Τμήμα Μηχανικών Πληροφορικής Τ.Ε.
Τ.Ε.Ι. Δυτικής Ελλάδας

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Χρήση του MQTT πρωτόκολλου για απομακρυσμένη λήψη τιμών από αισθητήρες κινητού τηλεφώνου»

Γεώργιος Δρίβας
Α.Μ. 1624

Επιβλέπων: Αντωνόπουλος Χρήστος

Ναύπακτος
Ιούνιος 2016

Περιεχόμενα

Εισαγωγή

Η παρούσα πτυχιακή εργασία κάνει μια διερεύνηση του χώρου των τεχνολογιών του Διαδικτύου των Πραγμάτων (Internet of Things – IoT) και ασχολείται ειδικότερα με το πρωτόκολλο MQTT το οποίο επιτρέπει σε συσκευές να ανταλλάσσουν μηνύματα σε δύσκολα κι ασταθή περιβάλλοντα όσο αφορά την αυτονομία και συνδεσιμότητα.

Το IoT είναι κατά πολλούς είναι το επόμενο βήμα εξέλιξης του Internet όπως το ξέρουμε ως σήμερα. Η εξέλιξη αυτή έχει παρασύρει και τις ίδιες τις διάφορες υπολογιστικές συσκευές που χρησιμοποιεί ο άνθρωπος στην καθημερινότητά του, εξέλιξη που είναι απόρροια του οράματος του Mark Weiser για το πως θα έπρεπε να είναι ο υπολογιστής (και γενικότερα ο υπολογισμός) στον 21^{ο} αιώνα και που περιγράφεται στο κεφάλαιο 1.

Στο κεφάλαιο 2 περιγράφεται το IoT, δίνονται παραδείγματα του που υπάρχουν ή που είναι αντικείμενο έρευνας και πως αυτά μπορούν να βελτιώσουν τη ζωή του ανθρώπου είτε σε ατομικό είτε σε συλλογικό επίπεδο. Παρουσιάζονται επίσης οι βασικές τεχνολογίες που αποτελούν το IoT.

Το κεφάλαιο 3 αναλύει το πρωτόκολλο MQTT που έχει είναι το βασικό αντικείμενο μελέτης της εργασίας αυτής. Παρουσιάζονται τόσο η κεντρική του ιδέα όσο και οι βασικές του λειτουργίες όπως αυτές αναφέρονται στις προδιαγραφές του. Παρουσιάζονται επίσης και οι βασικότεροι MQTT brokers που υπάρχουν διαθέσιμοι την εποχή που γράφεται η εργασία αυτή.

Στο κεφάλαιο 4 περιγράφεται το σύστημα MapTrack που αναπτύχθηκε στα πλαίσια της πτυχιακής εργασίας. Το σύστημα βασίζεται στην τεχνολογία MQTT,

παρακολουθεί και αναπαριστά στο χάρτη θέσεις χρηστών. Τα βασικά σημεία του κώδικα του MapTrack δίνονται σε ειδικό παράρτημα στο τέλος της εργασίας.

Στα τελευταία δύο κεφάλαια έχουν γραφεί αναλυτικές οδηγίες για το πως μπορεί κανείς να κατεβάσει και να εγκαταστήσει τα σχετικά λογισμικά που χρησιμοποιήθηκαν στα πλαίσια αυτής της πτυχιακής ώστε να μπορεί ο κάθε ενδιαφερόμενος να τις χρησιμοποιήσει.

1. Οι υπολογιστές σήμερα

Ο Mark Weiser το 1991 στο άρθρο του «Ο υπολογιστής του 21ου αιώνα» [1] εισήγαγε ένα νέο τρόπο αντίληψης και χρήσης των υπολογιστών. Σύμφωνα με αυτόν, ο υπολογιστής γενικής χρήσης με τον τρόπο που τον γνωρίζουμε και χρησιμοποιούμε αντικαθίσταται από ένα σύνολο μικρότερων και εξειδικευμένων υπολογιστών οι οποίοι είναι ενσωματωμένοι σε αντικείμενα ή συσκευές καθημερινής χρήσης. Οι υπολογιστές αυτοί δεν λειτουργούν αυτόνομα αλλά έχουν τη δυνατότητα να δημιουργούν ένα δίκτυο και να επικοινωνούν. Στόχος είναι να μας βοηθούν στην εξυπηρέτηση καθημερινών εργασιών δίχως όμως να απαιτούν τον τρόπο εργασίας που έχουμε μάθει με το μοντέλο του σταθερού υπολογιστή.

Ο Weiser δίνει μεγάλη έμφαση στην έννοια της αόρατης τεχνολογίας με την έννοια ότι πρέπει να υπάρχει εκεί πάντοτε χωρίς απαραίτητα να την βλέπουμε και απλά να την χρησιμοποιούμε. Με την μετακίνηση της τεχνολογίας στο παρασκήνιο, ο χρήστης συγκεντρώνεται μόνο στην εργασία του που έχει να φέρει σε πέρας. Αντίστοιχα παραδείγματα αόρατων τεχνολογία που έχουν αφομοιωθεί με τον τρόπο στην καθημερινή ζωή όπως είναι η γραφή ή ο ηλεκτρισμός και η εφαρμογές του. Οποιοσδήποτε μπορεί να χρησιμοποιήσει μια ηλεκτρική συσκευή –εφόσον αντιλαμβάνεται τους κινδύνους και είναι σε θέση να τηρήσει τις βασικές αρχές ασφαλείας. Πόσοι όμως μπορούν ακόμα και σήμερα, μετά από 25 χρόνια από την εργασία του Weiser, να γράψουν ένα κείμενο σε ηλεκτρονική μορφή με την ίδια ευκολία που γράφουν ένα κείμενο στο χαρτί; Σίγουρα τα πράγματα έχουν βελτιωθεί σε σχέση με την εποχή του Weiser και αυτό φαίνεται στην καθημερινότητά μας όπου πολλές εργασίες μας γίνονται για παράδειγμα μέσω έξυπνων κινητών τηλεφώνων. Ακόμα κι ένας μεσήλικας που ενδεχομένως να μην

έχει χρησιμοποιήσει στη ζωή του σταθερό υπολογιστή, μέσω του έξυπνου κινητού μπορεί σήμερα με σχετική ευκολία να γράψει ένα μήνυμα κειμένου, να δει την πρόγνωση του καιρού, ακόμα και να μοιραστεί φωτογραφίες που βγάζει με αυτό με φίλους και οικεία πρόσωπα μέσω εφαρμογών επικοινωνίας (πχ Viber) ή ακόμα και μέσω κοινωνικής δικτύωσης (βλ. Facebook).

Ένα νέο πεδίο έρευνας δημιουργήθηκε που ονομάστηκε “Διάχυτος υπολογισμός" (Ubiquitous Computing) και βασίστηκε στις αρχές που οραματίστηκε ο Weiser. Τα αποτελέσματα του είναι ορατά σε σημαντικό βαθμό στις μέρες μας όπως είδαμε και με το παράδειγμα του έξυπνου κινητού. Σημαντικό κομμάτι στο οποίο βασίζονται οι εφαρμογές του διάχυτου υπολογισμού είναι το IoT (Internet of Things) το οποίο περιγράφεται αναλυτικότερα στο επόμενο κεφάλαιο.

2. Internet of Things

Το διαδίκτυο των πραγμάτων (Internet of Things) ή για συντομία IoT, είναι από τους πιο δημοφιλείς όρους στις μέρες και αναφέρονται σε αυτόν ειδικοί και μη χαρακτηρίζοντας το ως την εξέλιξη του Internet.

Πρόκειται για ένα δίκτυο καθημερινών αντικειμένων τα οποία βρίσκονται σε σύνδεση μεταξύ τους και ανταλλάσουν πληροφορίες. Στον όρο «πράγματα» μπορούν να συμπεριλαμβάνονται συσκευές, αυτοκίνητα, έπιπλα ή ακόμα και κτίρια τα οποία φέρουν τον σχετικό ηλεκτρονικό εξοπλισμό που τους επιτρέπει να έχουν επαυξημένες δυνατότητες:

1. Έχουν ταυτότητα μέσω κάποιας σχετικής τεχνολογίας ταυτοποίησης
2. Μπορούν να αντιλαμβάνονται τιμές του περιβάλλοντος μέσω αισθητήρων που φέρουν
3. Εχουν δικτυακές δυνατότητες μέσω σχετικού ηλεκτρονικού εξοπλισμού και μπορούν να επικοινωνούν με άλλες συσκευές του χώρου χρησιμοποιώντας υπάρχουσες τεχνολογίες δικτύωσης

Η βασική ιδιότητα ενός τέτοιου δικτύου είναι ότι επιτρέπει την παρακολούθηση (monitoring) συσκευών από μικρή ή μεγάλη απόσταση μέσα από την υπάρχουσα διαδικτυακή υποδομή δίνοντας έτσι τη δυνατότητα για αμεσότερη ενσωμάτωση στοιχείων του πραγματικού κόσμου στον κόσμο των υπολογιστών.

Σε εκτίμηση της Cisco από το 2011 εκτιμάται ότι το 2020 το IoT θα διασυνδέει σχεδόν 50 δις αντικείμενα [2]. Στον παρακάτω πίνακα φαίνεται ο ρυθμός αύξησης των συνδεδεμένων συσκευών ανά κάτοικο εως σήμερα και ποια αναμένεται να είναι η αντίστοιχη εικόνα το 2020. Με βάση τους ρυθμούς αύξησης που

προκύπτουν, φαίνεται ότι περίπου στο 2009 οι συσκευές ξεπέρασαν τον πληθυσμό της γης. Εξέλιξη λογική αν λάβουμε υπόψη τη διάδοση των έξυπνων συσκευών όπως έξυπνα τηλέφωνα και tablets.

Έτος	**2003**	**2010**	**2015**	**2020**
Παγκόσμιος πληθυσμός	6.3 δις	6.8 δις	7.2 δις	7.6 δις
Συνδεδεμένες συσκευές	500 εκατ.	12.5 δις	25 δις	50 δις
Συνδεδεμένες συσκευές ανά κάτοικο	0.08	1.84	3.47	6.58

Πίνακας 1: Εξέλιξη αριθμού συσκευών ανά κάτοικο

2.1 Τεχνολογίες ταυτοποίησης

Ενα θέμα που αντιμετωπίζεται στο πεδίο του IoT είναι πως μπορεί να δοθεί μοναδική ταυτότητα σε μια συσκευή.

1. RFID *ταυτοποίηση μέσω ραδιοσυχνοτήτων* (**Radio Frequency Identification**). Στη δεκαετία του 2000, RFID ήταν η κυρίαρχη τεχνολογία. Πρόκειται για ένα πολύ μικρό ηλεκτρονικό κύκλωμα που μπορεί να αποθηκεύσει μερικά bytes πληροφορίας που μπορεί να διαβαστεί από κάποια συσκευή αναγνώστη που βρίσκεται σε κοντινή απόσταση.
2. Επικοινωνία κοντινού πεδίου NFC (near field communication). Η τεχνολογία NFC έχει γίνει κοινή σε smart phones από αρχές της δεκαετίας του 2010, με χρήσεις όπως η ανάγνωση NFC ετικέτες ή για την πρόσβαση στα μέσα μαζικής μεταφοράς αλλά και γρήγορης πληρωμής σε καταστήματα
3. Οπτικές ετικέτες. - Αυτό χρησιμοποιείται για χαμηλό κόστος tagging. Συνήθως εννοούμε το QR Code που είναι μια παραλλαγή του ραβδωρού κώδικα (QR code) Ένα κινητό (έξυπνο) τηλέφωνο μπορεί μέσω της κάμερας

του να αποκωδικοποιήσει QR code χρησιμοποιώντας τεχνικές επεξεργασίας εικόνας.

4. Bluetooth χαμηλής ενέργειας (BLE). Αυτό είναι ένα από τα τελευταία επιτεύγματα της τεχνολογίας. Όλα τα νέα smart phones έχουν BLE λογισμικό.

2.2 Εφαρμογές

Το IoT αποτελεί μια πλατφόρμα μέσω της οποίας καθίσταται εφικτή η επικοινωνία μεταξύ μέγαλου εύρους συσκευών και η πρόσβαση σε αυτές. Οι δυνατότητες έτσι για δημιουργία νέων πεδίων εφαρμογών είναι πρακτικά άπειρες. Αναφέρονται εδώ ορισμένα πεδία που αποτελούν κυρίως πεδίο έρευνας αλλά και σε αρκετές περιπτώσεις έχουν υλοποιηθεί και χρησιμοποιούνται σε πολλά σημεία του πλανήτη.

2.2.1 Έξυπνες πόλεις

Ο όρος *έξυπνη πόλη* (smart city) χρησιμοποιείται για να περιγράψει πως οι τεχνολογίες Πληροφορικής κι Επικοινωνιών (ΤΠΕ) μπορούν να συμβάλλουν στην λειτουργία μιας πόλης ενισχύοντας την αποτελεσματικότητά της και προσφέροντας λύσεις για την αντιμετώπιση των ανισοτήτων, του κοινωνικού αποκλεισμού και τη βελτίωση του φυσικού περιβάλλοντος σε αυτές. Πως δηλαδή με τη χρήση της τεχνολογίας μπορούν να αντιμετωπιστούν επιτυχώς τα προβλήματα που αντιμετωπίζει καθημερινά ο κάτοικος ενός αστικού περιβάλλοντος στις μέρες μας [3].

Μερικά παραδείγματα:

- **Διαχείριση αποβλήτων**: Με τη χρήση «έξυπνων» κάδων που θα έδιναν στοιχεία μέσω IoT για την κατάστασή τους, θα μπορούσε να βελτιστοποιηθεί η περισυλλογή των απορρριμάτων από τα

απορριματοφόρα ώστε να γίνεται σε συντομότερο χρονικό διάστημα με σημαντικό περιβαλλοντικό αλλά και οικονομικό όφελος.

- **Παρακολούθηση κατανάλωσης ενέργειας**: Μέσω του αστικού IoT είναι ευκολότερη η παρακολούθηση της ενέργειας που απαιτείται (ή και σπαταλάται) για τις κοινόχρηστες υπηρεσίες (φωτισμός δρόμων και κτιρίων, κλιματισμός και θέρμανση δημοσίων κτιρίων κλπ), δίνοντας εικόνα στους πολίτες για το πως αξιοποιούνται οι φόροι αλλά και εντοπίζοντας σημεία που χρήζουν εξοικονόμησης.
- **Έξυπνη στάθμευση:** Η χρήση αισθητήρων στις θέσεις στάθμευσης σε συνδυασμό με τη χρήση «έξυπνης» τεχνολογίας στα αυτοκίνητα μπορεί να μειώσει σημαντικά το χρόνο αλλά και την ενέργεια που απαιτείται για την εύρεση στάθμευσης ενός από τα σημαντικότερα προβλήματα που αντιμετωπίζει ο κάτοικος μιας πόλης.
- **Αναφορά κι εντοπισμός βλαβών:** Με τη χρήση IoT μπορεί να γίνει εύκολα ο εντοπισμός μιας βλάβης σε αστικό περιβάλλον και να δρομολογηθεί η επισκευή της. Η διαδικασία μπορεί να γίνεται αυτόματα μεσω της υποδομής του συγκεκριμένου σημείο (πχ κολωνάκι που φέρει αισθητήρα κλίσης και δίνει την πληροοφορία αυτή μέσω του IoT της πολης οπότε ένα χτύπημα από αυτοκίνητο γίνεται άμεσο αντιληπτό από τις αρχές) αλλά μπορεί να ενθαρρύνεται κι η συμμετοχή των πολιτών οι οποίοι με χρήση ειδικής εφαρμογής στο κινητό τους θα μπορούν να αναφέρουν με συντομία αλλά πολύ αποτελεσματικά μια βλάβη στις δημοτικές αρχές.

Ο κατάλογος είναι βέβαια μακρύς και δεν περιορίζεται στα παραπάνω παραδείγματα.

2.2.2 Έξυπνο σπίτι

Το έξυπνο σπίτι κατ' αντιστοιχία με τη έξυπνη πόλη αντιμετωπίζει μέσω της τεχνολογίας προβλήματα των ενοίκων. Με την «εξυπνοποίηση» οικιακών συσκευών και τη διασύνδεσή τους μέσω IoT το σπίτι προσαρμόζεται στις ανάγκες

των ενοίκων του. Ενα έξυπνο ψυγείο είναι σε θέση να γνωρίζει τα περιεχόμενά του, ποια εξαντλούνται ή ποια λήγουν και μπορεί να υπενθυμίζει την ανάγκη για προμήθειες ή ακόμα και το ίδιο μέσω του IoT να προβαίνει σε παραγγελίες από το ηλεκτρονικό σούπερ μάρκετ. Ο φωτισμός προσαρμόζεται στην ώρα αλλά και στις προτιμήσεις των ενοίκων. Σημαντικές ειδοποιήσεις (πχ σημαντικό SMS ή εισερχόμενη κλήση) μπορούν να προβάλλονται στην κατάλληλη συσκευή ανάλογα με τη θέση του ενοίκου μέσα στο σπίτι.

