ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Υποστήριξη αποδοτικής δημιουργίας μικρό-πρακτόρων σε ετερογενή Ασύρματα Δίκτυα Αισθητήρων

Supporting efficient micro-agent creation in heterogeneous Wireless Sensor Networks

Διπλωματική Εργασία

Αθανάσιος Γρηγορόπουλος

Επιβλέποντες Καθηγητές: Σπυρίδων-Γεράσιμος Λάλης
Αναπληρωτής Καθηγητής Π.Θ.

Πέτρος Λάμψας
Αναπληρωτής Καθηγητής
Τμήμα Πληροφορικής και Τεχνολογίας Υπολογιστών
ΤΕΙ Λαμίας

ΒΟΛΟΣ 2013

Ευχαριστίες

Με την περάτωση της παρούσας Διπλωματικής Εργασίας, θα ήθελα να ευχαριστήσω τον επιβλέποντα μου κ. Σπύρο Λάλη για την καθοδήγηση που μου παρείχε κατά τη διάρκεια εκπόνησής της καθώς και για την γενικότερη εμπιστοσύνη που έχει επιδείξει στο πρόσωπό μου.

Επίσης, ευχαριστώ τους φίλους και συμφοιτητές μου για την αμέριστη υποστήριξή τους και την δημιουργία ενός ευχάριστου και δημιουργικού περιβάλλοντος κατά τη διάρκεια των σπουδών μου.

Τέλος, οφείλω ένα μεγάλο ευχαριστώ στην οικογένειά μου, Σπύρο, Εύα και Αλεξία που με βοήθησαν να διαμορφώσω ένα ισχυρό υπόβαθρο και βρίσκονται πάντα δίπλα μου όλα αυτά τα χρόνια.

Περιεχόμενα

Κεφάλαιο 1 Εισαγωγή 5

1.1 Πλαίσιο και Σκοπός της Διπλωματικής Εργασίας 5

1.2 Διάρθρωση της Διπλωματικής Εργασίας 5

Κεφάλαιο 2 Βασικές έννοιες 7

2.1 Ασύρματα Δίκτυα Αισθητήρων (WSNs) 7

2.2 nesC 9

2.3 TinyOS 11

Κεφάλαιο 3 POBICOS 14

3.1 Στόχος-Βασικά χαρακτηριστικά 14

3.2 POBICOS API 15

3.3 Λογική δομή των εφαρμογών POBICOS και μικρό-πράκτορες 17

3.4 Δομή λογισμικού στους κόμβους 19

Κεφάλαιο 4 Δημιουργία ομάδας μικρό-πρακτόρων στο POBICOS 22

4.1 Εισαγωγή στη δημιουργία πρακτόρων στο POBICOS 22

4.2 Υπάρχον πρωτόκολλο δημιουργίας ομάδας πρακτόρων 24

4.2.1 Host Candidate Discovery Protocol 25

4.2.2 Agent Creation Protocol 26

4.3 Πιο αποδοτική δημιουργία ομάδας μικρό-πρακτόρων 28

4.3.1 Πρωτόκολλο 28

4.3.2 Μηνύματα 29

4.3.3 Διάγραμμα ροής μηνυμάτων 30

4.4 Κύριες διαφορές νέου και παλιού πρωτοκόλλου 31

Κεφάλαιο 5 Πειραματικά Αποτελέσματα 33

5.1 Εργαλεία 33

5.1.1 TOSSIM 33

5.1.2 IntelMote2 (iMote2) 33

5.2 Εφαρμογές δημιουργίας πρακτόρων στο POBICOS 34

5.3 Πειραματικά αποτελέσματα στο TOSSIM 34

5.3.1 Τοπολογία αστεριού 36

5.3.2 Τοπολογία αλυσίδας 37

5.4 Πειραματικά αποτελέσματα σε πραγματικούς κόμβους 40
5.4.1 Τοπολογία αστεριού 41
5.4.2 Τοπολογία αλυσίδας 41
5.5 Σύγκριση νέου και παλιού πρωτοκόλλου 42
5.5.1 Τοπολογία αστεριού 44
5.5.2 Τοπολογία αλυσίδας 45
5.6 Σύνοψη αποτελεσμάτων 47
Κεφάλαιο 6 Επίλογος 48
6.1 Σύνοψη 48
6.2 Μελλοντική εργασία 48
Βιβλιογραφία 49

Κεφάλαιο 1 Εισαγωγή

1.1 Πλαίσιο και Σκοπός της Διπλωματικής Εργασίας

Τα Ασύρματα Δίκτυα Αισθητήρων (WSNs), των οποίων η ανάπτυξη παρακινήθηκε από στρατιωτικές εφαρμογές, σήμερα γνωρίζουν μεγάλη ανάπτυξη καθώς χρησιμοποιούνται σε πολλές βιομηχανικές και καταναλωτικές εφαρμογές όπως η παρακολούθηση και ο έλεγχος της βιομηχανικής διαδικασίας, η συντήρηση μηχανημάτων και ο έλεγχος της γεωργικής παραγωγής.

Ένα άλλο πεδίο εφαρμογών, που αναπτύσσεται όλο και περισσότερο, είναι αυτό του ελέγχου των «έξυπνων» σπιτιών (smart home monitoring) όπου ο έλεγχος επιτυγχάνεται με τη χρήση αισθητήρων και ελεγκτών που είναι ενσωματωμένοι σε ετερογενή αντικείμενα καθημερινής χρήσης τα οποία αποτελούν ένα WSN.

Μια πλατφόρμα που αναπτύχθηκε για την υποστήριξη των εφαρμογών σε αυτό το πεδίο είναι αυτή του ενδιάμεσου λογισμικού POBICOS, στο οποίο λογικές μονάδες κάθε εφαρμογής είναι οι μικρό-πράκτορες. Πιο συγκεκριμένα, κάθε εφαρμογή αποτελείται από μια ομάδα πρακτόρων που οργανώνονται σε ένα ιεραρχικό δέντρο, τρέχουν και αλληλεπιδρούν με ταυτόχρονο (παράλληλο) τρόπο.

Σκοπός της εργασίας είναι ο σχεδιασμός και η υλοποίηση ενός πρωτοκόλλου για την υποστήριξη αποδοτικής δημιουργίας μικρό-πρακτόρων σε τέτοιου είδους ετερογενή Ασύρματα Δίκτυα Αισθητήρων που υποστηρίζουν το ενδιάμεσο λογισμικό POBICOS.

1.2 Διάρθρωση της Διπλωματικής Εργασίας

Στο Κεφάλαιο 2 γίνεται μια εισαγωγή στα Ασύρματα Δίκτυα Αισθητήρων και τις κύριες τεχνολογίες που χρησιμοποιήθηκαν για την ολοκλήρωση της Διπλωματικής Εργασίας οι οποίες αποτελούνται από την γλώσσα προγραμματισμού nesC και το λειτουργικό σύστημα TinyOS.

Στο Κεφάλαιο 3 παρουσιάζεται το ενδιάμεσο λογισμικό POBICOS και τα βασικά χαρακτηριστικά του.

Στο Κεφάλαιο 4 παρουσιάζεται το υπάρχον πρωτόκολλο δημιουργίας ομάδας μικρό-πρακτόρων στο POBICOS και το πρωτόκολλο για την πιο αποδοτική δημιουργία τους που υλοποιήθηκε στα πλαίσια της Διπλωματικής Εργασίας ενώ αναδεικνύονται και οι κύριες διαφορές τους.

Στο Κεφάλαιο 5 αναφέρονται τα πειραματικά αποτελέσματα του αναπτυχθέντος πρωτοκόλλου δημιουργίας ομάδας μικρό-πρακτόρων τόσο σε περιβάλλον προσομοίωσης, όσο και σε πραγματικούς κόμβους και μέσω της σύγκρισης με το προϋπάρχον πρωτόκολλο αναδεικνύεται η βελτίωση της αποδοτικότητας που επιτυγχάνεται με τη χρήση του.

Στο Κεφάλαιο 6 συνοψίζονται τα κίνητρα και τα αποτελέσματα της Διπλωματικής Εργασίας, ενώ αναφέρονται και κάποιες μελλοντικές επεκτάσεις που θα μπορούσαν να γίνουν προς την ίδια κατεύθυνση.

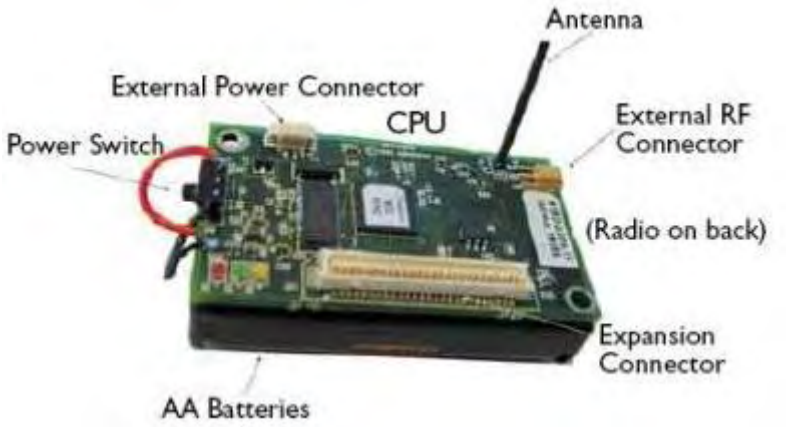
Κεφάλαιο 2 Βασικές έννοιες

2.1 Ασύρματα Δίκτυα Αισθητήρων (WSNs)

Οι πρόσφατες εξελίξεις στις τεχνολογίες ανίχνευσης (κατόπτευσης), υπολογισμού και επικοινωνίας σε συνδυασμό με την συνεχή παρακολούθηση των φυσικών φαινομένων οδήγησε στην ανάπτυξη των Ασύρματων Δικτύων Αισθητήρων. Ένα **Ασύρματο Δίκτυο Αισθητήρων (Wireless Sensor Network-WSN)** αποτελείται από ένα σύνολο κόμβων με ενσωματωμένους αισθητήρες που οργανώνονται σε ένα δίκτυο συνεργασίας. Σκοπός τέτοιων δικτύων είναι η παρακολούθηση φυσικών ή περιβαλλοντικών συνθηκών όπως η θερμοκρασία, ο ήχος, η πίεση κ.λπ. και η αποστολή των σχετικών δεδομένων σε μια κύρια τοποθεσία όπου πραγματοποιείται η επιπλέον επεξεργασία τους. Επιπλέον, ανάλογα με τη φύση της εφαρμογής, υπάρχουν δίκτυα που επιτρέπουν και τον έλεγχο της δραστηριότητας του αισθητήρα με τη χρήση ενεργοποιητών.

Αναλύοντας την δομή των **κόμβων (nodes)** του δικτύου, τα βασικά χαρακτηριστικά τους είναι τα παρακάτω:

i. ικανότητα επεξεργασίας μέσω ενός ή περισσοτέρων μικροελεγκτών (microcontrollers-MCU), επεξεργαστών (CPU) ή DSP chip,
ii. πολλαπλά είδη μνήμης όπως προγράμματος, δεδομένων και flash μνήμες,
iii. ένας πομποδέκτης RF, συνήθως μια omni-directional κεραία,
iv. μία ή περισσότερες πηγές ενέργειας, π.χ. μπαταρίες και ηλιακές κυψέλες και
v. δυνατότητα φιλοξενίας διάφορων αισθητήρων (sensors) και ενεργοποιητών (actuators).



Σχ. 1: Ενδεικτική δομή κόμβου Ασύρματου Δικτύου Αισθητήρων.

Το μέγεθός των κόμβων ενδέχεται να ποικίλει από αυτό ενός κουτιού παπουτσιών ως αυτό ενός κόκκου σκόνης, αν και λειτουργικοί κόμβοι μικροσκοπικών διαστάσεων δεν έχουν αναπτυχθεί ακόμα. Το κόστος τους είναι επίσης μεταβαλλόμενο και κυμαίνεται από λίγα έως εκατοντάδες δολάρια/ευρώ, ανάλογα με την πολυπλοκότητα των αισθητήρων που χρησιμοποιούνται. Οι παραπάνω περιορισμοί μεγέθους και κόστους έχουν ως αποτέλεσμα αντίστοιχους περιορισμούς σε πόρους όπως η ενέργεια, η μνήμη, η υπολογιστική ισχύς και το εύρος ζώνης.

Οι κόμβοι επικοινωνούν ασύρματα και αφού παραταχθούν με ένα αδόμητο (ad hoc) τρόπο στην περιοχή εφαρμογής, συχνά αυτό-οργανώνονται και σχηματίζουν μια κατάλληλη δομή ώστε από κοινού να εκτελέσουν μια συγκεκριμένη εργασία. Η τοπολογία των WSNs, δηλαδή, μπορεί να ποικίλει από ένα απλό δίκτυο αστεριού (star network) ως ένα δίκτυο πλέγματος πολλαπλών αλμάτων (multi-hop mesh network) ενώ στις περισσότερες των περιπτώσεων μεταβάλλεται δυναμικά είτε από εξωγενείς παράγοντες (π.χ. καταστροφή ή μετακίνηση κόμβων λόγω φυσικών φαινομένων ή παρεμβολής ζωντανών οργανισμών) είτε από ενδογενείς παράγοντες (π.χ. δίκτυο από κόμβους που έχουν δυνατότητα μετακίνησης). Η μετάδοση πακέτων μεταξύ των hop του δικτύου μπορεί να γίνεται με δρομολόγηση (routing) ή πλημμύρα (flooding).

Υπάρχουν διάφοροι ρόλοι που μπορεί να έχει ένας κόμβος στο δίκτυο:

i. Τελικός αποδέκτης (sink), εάν σε αυτόν γίνεται η τελική παραλαβή και επεξεργασία των δεδομένων.
ii. Πηγή (source), εάν συλλέγει δεδομένα από το περιβάλλον και τα στέλνει προς τον τελικό αποδέκτη.
iii. Αναμεταδότης (relay), εάν είναι υπεύθυνος για την παραλαβή πακέτων δεδομένων που προέρχονται από τις πηγές και την προώθηση των δεδομένων προς τον τελικό αποδέκτη (με ή χωρίς επεξεργασία).
iv. Πύλη (gateway), εάν επικοινωνεί και με εξωτερικά συστήματα και δίκτυα στα οποία μπορεί να στέλνει τα δεδομένα.

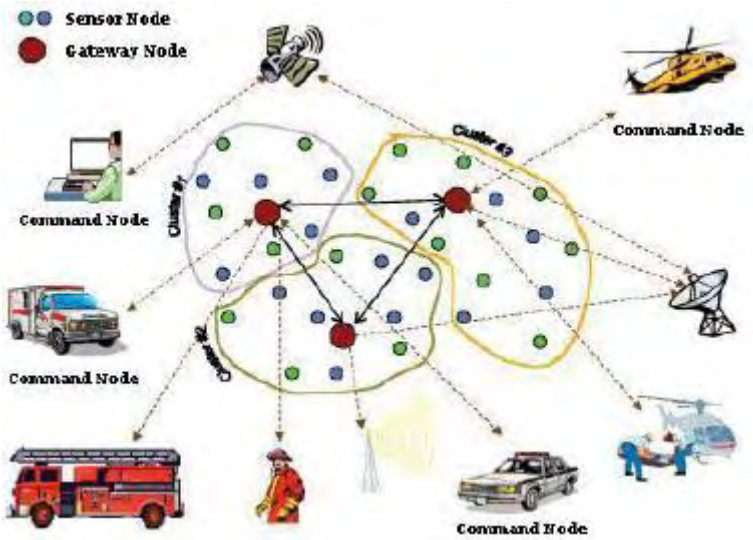
Κάθε κόμβος μπορεί να έχει περισσότερους από έναν ρόλους ανάλογα με τις ιδιότητές του, την θέση του στο δίκτυο και την τρέχουσα εφαρμογή.

Η έρευνα των Ασύρματων Δικτύων Αισθητήρων είναι **εφαρμογοκεντρική** και έχει εστιάσει στους παρακάτω τομείς:

i. Ενεργειακή απόδοση υλικού (hardware) και πρωτοκόλλου σχεδιασμού (protocol design)
ii. Εντοπισμός εναλλακτικών πηγών ενέργειας
iii. Υλοποίηση κατανεμημένων τεχνικών ανίχνευσης (distributed sensing techniques)
iv. Δια-στρωματική βελτιστοποίηση (cross-layer optimization) για την βελτίωση του QoS
v. Εντοπισμός της θέσης των αισθητήρων (localization)
vi. Συγχρονισμός των ρολογιών των κόμβων (clock synchronization)

Επί του παρόντος, τα Ασύρματα Δίκτυα Αισθητήρων αναπτύσσονται με ταχείς ρυθμούς και η δυναμική τους έγκειται στην δυνατότητά που έχουν να δημιουργούν ένα διάχυτο περιβάλλον που επιτρέπει τον έλεγχο και την απομακρυσμένη παρακολούθηση, με τα οφέλη της γνώσης του τί συμβαίνει σε μακρινές ή δυσπρόσιτες περιοχές να είναι σημαντικά.

Όπως αναφέραμε και στην ενότητα 1.1, η ανάπτυξη των WSNs παρακινήθηκε από στρατιωτικές χρήσεις όπως η επιτήρηση πεδίων μαχών, πλέον όμως τέτοια δίκτυα είναι κατάλληλα για πολλές ιατρικές και ψυχαγωγικές εφαρμογές καθώς και για εφαρμογές όπως ο έλεγχος της γεωργικής παραγωγής με σκοπό τη γεωργία ακριβείας, η παρακολούθηση βιοτόπων, η διαχείριση της οδικής κυκλοφορίας, η αυτοματοποίηση, ο εντοπισμός βλαβών-καταστροφών και ο αυτοματισμός σπιτιών.



Σχ. 2: Ενδεικτικές εφαρμογές Ασύρματων Δικτύων Αισθητήρων.

2.2 nesC

Η **nesC** (**n**etwork **e**mbedded **s**ystems **C**) είναι μια επέκταση της γλώσσας προγραμματισμού C που αναπτύχθηκε από μία κοινοπραξία (consortium) υπό την καθοδήγηση του Πανεπιστημίου Berkeley της Καλιφόρνια και απευθύνεται, όπως αναφέρει και το όνομά της, σε δικτυωμένα ενσωματωμένα συστήματα. Ένα παράδειγμα τέτοιου συστήματος είναι τα Ασύρματα Δίκτυα Αισθητήρων.

Η συνεισφορά της nesC σε αυτόν τον τομέα έγκειται στην υποστήριξη των ειδικών αναγκών που προκύπτουν, παρέχοντας ένα προγραμματιστικό μοντέλο που εμπεριέχει αλληλεπίδραση με το περιβάλλον μέσω:

i. εκτέλεσης με βάση τα γεγονότα (event-driven execution),
ii. ευέλικτο μοντέλο ταυτόχρονου προγραμματισμού,
iii. σχεδίαση εφαρμογών βασισμένη σε component (component-oriented design) και
iv. επικοινωνία.

Εφαρμόζοντας βελτιστοποιήσεις σε ολόκληρο το πρόγραμμα (whole-program optimizations) και ανίχνευση data-race προβλημάτων κατά την μεταγλώττιση (compile-time data-race detection) η nesC απλοποιεί την ανάπτυξη εφαρμογών, μειώνει το μέγεθος του κώδικα, και εξαλείφει πολλές πηγές πιθανών σφαλμάτων.

