



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

**Μελέτη πρωτοκόλλων ανακάλυψης και παράδοσης
υπηρεσίας σε MANETs**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ Χ. ΤΡΑΓΑΝΟΣ

Επιβλέπων : Ιάκωβος Στ. Βενιέρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2007



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

Μελέτη πρωτοκόλλων ανακάλυψης και παράδοσης υπηρεσίας σε MANETs

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ Χ. ΤΡΑΓΑΝΟΣ

Επιβλέπων : Ιάκωβος Στ. Βενιέρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 25^η Οκτωβρίου 2007.

.....
Ι. Βενιέρης
Καθηγητής Ε.Μ.Π.

.....
Δ. Κακλαμάνη
Αν. Καθηγήτρια Ε.Μ.Π.

.....
Ν. Ουζούνoglou
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2007

.....
Κωνσταντίνος Χ. Τραγανός

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

Copyright © Κωνσταντίνος Τραγανός, 2007

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

ΠΕΡΙΛΗΨΗ

Το αντικείμενο της παρούσας διπλωματικής είναι η μελέτη των πρωτοκόλλων ανακάλυψης και παράδοσης υπηρεσίας σε κινητά ad hoc δίκτυα.

Τα ad hoc δίκτυα είναι ασύρματα, ετερογενή, αυτο-οργανωμένα δίκτυα που δε στηρίζονται σε υπάρχουσα δικτυακή υποδομή. Κύριο χαρακτηριστικό τους είναι η κινητικότητα των κόμβων που προκαλεί αλλαγές στη τοπολογία του δικτύου καθώς και οι ενεργειακοί περιορισμοί των κόμβων. Έχουν επαγγελματικές και εμπορικές εφαρμογές (π.χ. enterprise networks), εφαρμογές σε καταστάσεις διαχείρισης κρίσεων καθώς και στρατιωτικές εφαρμογές.

Μία από τις σημαντικότερες λειτουργίες σε αυτά τα δίκτυα είναι η ανακάλυψη υπηρεσιών (service discovery). Οι κόμβοι μπορεί να είναι πάροχοι υπηρεσιών είτε πελάτες (ή ενδιάμεσοι αποθηκευτικοί κόμβοι σε κάποιες εφαρμογές). Οι χρήστες - πελάτες, αναζητούν τις προσφερόμενες υπηρεσίες στο δίκτυο και έχουν αναπτυχθεί διάφορα πρωτόκολλα γι' αυτό το σκοπό. Ένα από αυτά είναι το Service Location Protocol (SLP) που θα μας απασχολήσει κυρίως σε αυτή τη διπλωματική.

Το SLP αναπτύχθηκε για σταθερά IP δίκτυα υπολογιστών, ωστόσο, οι περιορισμοί και τα ιδιαίτερα χαρακτηριστικά των κινητών ad hoc δικτύων επιβάλουν την τροποποίησή του για λειτουργία σε αυτά. Αναλύοντας το κλασσικό SLP και παρουσιάζοντας εν συνεχεία αποτελέσματα προσομοιώσεων από τις κύριες, δύο αρχιτεκτονικές του (κεντρική και κατακευματισμένη), καταλήγουμε στο συμπέρασμα πως είναι αποδοτικότερη η λειτουργία χωρίς την παρουσία ενδιάμεσων κόμβων.

Τέλος, προτείνουμε έναν αλγόριθμο που δίνει τη δυνατότητα στο SLP να λαμβάνει υπόψη του τους ενεργειακούς περιορισμούς που προκύπτουν. Η γνώση της εναπομένουσας διάρκειας ζωής της μπαταρίας κάποιων κόμβων ή της ενέργειας ενός μονοπατιού, οδηγεί σε καλύτερες επιλογές κατά την ανακάλυψη υπηρεσιών στο δίκτυο.

Λέξεις κλειδιά

Κινητά ad hoc δίκτυα, ετερογενή, ανακάλυψη υπηρεσίας, πάροχοι υπηρεσιών, χρήστες υπηρεσιών, ενδιάμεσοι διαμεσολαβητές, πρωτόκολλο SLP, κεντρική λειτουργία πρωτοκόλλου, κατακευματισμένη λειτουργία, ενεργειακοί περιορισμοί, επέκταση πρωτοκόλλου.

ABSTRACT

This diploma thesis studies the service discovery and delivery protocols in Mobile Ad hoc Networks (MANETs).

Ad hoc are wireless, heterogeneous, self-organized networks without a fixed infrastructure. The main attributes of these networks are the nodes' mobility, leading to changes in topology, and the energy constraints. They apply in business and commercial applications (i.e. enterprise networks), crisis management and military applications.

One of the major procedures in MANETs is service discovery. Nodes can either be service agents or user agents (or directory agents in some schemes). The latter, search for available services and many protocols have been developed for service discovery. Service Location Protocol (SLP) is one of these and will be the dominant protocol in this thesis.

SLP was first developed for functioning in fixed, IP networks. However, energy constraints and the other characteristics of ad hoc networks impose the modification of this protocol. Analyzing the original SLP and presenting the results of simulating the two main architectures (Centralized SLP and Distributed SLP), we conclude that the protocol is more efficient to function without directory agents.

Finally, we propose an algorithm which allows SLP to take into account resources' constraints. The a priori knowledge of the remaining battery supplies of some nodes or of the energy resources of a path, points to better selection of services during service discovery procedure in a network.

Key words

Mobile Ad hoc Networks (MANETs), heterogeneous, service discovery, service agents, user agents, directory agents, Service Location Protocol (SLP), centralized SLP (CSLP), distributed SLP (DSLP), energy constraints, SLP extension.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον υπεύθυνο Καθηγητή της διπλωματικής μου, κ. Ι. Βενιέρη, για τη δυνατότητα που μου έδωσε να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα. Ιδιαίτερες ευχαριστίες οφείλω και στον Τομέα Συστημάτων Μετάδοσης Πληροφορίας & Τεχνολογίας Υλικών που ενέκρινε τη διπλωματική αυτή. Επίσης, θα ήθελα να ευχαριστήσω θερμά, τον επιβλέποντα αυτής της διπλωματικής και υποψήφιο Διδάκτορα, Χ. Κατσιγιάννη, για την υποστήριξη και την καθοδήγηση που μου παρείχε και συνέβαλε τα μέγιστα για την ολοκλήρωσή της.

Κωνσταντίνος Χ. Τραγανός,
Αθήνα, 25 Οκτωβρίου 2007.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1	15
1 ΓΕΝΙΚΑ ΓΙΑ ΤΑ AD HOC	15
1.1 ΕΙΣΑΓΩΓΗ	15
1.2 ΤΙ ΕΙΝΑΙ ΤΑ AD HOC ΔΙΚΤΥΑ	15
1.3 ΙΔΙΑΙΤΕΡΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΚΑΙ ΠΕΡΙΟΡΙΣΜΟΙ ΤΩΝ AD-HOC ΔΙΚΤΥΩΝ	16
1.4 ΕΦΑΡΜΟΓΕΣ ΤΩΝ MANETs	18
1.5 ΣΤΟΧΟΣ ΚΑΙ ΟΡΓΑΝΩΣΗ ΤΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ	20
ΚΕΦΑΛΑΙΟ 2	23
2 SERVICE DISCOVERY ΣΕ IP ΔΙΚΤΥΑ	23
2.1 ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΝΑΚΑΛΥΨΗ ΥΠΗΡΕΣΙΑΣ	23
2.2 ΑΝΑΚΑΛΥΨΗ ΥΠΗΡΕΣΙΑΣ ΣΕ IP ΔΙΚΤΥΑ	24
2.2.1 SLP	24
2.2.2 UPnP	25
2.2.3 Jini	25
2.2.4 UDDI	26
2.2.5 Salutation	27
ΚΕΦΑΛΑΙΟ 3	29
3 SDP ΣΕ AD HOC ΔΙΚΤΥΑ	29
3.1 ΑΝΑΓΚΗ ΝΕΩΝ ΛΥΣΕΩΝ	29
3.2 Bluetooth SDP	29
3.3 Konark	31
3.4 GSD	34
3.5 DEAPspace	37
ΚΕΦΑΛΑΙΟ 4	39
4 SERVICE LOCATION PROTOCOL (SLP)	39
4.1 ΕΙΣΑΓΩΓΗ	39
4.2 SLP	41
4.2.1 Ορολογία	41
4.2.2 Συνοπτική παρουσίαση του πρωτοκόλλου	42
4.2.3 Τα URLs που χρησιμοποιούνται στο Service Location	44
4.2.4 Χρήση των Ports, του UDP, και της Multicast εκπομπής	45
4.2.5 Επαναμετάδοση SLP μηνυμάτων	46
4.2.6 SLP Μηνύματα	46
4.2.7 Service Location Protocol Extensions	51
4.3 SLP ΣΕ AD HOC ΔΙΚΤΥΑ	52
4.4 ΕΠΙΛΟΓΗ ΛΕΙΤΟΥΡΓΙΑΣ SLP ΓΙΑ AD HOC ΔΙΚΤΥΑ	54
ΚΕΦΑΛΑΙΟ 5	55
5 ΣΥΓΚΡΙΣΗ CSLP ΚΑΙ DSLP	55
5.1 ΕΙΣΑΓΩΓΗ	55
5.2 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΓΙΑ ΑΝΑΚΑΛΥΨΗ ΥΠΗΡΕΣΙΑΣ ΣΕ MANET	55
5.2.1 Centralized Service Location Scheme (CSLP)	55
5.2.2 Distributed Service Location Scheme (DSLP)	57
5.3 ΑΠΟΤΙΜΗΣΗ ΑΠΟΔΟΣΗΣ ΠΡΩΤΟΚΟΛΛΟΥ	58

5.3.1	Προσομοίωση με χρήση του NS-2	58
5.3.2	Περιβάλλον προσομοίωσης	59
5.3.3	Αποτελέσματα προσομοιώσεων	60
5.4	ΣΥΜΠΕΡΑΣΜΑΤΑ - ΑΝΑΓΚΗ ΤΡΟΠΟΠΟΙΗΣΕΩΝ ΣΤΟ SLP.....	63
ΚΕΦΑΛΑΙΟ 6		65
6	ΕΠΕΚΤΑΣΗ ΤΟΥ SERVICE LOCATION PROTOCOL.....	65
6.1	ΕΙΣΑΓΩΓΗ	65
6.2	ΠΡΟΤΕΙΝΟΜΕΝΟΣ ΑΛΓΟΡΙΘΜΟΣ	66
6.3	ΣΥΜΠΕΡΑΣΜΑ.....	69
ΚΕΦΑΛΑΙΟ 7		71
7	ΣΥΜΠΕΡΑΣΜΑΤΑ	71
7.1	ΣΥΝΕΙΣΦΟΡΑ	71
7.2	ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ	71
8	ΒΙΒΛΙΟΓΡΑΦΙΑ	75
ΠΑΡΑΡΤΗΜΑ Α		79
A.1	Προσθήκη αρχείων για υποστήριξη του SLP	79
A.2	Τροποποίηση αρχείων για υποστήριξη του SLP	108
ΠΑΡΑΡΤΗΜΑ Β.....		115
B.1	Περιγραφή – Παράμετροι	115
B.2	Κώδικες σεναρίων	117

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

Σχήμα 1.1	Διάφορες μορφές σύγχρονων δικτύων	16
Σχήμα 1.2	Αλλαγή τοπολογίας δικτύου.....	17
Σχήμα 1.3	Προώθηση πακέτων	18
Σχήμα 2.1	Διάγραμμα ροής του τρόπου λειτουργίας της τεχνολογίας Jini	26
Σχήμα 3.1	Konark service discovery	32
Σχήμα 3.2	Παράδειγμα Δένδρου Υπηρεσιών	33
Σχήμα 3.3	Στοιχεία του service delivery	34
Σχήμα 4.1	Κεντρική λειτουργία δικτύου	40
Σχήμα 4.2	Κατανεμημένη λειτουργία δικτύου	40
Σχήμα 4.3	Εκπομπή μηνυμάτων SrvRqst και SrvRply.....	42
Σχήμα 4.4	Αποστολή μηνυμάτων μέσω Directory Agents.....	43
Σχήμα 4.5	Ανακάλυψη των Directory Agents	43
Σχήμα 4.6	Ανταλλαγή μηνυμάτων μέσω scopes	44
Σχήμα 4.7	URL Entries.....	45
Σχήμα 4.8	Επικεφαλίδα SLP μηνυμάτων	47
Σχήμα 4.9	SLP Μηνύματα σε IP δίκτυα.....	47
Σχήμα 4.10	Service Request	48
Σχήμα 4.11	Service Reply.....	49
Σχήμα 4.12	Directory Agent Advertisement	50
Σχήμα 4.13	Service Agent Advertisement.....	51
Σχήμα 4.14	Service Location Extension.....	52
Σχήμα 4.15	SLP Μηνύματα σε ad hoc δίκτυα.....	54
Σχήμα 5.1	Διαδικασία ανακάλυψης υπηρεσίας στο CSLP	56
Σχήμα 5.2	Διαδικασία ανακάλυψης υπηρεσίας στο DSLP	58
Σχήμα 5.3	Service hit ratio.....	60
Σχήμα 5.4	Αριθμός μηνυμάτων ελέγχου (control messages)	61
Σχήμα 5.5	Ανάλυση μηνυμάτων ελέγχου στο CSLP.....	62
Σχήμα 5.6	Χρόνος ανακάλυψης των SAs.....	62
Σχήμα 6.1	Παράδειγμα συνοχής δικτύου	65
Σχήμα 6.2	Δομή SLP μηνύματος.....	66
Σχήμα 6.3	Βήματα αλγορίθμου	67

