Ατομική Διπλωματική Εργασία

PPCOMM: POINT-TO-POINT ΕΠΙΚΟΙΝΩΝΙΑ ΚΟΜΒΩΝ ΣΤΟ FOGIFY

Φεσάς Χρίστος

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ιούνιος 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

PPComm: Point-to-Point Επικοινωνία κόμβων στο Fogify

Φεσάς Χρίστος

Επιβλέπων Καθηγητής

κ. Πάλλης Γεώργιος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου.

Ιούνιος 2021

Ευχαριστίες

Θα ήθελα να πω ένα μεγάλο ευχαριστώ στον επιβλέπων καθηγητή μου κύριο Πάλλη Γεώργιο για την τεράστια βοήθεια που μου πρόσφερε κατά την διάρκεια της μελέτης και εκπόνησης της διπλωματικής μου εργασίας.

Επίσης, θα ήθελα να ευχαριστήσω θερμά τον βοηθό έρευνας και υποψήφιο διδακτορικού τίτλου, κύριο Συμεωνίδη Μωυσή για την ευκαιρία που μου έδωσε να ασχοληθώ με το πολύ ενδιαφέρον Fogify Framework και για την ανεκτίμητη βοήθεια και υποστήριξη του πρόσφερε για την ολοκλήρωση της υλοποίησης της διπλωματικής εργασίας μου.

Τέλος, θα ήθελα να πω ένα τεράστιο ευχαριστώ στους γονείς μου, Πανίκο και Φάνη, για την ανεκτίμητη υποστήριξη και βοήθεια που μου πρόσφεραν καθ' όλη τη διάρκεια των σπουδών μου.

Περίληψη

Η συγκεκριμένη εργασία εστιάζει στην δημιουργία μιας επέκτασης στον υφιστάμενο εξομοιωτή Fogify (A Fog Computing Emulator Framework), η οποία ονομάζεται PPComm (**P**oint-to-**P**oint **Comm**unication). Το Fogify δημιουργήθηκε με σκοπό να ευκολύνει την μοντελοποίηση, την ανάπτυξη και τον πειραματισμό Fog testbeds. Παρέχει εργαλεία για την μοντελοποίηση fog τοπολογιών με σύνθετες δυνατότητες, την ανάπτυξη και το deployment της μοντελοποιημένης διαμόρφωσης και υπηρεσιών με την χρήση containerized infrastructure-as-code περιγραφών (Docker Containers) και τον πειραματισμό, μέτρηση και αξιολόγηση των deployments με διάφορους τρόπους στους οποίους θα αναφερθούμε αργότερα.

Στο υφιστάμενο σύστημα του Fogify, ο χρήστης μπορεί να ορίσει μέσω των Blueprints τις συσκευές που επιθυμεί στην τοπολογία του. Για τις συσκευές αυτές μπορούν να οριστούν συνδέσεις (links) με διάφορες δυνατότητες δικτύου και ποιότητας υπηρεσίας (QoS). Τα ορίσματα αυτά πρέπει να υπολογίζονται από τον χρήστη και να ανανεώνονται χειροκίνητα κάθε φορά που αυτά χρειάζεται να ενημερωθούν, λόγω μετακίνησης. Σε αυτό το σημείο έρχεται η επέκταση PPComm για να λύσει το πιο πάνω πρόβλημα δημιουργώντας ένα σύστημα συντεταγμένων, μέσω του οποίου μπορεί να υπολογιστεί η θέση, η συνδεσιμότητα και η ποιότητα υπηρεσίας των συσκευών της τοπολογίας. Ακολούθως, ενημερώνει αυτόματα τον εξομοιωτή του Fogify. Επίσης, η επέκταση επιτρέπει στον χρήστη να δημιουργήσει τροχιές (trajectories), οι οποίες είναι μετακινήσεις σε προκαθορισμένο χρόνο που θα γίνουν για κάθε συσκευή. Με αυτές τις προσθήκες, ανοίγονται νέα παράθυρα στα πιθανά σενάρια που μπορούν να δημιουργήσουν οι χρήστες του Fogify προκειμένου να εξετάσουν τις δικές τους ιδέες μοντελοποίησης.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Fog Computing

Το Fog Computing [1] (ή αλλιώς Edge Computing) είναι η αρχιτεκτονική η οποία χρησιμοποιεί edge συσκευές (δηλ. βρίσκονται στις άκρες του δικτύου) για την διεξαγωγή ενός σημαντικού ποσοστού υπολογισμών, αποθήκευσης και μεταφοράς δεδομένων σε ένα δίκτυο καθώς και για δρομολόγηση των πιο πάνω δεδομένων στο διαδίκτυο.

Η πιο πάνω αρχιτεκτονική προορίζεται για κατανεμημένους υπολογιστές όπου μεγάλος αριθμός από περιφερειακές συσκευές είναι συνδεδεμένες μεταξύ τους στο cloud. Η λέξη «fog» αναφέρεται στις ιδιότητες που προσφέρει η τεχνολογία αυτή, οι οποίες μοιάζουν με αυτές του cloud αλλά είναι πιο κοντά στο «έδαφος», όπως για παράδειγμα οι συσκευές Internet of Things (IoT). Πολλές από αυτές τις συσκευές παράγουν ογκώδη αριθμό ακατέργαστων δεδομένων από διάφορους αισθητήρες και τα δεδομένα αυτά πρέπει να τύχουν

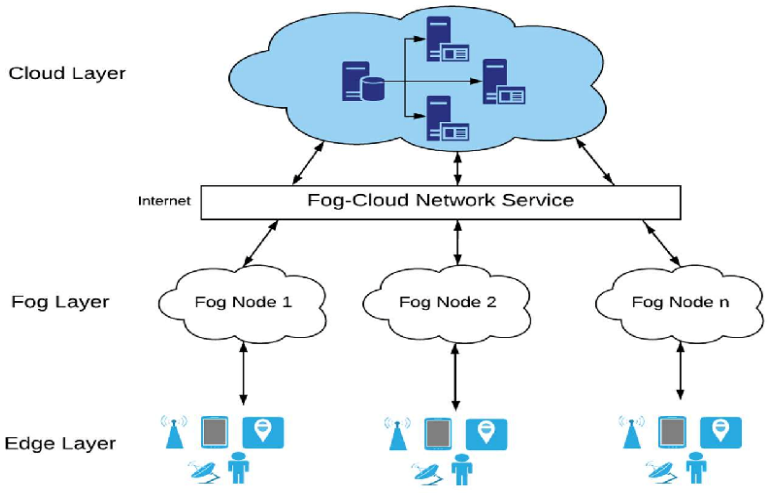
επεξεργασίας για την εξαγωγή αποφάσεων και συμπερασμάτων. Η ιδέα του fog computing είναι να γίνετε όσο το δυνατόν περισσότερη επεξεργασία δεδομένων στις ίδιες συσκευές ή σε συσκευές υπολογισμού που βρίσκονται σε κοντινή απόσταση από τις συσκευές συλλογής δεδομένων. Αυτό έχει ως αποτέλεσμα να μην χρειάζεται να σταλούν ακατέργαστα δεδομένα για επεξεργασία στους διακομιστές (servers) και να στέλνονται μόνο τα απαραίτητα και επεξεργασμένα δεδομένα. Με αυτό το τρόπο οι απαιτήσεις εύρους ζώνης (bandwidth) μειώνονται αισθητά. Ταυτόχρονα, ένα επιπρόσθετο πλεονέκτημα είναι ότι τα επεξεργασμένα δεδομένα μιας συσκευής, πιθανόν να χρειάζονται μόνο από την ίδια συσκευή και έτσι αποφεύγεται η απομακρυσμένη επεξεργασία. Ως αποτέλεσμα, ελαχιστοποιείται η καθυστέρηση (latency) αποστολής δεδομένων και απόκρισης. Η ιδέα αυτή δεν είναι εντελώς καινούρια. Κάτι παρόμοιο χρησιμοποιείται ήδη εδώ και πολύ καιρό σε συσκευές ειδικής χρήσης (π.χ. τσιπ επεξεργασίας σημάτων που εκτελούν Fast Fourier Transforms [2]) για να μειώσουν τον φόρτο εργασίας των επεξεργαστών και την καθυστέρηση.

Στο επίπεδο δεδομένων, το Fog computing επιτρέπει στις υπολογιστικές υπηρεσίες να βρίσκονται στην άκρη του δικτύου (δηλ. στις συσκευές edge), παρά σε διακομιστές που βρίσκονται σε κέντρα διακομιστών (data centers). Σε σύγκριση με το Cloud computing, το Fog computing δίνει έμφαση στα ακόλουθα σημεία:

i. Επεξεργασία δεδομένων σε κοντινή απόσταση (close proximity) από τον τελικό χρήστη.
ii. Μείωση του κόστους λειτουργίας της προσφερόμενης υπηρεσίας.
iii. Συμμόρφωση με πολιτικές ασφαλείας των δεδομένων.
iv. Εκμετάλλευση όλων των διαθέσιμων υπολογιστικών πόρων.

v. Πυκνή γεωγραφική κατανομή των υπηρεσιών στις συσκευές edge και επίγνωση της κατάστασης, σε ό,τι αφορά τους υπολογιστικούς πόρους των συσκευών.
vi. Μείωση της καθυστέρησης (latency) και εξοικονόμηση δεδομένων εύρους ζώνης (bandwidth) για την επίτευξη καλύτερης ποιότητας υπηρεσίας (QoS).
vii. Ανάλυση των άκρων του δικτύου και εξαγωγή πληροφοριών με σκοπό την βελτίωση της εμπειρίας του χρήστη.
viii. Πλεονασμός (redundancy) σε περίπτωση σφάλματος της υπηρεσίας ή βλάβης κάποιας συσκευής.

Τα δίκτυα Fog υποστηρίζουν την ιδέα του Internet of Things, στην οποία οι περισσότερες συσκευές που χρησιμοποιούνται από τους ανθρώπους σε καθημερινή βάση, συνδέονται μεταξύ τους. Μερικά παραδείγματα είναι τα κινητά τηλέφωνα, οι φορητές συσκευές παρακολούθησης υγείας, τα «έξυπνα» ρολόγια, τα οχήματα με δυνατότητα σύνδεσης στο διαδίκτυο και συσκευές επαυξημένης πραγματικότητας (augmented reality) όπως το Google Glass. Συσκευές όπως οι πιο πάνω, συνήθως έχουν περιορισμένες υπολογιστικές δυνατότητες και έτσι δεν είναι εύκολη η κρυπτογράφηση των δεδομένων. Στη περίπτωση αυτή, ένας κόμβος Fog (Fog Node) μπορεί να παρέχει ασφάλεια αναλαμβάνοντας τους υπολογισμούς κρυπτογράφησης.



Εικόνα 1: Αρχιτεκτονική Fog Computing.

1.2 Fogify (A Fog Computing Emulator Framework)

Το Fogify [3] είναι ένα framework εξομοίωσης δικτύων Fog το οποίο ευκολύνει την δημιουργία μοντέλων, την ανάπτυξη/deployment και πειραματισμό Fog testbeds. Πιο συγκεκριμένα, παρέχει τα εργαλεία για την μοντελοποίηση σύνθετων Fog τοπολογιών, των οποίων οι κόμβοι μπορούν να αποτελούνται από ετερογενείς πόρους, δυνατότητες δικτύου και κριτήρια ποιότητας υπηρεσίας (QoS). Ακολούθως, επιτρέπει την ανάπτυξη και deployment της μοντελοποιημένης διαμόρφωσης με τη βοήθεια του Docker σε τοπικό ή cloud περιβάλλον. Τέλος, δίνει την δυνατότητα να γίνει πειραματισμός, μέτρηση και αξιολόγηση του deployment εισάγοντας σφάλματα και προσαρμόζοντας την διαμόρφωση της μοντελοποίησης κατά τον χρόνο εκτέλεσης. Συνδυάζοντας τα πιο πάνω, οι χρήστες του Fogify μπορούν να δημιουργήσουν αμέτρητα πιθανά σενάρια προκειμένου να εξετάσουν κατά πόσο λειτουργούν, αφού μέσω του εξομοιωτή θα αποκαλυφθούν οι περιορισμοί μιας υπηρεσίας πριν αυτή παρουσιαστεί στο κοινό. Το framework δημιουργήθηκε από τα μέλη του Laboratory of Internet Computing (LInC) του Πανεπιστημίου Κύπρου.

Το Fogify παρέχει τις ακόλουθες λειτουργίες και δυνατότητες:

i. Εξομοίωση Fog κόμβων με ετερογενείς πόρους και δυνατότητες.
ii. Ρύθμιση της ποιότητας σύνδεσης των κόμβων, όπως τη καθυστέρηση δικτύου, την ταχύτητα εύρους ζώνης, το ποσοστό σφάλματος κλπ.
iii. Αναπαραγωγή ιχνών (traces) για συνδέσεις κόμβο-προς-κόμβο και κόμβο-προς-δίκτυο.
iv. Αλλαγή της τοπολογίας κατά τον χρόνο εκτέλεσης, με την εισαγωγή σφαλμάτων, την αλλαγή της ποιότητας δικτύου/συνδέσεων και με την εισαγωγή φόρτου εργασίας και υπολογισμών στις υπηρεσίες που τρέχουν οι κόμβοι.

v. Επεκτασιμότητα τοπολογιών, αφού μπορεί να μοντελοποιηθεί μια τοπολογία περιορισμένου αριθμού κόμβων που να εκτελείται σε ένα μόνο υπολογιστή μέχρι μια τοπολογία χιλιάδων κόμβων που να εκτελείται σε ολόκληρο σύμπλεγμα υπολογιστών (cluster) με τη χρήση του Docker swarm.
vi. Συλλογή, διαχείριση και επεξεργασία μετρήσεων από τους Fog κόμβους και τις συνδέσεις δικτύου της εξομοίωσης καθώς και λήψη πληροφοριών από το επίπεδο εφαρμογών (λογισμικό που τρέχουν οι κόμβοι - Services).
vii. Γρήγορη ανάπτυξη και deployment, αφού τα λειτουργικά πρωτότυπα μιας εφαρμογής γραμμένης σε docker-compose, δεν απαιτούν τροποποίηση προκειμένου να εκτελεστούν στο Fogify.