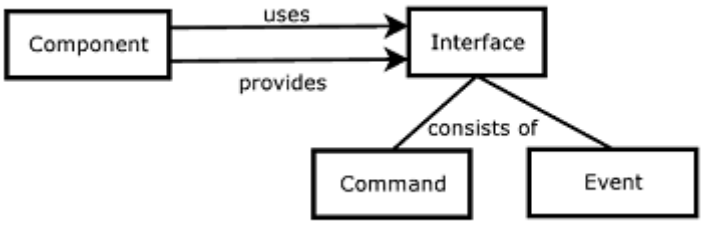
Τα βασικά χαρακτηριστικά της nesC είναι τα εξής:

- Μια nesC εφαρμογή **(nesC application)** αποτελείται από ένα ή περισσότερα εξαρτήματα/συστατικά **(components)** που είναι συνδεδεμένα **("wired")** μεταξύ τους. Υπάρχουν δύο είδη component:
    - **module**, που υλοποιούν μία ή περισσότερες διεπαφές (interfaces) ή/και περιέχουν τον εκτελέσιμο κώδικα
    - **configuration**, που υλοποιούν τις διασυνδέσεις των components (**wiring**)

- Η συμπεριφορά ενός component καθορίζεται από το σύνολο των διεπαφών **(interfaces)** που παρέχει **(provides)** και χρησιμοποιεί **(uses)**. Τα παρεχόμενα interfaces αντιπροσωπεύουν τη λειτουργικότητα που παρέχει το component στο χρήστη ενώ τα χρησιμοποιούμενα αντιπροσωπεύουν τη λειτουργικότητα που χρειάζεται το ίδιο το component για να εκτελέσει την εργασία του.

- Τα interfaces είναι *αμφίδρομα*: ορίζουν ένα σύνολο λειτουργιών που υλοποιούνται από τον πάροχο της διεπαφής **(interface provider)** και ονομάζονται εντολές **(commands)** και ένα σύνολο λειτουργιών που υλοποιούνται από τον χρήστη της διεπαφής **(interface user)** και ονομάζονται γεγονότα **(events)**. Για να έχει την δυνατότητα ένα component να καλεί τα commands ενός interface πρέπει να έχει υλοποιήσει τα events αυτού του interface.

- Τέλος, τα components είναι **στατικά συνδεδεμένα** μεταξύ τους μέσω των interfaces, κάτι που αυξάνει την αποτελεσματικότητα του χρόνου εκτέλεσης, ενθαρρύνει τον «στιβαρό» σχεδιασμό και επιτρέπει την καλύτερη στατική ανάλυση των προγραμμάτων.



Σχ. 3: Σχέση χρήσης των components με τα interfaces στη nesC.

Η nesC χρησιμοποιήθηκε για να υλοποιήσει την κατασκευαστική φιλοσοφία και το μοντέλο εκτέλεσης του TinyOS, ενός λειτουργικού συστήματος για Ασύρματα Δίκτυα Αισθητήρων. Τόσο η nesC όσο και το TinyOS έχουν υιοθετηθεί από μεγάλο αριθμό ερευνητικών ομάδων και τα αποτελέσματα δείχνουν ότι και οι δύο τεχνολογίες είναι ιδιαίτερα αποτελεσματικές σε αυτά τα δικτυωμένα περιβάλλοντα.

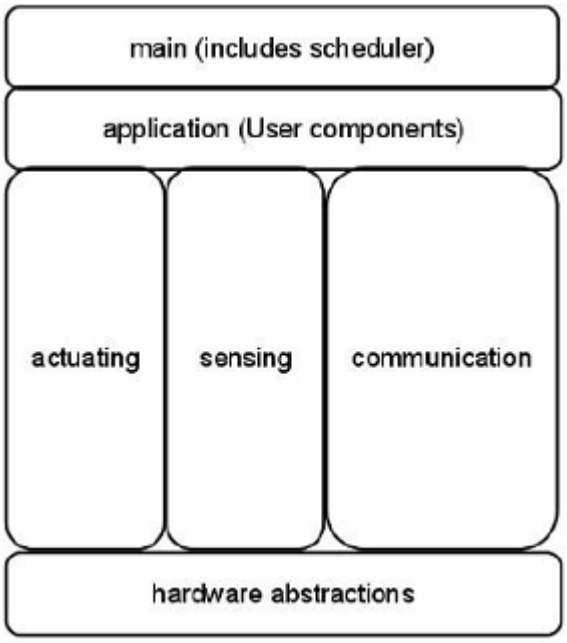
2.3 TinyOS

Το **TinyOS** είναι ένα λειτουργικό σύστημα ελεύθερου και ανοικτού κώδικα (free open source) ειδικά σχεδιασμένο για δίκτυα ενσωματωμένων συστημάτων όπως είναι τα WSNs. Ξεκίνησε ως ένα project του Πανεπιστημίου Berkeley της Καλιφόρνια ως μέρος του προγράμματος DASTA NEST σε συνεργασία με την Intel Research και την Crossbow Technology το 2000 και από τότε έχει επεκταθεί σε παγκόσμιο επίπεδο με τη συμμετοχή χιλιάδων ακαδημαϊκών και εμπορικών προγραμματιστών και χρηστών σχηματίζοντας μία διεθνή κοινοπραξία, την TinyOS Alliance. Οι εφαρμογές του TinyOS είναι γραμμένες σε nesC, τα συμπληρωματικά εργαλεία του σε Java και C και οι σχετικές βιβλιοθήκες σε C.

Τα βασικά χαρακτηριστικά του TinyOS είναι τα εξής:

Αρχιτεκτονική βασισμένη στην έννοια των component (Component-based architecture):

Το TinyOS παρέχει ένα σύνολο από επαναχρησιμοποιήσιμα components και interfaces για αφαιρέσεις όπως η αποστολή πακέτων (packet transmission), η δρομολόγηση (routing), η ανίχνευση (sensing), η ενεργοποίηση (actuation) και η αποθήκευση (storage). Τα προγράμματα αποτελούνται από components που συνδέονται μεταξύ τους με την χρήση ενός wiring specification.



Σχ. 4: Σχηματική απόδοση της αρχιτεκτονικής του TinyOS.

Tasks και ταυτόχρονος προγραμματισμός βασισμένος στα events (event-based):

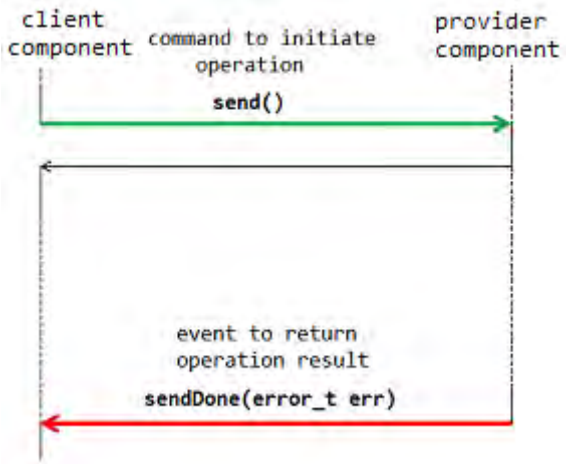
Στο TinyOS υπάρχουν δύο μηχανισμοί ταυτόχρονου προγραμματισμού: οι εργασίες (tasks) και τα γεγονότα (events).

Τα **tasks** είναι παρόμοια με μια Ασύγχρονη Κλήση Διαδικασίας (Deferred Procedure Call-DPC). Ένα component μπορεί να κάνει post ένα task και το λειτουργικό θα το δρομολογήσει μέσω του Task scheduler ώστε να τρέξει αργότερα. Τα tasks εκτελούνται με σειρά FIFO και τρέχουν μέχρι τέλους **(run to completion)** ενώ δεν διακόπτουν το ένα το άλλο **(non-preemptive)**. Επίσης, ένα task μπορεί να κάνει post ένα άλλο task ή τον ίδιο του τον εαυτό, όμως στην ουρά δεν γίνεται να υπάρχει το ίδιο task παραπάνω από μία φορά. Τα components μπορούν να χρησιμοποιήσουν αυτό το μηχανισμό όταν οι απαιτήσεις του χρόνου εκτέλεσης δεν είναι αυστηρές ενώ για να εξασφαλιστεί μικρή καθυστέρηση στην εκτέλεσή των tasks πρέπει να παραμένουν μικρά σε μέγεθος και χρονοβόρες διαδικασίες να χωρίζονται σε περισσότερα του ενός task.

Τα **events** τρέχουν επίσης μέχρι τέλους **(run to completion)**, αλλά μπορούν να διακόψουν την εκτέλεση ενός task ή ενός άλλου event **(preemptive)**. Τα events δηλώνουν είτε την ολοκλήρωση μιας λειτουργίας δύο φάσεων (αναλυτικότερα στην συνέχεια) είτε ένα γεγονός από το περιβάλλον όπως η λήψη ενός μηνύματος ή η λήξη ενός timer.

Λειτουργίες δύο φάσεων (split-phase operations):

Ορισμένες λειτουργίες στα Ασύρματα Δίκτυα Αισθητήρων μπορεί να έχουν μεγάλο χρόνο απόκρισης όπως π.χ. η αναμονή για ανάγνωση τιμής από έναν αισθητήρα ή για την ολοκλήρωση της αποστολής ενός πακέτου πάνω από το δίκτυο. Τα παραδοσιακά Λειτουργικά Συστήματα έχουν λύσει το συγκεκριμένο πρόβλημα με τη χρήση λειτουργιών κλειδώματος (blocking operations) και νημάτων (threads). Επειδή στο TinyOS, όπως αναφέρθηκε, τα tasks εκτελούνται χωρίς να αλληλοδιακόπτονται και δεν υπάρχουν λειτουργίες κλειδώματος, χρησιμοποιούνται λειτουργίες δύο φάσεων **(split-phase operations)**. Σε αυτές, η ενέργεια χωρίζεται σε ένα command το οποίο την εκκινεί και σε ένα event που υποδηλώνει την ολοκλήρωσή της. Χαρακτηριστικό παράδειγμα τέτοιας ενέργειας αποτελεί η αποστολή ενός πακέτου στο δίκτυο: ένα component μπορεί να καλέσει το command send για να ξεκινήσει τη διαδικασία αποστολής του πακέτου και το communication component σηματοδοτεί το sendDone event όταν η αποστολή έχει ολοκληρωθεί (Σχ. 5). Ωστόσο, υπάρχουν και συμβατικές ενέργειες που δεν είναι δύο φάσεων, όπως το άναμμα ενός LED, και δεν έχουν events που να υποδηλώνουν την ολοκλήρωσή τους.



Σχ. 5: Χρονοδιάγραμμα εκτέλεσης της ενέργειας αποστολής ενός πακέτου στο δίκτυο, η οποία αποτελεί χαρακτηριστική ενέργεια δύο φάσεων.

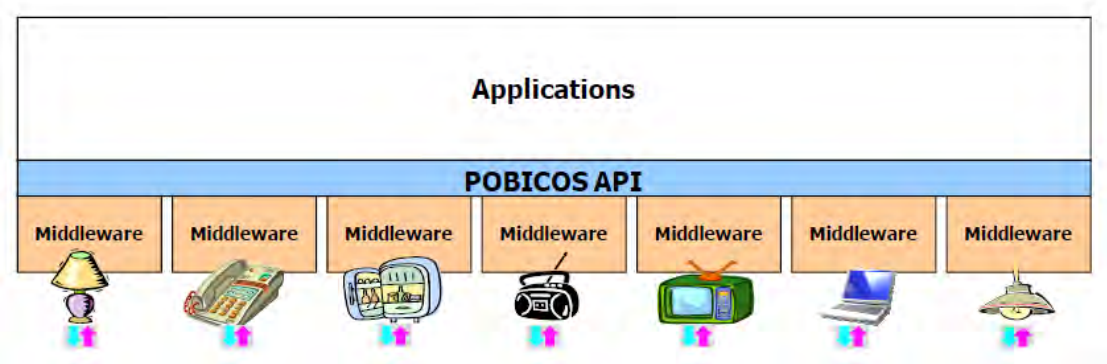
Αυτό το απλό μοντέλο ταυτόχρονου προγραμματισμού είναι ικανοποιητικό για εκτέλεση εφαρμογών TinyOS που οδηγούνται από γεγονότα τα οποία αντιπροσωπεύουν I/O hardware interrupts και επιτρέπει υψηλό ταυτοχρονισμό με χαμηλή επιβάρυνση λόγω της χρήσης μίας και μοναδικής στοίβας **(single stack)** σε αντίθεση με τα μοντέλα που βασίζονται στα νήματα και στα οποία οι στοίβες των νημάτων καταναλώνουν πολύτιμη μνήμη.

Κεφάλαιο 3 POBICOS

3.1 Στόχος-Βασικά χαρακτηριστικά

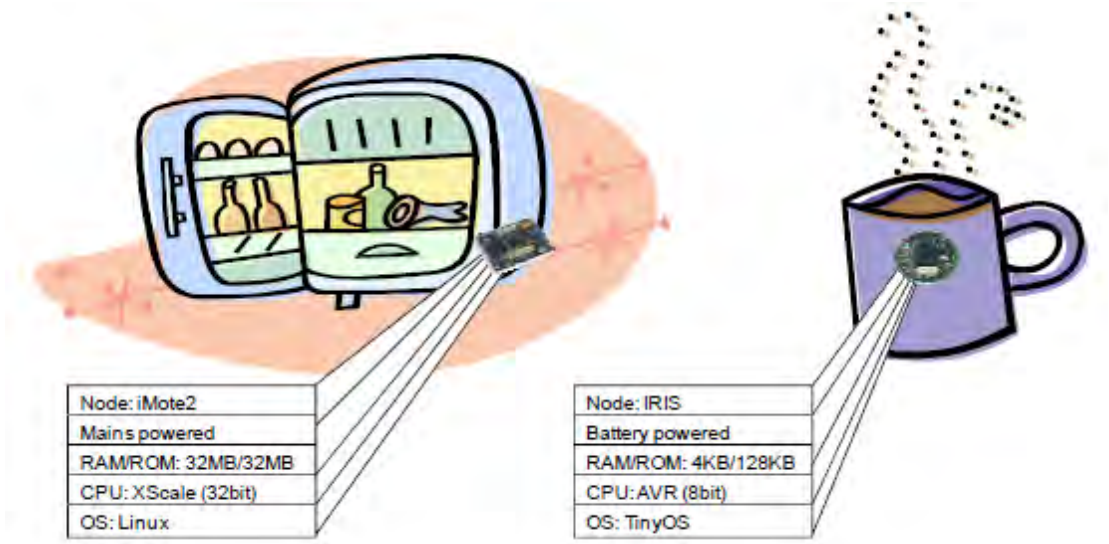
Το POBICOS **(Platform for Opportunistic Behavior in incompletely specified, heterogeneous Object Communities)** είναι ένα ενδιάμεσο λογισμικό **(middleware)** το οποίο στοχεύει σε υπολογιστικά περιβάλλοντα που αποτελούνται από συλλογές αντικειμένων τα οποία είναι εξοπλισμένα με ενσωματωμένους κόμβους ικανούς για συλλογή πληροφοριών από το περιβάλλον (sensing), υπολογισμό (computation), ασύρματη επικοινωνία (wireless communications) και έλεγχο εξαρτημάτων του αντικειμένου (actuating). Τα αντικείμενα αυτά συνδεόμενα ασύρματα μπορούν να σχηματίσουν μια **«κοινότητα αντικειμένων» (object community)** και να αρχίσουν να συνεργάζονται για να επιτύχουν ένα κοινό στόχο.

Στο POBICOS αυτή η συνεργασία των αντικειμένων οδηγείται από εφαρμογές οι οποίες, γενικά, δεν συνδέονται με κάποιο συγκεκριμένο αντικείμενο. Το POBICOS μετατρέπει μια τέτοια κοινότητα αντικειμένων σε ένα ανοιχτό, κατανεμημένο και διάχυτο υπολογιστικό περιβάλλον μέσω ενός στρώματος ενδιάμεσου λογισμικού (middleware layer) με ένα καλά καθορισμένο API. Το API αυτό εκθέτει όλους τους αισθητήρες, ενεργοποιητές και υπολογιστικούς πόρους της κοινότητας των αντικειμένων στο επίπεδο της εφαρμογής (application layer). Ένα τέτοιο περιβάλλον είναι ικανό για ανάπτυξη εφαρμογών ανεξάρτητα από τα συγκεκριμένα αντικείμενα που το απαρτίζουν.



Σχ. 6: Μετατροπή μιας κοινότητας αντικειμένων σε ένα υπολογιστικό περιβάλλον.

Ένα εμφανές χαρακτηριστικό αυτών των περιβαλλόντων είναι η ετερογένεια **(heterogeneity)** και προέρχεται πρώτον, από τους διαφορετικούς τύπους αισθητήρων / ενεργοποιητών που μπορεί να φιλοξενήσει κάθε αντικείμενο ανάλογα με την λειτουργικότητά του και δεύτερον, από τους υπολογιστικούς πόρους (ενέργεια, μνήμη, επεξεργαστική ισχύς) που είναι διαθέσιμοι από το αντικείμενο που φιλοξενεί τον κόμβο.



Σχ. 7: Ετερογένεια περιβάλλοντος ανάπτυξης εφαρμογών.

Επίσης, τα αντικείμενα που συνθέτουν μια τέτοια κοινότητα δεν είναι γνωστά εξαρχής με αποτέλεσμα τα περιβάλλοντα αυτά να είναι ελλιπώς καθορισμένα **(incompletely specified)** κατά τον σχεδιασμό μιας εφαρμογής και ο προγραμματιστής να μην γνωρίζει τους ακριβείς πόρους που θα έχει στη διάθεσή της η εφαρμογή κατά τον χρόνο εκτέλεσης.

Μια εφαρμογή που μπορεί να παρέχει με επιτυχία τη λειτουργικότητά της (ίσως με διαφορετικό βαθμό επιτυχίας) «πάνω» από διαφορετικές κοινότητες αντικειμένων λέμε ότι παρουσιάζει καιροσκοπική συμπεριφορά **(opportunistic behavior)**. Μια τέτοια εφαρμογή θα πρέπει να εκμεταλλευτεί ό,τι «ευκαιρίες»- όσον αφορά αισθητήρες, ενεργοποιητές και υπολογιστικούς πόρους- τυχαίνει να είναι διαθέσιμες στην τοπική κοινότητα.

Συνοψίζοντας, στόχος του POBICOS είναι η δημιουργία μιας πλατφόρμας (εργαλεία και ενδιάμεσο λογισμικό) που θα επιτρέπει την ανάπτυξη και εκτέλεση εφαρμογών που επιδεικνύουν ευκαιριακή συμπεριφορά σε ετερογενής, ελλιπείς ορισμένες κοινότητες αντικειμένων.

3.2 POBICOS API

Τα ιδιαίτερα χαρακτηριστικά του περιβάλλοντος στο οποίο εστιάζουν οι εφαρμογές POBICOS **(application domain)** κατηγοριοποιούνται στη μορφή ενός μοντέλου το οποίο ονομάζεται μοντέλο-τομέα **(domain-model)**. Ο ρόλος του domain-model είναι να συμπεριλάβει την ποικιλία των αντικειμένων σε ένα συγκεκριμένο τομέα (π.χ. ο αυτοματισμός σπιτιών) καθώς και να τα κατηγοριοποιήσει μαζί με τις λειτουργίες που πιθανώς να υποστηρίζονται από το σύνολο των αισθητήρων και ενεργοποιητών που μπορεί να είναι ενσωματωμένοι σε αυτά τα αντικείμενα.