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

Πίνακας 3.1 Peer-to-peer Caching.....	36
Πίνακας A.1.1 udp-slp.h.....	79
Πίνακας A.1.2 udp-slp.cc.....	82
Πίνακας A.1.3 slp-sa.h.....	84
Πίνακας A.1.4 slp-sa.cc.....	91
Πίνακας A.1.5 slp-ua.h.....	94
Πίνακας A.1.6 slp-ua.cc.....	104
Πίνακας A.1.7 slp-common.h.....	107
Πίνακας A.2.1 Αρχεία προς τροποποίηση.....	108
Πίνακας A.2.2 packet.h.....	109
Πίνακας A.2.3 agent.h.....	109
Πίνακας A.2.4 app.h.....	109
Πίνακας A.2.5 ns-packet.tcl.....	110
Πίνακας A.2.6 ns-default.tcl.....	110
Πίνακας A.2.7 cmu-trace.h.....	110
Πίνακας A.2.8 cmu-trace.cc.....	113
Πίνακας B.1.1 Χαρακτηριστικά κίνησης σεναρίων.....	117
Πίνακας B.1.2 Δικτυακοί παράμετροι σεναρίων.....	117
Πίνακας B.2.1 Random Waypoint mobility model (mobgen.c).....	121
Πίνακας B.2.2 original_common.tcl.....	125
Πίνακας B.2.3 conf.tcl.....	126
Πίνακας B.2.4 vehnet.tcl.....	126
Πίνακας B.2.5 mesht.tcl.....	127

ΚΕΦΑΛΑΙΟ 1

1 ΓΕΝΙΚΑ ΓΙΑ ΤΑ AD HOC

1.1 ΕΙΣΑΓΩΓΗ

Οι τελευταίες τεχνολογικές εξελίξεις και το μειούμενο κόστος των τεχνολογιών δικτύου έχουν οδηγήσει σε μια ραγδαία αύξηση καταναμημένων συστημάτων και στην ανάπτυξη χιλιάδων συσκευών δικτύου που χρησιμοποιούνται τόσο σε ενσύρματα όσο και σε ασύρματα δίκτυα. Έχουν ήδη σχεδιαστεί και υλοποιηθεί εφαρμογές, στις οποίες ο κλασσικός υπολογιστής δεν υφίσταται πλέον ως μια αυτόνομη, συγκροτημένη συσκευή πολλαπλών χρήσεων, αλλά ως ένα καταναμημένο σύστημα από μη-ορατά υπολογιστικά τμήματα. Τα τμήματα αυτά συνήθως έχουν πολύ μικρό μέγεθος και είναι ενσωματωμένα σε άλλα αντικείμενα, μη-υπολογιστικής (και συνήθως καθημερινής) χρήσης.

Τα συστήματα αυτά καλούνται συνήθως «δισεδυτικά υπολογιστικά συστήματα» (pervasive computer systems), αν και έχουν κατά καιρούς χρησιμοποιηθεί και άλλοι όροι, όπως ubiquitous computing, disappearing computers κλπ. Η βασική ιδέα πίσω από όλες αυτές τις προσεγγίσεις είναι ότι ο υπολογιστής «εξαφανίζεται» και οι υπηρεσίες που απαιτούν υπολογιστική ισχύ διατίθενται στους ανθρώπους δια μέσου του φυσικού τους περιβάλλοντος.

Προς τη κατεύθυνση αυτή και με σκοπό την καλυτέρευση της ποιότητας της ζωής του ανθρώπου, γίνεται προσπάθεια ανάπτυξης διάφορων μορφών δικτύων. Για παράδειγμα, η δημιουργία ασύρματων δικτύων που βασίζονται στην υπάρχουσα δικτυακή υποδομή, όπως το ευρέως διαδεδομένο κυψελωτό σύστημα (cellular).

Υπάρχουν όμως και τα ασύρματα δίκτυα που δεν εκμεταλλεύονται την υποδομή αυτή. Τέτοια δίκτυα είναι για παράδειγμα τα ad hoc που διαδραματίζουν θεμελιώδη ρόλο στον τομέα των «pervasive computer systems».

Σε αυτή τη διπλωματική θα ασχοληθούμε κατά κύριο λόγο με τη λειτουργία ανακάλυψης υπηρεσιών (service discovery) στα ασύρματα ad hoc δίκτυα μελετώντας τα υπάρχοντα πρωτόκολλα. Κυρίως θα εστιάσουμε στο Service Location Protocol και θα παρουσιάσουμε αποτελέσματα δύο κύριων αρχιτεκτονικών του. Επιπλέον, στα πλαίσια της μελέτης μας θα αναλύσουμε έναν καινούριο αλγόριθμο ανακάλυψης υπηρεσίας σε MANET, ο οποίος συσχετίζει τη διαδικασία του service discovery με τους περιορισμούς των ασυρμάτων ad hoc δικτύων. Επίσης θα παρουσιαστούν τα βασικά κριτήρια ανακάλυψης υπηρεσίας και επιλογής του πιο αποδοτικού παρόχου υπηρεσίας μέσα από μια σειρά κριτηρίων βελτιστοποίησης του εν λόγω αλγορίθμου.

1.2 ΤΙ ΕΙΝΑΙ ΤΑ AD HOC ΔΙΚΤΥΑ

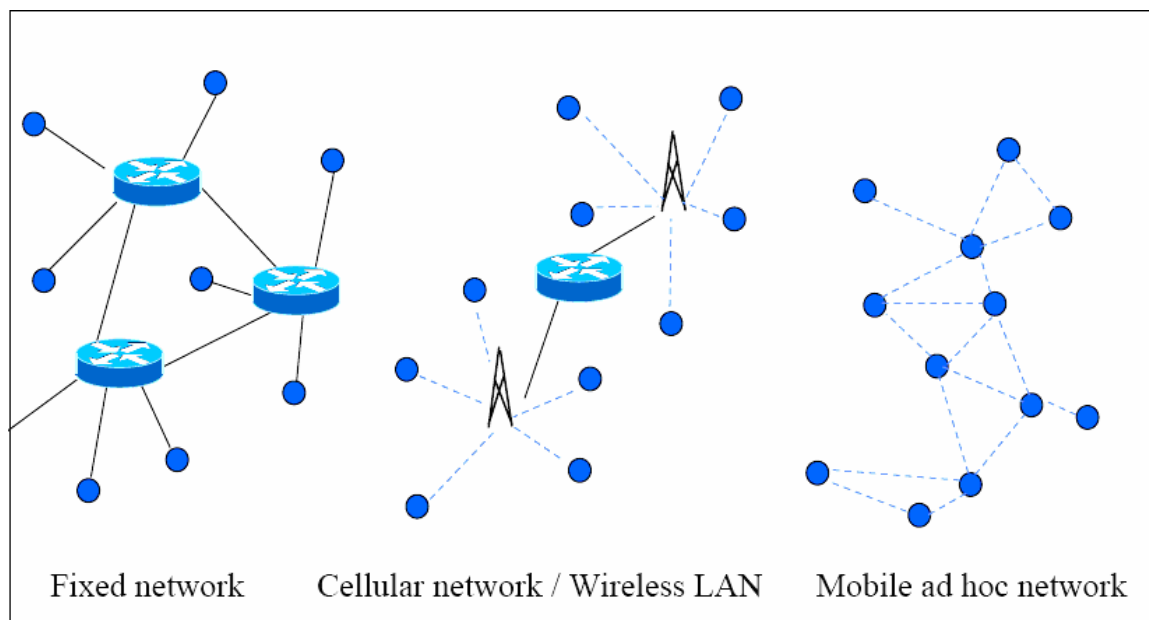
Ετυμολογικά, ο όρος «ad hoc» στα λατινικά σημαίνει «για αυτό το λόγο». Ο χαρακτηρισμός «ad hoc» για τα ασύρματα κινητά δίκτυα χρησιμοποιείται για να περιγράψει τον «αυθόρμητο» (spontaneous) τρόπο με τον οποίο δημιουργούνται, μετακινούν πληροφορία μεταξύ των κόμβων και τέλος χάνονται.

Τα MANETs (Mobile Ad Hoc Networks), όπως αλλιώς λέγονται, αποτελούνται συνεπώς από αυτόνομους κόμβους που αυτο-οργανώνουν ένα δίκτυο το οποίο και μάλιστα αυτο-διαχειρίζονται χωρίς να χρησιμοποιούν υπάρχουσα δικτυακή υποδομή. Τέτοιοι κόμβοι μπορεί να είναι συσκευές επικοινωνιών όπως laptops, PDAs (personal digital assistants), κινητά τηλέφωνα αλλά ακόμα και αισθητήρες.

Επίσης τα ad-hoc είναι multihop ασύρματα δίκτυα όπου όλοι οι κόμβοι βοηθούν για τη διατήρηση της διασυνδεσιμότητας του δικτύου. Δεν υπάρχει αρχηγός, κεντρικός διαχειριστής, ή συντονιστής του δικτύου ή της κυκλοφορίας. Κάθε σταθμός μπορεί να στείλει πληροφορία σε οποιοδήποτε άλλον σταθμό άμεσα, είτε ο παραλήπτης βρίσκεται εντός της εμβέλειας του είτε βρίσκεται εκτός, οπότε θα πρέπει να βρεθεί κατάλληλο και αποδοτικό μονοπάτι που θα αποτελείται από τους κόμβους από τους οποίους θα περάσει το μήνυμα προκειμένου να φτάσει στον παραλήπτη. Επομένως, όλοι οι σταθμοί είναι ισοδύναμοι, έχουν τα ίδια χαρακτηριστικά δικτύωσης και επιτελούν τις ίδιες λειτουργίες όταν πρόκειται για αποστολές, λήψεις ή προωθήσεις πακέτων πληροφορίας.

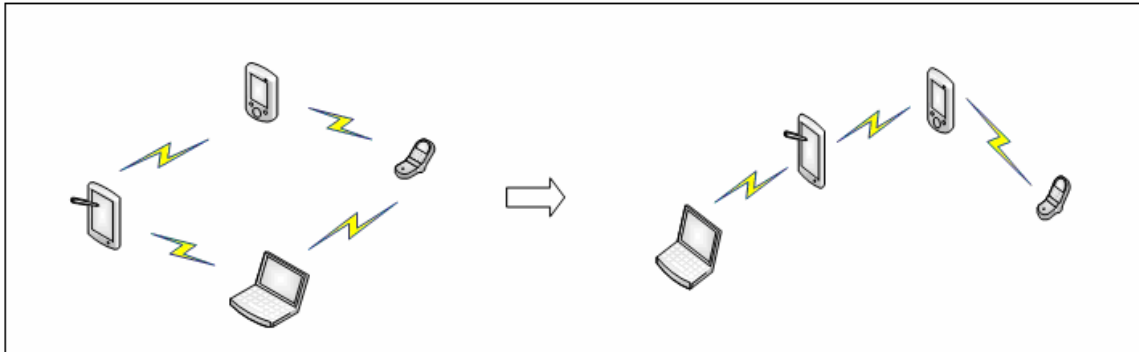
1.3 ΙΔΙΑΙΤΕΡΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΚΑΙ ΠΕΡΙΟΡΙΣΜΟΙ ΤΩΝ AD-HOC ΔΙΚΤΥΩΝ

Το κύριο χαρακτηριστικό των ad hoc δικτύων αλλά και ταυτόχρονα η ειδοποιός διαφορά από τα υπόλοιπα ασύρματα δίκτυα είναι πως δε στηρίζονται σε υπάρχουσα δικτυακή υποδομή. Τόσο σε θέματα δρομολόγησης, όσο και σε θέματα διαχείρισης, παρουσιάζεται μια αυτό-οργάνωση και αυτό-διαχείριση του δικτύου από τους ίδιους τους κόμβους του. Στο Σχήμα 1.1 βλέπουμε τις διάφορες μορφές δικτύων.



Σχήμα 1.1: Διάφορες μορφές σύγχρονων δικτύων

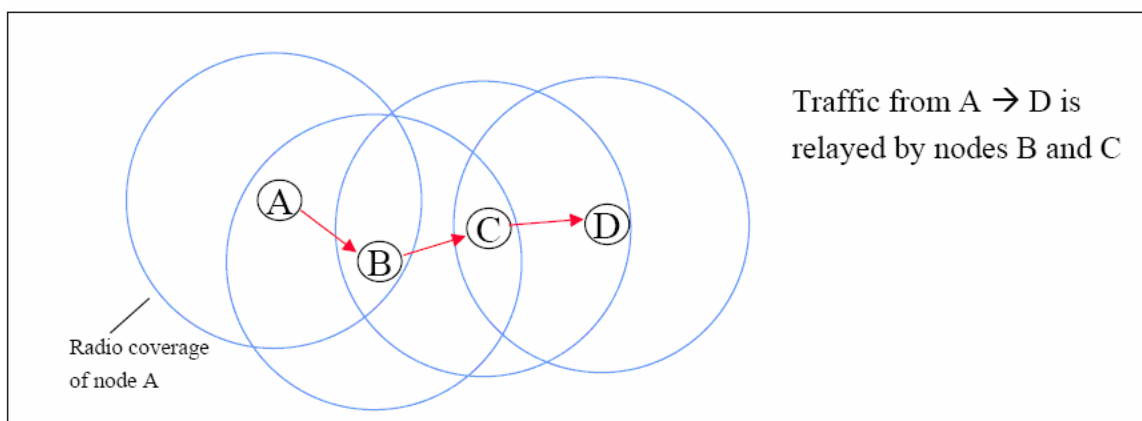
Άλλο χαρακτηριστικό των ad hoc δικτύων είναι η κινητικότητα (mobility) των κόμβων που προκαλεί δυναμική αλλαγή της τοπολογίας του δικτύου. Αλλαγές στην τοπολογία του δικτύου προκαλούνται και από τη μικρή διάρκεια της συμμετοχής στο δίκτυο για κάποιες συσκευές (κόμβοι), οι οποίες συμμετέχουν μόνο κατά τη διάρκεια της συνόδου τους. Σε αυτό το σημείο αξίζει να αναφερθεί ότι η δυναμική εισαγωγή και αποχώρηση κόμβων από το δίκτυο προκαλεί παράλληλα προβλήματα σε δρομολόγηση καθώς και διατήρησης συνεκτικότητας του δικτύου. Σχηματικά ο τρόπος με τον οποίο αλλάζει η τοπολογία ενός ad-hoc δικτύου περιγράφεται παρακάτω (Σχήμα 1.2):



Σχήμα 1.2: Αλλαγή τοπολογίας δικτύου

Επιπλέον περιορισμό αποτελούν οι ξεύξεις μικρού εύρους ζώνης και η έλλειψη κατευθυντικότητας των κεραιών των τερματικών. Με αυτό τον τρόπο δεν επιτυγχάνεται πάντα η διατήρηση συνδεσιμότητας μεταξύ 2 κόμβων, γεγονός που οδηγεί σε αλλαγή τοπολογίας ή σε μη αξιόπιστη μετάδοση δεδομένων.