2.2.3 Υποβοήθηση τρίτης ηλικίας

Επειδή ο μέσος όρος ηλικίας των ευρωπαϊκού πληθυσμού ανεβαίνει, η ανεργία των νέων επίσης, το ασφαλιστικό σύστημα δεν είναι σε θέση να καλύψει τις αυξημένες ανάγκες ιατροφαρμακευτικής περίθαλψης της τρίτης ηλικίας με τα παραδοσιακά μέσα (πχ νοσηλεία). Με τη χρήση ΤΠΕ και IoT μπορεί ο ηλικιωμένος να συνεχίσει να ζει στο σπίτι του και να απολαμβάνει τη καθημερινή φροντίδα και παρακολούθηση από ιατρικές υπηρεσίες χωρίς να απαιτείται να μετακινείται εκτός σπιτιού. Για παράδειγμα, η μέτρηση της αρτηριακής πίεσης μπορεί να αποστέλεται απ' ευθείας στο γιατρό, η θέση και η κινητικότητα του ατόμου στο σπίτι επίσης παρακολουθείται και μπορεί να επισημάνει επείγουσες καταστάσεις. Το πολύ σημαντικό είναι ότι μειώνει σημαντικά το οικονομικό κόστος αυτής της φροντίδας και σε μια εποχή όπου η παγκόσμια οικονομική κρίση δεν αφήνει και σημαντικά περιθώρια.

2.3 Κίνδυνοι

Με την έκθεση όλων των συσκευών αυτών σε «κοινή θέα» μέσω του IoT δημιουργούνται ορισμένα θέματα που δεν υπήρχαν ως τώρα και πρέπει να αντιμετωπιστούν.

Οι εταιρίες θα έχουν να αντιμετωπίσουν το θέμα της ασφάλειας αλλά και τον τεράστιο όγκο δεδομένων που θα συλλέγεται. Θα πρέπει να αναπτυχθούν ασφαλή, αξιόπιστα και ενεργειακά αποδοτικούς τρόποι αποθήκευσης και προστασίας των δεδομένων που θα παράγουν δισεκατομμύρια συσκευές. Λογικό

είναι μέσα από την συνδυασμένη αξιοποίηση τέτοιων δεδομένων να μπορούν να παραβιαστούν προσωπικά δεδομένα οπότε προκύπτουν σοβαρά θέματα.

Υπάρχουν και πολλά ακόμα ζητήματα ασφαλείας, καθώς το καταρχήν σημαντικό στοιχείο όλου αυτού, οι αισθητήρες, μιας και οι περισσότεροι από τους αυτούς δεν έχουν την δυνατότητα να δημιουργήσουν κρυπτογραφημένη σύνδεση, επειδή η χρήση πόρων είναι υπερβολική και έχει συνέπεια στην φυσική αυτονομία της συσκευής.

Υπάρχει ανάγκη βελτιστοποίησης των υπολογιστικών πόρων/ενέργειας από αισθητήρες και αντικείμενα, υλοποίηση μέτρων για εξασφάλιση εμπιστευτικότητας/ακεραιότητας/διαθεσιμότητας, εύρεση λύσεων στα διαφορετικά επίπεδα επεξεργασίας καθώς υπάρχει δυσκολία στον συντονισμό εμπλεκομένων μερών και η εύρεση λύσης στην ύπαρξη τρωτών σημείων.

2.4 Πρωτόκολλα

Η επικοινωνία μεταξύ συσκευών στο IoT απαιτεί τη χρήση πρωτοκόλλων. Υπάρχουν βέβαια τα κλασικά πρωτόκολλα TCP/IP απλά λόγω της φύσεως των συσκευών (χαμηλή υπολογιστική ισχύς, περιορισμένοι ενεργειακοί πόροι, διακοπτόμενη συχνά σύνδεση), έχουν αναπτυχθεί πρωτόκολλα τα οποία ανταποκρίνονται καλύτερα σε ένα τέτοιο περιβάλλον. Στη συνέχεια περιγράφονται τρία από τα γνωστότερα σχετικά πρωτόκολλα.

2.4.1 MQTT

Δεν είναι το παραδοσιακό client/server πρωτόκολλο όπου ο client ζητάει κάτι από το server και αυτός του απαντά. Υλοποιείται σε λογική publisher/subscriber. Ο client εκδηλώνει το ενδιαφέρον του (subscriber) για κάποια θέματα και ο server του απαντά μόνο όταν υπάρχει κάτι διαθέσιμο το οποίο έχει προέλθει από άλλον client (publisher). Η δημοφιλής εφαρμογή Facebook Messenger που χρησιμοποιείται από εκατομμύρια χρήστες στον κόσμο είναι υλοποιημένη πάνω

στο πρωτόκολλο αυτό. Υπάρχουν και παραλλαγές του όπως το MQTT-SN που χρησιμοποιείται για δίκτυα αισθητήρων (Sensor Networks).

Λόγω του ότι κεντρικό θέμα αυτής εργασίας ειναι το MQTT, αυτό περιγράφεται αναλυτικά στο κεφάλαιο 3.

2.4.2 CoAP

Το CoAP προέρχεται από τις λέξεις Constrained Application Protocol. Πρόκειται για απλοποιημένη μορφή του HTTP οπότε κι έχει πολλές ομοιότητες με αυτό. Λόγω της σχεδίασής του αυτής, μπορεί εύκολα να μετατραπεί σε κανονικό HTTP, γεγονός που επιτρέπει την εύκολη και διαφανή ενσωμάτωσή του σε περιβάλλον παγκοσμίου ιστού.

2.4.3 AMQP

Το όνομά του προέρχεται από τις λέξεις Advanced Message Queuing Protocol. Σχεδιάστηκε ως ανοικτό σύστημα για να αντικαταστήσει κλειστά εταιρικά (proprietary) messaging συστήματα. Έχει πολλές δυνατότητες και παρέχει πολλές δυνατότητες παραμετροποίησης. Μεταφέρει δεδομένα σε δυαδική αναπαράσταση (binary wire protocol) και λειτουργεί με διαφάνεια και συνδέει συστήματα διαφορετικών κατασκευαστών με επιτυχία. Για το λόγο αυτό έχει γνωρίσει αποδοχή περισσότερη από άλλα σχετικά πρωτόκολλα και χρησιμοποιείται από εταιρίες κολοσσούς για ανταλλαγή μηνυμάτων. Η JP Morgan το χρησιμοποίησε για να διεκπεραιώνει 1 δις μηνύματα την ημέρα ενώ χρησιμοποιείται από τη NASA και τη Google.

3. MQTT

Το πρωτόκολλο MQTT εφευρέθηκε το 1999 από τους Andy Stanford-Clark και Arlen Nipper των IBM και Cirrus Link αντίστοιχα. Ο στόχος ήταν να δημιουργηθεί ένα πρωτόκολλο επικοινωνίας χαμηλών απαιτήσεων σε ενέργεια και δικτυακό εύρος (bandwidth) ώστε να παρακολουθείται και να συντονίζεται η λειτουργία αγωγών πετρελαίου δια μέσου δορυφόρου [3].

Τα αρχικά MQTT προέρχονται από τα αρχικά των λέξεων MQ Telemetry Transport. Το MQ προέρχεται από το MQ Series, ένα προϊόν που αναπτύχθηκε από την IBM και υποστηρίζει και το MQTT. Παρόλο που εσφαλμένα στο διαδίκτυο χαρακτηρίζεται και ως message queue protocol, δεν υπάρχουν εδώ ουρές (queues) όπως σε άλλα παραδοσιακά συστήματα message queuing.

Οι αρχικές εκδόσεις του πρωτοκόλλου χρησιμοποιήθηκαν εσωτερικά από την IBM. Το 2010 η τρέχουσα έκδοση 3.1 δόθηκε ελεύθερα στο κοινό με άδεια royaltee-free ενώ η έκδοση 3.1.1 έγινε standard από τον οργανισμό OASIS τον Οκτώβριο του 2014 ([4], [5]).

Οι βασικές αρχές του πρωτοκόλλου είναι οι παρακάτω:

- Απλό να υλοποιηθεί
- Παρέχει επίπεδα ποιότητας της υπηρεσίας παράδοσης των μηνυμάτων (Quality of Service)
- Αντικειμενικά χαμηλές απαιτήσεις σε υπολογιστική ισχύ και δικτυακό εύρος
- Ανεξάρτητο του είδους των δεδομένων που μεταφέρονται (data agnostic)
- Continuous Session Awareness

3.1 Το πρότυπο publish/subscribe

Η φιλοσοφία του MQTT διαφέρει από την κλασική προσέγγιση πελάτη/εξυπηρετητή (client/server) όπου ένας πελάτης επικοινωνεί απευθείας με τον εξυπηρετητή και αντίστροφα. Εδω χρησιμοποιείται το πρότυπο (pattern) publish/subscribe (εκδότης/συνδρομητής στα ελληνικά)

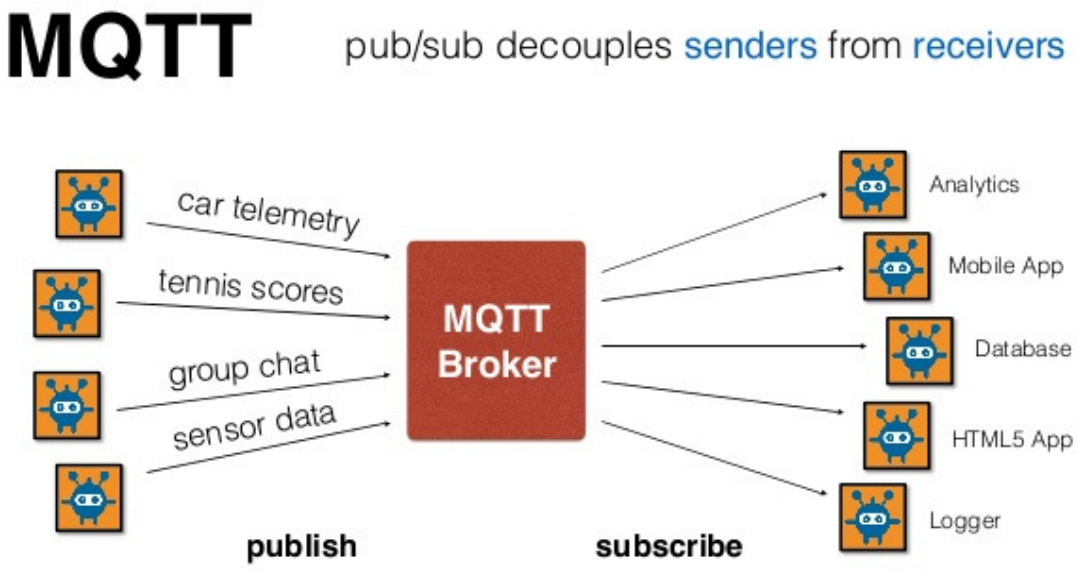
Οι οντότητες που επικοινωνούν είναι είτε εκδότες (**publishers**), είτε συνδρομητές (**subscribers**) ή και τα δύο. Η επικοινωνία γίνεται όταν οι εκδότες παράγουν μηνύματα τα οποία λαμβάνουν (καταναλώνουν) οι συνδρομητές.

Κάθε μήνυμα ανήκει σε ένα θέμα (**topic**). Ένας συνδρομητής εγγράφεται σε ένα ή περισσότερα θέματα που τον ενδιαφέρουν και μόνο μηνύματα αυτών των θεμάτων λαμβάνει. Αντίστοιχα ο εκδότης που παράγει ένα μήνυμα, το κατηγοριοποιεί σε ένα θέμα οπότε μόνο οι συνδρομητές αυτού του θέματος το λαμβάνουν.

Υπάρχει μια ενδιάμεση οντότητα που ονομάζεται **broker** και που γνωρίζουν τόσο οι εκδότες όσο και οι συνδρομητές και κάνει τα παρακάτω σε γενικές γραμμές:

- Εχει άποψη όλων των συνδρομητών του συστήματος
- Λαμβάνει τα μηνύματα από τους εκδότες και τα προωθεί στους ενδιαφερόμενους συνδρομητές

Από τα παραπάνω είναι φανερό ότι τελικοί αποδέκτες της επικοινωνίας (εκδότες και συνδρομητές) το μόνο που πρέπει να γνωρίζουν για να επικοινωνήσουν είναι ο broker και δεν απαιτείται να γνωρίζει ο ένας τον άλλον (για παράδειγμα, δεν χρειάζεται να γνωρίζει την διεύθυνση IP ή το port του). Επιπλέον η επικοινωνία είναι ασύγχρονη αφού δεν απαιτείται να λειτουργούν και τα δύο μέρη κατά την επικοινωνία (ή να έχουν σύνδεση με τον broker), ούτε το ένα μέρος μπλοκάρει τη λειτουργία του άλλου κατά την επικοινωνία.



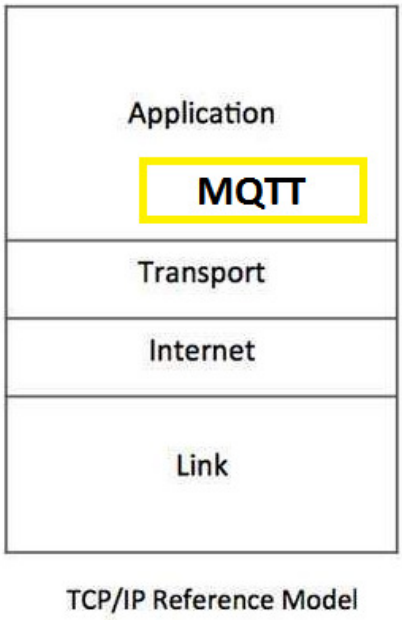
Εικόνα 1: Παράδειγμα συστήματος MQTT. Στο κέντρο απεικονίζεται ο broker, αριστερά οι clients που είναι εκδότες και δεξιά οι clients συνδρομητές. Όλοι οι clients επικοινωνούν έμμεσα και μόνο μέσω του broker.

3.2 Σύνδεση client και broker

3.2.1 Clients

Με όρους client/server, client στην περίπτωση του MQTT είναι τόσο οι εκδότες όσο και οι συνδρομητές. Μάλιστα ένας client μπορεί να είναι ταυτόχρονα και εκδότης και συνδρομητής. Server είναι ο broker και μόνο με αυτόν συνδέονται οι clients.

Ένας MQTT client μπορεί να είναι οποιοδήποτε είδος συσκευής (από έναν απλό micro controller ενδεχομένως σε ένα περιβάλλον με ασταθή συνδεσιμότητα έως και ένα υπερσύγχρονο και πανίσχυρο μηχάνημα server) η οποία φέρει μια βιβλιοθήκη υλοποίησης του MQTT και μπορεί να συνδέεται μέσω οποιουδήποτε δικτύου με τον broker. Αν μιλάμε για περιβάλλον internet (TCP/IP) τότε η βιβλιοθήκη MQTT υλοποιείται στο επίπεδο εφαρμογής όπως και άλλα παρόμοια πρωτόκολλα πχ HTTP (Εικόνα 2).



Εικόνα 2: Το πρωτόκολλο MQTT και η θέση του στην ιεραρχία επιπέδων του TCP/IP

3.2.2 Broker

Αποτελεί την καρδιά του συστήματος αφού λαμβάνει όλα τα μηνύματα, τα φιλτράρει και αποφασίζει σε ποιους συνδρομητές θα τα προωθήσει. Ταυτοποιεί επίσης όλους τους clients μέσω μεθόδων ταυτοποίησης (authentication method) ενώ παρακολουθεί όλες τις συνδέσεις με τους συνδρομητές και μπορεί να προωθήσει μηνύματα τα οποία έχασε κάποιος συνδρομητής επειδή ήταν προσωρινά εκτός σύνδεσης ή λειτουργίας.

Ο broker όντας λοιπόν ο server του όλου συστήματος θα πρέπει να είναι ισχυρός, να επιτρέπει εύκολη κλιμάκωση (scalability) ώστε να μπορεί να αντεπεξέλθει στο φόρτο εργασίας και θα πρέπει να μπορεί να ανακάμψει μετά από κάποια σφάλμα (fault-tolerant).

3.2.3 Εγκατάσταση σύνδεσης

Η σύνδεση μεταξύ ενός client και του broker γίνεται με την αποστολή ενός μηνύματος **CONNECT**. Εφόσον είναι εφικτή η σύνδεση, ο broker απαντά με ένα μήνυμα **CONNACK** (Connection Acknowledgement). Οι βασικές πληροφορίες που περιέχει ένα μήνυμα CONNECT φαίνονται στον πίνακα 2.