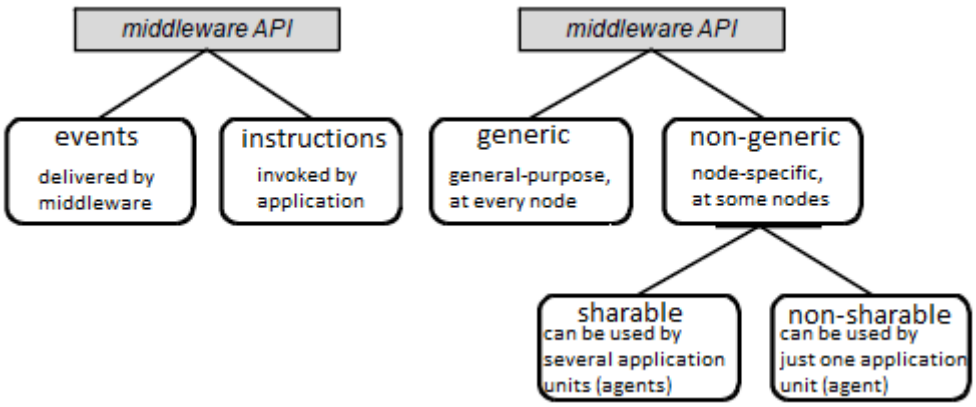
Έτσι, το API του POBICOS χωρίζεται σε δύο μέρη: το ένα αποτελεί τον πυρήνα της τεχνολογίας του POBICOS και είναι ανεξάρτητο του domain-model, για αυτό ονομάζεται **domain-neutral** και το άλλο συμπληρώνει το domain-neutral με στοιχεία που προέρχονται από το συγκεκριμένο domain-model και για αυτό ονομάζεται **domain-specific**.

Τα **προγραμματιστικά στοιχεία** (primitives) του POBICOS μπορεί να είναι είτε γεγονότα (events) είτε εντολές (instructions). Μια εφαρμογή χρησιμοποιεί **εντολές POBICOS** για να καλέσει υπηρεσίες του middleware. Αντίστροφα, το middleware προκαλεί **γεγονότα POBICOS** για να ειδοποιήσει την εφαρμογή για κάποιο συμβάν που πραγματοποιήθηκε ώστε αυτή να το χειριστεί όπως απαιτείται.

Τα **domain-neutral προγραμματιστικά στοιχεία** είναι αυτά που αφορούν κυρίως την κατανεμημένη δομή και την επικοινωνία μέσα σε μια εφαρμογή. Το middleware του POBICOS υποστηρίζει όλο το σύνολο αυτών των στοιχείων σε όλους τους κόμβους και για αυτό ονομάζονται και **generic**.

Αντιθέτως, τα **domain-specific προγραμματιστικά στοιχεία** αντιπροσωπεύουν αντικείμενα, αισθητήρες και ενεργοποιητές και το middleware σε κάθε κόμβο υποστηρίζει μόνο ένα υποσύνολο τους. Για το λόγο αυτό ονομάζονται και **non-generic**.

Επιπλέον, τα non-generic προγραμματιστικά στοιχεία χωρίζονται σε **κοινόχρηστα** (sharable), τα οποία μπορούν να χρησιμοποιηθούν από πολλαπλές οντότητες/συνιστώσες της εφαρμογής (που ονομάζονται πράκτορες και αναλύονται στην παρακάτω ενότητα) που τρέχουν στον ίδιο κόμβο, και σε **μη-κοινόχρηστα** (non-sharable) που μπορούν να χρησιμοποιηθούν από το πολύ μία τέτοια οντότητα σε κάθε κόμβο.

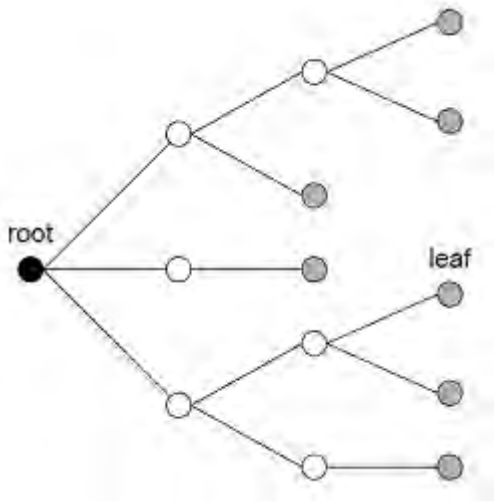


Σχ. 8: Διάρθρωση του POBICOS middleware API.

3.3 Λογική δομή των εφαρμογών POBICOS και μικρό-πράκτορες

Μια εφαρμογή POBICOS αφού αναπτυχθεί, τοποθετείται σε μια φυσική συσκευή η οποία ονομάζεται **application pill**. Εσωτερικά, κάθε εφαρμογή αποτελείται από ένα σύνολο ελαφριών, συνεργαζόμενων οντοτήτων (μονάδων κώδικα) που ονομάζονται **μικρό-πράκτορες** (micro-agents) ή απλά **πράκτορες** (agents) και οργανώνονται σε μια ιεραρχική, δενδρική δομή ανάλογα με τις σχέσεις πατέρα-παιδιού.

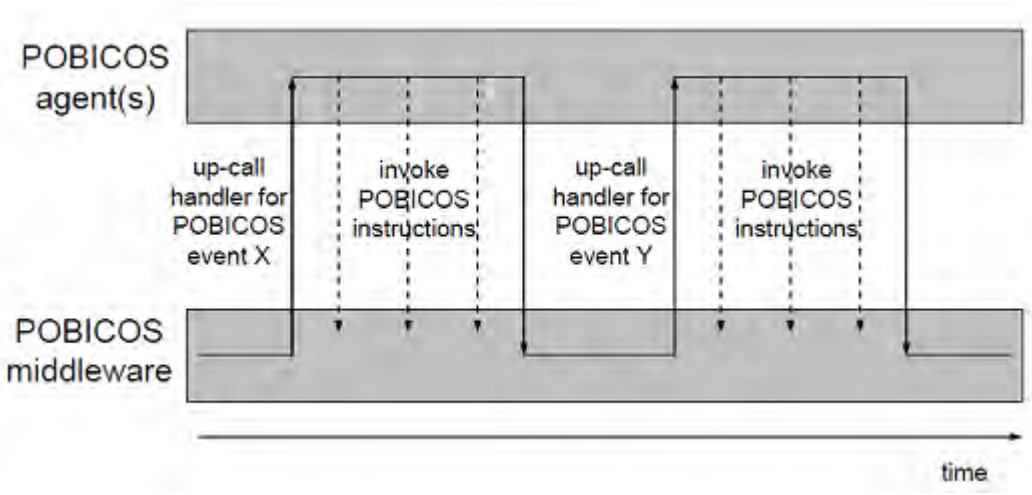
Ο **root agent** δημιουργείται αυτόματα από το ενδιάμεσο λογισμικό POBICOS για να ξεκινήσει η εκτέλεση της εφαρμογής και συνήθως βρίσκεται στο application pill (αν και μπορεί να μεταναστεύσει σε κάποιον άλλο κόμβο του δικτύου). Στην συνέχεια, αυτός μπορεί να δημιουργήσει ένα ή περισσότερα παιδιά, τα οποία με τη σειρά τους μπορούν να δημιουργήσουν δικά τους παιδιά. Οι πράκτορες-φύλλα του δέντρου αλληλεπιδρούν με το φυσικό περιβάλλον μαζεύοντας πληροφορίες μέσω των αισθητήρων ή εκτελώντας ενέργειες μέσω των εξαρτημάτων του αντικειμένου στο οποίο είναι ενσωματωμένος ο κόμβος στον οποίο βρίσκονται (ενεργοποιητές).



Σχ. 9: Ενδεικτική δομή πρακτόρων μιας εφαρμογής POBICOS.

Η συνεργασία των πρακτόρων πραγματοποιείται με την ανταλλαγή μηνυμάτων, η οποία επιτρέπεται μόνο ανάμεσα σε πράκτορες που έχουν σχέση πατέρα-παιδιού: τα μηνύματα του πατέρα προς τα παιδιά του ονομάζονται **commands** και τα μηνύματα των παιδιών προς τον πατέρα **reports**.

Κάθε μικρό-πράκτορας αποτελείται από ένα σύνολο **χειριστών γεγονότων** (event-handlers) για γεγονότα POBICOS και κάθε χειριστής μπορεί να καλεί μία ή περισσότερες εντολές POBICOS. Η εκτέλεση των πρακτόρων είναι οδηγούμενη από τα γεγονότα **(event-driven)**, δηλαδή οι πράκτορες δεν έχουν το δικό τους νήμα εκτέλεσης αλλά οι χειριστές του πράκτορα καλούνται από το ενδιάμεσο λογισμικό όποτε το αντίστοιχο γεγονός συμβαίνει, όπως φαίνεται και στο παρακάτω σχήμα.

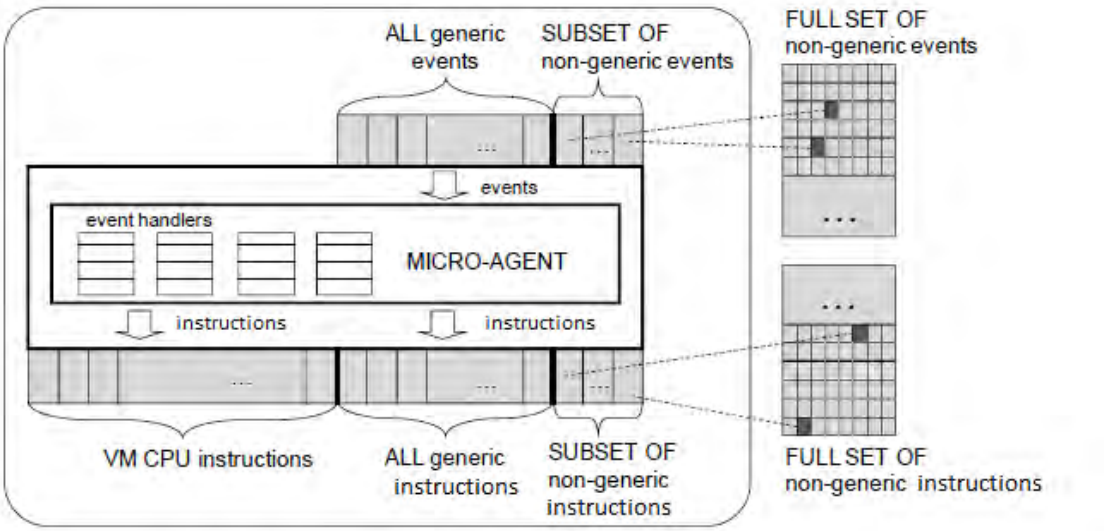


Σχ. 10: Εκτέλεση των event-handlers των μικρό-πρακτόρων με την πάροδο του χρόνου.

Οι μικρό-πράκτορες μπορεί να είναι δύο ειδών: γενικού σκοπού (generic) ή ειδικού σκοπού (non-generic).

Οι **generic πράκτορες** χειρίζονται μόνο domain-neutral (generic) γεγονότα POBICOS και καλούν μόνο domain-neutral (generic) εντολές POBICOS. Για το λόγο αυτό μπορούν να τρέξουν σε κάθε POBICOS κόμβο.

Οι **non-generic πράκτορες** χειρίζονται τουλάχιστον ένα domain-specific (non-generic) γεγονός POBICOS και/ή καλούν τουλάχιστον μία domain-specific (non-generic) εντολή POBICOS. Ένας non-generic πράκτορας μπορεί να τρέξει μόνο σε κόμβους POBICOS που υποστηρίζουν τα αντίστοιχα non-generic στοιχεία (δηλαδή διαθέτουν τους κατάλληλους αισθητήρες και/ή ενεργοποιητές).

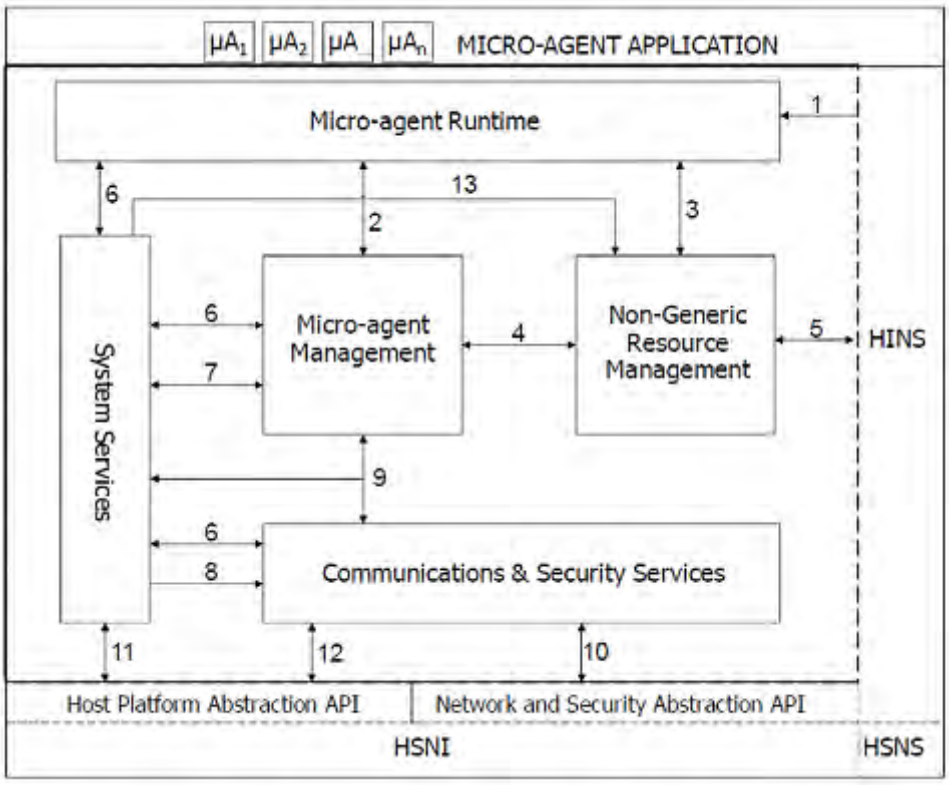


Σχ. 11: Αλληλεπίδραση ανάμεσα στο ενδιάμεσο λογισμικό POBICOS και έναν πράκτορα. Η διεπαφή μεταξύ ενός πράκτορα και του ενδιάμεσου λογισμικού χωρίζεται στο σύνολο εντολών μηχανής (instruction set) της VM CPU και στο middleware API.

3.4 Δομή λογισμικού στους κόμβους

Η δομή λογισμικού στους κόμβους δίνει έμφαση στη σπονδυλωτή κατασκευή και στην φορητότητα. Αυτό επιτυγχάνεται κυρίως με τον εντοπισμό των υποδομών λογισμικού και υλικού (software and hardware infrastructure) του κόμβου και τον σαφή διαχωρισμό των components του ενδιάμεσου λογισμικού που εξαρτώνται άμεσα από τις υποδομές του κόμβου από αυτά που είναι ανεξάρτητα. Όπως είναι φυσικό, ένας σχεδιαστικός στόχος είναι η μεγιστοποίηση του μέρους του ενδιάμεσου λογισμικού που είναι ανεξάρτητο ώστε να ελαχιστοποιηθούν οι αλλαγές που απαιτούνται για την μεταφορά του σε έναν διαφορετικού είδους κόμβο.

Η **δομή** του πυρήνα του ενδιάμεσου λογισμικού POBICOS, ο οποίος είναι κοινός για όλους τους κόμβους, παρουσιάζεται στο παρακάτω σχήμα:



Σχ. 12: Διάγραμμα δομής του πυρήνα του ενδιάμεσου λογισμικού POBICOS.

Ο πυρήνας δηλαδή αποτελείται από:

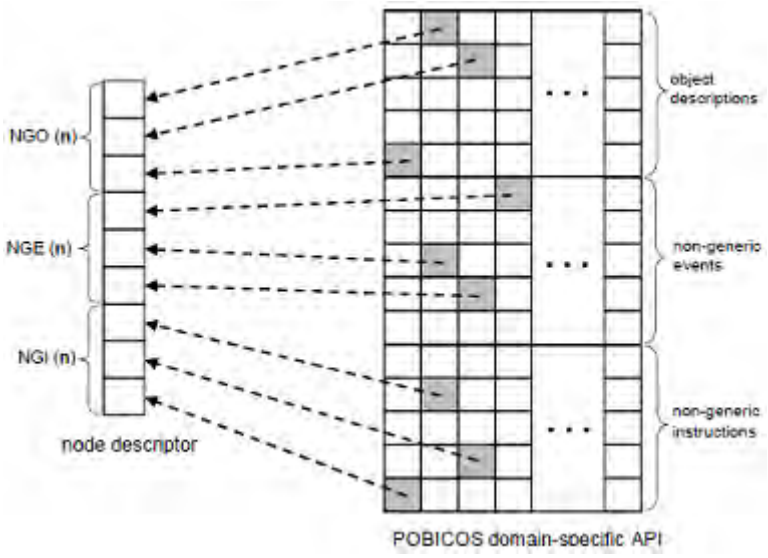
1. Μονάδα για την εκτέλεση των Μικρό-πρακτόρων **(Micro-agent Runtime)**, η οποία παρέχει ένα πολύ-πρακτορικό περιβάλλον εκτέλεσης που βασίζεται στην αρχιτεκτονική μιας Virtual Machine CPU για την διερμηνεία των VM CPU instructions και των POBICOS instructions και εκτελεί όλα τα βήματα για την εκτέλεση ενός event handler ως απόκριση σε ένα POBICOS event.

2. Μονάδα Διαχείρισης Μικρό-πρακτόρων **(Micro-agent Management)**, υπεύθυνη για την δημιουργία και την τοποθέτηση των πρακτόρων σε κόμβους, την μεταφορά του κώδικά τους, την μετανάστευσή τους, καθώς και την επικοινωνία μεταξύ τους.
3. Μονάδα Διαχείρισης Non-generic Πόρων **(Non-generic Resource Management).**
4. Μονάδα Υπηρεσιών Συστήματος **(System Services)**, η οποία συγκεντρώνει διάφορες υπηρεσίες επιπέδου ενδιάμεσου λογισμικού (όπως timers και διαχείριση μνήμης) και είναι υπεύθυνη για την παρακολούθηση και τον έλεγχο ολόκληρης της κοινότητας των αντικειμένων POBICOS.
5. Μονάδα Υπηρεσιών Επικοινωνίας και Ασφάλειας **(Communications and Security Services)**, υπεύθυνη για την παροχή υπηρεσιών επικοινωνίας για τα πρωτόκολλα του POBICOS και για την υλοποίηση της Αρχιτεκτονικής Ασφάλειας του POBICOS.

Η **υποδομή του κόμβου** για το ενδιάμεσο λογισμικό POBICOS αποτελείται από τα ακόλουθα στοιχεία:

i. την πλατφόρμα υποδοχής, συμπεριλαμβανομένου του λειτουργικού συστήματος (π.χ. Imote2 & TinyOS),
ii. την τεχνολογία δικτύωσης (π.χ. ZigBee),
iii. τους πόρους αίσθησης και ελέγχου των εξαρτημάτων των αντικειμένων που πρέπει να είναι ορατοί στις εφαρμογές καθώς και
iv. τις ιδιότητες του αντικειμένου στο οποίο είναι ενσωματωμένος ο κόμβος.