1.3 Fogify – Τρόπος χρήσης

Η διαδικασία μοντελοποίησης ξεκινάει με τον χρήστη να ρυθμίζει το αρχείο docker-compose μιας εφαρμογής IoT, και στη συνέχεια να το επεκτείνει συμπεριλαμβάνοντας στο αρχείο αυτό, το μοντέλο του Fogify. Το μοντέλο του Fogify αποτελείται από τα ακόλουθα:

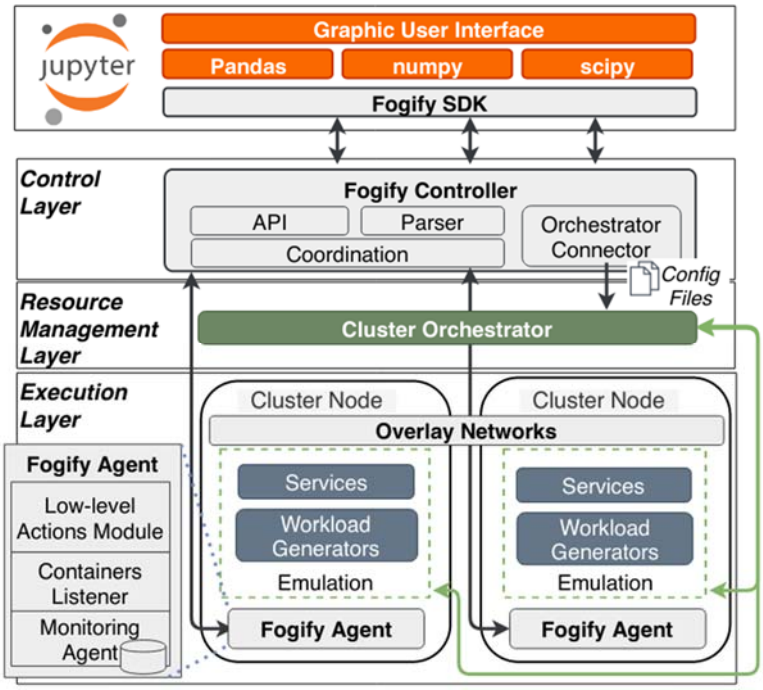
i. Πρότυπα Fog, τα οποία επιτρέπουν την περιγραφή των υπηρεσιών (Services), των κόμβων (Nodes) και των δικτύων (Networks).
ii. Τοπολογία Fog, η οποία επιτρέπει τον προσδιορισμό των Blueprints. Τα Blueprints αναπαριστούν κάθε συσκευή στην εξομοίωση, η οποία αποτελείται από τον συνδυασμό ενός κόμβου, μιας υπηρεσίας, πολλαπλών δικτύων, αριθμού διπλοτύπων και μιας ετικέτας. Οι υπηρεσίες κληρονομούνται από το docker-compose ενώ τα υπόλοιπα παρέχονται από το μοντέλο x-fogify που δημιουργεί ο χρήστης στο docker-compose. Με αυτό το τρόπο, ο χρήστης μπορεί να αναπτύξει την εφαρμογή του με τα γνωστά docker constructs και ταυτόχρονα να έχει την πρόσθετη

λειτουργικότητα που παρέχει το Fogify, χωρίς να επηρεάζεται η φορητότητα της εφαρμογής.

Όταν η περιγραφή του μοντέλου είναι έτοιμη, ο χρήστης κάνει deploy την εφαρμογή του με την χρήση του FogifySDK μέσω ενός Jupyter notebook, από την περιγραφή που λαμβάνει το Controller του Fogify. Εάν το Controller δεν ανιχνεύσει κάποιο σφάλμα, δημιουργεί τις εξομοιωμένες συσκευές και το πλέγμα δικτύων μεταξύ των συσκευών. Ακολούθως δημιουργεί τις υπηρεσίες και μεταδίδει στους Fogify Agents τους περιορισμούς του δικτύου, εάν υπάρχουν. Πιο συγκεκριμένα, ο Controller μεταφράζει τις προδιαγραφές του μοντέλου σε εκτελέσιμο κώδικα του Cluster Orchestrator, ο οποίος διασφαλίζει την εγκατάσταση των containerized υπηρεσιών στο περιβάλλον εξομοίωσης. Οι Fogify Agents οι οποίοι τοποθετούνται σε κάθε cluster node, δημιουργούν ένα Application Programming Interface (API), το οποίο δέχεται αιτήματα από τον Controller, εφαρμόζουν τις ρυθμίσεις ποιότητας υπηρεσίας (QoS) των δικτύων και παρακολουθούν τις συσκευές της εξομοίωσης.

Σε ένα deployment εξομοίωσης που βρίσκεται σε εκτέλεση, το Fogify επιτρέπει στο χρήστη να εκτελέσει ενέργειες (Actions) και σενάρια "what-if" (ακολουθίες ενεργειών) στις υπηρεσίες, όπως εισαγωγές σφαλμάτων ad-hoc και αλλαγές τοπολογίας. Οι ενέργειες και τα σενάρια γράφονται από τον χρήστη ακολουθώντας το Fogify Runtime Evaluation Model. Όταν εκτελεστεί μια ενέργεια ή ένα σενάριο, ο Controller του Fogify συντονίζει την εκτέλεση με τη βοήθεια του Cluster Orchestrator και τα αντίστοιχα Fogify Agents. Επιπλέον, το Fogify καταγράφει την απόδοση των υπηρεσιών μέσω της μονάδας παρακολούθησης που περιέχει ο κάθε Fogify Agent.

Τέλος, το Fogify παρέχει ένα διαδραστικό εργαλείο ανάλυσης της εξομοίωσης μέσω του FogifySDK και του Jupyter notebook. Συγκεκριμένα, έχει προ εγκατασταθεί η βιβλιοθήκη FogifySDK στο Jupyter έτσι ώστε ο χρήστης να μπορεί να κάνει deploy και undeploy μια τοπολογία Fog, να εκτελεί ενέργειες και σενάρια, και το πιο σημαντικό, να ανακτήσει μετρήσεις απόδοσης του χρόνου εκτέλεσης. Για την ανάκτηση των προαναφερθέντων το Fogify αποθηκεύει τις μετρήσεις σε μια δομή δεδομένων στη μνήμη (panda's data frame). Η δομή αυτή παρέχει μεθόδους ανάλυσης δεδομένων, οι οποίες περιλαμβάνουν την δημιουργία γραφικών παραστάσεων και στατιστικών πινάκων. Εκτός από τη δομή αυτή, το FogifySDK έχει επεκταθεί με λειτουργίες που παρέχουν γραφικές παραστάσεις που απεικονίζουν και εξηγούν τις επιπτώσεις των ενεργειών και σεναρίων στην απόδοση της εφαρμογής. Συνδυάζοντας τα πιο πάνω, ο χρήστης μπορεί να εξάγει πολύ χρήσιμες πληροφορίες σχετικά με την ποιότητα υπηρεσίας, το κόστος και προγνωστική ανάλυση, η οποία είναι εξαιρετικά χρήσιμη για την αποφυγή προβλημάτων.



Εικόνα 2: Επισκόπηση του Fogify

1.4 Docker

Το Docker [4] είναι μια πλατφόρμα η οποία επιτρέπει την ανάπτυξη και εκτέλεση εφαρμογών, η οποία χρησιμοποιεί virtualization στο επίπεδο του λειτουργικού συστήματος για την διανομή λογισμικού σε πακέτα που ονομάζονται Containers. Το λογισμικό στο οποίο εγκαθίστανται τα Containers ονομάζεται Docker Engine. Θα αναφερθούμε περιληπτικά στην τεχνολογία του Docker και σε ορισμένα εργαλεία του, κυρίως σε αυτά που χρησιμοποιούνται από το Fogify για να παρέχει τις δυνατότητες που αναφέραμε πιο πάνω, όπως το Docker Compose και το Docker Swarm.

Αρχικά, να αναφέρουμε ότι τα Containers [5] είναι απομονωμένα το ένα από το άλλο και το κάθε ένα περιλαμβάνει το δικό του λογισμικό, βιβλιοθήκες και αρχεία διαμόρφωσης. Μπορούν ωστόσο να επικοινωνήσουν μεταξύ τους μέσω καθορισμένων καναλιών. Το μεγαλύτερο πλεονέκτημα των Containers είναι η χρήση λιγότερων πόρων σε σύγκριση με τις συνηθισμένες εικονικές μηχανές (virtual machines). Αυτό συμβαίνει διότι τα Containers μοιράζονται τις υπηρεσίες από το ίδιο kernel του λειτουργικού συστήματος.

Το Docker Compose [6] είναι ένα εργαλείο που επιτρέπει τον καθορισμό και την εκτέλεση εφαρμογών Docker με πολλαπλά containers. Χρησιμοποιεί αρχεία YAML για την διαμόρφωση των υπηρεσιών και εκτελεί την διαδικασία δημιουργίας και εκκίνησης όλων των containers με μία μόνο εντολή. Το βοηθητικό πρόγραμμα docker-compose CLI επιτρέπει στους χρήστες να εκτελούν εντολές σε πολλά containers ταυτόχρονα. Με τις εντολές αυτές ο χρήστης μπορεί να δημιουργήσει docker images, να κάνει κλιμάκωση (scaling) των containers, να εκτελέσει containers που είναι σταματημένα και άλλα πολλά. Το αρχείο docker-compose.yaml χρησιμοποιείται για τον καθορισμό των

υπηρεσιών μιας εφαρμογής και περιλαμβάνει διάφορες επιλογές διαμόρφωσης. Αυτό είναι το πρώτο εργαλείο που εκμεταλλεύεται το Fogify, αφού χρησιμοποιεί το docker-compose.yaml για τον καθορισμό του μοντέλου του Fogify και την χρήση των services που δηλώνονται σε αυτό.

Το Docker Swarm [7] είναι ένα ακόμα εργαλείο του Docker, το οποίο παρέχει την λειτουργία ομαδοποίησης των Containers (δημιουργία σμήνους, swarm). Η λειτουργία αυτή επιτρέπει στον χρήστη να δημιουργήσει muti-host deployments από Containers. Τα Containers αυτά μπορούν να τρέχουν σε διαφορετικές μηχανές του Swarm Cluster. Το βοηθητικό πρόγραμμα docker swarm CLI, επιτρέπει στους χρήστες να εκτελούν Swarm containers, να δημιουργούν tokens εντοπισμού για την ομαδοποίηση κόμβων στα swarms και πολλά άλλα. Το βοηθητικό πρόγραμμα docker node CLI επιτρέπει στους χρήστες να εκτελούν διάφορες εντολές για την διαχείριση κόμβων σε ένα swarm, όπως την περιήγηση στους κόμβους, καθώς και την ενημέρωση και αφαίρεση τους από κάποιο swarm. Το Docker Swarm είναι ακόμα ένα εργαλείο το οποίο δίνει τη δυνατότητα στους χρήστες του Fogify να εκμεταλλευτούν, εάν κρίνετε απαραίτητο από τις ανάγκες του προγράμματος τους.

Κεφάλαιο 2

Επέκταση PPComm

2.1 Εισαγωγή

Η επέκταση PPComm (**P**oint-to-**P**oint **Comm**unication) είναι μια επέκταση του εξομοιωτή Fogify (A Fog Computing Emulator Framework) και είναι γραμμένη στη γλώσσα προγραμματισμού Python. Δημιουργήθηκε το 2022 ως μέρος της εκπλήρωσης της Ατομικής Διπλωματικής Εργασίας μου. Η επέκταση εκμεταλλεύεται το Software Development Kit (SDK) του Fogify για την προσθήκη ορισμένων καινούριων δυνατοτήτων. Οι δυνατότητες αυτές υλοποιήθηκαν με τέτοιο τρόπο που να διευκολύνουν τον χρήστη όσο το δυνατόν περισσότερο. Πιο κάτω θα αναφερθούμε στο πρόβλημα και την λύση που προσφέρει η επέκταση PPComm.

2.2 Πρόβλημα

Στο υφιστάμενο σύστημα του Fogify, ο χρήστης μπορεί να ορίσει μέσω των Blueprints όσες συσκευές θέλει στην τοπολογία του και η κάθε συσκευή περιέχει μια λίστα με δίκτυα τα οποία επικοινωνεί. Για τις συσκευές αυτές μπορούν να οριστούν συνδέσεις (links), στις οποίες μπορούν να καθοριστούν οι δυνατότητες του δικτύου και η ποιότητα υπηρεσίας (QoS), όπως ο ρυθμός αποστολής δεδομένων (data rate), το ποσοστό χαμένων πακέτων (packet drop), και η καθυστέρηση (latency/delay) μεταξύ δύο συσκευών. Οι point-to-point συνδέσεις μεταξύ των συσκευών πρέπει να καθοριστούν μία προς μία από τον χρήστη και τα ορίσματα των συνδέσεων αυτών χρειάζεται να υπολογίζονται χειροκίνητα. Σε περίπτωση που θέλει να τα διαφοροποιήσει πρέπει να το κάνει μέσω του FogifySDK, όπου ξανά πρέπει να εισάγει χειροκίνητα τις νέες τιμές των ορισμάτων αυτών.