Εφόσον δεν υπάρχει δικτυακή υποδομή, κάθε κόμβος λειτουργεί τόσο ως δρομολογητής (router) όσο και ως host. Αυτό σημαίνει ότι αν ένας κόμβος θελήσει να συνδεθεί με κάποια συσκευή σε μεγάλη απόσταση όπου η σύνδεση δεν είναι ενός βήματος (single-hop), τότε αναγκαστικά οι δύο κόμβοι συνδέονται μέσω διαδρομής πολλαπλών βημάτων (multi-hopping). Έτσι κάθε κόμβος συμμετέχει στη διαδικασία της δρομολόγησης, προωθώντας πακέτα προς τον κόμβο-προορισμό, όπως παρουσιάζεται και στο ακόλουθο Σχήμα 1.3.



Σχήμα 1.3: Multi-hopping προώθηση πακέτων

Ένα επιπλέον χαρακτηριστικό των MANETs είναι οι ενεργειακοί περιορισμοί που παρουσιάζουν οι κόμβοι του. Οι κινητοί κόμβοι των MANETs δεν τροφοδοτούνται με ενέργεια από γραμμές τροφοδοσίας (line-powered) και συνεπώς τα παρουσιάζόμενα ενεργειακά προβλήματα οφείλονται στην περιορισμένη ισχύ των μπαταριών που αποτελούν τη βασική πηγή ενέργειας των κόμβων. Η κατανάλωση ενέργειας σχετίζεται και με την ακτίνα μετάδοσης δεδομένων μεταξύ των συσκευών, καθώς και με τις εσωτερικές διαδικασίες λήψης, αποστολής και αποκωδικοποίησης μηνυμάτων που λαμβάνουν χώρα σε κάθε κόμβο. Σκοπός, τόσο των αλγορίθμων που χρησιμοποιούνται σε κάθε επίπεδο (δρομολόγηση, ανακάλυψη υπηρεσιών κ.τ.λ.), όσο και των διαδικασιών αποστολής και λήψης πληροφοριών, είναι η ελαχιστοποίηση της ενέργειας που δαπανάται.

Τελειώνοντας την αναφορά μας στις ιδιότητες των MANETs, θα πρέπει να αναφέρουμε ότι τα ασύρματα ad hoc δίκτυα χαρακτηρίζονται και σαν ετερογενή δίκτυα. Ουσιαστικά παρουσιάζουν δυνατότητα προσαρμογής και δημιουργίας δικτύου μεταξύ συσκευών που ανήκουν σε διαφορετικές μορφές δικτύου. Τα ad hoc παρουσιάζουν την ικανότητα να συνδέονται σε δίκτυα που έχουν υποδομή, δημιουργώντας υβριδικά δίκτυα, γεγονός που αποδεικνύει την ανεξαρτησία τους σε σχέση με το υπερκείμενο δίκτυο πρόσβασης (Access Network) της κάθε συσκευής. Έτσι τα ad hoc εμφανίζονται τόσο σε μια μορφή που χαρακτηρίζονται ως απλά ad hoc (pure ad hoc) όπου όλοι οι κόμβοι είναι χρήστες ή πελάτες, όσο και σε μια υβριδική μορφή όπου πέραν των κόμβων χρηστών υπάρχουν και οι κόμβοι υποδομής. Οι τελευταίοι, στο υβριδικό δίκτυο, αναλαμβάνουν εν μέρει τη δρομολόγηση προς τους κόμβους πελάτες, λειτουργία που σε ένα pure ad hoc έχουν αποκλειστικά οι κόμβοι χρήστες. Τα πιο συνηθισμένα και πιο χρησιμοποιούμενα στις μέρες ad hoc δίκτυα είναι τα υβριδικά.

1.4 ΕΦΑΡΜΟΓΕΣ ΤΩΝ MANETs

Οι εφαρμογές των ad hoc δικτύων μπορούν να κατηγοριοποιηθούν ως εξής:

1. Επαγγελματικές και εμπορικές εφαρμογές

(α) Καταστάσεις χαμηλής κινητικότητας

Πρόκειται για τα λεγόμενα enterprise networks. Αυτά είναι δίκτυα τα οποία εγκαθίστανται μέσα στα κτίρια εταιρειών που συνεργάζονται. Προτιμούνται λόγω του χαμηλού τους κόστους και της ευκολίας εγκατάστασής τους, εξυπηρετώντας ταυτόχρονα την κινητικότητα των εργαζομένων, σε μεγαλύτερο βαθμό από τις υπόλοιπες μορφές δικτύων. Χρησιμοποιούνται σε περιπτώσεις επαγγελματικών συναντήσεων (meetings) ή κάποιας συνέντευξης μέσω δικτύου. Παρέχουν τη δυνατότητα στον ομιλητή να μεταδώσει προς πολλές πηγές τις παρουσιάσεις του, περιγράφοντας μέσω του video conference στους συμμετέχοντες τις ιδέες του και δίνοντας τη δυνατότητα σε αυτούς να ρωτούν.

(β) Καταστάσεις υψηλής κινητικότητας

Χρησιμοποιούνται στις περιπτώσεις κάλυψης γεγονότων όπου τηλεοπτικά μέσα και κοινό παρακολουθούν κάποιον ομιλητή.

2. Εφαρμογές διαχείρισης κρίσεων

(α) Σχεδόν στατικοί στόχοι

Βρίσκουν εφαρμογή σε περιπτώσεις διάσωσης αποκλεισμένου ατόμου ή σε παροχή βοήθειας.

(β) Κινούμενοι στόχοι

Πρόκειται για τα δίκτυα αντιμετώπισης έκτακτων περιστάσεων (emergency response networks). Σε πολύ σύντομο χρονικό διάστημα και χωρίς την ανάγκη ύπαρξης υποδομής, σχηματίζεται δίκτυο προκειμένου να υπάρχει συνεχής επικοινωνία και ενημέρωση ανάμεσα στις δυνάμεις άμεσης επέμβασης όπως αστυνομίας ή πυροσβεστικής. Με χρήση τέτοιου δικτύου συντονίζονται οι μονάδες που συμμετέχουν σε αυτές τις δυνάμεις και αποκαθίσταται η επικοινωνία με τα κέντρα π.χ. αστυνομικά τμήματα προκειμένου να γίνει καλύτερος συντονισμός των ομάδων και άμεση επέμβαση.

3. Προσωπικές εφαρμογές – home and mesh networks

Με τα προσωπικά δίκτυα σπιτιού (home networks) παρέχεται η δυνατότητα στο χρήστη να έχει πρόσβαση στο διαδίκτυο από περισσότερες της μιας θέσης από το σπίτι του. Το σημαντικότερο όμως είναι ότι ο χρήστης μπορεί να δημιουργήσει δίκτυο στο σπίτι του χρησιμοποιώντας διαφορετικές συσκευές όπως έναν φορητό υπολογιστή (laptop), ένα PDA, ένα κινητό τηλέφωνο ή και άλλες συσκευές. Με αυτό τον τρόπο βελτιώνεται η ποιότητα της ζωής του χρήστη, αφού αυτές οι συσκευές επικοινωνούν μεταξύ τους, ενημερώνοντας το χρήστη για τυχόν ελλείψεις. Επιπλέον, ο χρήστης στα πλαίσια ενός «ψηφιακού σπιτιού» μπορεί να ελέγχει τις ηλεκτρικές συσκευές στο σπίτι του.

Επίσης, χωρίς ανάγκη καλωδίωσης, αναπτύσσονται τα mesh networks όπου και ολόκληρη κοινότητα μπορεί να επικοινωνεί ταχύτερα και ευκολότερα. Επιπλέον με τη χρήση των mesh δικτύων δημιουργείται ένα νέο επίπεδο συνεργασίας μεταξύ των κοινωνιών και της ανθρώπινης επικοινωνίας.

4. Δίκτυα αυτοκινήτων

Αυτά δημιουργούνται ανάμεσα σε αυτοκίνητα που κινούνται και σε δίκτυα σταθερής υποδομής που καθορίζουν την κυκλοφορία, όπως παραδείγματος χάριν τα φανάρια ή τα ηλεκτρονικά σήματα. Έτσι, μπορεί ο χρήστης-οδηγός να αποφύγει δρόμους όπου υπάρχει κυκλοφοριακή συμφόρηση ή ακόμη να ενημερωθεί και για υπηρεσίες όπως για βενζινάδικα ή συνεργεία που βρίσκονται σε κοντινή απόσταση. Επιπλέον, ένα συγκεκριμένο «office-car»

μπορεί να περιέχει fax, εκτυπωτές, τηλέφωνο, όπου χρήστες - περαστικοί μπορούν να συνδεθούν και να τα χρησιμοποιήσουν μέσω των laptop τους και των PDA.

5. Στρατιωτικές εφαρμογές

Τα MANETs χρησιμοποιούνται ευρέως και για στρατιωτικές εφαρμογές, όπως σε περιπτώσεις μάχης, όπου η ασύρματη επικοινωνία μεταξύ των στρατιωτικών μονάδων είναι απαραίτητη. Πιο συγκεκριμένα, ιδιαίτερης σημασίας είναι η ασύρματη επικοινωνία μεταξύ στρατιωτών, ελικοπτέρων, στρατιωτικών οχημάτων και σταθμών βάσης, ώστε να ενημερώνεται διαρκώς η βάση για τη θέση και την κίνηση των στρατιωτικών μονάδων σε εχθρικά πεδία μάχης.

6. Δίκτυα Αισθητήρων - Sensor networks

Ένα από τα πιο σημαντικά πεδία χρήσης των ad hoc δικτύων, είναι η δυνατότητα δημιουργίας sensor networks. Τέτοια δίκτυα μπορούν να χρησιμοποιηθούν για στρατιωτικούς λόγους όπως για ανίχνευση εχθρικής κίνησης και προειδοποίηση για καταπάτηση στρατοπέδων, αλλά και για προστασία και έγκαιρη ειδοποίηση σε περίπτωση χρήσης χημικών ή βιολογικών όπλων. Όμως πέραν των στρατιωτικών χρήσεων τα sensor networks χρησιμοποιούνται και για την προστασία του περιβάλλοντος, όπως π.χ. αποφυγή πυρκαγιών, αλλά και για την αντιμετώπιση της κυκλοφοριακής συμφόρησης σε μια πόλη αφού μπορεί να ανιχνευθεί και να γίνει σωστή εποπτεία και παρακολούθηση των δρόμων, πράγμα που οδηγεί και σε περιορισμό παραβάσεων. Επίσης μπορεί να εφαρμοστεί σε αθλητικές δραστηριότητες, όπως μέτρηση επιδόσεων αθλητών με συσκευές που ενσωματώνονται στο ανθρώπινο σώμα (BAN – Body Area Networks).

Τα τελευταία χρόνια με τη συνεχώς αυξανόμενη εμφάνιση ασθενειών που σχετίζονται με απώλεια μνήμης ή και κινητικά προβλήματα, όπου οι ασθενείς χρήζουν συνεχούς παρακολούθησης τα sensor networks έχουν διευκολύνει την ιατρική κοινότητα. Τα χαρακτηριζόμενα «έξυπνα περιβάλλοντα» (smart environments), τα οποία βασίζονται στη δημιουργία sensor δικτύων στο σπίτι ενός ασθενούς, διευκολύνουν τις ιατρικές μονάδες στη διαρκή παρακολούθηση ασθενών.

1.5 ΣΤΟΧΟΣ ΚΑΙ ΟΡΓΑΝΩΣΗ ΤΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ

Αντικείμενο μελέτης αναφορικά με τα ασύρματα ad hoc δίκτυα, είναι ο τρόπος με τον οποίο οι κόμβοι ανακαλύπτουν πόρους και υπηρεσίες που υπάρχουν σε άλλες συσκευές του δικτύου. Σε αυτό το πλαίσιο, η παρούσα διπλωματική εργασία, θα παρουσιάσει τα υπάρχοντα πρωτόκολλα ανακάλυψης υπηρεσίας (Service Discovery Protocols) τόσο σε σταθερά IP δίκτυα όσο και σε ασύρματα ad hoc δίκτυα.

Ένα από αυτά τα πρωτόκολλα που χρησιμοποιείται ευρέως είναι και το SLP (Service Location Protocol). Αφού παρουσιαστεί η γενικότερη λειτουργία του καθώς και η εφαρμογή του στα MANETs, θα συγκριθούν οι δύο διαφορετικές μορφές του, κεντριοποιημένη και κατανεμημένη. Εν συνεχεία, θα παρουσιαστεί μια επέκταση του SLP όπου η ανακάλυψη υπηρεσίας θα σχετίζεται και με τους ενεργειακούς περιορισμούς που τίθενται από τα MANETs.

Η διπλωματική είναι οργανωμένη ως εξής: Στο δεύτερο κεφάλαιο εισάγεται η έννοια του service discovery στα IP δίκτυα όπου παρουσιάζονται τα πιο χαρακτηριστικά και ευρέως διαδεδομένα πρωτόκολλα ανακάλυψης και παράδοσης υπηρεσιών. Εν συνεχεία, αναλύονται οι αλγόριθμοι ανακάλυψης και παράδοσης υπηρεσίας που έχουν ήδη εφαρμοστεί στα κινητά ad hoc δίκτυα. Στο τέταρτο κεφάλαιο μελετάται η λειτουργία του SLP, ενώ στο επόμενο κεφάλαιο παρουσιάζονται και συγκρίνονται τα αποτελέσματα από την υλοποίηση των δύο διαφορετικών αρχιτεκτονικών του SLP. Στο έκτο κεφάλαιο, προτείνουμε μια αρχιτεκτονική που λαμβάνει υπόψη της τους ενεργειακούς πόρους κάθε κόμβου ώστε να ανακαλυφθεί το βέλτιστο ενεργειακά μονοπάτι για παράδοση υπηρεσίας και στο τελευταίο κεφάλαιο της εργασίας αναλύονται τα συμπεράσματα της παρούσας διπλωματικής και δίνονται κατευθύνσεις για μελλοντική έρευνα.