Για την περιγραφή των (iii) και (iv) σε ορολογία του domain-specific API, χρησιμοποιείται μια δομή δεδομένων που ονομάζεται **node descriptor**. Στην ουσία αυτή η δομή περιγράφει το αντικείμενο στο οποίο είναι ενσωματωμένος ο κόμβος και τους αισθητήρες /ενεργοποιητές του κόμβου σε μια υψηλού-επιπέδου «γλώσσα τομέα» ("domain language") που αποτελείται από domain-specific στοιχεία όπως φαίνεται και στο παρακάτω σχήμα.



Σχ. 13: Δομή του node descriptor.

Κλείνοντας το κεφάλαιο, η εισαγωγή που επιχειρήθηκε στο ενδιάμεσο λογισμικό POBICOS τόσο σε επίπεδο API, όσο και σε επίπεδο αρχιτεκτονικής είχε σκοπό να γίνουν γνωστά τα στοιχεία που είναι απαραίτητα ώστε να γίνει κατανοητό το πρωτόκολλο που αναπτύχθηκε για την αποδοτική δημιουργία ομάδας πρακτόρων και το οποίο παρουσιάζεται στο επόμενο κεφάλαιο.

Κεφάλαιο 4 Δημιουργία ομάδας μικρό-πρακτόρων στο POBICOS

4.1 Εισαγωγή στη δημιουργία πρακτόρων στο POBICOS

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο (Ενότητα 3.3), προκειμένου μια εφαρμογή να ξεκινήσει την εκτέλεσή της, πρέπει το ενδιάμεσο λογισμικό POBICOS να δημιουργήσει επιτυχώς τον root agent. Η δημιουργία των υπόλοιπων πρακτόρων πραγματοποιείται από τη ίδια την εφαρμογή υποβάλλοντας αιτήσεις στο ενδιάμεσο λογισμικό POBICOS μέσω generic instructions. Η δημιουργία των πρακτόρων είναι ασύγχρονη καθώς οι αιτήσεις μπαίνουν σε μια FIFO ουρά και επεξεργάζονται από ένα task που καλείται περιοδικά μέσω ενός timer.

Αιτήσεις δημιουργίας πρακτόρων
Υπάρχουν δύο επιλογές για το πλήθος των στιγμιότυπων του πράκτορα που θα δημιουργηθούν: δημιουργία ενός πράκτορα **(single mode)** και δημιουργία ομάδας πρακτόρων **(multiple mode)**.

Οι αιτήσεις δημιουργίας generic πρακτόρων είναι πάντα σε single mode, δηλαδή η εφαρμογή υποβάλλει μια ξεχωριστή εντολή για κάθε στιγμιότυπο που θέλει να δημιουργήσει. Αντιθέτως, οι αιτήσεις δημιουργίας non-generic πρακτόρων μπορούν να είναι τόσο σε single όσο και σε multiple mode και έτσι μπορούν να δημιουργηθούν πολλαπλά στιγμιότυπα του πράκτορα **(ομάδα μικρό-πρακτόρων)** σε διαφορετικά αντικείμενα με μια και μόνο εντολή. Σε αυτή την περίπτωση μπορεί να παρέχεται από τον προγραμματιστή της εφαρμογής ένα προσδιοριστικό αντικειμένου **(object qualifier)** ως επιπλέον όρισμα στην εντολή για τον προσδιορισμό του είδους των αντικειμένων που μπορούν να φιλοξενήσουν τον συγκεκριμένο τύπο πράκτορα.

Δύο επιπλέον ορίσματα που περιλαμβάνονται στις αιτήσεις δημιουργίας πρακτόρων είναι οι **ρυθμίσεις διαμόρφωσης** (configuration settings), που δίνουν τη δυνατότητα στο χρήστη να προσαρμόσει τη συμπεριφορά των πρακτόρων της εφαρμογής κατά τον χρόνο εκτέλεσης, και η **προτεραιότητα της εφαρμογής** (application priority) η οποία χρησιμοποιείται κυρίως για την απομάκρυνση των πρακτόρων μιας μη-κρίσιμης εφαρμογής που κάνουν χρήση μη-κοινόχρηστων στοιχείων (non-sharable primitives).

Τοποθέτηση πρακτόρων
Για κάθε πράκτορα που πρόκειται να δημιουργηθεί, το ενδιάμεσο λογισμικό POBICOS επιλέγει το αντικείμενο που θα αποτελέσει τον **ξενιστή** (host) και μεταφέρει τον κώδικα του πράκτορα σε αυτό.

Η τοποθέτηση ενός generic πράκτορα είναι εξ ολοκλήρου στη διακριτική ευχέρεια του ενδιάμεσου λογισμικού, δηλαδή, ένας generic πράκτορας μπορεί να τοποθετηθεί σε οποιοδήποτε αντικείμενο έχει αρκετό χώρο για την αποθήκευση των δεδομένων και του κώδικά του, με τρόπο εντελώς διαφανή στον προγραμματιστή της εφαρμογής.

Η τοποθέτηση ενός non-generic πράκτορα περιορίζεται από τον object qualifier και τα non-generic στοιχεία (εντολές και γεγονότα) που χρησιμοποιούνται από τον πράκτορα. Μόνο ένα αντικείμενο του οποίου ο node descriptor ταιριάζει με τον object qualifier και υποστηρίζει τα απαιτούμενα non-generic προγραμματιστικά στοιχεία θεωρείται **επιλέξιμο** (eligible) και μπορεί να αποτελέσει ξενιστή, ωστόσο, εάν υπάρχουν πολλαπλά τέτοια αντικείμενα και είναι να δημιουργηθεί μόνο ένα στιγμιότυπο πράκτορα (η αίτηση είναι σε single mode), τότε είναι ευθύνη του ενδιάμεσου λογισμικού να διαλέξει το αντικείμενο.

Αποτυχία δημιουργίας πρακτόρων
Η δημιουργία ενός πράκτορα είναι δυνατόν να αποτύχει λόγω έλλειψης ελεύθερου χώρου (μνήμης), ενώ συγκεκριμένα στους non-generic πράκτορες μπορεί να μην βρεθεί αντικείμενο που να ταιριάζει με τον object qualifier (δηλαδή, να μην υποστηρίζει τα απαιτούμενα non-generic στοιχεία) ή απλά μπορεί τα μη-κοινόχρηστα στοιχεία που χρειάζεται ο πράκτορας να είναι δεσμευμένα από κάποιον άλλο που βρίσκεται ήδη στο αντικείμενο και ανήκει σε εφαρμογή μεγαλύτερης προτεραιότητας.

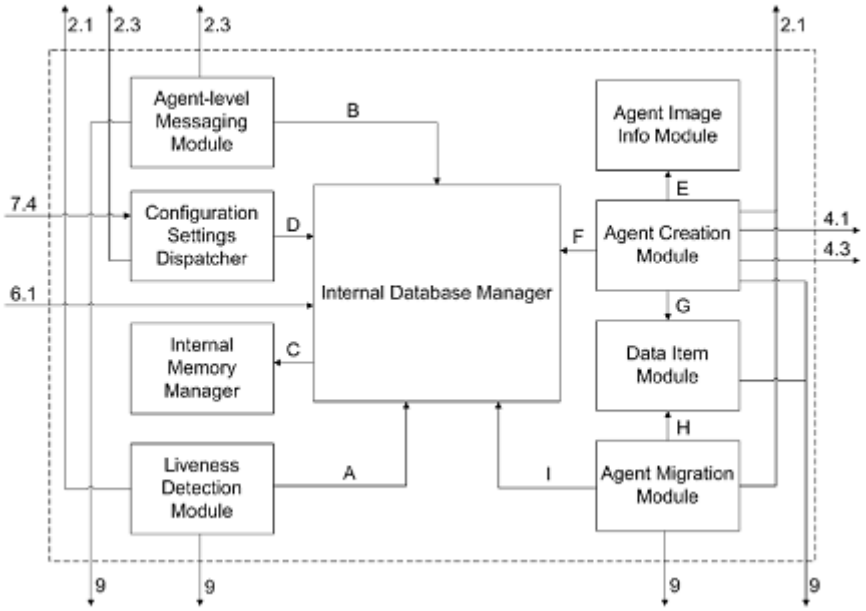
Agent Creation Module και αλληλεπιδράσεις
Η εξυπηρέτηση των αιτήσεων δημιουργίας πρακτόρων, η υλοποίηση του πρωτοκόλλου δημιουργίας πρακτόρων και η δημιουργία στιγμιότυπων των πρακτόρων τοπικά ή απομακρυσμένα υλοποιούνται στην Μονάδα Δημιουργίας Πρακτόρων **(Agent Creation Module)** που αποτελεί μέρος της Μονάδας Διαχείρισης Μικρό-πρακτόρων **(Micro-agent Management Component).**

Το Agent Creation Module στο πλαίσιο του Micro-agent Management Component , όπως φαίνεται στο Σχ.14 , αλληλεπιδρά με:

i. το Agent Image Info Module, για την ανάκτηση των non-generic απαιτήσεων για ένα συγκεκριμένο τύπο πράκτορα.
ii. τον Internal Database Manager, για την εισαγωγή/ανάκτηση/διαγραφή περιγραφέων για τις αιτήσεις δημιουργίας πρακτόρων (creation request descriptors), την εισαγωγή του περιγραφέα ενός πράκτορα (agent descriptor) που δημιουργήθηκε τοπικά στην αντίστοιχη τοπική λίστα περιγραφέων και την εισαγωγή του περιγραφέα παιδιού (child descriptor) στην λίστα των παιδιών ενός τοπικού parent agent.
iii. το Data Item Module, για την ανάκτηση του κώδικα (ή αλλιώς του εκτελέσιμου) ενός συγκεκριμένου τύπου πράκτορα.

Επιπλέον, αλληλεπιδρά και με άλλα components του πυρήνα του POBICOS middleware (τον οποίο παρουσιάσαμε στην Ενότητα 3.4):

i. το Micro-agent Runtime Component, για την δημιουργία/κατάργηση στιγμιότυπων πρακτόρων, την διακοπή/επανεκκίνηση της εκτέλεσης πρακτόρων καθώς και την σειριοποίηση/αποσειριοποίηση της τρέχουσας κατάστασης πρακτόρων.
ii. το Non-generic Resource Management Component, για την σύγκριση του object qualifier που περιέχεται σε μια αίτηση με τον node descriptor και την καταχώριση/απομάκρυνση non-generic πρακτόρων.
iii. το System Services Component, για την αποστολή αξιόπιστων (reliable) και αναξιόπιστων (unreliable-best effort) μηνυμάτων στο δίκτυο POBICOS καθώς και για την ανίχνευση της προσβασιμότητας ενός απομακρυσμένου κόμβου.
iv. το Communication & Security Services Component, από το οποίο ενημερώνεται για την λήψη αξιόπιστων (reliable) ή αναξιόπιστων (unreliable-best effort) μηνυμάτων από το δίκτυο POBICOS.



Σχ. 14: Διάγραμμα Αρχιτεκτονικής της Μονάδας Διαχείρισης Μικρό-πρακτόρων.

4.2 Υπάρχον πρωτόκολλο δημιουργίας ομάδας πρακτόρων

Στην Διπλωματική Εργασία εστιάζουμε στην δημιουργία non-generic πρακτόρων και πιο συγκεκριμένα, στην περίπτωση που επιθυμούμε την δημιουργία ομάδας μικρό-πρακτόρων (δηλαδή η αίτηση δημιουργίας είναι σε multiple mode).

Το υπάρχον πρωτόκολλο δημιουργίας ομάδας πρακτόρων χωρίζεται σε δύο φάσεις: το Host Candidate Discovery Protocol,το οποίο χρησιμοποιείται για την ανακάλυψη των κόμβων που είναι ικανοί-κατάλληλοι να λειτουργήσουν ως ξενιστές ενός στιγμιότυπου του πράκτορα, και το Agent Creation Protocol στο οποίο δημιουργείται ένα στιγμιότυπο του πράκτορα σε έναν (συγκεκριμένο) υποψήφιο κόμβο.

4.2.1 Host Candidate Discovery Protocol

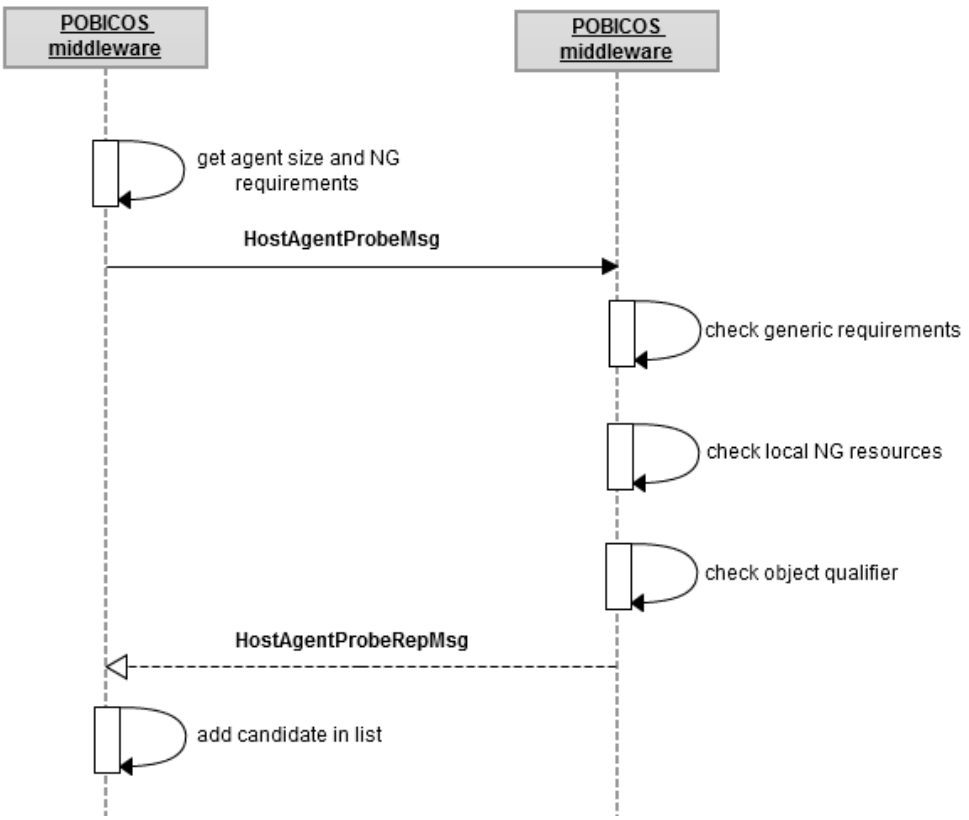
Αρχικά, ο κόμβος που εξυπηρετεί την αίτηση δημιουργίας (εκεί όπου βρίσκεται ο parent agent δηλαδή) λαμβάνει τον κώδικα του πράκτορα από το application pill και εξάγει τις απαιτήσεις που πρέπει να ικανοποιεί ένας κόμβος ώστε να λειτουργήσει ως ξενιστής. Αυτές οι απαιτήσεις περιλαμβάνουν το μέγεθος του πράκτορα, τα non-generic προγραμματιστικά στοιχεία (εντολές και γεγονότα) που χρησιμοποιεί και τον object qualifier. Στην συνέχεια, ελέγχει αν ο ίδιος είναι επιλέξιμος και αν αυτό ισχύει, τοποθετεί τον εαυτό του στη λίστα υποψηφίων.

Έπειτα, το middleware προσπαθεί να βρει άλλους υποψήφιους κόμβους για την φιλοξενία ενός στιγμιότυπου του πράκτορα και για να το πετύχει αυτό κάνει broadcast ένα *HostAgentProbe* μήνυμα στο δίκτυο και περιμένει για *HostAgentProbeRep* μηνύματα για ένα συγκεκριμένο χρονικό διάστημα. Τα *HostAgentProbe* μηνύματα περιέχουν τον τύπο του πράκτορα, τις απαιτήσεις που πρέπει να ικανοποιεί ένας κόμβος για να λειτουργήσει ως ξενιστής του καθώς και την προτεραιότητα της εφαρμογής.

Όταν το middleware λάβει ένα *HostAgentProbe* μήνυμα, ελέγχει αν οι τοπικά διαθέσιμοι υπολογιστικοί πόροι είναι επαρκείς για να φιλοξενήσουν τον κώδικα του πράκτορα και τα στατικά δεδομένα. Ακόμα, ελέγχει αν ικανοποιούνται οι non-generic απαιτήσεις, αν υπάρχει ήδη στον κόμβο κάποιος πράκτορας ίδιου τύπου και αν υπάρχουν τοπικά άλλοι non-generic πράκτορες ίσης ή μεγαλύτερης προτεραιότητας που να χρησιμοποιούν έναν μη-κοινόχρηστο πόρο που χρειάζεται και ο ίδιος. Στη συνέχεια, ένα *HostAgentProbeRep* μήνυμα στέλνεται πίσω στον αποστολέα της αίτησης μεταφέροντας το αντίστοιχο αποτέλεσμα των ελέγχων. Σε αυτή την πρώτη φάση, μια απάντηση λειτουργεί απλά ως υπόδειξη, δηλαδή, ο κόμβος που απαντάει δεν δεσμεύει τοπικούς πόρους.

Ο κόμβος που έστειλε την αίτηση, όταν λάβει ένα *HostAgentProbeRep* μήνυμα προσθέτει τον αποστολέα του σε μια λίστα υποψηφίων. Μετά την πάροδο συγκεκριμένου χρονικού διαστήματος οι υποψήφιοι εξετάζονται ένας προς έναν σε δεύτερη φάση (βλέπε Ενότητα 4.2.2).

Τόσο το *HosAgentProbe* όσο και το *HostAgentProbeRep* στέλνονται ως αναξιόπιστα μηνύματα μέσω της αντίστοιχης υπηρεσίας δικτύωσης.



Σχ. 15: Διάγραμμα ακολουθίας μηνυμάτων στο Host Candidate Discovery Protocol.