Για παράδειγμα, σε ένα σενάριο όπου υπάρχουν δεκάδες συσκευές (π.χ. drones), ο αριθμός των μεταξύ τους συνδέσεων μπορεί να είναι μεγαλύτερος από 100. Με την υφιστάμενη υλοποίηση ο χρήστης πρέπει να εισάγει όλες αυτές τις point-to-point συνδέσεις και να υπολογίσει τα ορίσματα κάθε σύνδεσης. Θα χρειαστούν χιλιάδες υπολογισμοί έτσι ώστε να βρεθεί το latency και bandwidth όλων των point-to-point συνδέσεων. Όλα τα πιο πάνω θα χρειαστεί να γίνουν για την αρχικοποίηση μιας μοντελοποίησης. Τι γίνετε όταν κατά την εκτέλεση της εξομοίωσης ο χρήστης χρειάζεται να αλλάξει την τοποθεσία κάποιων συσκευών; Όλα τα πιο πάνω είναι πλέον άχρηστα και πρέπει να υπολογιστούν ξανά. Αντιλαμβάνεστε ότι το πιο πάνω σενάριο κάνει την χρήση του Fogify δύσκολη, μη παραγωγική και εξαιρετικά χρονοβόρα.

2.3 Λύση

Η επέκταση PPComm λύνει το πιο πάνω πρόβλημα δημιουργώντας ένα σύστημα συντεταγμένων, μέσω του οποίου μπορεί να υπολογιστεί η θέση, η συνδεσιμότητα και η ποιότητα υπηρεσίας των συσκευών της τοπολογίας. Ο χρήστης μπορεί να επιλέξει ένα από τους δύο διαθέσιμους τύπους συστημάτων που προσφέρει η επέκταση κατά την μοντελοποίηση της τοπολογίας. Ο πρώτος τύπος είναι το Τρισδιάστατο Σύστημα Συντεταγμένων (Three-Dimensional Coordinate System) όπου ο χρήστης ορίζει το x, y και z για να προσδιορίσει την θέση κάθε συσκευής στην τοπολογία. Ο δεύτερος τύπος είναι το Γεωγραφικό Σύστημα συντεταγμένων (Geographic Coordinate System) όπου ο χρήστης ορίζει το γεωγραφικό πλάτος (latitude) και το γεωγραφικό μήκος (longitude) για τον προσδιορισμό της θέσης κάθε συσκευής. Στην ενότητα 2.4 υπάρχουν περισσότερες λεπτομέρειες σχετικά με το προτεινόμενο σύστημα συντεταγμένων αναλόγως της μοντελοποίησης.

Ακολούθως, ο χρήστης ορίζει μια λίστα προδιαγραφών για την κάθε συσκευή η οποία περιλαμβάνει τα ακόλουθα: carrier frequency, bandwidth, transmitter power, RU antenna gain, UE antenna gain και UE noise figure. Έχοντας όλα τα πιο πάνω δεδομένα στη διάθεση της, η επέκταση αναλαμβάνει τον υπολογισμό πιθανής συνδεσιμότητας μεταξύ των συσκευών και τον υπολογισμό των δυνατοτήτων της σύνδεσης, όπως καθυστέρηση (latency - milliseconds) και ρυθμό αποστολής δεδομένων (data rate – bits per second). Στη συνέχεια μετατρέπει την τοπολογία αυτή σε μοντέλο του Fogify και δημιουργεί ένα καινούριο docker-compose.yaml το οποίο στέλνει στον Fogify Controller μέσω του FogifySDK.

Μια ακόμη πολύ σημαντική δυνατότητα που προσφέρει η επέκταση PPComm είναι ο ορισμός διαφόρων trajectories (τροχιές). Η δυνατότητα αυτή επιτρέπει την μαζική αλλαγή συντεταγμένων των συσκευών σε καθορισμένο χρόνο κατά τον χρόνο εκτέλεσης της εξομοίωσης. Αυτό συνεπάγεται με την ενημέρωση των δυνατοτήτων συνδεσιμότητας και ποιότητας υπηρεσίας (QoS) των συσκευών. Με την εκτέλεση ενός trajectory η τοπολογία της εξομοίωσης ανανεώνεται με τα καινούρια δεδομένα και μπορεί να εξομοιωθεί η κίνηση των συσκευών στον χώρο. Περισσότερες λεπτομέρειες για τον τρόπο λειτουργίας των trajectories θα δοθούν στη συνέχεια.

Σε αυτό το σημείο, ο χρήστης μπορεί να κάνει deploy στον εξομοιωτή του Fogify την καινούρια τοπολογία που δημιούργησε η επέκταση PPComm και να χρησιμοποιήσει όλες τις δυνατότητες που προσφέρει το Fogify, όπως την εκτέλεση ενεργειών και σεναρίων, καθώς και ανάκτηση των μετρήσεων που προσφέρονται. Επίσης στη διάθεση του χρήστη είναι οι νέες δυνατότητες που προσφέρει το PPComm όπως η εκτέλεση trajectories και η μεμονωμένη αλλαγή της θέσης μιας συσκευής στη τοπολογία.

2.4 Τρόπος χρήσης

Η διαδικασία μοντελοποίησης ξεκινάει με τον χρήστη να δημιουργεί ένα αρχείο docker-compose στο οποίο τοποθετεί τις υπηρεσίες (services) μιας εφαρμογής IoT. Ακολούθως, ο χρήστης το επεκτείνει με το μοντέλο του Fogify (x-fogify) το οποίο περιλαμβάνει τους κόμβους (nodes), τα δίκτυα (networks) και την τοπολογία (topology). Στην τοπολογία του Fogify, ο χρήστης μπορεί να προσθέσει συσκευές στις οποίες δεν απαιτείται point-to-point επικοινωνία, εφόσον υπάρχει τέτοια απαίτηση στη μοντελοποίηση. Οι συσκευές οι οποίες

απαιτούν point-to-point επικοινωνία, θα πρέπει να προστεθούν στο μοντέλο της επέκτασης PPComm. Στα δίκτυα (networks) είναι απαραίτητη η προσθήκη ενός δικτύου το οποίο θα χρησιμοποιηθεί από την επέκταση PPComm. Το δίκτυο αυτό θα χρησιμοποιηθεί για την δημιουργία point-to-point συνδέσεων μεταξύ των συσκευών. Το δίκτυο αυτό δεν δημιουργείται αυτόματα από την επέκταση για να δίνεται η δυνατότητα στον χρήστη να διαμορφώσει τα ορίσματα που προσφέρει το Fogify, όπως το μέγιστο bandwidth (data rate) του συγκεκριμένου δικτύου. Περισσότερες λεπτομέρειες για τις ρυθμίσεις μοντελοποίησης του Fogify μπορείτε να βρείτε στην επίσημη σελίδα του.

Στη συνέχεια, ο χρήστης επεκτείνει το docker-compose, προσθέτοντας το μοντέλο της επέκτασης PPComm (x-ppcomm). Το μοντέλο αυτό περιλαμβάνει τρεις ενότητες. Η πρώτη ενότητα είναι το config, η οποία περιλαμβάνει τον τύπο συστήματος συντεταγμένων, το δίκτυο του Fogify στο οποίο θα προστεθούν οι συνδέσεις, την επιλογή εύρεσης μονοπατιού για τη δημιουργία σύνδεσης μεταξύ συσκευών και την ρύθμιση αλλαγής του ελάχιστου signal-to-noise ratio για την δημιουργία σύνδεσης. Η δεύτερη ενότητα του μοντέλου είναι τα nodes, όπου δηλώνονται όλες οι συσκευές συμπεριλαμβανομένων της τοποθεσίας και των προδιαγραφών τους. Η τρίτη ενότητα του μοντέλου, η οποία είναι προαιρετική, είναι τα trajectories, όπου δηλώνονται όλα τα σενάρια αλλαγής συντεταγμένων των συσκευών σε προκαθορισμένους χρόνους κατά τον χρόνο εκτέλεσης της εξομοίωσης.

```
…
x-ppcomm:
  config:
    coord_type: latlong # latlong / xyz
    network: ppc
    find_path: true     # allow nodes to communicate through other nodes?(true/false)
    minimum_snr: 10     # minimum signal-to-noise ratio in dB.
  nodes:
    …
  trajectories:
    …
```

Εικόνα 3: Μοντέλο PPComm – Ενότητα config

config:

i. **Σύστημα συντεταγμένων (coord_type)**

Ο χρήστης μπορεί να επιλέξει μεταξύ των επιλογών: latlong, xyz.

Η επιλογή «latlong» ενεργοποιεί την χρήση του Γεωγραφικού Συστήματος Συντεταγμένων, ενώ η επιλογή «xyz» ενεργοποιεί την χρήση του Τρισδιάστατου Συστήματος Συντεταγμένων. Στην περίπτωση που ο χρήστης θέλει να χαρτογραφήσει ένα δίκτυο με αρχικό σημείο αναφοράς και αποστάσεις μεταξύ των συσκευών τις οποίες γνωρίζει, τότε προτείνεται η χρήση του τρισδιάστατου συστήματος συντεταγμένων. Διαφορετικά, σε περίπτωση που γνωρίζει την γεωγραφική τοποθεσία των συσκευών, ή γνωρίζει τις γεωγραφικές τοποθεσίες στις οποίες θα μετακινηθούν οι συσκευές, τότε προτείνεται η χρήση του γεωγραφικού συστήματος συντεταγμένων.

ii. **Δίκτυο (network)**

Ο χρήστης επιλέγει ένα από τα δίκτυα που δήλωσε στο μοντέλο του Fogify, το οποίο θα χρησιμοποιηθεί για να προστεθούν οι συνδέσεις των συσκευών. Το δίκτυο αυτό δεν δημιουργείται αυτόματα από την επέκταση για να δίνεται η δυνατότητα στον χρήστη να διαμορφώσει τα ορίσματα που προσφέρει το Fogify, όπως το μέγιστο bandwidth (data rate) του συγκεκριμένου δικτύου.

iii. **Εύρεση μονοπατιού για δημιουργία συνδέσεων (find_path)**

Ο χρήστης μπορεί να επιλέξει μεταξύ των επιλογών: true, false.

Σε περίπτωση που δύο συσκευές δεν έχουν συνδεσιμότητα μεταξύ τους, αλλά υπάρχουν ενδιάμεσοι κόμβοι με τους οποίους επικοινωνούν, τότε επιλέγοντας true, το PPComm αναλαμβάνει να βρει το μονοπάτι με τη

χαμηλότερη καθυστέρηση (latency) μεταξύ των δύο συσκευών, εφόσον υπάρχει.

iv. Ελάχιστο signal-to-noise ratio (minimum_snr)

Η επιλογή αυτή επιτρέπει στον χρήστη να ορίσει την ελάχιστη τιμή του signal-to-noise ratio προκειμένου να υπάρξει σύνδεση μεταξύ δύο συσκευών. Η επιλογή αυτή είναι διαθέσιμη στο χρήστη για περισσότερη ευελιξία.

- 5dB μέχρι 10dB: είναι πιο κάτω από το ελάχιστο επίπεδο για τη δημιουργία σύνδεσης, επειδή το επίπεδο θορύβου (noise) είναι τόσο ψηλό που δεν μπορούν να ληφθούν χρήσιμες πληροφορίες.
- 10dB μέχρι 15dB: είναι το ελάχιστο αποδεκτό για την δημιουργία μιας αναξιόπιστης σύνδεσης.
- 15dB μέχρι 25dB: θεωρείται το ελάχιστο αποδεκτό για την δημιουργία μιας κακής σύνδεσης.
- 25dB μέχρι 40dB: θεωρείται καλό για την δημιουργία σύνδεσης.
- 41dB και ψηλότερο: θεωρείται εξαιρετικό για την δημιουργία σύνδεσης.

```
x-ppcomm:
  config:
    …
  nodes:
    - name: drone_0
      fogify:
        node: drone-node
        service: mec-svc-1
      coords:
        latitude: 0
        longitude: 0
      specs:
        carrier_frequency: 28   #ghz
        bandwidth: 0.100        #ghz
        transmit_power: 30      #dBm
        ru_antenna_gain: 6      #dbi
        ue_antenna_gain: 6      #dbi
        ue_noise_figure: 7.8    #dB
    - name: drone_1
      fogify:
        node: drone-node
```

```
      service: mec-svc-1
    coords:
      latitude: 0.00006
      longitude: 0.00006
    specs:
      carrier_frequency: 28   #ghz
      bandwidth: 0.100        #ghz
      transmit_power: 30      #dBm
      ru_antenna_gain: 6      #dbi
      ue_antenna_gain: 6      #dbi
      ue_noise_figure: 7.8    #dB
  - name: drone_2
    fogify:
      node: drone-node
      service: mec-svc-1
    coords:
      latitude: 0.0003875
      longitude: 0.0003875
    specs:
      carrier_frequency: 28   #ghz
      bandwidth: 0.100        #ghz
      transmit_power: 30      #dBm
      ru_antenna_gain: 6      #dbi
      ue_antenna_gain: 6      #dbi
      ue_noise_figure: 7.8    #dB
trajectories:
  …
```

Εικόνα 4: Μοντέλο PPComm – Ενότητα nodes

nodes:

Στην *Εικόνα 4* παρουσιάζετε ο τρόπος με τον οποίο ο χρήστης μπορεί να ορίσει την λίστα συσκευών της τοπολογίας. Ο χρήστης δίνει τα ακόλουθα δεδομένα για την δημιουργία συσκευής:

i. Όνομα συσκευής
ii. Όνομα του κόμβου (node) που θα χρησιμοποιήσει η συσκευή.
    Είναι απαραίτητο το node να βρίσκεται δηλωμένο στο μοντέλο του Fogify.
iii. Όνομα της υπηρεσίας (service) που θα τρέξει η συσκευή μέσω του Fogify.
    Είναι απαραίτητο το service να βρίσκεται δηλωμένο στο docker-compose.
iv. Συντεταγμένες συσκευής
v. Προδιαγραφές συσκευής
    - Carrier frequency (GHz): Είναι η συχνότητα που χρησιμοποιεί ο κατασκευαστής της συσκευής για την μετάδοση σημάτων (signals).