ΚΕΦΑΛΑΙΟ 2

2 SERVICE DISCOVERY ΣΕ IP ΔΙΚΤΥΑ

2.1 ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΝΑΚΑΛΥΨΗ ΥΠΗΡΕΣΙΑΣ

Ως *υπηρεσία (service)* θεωρούμε κάθε πόρο (είτε υλικό - hardware είτε λογισμικό - software) που θεωρείται προσπελάσιμος μέσω δικτύου και είναι ικανός να αποστείλει ή να παραλάβει δεδομένα. Αναφερόμαστε στους κόμβους που «φιλοξενούν» υπηρεσίες ως «*πάροχοι υπηρεσιών*» (service providers ή απλά servers) και στους κόμβους που «τρέχουν» εφαρμογές και υπηρεσίες ως «*πελάτες*» ή χρήστες (clients).

Η ανακάλυψη υπηρεσίας (service discovery) είναι η διαδικασία εύρεσης μιας υπηρεσίας που να ικανοποιεί κάποιες επιθυμητές ιδιότητες και η αντιστοίχισή της με την τοποθεσία της υπηρεσίας (service location). Η τοποθεσία αποτελείται συνήθως από μια διεύθυνση και από τη θύρα όπου διοχετεύονται οι εισερχόμενες αιτήσεις (πχ. 174.15.2.1:80). Επειδή μπορεί να υπάρχουν πολλαπλά αντίγραφα της ίδιας υπηρεσίας στο δίκτυο και καθώς μια περιγραφή υπηρεσίας μπορεί να ταιριάζει σε παρόμοιες υπηρεσίες, η διαδικασία ανακάλυψης υπηρεσίας οδηγεί σε πολλαπλές αντιστοιχίσεις. Ένας πελάτης επιλέγει ένα συγκεκριμένο πάροχο μεταξύ των πολλών διαθέσιμων αντιστοιχίσεων με διάφορα κριτήρια. Τα πιο χαρακτηριστικά κριτήρια είναι οι συγκεκριμένες απαιτήσεις της υπηρεσίας (π.χ. ο λιγότερο «φορτωμένος» εκτυπωτής) ή οι παράμετροι του δικτύου (π.χ. εύρος ζώνης, βήματα μεταξύ των κόμβων).

Για να γίνει πιο κατανοητή η έννοια της ανακάλυψης υπηρεσίας, παραθέτουμε ένα παράδειγμα από την καθημερινότητα. Ας υποθέσουμε ότι αναζητούμε τον τηλεφωνικό αριθμό ενός προσώπου, τότε μπορούμε να τον αναζητήσουμε στην ατζέντα μας και αν υπάρχει μπορούμε να τον καλέσουμε. Σε περίπτωση όμως που ο τηλεφωνικός αριθμός δε βρίσκεται στην ατζέντα μας, μπορούμε να τον ψάξουμε στον τηλεφωνικό κατάλογο, να τον αναζητήσουμε σε μια σχετική ιστοσελίδα του διαδικτύου, να καλέσουμε τις πληροφορίες καταλόγου ή να ρωτήσουμε φίλους ή συγγενείς του. Στην πρώτη περίπτωση έχουμε αναζήτηση της υπηρεσίας σε μια τοποθεσία (ατζέντα) και παράδοση της υπηρεσίας από έναν πάροχο, ενώ στην δεύτερη περίπτωση έχουμε πολλαπλούς παρόχους και τοποθεσίες για την εξυπηρέτηση του χρήστη.

Ωστόσο, στα τηλεπικοινωνιακά συστήματα δεν υπάρχει πάντα αφθονία στην εξεύρεση διαθέσιμων παρόχων όπως φαίνεται και στη σχετική βιβλιογραφία ([1]), ή ακόμα και όταν υπάρχει θα πρέπει να επιλεγεί ο κατάλληλος πάροχος. Έτσι, στην περίπτωση τηλεπικοινωνιακών υπηρεσιών όπου πολλαπλές αντιστοιχίσεις της ίδιας υπηρεσίας παρέχονται από διαφορετικούς εξυπηρετητές (π.χ. υπάρχει η ίδια πληροφορία σε διαφορετικές τοποθεσίες), ο χρήστης πρέπει να επιλέξει έναν συγκεκριμένο πάροχο της υπηρεσίας βάσει διάφορων κριτηρίων. Οι παράγοντες που επηρεάζουν την επιλογή του πελάτη, μπορεί να είναι είτε παράμετροι δικτύου, είτε ο αριθμός των βημάτων από κόμβο σε κόμβο είτε η φυσική απόσταση μεταξύ εξυπηρετητή και πελάτη. Η τελευταία παράμετρος θα ήταν χρήσιμη σε κινητά περιβάλλοντα, όπου η δρομολόγηση είναι βασισμένη όχι μόνο στον αριθμό των

βημάτων που θα κάνει ένα μήνυμα από την πηγή στον προορισμό, αλλά και στην απόσταση μεταξύ των κόμβων.

Η κυριότερη λοιπόν πρόκληση και απαίτηση είναι για τους χρήστες να ανακαλύπτουν υπηρεσίες, να τις εντοπίζουν μεταξύ χιλιάδων παρόμοιων και τελικά να αλληλεπιδρούν με ασφαλή τρόπο ώστε να τις χρησιμοποιήσουν. Αυτή η αλληλεπίδραση πρέπει να γίνει χωρίς προηγούμενη ή με ελάχιστη γνώση λεπτομερειών του δικτυακού περιβάλλοντος.

Διάφοροι ερευνητές τα περασμένα χρόνια έχουν ψάξει λύσεις στο πρόβλημα της αυτόματης ανακάλυψης υπηρεσιών. Σήμερα, υπάρχουν δεκάδες πρωτόκολλα ανακάλυψης υπηρεσιών (Service Discovery Protocols). Ένας ορισμός τους δίνεται από τη wikipedia [2]: «**Service discovery** protocols are network protocols which allow automatic detection of devices and services offered by these devices on a computer network». Κάποια από αυτά έχουν αναπτυχθεί εμπορικά όπως το SLP [3], το Jini [4], το Salutation [5], το Universal Plug and Play (UPnP) [6], και το Bluetooth SDP [7]. Άλλα έχουν προταθεί όπως το DEAPspace [8], το MARE [9], το Splendor [10] και το Konark [11]. Το σίγουρο είναι πως κανένα πρωτόκολλο δεν αντιμετωπίζει όλα τα πιθανά προβλήματα. Το κάθε πρωτόκολλο εξειδικεύεται και επιλύει μέρος των παρουσιαζόμενων προβλημάτων και εφαρμόζεται καλύτερα σε διαφορετικά σύνολα δικτυακών παραμέτρων.

Στη συνέχεια, θα αναφέρουμε μερικά πρωτόκολλα ανακάλυψης υπηρεσίας που εφαρμόζονται σε σταθερά IP δίκτυα.

2.2 ΑΝΑΚΑΛΥΨΗ ΥΠΗΡΕΣΙΑΣ ΣΕ IP ΔΙΚΤΥΑ

Τα πρωτόκολλα ανακάλυψης υπηρεσιών πρωτοσχεδιάστηκαν για τοπικά δίκτυα όπως το δίκτυο μιας εταιρείας, το δίκτυο γραφείου ή το οικιακό δίκτυο. Καθώς ο συνολικός αριθμός των συσκευών είναι περιορισμένος ανά διαχειριστικό τομέα, τα τοπικά δίκτυα (LANs) παρέχουν υπηρεσίες μέσω Dynamic Host Configuration Protocol (DHCP). Οι συσκευές είναι συνήθως υπολογιστές, εκτυπωτές, fax, ενώ σε κάποιες περιπτώσεις χρησιμοποιούνται και οικιακές συσκευές.

Τα LANs επιτρέπουν ανταλλαγή μεγάλου όγκου μηνυμάτων. Εφόσον δεν αντιμετωπίζετε πρόβλημα κινητικότητας των κόμβων, η ανακάλυψη υπηρεσιών είναι «εύκαμπτη» στο σχεδιασμό και αξιόπιστη στη λειτουργία. Υπάρχοντα πρωτόκολλα σε αυτή τη κατηγορία είναι το SLP, το Jini, το UDDI [12] και το UPnP.

2.2.1 SLP

Το Service Location Protocol (SLP) [3] είναι η λύση που προτείνει το Internet Engineering Task Force (IETF) [13] για την ανακάλυψη υπηρεσιών. Στο πρωτόκολλο αυτό, οι υπηρεσίες που παρέχονται στο δίκτυο αντιπροσωπεύονται από «Πράκτορες Υπηρεσιών», Service Agents (SAs), ενώ οι «Πράκτορες Χρήστη», User Agents (UAs) αντιστοιχούν στους χρήστες και τους πελάτες. Επιπλέον, υπάρχουν και οι Directory Agents (DAs) που αντιστοιχούν σε «Διαμεσολαβητές» και ενώ είναι προαιρετικοί, απαιτούνται κυρίως για καλύτερη αποδοτικότητα. Το SLP μπορεί να λειτουργήσει τόσο σε κεντρικές (centralized) όσο και σε κατακεντρωμένες (distributed) αρχιτεκτονικές δικτύων.

Το SLP αναλύεται στο κεφάλαιο 4 καθώς είναι το πρωτόκολλο που θα μας απασχολήσει κυρίως σε αυτή την διπλωματική.

2.2.2 UPnP

Το Universal Plug and Play (UPnP) [6], που υλοποιήθηκε από τη Microsoft, βασίζεται σε μία συλλογή ευρέως διαθέσιμων πρωτοκόλλων web και σε standards «ανοιχτά» σε όλους ώστε να παρέχει μια δυνατή λύση στο πρόβλημα της ανακάλυψης υπηρεσίας. Το UPnP είναι βασισμένο στη στοίβα πρωτοκόλλων TCP/IP. Οι νεοεισερχόμενοι χρήστες, αποκτούν αυτόματα μια IP διεύθυνση μέσω του DHCP πρωτόκολλου.

Στο UPnP, οι διαμεσολαβητές, που ονομάζονται σημεία ελέγχου (control points), είναι προαιρετικοί. Οι κόμβοι που παρέχουν υπηρεσίες, ανακοινώνουν την παρουσία τους χρησιμοποιώντας το Simple Service Discovery Protocol (SSDP). Αυτό βασίζεται στο HyperText Transfer Protocol (HTTP) χρησιμοποιώντας πακέτα User Datagram Protocol (UDP). Τα ονόματα των υπηρεσιών και των χαρακτηριστικών που αυτές έχουν, περιγράφονται με τη γλώσσα eXtensible Markup Language (XML). Τα μηνύματα που «διαφημίζουν» τις υπηρεσίες περιέχουν ένα καθολικό εντοπιστή πόρου (URL – Universal Resource Locator) που δείχνει στο αρχείο περιγραφής των υπηρεσιών. Για την υλοποίηση και τον έλεγχο μιας υπηρεσίας, χρησιμοποιείται το Simple Object Access Protocol (SOAP), ενώ τα μηνύματα κατάστασης βασίζονται στην αρχιτεκτονική General Event Notification Architecture (GENA). Τα μηνύματα αυτά στέλνονται όταν η κατάσταση μιας υπηρεσίας αλλάξει. Οι ενδιαφερόμενοι πελάτες και διαμεσολαβητές πρέπει να εγγραφούν στους κόμβους που προσφέρουν κάποια υπηρεσία, ώστε να λαμβάνουν αυτά τα μηνύματα.

Τέλος, το UPnP είναι ικανό να παρέχει URLs στο περιβάλλον του χρήστη, επιτρέποντας του να ελέγχει αλλά και να βλέπει τη κατάσταση των υπηρεσιών.

2.2.3 Jini

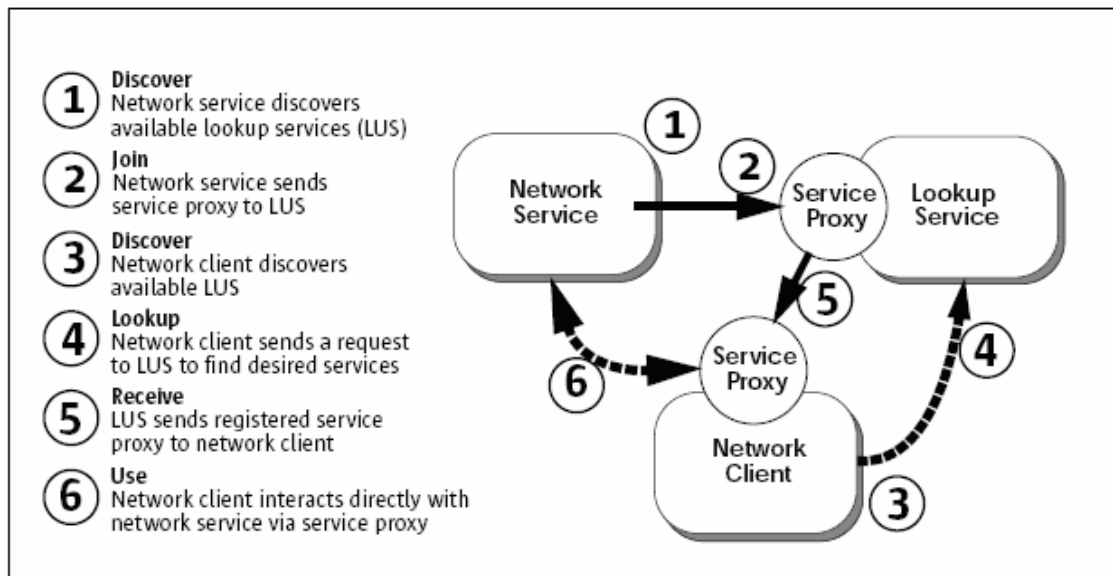
Το Jini [4] αναπτύχθηκε από τη Sun Microsystems, χρησιμοποιώντας τεχνολογία Java για να παρέχει μια λύση-πλατφόρμα στο service discovery. Απαραίτητες προϋποθέσεις είναι οι συσκευές να διαθέτουν πακέτα της γλώσσας Java και ως πρωτόκολλο στηρίζεται στη χρήση διαμεσολαβητών (directories), που καλούνται «πίνακες αναζήτησης στοιχείων» (lookup tables ή lookup services). Πρόκειται για μία βάση δεδομένων που είναι αποθηκευμένη σε έναν εξυπηρετητή (server). Οι πελάτες και οι κόμβοι-υπηρεσίες πρέπει πρώτα να ανακαλύψουν τους lookup tables. Ένας πάροχος υπηρεσιών χρησιμοποιεί ένα πληρεξούσιο (proxy) για να ανακαλύψει τις lookup services μέσα στο δίκτυο ή σε ένα υποσύνολό του. Έπειτα, καταχωρεί το αντικείμενο της υπηρεσίας του και των χαρακτηριστικών της στους πίνακες μαζί με τον αντίστοιχο κώδικα. Όταν ο πελάτης ζητά μια υπηρεσία, τότε ο κώδικας της υπηρεσίας «κατεβαίνει» στην εικονική του μηχανή Java (JVM).