4.2.2 Agent Creation Protocol

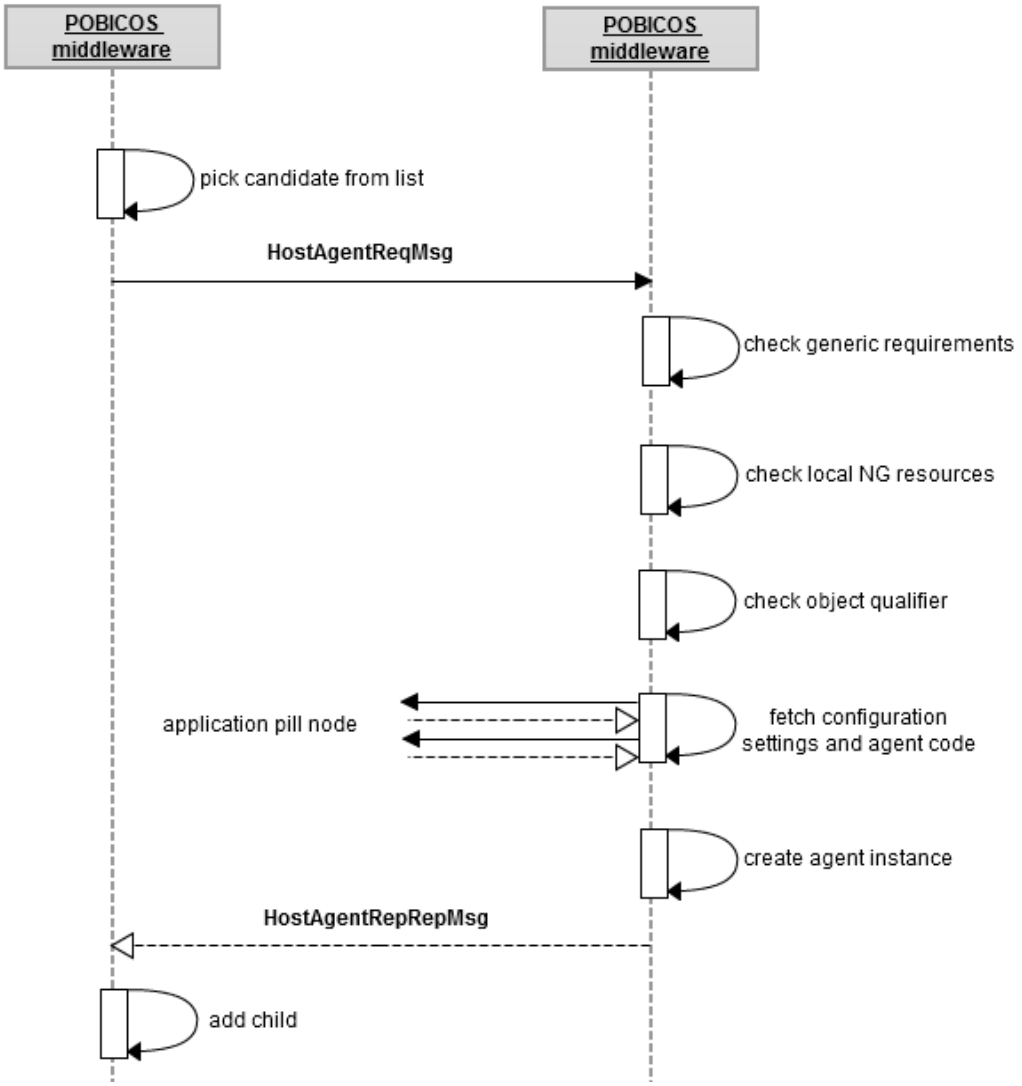
Έχοντας επιλέξει έναν υποψήφιο κόμβο για να φιλοξενήσει ένα στιγμιότυπο του πράκτορα, το middleware στέλνει ένα *HostAgentReq* μήνυμα σε αυτόν και περιμένει για ένα *HostAgentReqRep* μήνυμα.

Όταν το middleware λάβει ένα *HostAgentReq* μήνυμα, εφόσον δεν εξυπηρετεί κάποια άλλη αίτηση, ελέγχει αν ο object qualifier ταιριάζει με τον τοπικό node descriptor. Στη συνέχεια, αν δεν είναι ήδη διαθέσιμες τοπικά οι ρυθμίσεις διαμόρφωσης για τον συγκεκριμένο τύπο πράκτορα, ο κόμβος τις αιτείται από το application pill (συνήθως είναι ο κόμβος όπου βρίσκεται ο root agent) και αφού επιτύχει η μεταφορά τους, αιτείται και τον κώδικα για τον συγκεκριμένο τύπο πράκτορα. Τέλος, ελέγχει αν οι τοπικά διαθέσιμοι generic πόροι επαρκούν για να φιλοξενήσουν τον συγκεκριμένο τύπο πράκτορα, αν ο κόμβος πληροί τις απαιτούμενες non-generic απαιτήσεις και ότι δεν υπάρχει άλλος πράκτορας τοπικά ίσης ή μεγαλύτερης προτεραιότητας που να χρησιμοποιεί μη-κοινόχρηστους πόρους. Αν αυτοί οι έλεγχοι είναι επιτυχείς, ένας νέος πράκτορας του αιτούμενου τύπου δημιουργείται τοπικά (μέσω κλήσεων στο runtime) και ένα *HostAgentReqRep* μήνυμα στέλνεται πίσω στον αποστολέα της αίτησης το οποίο περιέχει το αναγνωριστικό του πράκτορα που μόλις δημιουργήθηκε. Αν οποιοσδήποτε από τους παραπάνω ελέγχους αποτύχει ή αποτύχει η μεταφορά των ρυθμίσεων διαμόρφωσης ή του κώδικα του πράκτορα ή το runtime αποτύχει να δημιουργήσει το στιγμιότυπο του πράκτορα, αντί του αναγνωριστικού του πράκτορα επιστρέφεται η τιμή μηδέν (0).

Όσο ο αποστολέας περιμένει απάντηση από έναν κόμβο, στέλνει περιοδικά *Ping* μηνύματα σε αυτόν για να ελέγξει την λειτουργία, και να βεβαιωθεί ότι έχει νόημα να συνεχίσει να περιμένει. Αν το δίκτυο αναφέρει ότι ήταν αδύνατο να παραδοθεί ένα τέτοιο μήνυμα στον προορισμό του, επιλέγεται ο επόμενος υποψήφιος (αν υπάρχει κάποιος).

Όταν το middleware λάβει ένα θετικό *HostAgentReqRep* μήνυμα, ανανεώνεται η δομή για τα παιδιά που κρατάει ο τοπικός parent agent και ο ίδιος ο parent agent ενημερώνεται για την δημιουργία του πράκτορα-παιδιού του. Στην συνέχεια, ή στην περίπτωση που η απάντηση ήταν αρνητική, αποστέλλεται ένα *HostAgentReq* μήνυμα στον επόμενο υποψήφιο (αν υπάρχει κάποιος). Αν η διάρκεια ζωής της αίτησης δημιουργίας του πράκτορα δεν έχει λήξει, τότε η συγκεκριμένη αίτηση τοποθετείται στο τέλος της ουράς αιτήσεων για να επανεξεταστεί με ένα round-robin τρόπο αργότερα.

Τα *HostAgentReq,* τα *HostAgentReqRep* μηνύματα καθώς και τα Ping μηνύματα στέλνονται αξιόπιστα μέσω της αντίστοιχης υπηρεσίας δικτύωσης.



Σχ. 16: Διάγραμμα ακολουθίας μηνυμάτων στο Agent Creation Protocol.

4.3 Πιο αποδοτική δημιουργία ομάδας μικρό-πρακτόρων

Για να γίνει πιο αποδοτική η δημιουργία μιας ομάδας μικρό-πρακτόρων, οι δύο φάσεις του παραπάνω πρωτοκόλλου συγχωνεύτηκαν σε μία.

4.3.1 Πρωτόκολλο

Αρχικά, ο κόμβος που εξυπηρετεί την αίτηση δημιουργίας (εκεί όπου βρίσκεται ο parent agent δηλαδή) λαμβάνει τον μέγεθος του κώδικα του πράκτορα από το application pill καθώς και τις απαιτήσεις που πρέπει να ικανοποιεί ένας κόμβος ώστε να αποτελέσει ξενιστή. Αυτές οι απαιτήσεις περιλαμβάνουν το μέγεθος του πράκτορα, τα non-generic προγραμματιστικά στοιχεία (εντολές και γεγονότα) που χρησιμοποιεί και τον object qualifier. Στην συνέχεια, ελέγχει αν ο ίδιος πληροί αυτές τις απαιτήσεις και αν αυτό ισχύει, προσπαθεί να λάβει από το application pill τις ρυθμίσεις διαμόρφωσης. Σε περίπτωση που το επιτύχει, προσπαθεί να δημιουργήσει ένα στιγμιότυπο του πράκτορα τοπικά και αν τα καταφέρει προσθέτει τον πράκτορα στην λίστα των παιδιών του parent agent.

Έπειτα, στην προσπάθεια να δημιουργηθούν περισσότερα στιγμιότυπα του πράκτορα σε απομακρυσμένους κόμβους, το middleware στον κόμβο που εξυπηρετείται η αίτηση κάνει broadcast ένα *HostMultiAgentReq* μήνυμα στο δίκτυο. Τα *HostMultiAgentReq* μηνύματα περιέχουν τον τύπο του πράκτορα, τις απαιτήσεις για να λειτουργήσει ένας κόμβος ως ξενιστής του, την προτεραιότητα της εφαρμογής καθώς και ένα μοναδικό αναγνωριστικό (*sequence number*). Ακόμα, περιέχουν την διεύθυνση του κόμβου όπου βρίσκεται ο parent agent , δηλαδή του κόμβου που εξυπηρετεί την αίτηση δημιουργίας (*source address*), και του κόμβου από τον οποίο οι ικανοί κόμβοι να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα θα ζητήσουν τις ρυθμίσεις διαμόρφωσης και τον κώδικα του (*code address*).

Όταν το middleware λάβει ένα *HostMultiAgentReq* μήνυμα, αρχικά ελέγχει αν εξυπηρετείται κάποια άλλη αίτηση και αν δεν ισχύει κάτι τέτοιο ελέγχει αν είναι η πρώτη φορά που την λαμβάνει ή αν είναι διπλότυπο. Αυτός ο έλεγχος γίνεται με χρήση μιας δομής που κρατά το sequence number και τον τύπο του πράκτορα των πρόσφατα ληφθέντων αιτήσεων. Στην περίπτωση που είναι διπλότυπο, η αίτηση απορρίπτεται, ενώ αν δεν είναι, γίνεται έλεγχος εφικτότητας δημιουργίας του πράκτορα στο συγκεκριμένο κόμβο. Αυτό περιλαμβάνει ελέγχους διαθεσιμότητας generic πόρων (agent manager & runtime manager resources, memory feasibility), έλεγχο υποστήριξης των non-generic απαιτήσεων από τον κόμβο, σύγκριση του object qualifier με τον node descriptor και έλεγχο ύπαρξης άλλου πράκτορα του ιδίου τύπου. Εάν ο κόμβος αποδειχτεί ικανός να φιλοξενήσει ένα στιγμιότυπο του πράκτορα, το middleware αιτείται τις ρυθμίσεις διαμόρφωσης και τον κώδικα του από τον κόμβο με code address. Αν και οι δύο αιτήσεις είναι επιτυχείς, γίνονται κλήσεις στο runtime ώστε ο πράκτορας να δημιουργηθεί τοπικά και αν το αποτέλεσμα είναι θετικό αποστέλλεται ένα μήνυμα *HostMultiAgentReqRep* στον κόμβο όπου βρίσκεται ο parent agent (source address) το οποίο περιέχει το αναγνωριστικό του child agent που μόλις δημιουργήθηκε.

Εφόσον ήταν η πρώτη φορά που ο κόμβος έλαβε την αίτηση, κάνει broadcast στο δίκτυο το *HostMultiAgentReq* μήνυμα. Στην περίπτωση που ο κόμβος ήταν επιλέξιμος και υπήρξε επιτυχής μεταφορά των ρυθμίσεων διαμόρφωσης και της εικόνας του πράκτορα (άσχετα αν δημιουργήθηκε ο πράκτορας τοπικά ή όχι), το πεδίο του code address στο *HostMultiAgentReq* μήνυμα τίθεται ίσο με την διεύθυνση του κόμβου. Έτσι, όσοι κόμβοι λάβουν την αίτηση με τη δική του code address θα αιτηθούν από αυτόν τα συγκεκριμένα στοιχεία.

Όταν το middleware του κόμβου όπου βρίσκεται ο parent agent (source address) λάβει ένα *HostMultiAgentReqRep* μήνυμα, προσθέτει το αναγνωριστικό του child agent και την διεύθυνση του κόμβου όπου δημιουργήθηκε, στην δομή που κρατάει ο parent agent για τα παιδιά του.

Τα *HostMultiAgentReq* μηνύματα που γίνονται broadcast στέλνονται ως αναξιόπιστα ενώ τα *HostMultiAgentReqRep* unicast μηνύματα στέλνονται ως αξιόπιστα datagrams μέσω των αντίστοιχων υπηρεσιών δικτύωσης.

4.3.2 Μηνύματα

Η μορφή των μηνυμάτων, σε EBNF, είναι η εξής:

```
HostMultiAgentReqMsg =  HostMultiAgentReqMsgFlag seqno appId prio srcAddr
                        codeAddr pillAddr atype pid agentSize ngLen
                        nonGenReqs oq->size oq->package


HostMultiAgentReqMsgFlag = 0x09

seqno = uint32_t         //child request sequence number
appId = uint32_t         //application id ; the root agent id
prio = uint8_t           //application priority ; for resource conflicts
srcAddr = uint16_t       //address of request's source
codeAddr = uint16_t      //code address
pillAddr = uint16_t      //application pill address
atype = uint32_t         //agent type id
pid = uint32_t           //id of parent agent
agentSize = uint32_t     //agent code size

ngLen = uint8_t    ngLen //length of non-generic requirement blob
nonGenReqs = {byte}      //non-generic requirements blob

oq->size = uint8_t(oq->size)//length of object qualifier blob
oq->package = {byte}     //object qualifier blob
```

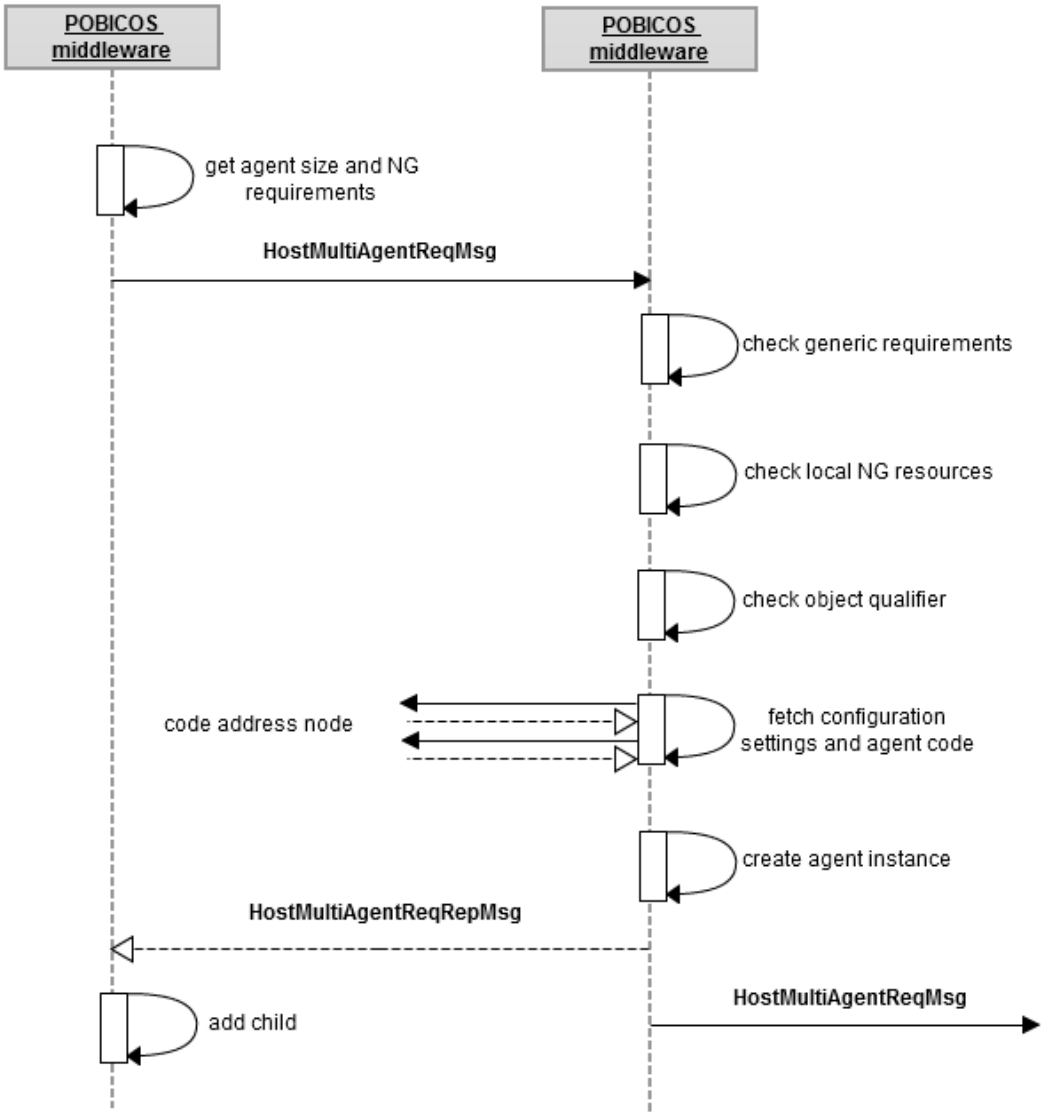
Σχ. 17: Μορφή των HostMultiAgentReq μηνυμάτων σε EBNF.

```
HostMultiAgentReqRepMsg =     HostMultiAgenrReqRepMsgFlag epoch seqno atype
                              cid status


HostMultiAgenrReqRepMsgFlag = 0x0a

epoch = uint32_t         //epoch number
seqno = uint32_t         //request sequence number
atype = uint32_t         //agent type id
cid = uint32_t           //id of the child agent created
status =uint8_t          //status of the request ; success or reason of
                         //failure
```

Σχ. 18: Μορφή των HostMultiAgentReqRep μηνυμάτων σε EBNF.

4.3.3 Διάγραμμα ροής μηνυμάτων

Η πρωτότυπη αλληλεπίδραση για αυτό το πρωτόκολλο έχει ως εξής:

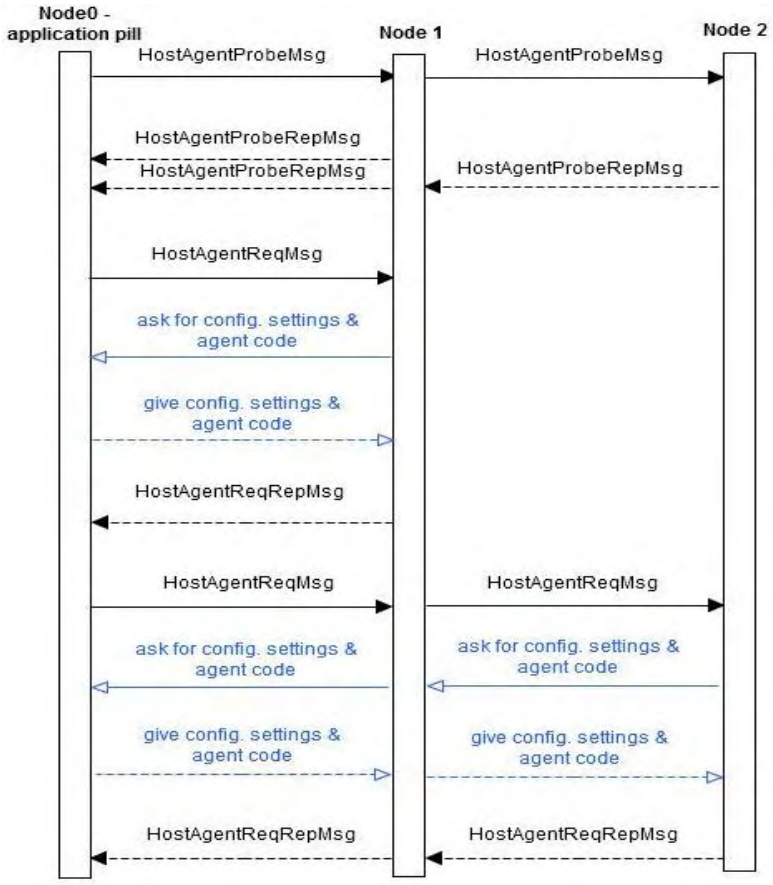


Σχ. 19: Διάγραμμα ακολουθίας μηνυμάτων του υλοποιηθέντος πρωτοκόλλου δημιουργίας ομάδας μικρο-πρακτόρων.

