projects. [Accessed 09 06 2018].

[14] Dublin City Council, "Smart Dublin", [Online]. Available: http://smartdublin.ie/. [Accessed 09 06 2018].

[15] T. Z. Mung Chiang, "Fog and IoT: An Overview of Research Opportunities", *IEEE Internet of Things Journal,* vol. 3, no. 6, pp. 854-864, 2016.

[16] N. Cochrane, "US smart grid to generate 1000 petabytes of data a year", 23 03 2010. [Online]. Available: https://www.itnews.com.au/news/us-smart-grid-to-generate-1000-petabytes-of-data-a-year-170290#ixzz458VaITi6. [Accessed 09 06 2018].

[17] Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, Nigel Davies, "The Case for VM-Based Cloudlets in Mobile Computing", *IEEE Pervasive Computing,* vol. 8, no. 4, pp. 14-23, 2009.

[18] W. Ashford, "ComputerWeekly.com", 15 10 2014. [Online]. Available: https://www.computerweekly.com/news/2240232680/Industrial-control-systems-What-are-the-security-challenges. [Accessed 09 06 2018].

[19] Ola Salman, Imad Elhajj, Ayman Kyssi, Ali Chehab, "Edge computing enabling the Internet of Things", in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)* , 2015.

[20] Pedro Garcia Lopez et al, "Edge-centric Computing: Vision and Challenges", *ACM SIGCOMM Computer Communication Review,* vol. 45, no. 5, 2015.

[21] Flavio Bonomi, Rodolfo Milito, Jianz Zhi, Sateesh Addepalli, "Fog computing and its role in the internet of things", in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 2012.

[22] OpenFog, "OpenFog Reference Architecture for Fog Computing", 2017.

[23] OASIS, "MQTT Version 3.1.1", 29 10 2014. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html. [Accessed 09 06 2018].

[24] A. Stanford-Clark, 17 06 2014. [Online]. Available: https://groups.google.com/forum/#!msg/mqtt/zRqd8JbY4oM/XrMwlQ5TU0EJ. [Accessed 09 06 2018].

[25] Eclipse, "Paho Javadoc", [Online]. Available: https://www.eclipse.org/paho/files/javadoc/index.html. [Accessed 09 06 2018].

[26] Prakash P, Darshaun K. G., Yaazhlene P, Medidhi Venkata Ganesh, Vasudha B, "Fog Computing: Issues, Challenges and Future Directions", *International Journal of Electrical and Computer Engineering (IJECE),* vol. 7, no. 6, pp. 3669-3673, 2017.

[27] Tom H. Luan et al, "Fog Computing: Focusing on Mobile Users at the Edge", 30 03 2016. [Online]. Available: https://arxiv.org/abs/1502.01815. [Accessed 09 06 2018].

[28] Manos Antonakakis et al, "Understanding the Mirai Botnet", in *26th USENIX Security Symposium* , 2017.

[29] Nikolas Roman Herbst, Samuel Kounev, Ralf Reussner, "Elasticity in Cloud Computing: What It Is, and What It Is Not", in *International Conference on Autonomic Computing*, 2013.