Η αρχιτεκτονική Jini χρησιμοποιεί την έννοια της χρονομίσθωσης (leasing). Οι υπηρεσίες παραμένουν στη βάση δεδομένων μόνο για συγκεκριμένο χρονικό διάστημα. Πριν λήξει αυτή η περίοδος, η υπηρεσία μπορεί να διαπραγματευτεί ξανά

τη μίσθωση. Αν η υπηρεσία αποσυρθεί από το δίκτυο, η εγγραφή της στο lookup table διαγράφεται αυτόματα όταν λήξει η χρονομίσθωση.

Ως παράδειγμα εφαρμογής της τεχνολογίας Jini, ας φανταστούμε ένα κτιριακό συγκρότημα μιας εταιρείας που συνδέεται με δίκτυο που υποστηρίζει το Jini πρωτόκολλο. Διάφορες υπηρεσίες καταγράφονται στα lookup tables στέλνοντας τα αντίστοιχα proxies στο lookup service. Τέτοια πληρεξούσια (proxies) μπορεί να είναι υπηρεσίες τηλεφώνου, πληροφορίες και διαχείριση συσκευών φωτισμού, θέρμανσης, ενέργειας, ακόμα και υπηρεσίες ασφάλειας των εγκαταστάσεων. Όλες αυτές οι υπηρεσίες καταγράφονται ιεραρχικά στο lookup service όπου μπορούν να τις αναζητήσουν οι διάφοροι χρήστες, που μπορεί να είναι κάτοχοι, ένοικοι, εργαζόμενοι, προμηθευτές.

Ο μηχανισμός επικοινωνίας Jini βασίζεται στο Java Remote Method Invocation (RMI), που επιτρέπει την μετακίνηση κώδικα μέσα στο δίκτυο. Αυτό το προνόμιο μετακίνησης και μεταφερσιμότητας του κώδικα αντικαθιστά την ανάγκη προεγκατάστασης drivers στον πελάτη. Η όλη λειτουργία του Jini μπορεί να παρασταθεί με ένα διάγραμμα ροής όπως φαίνεται στο ακόλουθο Σχήμα 2.1:



Σχήμα 2.1: Διάγραμμα ροής του τρόπου λειτουργίας της τεχνολογίας Jini

2.2.4 UDDI

Το UDDI (Universal Description, Discovery and Integration) [12] αποτελεί ένα πρωτόκολλο καταχώρησης των web services. Χρησιμοποιείται για να μπορούμε να παρέχουμε πληροφορίες σχετικά με τις web services. Κάθε καταχώρηση περιέχει ένα WSDL (Web Service Description Language) αρχείο καθώς και τη διεύθυνση λειτουργίας της υπηρεσίας στο Internet. Επιπρόσθετα, σε κάθε καταχώρηση (web registry) υπάρχουν και διάφορες άλλες πληροφορίες για την υπηρεσία που σχετίζονται με τον ιδιοκτήτη της, την πολιτική του κ.ά.

Υπάρχουν διαφορετικοί τύποι καταχωρήσεων μίας υπηρεσίας. Πιο συγκεκριμένα, υπάρχουν καταχωρήσεις που μπορούν να γίνουν για υπηρεσίες από

όλο τον κόσμο και που απευθύνονται σε όλο τον κόσμο, αλλά και καταχωρήσεις που απευθύνονται μόνο σε εξειδικευμένες επιχειρήσεις προωθώντας έτσι και το B2B (business-to-business, ηλεκτρονικό εμπόριο μεταξύ επιχειρήσεων) μοντέλο συνεργασίας. Τέλος, υπάρχουν και καταχωρήσεις υπηρεσιών για πιο εξειδικευμένες περιπτώσεις.

Μια κατηγοριοποίηση των web registries είναι η εξής:

- Public UDDI
- Intra Enterprise UDDI
- Inter Enterprise UDDI

Ένα μοντέλο πληροφοριών αποτελούμενο από στιγμιότυπα συνεχόμενων δεδομένων καλείται οντότητα. Οι οντότητες αναπαρίστανται σε μορφή XML και αποθηκεύονται από κόμβους UDDI. Τα στιγμιότυπα έχουν έναν από τους ακόλουθους τύπους οντοτήτων:

- businessEntity
- businessService
- bindingTemplate
- tModel
- publisherAssertion
- subscription.

Στο πρωτόκολλο UDDI, το tModel (technical Model) για παράδειγμα, χρησιμοποιείται για να περιγράψει τις υπηρεσίες. Υπάρχουν πολλαπλά tModels που συνεργάζονται. Στην ανακάλυψη υπηρεσιών χρησιμοποιείται συχνά το HTTP Transport tModel που μεταφέρει μηνύματα μέσω του HTTP πρωτοκόλλου.

2.2.5 Salutation

Το Salutation [5] αναπτύχθηκε από μια ανοιχτή συνεργασία (Salutation Consortium). Το πρωτόκολλο Salutation είναι ανεξάρτητο του φυσικού επιπέδου και του πρωτοκόλλου μεταφοράς. Ένας Transport Manager (TM) είναι απαραίτητος για κάθε υποδομή ώστε να παρέχει ένα αξιόπιστο δίαυλο επικοινωνίας. Οι διαμεσολαβητές (directories), που ονομάζονται Salutation Managers (SMs), πρέπει να «κάθονται» πάνω από ένα ή περισσότερους TM(s). Ο SM δρα κυρίως ως μεσίτης υπηρεσιών που παρέχει δυνατότητες ανακάλυψης, ψαξίματος στοιχείων και πρόσβασης μέσω ενός ανεξάρτητου πρωτοκόλλου επικοινωνίας. Οι SMs είναι κατανεμημένοι παντού σε ένα δίκτυο και μπορούν να ανταλλάσσουν πληροφορίες μεταξύ τους ώστε να αυξήσουν την ευρύτητα αναζήτησης υπηρεσιών. Το Salutation επίσης, προσφέρει μια έκδοση του πρωτοκόλλου του, το Salutation Lite, που ταιριάζει καλύτερα σε κινητές συσκευές με περιορισμένες δυνατότητες.

ΚΕΦΑΛΑΙΟ 3

3 SDP ΣΕ AD HOC ΔΙΚΤΥΑ

3.1 ΑΝΑΓΚΗ ΝΕΩΝ ΛΥΣΕΩΝ

Τα ad hoc δίκτυα, όπως έχουμε πει, είναι δυναμικά δίκτυα και χαρακτηρίζονται από υψηλή κινητικότητα των κόμβων και συχνές αλλαγές στην τοπολογία τους. Αυτά τα χαρακτηριστικά, σε συνδυασμό με τους ενεργειακούς περιορισμούς, αποτρέπουν την εφαρμογή των κλασσικών πρωτοκόλλων ανακάλυψης υπηρεσιών. Συνεπώς, αλγόριθμοι όπως το SLP και το Jini, δεν προσφέρουν αποδοτικές λύσεις για ανακάλυψη και παράδοση υπηρεσίας στα MANETs, τουλάχιστον με την παραδοσιακή κεντροποιημένη μορφή τους.

Η ύπαρξη DAs (Directory Agents) και Lookup servers αυξάνουν την πολυπλοκότητα στην εύρεση παρόχων. Επιπλέον, η αποστολή μηνυμάτων για αίτηση υπηρεσίας (SrvReq) αυξάνει την κίνηση στο δίκτυο οδηγώντας ταυτόχρονα σε αύξηση της δαπανώμενης ενέργειας των κόμβων.

Γίνεται λοιπόν κατανοητό ότι πρέπει να βρεθούν λύσεις που προτείνουν αποκεντρωμένες (decentralized) αρχιτεκτονικές όπου ο κάθε κόμβος δεν πρέπει να εξαρτάται από άλλον για να διαφημίσει τις υπηρεσίες του. Κάθε υπηρεσία πρέπει να είναι αυτόνομη παρέχοντας τη δυνατότητα εύκολης και γρήγορης ανακάλυψης από τον χρήστη.

Οι προτεινόμενες λύσεις για ανακάλυψη υπηρεσίας σε ad hoc δίκτυα, βασίζονται στην υλοποίηση πρωτοκόλλων δρομολόγησης που επιτυγχάνουν και προώθηση πακέτων και ανακάλυψη υπηρεσιών. Μηχανισμοί καθολικής (broadcast) και πολλαπλής (multicast) εκπομπής χρησιμοποιούνται ευρέως σε αυτά τα δίκτυα. Παραδείγματα είναι το πρωτόκολλο Bluetooth SDP, το Konark, το GSD, και το DEAPspace. Λύση αποτελεί και η υλοποίηση του γνωστού SLP με μία αποκεντρωμένη (decentralized) προσέγγιση.

3.2 Bluetooth SDP

Το Bluetooth SDP [7] αναπτύχθηκε από το Bluetooth Special Interest Group (SIG). Το πρωτόκολλο αυτό βρίσκει εφαρμογή σε συσκευές που υποστηρίζουν τη τεχνολογία Bluetooth. Ο πελάτης αιτείται μιας υπηρεσίας απευθείας με τον τύπο και τα χαρακτηριστικά της γνωρίσματα. Όλες οι αντίστοιχες υπηρεσίες απαντούν επιτρέποντας έτσι στο χρήστη να τις εντοπίσει. Διαμεσολαβητές (directories) δεν υπάρχουν. Το εν λόγω πρωτόκολλο όμως δεν επιτρέπει άμεση πρόσβαση και χρήση αυτών των υπηρεσιών, απαιτώντας τη συνεργασία με άλλους αλγορίθμους.

Ειδικότερα, το πρωτόκολλο αυτό ορίζει πώς ένας πελάτης μπορεί να ψάξει για μια υπηρεσία βασισμένος σε συγκεκριμένα χαρακτηριστικά (attributes) αυτής. Ωστόσο, δε χρειάζεται να γνωρίζει εκ των προτέρων τις διαθέσιμες υπηρεσίες. Όλες οι πληροφορίες για μια υπηρεσία, διατηρούνται σε έναν SDP Server μέσα σε μία

μοναδική εγγραφή (service record). Η εγγραφή αυτή περιλαμβάνει μια λίστα από τα χαρακτηριστικά της υπηρεσίας. Κάθε service attribute περιγράφει ένα γνώρισμα της συγκεκριμένης υπηρεσίας. Το Bluetooth SDP έχει τα εξής service attributes:

- ServiceRecordHandle,
- ServiceClassIDList,
- ServiceRecordState,
- ServiceID,
- ProtocolDescriptionList,
- BrowseGroupList,
- LanguageBaseAttributeIDList,
- ServiceInfoTimeToLive,
- ServiceAvailability,
- BluetoothProfileDescriptorList,
- DocumentationURL,
- ClientExecutableURL,
- IconURL,
- ServiceName,
- ServiceDescription,
- ProviderName.

Κάποια ορίσματα είναι κοινά για όλες τις εγγραφές υπηρεσιών, ενώ κάποιοι πάροχοι υπηρεσιών μπορούν να ορίσουν επιπλέον δικά τους service attributes. Κάθε attribute αποτελείται από δύο στοιχεία: Τον αριθμό attribute ID και την τιμή του χαρακτηριστικού (attribute value).

- Το attribute ID είναι ένας 16-bit μη προσημασμένος ακέραιος αριθμός που ξεχωρίζει κάθε γνώρισμα από τα υπόλοιπα μέσα στην εγγραφή της υπηρεσίας. Επίσης, διακρίνει και τη σημασία κάθε τιμής του χαρακτηριστικού.
- Το attribute value είναι ένα μεταβλητού μήκους πεδίο. Η σημασία της τιμής αυτής καθορίζεται από το αντίστοιχο attribute ID καθώς και από την κλάση της υπηρεσίας της εγγραφής στην οποία περιέχεται. Στο πρωτόκολλο, το attribute value παριστάνεται ως ένα στοιχείο δεδομένων.

Στο Bluetooth SDP κάθε υπηρεσία είναι ένα στιγμιότυπο μιας κλάσης υπηρεσίας (service class). Ο ορισμός του service class παρέχει τους ορισμούς όλων των attributes της εγγραφής που αναπαριστά το στιγμιότυπο. Ο ορισμός κάθε attribute καθορίζει την αριθμητική τιμή του attribute ID, την προτεινόμενη σημασία και τη μορφή του attribute value. Μια εγγραφή υπηρεσίας περιέχει συγκεκριμένα attributes της αντίστοιχης κλάσης καθώς και καθολικά attributes που είναι κοινά σε πολλές υπηρεσίες.