CONNECT		
Πεδίο	**Υποχρεωτικό**	**Παράδειγμα τιμής**
clientId	Ναι	"a34rut222"
cleanSession	Ναι	"true"
username	'Οχι	"user1"
password	'Οχι	"$rrfIff$%_A"
lastWillTopic	'Οχι	"/user1/will"
lastWillQoS	'Οχι	2
lastWillMessage	'Οχι	"bye bye"
keepAlive	Ναι	30

Πίνακας 2: Οι βασικές πληροφορίες μηνύματος CONNECT[1]

ClientId

Το αναγνωριστικό του κάθε MQTT client το οποίο θα πρέπει να είναι μοναδικό ανά broker.

CleanSession

Καθορίζει αν θα φυλάσσονται τα μηνύματα τα οποία για κάποιο λόγο δεν παραδόθηκαν στον client όταν έχει κάνει subscription με QoS > 0.

Username/Password

Επιτρέπουν την ταυτοποίηση του client σε περίπτωση που κάτι τέτοιο απαιτείται. Αυτά όμως δεν κρυπτογραφούνται αλλά στέλνονται ως έχουν επομένως ενδείκνυται να χρησιμοποιούνται μόνο σε ασφαλείς συνδέσεις.

Will message

Αυτή η λειτουργία επιτρέπει να ενημερωθούν οι ενδιαφερόμενοι clients ότι ένας client αποσυνδέθηκε ξαφνικά από το σύστημα. Πρόκειται για μήνυμα που παραδίδει ο broker στους ενδιαφερόμενους clients όταν αντιληφθεί ότι ο client εκδότης έχει πάψει να λειτουργεί.

Keep alive

Πρόκειται για το χρονικό διάστημα που μεσολαβεί για να στείλει ο client ένα μήνυμα τύπου **PING Request** στον broker ο οποίος απαντά με μήνυμα **PING**

[1] Η δομή του κάθε μηνύματος (ή πακέτου) δίνεται στην εργασία αυτή περιγραφικά και όχι με ακρίβεια bit και byte όπως συμβαίνει στις ακριβείς προδιαγραφές του πρωτοκόλλου MQTT που δίνει ο οργανισμός πιστοποίησης [5].

Response. Ο μηχανισμός αυτός χρησιμοποιείται ώστε να γίνει αντιληπτό και από τα δύο μέρη ότι η σύνδεση έχει χαθεί.

Όταν ο broker λάβει ένα μήνυμα **CONNECT**, απαντά με ένα μήνυμα τύπου **CONNACK** το οποίο περιλαμβάνει τις πληροφορίες που φαίνονται στον πίνακα 3.

CONNACK		
Πεδίο	**Υποχρεωτικό**	**Παράδειγμα τιμής**
`sessionPresent`	Ναι	True
`returnCode`	Ναι	0

Πίνακας 3: Η δομή ενός μηνύματος CONNACK

sessionPresent

Δηλώνει αν υπάρχει ήδη συνεδρία από προηγούμενη σύνδεση. Αν όμως ο client στο CONNECT μήνυμά του έχει δώσει true ως τιμή για το cleanSession, τότε αυτή τιμή θα είναι πάντα false. Σε αντίθετη περίπτωση η τιμή της εξαρτάται από το αν πραγματικά υπάρχει από πριν συνεδρία για τον συγκεκριμένο client στον broker.

returnCode

Πρόκειται για κωδικό σφάλματος που δίνει την κατάσταση της αίτησης σύνδεσης. Αν η αίτηση έγινε αποδεκτή τότε ο κωδικός έχει τιμή 0, σε διαφορετική περίπτωση επιστρέφεται σχετικός κωδικός που δίνει και το είδος του σφάλματος σύμφωνα με τις προδιαγραφές του πρωτοκόλλου [5].

3.3 Βασικές ενέργειες πρωτοκόλλου

Με βάση την περιγραφή του μοτίβου publish/subscribe και την χρήση του στο MQTT που περιγράφηκε στην προηγούμενη ενότητα, σε αυτή την ενότητα θα αναλυθούν με περισσότερες τεχνικές λεπτομέρειες οι βασικές ενέργειες του πρωτοκόλλου που υλοποιούν τα ενδιαφερόμενα μέρη και είναι οι παρακάτω:

- Publish (Δημοσίευση)
- Subscribe (εγγραφή συνδρομής)
- Unsubscribe (ακύρωση συνδρομής)

3.3.1 Publish

Είναι η ενέργεια που πραγματοποιείται από έναν client-εκδότη που θέλει να στείλει ένα μήνυμα στους ενδιαφερόμενους clients-συνδρομητές. Το μόνο που ενδιαφέρει τον εκδότη είναι να φθάσει το μήνυμα στον broker. Δεν μπορεί να πληροφορηθεί για το αν και πόσοι clients-συνδρομητές πήραν το μήνυμα.

Μόλις ένας client συνδεθεί επιτυχώς στον broker μετά την ανταλλαγή μηνυμάτων CONNECT και CONNACK, είναι σε θέση να δημοσιεύσει μηνύματα τα οποία στέλνει στον broker και από εκεί προωθούνται προς τους ενδιαφερόμενους clients.

Ένα μήνυμα δημοσιεύεται με ένα ειδικό MQTT πακέτο τύπου **PUBLISH** το οποίο έχει τη βασική δομή που φαίνεται στον πίνακα 4 και μέρος του οποίου είναι και το μήνυμα προς μεταφορά.

PUBLISH		
Πεδίο	**Υποχρεωτικό**	**Παράδειγμα τιμής**
packetId	Ναι. Εχει την τιμή μηδέν όταν qos=0	5236
topicName	Ναι	"cars/location/100"
qos	Ναι	1
retainFlag	Ναι	true
payload	Ναι	“lng: 4.234345342, lat: 4.3457645566”
duplicateFlag	Ναι	0

Πίνακας 4: Η δομή του μηνύματος PUBLISH

packetId

Μια μοναδική τιμή που χαρακτηρίζει ένα μήνυμα. Χρησιμοποιείται στις περιπτώσεις που το QoS είναι μεγαλύτερο του μηδεν (1 ή 2) και άρα ελέγχεται η πιθανότητα ένα μήνυμα να μη φθάσει στον αποδέκτη.

duplicateFlag

Δηλώνει ότι το συγκεκριμένο μήνυμα έχει σταλεί πάλι άρα υπάρχει πιθανότητα ο client να το έχει λάβει ξανά.

payload

Είναι το μήνυμα που στέλνεται από τον publisher στους subscribers.

retainFlag

Εαν κάποιο μήνυμα φθάσει στον broker με αυτή τη flag ενεργοποιημένη, τότε όποιος νέος subscriber εγγραφεί στο συγκεκριμένο topic, θα λάβει αυτό το μήνυμα.

3.3.2 Subscribe

Με την ενέργεια αυτή ο client εγγράφεται σε ένα topic ή περισσότερα topics και ζητά να λαμβάνει μηνύματα που δημοσιεύονται από τους clients-εκδότες σε αυτά. Για το σκοπό αυτό αποστέλεται στον broker μήνυμα τύπου SUBSCRIBE με τις πληροφορίες που φαίνονται στον πίνακα 5. Υπάρχουν N ζεύγη πεδίων (Topic_i, QoS_i) με i=1…N, ένα για κάθε topic που εγγράφεται ο client. Εκτός από το όνομα του topic, ο client ορίζει και με τι επίπεδο Quality of Service θέλει να λαμβάνει τα αντίστοιχα μηνύματα από τον broker. Το Quality of Service περιγράφεται αναλυτικότερα στην παράγραφο 3.5.

SUBSCRIBE		
Πεδίο	**Υποχρεωτικό**	**Παράδειγμα τιμής**
packetId	Ναι. Εχει την τιμή μηδέν όταν qos=0	5236
Topic_1		
QoS_1		
Topic_2		
QoS_2		
…		
Topic_N		
QoS_N		

Πίνακας 5: Το μήνυμα SUBSCRIBE έχει μεταβλητό αριθμό πεδίων αφού έχει N ζεύγη πεδίων (Topic_

Αντίστοιχα ο broker για κάθε μήνυμα SUBSCRIBE που λαμβάνει απαντά με μήνυμα SUBACK το οποίο περιλαμβάνει τόσους κωδικούς σφάλματος όσα και τα topics για τα οποία εκδηλώθηκε ενδιαφέρον.

Η δομή του μηνύματος SUBACK έχεις ως εξής:

Κωδικός	**Περιγραφή**
packetId	Ο κωδικός πακέτου του SUBSCRIBE για το οποίο είναι αυτή η απάντηση
Return code 1	Η απάντηση για το 1^{ο} topic

Return code 2	Η απάντηση για το 2ο topic
…	…
Return code N	…

Πίνακας 6: Το μήνυμα SUBACK

Ενώ οι κωδικοί που επιστρέφονται δίνονται στον παρακάτω πίνακα:

Κωδικός	Περιγραφή
0	Επιτυχία – μέγιστο QoS 0
1	Επιτυχία – μέγιστο QoS 1
2	Επιτυχία – μέγιστο QoS 2
128	Αποτυχία

Τα θέματα (topics) περιγράφονται αναλυτικότερα στην παράγραφο 3.4.

3.3.3 Unsubscribe

Όταν ένας client επιθυμεί να μη λαμβάνει πλέον μηνύματα από ένα ή περισσότερα συγκεκριμένα topics, στέλνει ένα μήνυμα τύπου **UNSUBSCRIBE** στον broker το οποίο έχει παρόμοια δομή με το SUBSCRIBE, έχει δηλαδή τους τίτλους των topics για τα οποία δεν ενδιαφέρεται πλέον, απλά δεν έχει τις τιμές για το QoS αφού εδώ δεν χρειάζονται.

UNSUBSCRIBE	
Πεδίο	**Παράδειγμα τιμής**
packetId	Ο κωδικός του πακέτου
Topic_1	
Topic_2	
…	
Topic_N	

Πίνακας 7: Η δομή του UNSUBSCRIBE

Ο broker απαντά με ένα μήνυμα τύπου UNSUBACK που περιέχει μόνο την τιμή packetId του αντίστοιχου μηνύματος UNSUBSCRIBE.

3.4 Θέματα (topics)

Κάθε μήνυμα που στέλνεται στον broker από έναν client-εκδότη ανήκει σε ένα θέμα. Τα θέματα είναι αλφαριθμητικά σε κωδικοποίηση UTF-8 με βάση τα οποία ο

broker είναι σε θέση να γνωρίζει σε ποιους clients-συνδρομητές θα προωθήσει ένα μήνυμα όταν το λάβει.

Μπορούμε να πούμε ότι το θέμα είναι ένα είδος ουράς που δημιουργείται στον broker. Δεν χρειάζεται όμως να δημιουργηθεί εκ των προτέρων αλλά αυτό γίνεται όταν ληφθεί το πρώτο μήνυμα από τον client-εκδότη. Αντίστοιχα δεν χρειάζεται να κάνει κάτι εκ των προτέρων ένας client που επιθυμεί να εγγραφεί σε ένα θέμα. Απλά στέλνει ένα SUBSCRIBE μήνυμα στον broker ακόμα κι αν ο broker δεν έχει λάβει ακόμα κανένα μήνυμα για αυτό το θέμα.

3.4.1 Επίπεδα

Ένα θέμα μπορεί να αποτελείται από ένα ή περισσότερα επίπεδα. Τα επίπεδα ορίζονται στην ονομασία του θέματος και διαχωρίζονται μεταξύ τους με ένα forward slash (/). Για παράδειγμα:

Παράδειγμα 1

```
Home/groundfloor/livingroom/temperature
Home/groundfloor/livingroom/humidity
Home/groundfloor/kitchen/temperature
Home/groundfloor/bathroom/temperature
Home/garage/temperature
```

Όταν ένας client εγγράφεται σε θέμα με τη λειτουργία SUBSCRIBE, μπορεί να εγγραφεί σε ένα θέμα δίνοντας ως ζητούμενο θέμα όλο τον τίτλο του, όπως για παράδειγμα `Home/groundfloor/livingroom/temperature`. Μπορεί όμως να εγγραφεί σε πολλά θέματα ταυτόχρονα με τη χρήση χαρακτήρων **wildcards**.

Υπάρχουν δύο τέτοιοι χαρακτήρες στην περίπτωσή μας: το '+' και το '#'. Το '+' αντικαθιστά ένα μόνο επίπεδο ενώ το '#' ένα ή περισσότερα. Για παράδειγμα, θεωρούμε ότι στον broker φθάνουν μηνύματα για τα θέματα στο παράδειγμα 1. Αν ο client ζητήσει subscription δίνοντας ως θέμα:

```
Home/groundfloor/+/temperature
```

τότε θα εγγραφεί στα εξής θέματα:

```
Home/groundfloor/livingroom/temperature
Home/groundfloor/kitchen/temperature
Home/groundfloor/bathroom/temperature
```

Σε αντίθεση με τον χαρακτήρα '+' που μπορεί να μπει σε οποιοδήποτε σημείο , ο χαρακτήρας '#' τοποθετείται αυστηρά στο τέλος του θέματος που ζητείται και ταιριάζει όλα τα εναπομείναντα επίπεδα. Παραδείγματα με τη χρήση του '#' με βάση και τα προηγούμενα δίνονται στον παρακάτω πίνακα:

Ζητούμενο θέμα	Θέματα που ταιριάζουν
Home/groundfloor/#	Home/groundfloor/livingroom/temperature Home/groundfloor/livingroom/humidity Home/groundfloor/kitchen/temperature Home/groundfloor/bathroom/temperature
Home/#	Όλα τα θέματα: Home/groundfloor/livingroom/temperature Home/groundfloor/livingroom/humidity Home/groundfloor/kitchen/temperature Home/groundfloor/bathroom/temperature Home/garage/temperature

Πίνακας 8: Παραδείγματα χρήσης wildcards για τα θέματα του παραδείγματος 1

Μπορεί να δοθεί μόνο ο χαρακτήρας # ως ζητούμενο topic αλλά αυτό καλό είναι να αποφεύγεται αφού τότε ο client εγγράφεται σε όλα τα θέματα και θα λαμβάνει κάθε μήνυμα που φθάνει στον broker.

Η πολυεπιπεδική δομή των θεμάτων αν και απλή στην υλοποίηση, επιτρέπει μεγάλη ευελιξία στην ιεράρχηση μηνυμάτων ενώ με σωστή σχεδίαση επιτρέπει και την ομαλή επέκταση του συστήματος όταν αυτό χρειαστεί.

3.4.2 Θέματα του συστήματος

Υπάρχει μια κατηγορία θεμάτων μηνύματα των οποίων μπορούν να δώσουν πληροφορίες σχετικά με το σύστημα (broker). Τα θέματα αυτά ξεκινούν με τον χαρακτήρα **$** και είναι ο μόνος χαρακτήρας που δεν επιτρέπεται να

χρησιμοποιήσουμε σε topics που δημιουργούμε. Συνήθως τα θέματα αυτά ξεκινούμε με το πρόθεμα **$SYS** ενώ οι διάφορες υλοποιήσεις των brokers που υπάρχουν υλοποιούν κάποια από αυτά.

Κάποια από τα θέματα που προτείνεται να υλοποιούνται είναι τα εξής [6]:

$SYS/broker/clients/total
$SYS/broker/clients/connected
$SYS/broker/clients/disconnected
$SYS/broker/messages/sent

3.5 Quality of Service (QoS)

Ο όρος quality of service (ποιότητα παρεχόμενης υπηρεσίας) χρησιμοποιείται στα συστήματα επικοινωνίας για να δηλώσει την απόδοση ενός συστήματος όπως την αντιλαμβάνονται οι χρήστες του. Στο MQTT, το QoS level είναι μια συμφωνία μεταξύ των δύο μερών (αποστολέας και παραλήπτης) για το πόσο εγγυημένη είναι η παράδοση ενός μηνύματος στον προορισμό του.