- Bandwidth (GHz): Είναι η συχνότητα που χρησιμοποιείται για τον καθορισμό της ποσότητας δεδομένων που μπορεί να λάβει/στείλει η συσκευή ανά δευτερόλεπτο.
- Transmitter power (dBm): Είναι η ποσότητα ηλεκτρικής ισχύος που χρησιμοποιείται για την μετάδοση σημάτων από την συσκευή. Η προτεινόμενη τιμή κυμαίνεται μεταξύ 14-18dBm.
- RU antenna gain (dBi): Καθορίζει πόσο καλά η κεραία μετατρέπει την ηλεκτρική ισχύ εισόδου σε ραδιοκύματα που κατευθύνονται σε μια καθορισμένη κατεύθυνση.
- UE antenna gain (dBi): Καθορίζει πόσο καλά η κεραία μετατρέπει τα ραδιοκύματα που φτάνουν από μια καθορισμένη κατεύθυνση σε ηλεκτρική ισχύ.
- UE noise figure (dB): Καθορίζει την ποσότητα θορύβου που προσθέτει η συσκευή στη μετάδοση σημάτων. Όσο χαμηλότερη είναι η τιμή του θορύβου, τόσο καλύτερη είναι η απόδοση.

```yaml
x-ppcomm:
  config:
    …
  nodes:
    …
  trajectories:
    - name: tran_1
      changes:
        - time: 1
          nodes:
            - name: drone_2
              coords:
                latitude: 0.005
                longitude: 0.005
            - name: drone_0
              coords:
                latitude: 0.005
                longitude: 0.005
        - time: 2
          nodes:
            - name: drone_1
              coords:
                latitude: 0.005
```

```
              longitude: 0.005
  - name: tran_2
    changes:
      - time: 5
        nodes:
          - name: drone_5
            coords:
              latitude: 0.00025
              longitude: 0.00051
          - name:  drone_0
            coords:
              latitude: 0.005
              longitude: 0.005
      - time: 10
        nodes:
          - name: drone_1
            coords:
              latitude: 0.002
              longitude: 0.003
```

Εικόνα 5: Μοντέλο Fogify – Ενότητα trajectories

trajectories:

Στην *Εικόνα 5* παρουσιάζετε ο τρόπος με τον οποίο ο χρήστης μπορεί να ορίσει την λίστα των trajectories, τα οποία θα είναι διαθέσιμα κατά τον χρόνο εκτέλεσης της εξομοίωσης. Κάθε trajectory χρειάζεται ένα όνομα (name) και την λίστα των αλλαγών (changes) που θα συμβούν ανά χρονική στιγμή. Κάθε αντικείμενο στη λίστα αλλαγών πρέπει να έχει την χρονική στιγμή (time) που θα γίνουν οι αλλαγές συντεταγμένων και μια λίστα από συσκευές (nodes) που θα μετακινηθούν. Έτσι μπορούν να δημιουργηθούν αμέτρητα σενάρια μετακίνησης των συσκευών ανάλογα με τις ανάγκες του χρήστη.

Σε αυτό το σημείο, ολοκληρώνεται η μοντελοποίηση του PPComm. Επόμενο βήμα είναι η εκτέλεση της επέκτασης PPComm, η οποία μπορεί να γίνει μέσω του Jupyter notebook, με τον ίδιο τρόπο δηλαδή που μπορεί κάποιος να εκτελέσει το SDK του Fogify. Στην *Εικόνα 6* υπάρχει παράδειγμα του κώδικα που μπορεί να χρησιμοποιηθεί για την εκτέλεση της επέκτασης και των διαθέσιμων λειτουργιών.

```python
from ppcomm.ppcomm import PPComm

ppcomm = PPComm('http://controller:5000', 'docker-compose.yaml')

# all available FogifySDK API methods can be accessed here.
fogify = ppcomm.get_fogify_sdk()

# fogify.deploy()
# fogify.undeploy()

# ppcomm.get_trajectories()
# ppcomm.get_topology()

# ppcomm.topology.get_nodes()
# ppcomm.topology.get_connections()

# ppcomm.print_topology()
# ppcomm.print_trajectories()

# ppcomm.change_node_coords('drone_0', {'latitude':0, 'longitude':0})
# ppcomm.exec_trajectory('tran_1')
```

Εικόνα 6: Βασικό παράδειγμα εκτέλεσης

Αρχικά, ο χρήστης πρέπει να κάνει import την επέκταση και ακολούθως να δημιουργήσει ένα αντικείμενο της κλάσης PPComm με παραμέτρους την διεύθυνση του Fogify Controller και το όνομα του αρχείου docker-compose, το οποίο πρέπει να βρίσκεται στον ίδιο φάκελο με το αρχείο εκτέλεσης. Στη συνέχεια με τη χρήση της μεθόδου ppcomm.get_fogify_sdk() αποκτάει πρόσβαση σε όλες τις διαθέσιμες μεθόδους του FogifySDK, για τις οποίες μπορεί να μάθει περισσότερες πληροφορίες στην επίσημη σελίδα του Fogify.

Ο χρήστης έχει επίσης πρόσβαση σε όλες τις διαθέσιμες μεθόδους του PPComm οι οποίες παρουσιάζονται πιο κάτω:

i. **ppcomm.get_trajectories()**

   Η μέθοδος αυτή επιστρέφει την λίστα των trajectories, σε μορφή Python Dictionary.

ii. **ppcomm.get_topology()**

   Η μέθοδος αυτή επιστρέφει την λίστα των συσκευών και των συνδέσεων που βρίσκονται στην τοπολογία του PPComm, σε μορφή Python Dictionary.

iii. **ppcomm.print_topology()**

Η μέθοδος αυτή τυπώνει με ευανάγνωστο τρόπο την τοπολογία του PPComm.

iv. **ppcomm.print_trajectories()**

Η μέθοδος αυτή τυπώνει με ευανάγνωστο τρόπο την λίστα των trajectories.

v. **ppcomm.change_node_coords(name, coords)**

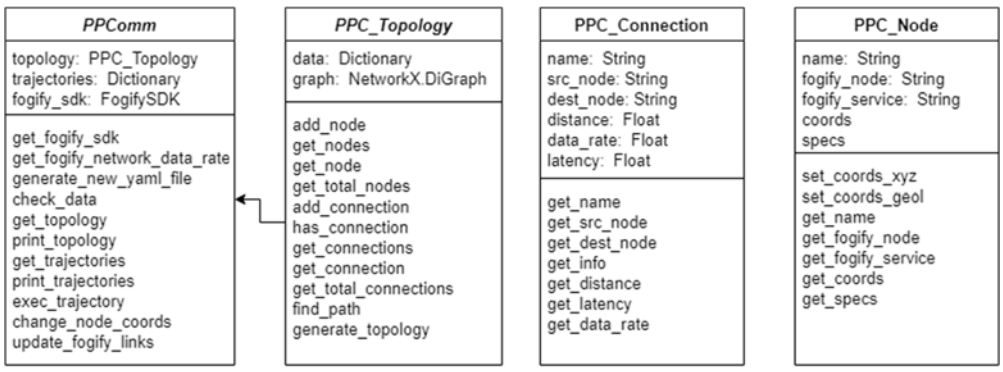
Η μέθοδος αυτή αλλάζει τις συντεταγμένες της συσκευής και τυπώνει το αποτέλεσμα της ενημέρωσης που έγινε μέσω του Fogify Controller. Η μέθοδος αυτή μπορεί να χρησιμοποιηθεί αν θέλει ο χρήστης να αλλάξει τις συντεταγμένες κάποιας συσκευής μέσω του κώδικα, χωρίς να χρειαστεί να εκτελέσει κάποιο trajectory.

vi. **ppcomm.exec_trajectory(name)**

Η μέθοδος αυτή εκτελεί το trajectory που επιλέγει ο χρήστης και τυπώνει στην οθόνη το αποτέλεσμα των ενεργειών που εκτελούνται μέσω του trajectory.

vii. **ppcomm.topology**

Μέσω του ppcomm.topology, ο χρήστης μπορεί να αποκτήσει πρόσβαση στην κλάση PPC_Topology που βρίσκεται πίσω από το PPComm και να χρησιμοποιήσει τις μεθόδους που προσφέρει ή να αποκτήσει πρόσβαση στον γράφο της κλάσης. Περισσότερες λεπτομέρειες για τις διαθέσιμες μεθόδους της κλάσης μπορείτε να βρείτε στο Παράρτημα-Α – Κώδικας Επέκτασης.



Εικόνα 7: Διάγραμμα κλάσεων PPComm

2.5 Μαθηματικό υπόβαθρο

Ένα κομμάτι της υλοποίησης του PPComm αναλαμβάνει τον υπολογισμό της απόστασης των συσκευών καθώς και τον υπολογισμό των δυνατοτήτων δικτύου, όπως καθυστέρηση (latency) και ρυθμός αποστολής δεδομένων (data rate). Για τον υπολογισμό των πιο πάνω απαιτούνται ορισμένες μαθηματικές πράξεις στις οποίες θα αναφερθούμε σε αυτή την ενότητα.

Φόρμουλα Τρισδιάστατής Απόστασης

Για τον υπολογισμό της απόστασης δύο συσκευών όταν χρησιμοποιείται το Τρισδιάστατο Σύστημα Συντεταγμένων, χρησιμοποιείται η φόρμουλα τρισδιάστατης απόστασης [8]. Η απόσταση μεταξύ δύο συσκευών που βρίσκονται στα σημεία $P_1(x_1,y_1,z_1)$ και $P_2(x_2,y_2,z_2)$ αντίστοιχα μπορεί να υπολογιστεί με τη χρήση της παρακάτω φόρμουλας:

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Φόρμουλα Γεωγραφικής Απόστασης

Για τον υπολογισμό της απόστασης δύο συσκευών όταν χρησιμοποιείται το Γεωγραφικό Σύστημα Συντεταγμένων, χρησιμοποιείται η φόρμουλα Haversine [9]. Η απόσταση μεταξύ δύο συσκευών που βρίσκονται στα σημεία P1(φ1,λ1) και P2(φ2,λ2) αντίστοιχα μπορεί να υπολογιστεί με την χρήση της παρακάτω φόρμουλας (φ: γεωγραφικό πλάτος, λ: γεωγραφικό μήκος):

$$\begin{aligned} d &= 2r \arcsin\left(\sqrt{\operatorname{hav}(\varphi_2 - \varphi_1) + (1 - \operatorname{hav}(\varphi_1 - \varphi_2) - \operatorname{hav}(\varphi_1 + \varphi_2)) \cdot \operatorname{hav}(\lambda_2 - \lambda_1)}\right) \\ &= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \left(1 - \sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) - \sin^2\left(\frac{\varphi_2 + \varphi_1}{2}\right)\right) \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \\ &= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right). \end{aligned}$$

Φόρμουλα για Pathloss

Για τον υπολογισμό του pathloss χρησιμοποιήθηκε η ακόλουθη φόρμουλα:

$$PL(dB) = 28 + 22 * log_{10}(d) + 20 * log_{10}(c)$$

Όπου

d: απόσταση δύο συσκευών,

c: carrier frequency της συσκευής πηγή (source) σε Ghz.

Χρησιμοποιήθηκε το UMa LOS (PL1) από το specification της ETSI για το 5G [10]. Ωστόσο ο χρήστης μπορεί να ενημερώσει φόρμουλα αυτή και να χρησιμοποιηθεί κάποια άλλη.

Φόρμουλα για Receiver Power

Για τον υπολογισμό του receiver power χρησιμοποιήθηκε η ακόλουθη φόρμουλα [11]:

$$RP(dB) = p + r + u - PL$$

Όπου

p: transmitter power της συσκευής πηγή (source) σε dBm.

r: RU antenna gain της συσκευής πηγή σε dBi.

u: UE antenna gain της συσκευής προορισμού (dest) σε dBi.

PL: Pathloss σε dB.

Φόρμουλα για Noise (Θόρυβος σήματος)

Για τον υπολογισμό του θορύβου σήματος μεταξύ δύο συσκευών χρησιμοποιήθηκε η ακόλουθη φόρμουλα [11]:

$$N = -174 + 10 * log_{10}(b) + u$$

Όπου

b: bandwidth της συσκευής πηγή (source) σε Hz

u: UE noise figure της συσκευής προορισμού (dest) σε dBi.

Φόρμουλα για Signal-to-Noise Ratio (SNR)

Για τον υπολογισμό του signal-to-noise ratio [11] χρησιμοποιήθηκε η ακόλουθη φόρμουλα, η οποία χρησιμοποιεί τις φόρμουλες Receiver Power και Noise για τον υπολογισμό της:

$$SNR(dB) = RP - N$$

Όπου

RP: receiver power σε dB.

N: noise σε dB.

Φόρμουλα για ρυθμό αποστολής δεδομένων (data rate)

Για τον υπολογισμό του ρυθμού αποστολής δεδομένων χρησιμοποιήθηκε ο τύπος του Shannon. Η φόρμουλα [11] είναι η ακόλουθη και χρησιμοποιεί την φόρμουλα SNR για τον υπολογισμό της:

$$DR(bits) = b * log(1 + SNR)$$

Όπου

b: bandwidth της συσκευής πηγή (source) σε hz

SNR: signal-to-noise ratio σε dB.

Φόρμουλα για καθυστέρηση (latency)

Για τον υπολογισμό της καθυστέρησης (latency) σε μια σύνδεση χρησιμοποιήθηκε η ακόλουθη φόρμουλα [12], η οποία χρησιμοποιεί την φόρμουλα SNR για τον υπολογισμό της:

$$Latency(ms) = (58.854 * exp(-0.055837 * SNR)) / 2$$

Όπου

SNR: signal-to-noise ratio σε dB.