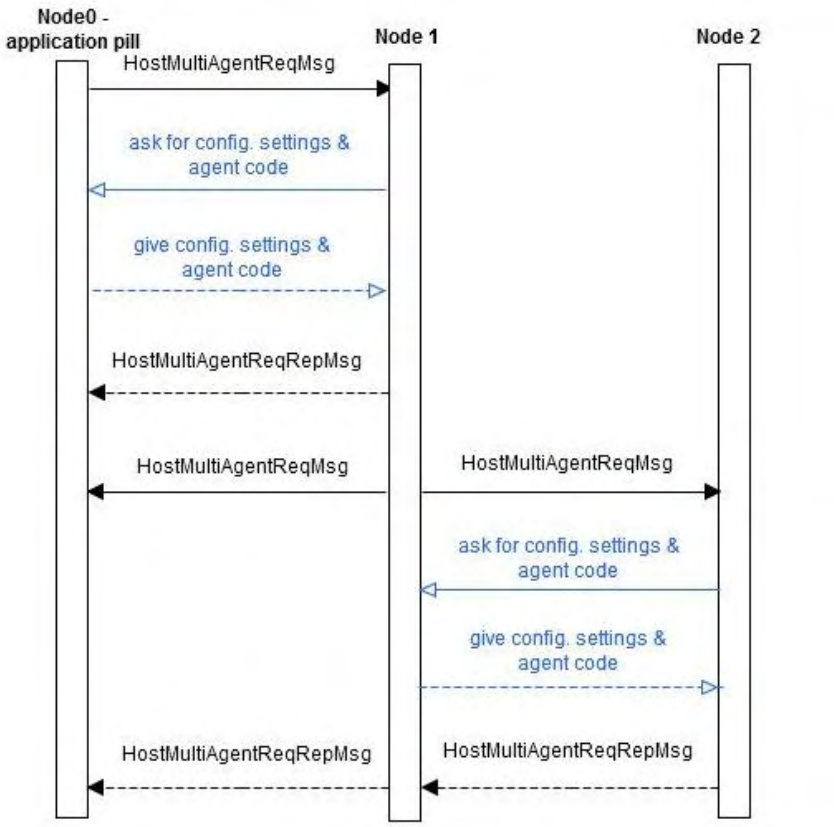
4.4 Κύριες διαφορές νέου και παλιού πρωτοκόλλου

Στο νέο πρωτόκολλο η αίτηση δημιουργίας του πράκτορα μεταδίδεται σε ολόκληρο το δίκτυο μέσω flooding αφού κάθε κόμβος όταν την λαμβάνει για πρώτη φορά την κάνει broadcast. Ακόμα, οι δύο φάσεις του προηγούμενου πρωτοκόλλου συγχωνεύτηκαν σε μία και πλέον οι κόμβοι που είναι ικανοί να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα δεν μπαίνουν σε λίστα ώστε να εξεταστούν ο καθένας ξεχωριστά, αλλά αιτούνται κατευθείαν τις ρυθμίσεις διαμόρφωσης και τον κώδικα του πράκτορα και αν επιτύχει η τοπική δημιουργία του στιγμιοτύπου στέλνουν ένα μήνυμα στον κόμβο που ξεκίνησε το flooding της αίτησης (δηλαδή εκεί που βρίσκεται ο parent agent). Τέλος, σε αντίθεση με το προηγούμενο πρωτόκολλο όπου όλοι οι κόμβοι αιτούνταν τις ρυθμίσεις και τον κώδικα από το application pill, πλέον κάθε κόμβος αιτείται αυτά τα στοιχεία από τον κόμβο με code address, που είναι στην ουσία ο κοντινότερος κόμβος στο δέντρο που δημιουργείται από το flooding της αίτησης δημιουργίας, ο οποίος αιτήθηκε και έλαβε αυτά τα στοιχεία με επιτυχία.

Για να γίνουν καλύτερα αντιληπτές οι διαφορές των δύο πρωτοκόλλων, στα επόμενα δύο σχήματα παρουσιάζονται τα διαγράμματα ροής μηνυμάτων σε ένα δίκτυο που αποτελείται από τρεις κόμβους οι οποίοι συνδέονται σε σειρά. Ο κόμβος 0 είναι το application pill που κάνει broadcast μια αίτηση για δημιουργία ομάδας μικρό-πρακτόρων και οι κόμβοι 1,2 ικανοποιούν τις non-generic απαιτήσεις. Για το παλιό πρωτόκολλο θεωρούμε ότι τα broadcast μηνύματα μεταδίδονται για MAX_HOPS=2 hops στο δίκτυο.



Σχ. 20: Ροή μηνυμάτων για τη δημιουργία δύο non-generic πρακτόρων σύμφωνα με το παλιό πρωτόκολλο.



Σχ. 21: Ροή μηνυμάτων για τη δημιουργία δύο non-generic πρακτόρων σύμφωνα με το νέο πρωτόκολλο.

Κεφάλαιο 5 Πειραματικά Αποτελέσματα

5.1 Εργαλεία

Για την διεξαγωγή των πειραμάτων για τον έλεγχο του πρωτοκόλλου που υλοποιήθηκε στα πλαίσια της Διπλωματικής Εργασίας χρησιμοποιήθηκε αρχικά το TOSSIM, ο προσομοιωτής του TinyOS, ενώ στην τελική φάση χρησιμοποιήθηκαν πραγματικοί κόμβοι IntelMote2.

5.1.1 TOSSIM

Το TOSSIM είναι ένας προσομοιωτής διακριτών γεγονότων για Ασύρματα Δίκτυα Αισθητήρων όπου στους κόμβους τρέχει το λειτουργικό σύστημα TinyOS.

Πιο συγκεκριμένα, το TOSSIM εκμεταλλεύεται την δομή του TinyOS και την μεταγλώττιση ολόκληρου του προγράμματος που αυτό κάνει, για να δημιουργήσει προσομοιώσεις διακριτών γεγονότων κατευθείαν από τους γράφους σύνδεσης των components του TinyOS. Στην ουσία, «τρέχει» σχεδόν τον ίδιο κώδικα που «τρέχει» και στο hardware ενός δικτύου αισθητήρων, αλλά αντικαθιστά μερικά χαμηλού-επιπέδου components με δικές του υλοποιήσεις ώστε να μετατρέπει τα hardware interrupts σε διακριτά ξεχωριστά γεγονότα προσομοίωσης τα οποία οδηγούν την εκτέλεση της εφαρμογής.

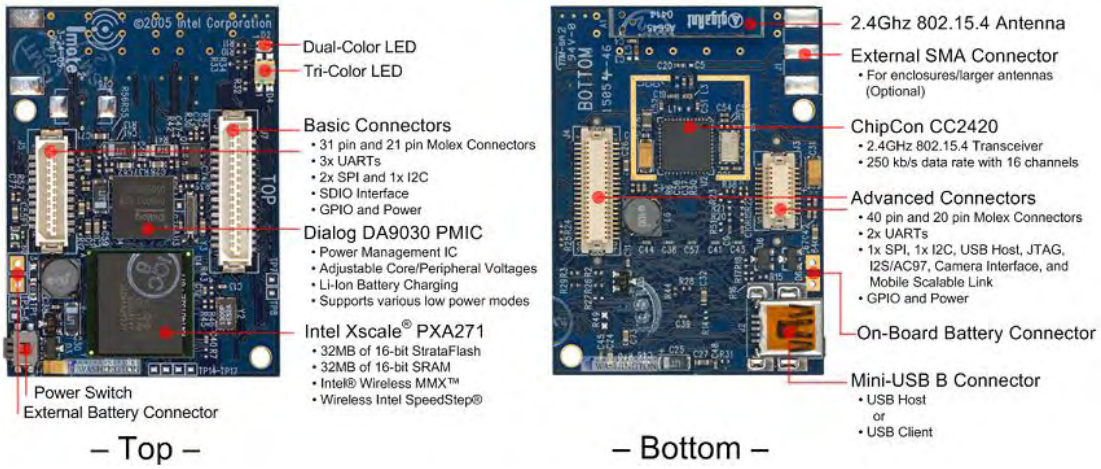
Για την διαχείριση της έναρξης ολόκληρου του POBICOS runtime υπάρχει το εργαλείο του *proxy manager*. Το συγκεκριμένο εργαλείο είναι μια εφαρμογή που ξεκινά μια ξεχωριστή διεργασία για κάθε runtime εκτελέσιμο που εκκινεί. Αυτά τα εκτελέσιμα αντιπροσωπεύουν κόμβους POBICOS ο καθένας από τους οποίους μπορεί να έχει διαφορετικά χαρακτηριστικά. Αφού δημιουργηθούν όλες οι διεργασίες, το εργαλείο *portplex* ξεκινά και αυτό ως μια ξεχωριστή διεργασία και συνδέεται σε όλα τα runtimes στις TCP/IP θύρες ώστε να προσφέρει επικοινωνία ανάμεσα στους διάφορους κόμβους του εικονικού δικτύου.

5.1.2 IntelMote2 (iMote2)

Η Imote2 είναι μια προηγμένη πλατφόρμα ασύρματου κόμβου αισθητήρα γενικού σκοπού. Αναπτύχθηκε στην Intel Research ως μέρος της Platform X και είναι χτισμένη γύρω από μια χαμηλής κατανάλωσης PXA271 XScale CPU. Επιπλέον, ενσωματώνει ένα δέκτη που υποστηρίζει επικοινωνία στα πρότυπα του πρωτοκόλλου IEEE 802.15.4. Ο σχεδιασμός της είναι σπονδυλωτός και διαθέτει υποδοχείς διεπαφών για σύνδεση με πλακέτες επέκτασης τόσο στην πάνω όσο και στην κάτω μεριά ενώ προσφέρει και δυνατότητα διασύνδεσης μέσω θύρας USB.

Τα βασικά της τεχνικά χαρακτηριστικά είναι:

- PXA271 XScale Processor @ 13-416 MHz
  - 256 KB SRAM
  - 32 MB SDRAM
  - 32 MB Flash
- Zigbee (IEEE 802.15.4) Radio (TI CC2420)



Σχ. 22: Άνω και κάτω όψη της πλακέτας iMote2.

5.2 Εφαρμογές δημιουργίας πρακτόρων στο POBICOS

Για την δοκιμή του πρωτοκόλλου που υλοποιήσαμε επιλέχθηκαν δύο εφαρμογές:

- Στην **εφαρμογή Α**, η οποία είναι η απλούστερη δυνατή, ο root agent αιτείται την δημιουργία μιας ομάδας non-generic πρακτόρων οι οποίοι δεν εκτελούν κάποια συγκεκριμένη εργασία.
- Στην **εφαρμογή Β**, ο root agent αιτείται την δημιουργία μιας ομάδας non-generic πρακτόρων οι οποίοι μετράνε περιοδικά την θερμοκρασία και στην περίπτωση που η τιμή της μέτρησης υπερβεί ένα άνω όριο στέλνουν ένα μήνυμα στον root agent.

Το κριτήριο της επιλογής των παραπάνω δύο εφαρμογών ήταν το μέγεθος των non-generic πρακτόρων των οποίων η δημιουργία ζητείται και κατά συνέπεια το πλήθος των πακέτων που απαιτείται για την αποστολή του κώδικα των πρακτόρων σε έναν κόμβο που ικανοποιεί τις εκάστοτε non-generic απαιτήσεις. Πιο συγκεκριμένα, για την μεταφορά του κώδικα του non-generic πράκτορα της εφαρμογής Α (στον οποίον από εδώ και στο εξής θα αναφερόμαστε ως **πράκτορα Α**) απαιτείται ένα μοναδικό πακέτο κώδικα, ενώ για την μεταφορά του κώδικα του non-generic πράκτορα της εφαρμογής Β (στον οποίον από εδώ και στο εξής θα αναφερόμαστε ως **πράκτορα Β**) απαιτείται η αποστολή δέκα (10) πακέτων κώδικα.

5.3 Πειραματικά αποτελέσματα στο TOSSIM

Στο προηγούμενο πρωτόκολλο δημιουργίας πρακτόρων η αποστολή και η λήψη μηνυμάτων γινόταν μέσω του transport layer του POBICOS που είχε υλοποιηθεί πάνω από τεχνολογία ZigBee. Για το λόγο αυτό γινόταν η υπόθεση ότι όλοι οι κόμβοι μπορεί να επικοινωνούν απευθείας μεταξύ τους, δηλαδή ο γράφος επικοινωνίας ήταν πάντα πλήρως διασυνδεδεμένος (βέβαια αυτό δεν γίνεται σε φυσικό επίπεδο καθώς το ZigBee υλοποιεί δική του δρομολόγηση).

Για τον έλεγχο της λειτουργικότητας του νέου πρωτόκολλου απαιτούνταν η χρήση τοπολογιών και ήταν απαραίτητη η δυνατότητα επικοινωνίας μεταξύ κόμβων που δεν είναι άμεσα συνδεδεμένοι. Για αυτό χρησιμοποιήθηκε ένα διαφορετικό transport layer που περιλαμβάνει μια υλοποίηση του AODV πρωτοκόλλου ειδικά για το TinyOS (TinyAODV[1]).

Οι τοπολογίες που επιλέχθηκαν για την διενέργεια των πειραμάτων στο TOSSIM ήταν οι εξής:

- Τοπολογία αστεριού με 5 κόμβους: όλοι οι κόμβοι του δικτύου είναι συνδεδεμένοι με έναν κεντρικό κόμβο. Σε όλα τα πειράματα ως application pill λειτουργεί ο κεντρικός κόμβος ο οποίος δεν ικανοποιεί τις non-generic απαιτήσεις.

Σχ. 23: Η τοπολογία αστεριού που χρησιμοποιήθηκε στα πειράματα.

- Τοπολογία αλυσίδας με 5 κόμβους: κάθε κόμβος είναι συνδεδεμένος με τους άμεσους γείτονές του στην αλυσίδα. Σε όλα τα πειράματα ως application pill λειτουργεί ο πρώτος κόμβος της αλυσίδας ο οποίος δεν ικανοποιεί τις non-generic απαιτήσεις.

Σχ. 24: Η τοπολογία αλυσίδας που χρησιμοποιήθηκε στα πειράματα.

Οι μετρικές που χρησιμοποιήθηκαν για την αξιολόγηση του πρωτοκόλλου ήταν οι εξής:

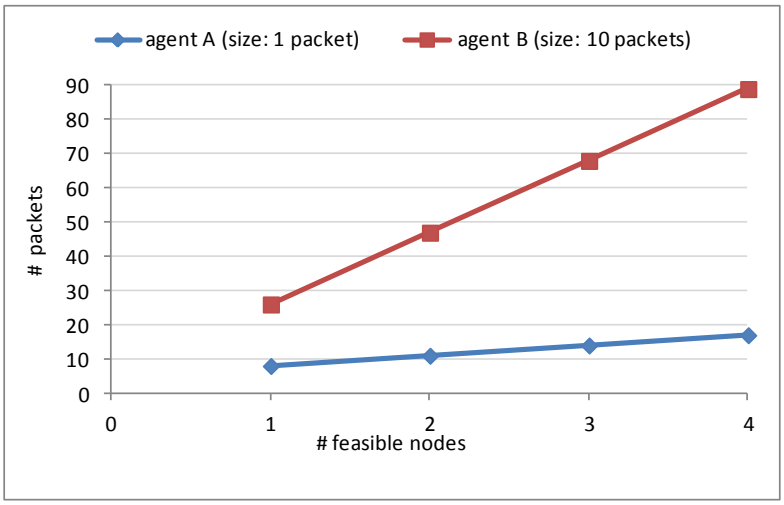
- Κάλυψη (Coverage): πόσοι κόμβοι έλαβαν την αίτηση δημιουργίας του non-generic πράκτορα.
- Ποσοστό επιτυχίας (Success): πόσοι κόμβοι από αυτούς που ικανοποιούσαν τις non-generic απαιτήσεις τελικά δημιούργησαν ένα στιγμιότυπο του πράκτορα τοπικά.
- Καθυστέρηση δημιουργίας (Creation latency): πόσο χρόνο πήρε στον τελευταίο κόμβο που ικανοποιούσε τις non-generic απαιτήσεις να δημιουργήσει τον πράκτορα.
- Συνολικός αριθμός μεταδιδόμενων πακέτων (Total number of packets transmitted): ο συνολικός αριθμός των πακέτων που μεταδόθηκαν στο δίκτυο *στο πλαίσιο του πρωτοκόλλου* δημιουργίας πρακτόρων.

[1] http://vasiliadisv.wordpress.com/2012/03/11/aodv-implementation-based-on-tinyaodv/

5.3.1 Τοπολογία αστεριού

Σε όλα τα πειράματα που διενεργήθηκαν με αυτή την τοπολογία, η κάλυψη και το ποσοστό επιτυχίας ήταν 100.00%.

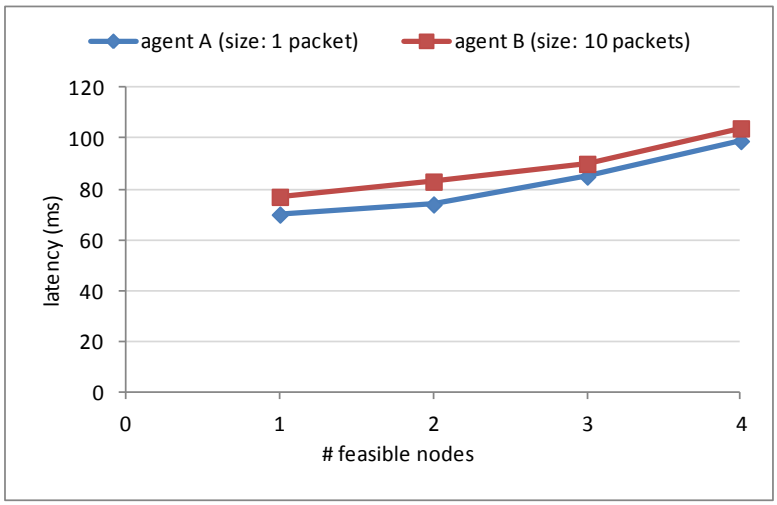
Οι μετρήσεις για τον αριθμό των πακέτων που μεταδόθηκαν ως προς το πλήθος των ικανών κόμβων να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα παρουσιάζονται γραφικά στο Σχ. 25.



Σχ. 25: Συνολικός αριθμός μεταδιδόμενων πακέτων για την δημιουργία των πρακτόρων Α και Β ως προς το πλήθος των κόμβων που ικανοποιούν τις non-generic απαιτήσεις στην τοπολογία αστεριού.

Όπως φαίνεται, το πλήθος των μεταδιδόμενων πακέτων αυξάνεται γραμμικά τόσο για τον πράκτορα Α όσο και για τον πράκτορα Β με τον δεύτερο να δημιουργεί αρκετά μεγαλύτερο φόρτο στο δίκτυο, κάτι που ήταν αναμενόμενο λόγω του μεγαλύτερου μεγέθους του.

Οι μετρήσεις για την καθυστέρηση δημιουργίας ως προς το πλήθος των ικανών κόμβων να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα παρουσιάζονται γραφικά στο Σχ. 26.



Σχ. 26: Καθυστέρηση δημιουργίας για τους πράκτορες Α και Β ως προς το πλήθος των κόμβων που ικανοποιούν τις non-generic απαιτήσεις στην τοπολογία αστεριού.