Επίπεδο	Ονομασία	Περιγραφή
0	At most once	Το μήνυμα θα παραδοθεί στον προορισμό του το πολύ μια φορά (μπορεί δηλαδή να μη φθάσει και ποτέ)
1	At least once	Θα παραδοθεί **τουλάχιστον** μια φορά
2	Exactly once	Θα παραδοθεί **ακριβώς** μια φορά

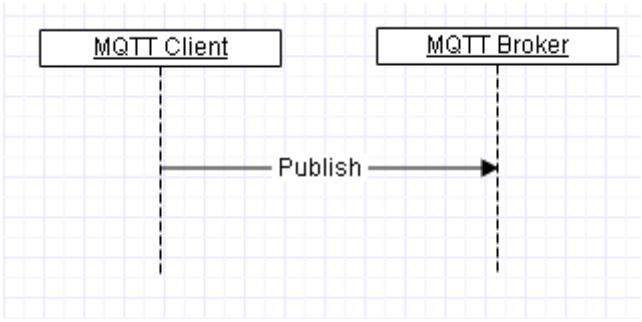
Πίνακας 9: Τα επίπεδα QoS στο MQTT

Όταν ένας client-εκδότης δημοσιεύει ένα μήνυμα, τότε αυτό φθάνει στον προορισμό του (στους clients-συνδρομητές) ως γνωστόν σε δύο βήματα: (α) στον broker και (β) από τον broker στον client-συνδρομητή. Για το βήμα (α), το QoS καθορίζεται από τον publisher και ορίζεται για κάθε μήνυμα (§ 3.3.1). Για το (β), το QoS ορίζεται όταν ο συνδρομητής εγγράφεται στο συγκεκριμένο θέμα (§ 3.3.2).

3.5.1 QoS 0

Ο αποστολέας στέλνει το μήνυμα αλλά δεν είναι σε θέση να γνωρίζει αν έφτασε αφού ο παραλήπτης ακόμα κι αν το λάβει δεν στέλνει επιβεβαίωση. Σχηματικά

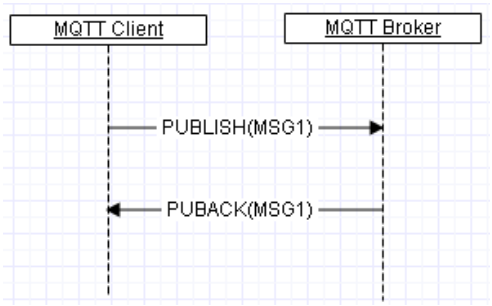
αυτό απεικονίζεται στην Εικόνα 3. Έτσι ο παραλήπτης θα λάβει το μήνυμα το πολύ μία φορά (at least once).



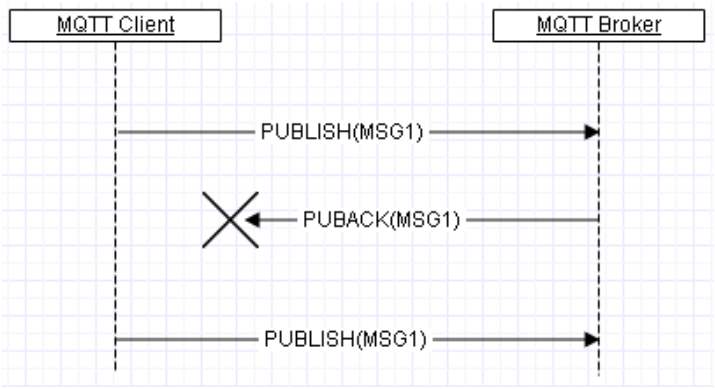
Εικόνα 3: Δημοσίευση μηνύματος με QoS = 0

3.5.2 QoS 1

Στην περίπτωση αυτή για κάθε μήνυμα που λαμβάνει ο broker στέλνει μήνυμα επιβεβαίωσης στον αποστολέα (Εικόνα 4) που σημαίνει ότι το μήνυμα έφθασε στον προορισμό του. Στην περίπτωση όμως που βλέπουμε στην Εικόνα 5, το μήνυμα αρχικά φθάνει στον broker, το PUBACK όμως για κάποιο λόγο δεν φθάνει ποτέ στον αποστολέα. Επειδή όμως ο αποστολέας δεν είναι σε θέση να γνωρίζει αν όντως το μήνυμα έφθασε στον προορισμό του ή όχι, μετά από ένα ορισμένο χρονικό διάστημα (timeout) ξαναστέλνει το ίδιο μήνυμα. Το αποτέλεσμα όμως είναι ότι υπάρχει μεγάλη πιθανότητα ο παραλήπτης να λάβει δύο ίδια μηνύματα, κάτι βέβαια που το γνωρίζει λόγω του επιπέδου QoS που χρησιμοποιείται. Εξ ού και ο χαρακτηρισμός του επιπέδου υπηρεσίας ως at least once (τουλάχιστον μία φορά).



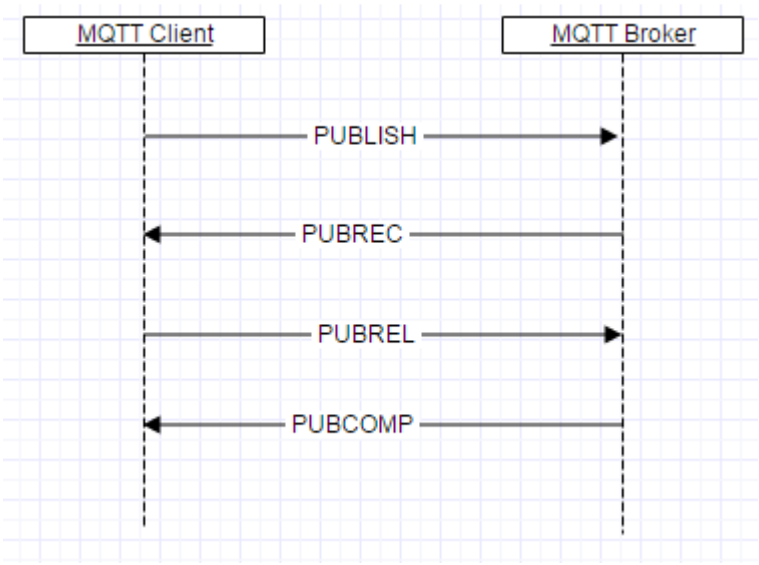
Εικόνα 4: Αποστολή μηνύματος με QoS = 1. Ο broker απαντά με PUBACK επιτυχώς.



Εικόνα 5: QoS = 1. Η επιβεβαίωση PUBACK δεν φθάνει στον client με αποτέλεσμα το αρχικό μήνυμα να σταλεί ξανά.

3.5.3 QoS 2

Το επίπεδο αυτό εγγυάται ότι το μήνυμα θα παραδοθεί **ακριβώς μία φορά** στον παραλήπτη. Αυτό επιτυγχάνεται με μια διπλή επικοινωνία μεταξύ των δύο μερών με συνολικά τέσσερα βήματα (4-step handshake) όπως φαίνεται και στην Εικόνα 6.



Εικόνα 6: Περίπτωση QoS = 2**.** Τα τέσσερα βήματα που απαιτούνται για να εξασφαλιστεί ότι το μήνυμα παραδίδεται ακριβώς μία φορά στον προορισμό του.

Ο broker όταν λάβει το μήνυμα PUBLISH, το κρατάει μέχρι να στείλει το πακέτο PUBCOMP.

PUBREC (Publish Received)

Στέλνεται από τον broker ως επιβεβαίωση για το πακέτο PUBLISH του client και περιέχει ως packet id το αρχικό αναγνωριστικό του μηνύματος PUBLISH. Επομένως

αν ο αποστολέας δεν λάβει PUBREC επαναλαμβάνει την αποστολή PUBLISH. Όταν όμως το λάβει, ξέρει ότι ο broker έλαβε το πακέτο άρα μπορεί να το διαγράψει. Στέλνει τότε ένα πακέτο PUBREL το οποίο και αυτό περιέχει ως packet id το αναγνωριστικό από το αρχικό PUBLISH.

PUBREL (Publish Release)

Όταν λάβει τέτοιο μήνυμα ο broker, διαγράφει από τη μνήμη του ό,τι σχετικό με το αρχικό πακέτο και απαντά με μήνυμα PUBCOMP.

PUBCOMP (Publish Complete)

Ειναι η αναγνώριση του πακέτου PUBREL. Αν ο αποστολέας δεν λάβει PUBCOMP, ξαναστέλνει PUBREL μέχρι τελικώς να λάβει PUBCOMP. Όταν συμβεί αυτό και το δύο μέρη –αποστολέας και δέκτης- ξέρουν ότι το αρχικό μήνυμα παραδόθηκε μία φορά.

Το επίπεδο υπηρεσίας 2 είναι το πιο αργό και απαιτητικό σε πόρους, λογικό βέβαια αφού απαιτεί περισσότερη επικοινωνία αλλά και χρήση μνήμης μέχρι την ολοκλήρωση της ενέργειας.

Επισήμανση: *Στα παραδείγματα που αναφέρθηκαν για το QoS, αναλύεται η επικοινωνία μεταξύ publisher και broker. Το ίδιο επαναλαμβάνεται και στο $2^{ο}$ μέρος της επικοινωνίας, αυτό μεταξύ του broker και του subscriber.*

3.6 Brokers

Υπάρχει πληθώρα διαθέσιμων brokers που μπορούν χρησιμοποιηθούν ακόμα και δωρεάν. Παρακάτω περιγράφονται οι δημοφιλέστεροι εξ' αυτών.

3.6.1 Apache ActiveMQ

http://activemq.apache.org/index.html

Δωρεάν και open-source, υποστηρίζει πολλαπλά πρωτόκολλα (MQTT, OpenWire, STOMP, AMQP). Είναι υλοποιημένος σε Java.

3.6.2 Apache Apollo

http://activemq.apache.org/apollo/

Εξέλιξη του ActiveMQ. Θεωρείται πιο αξιόπιστος και πιο γρήγορος του ActiveMQ καθώς επίσης κι ευκολότερα διαχειρίσιμος και παραμετροποιήσιμος.

3.6.3 IBM Websphere MQ Telemetry

http://www-03.ibm.com/software/products/en/wmq-telemetry

Εμπορική έκδοση που υποστηρίζει πλήρως το MQTT 3.1

3.6.4 IBM MessageSight

http://www-03.ibm.com/software/products/en/iot-messagesight

Έτοιμος server (DMZ appliance) για απαιτητικές εφαρμογές. Υποστηρίζει πληθώρα ειδών clients και πρωτοκόλλων.

3.6.5 Mosquitto

http://mosquitto.org/

Δωρεάν και open-source, υποστηρίζει πλήρως το MQTT. Διατίθεται για πλατφόρμες Windows, Mac και διαφόρων εκδόσεων Linux μεταξύ των οποίων και υλοποίηση για Raspberry Pi.

3.6.6 Emqttd

https://github.com/emqtt/emqttd

Δωρεάν και Open-source broker (Apache Version 2.0 license) με έμφαση στην κλιμακωσιμότητα (scalability) ενώ υποστηρίζει κι άλλα πρωτόκολλα όπως WebSocket, STOMP, SockJS, CoAP.

3.6.7 RabbitMQ

http://www.rabbitmq.com/

Βασικά πρόκειται για AMQP broker υποστηρίζει το MQTT μέσω plugin. Διατίθεται δωρεάν για πλατφόρμες Windows, Mac OS X και Linux.

3.6.8 HiveMQ

http://www.hivemq.com/

Εμπορικός broker εξ' ολοκλήρου προϊόν ανάπτυξης της ομώνυμης εταιρίας. Σχεδιάστηκε με έμφαση στην κλιμακωσιμότητα (scalabity) και επιχειρησιακή ασφάλεια.

Παρέχει και open source SDK plugin για να μπορούν να γράφονται επεκτάσεις του από τρίτους αλλά και να διασυνδέεται με άλλα συστήματα.

Διαθέτει και online server ελεύθερο μέσω του δικτύου για δοκιμές (http://www.hivemq.com/try-out/)

4. Το σύστημα MapTrack

«Τα πράγματα που πρέπει να κάνεις, τα μαθαίνεις κάνοντάς τα»

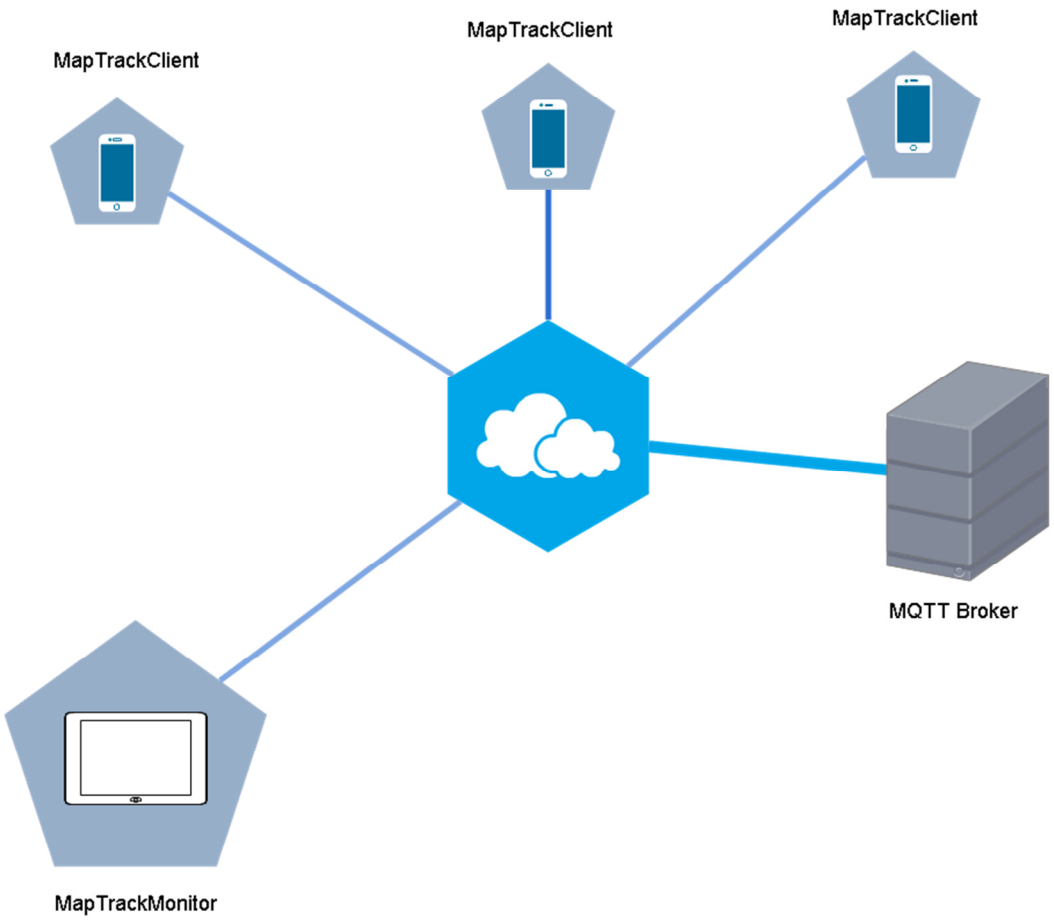
Αριστοτέλης

Για να δοκιμαστεί στην πράξη το πρωτόκολλο MQTT και να γίνει ευκολότερη η διερεύνηση των διαφόρων παραμέτρων του, υλοποιήθηκε το σύστημα MapTrack. Πρόκειται για σύστημα που μπορεί να κατηγοριοποιηθεί στην κατηγορία των εφαρμογών IoT αφού τα δεδομένα που ανταλλάσσονται αφορούν τιμές αισθητήρων (δέκτες GPS) που συλλέγονται από κινητά τηλέφωνα.

Η βασική ιδέα είναι να συγκεντρώνονται μετρήσεις από έξυπνα τηλέφωνα που «τρέχουν» την εφαρμογή, να κωδικοποιούνται και να στέλνονται σε έναν MQTT broker με τέτοιο τρόπο ώστε να λαμβάνονται από τους ενδιαφερόμενους.

Επιλέχθηκε να στέλνονται δεδομένα GPS, η θέση δηλαδή της κάθε συσκευής. Ο λόγος που έγινε αυτό είναι επειδή όλα τα έξυπνα τηλέφωνα στις μέρες μας έχουν δέκτη GPS ενώ υπάρχει και πληθώρα εφαρμογών που απαιτούν την παρακολούθηση θέσεων.

Αναπτύχθηκαν δύο εφαρμογές: Η MapTrackClient που τρέχει στις κινητές συσκευές που μεταδίδουν μετρήσεις και η MapTrackMonitor που δείχνει τη θέση της κάθε συσκευής στο χάρτη. Μια ενδεικτική τοπολογία του συστήματος δίνεται στην Εικόνα 7.



Εικόνα 7: Τοπολογία του συστήματος MapTrack

4.1 Η εφαρμογή MapTrackClient

Η εφαρμογή MapTrackClient τρέχει στο κινητό τηλέφωνο και λαμβάνει ενημερώσεις από το GPS όταν αλλάζει η θέση της. Τις συντεταγμένες της νέας θέσης τις δημοσιεύει στον broker σε ένα συγκεκριμένο topic.