Κεφάλαιο 3

Πειραματισμός & αξιολόγηση

3.1 Εισαγωγή

Στο κεφάλαιο αυτό θα γίνει πειραματισμός ορισμένων σεναρίων, διαφοροποιώντας την μοντελοποίηση x-ppcomm στο docker-compose και ακολούθως θα γίνει σχολιασμός και αξιολόγηση των αποτελεσμάτων. Επίσης, τα σενάρια αυτά αποτελούν παραδείγματα χρήσης της επέκτασης PPComm.

Δεν λήφθηκαν μετρήσεις που δημιουργούνται και εξάγονται από το Fogify καθώς τα παραδείγματα πιο κάτω εστιάζουν στο σχολιασμό των λειτουργιών τις οποίες προσφέρει το PPComm και όχι σε αυτές που προσφέρονται από το Fogify.

3.2 Σενάριο 1ο: Τρισδιάστατο σύστημα συντεταγμένων

Στο σενάριο αυτό θα χρησιμοποιήσουμε το τρισδιάστατο σύστημα συντεταγμένων. Το μοντέλο του PPComm θα αποτελείται από 4 συσκευές

τοποθετημένες σε κοντινή απόσταση, έτσι ώστε να υπάρχει συνδεσιμότητα μεταξύ όλων. Όλες οι συσκευές είναι οι ίδιες οπότε το ίδιο συμβαίνει και με τις προδιαγραφές τους. Η επιλογή find_path είναι απενεργοποιημένη.

```
x-ppcomm:
  config:
    coord_type: xyz
    network: ppc
    find_path: false
    minimum_snr: 25
  nodes:
    - name: drone_0
      fogify:
        node: drone-node
        service: mec-svc-1
      coords:
        x: 0
        y: 0
        z: 0
      specs:
        carrier_frequency: 28   #ghz
        bandwidth: 0.100        #ghz
        transmit_power: 30      #dBm
        ru_antenna_gain: 6      #dbi
        ue_antenna_gain: 6      #dbi
        ue_noise_figure: 7.8    #dB
    - name: drone_1
      fogify:
        node: drone-node
        service: mec-svc-1
      coords:
        x: 5
        y: 5
        z: 0
      specs:
        carrier_frequency: 28   #ghz
        bandwidth: 0.100        #ghz
        transmit_power: 30      #dBm
        ru_antenna_gain: 6      #dbi
        ue_antenna_gain: 6      #dbi
        ue_noise_figure: 7.8    #dB
    - name: drone_2
      fogify:
        node: drone-node
        service: mec-svc-1
      coords:
        x: 45
        y: 45
        z: 0
      specs:
        carrier_frequency: 28   #ghz
        bandwidth: 0.100        #ghz
        transmit_power: 30      #dBm
        ru_antenna_gain: 6      #dbi
        ue_antenna_gain: 6      #dbi
        ue_noise_figure: 7.8    #dB
    - name: drone_3
      fogify:
        node: drone-node
        service: mec-svc-1
```

```
      coords:
        x: 15
        y: 15
        z: 0
      specs:
        carrier_frequency: 28   #ghz
        bandwidth: 0.100        #ghz
        transmit_power: 30      #dBm
        ru_antenna_gain: 6      #dbi
        ue_antenna_gain: 6      #dbi
        ue_noise_figure: 7.8    #dB
  trajectories:
    - name: tran_1
      changes:
        - time: 5
          nodes:
            - name: drone_2
              coords:
                x: 55
                y: 55
                z: 0
```

Εικόνα 8: docker-compose 1ου πειράματος.

Στο docker-compose της *Εικόνας 8* παρατηρούμε ότι υπάρχουν 4 συσκευές (drones), τοποθετημένες σε διαφορετικές αποστάσεις. Επίσης έχει οριστεί ένα trajectory (tran_1) το οποίο όταν εκτελεστεί, θα μετακινήσει το drone_2, στις συντεταγμένες (x=55,y=55,z=0) μετά από 5 δευτερόλεπτα.

Με την εκτέλεση του προγράμματος, η επέκταση δημιουργεί καινούριο αρχείο docker-compose το οποίο στέλνει στο Fogify Controller. Στην *Εικόνα 9* παρατηρούμε το αρχείο αυτό, στο οποίο έχουν προστεθεί οι συσκευές του PPComm στην τοπολογία του μοντέλου Fogify. Επίσης έχουν προστεθεί όλες οι point-to-point συνδέσεις των συσκευών στα links του δικτύου «ppc». Παρατηρούμε ότι το μέγιστο bandwidth των συνδέσεων είναι στα 20Mbps, ασχέτως αν στη τοπολογία του PPComm το bandwidth των συνδέσεων είναι μεγαλύτερο. Αυτό συμβαίνει γιατί οι συνδέσεις περιορίζονται από το bandwidth του downlink στο δίκτυο «ppc» του Fogify.

```
services:
  …
version: '3.7'
x-fogify:
  networks:
```

```
- downlink:
    bandwidth: 20Mbps
    latency:
      delay: 0ms
  links:
  - bidirectional: false
    from_node: drone_0
    properties:
      bandwidth: 20Mbps
      latency:
        delay: 1ms
    to_node: drone_1
  - bidirectional: false
    from_node: drone_0
    properties:
      bandwidth: 20Mbps
      latency:
        delay: 5ms
    to_node: drone_2
  - bidirectional: false
    from_node: drone_0
    properties:
      bandwidth: 20Mbps
      latency:
        delay: 2ms
    to_node: drone_3
  - bidirectional: false
    from_node: drone_1
    properties:
      bandwidth: 20Mbps
      latency:
        delay: 1ms
    to_node: drone_0
  - bidirectional: false
    from_node: drone_1
    properties:
      bandwidth: 20Mbps
      latency:
        delay: 4ms
    to_node: drone_2
  - bidirectional: false
    from_node: drone_1
    properties:
      bandwidth: 20Mbps
      latency:
        delay: 2ms
    to_node: drone_3
  - bidirectional: false
    from_node: drone_2
    properties:
      bandwidth: 20Mbps
      latency:
        delay: 5ms
    to_node: drone_0
  - bidirectional: false
    from_node: drone_2
    properties:
      bandwidth: 20Mbps
      latency:
        delay: 4ms
    to_node: drone_1
  - bidirectional: false
    from_node: drone_2
    properties:
      bandwidth: 20Mbps
```

```
      latency:
        delay: 4ms
    to_node: drone_3
  - bidirectional: false
    from_node: drone_3
    properties:
      bandwidth: 20Mbps
      latency:
        delay: 2ms
    to_node: drone_0
  - bidirectional: false
    from_node: drone_3
    properties:
      bandwidth: 20Mbps
      latency:
        delay: 2ms
    to_node: drone_1
  - bidirectional: false
    from_node: drone_3
    properties:
      bandwidth: 20Mbps
      latency:
        delay: 4ms
    to_node: drone_2
  name: ppc
  uplink:
    bandwidth: 10Mbps
    drop: 0.1%
    latency:
      delay: 0ms
nodes:
- capabilities:
    memory: 1G
    processor:
      clock_speed: 1400
      cores: 2
  name: drone-node
- capabilities:
    memory: 4G
    processor:
      clock_speed: 1400
      cores: 4
  name: server-node
topology:
- label: server
  networks:
  - ppc
  node: server-node
  replicas: 1
  service: cloud-server
- label: drone_0
  networks:
  - ppc
  node: drone-node
  replicas: 1
  service: mec-svc-1
- label: drone_1
  networks:
  - ppc
  node: drone-node
  replicas: 1
  service: mec-svc-1
- label: drone_2
  networks:
  - ppc
```

```
    node: drone-node
    replicas: 1
    service: mec-svc-1
  - label: drone_3
    networks:
    - ppc
    node: drone-node
    replicas: 1
    service: mec-svc-1
```

Εικόνα 9: Ολοκληρωμένο docker-compose 1ου πειράματος.

Στην *Εικόνα 11* υπάρχουν τα αποτελέσματα εκτέλεσης του 1ου σεναρίου, όπου αρχικά τυπώνεται η τοπολογία του PPComm, ακολούθως γίνετε εκτέλεση του trajectory tran_1 και στο τέλος τυπώνεται ξανά η ενημερωμένη τοπολογία. Παρατηρούμε ότι την δεύτερη φορά που γίνετε εκτύπωση της τοπολογίας, οι συνδέσεις στις οποίες άλλαξε η απόσταση, έχουν καινούριες τιμές στα distance (m), latency (ms) και data rate (bps). Επίσης, παρατηρούμε ότι κατά την εκτέλεση του trajectory, η επέκταση εκτύπωσε τα αποτελέσματα ενημέρωσης του Fogify controller για κάθε σύνδεση.

```
from ppcomm.ppcomm import PPComm

ppcomm = PPComm('http://10.16.3.26:5000', 'docker-compose-s1.yaml')

# all available FogifySDK API methods can be accessed here.
fogify = ppcomm.get_fogify_sdk()

ppcomm.print_topology()

ppcomm.exec_trajectory('tran_1')

ppcomm.print_topology()
```

Εικόνα 10: Αρχείο εκτέλεσης 1ου πειράματος.

```
ubuntu@fogifyworker-1:~/testimpl/demo$ python demo.py
Nodes (4):
        drone_0 | Coordinates: {'x': 0.0, 'y': 0.0, 'z': 0.0}
        drone_1 | Coordinates: {'x': 5.0, 'y': 5.0, 'z': 0.0}
        drone_2 | Coordinates: {'x': 45.0, 'y': 45.0, 'z': 0.0}
        drone_3 | Coordinates: {'x': 15.0, 'y': 15.0, 'z': 0.0}

Connections (12):
        drone_0 to drone_1 | {'distance': 7.0710678118654755, 'latency':
1.563088621775903, 'data_rate': 398095503.5806224}
        drone_0 to drone_2 | {'distance': 63.63961030678928, 'latency':
5.0473765215406265, 'data_rate': 348354003.0681573}
        drone_0 to drone_3 | {'distance': 21.213203435596427, 'latency':
2.8088248095313797, 'data_rate': 376286156.5456713}
        drone_1 to drone_0 | {'distance': 7.0710678118654755, 'latency':
```

```
1.563088621775903, 'data_rate': 398095503.5806224}
        drone_1 to drone_2 | {'distance': 56.568542494923804, 'latency':
4.739976286779486, 'data_rate': 351750346.38005483}
        drone_1 to drone_3 | {'distance': 14.142135623730951, 'latency':
2.262461117294738, 'data_rate': 384898755.5091656}
        drone_2 to drone_0 | {'distance': 63.63961030678928, 'latency':
5.0473765215406265, 'data_rate': 348354003.0681573}
        drone_2 to drone_1 | {'distance': 56.568542494923804, 'latency':
4.739976286779486, 'data_rate': 351750346.38005483}
        drone_2 to drone_3 | {'distance': 42.42640687119285, 'latency':
4.065576851066493, 'data_rate': 359590968.33824724}
        drone_3 to drone_0 | {'distance': 21.213203435596427, 'latency':
2.8088248095313797, 'data_rate': 376286156.5456713}
        drone_3 to drone_1 | {'distance': 14.142135623730951, 'latency':
2.262461117294738, 'data_rate': 384898755.5091656}
        drone_3 to drone_2 | {'distance': 42.42640687119285, 'latency':
4.065576851066493, 'data_rate': 359590968.33824724}

Executing trajectory 'tran_1'.
At 5 seconds:
        Node 'drone_2' moved to {'x': 55.0, 'y': 55.0, 'z': 0.0}.

Topology updated!

Updating fogify network links...
Link updated: {'drone_0': {'message': 'OK'}}
Link updated: {'drone_0': {'message': 'OK'}}
Link updated: {'drone_0': {'message': 'OK'}}
Link updated: {'drone_1': {'message': 'OK'}}
Link updated: {'drone_1': {'message': 'OK'}}
Link updated: {'drone_1': {'message': 'OK'}}
Link updated: {'drone_2': {'message': 'OK'}}
Link updated: {'drone_2': {'message': 'OK'}}
Link updated: {'drone_2': {'message': 'OK'}}
Link updated: {'drone_3': {'message': 'OK'}}
Link updated: {'drone_3': {'message': 'OK'}}
Link updated: {'drone_3': {'message': 'OK'}}
Fogify network links updated!

Nodes (4):
        drone_0 | Coordinates: {'x': 0.0, 'y': 0.0, 'z': 0.0}
        drone_1 | Coordinates: {'x': 5.0, 'y': 5.0, 'z': 0.0}
        drone_2 | Coordinates: {'x': 55.0, 'y': 55.0, 'z': 0.0}
        drone_3 | Coordinates: {'x': 15.0, 'y': 15.0, 'z': 0.0}

Connections (12):
        drone_0 to drone_1 | {'distance': 7.0710678118654755, 'latency':
1.563088621775903, 'data_rate': 398095503.5806224}
        drone_0 to drone_2 | {'distance': 77.78174593052023, 'latency':
5.617715595322748, 'data_rate': 342287831.7426075}
        drone_0 to drone_3 | {'distance': 21.213203435596427, 'latency':
2.8088248095313797, 'data_rate': 376286156.5456713}
        drone_1 to drone_0 | {'distance': 7.0710678118654755, 'latency':
1.563088621775903, 'data_rate': 398095503.5806224}
        drone_1 to drone_2 | {'distance': 70.71067811865476, 'latency':
5.339210254528831, 'data_rate': 345214931.3098217}
        drone_1 to drone_3 | {'distance': 14.142135623730951, 'latency':
2.262461117294738, 'data_rate': 384898755.5091656}
        drone_2 to drone_0 | {'distance': 77.78174593052023, 'latency':
5.617715595322748, 'data_rate': 342287831.7426075}
        drone_2 to drone_1 | {'distance': 70.71067811865476, 'latency':
5.339210254528831, 'data_rate': 345214931.3098217}
        drone_2 to drone_3 | {'distance': 56.568542494923804, 'latency':
4.739976286779486, 'data_rate': 351750346.38005483}
        drone_3 to drone_0 | {'distance': 21.213203435596427, 'latency':
```

```
2.8088248095313797, 'data_rate': 376286156.5456713}
        drone_3 to drone_1 | {'distance': 14.142135623730951, 'latency':
2.262461117294738, 'data_rate': 384898755.5091656}
        drone_3 to drone_2 | {'distance': 56.568542494923804, 'latency':
4.739976286779486, 'data_rate': 351750346.38005483}

ubuntu@fogifyworker-1:~/testimpl/demo$
```

Εικόνα 11: Αποτελέσματα εκτέλεσης 1ου πειράματος.