Κάθε service class προσδιορίζεται από ένα μοναδικό αναγνωριστικό, ένα UUID (Universally Unique Identifier) που είναι καθολικό και εγγυάται μοναδικότητα σε όλη τη διάρκεια και σε όλο το δίκτυο. Τα UUID μπορούν να δημιουργηθούν ανεξάρτητα σε μια κατανομημένη αρχιτεκτονική. Δεν απαιτείται κεντρική εγγραφή τους. Είναι, κατά κανόνα, μια τιμή των 128-bit αλλά συχνά και των 16 ή 32 που επεκτείνεται στο στάνταρ μέγεθος με την εξής διαδικασία:

$$128_bit_value = 16_bit_value * 2^{96} + Bluetooth_Base_UUID$$

$$128_bit_value = 32_bit_value * 2^{96} + Bluetooth_Base_UUID$$

Στο Bluetooth SDP, η ανακάλυψη υπηρεσίας παίρνει δύο ερμηνείες. Η μία είναι το *searching for services* που σημαίνει αναζήτηση συγκεκριμένων υπηρεσιών. Η διαδικασία αυτή γίνεται από τους πελάτες που αναζητούν τιμές των χαρακτηριστικών γνωρισμάτων κάθε υπηρεσίας. Παρέχεται η δυνατότητα αναζήτησης attributes των οποίων η τιμή είναι συγκεκριμένο UUID. Το service search είναι λοιπόν μια λίστα από UUIDs που χρησιμοποιούνται για να εντοπίσουν τις υπηρεσίες που ανταποκρίνονται στις απαιτήσεις.

Η άλλη ερμηνεία είναι το *Browsing for Services* που σημαίνει η αναζήτηση του ποιες υπηρεσίες παρέχονται εκείνη τη στιγμή. Ο μηχανισμός του browsing βασίζεται σε ένα attribute που είναι κοινό σε όλες τις κλάσεις υπηρεσιών και τα οποίο καλείται BrowseGroupList attribute. Η τιμή του περιέχει μία λίστα από UUIDs. Κάθε UUID αναπαριστά μια ομάδα φυλλομέτρησης (browse group) με την οποία σχετίζεται μια υπηρεσία. Όταν ο πελάτης επιθυμεί να φυλλομετρήσει τις υπηρεσίες ενός SDP server, δημιουργεί ένα πλάνο εύρεσης υπηρεσίας. Το πλάνο αυτό περιέχει το UUID που αναπαριστά το αντίστοιχο browse group. Έτσι, όλες οι υπηρεσίες που προβάλλονται, θέτουν την τιμή του UUID του αντίστοιχου browse group ως τιμή του attribute BrowseGroupList.

3.3 Konark

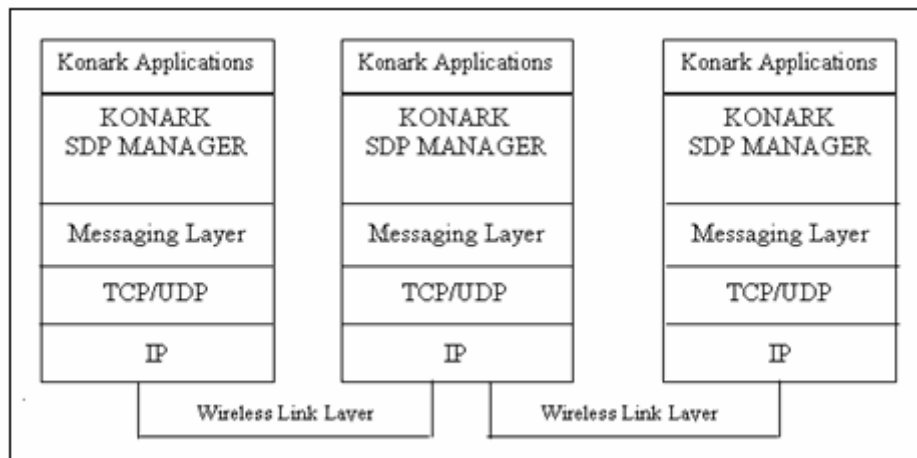
Το Konark [11] είναι ένα πρωτόκολλο ανακάλυψης και παράδοσης υπηρεσίας, σχεδιασμένο για μεγάλα ασύρματα multi-hop ad hoc δίκτυα. Το Konark είναι βασισμένο σε peer-to-peer μοντέλα όπου κάθε συμμετέχων έχει τη δυνατότητα να φιλοξενεί υπηρεσίες, να τις διανέμει μέσω ενός HTTP server, να ρωτάει το δίκτυο για άλλες παρεχόμενες υπηρεσίες και τελικά να τις χρησιμοποιεί.

Το πρωτόκολλο Konark είναι ανεξάρτητο των κατώτερων στρωμάτων δικτύου που μπορεί να είναι είτε το 802.11, ή το IrDA, ή το Bluetooth είτε κάποιο άλλο πρωτόκολλο που υλοποιείται σε επίπεδο IP.

Σημαντικό ζήτημα στα SDPs είναι πώς θα διατίθενται στο δίκτυο οι πληροφορίες για υπηρεσίες. Το Konark υποστηρίζει τόσο τη διαφήμιση (advertisement) όσο και την εύρεση (discovery). Οι πάροχοι υπηρεσιών προωθούν τις υπηρεσίες τους στο δίκτυο. Ο ρυθμός που γίνεται αυτό εξαρτάται από πολλούς παράγοντες. Άλλες φορές γίνεται σε περιοδική βάση, άλλες όταν εισέρχεται στο δίκτυο ένας νέος κόμβος και άλλοτε εξαρτάται από γεωγραφικούς ή χρονικούς παράγοντες. Τα διαφημιστικά αυτά μηνύματα περιέχουν τη πληροφορία time-to-live (TTL) ώστε να διατηρείται το δίκτυο ενημερωμένο. Οι πελάτες, αποθηκεύουν τις πληροφορίες με τις υπηρεσίες και μπορούν να τις χρησιμοποιήσουν αργότερα.

Κάθε συσκευή περιλαμβάνει μια Εφαρμογή Konark (Konark Application) που διευκολύνει την ανθρώπινη παρέμβαση στην αρχικοποίηση, στον έλεγχο των διαδικασιών διαφήμισης και ανακάλυψης και στη χρήση των υπηρεσιών. Περιλαμβάνει επίσης SDP Managers και καταχωρητές (registries). Και οι δύο διατηρούν αντικείμενα υπηρεσιών και πληροφορίες που προσφέρονται από τους συμμετέχοντες κόμβους. Επίσης, όπως αναφέραμε και προηγουμένως, χρησιμοποιούνται τα πρωτόκολλα HTTP και SOAP για την αποστολή των υπηρεσιών, η περιγραφή των οποίων γίνεται με τη γλώσσα XML.

Πιο αναλυτικά, όσον αφορά το service discovery, χρησιμοποιείται η παρακάτω (Σχήμα 3.1) λίστα πρωτοκόλλων του Konark service discovery:

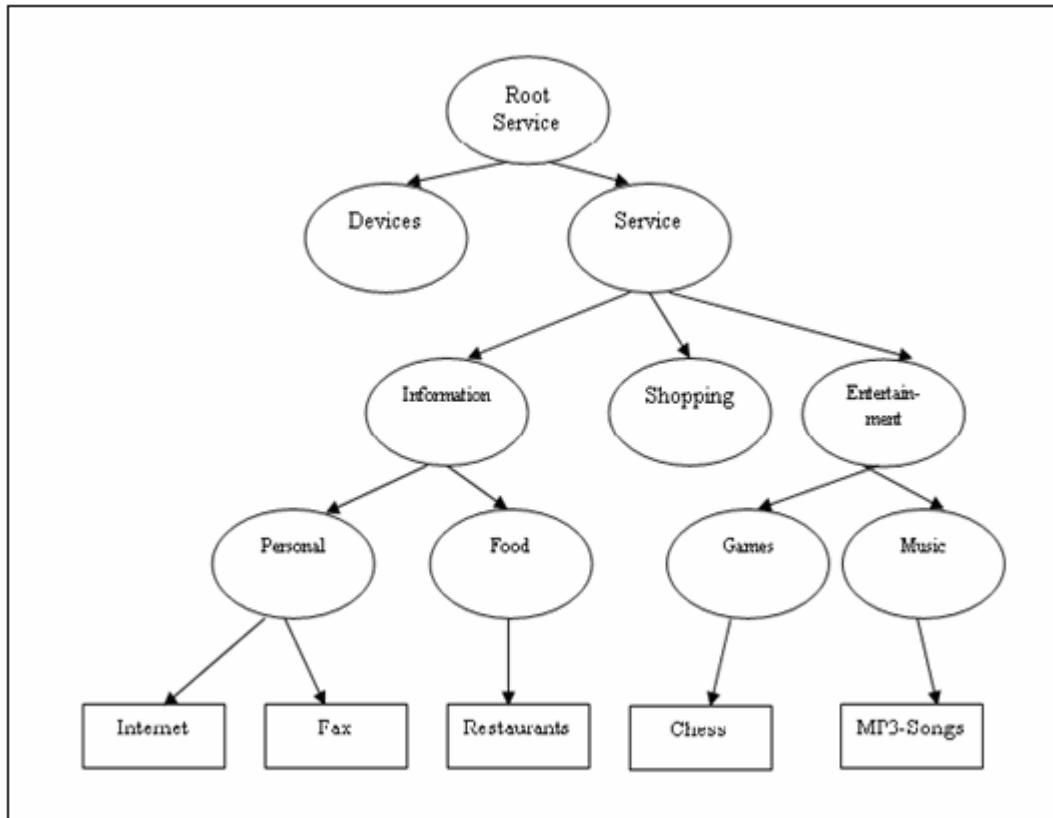


Σχήμα 3.1: Konark service discovery

Όπως βλέπουμε, ο Konark SDP Manager αλληλεπιδρά με το επίπεδο μηνυμάτων για να στείλει και να παραλάβει μηνύματα ανακάλυψης και διαφήμισης. Το επίπεδο messaging είναι πάνω από το επίπεδο μεταφοράς. Η διαδικασία του service discovery σχετίζεται με διαφημιστικούς μηχανισμούς και δομές καταχώρησης υπηρεσιών (service registry).

Το service registry είναι μια δομή που επιτρέπει σε συσκευές να αποθηκεύουν τις υπηρεσίες τους καθώς και να διατηρούν πληροφορίες για υπηρεσίες που έχουν ανακαλύψει ή παραλάβει. Ο Konark SDP Manager διατηρεί μια δενδρική δομή για τις καταχωρήσεις. Επιτρέπει στα μονοπάτια του δένδρου να χρησιμοποιηθούν απευθείας σε μια ερώτηση ενός πελάτη ή σε διαφημιστικό μήνυμα.

Το δένδρο έχει έναν αριθμό επιπέδων που αναπαριστά την ταξινόμηση των υπηρεσιών. Καθώς μετακινούμαστε από τη ρίζα προς τα φύλλα, οι υπηρεσίες γίνονται πιο εξειδικευμένες. Ένα παράδειγμα παρουσιάζεται στο επόμενο Σχήμα 3.2:



Σχήμα 3.2: Παράδειγμα Δένδρου Υπηρεσιών

Το δένδρο υπηρεσιών (service tree) είναι χρήσιμο για τις συσκευές είτε ενεργούν ως πελάτες είτε ως πάροχοι υπηρεσιών. Οι εξυπηρετητές το χρησιμοποιούν για να αποθηκεύσουν τις τοπικές υπηρεσίες τους, να τις διαφημίσουν και να απαντήσουν σε αιτήσεις πελατών. Αντίστοιχα, οι πελάτες το χρησιμοποιούν για να ανακαλύψουν κάθε επιπέδου υπηρεσίες.

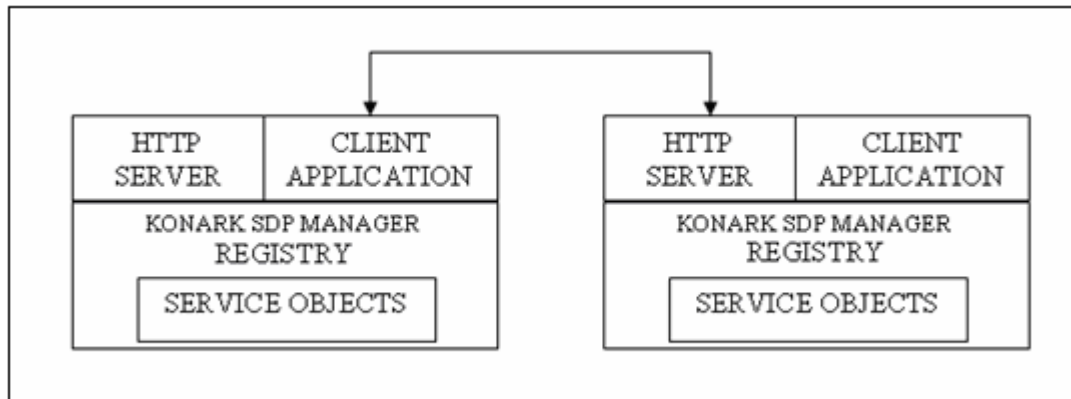
Η διαδικασία της ανακάλυψης έχει δύο βήματα. Στο πρώτο βήμα, ο πελάτης στέλνει μηνύματα ανακάλυψης σε μια σταθερή ομάδα (group). Στο δεύτερο βήμα, απαντούν όλοι οι πάροχοι που έχουν τις αντίστοιχες υπηρεσίες. Το μήνυμα ανακάλυψης που στέλνει ο πελάτης περιέχει είτε το μονοπάτι του service tree είτε μια λέξη-κλειδί. Το μονοπάτι χρησιμοποιείται σε περιπτώσεις αναζήτησης γενικών υπηρεσιών. Η λέξη-κλειδί χρησιμοποιείται για αναζήτηση εξειδικευμένων υπηρεσιών. Αυτά τα μηνύματα έχουν επίσης και ένα πεδίο με τη θύρα στην οποία ο πελάτης δέχεται τις απαντήσεις των servers.

Αντίστοιχα, όταν ο πελάτης λάβει ένα μήνυμα διαφήμισης, κοιτάει να αντιστοιχήσει τις αιτούμενες υπηρεσίες. Αν το μήνυμα βασίζεται σε μονοπάτι, ο server το αντιστοιχεί με το registry tree και επιλέγει όλες τις αντίστοιχες υπηρεσίες. Αν το μήνυμα ανακάλυψης βασίζεται σε λέξη-κλειδί, ο server το αντιστοιχεί με τις δικές του λέξεις-κλειδιά. Όταν λοιπόν βρεθεί η υπηρεσία, ο server δημιουργεί ένα μήνυμα που περιέχει την ονομασία της υπηρεσίας, το μονοπάτι της, τον τύπο, το URL όπου η περιγραφή της υπηρεσίας είναι διαθέσιμη και τη τιμή του TTL.