4.1.1 Περιβάλλον ανάπτυξης

Η υλοποίηση έγινε σε Java και μόνο για συσκευές με λειτουργικό σύστημα Android. Χρησιμοποιήθηκε το περιβάλλον ανάπτυξης Android Studio[2] 1.5.1 το οποίο ήταν εγκατεστημένο σε λειτουργικό σύστημα Windows 7 64bit (Εικόνα 10). Το Android Studio εγκαθιστά και τον Android SDK που απαιτείται για τη δημιουργία εφαρμογών για κινητά Android.

[2] https://developer.android.com/studio/index.html

Τόσο στην εφαρμογή αυτή όσο και στην MapTrackMonitor χρησιμοποιείται η δωρεάν και ανοικτή βιβλιοθήκη Eclipse Paho Project[3] που υλοποιεί το πρωτόκολλο MQTT από τη μεριά του client (subscriber και publisher).

4.1.2 Topics

Κάθε publisher στέλνει τιμές σε ένα ειδικά σχηματισμένο topic το οποίο θα χρησιμοποιείται για να αποκαλύπτει και την ταυτότητα του client. Το topic θα έχει την παρακάτω μορφή:

`maptrack/<ταυτότητα client>/<είδος μέτρησης>`

Η ταυτότητα client είναι μοναδική ανά client και χρησιμοποιείται και κατά τη σύνδεσή του με τον broker. Προς το παρόν η εφαρμογή στέλνει μόνο θέση GPS της συσκευής οπότε το topic στο οποίο στέλνει ο κάθε client είναι το

`maptrack/<ταυτότητα client>/location`

Η αποστολή της ταυτότητας του client με χρήση επιπέδου στο topic έγινε επειδή με αυτό τον τρόπο απεμπλέκεται το καθαρό περιεχόμενο του μηνύματος από την ανάγκη να περιέχει και την ταυτότητα, πράγμα που κάνει ευκολότερη επέκταση του συστήματος με νέες κατηγορίες μετρήσεων στο μέλλον. Αν για παράδειγμα στο μέλλον η εφαρμογή στέλνει και πχ τιμές θερμοκρασίας, θα μπορεί να στέλνει την τιμή στο topic `maptrack/<ταυτότητα client>/temperature`.

Προφανώς κάθε client θα στέλνει στο δικό του topic που θα περιέχει το client id. Η εφαρμογή `MapTrackMonitor` που ενδιαφέρεται να παρακολουθεί όλους τους clients που στέλνουν τη θέση τους, εγγράφεται σε όλα τα σχετικά topics με τη χρήση wildcard δίνοντας ως topic maptrack/+/location κατά την αίτηση SUBSCRIBE.

[3] http://www.eclipse.org/paho/

Η τοποθεσία από κάθε κινητή συσκευή όταν λαμβάνεται από τον δέκτη GPS δημοσιεύεται ως κείμενο στο topic που περιγράφεται στην παράγραφο 4.1.2 με την εξής μορφή:

<Longitude>#<Latitude>

Για παράδειγμα:

```
37.9549#22.8102
```

Η εφαρμογή λαμβάνει ενημέρωση από το Android κάθε φορά που υπολογίζεται νέα θέση μέσω του GPS. Δεν στέλνεται όμως πάντα η νέα θέση αυτή στον broker. Ο λόγος που γίνεται αυτό είναι ότι υπάρχει ένα σφάλμα στον υπολογισμό με αποτέλεσμα ακόμα κι αν η πραγματική θέση δεν αλλάζει, η θέση που υπολογίζεται να αλλάζει. Για το λόγο αυτό, μια νέα θέση στέλνεται μόνο όταν απέχει πάνω από ένα συγκεκριμένο διάστημα[4] από την τελευταία τιμή που στάλθηκε στον broker.

Οι δοκιμές και η αποσφαλμάτωση (debugging) έγιναν απευθείας σε κινητό τηλέφωνο Samsung S3 Neo (με Android 4.4.2) μέσω καλωδίου USB.

4.1.3 Οθόνες εφαρμογής

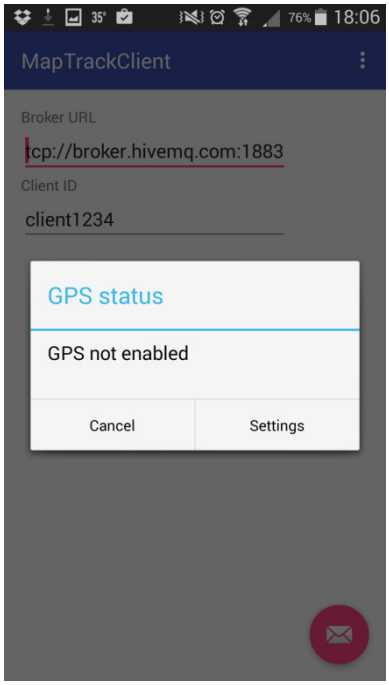
Η βασική οθόνη της εφαρμογής φαίνεται στην Εικόνα 8. Εδώ εισάγουμε την URL του broker καθώς επίσης και την ταυτότητα μας (client id) που πρέπει να είναι μοναδική στα πλαίσια του broker που συνδεόμαστε.

Για να ξεκινήσει η λήψη του γεωγραφικού στίγματος και η αποστολή της θέσης στον broker πατάμε το πλήκτρο με το φακελάκι κάτω δεξιά. Θα πρέπει βέβαια να έχουμε ενεργοποιήσει τον δέκτη GPS από τις ρυθμίσεις του κινητού. Αν κάτι τέτοιο δεν έχει συμβεί τότε μας εμφανίζεται σχετικό μήνυμα (Εικόνα 9). Πατώντας εδώ [Settings] μεταφερόμαστε στην οθόνη ρυθμίσεων της συσκευής από όπου μπορούμε να ενεργοποιήσουμε το GPS.

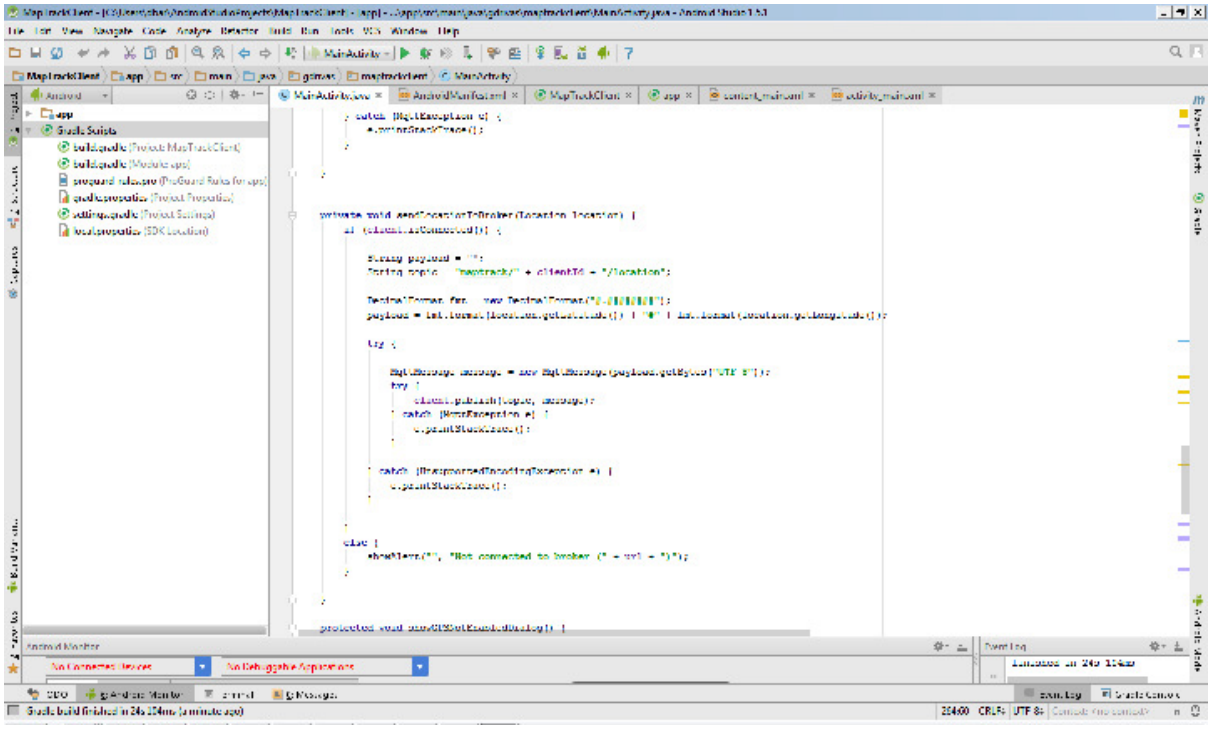
[4] Συγκεκριμένα το διάστημα αυτό ορίστηκε αυθαίρετα στα 25m. Σε μελλοντική έκδοση θα μπορούσε να παραμετροποιείται από το χρήστη.

Αν δεν παρουσιαστεί κάποιο θέμα, η εφαρμογή ξεκινάει κανονικά την αποστολή της θέσης της συσκευής στον broker.

Εικόνα 8: Η βασική οθόνη της MapTrackClient



Εικόνα 9: Το μήνυμα που εμφανίζεται όταν δεν έχουμε ενεργοποιήσει το δέκτη GPS



Εικόνα 10: Το περιβάλλον ανάπτυξης του Android Studio με το project MapTrackClient

4.1.4 Λεπτομέρειες υλοποίησης

Η οθόνη της εφαρμογής είναι υλοποιημένη ως Activity μιλώντας με όρους Android Development. Για να μπορεί να λαμβάνει ενημερώσεις από το GPS, υλοποιεί το interface android.location.LocationListener το οποίο παρέχει σχετικές μεθόδους οι οποίες καλούνται όταν αλλάξει κάτι στην κατάσταση του GPS (συνδέεται ή αποσυνδέεται) ή όταν υπολογιστεί νέα θέση. Στην τελευταία περίπτωση καλείται από το Android η onLocationChanged στην οποία περνάει ως παράμετρο το Location object με τις συντεταγμένες της νέας θέσης. Στο παρακάτω τμήμα κώδικα δίνεται η υλοποίηση της onLocationChanged ενώ φαίνεται ο έλεγχος που γίνεται για το πόσο απέχει η νέα θέση από την προηγούμενη που στάλθηκε ώστε να αποφασισθεί αν θα σταλεί και αυτή:

```
public static final float MIN_DISTANCE = 25;

@Override
public void onLocationChanged(Location location) {

    if(previousLocation == null) {
        previousLocation = location;
        sendLocationToBroker(location);
        return;
    }
```

```
    Date now = new Date();

    if((now.getTime()-location.getTime()>1000)||location.getTime() == 0) {
        // probably this location is a cached value,
        // wait for a new one
        return;
    }

    float distance =  location.distanceTo(previousLocation);
    if(distance < MIN_DISTANCE ) {
        return;
    }

    previousLocation = location;
    sendLocationToBroker(location);

}
```

Η sendLocationToBroker που καλείται, κωδικοποιεί και τις συντεταγμένες σε μια string μεταβλητή και τις δημοσιεύει στο σωστό topic (όπως αναφέρεται στην παράγραφο 4.1.2):

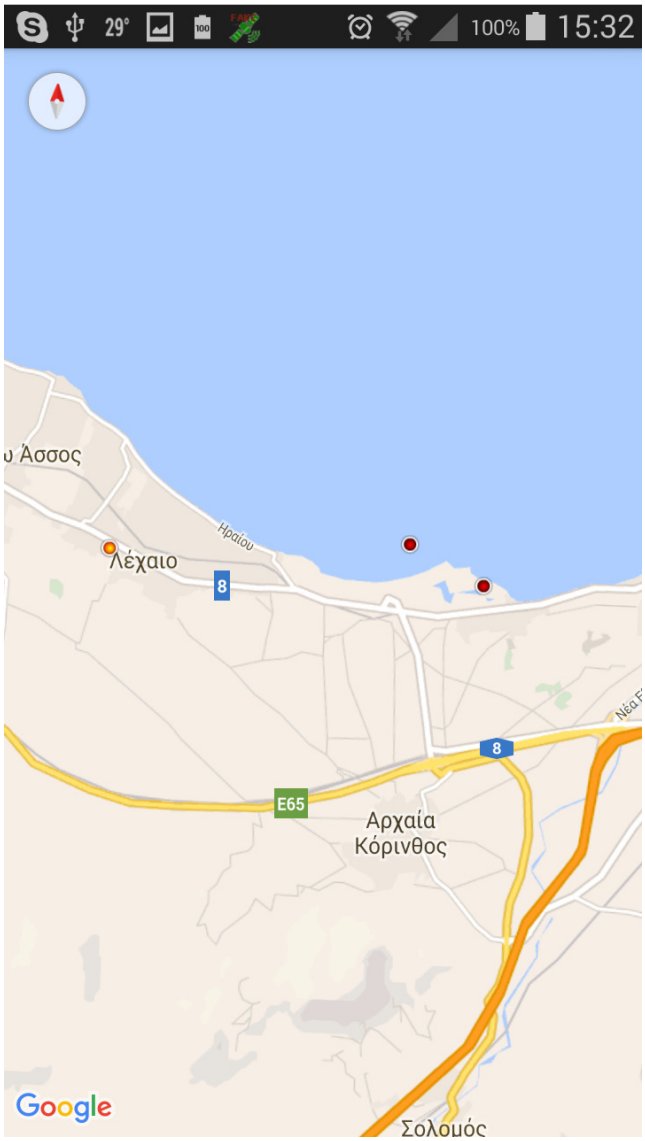
```
private void sendLocationToBroker(Location location) {
    if (client.isConnected()) {

        String payload = "";
        String topic = "maptrack/" + clientId + "/location";

        DecimalFormat fmt = new DecimalFormat("#.########");
        payload = fmt.format(location.getLatitude()) + "#" +
fmt.format(location.getLongitude());

        try {

           MqttMessage message = new MqttMessage(payload.getBytes("UTF-8"));
            try {
                client.publish(topic, message);
            } catch (MqttException e) {
                e.printStackTrace();
            }

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

    }
    else {
        showAlert("", "Not connected to broker (" + url + ")");
    }

}
```

4.2 Η εφαρμογή MapTrackMonitor

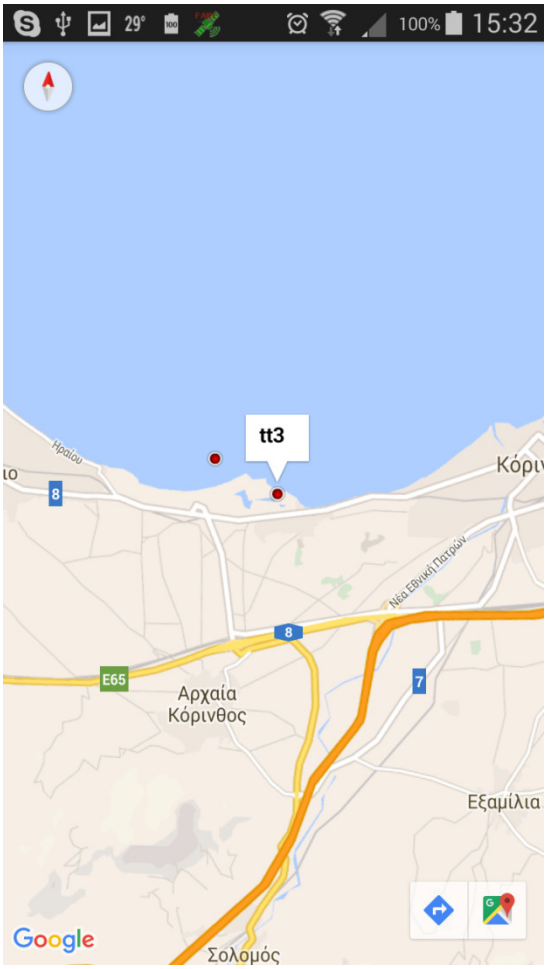
Η εφαρμογή αυτή δείχνει τη θέση της κάθε συσκευής που «τρέχει» την MapTrackClient. Τρέχει σε συσκευή Android. Συνδέεται στον broker και εγγράφεται

στα topics που στέλνουν τις θέσεις τους οι συσκευές. Κάνει δηλαδή SUBSCRIBE στα topics maptrack/+/location.

Το μόνο που δείχνει η εφαρμογή είναι ενα χάρτη πάνω στον οποίο δίνεται τόσο η θέση της συσκευής που τρέχει το MapTrackMonitor, όσο και οι θέσεις των συσκευών που τρέχουν την MapTrackClient (Εικόνα 11). Κάνοντας κλικ πάνω σε έναν MapTrackClient μας εμφανίζεται η ταυτότητά του.



Εικόνα 11: Ο χάρτης της εφαρμογής. Με το πορτοκαλί εικονίδιο (στο κέντρο αριστερά, κοντά στο Λέχαιο) είναι η θέση της συσκευής που τρέχει την MapTrackMonitor. Οι συσκευές με το MapTrackClient αναπαριστώνται με το κόκκινο εικονίδιο (κέντρο και δεξιά).