3.3 Σενάριο 2ο: Εύρεση μονοπατιού με βάση τη καθυστέρηση

Στο σενάριο αυτό θα χρησιμοποιήσουμε το γεωγραφικό σύστημα συντεταγμένων και θα τοποθετήσουμε ορισμένους κόμβους σε μακρινή απόσταση για να δείξουμε ότι δεν υπάρχει σύνδεση μεταξύ τους. Στη συνέχεια θα τρέξουμε το ίδιο σενάριο αλλά θα ενεργοποιήσουμε την επιλογή find_path και θα παρατηρήσουμε κατά πόσο υπάρχει σύνδεση των απομακρυσμένων συσκευών με την βοήθεια ενδιάμεσων συσκευών. Το μοντέλο του PPComm θα αποτελείται από 3 συσκευές.

```yaml
x-ppcomm:
  config:
    coord_type: latlong
    network: ppc
    find_path: false
    minimum_snr: 25
  nodes:
    - name: drone_0
      fogify:
        node: drone-node
        service: mec-svc-1
      coords:
        latitude: 0
        longitude: 0
      specs:
        carrier_frequency: 28   #ghz
        bandwidth: 0.100        #ghz
        transmit_power: 30      #dBm
        ru_antenna_gain: 6      #dbi
        ue_antenna_gain: 6      #dbi
        ue_noise_figure: 7.8    #dB
    - name: drone_1
      fogify:
        node: drone-node
        service: mec-svc-1
      coords:
        latitude: 0.00025
        longitude: 0.00025
```

```
      specs:
        carrier_frequency: 28   #ghz
        bandwidth: 0.100        #ghz
        transmit_power: 30      #dBm
        ru_antenna_gain: 6      #dbi
        ue_antenna_gain: 6      #dbi
        ue_noise_figure: 7.8    #dB
    - name: drone_2
      fogify:
        node: drone-node
        service: mec-svc-1
      coords:
        latitude: 0.001
        longitude: 0.001
      specs:
        carrier_frequency: 28   #ghz
        bandwidth: 0.100        #ghz
        transmit_power: 30      #dBm
        ru_antenna_gain: 6      #dbi
        ue_antenna_gain: 6      #dbi
        ue_noise_figure: 7.8    #dB
```

Εικόνα 12: docker-compose 2ου πειράματος.

Παρατηρώντας τα αποτελέσματα εκτέλεσης στην *Εικόνα 13*, δεν υπάρχει σύνδεση μεταξύ των συσκευών drone_0 και drone_2 γιατί βρίσκονται εκτός εμβέλειας. Ωστόσο η συσκευή drone_1 επικοινωνεί με τις υπόλοιπες 2 συσκευές.

```
ubuntu@fogifyworker-1:~/testimpl/demo$ python demo.py
Nodes (3):
        drone_0 | Coordinates: {'latitude': 0.0, 'longitude': 0.0}
        drone_1 | Coordinates: {'latitude': 0.00025, 'longitude': 0.00025}
        drone_2 | Coordinates: {'latitude': 0.001, 'longitude': 0.001}

Connections (4):
        drone_0 to drone_1 | {'distance': 39.31334333189173, 'latency':
3.903601990313394, 'data_rate': 361568921.00226307}
        drone_1 to drone_0 | {'distance': 39.31334333189173, 'latency':
3.903601990313394, 'data_rate': 361568921.00226307}
        drone_1 to drone_2 | {'distance': 117.94002999193285, 'latency':
7.014659286743041, 'data_rate': 328392517.4051518}
        drone_2 to drone_1 | {'distance': 117.94002999193285, 'latency':
7.014659286743041, 'data_rate': 328392517.4051518}
```

Εικόνα 13: Αποτελέσματα εκτέλεσης 2ου πειράματος.

Ακολούθως, αλλάζουμε την επιλογή find_path σε true χωρίς να διαφοροποιήσουμε κάτι άλλο το μοντέλο του x-ppcomm. Στην *Εικόνα 14*, παρατηρούμε ότι πλέον υπάρχει σύνδεση μεταξύ των συσκευών drone_0 και drone_2 αφού βρέθηκε μονοπάτι μέσω της συσκευής drone_1 για την δημιουργία σύνδεσης. Στην περίπτωση αυτή βλέπουμε ότι το data rate της

σύνδεσης αυτής περιορίζεται στο ελάχιστο data rate των συνδέσεων του μονοπατιού και η καθυστέρηση (latency) αυξάνεται αφού έχουν προστεθεί όλες οι καθυστερήσεις των συνδέσεων του μονοπατιού.

```
Nodes (3):
        drone_0 | Coordinates: {'latitude': 0.0, 'longitude': 0.0}
        drone_1 | Coordinates: {'latitude': 0.00025, 'longitude': 0.00025}
        drone_2 | Coordinates: {'latitude': 0.001, 'longitude': 0.001}

Connections (6):
        drone_0 to drone_1 | {'distance': 39.31334333189173, 'latency':
3.903601990313394, 'data_rate': 361568921.00226307}
        drone_0 to drone_2 | {'distance': 157.25337332382458, 'latency':
10.918261277056434, 'data_rate': 328392517.4051518}
        drone_1 to drone_0 | {'distance': 39.31334333189173, 'latency':
3.903601990313394, 'data_rate': 361568921.00226307}
        drone_1 to drone_2 | {'distance': 117.94002999193285, 'latency':
7.014659286743041, 'data_rate': 328392517.4051518}
        drone_2 to drone_1 | {'distance': 117.94002999193285, 'latency':
7.014659286743041, 'data_rate': 328392517.4051518}
        drone_2 to drone_0 | {'distance': 157.25337332382458, 'latency':
10.918261277056434, 'data_rate': 328392517.4051518}
```

Εικόνα 14: Αποτελέσματα εκτέλεσης 2ου πειράματος.

Κεφάλαιο 4

Συμπεράσματα και περιορισμοί

4.1 Συμπεράσματα

Η προσθήκη ενός συστήματος συντεταγμένων στο Fogify ευκολύνει σε τεράστιο βαθμό τους χρήστες να δημιουργήσουν μοντέλα εξομοίωσης τα οποία να αποτελούνται από συσκευές συνδεδεμένες μεταξύ τους (point-to-point), κινούνται στο χώρο και απαιτείται η συνεχής μετακίνηση τους. Πιο συγκεκριμένα, η υλοποίηση των trajectories κάνει τη μοντελοποίηση τέτοιων μετακινήσεων πολύ απλή, χωρίς να χρειάζεται ο χρήστης του Fogify να εμπλακεί σε χιλιάδες υπολογισμούς που μπορούν να αποφευχθούν με το γράψιμο της επέκτασης αυτής.

Η προσθήκη του πιο πάνω συστήματος έγινε με την μορφή επέκτασης, η οποία χρησιμοποιεί το FogifySDK. Το PPComm αποτελεί τη πρώτη επέκταση που δημιουργήθηκε για το Fogify και μπορεί να ανοίξει το δρόμο σε καινούριες επεκτάσεις οι οποίες μπορεί να κριθούν σημαντικές στην ανάπτυξη του Fogify framework. Με προσθήκες σαν και αυτές, ανοίγονται νέα παράθυρα στα πιθανά σενάρια που μπορούν να δημιουργήσουν οι χρήστες του Fogify

προκειμένου να εξετάσουν τις δικές τους ιδέες μοντελοποίησης και να ανακαλύψουν τα όρια των τοπολογιών τους.

4.2 Περιορισμοί

Κατά το γράψιμο και παράδοση της Ατομικής Διπλωματικής Εργασίας μου, το FogifySDK παρουσίαζε ορισμένα προβλήματα στην κλήση ορισμένων μεθόδων. Πιο συγκεκριμένα:

i. η κλήση της deploy() παρουσιάζει exception, ωστόσο το deployment γίνετε με επιτυχία.
ii. Η μέθοδος add_link() η οποία χρησιμοποιείται για την προσθήκη νέων συνδέσεων δεν επιστρέφει κάποιο μήνυμα επιτυχίας ή λάθους και έτσι εμφανίζεται το μήνυμα None όταν γίνεται κλήση από την επέκταση PPComm.
iii. Η μέθοδος update_link() η οποία χρησιμοποιείται για την ενημέρωση των υφιστάμενων συνδέσεων, επιστρέφει σφάλμα σε περίπτωση που έχει γίνει deployment. Αν δεν προηγηθεί deploy() πριν την εκτέλεση της update_link(), δεν παρουσιάζετε σφάλμα.

Τα πιο πάνω προβλήματα επηρεάζουν την λειτουργία της επέκτασης PPComm. Ωστόσο με την επιδιόρθωση τους, δεν θα χρειαστεί να αλλάξει κάτι στον κώδικα της επέκτασης.

Βιβλιογραφία

[1] Wikipedia, «Fog computing».
https://en.wikipedia.org/wiki/Fog_computing

[2] Analog Devices, «Intelligence at the Edge: Reduced Time to Insight»
https://www.analog.com/en/technical-articles/intelligence-at-the-edge-part-2-reduced-time-to-insight.html

[3] LinC, «Fogify: A Fog Computing Emulation Framework».
http://linc.ucy.ac.cy/assets/files/publications/pdfs/fogify_moysis_symeonides_2020.pdf

[4] Wikipedia, «Docker (software)».
https://en.wikipedia.org/wiki/Docker_(software)

[5] Docker, «What is a Container?».
https://www.docker.com/resources/what-container

[6] Docker, «Overview of Docker Compose».
https://docs.docker.com/compose/overview/

[7] Docker, «Docker Swarm».
https://docs.docker.com/swarm/

[8] math.usm.edu, «Three-Dimensional Coordinate Systems».
https://www.math.usm.edu/lambers/mat169/fall09/lecture17.pdf

[9] Wikipedia, «Haversine Formula».
https://en.wikipedia.org/wiki/Haversine_formula

[10] «5G Study on channel model for frequencies from 0.5 to 100 GHz)».
https://www.etsi.org/deliver/etsi_tr/138900_138999/138901/14.03.00_60/tr_138901v140300p.pdf

[11] «5G-Slicer: An emulator for mobile IoT applications deployed over 5G network slices».
Symeonides Moysis, Trihinas Demetris, Pallis George, Dikaiakos Marios D., Psomas Constantinos, Krikidis Ioannis.
«Proceedings of the 7th ACM/IEEE Conference on Internet of Things Design and Implementation».

[12] «Emulating a Software Defined LTE Radio Access Network Towards 5G».
I. Santos, P. Vieira, R. Borralho, M. P. Queluz and A. Rodrigues.
«2018 International Conference on Communications (COMM)».

Παράρτημα Α – Κώδικας επέκτασης

Ακολουθεί σχολιασμός του κώδικα της επέκτασης, όπου θα γίνει σύντομη περίληψη του κάθε αρχείου Python. Περισσότερες λεπτομέρειες σχετικά με το τρόπο λειτουργίας του κώδικα υπάρχουν σε μορφή σχόλιων (comments) τα οποία είναι ενσωματωμένα στον κώδικα. Στα σημεία όπου η λειτουργία του κώδικα είναι εύκολη στην κατανόηση, αποφεύγεται η χρήση σχόλιων για να διατηρηθεί καθαρός.