Η διαδικασία του service delivery αποτελείται από δύο βήματα-διαδικασίες:

- τη περιγραφή της υπηρεσίας (service description), όπου ο πελάτης ενημερώνεται για τις ιδιότητες και τις δυνατότητες της εκάστοτε υπηρεσίας
- τη χρήση της υπηρεσίας (service usage), όπου ο πελάτης χρησιμοποιεί προς όφελός του τις παρεχόμενες από της υπηρεσίες δυνατότητες.

Το service delivery περιλαμβάνει αρχικά την επικοινωνία μεταξύ της Client – Application και του micro-HTTP server, που διαθέτει ο πάροχος όπως φαίνεται στο παρακάτω Σχήμα 3.3:



Σχήμα 3.3: Στοιχεία του service delivery

Όταν ανακαλυφθεί μια υπηρεσία από έναν πελάτη, γνωστά είναι μόνο το όνομα, ο τύπος, η IP διεύθυνση και η διάρκεια διαθεσιμότητας. Βασισμένος σε αυτά, ψάχνει για περισσότερες λεπτομέρειες. Αυτό το στάδιο είναι γνωστό ως περιγραφή υπηρεσιών (service description) και είναι το πρώτο στάδιο στο service delivery. Η συσκευή του πελάτη ακολουθεί το URL (που είναι της μορφής <http://169.234.54.5/music.xml>) για περισσότερες πληροφορίες.

Το αρχείο του service description περιέχει ολοκληρωμένες πληροφορίες για τις ιδιότητες της υπηρεσίας και ο χρήστης μπορεί να αλληλεπιδράσει με οποιαδήποτε λειτουργία χρησιμοποιώντας κατάλληλες παραμέτρους. Το μήνυμα αλληλεπίδρασης είναι ένα SOAP μήνυμα και στέλνεται στον micro-HTTP server.

3.4 GSD

Το GSD (group-based distributed service discovery protocol) [14] έχει προταθεί για τα κινητά ad hoc δίκτυα και χρησιμοποιεί αποτελεσματικά το εύρος ζώνης του δικτύου για την εύρεση υπηρεσίας μέσα σε αυτό. Βασίζεται στην αρχή αποθήκευσης διαφημιστικών μηνυμάτων σε peer-to-peer δίκτυα και στην επιλεκτική group-based προώθηση αιτήσεων. Για την περιγραφή αυτών των αιτήσεων, το πρωτόκολλο εκμεταλλεύεται τις δυνατότητες της DARPA Agent Markup Language (DAML), αυξάνοντας έτσι την ευκαμψία του δικτύου στην ανακάλυψη υπηρεσιών. Η peer-to-peer αποθήκευση υπηρεσιών, δίνει στο πρωτόκολλο μια κατανεμημένη (distributed)

προσέγγιση, διαφορετική από τις κεντροποιημένες (centralized) αρχιτεκτονικές των κλασσικών πρωτοκόλλων ανακάλυψης και παράδοσης υπηρεσίας. Πρόκειται για μια λύση που ταιριάζει στον ετερογενή χαρακτήρα των MANETs.

Αναλύοντας περισσότερο την αρχιτεκτονική του GSD, αξίζει να σημειωθεί ότι οι αιτήσεις υπηρεσιών εκφράζονται με χρήση της DARPA και αντιστοιχίζονται στις περιγραφές των υπηρεσιών μέσω ενός σχήματος «ταιριάσματος» υπηρεσίας (service-matching). Οι υπηρεσίες των κόμβων ταξινομούνται σε αρκετές ομάδες (group) με βάση την ιεραρχία των κλάσεων και υποκλάσεων της DAML. Έτσι, μια υπηρεσία ανήκει στην ιεραρχία ενός group ξεκινώντας από το αρχικό group που ονομάζεται «Service». Κάθε κόμβος διαφημίζει τις υπηρεσίες του στους γειτονικούς κόμβους που απέχουν n^1 βήματα. Συνεπώς, κάθε κόμβος δε χρειάζεται να διατηρεί στη μνήμη του όλες τις υπηρεσίες του δικτύου. Η διαφήμιση επίσης, περιλαμβάνει μια λίστα με τα group που ο κόμβος-αποστολέας έχει ανακαλύψει στην περιοχή κάλυψής του. Αυτό το group χρησιμοποιείται για να προωθήσει επιλεκτικά μια αίτηση σε άλλους κόμβους μέσα στο δίκτυο. Χρησιμοποιώντας τα στοιχεία και τις δυνατότητες της DAML, μειώνουμε το φορτίο που κυκλοφορεί στο δίκτυο παρέχοντας παράλληλα εγγύηση ότι οι αιτήσεις θα φτάσουν στους κόμβους που έχουν την ζητούμενη υπηρεσία.

Ακολουθώς αναλύονται οι τρεις κύριες λειτουργίες του GSD πρωτοκόλλου, τα μηνύματα service advertisements, το peer-to-peer caching και το request routing:

- **Service advertisements:** Τα διαφημιστικά μηνύματα ξεκινούν από κάθε κόμβο που έχει μία ή περισσότερες υπηρεσίες. Κάθε υπηρεσία έχει μια αντίστοιχη περιγραφή βασισμένη σε μια οντότητα DAML. Ο κόμβος, μετά από την πάροδο ADV TIME INTERVAL, στέλνει μια λίστα των υπηρεσιών του στους κόμβους της εμβέλειάς του. Οι ομότιμοι κόμβοι θεωρούνται ότι είναι εντός ενός βήματος. Ο κόμβος αποστολέας μπορεί να στείλει και σε πολλαπλά βήματα. Σε αυτή τη περίπτωση, το πεδίο ADV DIAMETER στο διαφημιστικό πακέτο τίθεται στη τιμή των βημάτων που αναμένεται να κάνει το πακέτο. Όσο μεγαλύτερος ο αριθμός των βημάτων, τόσοι περισσότεροι κόμβοι θα λάβουν τη διαφημιστική λίστα.

Το ADV TIME INTERVAL είναι μια παράμετρος που εξαρτάται από την κινητικότητα των κόμβων και τη γενικότερη κινητικότητα του δικτύου MANET. Σε καταστάσεις υψηλής κινητικότητας, είναι επιθυμητό η τιμή της συγκεκριμένης παραμέτρου του χρόνου να είναι μικρή.

Ένα διαφημιστικό πακέτο έχει τα εξής πεδία:

- ο Packet-type,
- ο Source-Address,
- ο Service-Description,
- ο Service-Groups,
- ο Other-Groups,
- ο Lifetime,
- ο ADV DIAMETER.

Κάθε κόμβος διαθέτει ένα broadcast-id που αυξάνει μονοτονικά με κάθε broadcast εκπομπή από αυτόν. Μία source-address και ένα broadcast-id χρησιμοποιούνται για να αποσαφηνίζουν διπλότυπα διαφημίσεων. Το πεδίο Service-description και το Servicegroups περιέχουν πληροφορίες για τις υπηρεσίες και τα group που θεωρούνται τοπικά για τον αντίστοιχο κόμβο. Το πεδίο Other-groups περιέχει μια αριθμημένη λίστα με τα μη τοπικά group που ο

¹ Το n να κυμαίνεται από 1 έως MAX ADVERTISEMENT DIAMETER

κόμβος έχει παρατηρήσει στη γειτονική του περιοχή. Αυτή η πληροφορία παραλαμβάνεται από το service cache. Το Lifetime καθορίζει το χρόνο που μια διαφήμιση θα παραμείνει αποθηκευμένη στο κόμβο-παραλήπτη. Η καταχώρηση διαγράφεται όταν παρέλθει ορισμένο χρονικό διάστημα.

- **Peer-to-peer Caching:** Κάθε κόμβος στο δίκτυο διατηρεί μια πεπερασμένη προσωρινή μνήμη (cache) για να αποθηκεύσει τις περιγραφές των υπηρεσιών. Όταν λαμβάνει ένα διαφημιστικό μήνυμα, κάνει τα εξής βήματα:

```

if (Duplicate(Advertisement)) // εάν το διαφημιστικό μήνυμα είναι
                             // διπλότυπο
then discard packet;        // απόρριψε το πακέτο
else {                       // αλλιώς
Extract Service Description; // άντλησε τη πληροφορία από το πεδίο
                             // Service Description
Extract group Information;    // άντλησε τη πληροφορία από το group
Store Entry in Service Cache; // αποθήκευσε την καταχώρηση στο
                             // Service Cache
Set the lifetime on the Entry; // όρισε το πεδίο lifetime στη
                             // καταχώρηση
}
    
```

Πίνακας 3.1 Peer-to-peer Caching

Κάθε εγγραφή στο service cache περιέχει τα ακόλουθα πεδία:

- o Source-Address,
- o local,
- o Service-Description,
- o Service-Groups,
- o Other-Groups,
- o Lifetime.

Το service cache σε κάθε κόμβο περιέχει μία λίστα με τις τοπικές αλλά και τις πιο απομακρυσμένες υπηρεσίες που ο κόμβος έχει καταγράψει μέσω διαφημιστικών μηνυμάτων. Το πεδίο Other-Groups περιέχει μία λίστα με τα groups που το αντίστοιχο Source-Address έχει παρατηρήσει στη γειτονική του περιοχή. Στη περίπτωση τοπικών υπηρεσιών, το πεδίο είναι κενό. Το lifetime μιας εγγραφής στη μνήμη καθορίζεται από το αντίστοιχο διαφημιστικό πακέτο. Γρήγορα μετακινούμενοι κόμβοι έχουν μικρό lifetime. Αν μια service cache είναι γεμάτη, τότε χρησιμοποιούμε μια πολιτική χαμηλότερου lifetime ώστε να απομακρύνουμε εγγραφές. Έτσι απομακρύνονται εγγραφές με το μικρότερο lifetime. Οι εγγραφές σβήνονται και όταν το lifetime λήξει.

- **Request Routing:** Μία αίτηση υπηρεσίας (Service Request) πηγάζει από έναν κόμβο του οποίου το επίπεδο εφαρμογών αναζητεί μια υπηρεσία. Αρχικά, η περιγραφή της αίτησης αντιστοιχίζεται με τις υπηρεσίες της τοπικής μνήμης του κόμβου-πηγή. Σαν αποτέλεσμα του peer-to-peer caching, η τοπική μνήμη (σε ένα ιδανικό σενάριο) περιέχει τις περιγραφές των υπηρεσιών σε κόμβους που απέχουν ADV DIAMETER άλματα. Έτσι, συγκρίνοντας την αίτηση, το πρωτόκολλο αυτόματα ελέγχει τη διαθεσιμότητα των υπηρεσιών. Ένα απλό πρωτόκολλο που βασίζεται σε broadcast εκπομπή (χωρίς caching) θα έκανε broadcast την αίτηση σε όλους τους κόμβους της εμβέλειας του. Συνεπώς, το GSD επιτυγχάνει αποτελεσματικότερη ανακάλυψη υπηρεσιών στην γειτνιάζουσα περιοχή. Όταν

δεν υπάρχει αντιστοιχία στην τοπική μνήμη, τότε δημιουργείται μια αίτηση που περιέχει τα παρακάτω πεδία:

- o Packet-type,
- o broadcastId,
- o Service-Description,
- o Request Groups,
- o Source-Address,
- o Last-Address,
- o Hop-Count.

Το πεδίο Request-Groups περιέχει το Service Group στο οποίο ανήκει η αίτηση. Το Hop-Count καθορίζει τον αριθμό των αλμάτων που η αίτηση μπορεί να κάνει. Οι κλασσικές broadcast λύσεις απαιτούν καθολική εκπομπή σε όλους τους γειτονικούς κόμβους κάτι που δεν είναι αποτελεσματικό από άποψη χρησιμοποίησης του εύρους ζώνης. Το πεδίο Other-Groups παρέχει τη πληροφορία σε κάθε εγγραφή του Service Cache. Έτσι, καθορίζεται ένα σύνολο γειτονικών κόμβων στο οποίο μπορεί επιλεκτικά να προωθήσει την αίτηση υπηρεσίας. Επιπρόσθετα, καθορίζει τα groups των υπηρεσιών που ο αποστολέας έχει δει στη γειτονική του περιοχή. Έτσι, εάν η αίτηση ανήκει σε κάποιο από αυτά τα group, τότε υπάρχει περίπτωση η ζητούμενη υπηρεσία να είναι διαθέσιμη σε κοντινό κόμβο. Αποτρέπεται άρα η καθολική εκπομπή της αίτησης.

3.5 DEAPspace

Το DEAPspace [8] προτάθηκε από την IBM Research για να λύσει το πρόβλημα του service discovery σε single-hop ad hoc δίκτυα. Στόχος είναι η ελαχιστοποίηση του χρόνου που απαιτείται για την ανακάλυψη νέων συσκευών στο δίκτυο χωρίς να υπάρχει σημαντική αύξηση της χρησιμοποίησης εύρους ζώνης και των ενεργειακών αναγκών.

Το πρωτόκολλο εφαρμόζει proactive τεχνική προώθησης των διαφημιστικών μηνυμάτων, όπως κάνουν όλα τα πρωτοκόλλα δρομολόγησης καθοδηγούμενα από πίνακα (βλ. [15] για table-driven πρωτόκολλα). Ωστόσο, κάθε συσκευή όχι μόνο ανακοινώνει τις υπηρεσίες που η ίδια παρέχει, αλλά και αυτών που γνωρίζει από άλλες συσκευές. Αυτό εξασφαλίζει πως μια χαμένη διαφήμιση δε θα επηρεάζει σημαντικά κάποιο κόμβο αφού εύκολα μπορεί να αποκτήσει τη πληροφορία από αλλού. Το ίδιο συμβαίνει και όταν εισέρχεται στο δίκτυο μια συσκευή, όπου δεν χρειάζεται να περιμένει όλες τις άλλες να διαφημιστούν ώστε να αποκτήσει γνώση των υπηρεσιών στο δίκτυο.