Εικόνα 12: Κάνοντας κλικ πάνω στο κόκκινο εικονίδιο ενός MapTrackClient μας εμφανίζει σε «μπαλονάκι» την ταυτότητά του.

Όταν ξεκινάει η εφαρμογή, αναζητάει την τρέχουσα θέση της συσκευής (αν είναι ενεργοποιημένος ο δέκτης GPS). Μόλις την εντοπίσει, ζουμάρει στο σημείο αυτό του χάρτη σε ύψος όσο φαίνεται στην Εικόνα 11. Μπορούμε βέβαια να μετακινηθούμε μέσα στο χάρτη όπως με οποιαδήποτε εφαρμογή χαρτών σε Android.

Όποτε λαμβάνει ενημερώσεις από τον broker, τις οπτικοποιεί αυτομάτως στον χάρτη.

4.2.1 Περιβάλλον ανάπτυξης

Το περιβάλλον ανάπτυξης και δοκιμής είναι το ίδιο με αυτό της MapTrackClient.

4.2.2 Λεπτομέρειες υλοποίησης

Ο χάρτης υλοποιήθηκε με τη βιβλιοθήκη GoogleMaps για Android. Η δυσκολία εδώ είναι στην ενσωμάτωση των χαρτών στην εφαρμογή αφού πέρα από την υλοποίηση η οποία είναι *σχετικά* εύκολη και υπάρχει πληθώρα παραδειγμάτων στο διαδίκτυο για το πως γίνεται, θα πρέπει να πάρουμε έναν ειδικό κωδικό (**API key**) από το **Google**, μοναδικό για την εφαρμογή μας τον οποίο θα ενσωματώσουμε σε αυτή για να μπορεί να κατεβάζει τους χάρτες κατά την λειτουργία της. Αυτό γίνεται στην παρακάτω διεύθυνση:

https://console.developers.google.com/project

ενώ απαιτείται να έχουμε λογαριασμό στην Google.

Το παρακάτω άρθρο περιγράφει την διαδικασία, από την δημιουργία του API key ως και την ενσωμάτωση και τον κώδικα που απαιτείται για την προβολή του χάρτη στην εφαρμογή μας:

http://code.tutsplus.com/series/getting-started-with-google-maps-for-android--cms-891

Ο κώδικας των δύο εφαρμογών του συστήματος MapTrack δίνεται στο κεφάλαιο "Πηγαίος Κώδικας MapTrack".

Βιβλιογραφία

[1] M. Weiser, «The computer for the 21st century,» *Scientific american,* τόμ. 265, αρ. 3, pp. 94-104, 1991.

[2] D. Evans, «The internet of things: How the next evolution of the internet is changing everything,» *CISCO white paper,* τόμ. 1, pp. 1-11, 2011.

[3] A. Zanella, N. Bui, A. Castellani, L. Vangelista και M. Zorze, «Internet of Things for Smart Cities,» *IEEE Internet of Things Journal,* τόμ. 1.1, pp. 22-32, 2014.

[4] IBM, «IBM Podcast,» [Ηλεκτρονικό]. Available: http://www.ibm.com/podcasts/software/websphere/connectivity/piper_diaz_nipper_mq_tt_11182011.pdf. [Πρόσβαση Μάρτιος 2016].

[5] «MQTT Version 3.1.1 becomes an OASIS Standard,» OASIS, [Ηλεκτρονικό]. Available: https://www.oasis-open.org/news/announcements/mqtt-version-3-1-1-becomes-an-oasis-standard. [Πρόσβαση Μάρτιος 2016].

[6] OASIS, «MQTT Version 3.1.1 - OASIS Standard,» Οκτώβριος 2014. [Ηλεκτρονικό]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

[7] O. Dominik, «SYS Topics - mqtt.github.io,» [Ηλεκτρονικό]. Available: https://github.com/mqtt/mqtt.github.io/wiki/SYS-Topics. [Πρόσβαση Μάιος 2016].

Πηγαίος κώδικας MapTrack

MapTrackClient

```
package gdrivas.maptrackclient;

import android.Manifest;
import android.app.AlertDialog;
import android.content.ContentResolver;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.provider.Settings;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;

import org.eclipse.paho.android.service.MqttAndroidClient;
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.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;


import java.util.Date;

public class MainActivity extends AppCompatActivity implements View.OnClickListener,
LocationListener {

    private LocationManager m_LocationManager;
    private Location previousLocation;

    public static final float MIN_DISTANCE = 25;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
```

```
    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();
        }
    });


    getGPSCoordinates();



}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}



@Override
public void onLocationChanged(Location location) {

    if(previousLocation == null) {
        previousLocation = location;
        sendLocationToBroker(location);
        return;
    }

    Date now = new Date();

    if (( now.getTime() - location.getTime() > 1000) || location.getTime() == 0) {
        // probably this location is a cached value,
        // wait for a new one
        return;
    }

    float distance =  location.distanceTo(previousLocation);
    if(distance < MIN_DISTANCE ) {
        return;
    }

    previousLocation = location;
    sendLocationToBroker(location);

}

@Override
public void onStatusChanged(String s, int i, Bundle bundle) {

}

@Override
public void onProviderEnabled(String s) {

}
```

```
    @Override
    public void onProviderDisabled(String s) {

    }

    @Override
    public void onClick(View view) {

    }

    public void getGPSCoordinates() {
        if (!gpsIsEnabled()) {
            showGPSNotEnabledDialog();
        } else {
            scanningForLocation();
            if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
                // TODO: Consider calling
                //    ActivityCompat#requestPermissions
                // here to request the missing permissions, and then overriding
                //   public void onRequestPermissionsResult(int requestCode, String[]
permissions,
                //                                          int[] grantResults)
                // to handle the case where the user grants the permission. See the
documentation
                // for ActivityCompat#requestPermissions for more details.
                return;
            }
            m_LocationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
this);
        }
    }

    private void scanningForLocation() {
        /*
        btnScan.setText(R.string.gettingLocation);
        btnScan.setEnabled(false);
        btnNext.setEnabled(false);
        setTextViewText(R.id.txtLat, "");
        setTextViewText(R.id.txtLng, "");
        */
    }

    private Boolean gpsIsEnabled() {
        ContentResolver contentResolver = getBaseContext().getContentResolver();
        return Settings.Secure.isLocationProviderEnabled(contentResolver,
LocationManager.GPS_PROVIDER);
    }


    private void sendLocationToBroker(Location location) {

    }

    protected void showGPSNotEnabledDialog() {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setMessage("GPS not enabled")
                .setCancelable(false)
                .setTitle("GPS status")
                .setPositiveButton(getString(R.string.action_settings),
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int id) {


                                Intent myIntent = new Intent(
                                        Settings.ACTION_SECURITY_SETTINGS);
                                startActivity(myIntent);
                                dialog.cancel();
                            }
                        })
                .setNegativeButton("Cancel",
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int id) {
                                // cancel the dialog box
                                dialog.cancel();
```

```
                            }
                        });
        AlertDialog alert = builder.create();
        alert.show();

    }
}
```

MapTrackMonitor

```
package gdrivas.maptrackmonitor;

import android.Manifest;
import android.app.AlertDialog;
import android.app.ProgressDialog;
import android.content.ContentResolver;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.content.pm.PackageManager;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.net.Uri;
import android.provider.Settings;
import android.support.v4.app.ActivityCompat;
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;

import com.google.android.gms.appindexing.Action;
import com.google.android.gms.appindexing.AppIndex;
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.maps.CameraUpdate;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;

import org.eclipse.paho.android.service.MqttAndroidClient;
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.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;

import java.util.Hashtable;
import java.util.Set;


public class MapsActivity extends FragmentActivity implements OnMapReadyCallback,
        GoogleApiClient.ConnectionCallbacks,
        GoogleApiClient.OnConnectionFailedListener,
        GoogleMap.OnInfoWindowClickListener,
        GoogleMap.OnMapLongClickListener,
        GoogleMap.OnMapClickListener,
        GoogleMap.OnMarkerClickListener,
        LocationListener,
        MqttCallback {

    private GoogleMap mMap;
    private boolean bMapIsReady = false;
    private Location m_currentLocation = null;
    private ProgressDialog pDialog;
```

```
    private LocationManager m_LocationManager;

    private MqttAndroidClient client;

    private LatLng lastLatLng;

    private String brokerURL = "tcp://broker.hivemq.com:1883";
    private String topic = "maptrack/+/location";
    private String clientId = "gdrv-maptrack-monitor";

    private Hashtable<String, Location> ht;
    /**
     * ATTENTION: This was auto-generated to implement the App Indexing API.
     * See https://g.co/AppIndexing/AndroidStudio for more information.
     */
    private GoogleApiClient client2;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_SENSOR_PORTRAIT);

        m_LocationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);

        // Obtain the SupportMapFragment and get notified when the map is ready to be used.
        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);

        ht = new Hashtable<String, Location>();


        connectToBroker();

        if (client.isConnected()) {
            try {
                client.subscribe(topic, 1);
            } catch (MqttException e) {
                e.printStackTrace();
            }
        } else {

        }
        // ATTENTION: This was auto-generated to implement the App Indexing API.
        // See https://g.co/AppIndexing/AndroidStudio for more information.
        client2 = new GoogleApiClient.Builder(this).addApi(AppIndex.API).build();
    }


    /**
     * Manipulates the map once available.
     * This callback is triggered when the map is ready to be used.
     * This is where we can add markers or lines, add listeners or move the camera. In this
case,
     * we just add a marker near Sydney, Australia.
     * If Google Play services is not installed on the device, the user will be prompted to
install
     * it inside the SupportMapFragment. This method will only be triggered once the user has
     * installed Google Play services and returned to the app.
     */
    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
        bMapIsReady = true;

        getGPSCoordinates();
    }

    private void setMapToCurrentPosition() {

    }

    @Override
    public void onConnected(Bundle bundle) {
```

```
    }

    @Override
    public void onConnectionSuspended(int i) {

    }

    @Override
    public void onConnectionFailed(ConnectionResult connectionResult) {

    }

    @Override
    public void onInfoWindowClick(Marker marker) {

    }

    @Override
    public void onMapClick(LatLng latLng) {

    }

    @Override
    public void onMapLongClick(LatLng latLng) {

    }

    @Override
    public boolean onMarkerClick(Marker marker) {
        return false;
    }

    @Override
    public void onLocationChanged(Location location) {
        if (pDialog.isShowing())
            pDialog.dismiss();

        if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION)
!= PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
            // TODO: Consider calling
            //    ActivityCompat#requestPermissions
            // here to request the missing permissions, and then overriding
            //   public void onRequestPermissionsResult(int requestCode, String[] permissions,
            //                                          int[] grantResults)
            // to handle the case where the user grants the permission. See the documentation
            // for ActivityCompat#requestPermissions for more details.
            return;
        }
        m_LocationManager.removeUpdates(this);


        lastLatLng = new LatLng(location.getLatitude(), location.getLongitude());


        CameraUpdate cameraUpdate = CameraUpdateFactory.newLatLngZoom(lastLatLng, 15);
        mMap.animateCamera(cameraUpdate);

        drawCurrentLocation();
    }

    @Override
    public void onStatusChanged(String s, int i, Bundle bundle) {

    }

    @Override
    public void onProviderEnabled(String s) {

    }

    @Override
    public void onProviderDisabled(String s) {

    }

    private void scanningForLocation() {
```

```
        pDialog = new ProgressDialog(this);
        pDialog.setMessage("Getting location from GPS");
        pDialog.setCancelable(false);
        pDialog.show();


    }

    public void getGPSCoordinates() {
        if (!gpsIsEnabled()) {
            showGPSNotEnabledDialog();
        } else {
            scanningForLocation();
            if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
                // TODO: Consider calling
                //    ActivityCompat#requestPermissions
                // here to request the missing permissions, and then overriding
                //   public void onRequestPermissionsResult(int requestCode, String[]
permissions,
                //                                          int[] grantResults)
                // to handle the case where the user grants the permission. See the
documentation
                // for ActivityCompat#requestPermissions for more details.
                return;
            }
            m_LocationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
this);
        }
    }

    protected void showGPSNotEnabledDialog() {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setMessage("GPS not enabled")
                .setCancelable(false)
                .setTitle("GPS status")
                .setPositiveButton(getString(R.string.action_settings),
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int id) {


                                Intent myIntent = new Intent(
                                        Settings.ACTION_LOCATION_SOURCE_SETTINGS);
                                startActivity(myIntent);
                                dialog.cancel();
                            }
                        })
                .setNegativeButton("Cancel",
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int id) {
                                // cancel the dialog box
                                dialog.cancel();
                            }
                        });
        AlertDialog alert = builder.create();
        alert.show();

    }

    private Boolean gpsIsEnabled() {
        ContentResolver contentResolver = getBaseContext().getContentResolver();
        return Settings.Secure.isLocationProviderEnabled(contentResolver,
LocationManager.GPS_PROVIDER);
    }


    private void drawCurrentLocation() {
        if( lastLatLng != null) {
            mMap.addMarker(new MarkerOptions()
                            .position(lastLatLng)
                            .title("your are here")

.icon(BitmapDescriptorFactory.fromResource(R.drawable.yellow_point))
            );
```

```
    }
}


@Override
public void connectionLost(Throwable cause) {

}

@Override
public void messageArrived(String topic, MqttMessage message) throws Exception {

    String[] s = topic.split("/");

    if (s.length == 3) {
        if (s[0].equalsIgnoreCase("maptrack")) {

            switch (s[2]) {
                case "location":
                    handleNewLocation(s[1], message.toString());
                    break;
                default:
                    break;
            }
        }
    }

}

private void handleNewLocation(String clientId, String sLocation) {
    String[] s = sLocation.split("#");
    if (s.length == 2) {
        double lat, lng;

        lat = Double.parseDouble(s[0]);
        lng = Double.parseDouble(s[1]);



        Location loc = ht.get(clientId);
        if (loc == null) {
            loc = new Location("");
            loc.setLongitude(lng);
            loc.setLatitude(lat);
            ht.put(clientId, loc);
        }

        loc.setLongitude(lng);
        loc.setLatitude(lat);
    } else {
        return;
    }

    drawMap();
}

private void drawMap() {
    mMap.clear();

    drawCurrentLocation();

    Set<String> keys = ht.keySet();
    for (String key : keys) {
        Location loc = ht.get(key);

        LatLng l = new LatLng(loc.getLatitude(), loc.getLongitude());


        mMap.addMarker(new MarkerOptions()
                        .position(l)
                        .title(key)
                        .icon(BitmapDescriptorFactory.fromResource(R.drawable.red_point))
        );


    }
```

```
    }

    @Override
    public void deliveryComplete(IMqttDeliveryToken token) {

    }

    private void connectToBroker() {

        client = new MqttAndroidClient(this.getApplicationContext(), brokerURL, clientId);
        client.setCallback(this);

        try {
            IMqttToken token = client.connect();
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {

                    if (client.isConnected()) {
                        try {
                            client.subscribe(topic, 1);
                        } catch (MqttException e) {
                            e.printStackTrace();
                        }
                    } else {
                        showAlert("", "Not connected to broker " + brokerURL);
                    }


                }

                @Override
                public void onFailure(IMqttToken asyncActionToken, Throwable exception) {

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

        /*
        if(client.isConnected() == false) {
            showAlert("", "Not connected to broker " + brokerURL);
        }
        */

    }

    protected void showAlert(String title, String message) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setMessage(message)
                .setCancelable(false)
                .setTitle(title)
                .setNegativeButton("OK",
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int id) {
                                dialog.cancel();
                            }
                        });
        AlertDialog alert = builder.create();
        alert.show();
    }

    @Override
    public void onStart() {
        super.onStart();

        // ATTENTION: This was auto-generated to implement the App Indexing API.
        // See https://g.co/AppIndexing/AndroidStudio for more information.
        client2.connect();
        Action viewAction = Action.newAction(
                Action.TYPE_VIEW, // TODO: choose an action type.
                "Maps Page", // TODO: Define a title for the content shown.
                // TODO: If you have web page content that matches this app activity's
content,
                // make sure this auto-generated web page URL is correct.
```

```
                // Otherwise, set the URL to null.
                Uri.parse("http://host/path"),
                // TODO: Make sure this auto-generated app deep link URI is correct.
                Uri.parse("android-app://gdrivas.maptrackmonitor/http/host/path")
        );
        AppIndex.AppIndexApi.start(client2, viewAction);
    }

    @Override
    public void onStop() {
        super.onStop();

        // ATTENTION: This was auto-generated to implement the App Indexing API.
        // See https://g.co/AppIndexing/AndroidStudio for more information.
        Action viewAction = Action.newAction(
                Action.TYPE_VIEW, // TODO: choose an action type.
                "Maps Page", // TODO: Define a title for the content shown.
                // TODO: If you have web page content that matches this app activity's
content,
                // make sure this auto-generated web page URL is correct.
                // Otherwise, set the URL to null.
                Uri.parse("http://host/path"),
                // TODO: Make sure this auto-generated app deep link URI is correct.
                Uri.parse("android-app://gdrivas.maptrackmonitor/http/host/path")
        );
        AppIndex.AppIndexApi.end(client2, viewAction);
        client2.disconnect();
    }
}
```