Κλάση PPC_Node (ppcomm_node.py)

Ένα αντικείμενο της κλάσης PPC_Node αναπαριστά ένα κόμβο/συσκευή στη τοπολογία του PPComm. Κατά την δημιουργία ενός αντικειμένου, αποθηκεύεται το όνομα της συσκευής, το όνομα του κόμβου στο fogify και το όνομα της υπηρεσίας στο fogify που αναπαριστά αυτή τη συσκευή, οι συντεταγμένες και οι προδιαγραφές της συσκευής. Μέσω των μεθόδων της κλάσης αυτής μπορεί να γίνει ανάκτηση των πιο πάνω πληροφοριών και να γίνει ενημέρωση των συντεταγμένων της συσκευής.

```
# represents a node in the graph of ppcomm topology (PPComm_Topology).
class PPC_Node:
    def __init__(self, node):
        self.name = node['name']

        # fogify node that this ppcomm node uses.
        self.fogify_node = node['fogify']['node']

        # fogify service that this node uses.
        self.fogify_service = node['fogify']['service']

        # node's coordinates.
        if 'latitude' in node['coords'].keys():
            self.latitude = float(node['coords']['latitude'])
            self.longitude = float(node['coords']['longitude'])
        else:
            self.x = float(node['coords']['x'])
            self.y = float(node['coords']['y'])
            self.z = float(node['coords']['z'])

        # node's specification.
```

```
        self.carrier_frequency = node['specs']['carrier_frequency']
        self.bandwidth = node['specs']['bandwidth']
        self.transmit_power = node['specs']['transmit_power']
        self.ru_antenna_gain = node['specs']['ru_antenna_gain']
        self.ue_antenna_gain = node['specs']['ue_antenna_gain']
        self.ue_noise_figure = node['specs']['ue_noise_figure']


    def set_coords_geol(self, lat, long):
        self.latitude = float(lat)
        self.longitude = float(long)

    def set_coords_xyz(self, x, y, z):
        self.x = float(x)
        self.y = float(y)
        self.z = float(z)

    def get_name(self):
        return self.name

    def get_fogify_node(self):
        return self.fogify_node

    def get_fogify_service(self):
        return self.fogify_service

    def get_coords(self):
        if hasattr(self, 'latitude'):
            return {
                'latitude': self.latitude,
                'longitude': self.longitude
            }
        return {
            'x': self.x,
            'y': self.y,
            'z': self.z
        }

    def get_carrier_frequency(self):
        return self.carrier_frequency

    def get_bandwidth(self):
        return self.bandwidth

    def get_transmit_power(self):
        return self.transmit_power

    def get_ru_antenna_gain(self):
        return self.ru_antenna_gain

    def get_ue_antenna_gain(self):
        return self.ue_antenna_gain

    def get_ue_noise_figure(self):
        return self.ue_noise_figure

    def get_specs(self):
        return {
            'carrier_frequency': self.carrier_frequency,
            'bandwidth': self.bandwidth,
            'transmit_power': self.transmit_power,
            'ru_antenna_gain': self.ru_antenna_gain,
            'ue_antenna_gain': self.ue_antenna_gain,
            'ue_noise_figure': self.ue_noise_figure
        }
```

```
    def __repr__(self):
        return 'PPC_Node() [' + self.name + ']'

    def __str__(self):
        return 'PPC_Node() [' + self.name + ']'
```

Κλάση PPC_Connection (ppcomm_connection.py)

Ένα αντικείμενο της κλάσης PPC_Connection αναπαριστά μια σύνδεση μεταξύ δύο κόμβων/συσκευών στη τοπολογία του PPComm. Κατά την δημιουργία ενός αντικειμένου, αποθηκεύεται το όνομα της σύνδεσης, ο κόμβος/συσκευή που αποτελεί την πηγή της σύνδεσης (PPC_Node), ο κόμβος/συσκευή που αποτελεί τον προορισμό της σύνδεσης (PPC_Node), η απόσταση μεταξύ των δύο συσκευών, η ταχύτητα αποστολής δεδομένων (data rate) και η καθυστέρηση (latency). Μέσω των μεθόδων της κλάσης αυτής μπορεί να γίνει ανάκτηση των πιο πάνω πληροφοριών.

```
# represents a connection between two PPC_Node instances
# in the graph of ppcomm topology (PPC_Topology).
class PPC_Connection:
    def __init__(self, connection):
        self.name = connection['name']

        # source node (instance of PPC_Node)
        self.src_node = connection['src_node']

        # destination node (instance of PPC_Node)
        self.dest_node = connection['dest_node']

        # distance between the two nodes.
        self.distance = connection['distance']

        # connection's data-rate and latency.
        self.data_rate = connection['data_rate']
        self.latency = connection['latency']

    def get_name(self):
        return self.name

    def get_src_node(self):
        return self.src_node

    def get_dest_node(self):
        return self.dest_node

    def get_info(self):
        return {
```

```
            'distance': self.distance,
            'latency': self.latency,
            'data_rate': self.data_rate
        }

    def get_distance(self):
        return self.distance

    def get_latency(self):
        return self.latency

    def get_data_rate(self):
        return self.data_rate

    def __repr__(self):
        return 'PPC_Connection() [' + self.name + ']'

    def __str__(self):
        return 'PPC_Connection() [' + self.name + ']'
```

Κλάση PPC_Topology (ppcomm_topology.py)

Ένα αντικείμενο της κλάσης PPC_Topology αναπαριστά την τοπολογία του PPComm. Κατά την δημιουργία ενός αντικειμένου, αποθηκεύονται οι ρυθμίσεις και το μοντέλο του PPComm που δηλώνει ο χρήστης στο αρχείο docker-compose.yaml. Επίσης, δημιουργείται ένας σταθμισμένος κατευθυνόμενος γράφος (weighted directed graph) στον οποίο γίνετε προσθήκη όλων των κόμβων/συσκευών (PPC_Node) και όλων των συνδέσεων (PPC_Connection) που γίνονται μεταξύ τους. Το βάρος που χρησιμοποιείται στον γράφο είναι η καθυστέρηση (latency) των συνδέσεων. Η κλάση αυτή επιτρέπει την ανάκτηση των πιο πάνω πληροφοριών, την προσθήκη νέων κόμβων/συσκευών στον γράφο και την προσθήκη νέων συνδέσεων. Τέλος, περιλαμβάνει μέθοδο που επιτρέπει την εύρεση μονοπατιού μεταξύ δύο κόμβων/συσκευών με βάση την χαμηλότερη καθυστέρηση (latency), εφόσον υπάρχει τέτοιο μονοπάτι.

```
import networkx as nx
import ppcomm.ppcomm_netops as netops
from ppcomm.ppcomm_node import PPC_Node
from ppcomm.ppcomm_connection import PPC_Connection
from itertools import product

# represents the topology of ppcomm and includes
# the graph that contains all nodes and connections.
```

```
class PPC_Topology:
    def __init__(self, data):
        # x-ppcomm in docker-compose yaml file.
        self.ppcomm = data

        # graph that saves nodes and connections(edges).
        self.graph = nx.DiGraph()

        self.generate_topology()

    def get_graph(self):
        return self.graph

    # get the fogify network that will be used for the
    # ppcomm point-to-point connections (using network links).
    def get_network(self):
        return self.ppcomm['config']['network']

    # get the coordination type that is used in this configuration.
    # should be specified in x-ppcomm (latlong/xyz)
    def get_coord_type(self):
        return self.ppcomm['config']['coord_type']

    # get the find_path boolean that is used in this configuration.
    # should be specified in x-ppcomm (true/false).
    # allows nodes to communicate through other nodes by finding the shortest path.
    def get_find_path(self):
        return self.ppcomm['config']['find_path']

    # get the minimum snr to establish a connection.
    # can be configured by the user for better flexibility.
    #  5dB to 10dB: is below the minimum level to establish a connection,
    #               due to the noise level being nearly indistinguishable
    #               from the desired signal(useful information).
    # 10dB to 15dB: is the accepted minimum to establish an unreliable connection.
    # 15dB to 25dB: is typically considered the minimally acceptable
    #               level to establish poor connectivity.
    # 25dB to 40dB: is deemed to be good.
    # 41dB or higher: is considered to be excellent.
    # https://resources.pcb.cadence.com/blog/2020-what-is-signal-to-noise-ratio-and-
how-to-calculate-it
    def get_minimum_snr(self):
        return self.ppcomm['config']['minimum_snr']

    def add_node(self, index, data):
        self.graph.add_node(index, data=PPC_Node(data))

    def get_nodes(self):
        return self.graph.nodes

    def get_node(self, index):
        # return nx.get_node_attributes(self.graph, 'data')[index]
        return self.graph.nodes[index]['data']

    def get_total_nodes(self):
        return self.graph.number_of_nodes()

    def add_connection(self, src_node, dest_node, data):
        # save latency at graph edge's data, so it can be used to find
        # the shortest path based on latency.
        self.graph.add_edge(src_node, dest_node, data=PPC_Connection(data),
latency=data['latency'])

    def get_connections(self):
        return self.graph.edges
```

```
    def get_connection(self, index):
        # return nx.get_edge_attributes(self.graph, 'data')[index]
        return self.graph.edges[index]['data']

    def has_connection(self, index):
        try:
            self.get_connection(index)
            return True
        except KeyError:
            return False

    def get_total_connections(self):
        return self.graph.number_of_edges()

    # find the shortest weighted(using latency) path and length between two nodes.
    # returns: total latency, path
    def find_path(self, src_node, dest_node):
        try:
            return nx.single_source_dijkstra(self.graph, src_node, dest_node, 1000,
'latency')
        except nx.NetworkXNoPath:
            return -1, []

    def generate_topology(self, update_topology=False):
        # if creating topology, import nodes from x-ppcomm configuration file.
        if not update_topology:
            nodes = self.ppcomm['nodes']
            for index, data in enumerate(nodes):
                self.add_node(index, data)

        # if updating topology, remove all connections and calculate them again.
        if update_topology:
            connections_list = []
            for cid in self.get_connections():
                connections_list.append(cid)
            self.graph.remove_edges_from(connections_list)

        total_nodes = self.get_total_nodes()

        for n1, n2 in product(range(total_nodes), range(total_nodes)):
            src_node = self.get_node(n1)
            dest_node = self.get_node(n2)

            if n1 == n2:
                continue

            # calculate distance based on selected coord_type option.
            if self.get_coord_type() == 'latlong':
                distance = netops.distance_geol(src_node, dest_node)
            else:
                distance = netops.distance_xyz(src_node, dest_node)

            if distance >= 0:
                noise = netops.noise(src_node, dest_node)
                pathloss = netops.pathloss(src_node, distance)
                receiver_power = netops.receiver_power(src_node, dest_node, pathloss)
                snr = netops.snr(receiver_power, noise)

                # add new connection if snr is greater than the user's configured
value in x-ppcomm.
                if snr >= self.get_minimum_snr():
                    data_rate = netops.data_rate(src_node, snr)
                    latency = netops.latency(snr)
                    self.add_connection(n1, n2, {
```

```
                        'name': '(' + str(n1) + ',' + str(n2) + ')',
                        'src_node': src_node,
                        'dest_node': dest_node,
                        'distance': distance,
                        'data_rate': data_rate,
                        'latency': latency
                    })

        # if x-ppcomm's find_path option is enabled, try to find a path for two nodes
using intermediate nodes.
        if self.get_find_path():
            for n1, n2 in product(range(total_nodes), range(total_nodes)):
                src_node = self.get_node(n1)
                dest_node = self.get_node(n2)

                if n1 == n2:
                    continue

                if not self.has_connection((n1, n2)):
                    latency, path = self.find_path(n1, n2)

                    if len(path) > 0:
                        total_distance = 0
                        total_latency = 0
                        max_data_rate = 1000000000
                        for index, value in enumerate(path[:len(path) - 1]):
                            # value: current node id in path
                            # path[index+1]: next node id in path
                            connection = self.get_connection((value, path[index +
1]))

                            total_distance += connection.get_distance()
                            total_latency += connection.get_latency()
                            data_rate = connection.get_data_rate()
                            if data_rate < max_data_rate:
                                max_data_rate = data_rate

                        self.add_connection(n1, n2, {
                            'name': '(' + str(n1) + ',' + str(n2) + ')',
                            'src_node': src_node,
                            'dest_node': dest_node,
                            'distance': total_distance,
                            'data_rate': max_data_rate,
                            'latency': total_latency
                        })

    def __str__(self):
        return 'PPC_Topology()'
```

Κλάση PPComm (ppcomm.py)

Ένα αντικείμενο της κλάσης PPComm αναλαμβάνει την δημιουργία σύνδεσης με το Fogify Controller μέσω του FogifySDK και την δημιουργία ενός αντικειμένου PPC_Topology. Ωστόσο, πριν γίνουν τα πιο πάνω, γίνεται έλεγχος της εγκυρότητας του μοντέλου που υπάρχει στο αρχείο docker-compose.yaml,