Όπως φαίνεται, τόσο για τον πράκτορα Α όσο και για τον πράκτορα Β, όταν αυξάνεται ο αριθμός των κόμβων που δημιουργούν τον πράκτορα τοπικά, αυξάνεται και η καθυστέρηση δημιουργίας. Στην τοπολογία αστεριού αν και οι περιφερειακοί κόμβοι λαμβάνουν ταυτόχρονα την αίτηση από τον κεντρικό κόμβο, στη συνέχεια αυτός θα πρέπει να στείλει ξεχωριστά μηνύματα σε όσους έχουν την δυνατότητα να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα με αποτέλεσμα να υπάρχει αυτή η μικρή αύξηση στην καθυστέρηση δημιουργίας όταν το πλήθος των κόμβων που μπορούν να λειτουργήσουν ως ξενιστές αυξάνεται. Επιπλέον, παρατηρούμε ότι η καμπύλη της καθυστέρησης δημιουργίας του πράκτορα Β βρίσκεται λίγο ψηλότερα από την αντίστοιχη καμπύλη του πράκτορα Α κάτι το οποίο οφείλεται στην διαφορά μεγεθών των δύο πρακτόρων.

5.3.2 Τοπολογία αλυσίδας

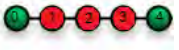
Σε όλα τα πειράματα που διενεργήθηκαν με αυτήν την τοπολογία, η κάλυψη και το ποσοστό επιτυχίας ήταν 100.00%.

Τα αποτελέσματα των μετρήσεων για τον αριθμό των πακέτων που μεταδόθηκαν ανάλογα με την τοπολογία του δικτύου, παρουσιάζονται στον Πίνακα 1 ομαδοποιημένα ως προς την συνολική απόσταση των κόμβων που ήταν ικανοί να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα από τους κόμβους που έλαβαν τον κώδικα του πράκτορα και ταξινομημένα σε κάθε περίπτωση ως προς την συνολική απόσταση των κόμβων αυτών από τον πρώτο κόμβο του δικτύου όπου βρίσκεται ο root agent.

Τα πεδία του Πίνακα 1 είναι τα εξής:

- C, η συνολική απόσταση σε hops των ικανών κόμβων να φιλοξενήσουν στιγμιότυπο του πράκτορα από την code address, δηλαδή τον κόμβο από τον οποίο αιτούνται και από τον οποίο λαμβάνουν τον κώδικα του πράκτορα
- S, η συνολική απόσταση σε hops των ικανών κόμβων να φιλοξενήσουν στιγμιότυπο του πράκτορα από την source address, δηλαδή τον κόμβο όπου βρίσκεται ο root agent
- Network topology, η τοπολογία του δικτύου όπου οι ικανοί να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα κόμβοι είναι μαρκαρισμένοι με κόκκινο χρώμα
- $\text{Packets}_{\text{A}}$, ο συνολικός αριθμός μεταδιδόμενων πακέτων για τον πράκτορα Α μεγέθους ενός (1) πακέτου
- $\text{Packets}_{\text{B}}$, ο συνολικός αριθμός μεταδιδόμενων πακέτων για τον πράκτορα Β μεγέθους δέκα (10) πακέτων

Πίνακας 1: Συνολικός αριθμός μεταδιδόμενων πακέτων για την δημιουργία των πρακτόρων Α και Β ανάλογα με την τοποθέτηση των κόμβων που ικανοποιούν τις non-generic απαιτήσεις στην τοπολογία αλυσίδας.

C	S	Network topology	Packets_A	Packets_B
1	1		8	26
2	2		11	47
	3		12	48
3	3		14	68
	4		15	69
	5		16	70
	6		17	71
4	4		17	89
	5		18	90
	6		19	91
	7		20	92
	8		21	93
	9		22	94
	10		23	95

Όπως παρατηρούμε, όσο αυξάνεται η συνολική απόσταση των κόμβων που ικανοποιούν τις non-generic απαιτήσεις από τους κόμβους από τους οποίους λαμβάνουν τον κώδικα του πράκτορα (C, πρώτη στήλη του πίνακα), τόσο αυξάνεται το πλήθος των μεταδιδόμενων πακέτων, σε συνάρτηση πάντα με την συνολική απόσταση των κόμβων αυτών από την πρώτο κόμβο του δικτύου ο οποίος αιτείται αρχικά την δημιουργία τους (S, δεύτερη στήλη του πίνακα).

Επίσης, φαίνεται ξεκάθαρα ότι το πλήθος των μεταδιδόμενων πακέτων για τον πράκτορα Β (Packets_B, πέμπτη στήλη του πίνακα) είναι πάντα μεγαλύτερο από το αντίστοιχο πλήθος μεταδιδόμενων πακέτων για τον πράκτορα Α (Packets_Aτέταρτη στήλη του πίνακα). Αυτό ήταν αναμενόμενο από τη στιγμή που η μεταφορά του κώδικα του πράκτορα Β απαιτεί τον δεκαπλάσιο αριθμό πακέτων κώδικα από αυτόν του πράκτορα Α.

Όσον αφορά την καθυστέρηση δημιουργίας, τα αποτελέσματα των μετρήσεων παρουσιάζονται στον Πίνακα 2 ομαδοποιημένα ως προς το πλήθος των ικανών κόμβων να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα και ταξινομημένα σε κάθε περίπτωση ως προς άθροισμα της συνολικής απόστασης των ικανών να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα κόμβων από την code και την source address.

Τα πεδία του Πίνακα 2 είναι τα εξής:

- F, το πλήθος των ικανών κόμβων να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα
- Network topology, η τοπολογία του δικτύου όπου οι ικανοί να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα κόμβοι είναι μαρκαρισμένοι με κόκκινο χρώμα
- Latency_A, η καθυστέρηση δημιουργίας για τον πράκτορα A μεγέθους ενός (1) πακέτου
- Latency_B, καθυστέρηση δημιουργίας για τον πράκτορα B μεγέθους δέκα (10) πακέτων

Πίνακας 2: Καθυστέρηση δημιουργία των πρακτόρων A και B ανάλογα με την τοποθέτηση των κόμβων που ικανοποιούν τις non-generic απαιτήσεις στην τοπολογία αλυσίδας.

F	Network topology	Latency_A	Latency_B
1		59.8 ms	78.3 ms
		68.2 ms	80 ms
		69.6 ms	81.5 ms
		75.2 ms	100.2 ms
2		95 ms	107.75 ms
		98.6 ms	112.2 ms
		100.4 ms	113 ms
		105 ms	116.3 ms
		105.5 ms	117 ms
		109.8 ms	120.8 ms
3		131.75 ms	145.8 ms
		137.2 ms	164 ms
		139 ms	169.5 ms
4		189.2 ms	196.2 ms

Το πρώτο που παρατηρούμε είναι ότι όσο αυξάνεται το πλήθος των κόμβων που έχουν τη δυνατότητα να δημιουργήσουν τον πράκτορα τοπικά, τόσο αυξάνεται και η καθυστέρηση δημιουργίας. Αυτό εξηγείται από το πρωτόκολλου που υλοποιήσαμε γιατί σε αυτό ένας ικανός κόμβος κάνει broadcast την αίτηση αφού πρώτα δημιουργήσει ένα στιγμιότυπο του πράκτορα. Άρα, όσο αυξάνει το πλήθος των ικανών κόμβων που προηγούνται από τον τελευταίο ικανό, τόσο αυξάνεται και η καθυστέρηση δημιουργίας.

Επιπλέον, παρατηρούμε ότι για την ίδια τοπολογία ο πράκτορας Α έχει μικρότερη καθυστέρηση δημιουργίας από τον Β, κάτι το οποίο οφείλεται στο μικρότερο μέγεθος του.

Τέλος, εξετάζοντας τις ομαδοποιημένες γραμμές του πίνακα βλέπουμε ότι:

- Στις πρώτες τέσσερεις γραμμές, που αφορούν *έναν* κόμβο που ικανοποιεί τις non-generic απαιτήσεις, βλέπουμε ότι όσο αυξάνεται η απόσταση από τον αρχικό κόμβο με τον οποίο ανταλλάσσονται μηνύματα για μεταφορά των ρυθμίσεων και του κώδικα του πράκτορα, τόσο αυξάνεται και η καθυστέρηση δημιουργίας.
- Στις επόμενες έξι γραμμές, οι οποίες αφορούν *δύο* κόμβους που ικανοποιούν τις non-generic απαιτήσεις, παρατηρούμε ότι όσο αυξάνεται η συνολική απόσταση των ικανών κόμβων από τους κόμβους που λαμβάνουν τις ρυθμίσεις και τον κώδικα του πράκτορα (code address), τόσο αυξάνεται και η καθυστέρηση δημιουργίας. Για παράδειγμα, όταν ικανοί κόμβοι είναι ο 1 και ο 2, η συνολική απόσταση από τους code address κόμβους είναι 2 (ο 1 λαμβάνει τα απαραίτητα στοιχεία από τον αρχικό κόμβο της αλυσίδας 0 ενώ ο 2 τα λαμβάνει από τον κόμβο 1) και η καθυστέρηση δημιουργίας είναι 95ms για πράκτορα Α και 107.75ms για τον πράκτορα Β. Όταν όμως ικανοί κόμβοι είναι ο 2 και ο 3, η συνολική απόσταση από τους code address κόμβους είναι 3 (ο 2 λαμβάνει τα απαραίτητα στοιχεία από τον αρχικό κόμβο της αλυσίδας 0 ενώ ο 3 τα λαμβάνει από τον κόμβο 2), η καθυστέρηση δημιουργίας αυξάνεται σε 98.6ms και 112.2ms για τους πράκτορες Α και Β αντίστοιχα.
- Το ίδιο συμπέρασμα εξάγουμε και από τις επόμενες τρεις γραμμές που αφορούν τρείς κόμβους που ικανοποιούν τις non-generic απαιτήσεις.

5.4 Πειραματικά αποτελέσματα σε πραγματικούς κόμβους

Οι εφαρμογές Α και Β, σε τελευταίο στάδιο αναπτύχθηκαν επιτυχώς σε πραγματικούς κόμβους IntelMote 2, όμως εξαιτίας της δυσκολίας δημιουργίας μεγάλων τοπολογιών σε εργαστηριακό περιβάλλον οι αντίστοιχες τοπολογίες που χρησιμοποιήθηκαν στα πειράματα ήταν οι εξής:

- Τοπολογία αστεριού με 3 κόμβους: δύο κόμβοι είναι συνδεδεμένοι με έναν κεντρικό κόμβο. Σε όλα τα πειράματα ως application pill λειτουργεί ο κεντρικός κόμβος ο οποίος δεν ικανοποιεί τις non-generic απαιτήσεις.

- Τοπολογία αλυσίδας με 3 κόμβους: κάθε κόμβος είναι συνδεδεμένος με τους άμεσους γείτονές του στην αλυσίδα. Σε όλα τα πειράματα ως application pill λειτουργεί ο πρώτος κόμβος της αλυσίδας ο οποίος δεν ικανοποιεί τις non-generic απαιτήσεις.

5.4.1 Τοπολογία αστεριού

Σε όλα τα πειράματα που διενεργήθηκαν με αυτήν την τοπολογία, η κάλυψη και το ποσοστό επιτυχίας ήταν 100.00%.

Τα αποτελέσματα για το πλήθος των μεταδιδόμενων πακέτων στο πλαίσιο του πρωτοκόλλου δημιουργίας μικρό-πρακτόρων παρουσιάζονται στον Πίνακα 3.

Τα πεδία του Πίνακα 3 είναι τα εξής:

- F, το πλήθος των ικανών κόμβων να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα
- Packets_A, ο συνολικός αριθμός μεταδιδόμενων πακέτων για τον πράκτορα A
- Packets_B, ο συνολικός αριθμός μεταδιδόμενων πακέτων για τον πράκτορα B

Πίνακας 3: Συνολικός αριθμός μεταδιδόμενων πακέτων για την δημιουργία των πρακτόρων A και B ανάλογα με το πλήθος των κόμβων που ικανοποιούν τις non-generic απαιτήσεις στην τοπολογία αστεριού.

F	Packets_A	Packets_B
1	6	24
2	9	45

5.4.2 Τοπολογία αλυσίδας

Σε όλα τα πειράματα που διενεργήθηκαν με αυτήν την τοπολογία, η κάλυψη και το ποσοστό επιτυχίας ήταν 100.00%.

Τα αποτελέσματα για το πλήθος των μεταδιδόμενων πακέτων στο πλαίσιο του πρωτοκόλλου δημιουργίας μικρό-πρακτόρων παρουσιάζονται στον Πίνακα .

Τα πεδία του Πίνακα 4 είναι τα εξής:

- F, το πλήθος των ικανών κόμβων να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα
- Network topology, η τοπολογία του δικτύου όπου οι ικανοί να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα κόμβοι είναι μαρκαρισμένοι με κόκκινο χρώμα
- Packets_A, ο συνολικός αριθμός μεταδιδόμενων πακέτων για τον πράκτορα A
- Packets_B, ο συνολικός αριθμός μεταδιδόμενων πακέτων για τον πράκτορα B

Πίνακας 4: Συνολικός αριθμός μεταδιδόμενων πακέτων για την δημιουργία των πρακτόρων Α και Β ανάλογα με την τοποθέτηση των κόμβων που ικανοποιούν τις non-generic απαιτήσεις στην τοπολογία αλυσίδας.

F	Network topology	$Packets_A$	$Packets_B$
1		6	24
		9	45
2		10	46

5.5 Σύγκριση νέου και παλιού πρωτοκόλλου

Ο συνολικός αριθμός των πακέτων που μεταδίδονται στο δίκτυο με χρήση της νέας υλοποίησης του πρωτοκόλλου είναι το άθροισμα των μηνυμάτων HostMultiAgentReq και HostMultiAgentReqRep καθώς και των μηνυμάτων που στέλνονται για την μεταφορά των ρυθμίσεων διαμόρφωσης και του κώδικα του πράκτορα.

Σύμφωνα με την παλιά υλοποίηση του πρωτοκόλλου, ο συνολικός αριθμός των πακέτων που μεταδίδονται στο δίκτυο είναι το άθροισμα των μηνυμάτων HostAgentProbe, HostAgentProbeRep, HostAgentReq, HostAgentReqRep και των μηνυμάτων που στέλνονται για την μεταφορά των ρυθμίσεων διαμόρφωσης και του κώδικα του πράκτορα.

Αν υποθέσουμε ότι οι ρυθμίσεις διαμόρφωσης δεν είναι απαραίτητο να σταλούν καθώς ο πράκτορας μπορεί να χρησιμοποιήσει τις προεπιλεγμένες και αν θεωρήσουμε ότι:

- N: ο συνολικός αριθμός των κόμβων του δικτύου
- S: η συνολική απόσταση σε hops των κόμβων που ικανοποιούν τις non-generic απαιτήσεις από την source address
- C: η συνολική απόσταση σε hops των κόμβων που ικανοποιούν τις non-generic απαιτήσεις από την code address, δηλαδή τον κόμβο από τον οποίο αιτούνται και από τον οποίο λαμβάνουν τον κώδικα του πράκτορα
- X: το πλήθος των πακέτων που απαιτούνται για να σταλεί ολόκληρος ο κώδικας του πράκτορα, το οποίο ισούται με το μέγεθος του κώδικα του πράκτορα προς το μέγιστο μέγεθος «χρήσιμης» πληροφορίας που χωράει σε ένα πακέτο

Τότε, στην νέα υλοποίηση:

- Το πλήθος των πακέτων των HostMultiAgentReq μηνυμάτων είναι ίσο με N (κάθε κόμβος κάνει broadcast την αίτηση όταν την λαμβάνει για πρώτη φορά)
- Το πλήθος των πακέτων των μηνυμάτων που στέλνονται για την λήψη του κώδικα του πράκτορα είναι 2*C*X (τα μηνύματα αιτήσεων κώδικα είναι όσα και τα μηνύματα παράδοσης κώδικα)
- Το πλήθος των πακέτων των HostMultiAgentReqRep μηνυμάτων είναι ίσο με S

Άρα, ο συνολικός αριθμός των πακέτων που μεταδίδονται δίνεται από την παρακάτω εξίσωση:

- $\mathbf{PacketsTransmitted_{new\ protocol} = 2 * C * X + S + N}$ (1)

Στην παλιά υλοποίηση:

- Το πλήθος των πακέτων των HostAgentProbe μηνυμάτων είναι και αυτό ίσο με N
- Το πλήθος των πακέτων των HostAgentProbeRep μηνυμάτων είναι ίσο με S
- Το πλήθος των πακέτων των HostAgentReq μηνυμάτων είναι ίσο με S
- Το πλήθος των πακέτων των μηνυμάτων που στέλνονται για την λήψη του κώδικα του πράκτορα είναι 2*S*X (τα μηνύματα αιτήσεων κώδικα είναι όσα και τα μηνύματα παράδοσης κώδικα)
- Το πλήθος των πακέτων των HostAgentReqRep μηνυμάτων είναι ίσο με S

Άρα, ο συνολικός αριθμός των πακέτων που μεταδίδονται δίνεται από την παρακάτω εξίσωση:

- $\mathbf{PacketsTransmitted_{old\ protocol} = 2 * S * X + 3 * S + N}$ (2)

Παρατηρούμε ότι:

- $C \leq S \overset{2*X}{\Longrightarrow} 2 * C * X \leq 2 * S * X$ (3)
- $S < 3 * S$ (4)

Προσθέτοντας κατά μέλη τις ανισότητες (3) και (4) και με χρήση των εξισώσεων (1) και (2) καταλήγουμε στα εξής συμπεράσματα:

- $2 * C * X + S < 2 * S * X + 3 * S \overset{+N}{\Rightarrow}$
- $2 * C * X + S + N < 2 * S * X + 3 * S + N \overset{(1),(2)}{\Longrightarrow}$
- $\mathbf{PacketsTransmitted_{new\ protocol} < PacketsTransmitted_{old\ protocol}}$

Άρα, η νέα υλοποίηση του πρωτοκόλλου δημιουργίας πρακτόρων δημιουργεί λιγότερη κίνηση στο δίκτυο και είναι πάντα πιο αποδοτική από την προηγούμενη υλοποίηση.

Για να γίνει αισθητή η σύγκριση των δύο πρωτοκόλλων έχουμε παραστήσει γραφικά το πλήθος των μεταδιδόμενων πακέτων για διάφορες τοπολογίες και διάφορα μεγέθη πρακτόρων. Πιο συγκεκριμένα έχουμε θεωρήσει τις περιπτώσεις όπου η μεταφορά του κώδικα του πράκτορα απαιτεί την αποστολή:

i. Ενός πακέτου κώδικα από τον κόμβο που τον έχει προς τον κόμβο που τον ζητάει , δηλαδή X=1.
ii. Δέκα πακέτων κώδικα από τον κόμβο που τον έχει προς τον κόμβο που τον ζητάει , δηλαδή X=10.
iii. Είκοσι πακέτων κώδικα από τον κόμβο που τον έχει προς τον κόμβο που τον ζητάει , δηλαδή X=20.

Στην συνέχεια, θα αναφερόμαστε στους πράκτορες που δημιουργούνται στις περιπτώσεις (i), (ii), (iii) ως **πράκτορες Α,Β** και **Γ** αντίστοιχα.