Οδηγίες εγκατάστασης ActiveMQ σε Windows 7

Οι τελευταίες εκδόσεις είναι διαθέσιμες στο παρακάτω URL:

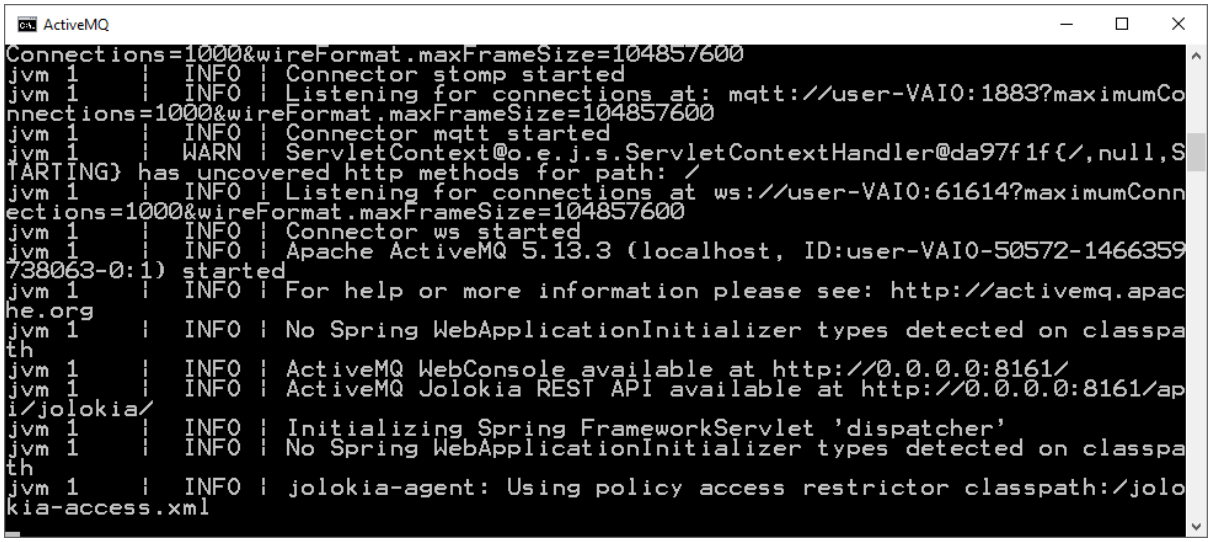
http://activemq.apache.org/download.html

Οι οδηγίες αυτές περιγράφουν τη διαδικασία εγκατάστασης της έκδοσης 5.13.3 που μεταφορτώθηκε από τη διεύθυνση http://activemq.apache.org/activemq-5133-release.html. Η εγκατάσταση γίνεται σε Windows 7 64bit.

Μόλις κατέβει το συμπιεσμένο αρχείο, το αποσυμπιέζουμε σε κάποιο φάκελο. Δεν χρειάζεται εγκατάσταση αφού δεν υπάρχει Installer. Από το φάκελο που το αποσυμπιέσαμε μπορούμε να το τρέξουμε. Αν για παράδειγμα η αποσυμπίεση έγινε στο φάκελο C:\apache-activemq-5.13.3 τότε μπαίνουμε στον υποφάκελο C:\apache-activemq-5.13.3\bin\win64 από όπου τρέχουμε το αρχείο activemq.bat.

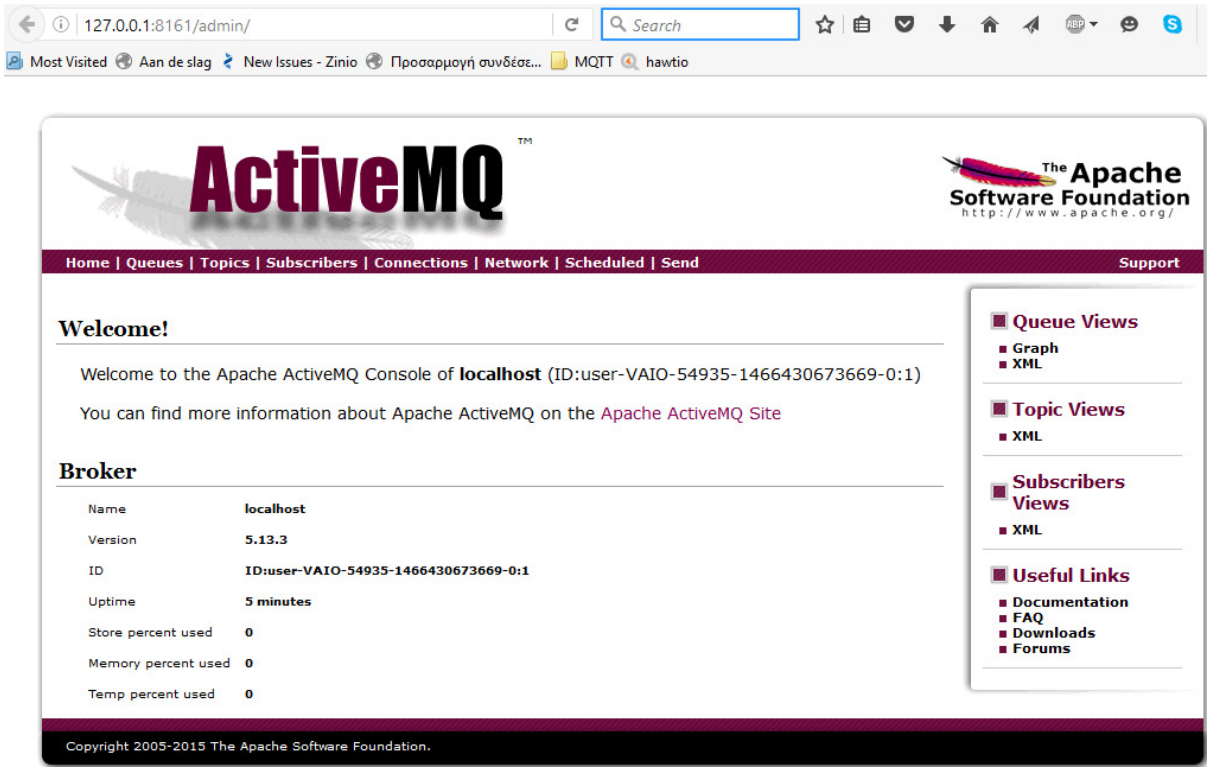
Αν όλα πάνε καλά τότε βλέπουμε μια εικόνα σαν την Εικόνα 13.

Υπάρχει και η δυνατότητα εκτελώντας το αρχείο InstallService.bat στον ίδιο φάκελο, να εγκατασταθεί ως service οπότε και θα τρέχει αυτόματα κάθε φορά που ανοίγει ο υπολογιστής.



Εικόνα 13: Επιτυχής εκκίνηση του broker

Με την εκκίνηση του broker, τρέχει κι ένας web server που μας επιτρέπει να συνδεθούμε στην κονσόλα του broker. Ο server αυτός είναι διαθέσιμος στην τοπική διεύθυνση http://127.0.0.1:8161



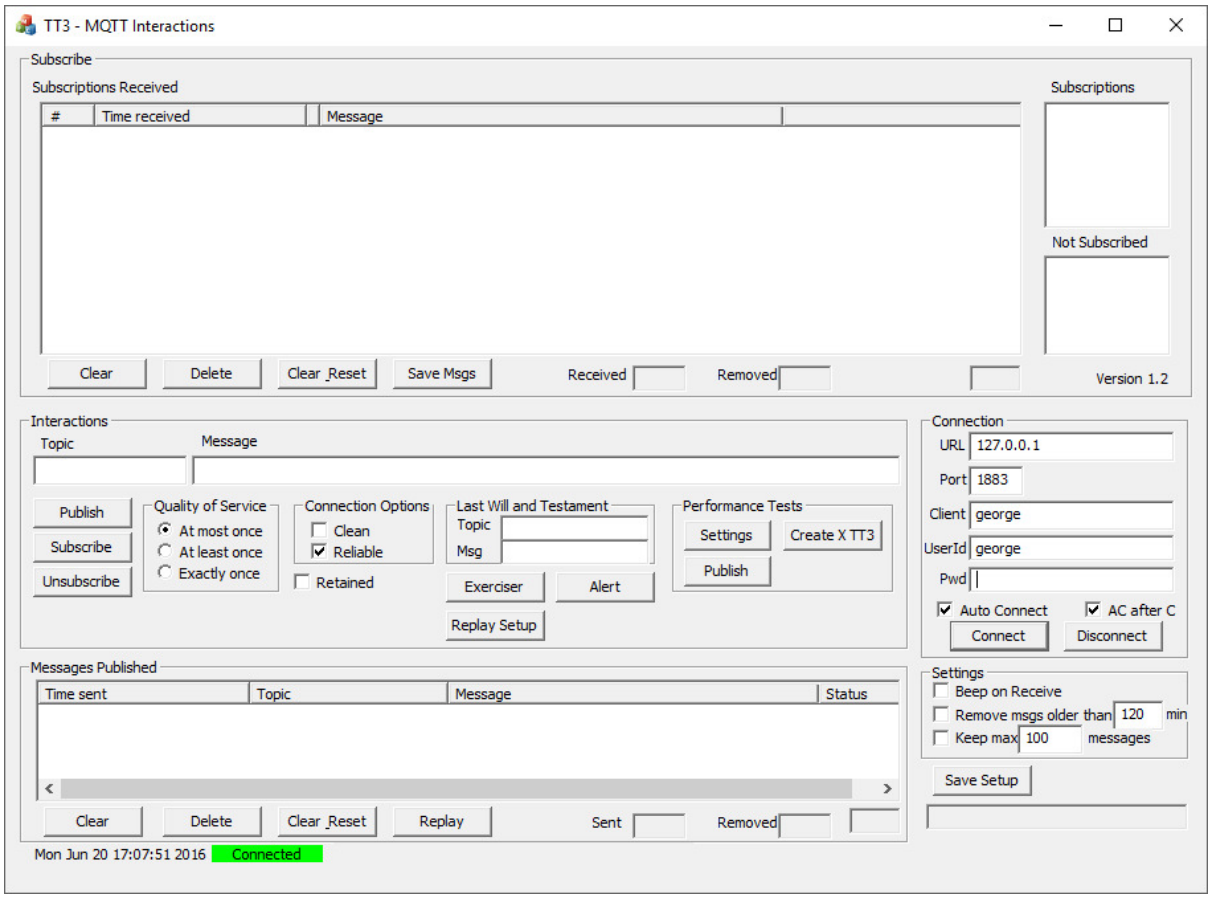
Συνδεόμαστε δίνοντας:

`Username: admin`

```
Password: admin
```

Από την κονσόλα μπορούμε να δούμε διάφορες σχετικές πληροφορίες όπως συνδεδεμένους clients, τρέχοντα topics κλπ.

Αν θέλουμε να δοκιμάσουμε τη σύνδεση με τον broker, μπορούμε να τρέξουμε έναν MQTT client. Στο παράδειγμα εδώ χρησιμοποιούμε τον TT3, ένα πρόγραμμα για Windows γραμμένο σε C++ και διαθέσιμο στο Github[5].



Εικόνα 14: Οθόνη του TT3

Στην οθόνη του TT3 που ανοίγει, στο πλαίσιο Connection (κέντρο δεξιά), συμπληρώνουμε τα παρακάτω πεδία όπως φαίνεται και στην Εικόνα 14.

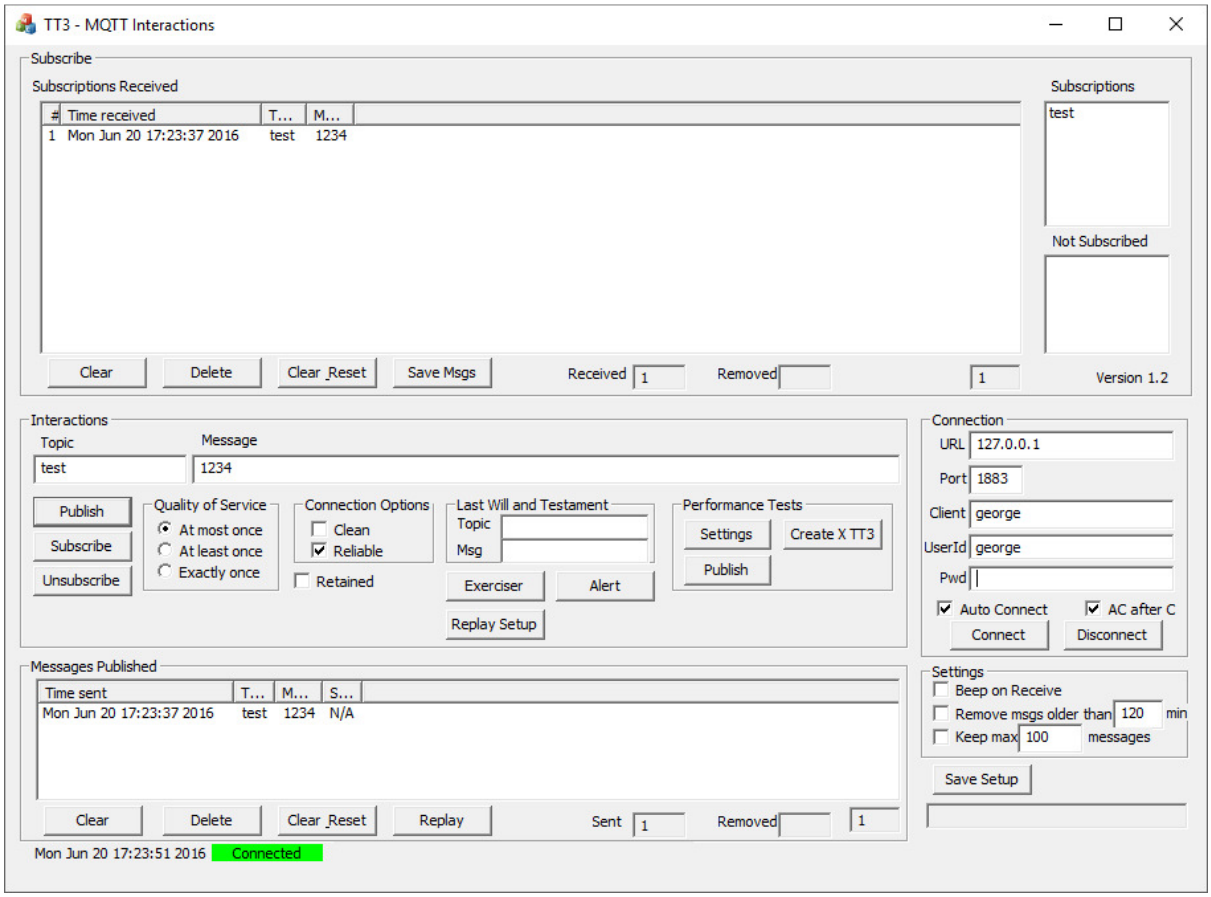
Πεδίο	**Τιμή**
URL	127.0.0.1

[5] https://github.com/francoisvdm/TT3

Port	1883
Client	Εδώ δίνουμε μια τιμή που θα είναι μοναδική στον broker και θα είναι το αναγνωριστικό μας ως clients του broker. Στο παράδειγμα εδώ έχει δοθεί η τιμή **George**.

Στη συνέχεια πατάμε το κουμπί Connect και αν όλα πάνε καλά, βλέπουμε κάτω αριστερά την τιμή Connected σε πράσινο φόντο.

Μπορούμε τώρα να δοκιμάσουμε να στείλουμε ένα μήνυμα στον broker. Δίνουμε ένα topic στο πεδίο topic του πλαισίου interactions και δίπλα στο πεδίο message γράφουμε το μήνυμα. Στη συνέχεια πατάμε το κουμπί Publish και το στέλνουμε.