και εφόσον είναι έγκυρο, δημιουργείται νέο αρχείο yaml στο οποίο μετατρέπεται το μοντέλο του PPComm σε μοντέλο του Fogify. Ακολούθως γίνεται η σύνδεση με το Fogify Controller, χρησιμοποιώντας το νέο αρχείο yaml. Τέλος, αποθηκεύεται η λίστα με τα διαθέσιμα trajectories. Μέσω της κλάσης αυτής ο χρήστης μπορεί να εκτελέσει κάποιο trajectory, να αλλάξει τις συντεταγμένες ενός κόμβου, να αποκτήσει πρόσβαση στην τοπολογία του PPComm και στο API του FogifySDK. Το τελευταίο είναι ιδιαίτερα σημαντικό καθώς μέσω της PPComm, είναι διαθέσιμες όλες οι μέθοδοι του FogifySDK API και έτσι ο χρήστης δεν περιορίζεται.

```
import yaml
from sys import exit
from copy import deepcopy
from time import sleep
from ppcomm.ppcomm_topology import PPC_Topology
from FogifySDK import FogifySDK

class PPCommError(Exception):
    pass

class PPComm:
    def __init__(self, controller, yaml_file):
        # read x-ppcomm and x-fogify configuration from docker-compose.yaml.
        try:
            stream = open(yaml_file, 'r')
            self.yaml_data = yaml.safe_load(stream)
            stream.close()
        except FileNotFoundError:
            raise PPCommError('We could not find the specified yaml file.')

        self.check_data()
        self.topology = PPC_Topology(self.yaml_data['x-ppcomm'])
        self.generate_new_yaml_file()

        # import trajectories from x-ppcomm.
        self.trajectories = {}
        if 'trajectories' in self.yaml_data['x-ppcomm'] \
                and self.yaml_data['x-ppcomm']['trajectories'] is not None:
            for trajectory in self.yaml_data['x-ppcomm']['trajectories']:
                name = trajectory['name']
                self.trajectories[name] = {
                    'changes': trajectory['changes']
                }

        # initialize connection with fogify's controller
        # using the new yaml generated by ppcomm.
        self.fogify_sdk = FogifySDK(controller, 'ppcomm-output.yaml')

    def get_fogify_sdk(self):
        return self.fogify_sdk

    def get_fogify_network_data_rate(self):
```

```
        for network in self.yaml_data['x-fogify']['networks']:
            if network['name'] == self.topology.get_network():
                return network['downlink']['bandwidth']
        return -1

    def generate_new_yaml_file(self):
        output = deepcopy(self.yaml_data)
        del output['x-ppcomm']

        # create topology if it does not exist.
        if ('topology' in output['x-fogify'] and output['x-fogify']['topology'] is
None) \
                or 'topology' not in output['x-fogify']:
            output['x-fogify']['topology'] = []

        # create networks if it does not exist.
        if ('networks' in output['x-fogify'] and output['x-fogify']['networks'] is
None) \
                or 'networks' not in output['x-fogify']:
            output['x-fogify']['networks'] = []

        # add ppcomm's nodes in fogify's topology.
        for nid in self.topology.get_nodes():
            node = self.topology.get_node(nid)

            output['x-fogify']['topology'].append({
                'label': node.get_name(),
                'node': node.get_fogify_node(),
                'service': node.get_fogify_service(),
                'replicas': 1,
                'networks': [self.topology.get_network()]
            })

        # add ppcomm's connections in fogify's network links.
        fg_network_data_rate = self.get_fogify_network_data_rate()
        for cid in self.topology.get_connections():
            connection = self.topology.get_connection(cid)

            src_name = connection.get_src_node().get_name()
            dest_name = connection.get_dest_node().get_name()
            delay = str(int(connection.get_latency())) + 'ms'
            bandwidth = str(int(connection.get_data_rate() / 1000000)) + 'Mbps'

            # if connection data_rate bigger than fogify's network data_rate then
limit to fogify's one.
            if bandwidth > fg_network_data_rate != -1:
                bandwidth = self.get_fogify_network_data_rate()

            for network in output['x-fogify']['networks']:
                if network['name'] == self.topology.get_network():
                    if ('links' in network and network['links'] is None) or 'links'
not in network:
                        network['links'] = []

                    network['links'].append({
                        'from_node': src_name,
                        'to_node': dest_name,
                        'bidirectional': False,
                        'properties': {
                            'bandwidth': bandwidth,
                            'latency': {
                                'delay': delay
                            }
                        }
                    })
```

```
                break

        # generate the new yaml file.
        with open('ppcomm-output.yaml', 'w') as output_file:
            yaml.dump(output, output_file, default_flow_style=False)


    # check if x-ppcomm configuration is valid.
    def check_data(self):
        data = self.yaml_data

        if 'x-ppcomm' not in data.keys():
            raise PPCommError('Could not find the \'x-ppcomm\' configuration in the
specified yaml file.')

        if data['x-ppcomm'] is None:
            raise PPCommError('\'x-ppcomm\' is empty. Required: config, nodes.
Optional: trajectories.')

        if 'config' not in data['x-ppcomm']:
            raise PPCommError('We could not find the \'config\' object of x-ppcomm.')

        if data['x-ppcomm']['config'] is None:
            raise PPCommError('\'config\' is empty. Required: coord_type, network,
find_path, minimum_snr.')

        if 'coord_type' not in data['x-ppcomm']['config']:
            raise PPCommError('\'coord_type\' is not set. Possible values:
\'latlong\', \'xyz\'.')

        if data['x-ppcomm']['config']['coord_type'] not in ['latlong', 'xyz']:
            raise PPCommError('\'coord_type\' value is not valid.')

        if 'network' not in data['x-ppcomm']['config']:
            raise PPCommError('\'network\' is not set. Possible values: a network
defined in x-fogify.')

        if 'find_path' not in data['x-ppcomm']['config']:
            raise PPCommError('\'find_path\' is not set. Possible values:
true/false.')

        if 'minimum_snr' not in data['x-ppcomm']['config']:
            raise PPCommError('\'minimum_snr\' is not set. Check documentation for
possible values.')

        if 'nodes' not in data['x-ppcomm']:
            raise PPCommError('We could not find the \'nodes\' object of x-ppcomm.')

        if data['x-ppcomm']['nodes'] is None:
            raise PPCommError('\'nodes\' object of x-ppcomm is empty.')

        ppcomm_network = data['x-ppcomm']['config']['network']
        network_exists = False
        for network in data['x-fogify']['networks']:
            if network['name'] == ppcomm_network:
                network_exists = True
                break

        if not network_exists:
            raise PPCommError('\'x-fogify\' configuration does not have a network
named \'' + ppcomm_network + '\'.')

    # get ppcomm's topology as Python Dictionary.
    def get_topology(self):
        output = {'nodes': {}, 'connections': {}}
```

```
        for index in self.topology.get_nodes():
            node = self.topology.get_node(index)
            output['nodes'][node] = {
                'name': node.get_name(),
                'fogify_node': node.get_fogify_node(),
                'fogify_service': node.get_fogify_service(),
                'coords': node.get_coords(),
                'specs': node.get_specs()
            }

        for index in self.topology.get_connections():
            connection = self.topology.get_connection(index)
            output['connections'][connection] = {
                'name': connection.get_name(),
                'src_node': connection.get_src_node(),
                'dest_node': connection.get_dest_node(),
                'specs': connection.get_info()
            }

        return output

    # get ppcomm's topology pretty printed.
    def print_topology(self):
        print('Nodes (' + str(self.topology.get_total_nodes()) + '):')
        for nid, node_ref in self.topology.get_nodes().items():
            node = node_ref['data']
            print('\t' + node.get_name() + ' | Coordinates: ' +
str(node.get_coords()))

        print('\nConnections (' + str(self.topology.get_total_connections()) + '):')
        for cid, connection_ref in self.topology.get_connections().items():
            connection = connection_ref['data']
            src_name = connection.get_src_node().get_name()
            dest_name = connection.get_dest_node().get_name()
            print('\t' + src_name + ' to ' + dest_name + ' | ' +
str(connection.get_info()))

        print()

    # get ppcomm's trajectories as Python Dictionary.
    def get_trajectories(self):
        return self.trajectories

    # get ppcomm's trajectories pretty printed.
    def print_trajectories(self):
        for name, trajectory in self.trajectories.items():
            print('Name: ' + name + '\nChanges:')
            for change in trajectory['changes']:
                print('- Time: ' + str(change['time']) + ' seconds')
                for node in change['nodes']:
                    print('\t\'' + node['name'] + '\' moves to ' +
str(node['coords']))
            print()

    # execute trajectory by name.
    def exec_trajectory(self, name):
        if name not in self.trajectories:
            raise PPCommError('We could not find a trajectory named \'' + name +
'\'.')

        trajectory = self.trajectories[name]
        prev_time = 0

        print('Executing trajectory \'' + name + '\'.')
```

```
        for change in trajectory['changes']:
            # save current graph before updating, so we can compare the differences.
            old_graph = deepcopy(self.topology.get_graph())

            # waiting the amount of time configured in trajectory options.
            time_to_sleep = change['time'] - prev_time
            prev_time = change['time']
            sleep(time_to_sleep)

            print('At ' + str(change['time']) + ' seconds:')
            # update the node positions in the ppcomm topology.
            for node_change in change['nodes']:
                node_found = False
                for nid, node_ref in self.topology.get_nodes().items():
                    node = node_ref['data']

                    if node.get_name() == node_change['name']:
                        node_found = True
                        if self.topology.get_coord_type() == 'latlong':
                            lat = node_change['coords']['latitude']
                            long = node_change['coords']['longitude']
                            node.set_coords_geol(lat, long)
                        else:
                            x = node_change['coords']['x']
                            y = node_change['coords']['y']
                            z = node_change['coords']['z']
                            node.set_coords_xyz(x, y, z)

                        print('\tNode \'' + node.get_name() + '\' moved to ' +
str(node.get_coords()) + '.')

                # if node name is not valid, notify and proceed to next one.
                if not node_found:
                    print('\tNode \'' + node_change['name'] + '\' does not exist.')

            print('\nUpdating topology...', end = '\r')
            # update the ppcomm's topology.
            self.topology.generate_topology(True)
            print('Topology updated!   \n')

            print('Updating fogify network links...')
            # update the fogify network links.
            self.update_fogify_links(old_graph)
            print('Fogify network links updated!\n')

    # change the coordinates of a node.
    def change_node_coords(self, name, coords):
        # save current graph before updating, so we can compare the differences.
        old_graph = deepcopy(self.topology.get_graph())

        node_found = False
        for nid, node_ref in self.topology.get_nodes().items():
            node = node_ref['data']

            if node.get_name() == name:
                node_found = True
                if self.topology.get_coord_type() == 'latlong':
                    lat = coords['latitude']
                    long = coords['longitude']
                    node.set_coords_geol(lat, long)
                else:
                    x = coords['x']
                    y = coords['y']
                    z = coords['z']
                    node.set_coords_xyz(x, y, z)
```

```
                print('Node \'' + node.get_name() + '\' moved to ' +
str(node.get_coords()) + '.')

        # if node name is not valid, notify and proceed to next one.
        if not node_found:
            raise PPCommError('Node \'' + name + '\' does not exist.')

        print('\nUpdating topology...', end='\r')
        # update the ppcomm's topology.
        self.topology.generate_topology(True)
        print('Topology updated!   \n')

        print('Updating fogify network links...')
        # update the fogify network links.
        self.update_fogify_links(old_graph)
        print('Fogify network links updated!\n')

    # uses the FogifySDK instance to add/update the network links
    # that represent the point-to-point connection of ppcomm' nodes.
    def update_fogify_links(self, old_graph):
        network = self.topology.get_network()
        old_graph_connections = old_graph.edges.items()
        new_graph_connections = self.topology.get_connections().items()

        # check connections in the updated topology.
        for cid, c_ref in new_graph_connections:
            connection = c_ref['data']

            from_node = connection.get_src_node().get_name()
            to_node = connection.get_dest_node().get_name()
            properties = {
                'bandwidth': connection.get_data_rate(),
                'latency': {
                    'delay': connection.get_latency()
                }
            }

            # if a connection is not in the old graph, then add link to fogify.
            # else a connection was already present and needs to be updated.
            if cid not in old_graph.edges:
                parameters = { 'properties': properties }
                result = self.fogify_sdk.add_link(network, from_node, to_node,
parameters, False)
                print('Link added: ' + str(result))
            else:
                result = self.fogify_sdk.update_link(network, from_node, to_node,
properties, False)
                print('Link updated: ' + str(result))

        # check connections in the old topology (old graph).
        for cid, c_ref in old_graph_connections:
            connection = c_ref['data']

            from_node = connection.get_src_node().get_name()
            to_node = connection.get_dest_node().get_name()
            properties = {
                'bandwidth': 0,
                'latency': {
                    'delay': 99999
                }
            }

            # if a connection of the old graph is not in the new graph,
            # then remove it from fogify's network links. Because it is
```

```
            # not possible to remove a fogify network link, then we update
            # the link with a very high latency and 0 bandwidth, making it
            # practically non-existent.
            # this is a work-around until fogifySDK allows the removal of links.
            if cid not in self.topology.get_connections():
                result = self.fogify_sdk.update_link(network, from_node, to_node,
properties, False)
                print('Link removed: ' + str(result))
```

Αρχείο ppcomm_netops.py

Το αρχείο ppcomm_netops.py περιλαμβάνει όλες τις μαθηματικές πράξεις που πρέπει να εκτελεστούν για να καθοριστούν οι συνδέσεις μεταξύ των κόμβων/συσκευών. Περισσότερες λεπτομέρειες για τις πιο κάτω πράξεις δίνονται στην ενότητα του Μαθηματικού Υπόβαθρου στο Κεφάλαιο 2.

```
from math import sqrt, radians, sin, cos, asin, log, log10, exp

# calculate distance between two points in three-dimensional coordinate system.
def distance_xyz(src_node, dest_node):
    # get nodes' coordinates.
    p1 = src_node.get_coords()
    p2 = dest_node.get_coords()

    return sqrt((p2['x'] - p1['x']) ** 2 + (p2['y'] - p1['y']) ** 2 + (p2['z'] -
p1['z']) ** 2)

# calculate distance between two points in geological coordinate system.
def distance_geol(src_node, dest_node):
    # get nodes' coordinates.
    p1 = src_node.get_coords()
    p2 = dest_node.get_coords()

    lat1 = radians(p1['latitude'])
    lat2 = radians(p2['latitude'])
    lon1 = radians(p1['longitude'])
    lon2 = radians(p2['longitude'])

    dlat = lat2 - lat1
    dlon = lon2 - lon1

    a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) ** 2
    c = 2 * asin(sqrt(a))

    # Radius of earth in kilometers. Use 3956 for miles
    r = 6371

    # calculate the result in meters.
    return c * r * 1000

# calculate signal noise between the two nodes.
# using bandwidth(hz) of source node and ue noise figure of destination node.
def noise(src_node, dest_node):
```

```
    bandwidth_hz = src_node.get_bandwidth() * 1000000000  # from ghz to hz.
    return -174 + 10 * log10(bandwidth_hz) + dest_node.get_ue_noise_figure()

# calculate signal pathloss according to distance of the two nodes and source node's
carrier frequency.
def pathloss(src_node, distance):
    # in case two nodes are in the same position.
    if distance == 0:
        distance = 0.001

    # wrong: return 28 + 22 * log(distance) + 20 *
log(src_node.get_carrier_frequency())
    return 28 + 22 * log10(distance) + 20 * log10(src_node.get_carrier_frequency())

# calculate receiver's power in db.
# need more info about this.
def receiver_power(src_node, dest_node, _pathloss):
    return src_node.get_transmit_power() + src_node.get_ru_antenna_gain() +
dest_node.get_ue_antenna_gain() - _pathloss

# calculate signal-to-noise ratio.
def snr(_receiver_power, _noise):
    return _receiver_power - _noise

# calculate connection's data-rate between two nodes.
def data_rate(node, _snr):
    bandwidth_hz = node.get_bandwidth() * 1000000000  # from ghz to hz
    return bandwidth_hz * log(1 + _snr)

# calculate connection's latency between two nodes.
def latency(_snr):
    return (58.854 * exp(-0.055837 * _snr)) / 2
```