5.5.1 Τοπολογία αστεριού

Για μια **τοπολογία αστεριού** όπου ο κεντρικός κόμβος είναι η source address, όλοι οι περιφερικοί κόμβοι λαμβάνουν την αίτηση δημιουργίας και (όσοι μπορούν να λειτουργήσουν ως ξενιστές) τον κώδικα του πράκτορα, από τον κεντρικό κόμβο. Δηλαδή, η source address είναι και code address και ισχύει ότι:

- $C = S$ (5)

Επίσης, η απόσταση όλων των κόμβων από τον κεντρικό ισούται με ένα (1), κάτι που σημαίνει ότι η συνολική απόσταση των κόμβων που ικανοποιούν τις non-generic απαιτήσεις από τον κεντρικό, ισούται με το πλήθος τους. Δηλαδή, αν συμβολίσουμε με F αυτό το πλήθος, ισχύει ότι:

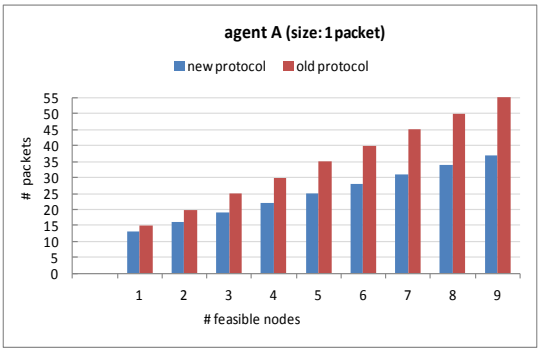
- $C = S = F$ (6)

Άρα:

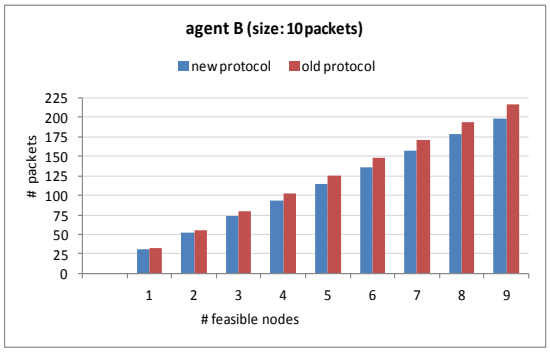
- (1) $\overset{(5),(6)}{\Longrightarrow}$ $\mathbf{PacketsTransmitted_{new\ protocol} = F * (2 * X + 1) + N}$ (7)
- (2) $\overset{(6)}{\Rightarrow}$ $\mathbf{PacketsTransmitted_{old\ protocol} = F * (2 * X + 3) + N}$ (8)

Στον Πίνακα 5 φαίνεται η σύγκριση του παλιού και του νέου πρωτοκόλλου για μια τοπολογία αστεριού που αποτελείται από δέκα κόμβους (N=10) για τους πράκτορες Α, Β και Γ.

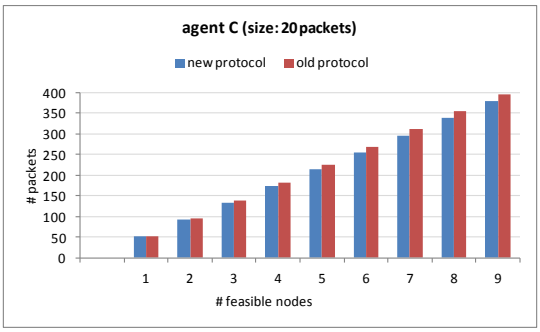
Πίνακας 5: Πλήθος μεταδιδόμενων πακέτων προς το πλήθος των ικανών κόμβων να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα (Σχ. 27-29) και επί τοις εκατό βελτίωση με τη χρήση του νέου πρωτοκόλλου (Σχ. 30) για τοπολογία αστεριού 10 κόμβων.



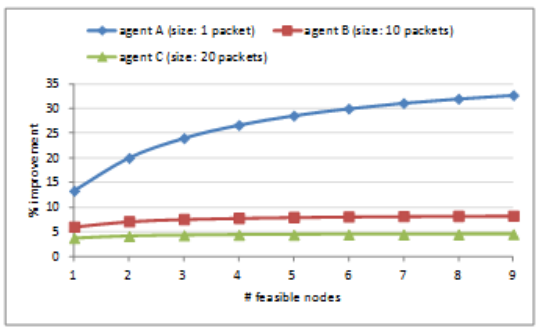
Σχ. 27



Σχ. 28



Σχ. 29



Σχ. 30

5.5.2 Τοπολογία αλυσίδας

Για την **τοπολογία αλυσίδας** θα εξετάσουμε τις δύο ακραίες περιπτώσεις, δηλαδή όταν υπάρχει μόνο ένας κόμβος που ικανοποιεί τις non-generic απαιτήσεις και όταν όλοι οι κόμβοι (εκτός της source address) ικανοποιούν τις non-generic απαιτήσεις.

❖ **Περίπτωση 1**

Όταν μόνο *ένας* κόμβος ικανοποιεί τις non-generic απαιτήσεις, αυτός ο κόμβος λαμβάνει τον κώδικα του πράκτορα από την source address.
Δηλαδή και εδώ ισχύει η σχέση (5):

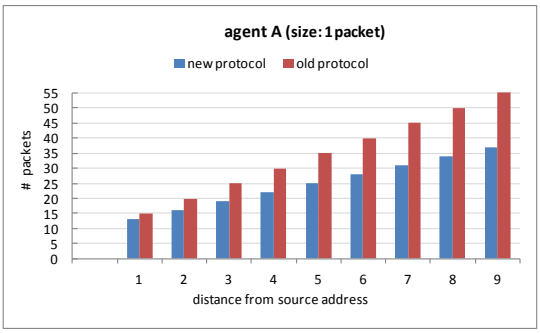
- $C = S$

Άρα:

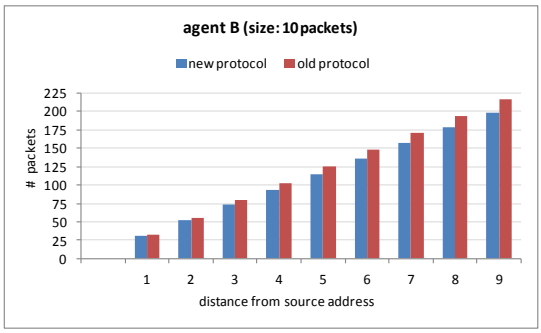
- $(1) \overset{(5)}{\Rightarrow} \mathbf{PacketsTransmitted_{new\ protocol}} = \mathbf{S} * (\mathbf{2} * \mathbf{X} + \mathbf{1}) + \mathbf{N}$ (9)
- $\mathbf{PacketsTransmitted_{old\ protocol}} = \mathbf{S} * (\mathbf{2} * \mathbf{X} + \mathbf{3}) + \mathbf{N}$ (10)

Στον Πίνακα 6 φαίνεται η σύγκριση του παλιού και του νέου πρωτοκόλλου για τοπολογίες αλυσίδας της περίπτωσης 1 που αποτελούνται από δέκα κόμβους (N=10) για τους πράκτορες Α, Β και Γ.

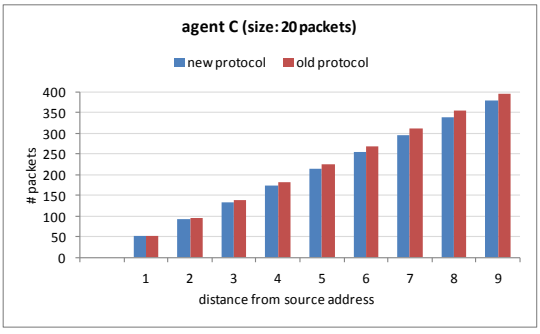
Πίνακας 6: Πλήθος μεταδιδόμενων πακέτων προς την απόσταση του μοναδικού ικανού να φιλοξενήσει ένα στιγμιότυπο του πράκτορα κόμβου από τον αρχικό κόμβο (source address) (Σχ.31-33) και επί τοις εκατό βελτίωση με τη χρήση του νέου πρωτοκόλλου (Σχ. 34) για τοπολογία αλυσίδας 10 κόμβων.



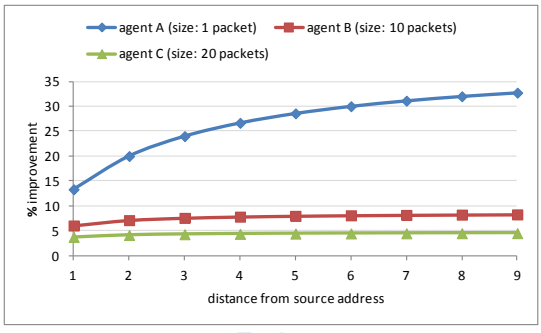
Σχ. 31



Σχ. 32



Σχ. 33



Σχ. 34

❖ Περίπτωση 2

Όταν *όλοι* οι κόμβοι του δικτύου (εκτός της source address) ικανοποιούν τις non-generic απαιτήσεις, στο νέο πρωτόκολλο, κάθε κόμβος λαμβάνει τον κώδικα του πράκτορα από τον κόμβο που προηγείται του ιδίου στην αλυσίδα.

Δηλαδή ισχύει ότι:

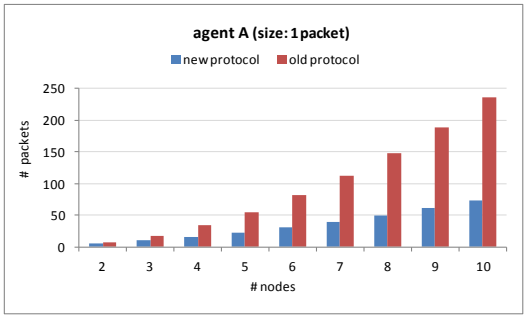
- $C = N - 1$ (6)

Άρα:

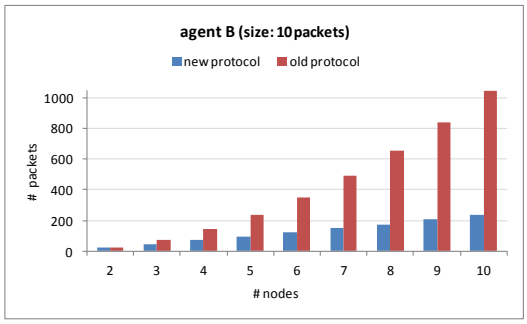
- (1) $\overset{(6)}{\Rightarrow}$ $\mathbf{PacketsTransmitted_{new\ protocol} = 2 * (N - 1) * X + S + N}$
- $\mathbf{PacketsTransmitted_{old\ protocol} = 2 * S * X + 3 * S + N}$

Στον Πίνακα 7 φαίνεται η σύγκριση του παλιού και του νέου πρωτοκόλλου για τοπολογίες αλυσίδας της περίπτωσης 2 που αποτελούνται από δύο έως δέκα κόμβους για τους πράκτορες Α, Β και Γ.

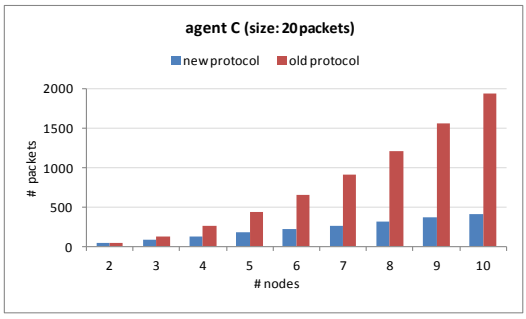
Πίνακας 7: Πλήθος μεταδιδόμενων πακέτων προς το πλήθος των κόμβων του δικτύου για τοπολογίες αλυσίδας όπου όλοι οι κόμβοι εκτός του αρχικού (source address) είναι ικανοί να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα (Σχ. 35-37) και επί τοις εκατό βελτίωση με τη χρήση του νέου πρωτοκόλλου (Σχ. 38).



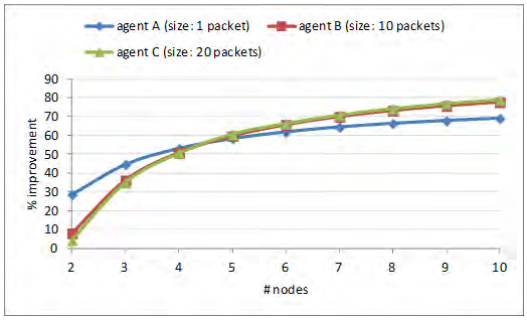
Σχ. 35



Σχ. 36



Σχ. 37



Σχ. 38

5.6 Σύνοψη αποτελεσμάτων

Συνοψίζοντας τα αποτελέσματα της ενότητας 5.5 καταλήγουμε στα εξής συμπεράσματα:

- Για τοπολογίες αστεριού το νέο πρωτόκολλο είναι πιο αποδοτικό από το παλιό και πιο συγκεκριμένα, οδηγεί στην μετάδοση $\mathbf{2} * \boldsymbol{F}$ λιγότερων πακέτων στο δίκτυο ανεξαρτήτως του μεγέθους του πράκτορα, όπου F το πλήθος των κόμβων που μπορούν να φιλοξενήσουν ένα στιγμιότυπο του. Στο Σχ. 30 φαίνεται ότι όσο αυξάνεται το F αυξάνεται και η ποσοστιαία ελάττωση του πλήθους των πακέτων, η οποία σε δίκτυο 10 κόμβων για μικρό μέγεθος πράκτορα φτάνει ως το 33% ενώ όσο μεγαλώνει το μέγεθος, το ποσοστό της βελτίωσης πέφτει στο 4,5%.
- Για τοπολογίες αλυσίδας όπου υπάρχει μόνο ένας ικανός κόμβος να φιλοξενήσει ένα στιγμιότυπο του πράκτορα, στο πλαίσιο του νέου πρωτοκόλλου μεταδίδονται $\mathbf{2} * \mathbf{S}$ λιγότερα πακέτα στο δίκτυο σε σχέση με το παλιό, όπου S η απόσταση του ικανού κόμβου από τον αρχικό ο οποίος αιτείται την δημιουργία του πράκτορα. Στο Σχ. 34 φαίνεται ότι όσο αυξάνεται το S, αυξάνεται και η επί τοις εκατό βελτίωση που προσφέρει το νέο πρωτόκολλο και η οποία σε δίκτυο 10 κόμβων φτάνει ως το 33% για μικρό μέγεθος πράκτορα και ως το 4,5% για εικοσαπλάσιο μέγεθος πράκτορα.
- Για τοπολογίες αλυσίδας όπου όλοι οι κόμβοι του δικτύου μπορούν να φιλοξενήσουν ένα στιγμιότυπο του πράκτορα, το νέο πρωτόκολλο είναι κατά πολύ πιο αποδοτικό από το παλιό. Πιο συγκεκριμένα, με χρήση του νέου πρωτοκόλλου μεταδίδονται $\mathbf{2} * \mathbf{S} + \mathbf{2} * (\mathbf{S} - \mathbf{C}) * \mathbf{X}$ λιγότερα πακέτα στο δίκτυο, όπου S το άθροισμα των αποστάσεων όλων των κόμβων από τον αρχικό από τον οποίο λαμβάνουν τον κώδικα στο παλιό πρωτόκολλο, C το άθροισμα των αποστάσεων όλων των κόμβων από τον κόμβο που λαμβάνουν τον κώδικα στο νέο πρωτόκολλο (και στην ουσία για τον καθένα είναι ο προηγούμενος του στην αλυσίδα, δηλαδή θα είναι C=N-1) και X το μέγεθος του πράκτορα. Στο Σχ. 38 φαίνεται ότι όσο αυξάνεται το πλήθος των κόμβων του δικτύου και το μέγεθος του πράκτορα, η βελτίωση που προσφέρει το νέο πρωτόκολλο αυξάνεται και φτάνει ως και το 79% για δίκτυο δέκα κόμβων και μέγεθος πράκτορα που ισούται με 20 πακέτα.

Κεφάλαιο 6 Επίλογος

6.1 Σύνοψη

Στην παρούσα Διπλωματική Εργασία παρουσιάσαμε ένα νέο και πιο αποδοτικό πρωτόκολλο για την δημιουργία μικρό-πρακτόρων στο πλαίσιο της πλατφόρμας POBICOS η οποία απευθύνεται σε ετερογενή Ασύρματα Δίκτυα Αισθητήρων. Βασική μας επιδίωξη ήταν η μείωση του φόρτου που προκαλείται στο δίκτυο από τη συγκεκριμένη διαδικασία, κάτι που επιτύχαμε αφού όπως φαίνεται στις ενότητες 5.5 και 5.6 σε ορισμένες περιπτώσεις το ποσοστό αυτής της ελάττωσης φτάνει ως το 79%. Η υλοποίηση έγινε στην γλώσσα προγραμματισμού nesC πάνω στο λειτουργικό σύστημα TinyOS που είναι σχεδιασμένο για τέτοιου είδους δίκτυα ενσωματωμένων συστημάτων.

6.2 Μελλοντική εργασία

Η ενασχόληση μας με την πλατφόρμα POBICOS προέκυψε λόγω της συνεχούς και αυξανόμενης χρήσης των Ασύρματων Δικτύων Αισθητήρα σε εφαρμογές καθημερινής χρήσης που στοχεύουν στον αυτοματισμό σπιτιών και στην μείωση της ενεργειακής τους κατανάλωσης ενώ επιπλέον κίνητρο για την υλοποίηση του πρωτοκόλλου που παρουσιάσαμε απετέλεσε η ανάγκη για την αύξηση της διάρκειας ζωής των δικτύων αυτών, τα οποία από την φύση τους διαθέτουν περιορισμένους πόρους.

Για την περαιτέρω μείωση του φόρτου του δικτύου και άρα την αύξηση της διάρκειας ζωής του, θα μπορούσαν να εφαρμοστούν παρόμοια σχήματα βελτιστοποίησης με αυτό που παρουσιάσαμε στην παρούσα εργασία σε άλλα πρωτόκολλα του ενδιάμεσου λογισμικού POBICOS. Πιο συγκεκριμένα, τα πρωτόκολλα που θα μπορούσαν να υλοποιηθούν πιο αποδοτικά είναι τα εξής:

- Heartbeat Protocol, το οποίο χρησιμοποιείται για την ανανέωση της διάρκειας ζωής ενός πράκτορα-παιδιού καθώς και για τον εντοπισμό των πρακτόρων που έχουν καταστεί απρόσιτοι.
- Agent-level Message Transport Protocol, το οποίο χρησιμοποιείται για την μεταφορά των μηνυμάτων από τους πράκτορες-γονείς στα παιδιά (commands) και την μεταφορά των μηνυμάτων από τους πράκτορες-παιδιά στους γονείς (reports).

Βιβλιογραφία

[1] Introduction to Wireless Sensor Networks- John A. Stankovic

[2] Wireless Sensor Network: Technology Roadmap – Uday B. Desai, B. N. Jain, S. N. Merchant

[3] http://en.wikipedia.org/wiki/Wireless_sensor_network

[4] The nesC Language: A Holistic Approach to Networked Embedded Systems, David Gay, Phil Levis, Rob von Behren, Matt Welsh, Eric Brewer, and David Culler.

[5] NesC 1.2 Language Reference Manual – D.Gay, P. Levis, D.Culler, E.Brewer

[6] http://en.wikipedia.org/wiki/NesC

[7] TinyOS Programming – Philip Levis

[8] http://en.wikipedia.org/wiki/TinyOS

[9] POBICOS Project - http://www.ict-pobicos.eu/summary.htm

[10] http://sourceforge.net/projects/pobicos/

[11] POBICOS project deliverables