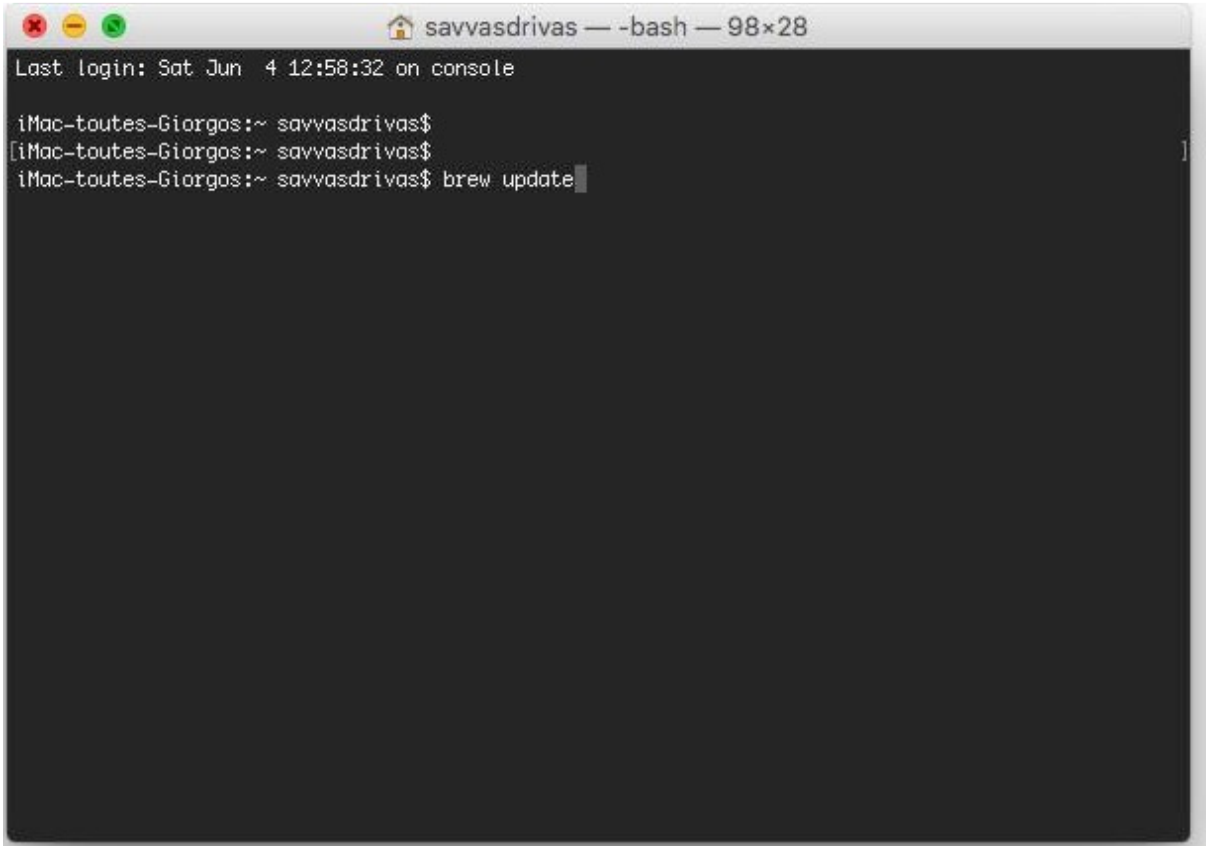
Εικόνα 15: Λήψη μηνύματος που εμείς οι ίδιοι στέλνουμε

Για να δούμε αν όντως το μήνυμα μεταφέρεται μπορούμε να κάνουμε subscribe στο topic που εμείς στέλνουμε κι έτσι ο broker θα στείλει πίσω το μήνυμα και σε εμας. Εγγραφόμαστε στο topic μας πατώντας το κουμπί subscribe ακριβώς κάτι από το Publish. Αν λοιπόν στείλουμε (δημοσιεύσουμε) μήνυμα, το βλέπουμε να μας έρχεται πίσω και το TT3 να μας το δείχνει στο πάνω μέρος της οθόνης και στο πλαίσιο Subscribe/Subscriptions received (Εικόνα 15).

Οδηγίες εγκατάστασης Apache Apollo σε MacOS X

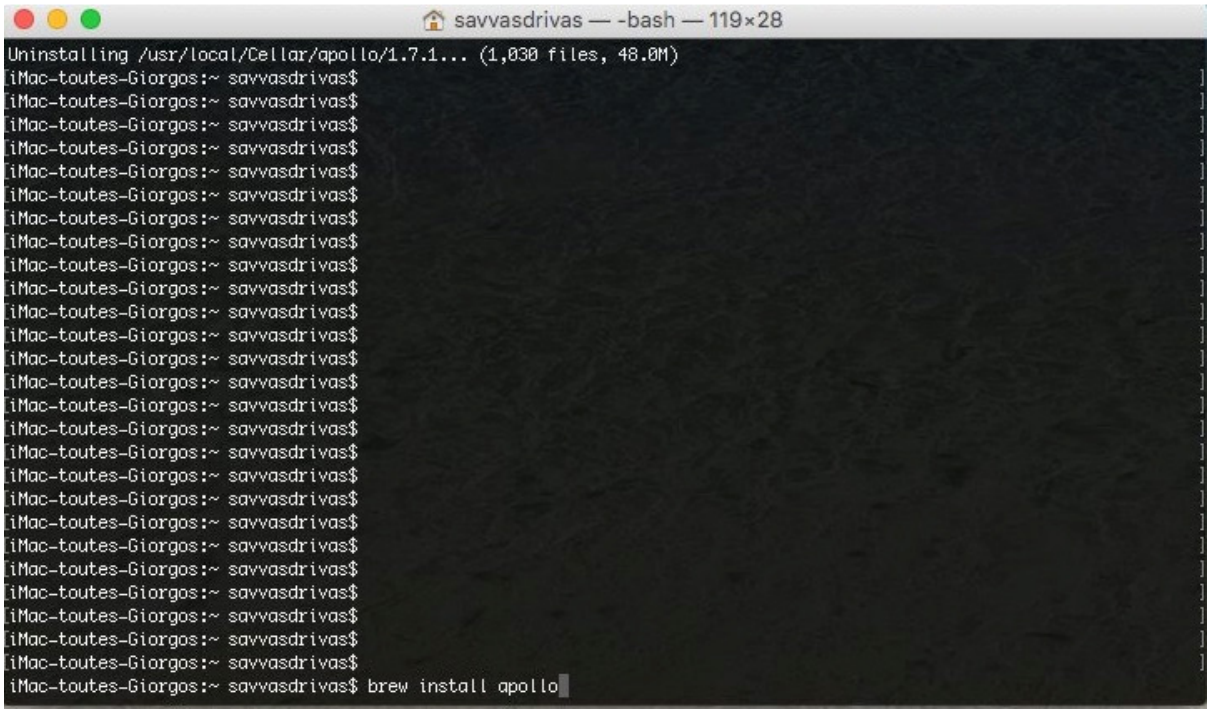
Η εγκατάσταση που περιγράφεται εδώ γίνεται μέσω του homebrew (http://brew.sh/) ενός package manager παρόμοιου με τον αντίστοιχο σε Linux. Διατίθεται δωρεάν και διευκολύνει σημαντικά την εγκατάσταση νέου λογισμικού. Είναι γραμμένο σε Ruby Script.

Πριν από κάθε εγκατάσταση, κάνουμε update πληκτρολογώντας brew update σε μια γραμμή εντολών:

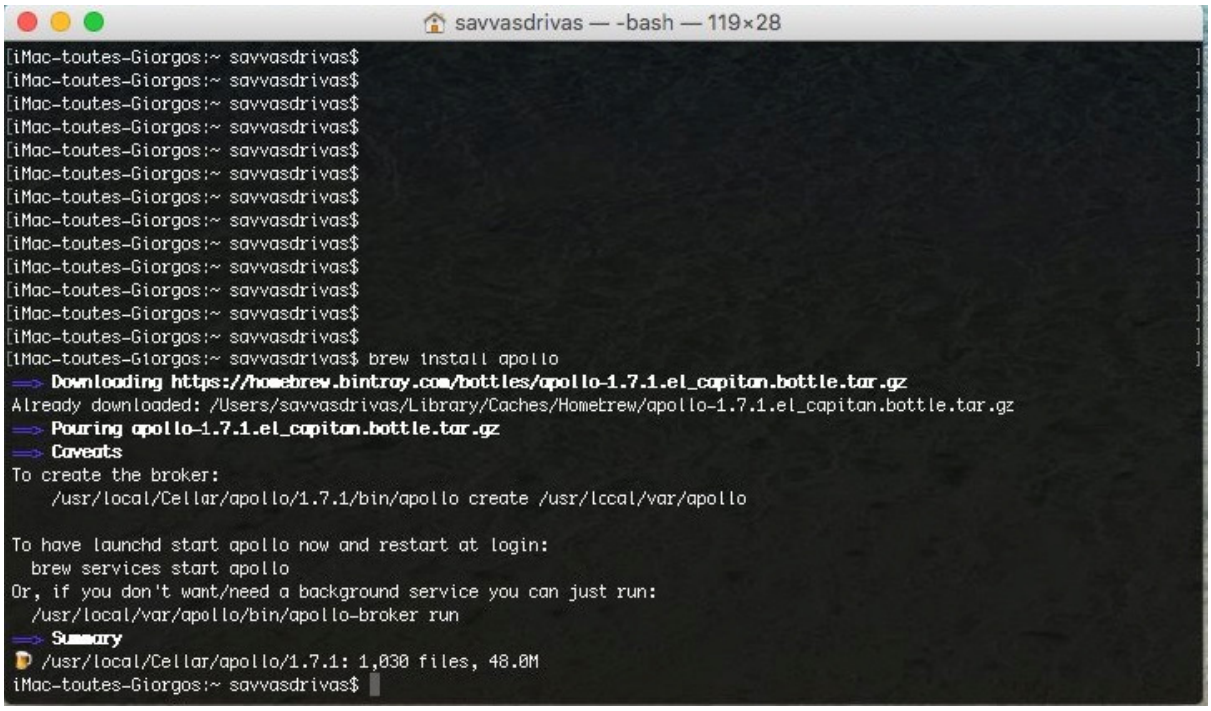


Η εγκατάσταση ξεκινάει πληκτρολογώντας:

```
brew install Apollo
```



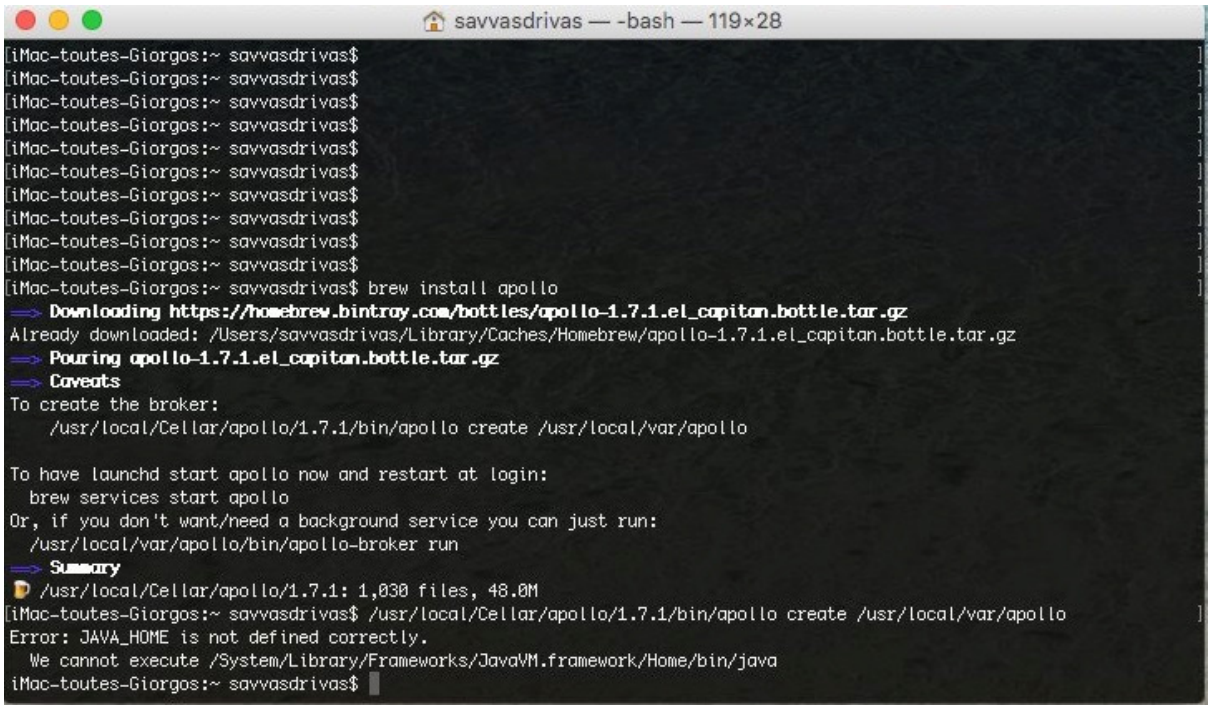
Και η εγκατάσταση ξεκινά:



Παρατηρείστε ότι δεν κάνουμε εμείς κάποια λήψη λογισμικού του Apollo. Το κάνει όπως φαίνεται και στην παραπάνω εικόνα το brew.

Μετά την εγκατάσταση το brew μας πληροφορεί πως μπορούμε να δημιουργήσουμε τον broker. Πληκτρολογούμε αυτό που μας υποδεικνύεται:

```
/usr/local/cellar/Apollo/1.7.1/bin/apollo create /usr/local/var/apollo
```



Αν μας εμφανίσει το μήνυμα λάθους που φαίνεται παραπάνω (Error: JAVA_HOME is not defined correctly) τότε δίνουμε τις παρακάτω εντολές:

```
export JAVA_HOME=$(/usr/libexec/java_home)
```

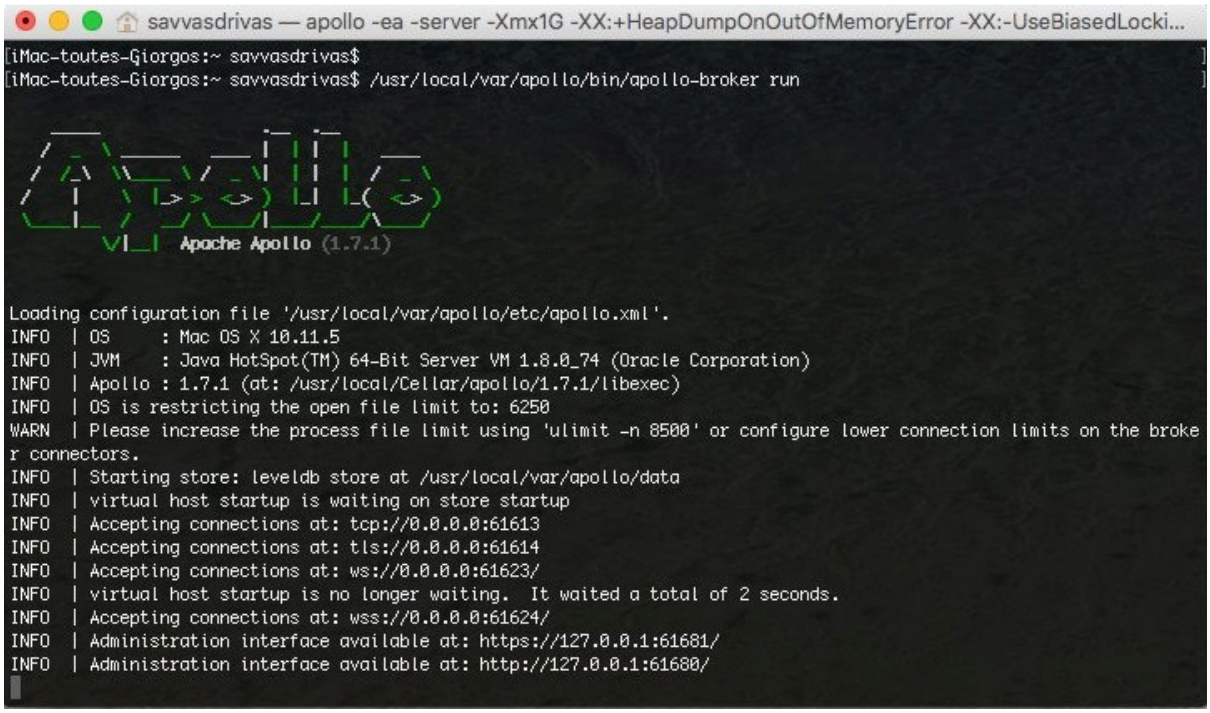
και

```
export PATH=$JAVA_HOME/jre/bin:$PATH
```

Στη συνέχεια πληκτρολογούμε πάλι την εντολή που αναφέρθηκε παραπάνω για τη δημιουργία του broker και μετά ξεκινάμε τον broker δίνοντας την εντολή:

```
/usr/local/var/apollo/bin/apollo/apollo-broker run
```

Και η οθόνη που βλέπουμε είναι σαν την παρακάτω:



Ο broker είναι έτοιμος για χρήση.