ΚΕΦΑΛΑΙΟ 4

4 SERVICE LOCATION PROTOCOL (SLP)

4.1 ΕΙΣΑΓΩΓΗ

Το Service Location Protocol (SLP) [3] παρέχει ένα εύκαμπτο και κλιμακωτό πλαίσιο εργασίας σε υπολογιστές (hosts) ώστε αυτοί να έχουν πρόσβαση σε πληροφορίες σχετικές με την ύπαρξη, τον εντοπισμό και τη διαμόρφωση δικτυακών υπηρεσιών. Παραδοσιακά, οι χρήστες πρέπει να ανακαλύψουν τις υπηρεσίες γνωρίζοντας το Domain Name System (DNS) του host (πρόκειται για όνομα που είναι μια αναγνώσιμη συμβολοσειρά, text string) το οποίο είναι ψευδώνυμο της δικτυακής διεύθυνσης. Το SLP εξαλείφει την ανάγκη να γνωρίζει ο χρήστης το όνομα του εκάστοτε host που υποστηρίζει κάποια υπηρεσία. Αντιθέτως, ο χρήστης εισάγει τον επιθυμητό τύπο καθώς και ένα σύνολο χαρακτηριστικών γνωρισμάτων που περιγράφουν την υπηρεσία. Βασισμένο σε αυτή τη περιγραφή, το Service Location Protocol βρίσκει τη διεύθυνση δικτύου της υπηρεσίας που αναζητά ο χρήστης.

Το SLP παρέχει ένα δυναμικό μηχανισμό διαμόρφωσης εφαρμογών σε τοπικά δίκτυα. Οι εφαρμογές (applications) μοντελοποιούνται ως πελάτες (clients) που χρειάζεται να βρουν εξυπηρετητές (servers), οι οποίοι είναι προσαρτημένοι σε κόμβους μέσα στο δίκτυο. Σε περιπτώσεις όπου υπάρχουν πολλοί διαφορετικοί πελάτες και/ή διαθέσιμες υπηρεσίες, το πρωτόκολλο προσαρμόζεται ώστε να χρησιμοποιήσει τους γειτονικούς Directory Agents, που ενδεικτικά αναφέραμε στο 2.2.1, και οι οποίοι προσφέρουν κεντρικοποιημένη λειτουργία υπηρεσιών.

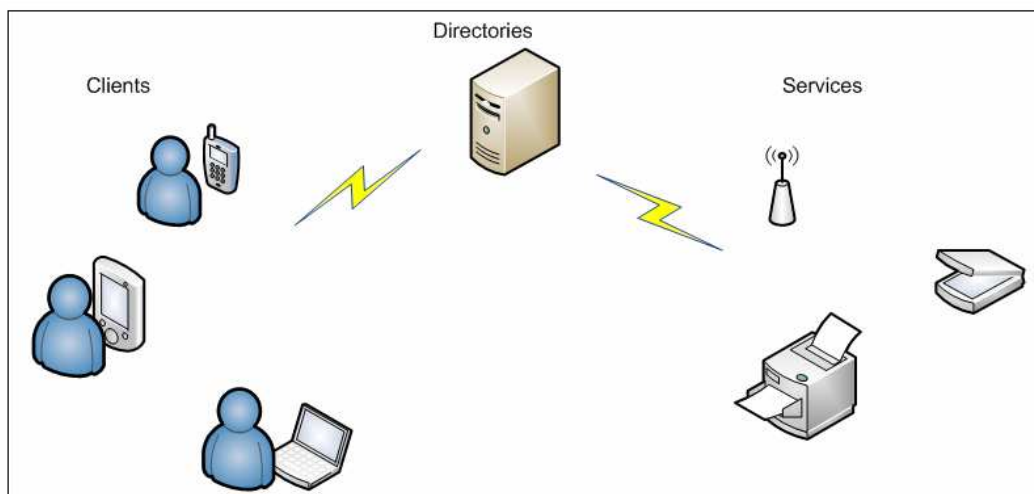
Στο SLP, η διαδικασία του service discovery επιτυγχάνεται χρησιμοποιώντας τρεις οντότητες οι οποίες χαρακτηρίζονται και ως πράκτορες (agents). Αυτοί λειτουργούν εκ μέρους των δικτυακών κόμβων και κατατάσσονται στις ακόλουθες κατηγορίες: i) *τους User Agents (UAs)*, ii) *τους Service Agents (SAs)* και iii) *τους Directory Agents (DAs)*. Οι Υπηρεσίες (Services) αντιπροσωπεύονται από τους SAs, ενώ οι UAs μεσολαβούν για χρήστες και εφαρμογές. Η λειτουργία ανακάλυψης έχει δύο μορφές, παθητική ή ενεργητική. Στην πρώτη, οι SAs εκπέμπουν καθολικά πληροφορίες υπηρεσιών προς τους DAs (passive discovery), ενώ στη δεύτερη απαντούν σε αιτήσεις που αφορούν υπηρεσίες (active discovery). Οι UAs αιτούνται πληροφορίες για διαθέσιμες υπηρεσίες (active discovery), ή δέχονται μηνύματα που εκπέμπονται καθολικά από τους DAs (passive discovery). Οι Διαμεσολαβητές, που αντιπροσωπεύονται από τους DAs, είναι ενδιάμεσες αποθήκες πληροφοριών για τις υπηρεσίες του δικτύου. Αποθηκεύουν τις πληροφορίες που στέλνουν οι SAs, και ικανοποιούν αντίστοιχα τα αιτήματα των UAs. Παρόμοια, οι DAs λειτουργούν είτε σε ενεργητική (active) είτε παθητική (passive) κατάσταση. Η αναζήτηση υπηρεσιών γίνεται με ερωτήματα βασισμένα σε τύπου string. Τα Universal Resource Locators (URLs) χρησιμοποιούνται για πρόσβαση στις υπηρεσίες. Παρέχουν στους χρήστες και τις εφαρμογές, την απαραίτητη πληροφορία, ώστε να επικοινωνήσουν απευθείας με την ευρεθείσα υπηρεσία.

Το SLP αναπτύχθηκε κυρίως για σταθερά IP δίκτυα. Μπορεί να λειτουργήσει τόσο σε κεντρικοποιημένη (centralized) όσο και σε κατανεμημένη (distributed)

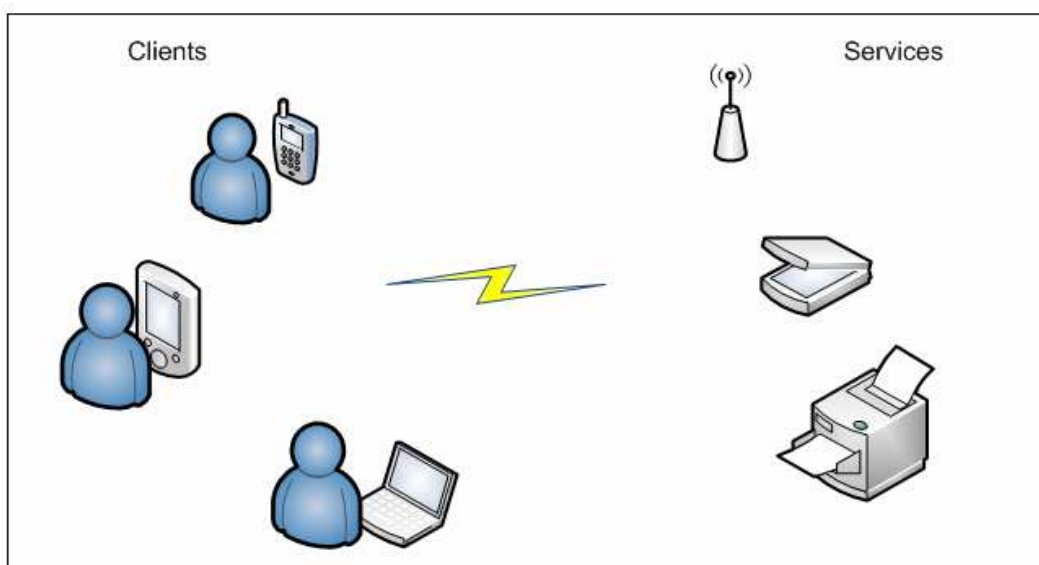
μορφή. Στη κεντροποιημένη λειτουργία, οι servers διαφημίζουν τις υπηρεσίες τους στους DAs και οι πελάτες εκπέμπουν τις αιτήσεις τους απευθείας στους DAs. Οι τελευταίοι, απαντούν με μία λίστα όλων των υπηρεσιών που αντιστοιχούν στις αιτήσεις. Οι πελάτες και οι πάροχοι μαθαίνουν για τους DAs διαβάζοντας ένα στατικό, διαμορφωμένο αρχείο , αναμένοντας περιοδικά διαφημιστικά μηνύματα από τους DAs ή στέλνοντας μηνύματα ώστε να προκαλέσουν μια διαφήμιση που θα αποσταλεί από τους DAs.

Στη κατανεμημένη λειτουργία, δεν υπάρχουν DAs και οι πελάτες αιτούνται υπηρεσιών κοινοποιώντας μηνύματα άμεσα στους παρόχους, μέσω μιας γνωστής διεύθυνσης. Ο server, όταν λάβει ένα αίτημα, απαντάει κατευθείαν στο πελάτη-χρήστη.

Σχηματικά το SLP περιγράφεται στα ακόλουθα σχήματα (Σχήμα 4.1, και Σχήμα 4.2):



Σχήμα 4.1: Κεντροποιημένη λειτουργία SLP



Σχήμα 4.2: Κατανεμημένη λειτουργία SLP

Η ύπαρξη Directory Agents στα ad hoc δίκτυα δεν είναι απαραίτητη καθώς όπως έχει προαναφερθεί, τα MANETs δεν απαιτούν την ύπαρξη κεντρικών κόμβων. Παρακάτω θα αναπτυχθούν τα βασικά στοιχεία του Service Location Protocol όπως παρουσιάστηκαν στο RFC2608 [16]. Εν συνεχεία, θα αναλυθούν οι προτεινόμενες βελτιστοποιήσεις του πρωτοκόλλου ώστε να λειτουργεί το δυνατόν πιο βέλτιστα για τα ασύρματα, κινητά δίκτυα.

4.2 SLP

4.2.1 Ορολογία

User Agent (UA)

Μία διαδικασία που εκτελείται για λογαριασμό ενός χρήστη ώστε να επιτύχει επικοινωνία με μία υπηρεσία/εφαρμογή. Ο UA ανακτά πληροφορία για τις υπηρεσίες του δικτύου από τους Service Agents ή τους Directory Agents.

Service Agent (SA)

Μία διαδικασία που εκτελείται για λογαριασμό ενός ή περισσότερων υπηρεσιών ώστε να τις διαφημίσει.

Directory Agent (DA)

Μία διαδικασία που συλλέγει διαφημίσεις υπηρεσιών. Μπορεί να υπάρχει μόνο ένας DA για κάθε δοσμένο host.

Service Type

Κάθε τύπος υπηρεσίας έχει μία μοναδική συμβολοσειρά Service Type (string).

Naming Authority

Η αντιπροσωπεία ή το group που κατηγοριοποιεί τα Service Types και τα χαρακτηριστικά (Attributes) των υπηρεσιών. Το εξ ορισμού (default) Naming Authority είναι το IANA.

Scope

Ένα σύνολο υπηρεσιών, που συνήθως σχηματίζουν ένα λογικό group διαχείρισης.

URL

Το Universal Resource Locator μεταφέρει τουλάχιστον το όνομα του πρωτοκόλλου πρόσβασης καθώς και τη διεύθυνση που βρίσκεται η υπηρεσία. Αυτή η πληροφορία θα χρησιμοποιηθεί από τον UA μετά από την διαδικασία ανακάλυψης υπηρεσίας με σκοπό την απευθείας επικοινωνία με την ευρισκόμενη υπηρεσία.

Strings

Όλες οι συμβολοσειρές κωδικοποιούνται χρησιμοποιώντας το Universal Transformation Format (UTF)-8 του συνόλου χαρακτήρων Unicode και όταν μεταδίδονται δεν είναι κενού περιεχομένου. Τα strings προηγούνται από ένα πεδίο μήκους δύο bytes.

<string-list>

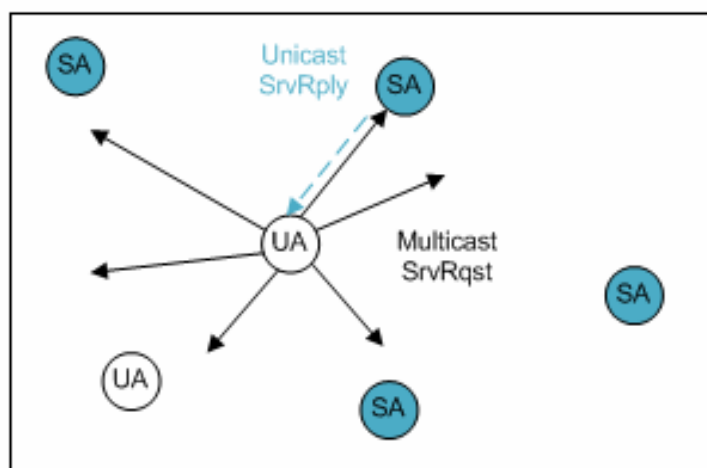
Μία λίστα από strings χωρισμένα με κόμμα με την ακόλουθη σύνταξη:
string-list = string / string ',' string-list

Σε διαγράμματα σχηματισμού των πλαισίων, κάθε πεδίο που τελειώνει με το σύμβολο \ υποδηλώνει ένα πεδίο μεταβλητού μήκους το οποίο δίνεται από προηγούμενο πεδίο στο πρωτόκολλο.

4.2.2 Συνοπτική παρουσίαση του πρωτοκόλλου

Ο User Agent εκπέμπει ένα μήνυμα 'Service Request' (SrvRqst) για λογαριασμό του πελάτη, ορίζοντας τα χαρακτηριστικά της υπηρεσίας που αναζητά ο πελάτης. Ο User Agent θα λάβει ένα μήνυμα Service Reply (SrvRply) που θα ορίζει τη δικτυακή τοποθεσία στο δίκτυο κάθε υπηρεσίας που ικανοποιεί το αρχικό αίτημα.

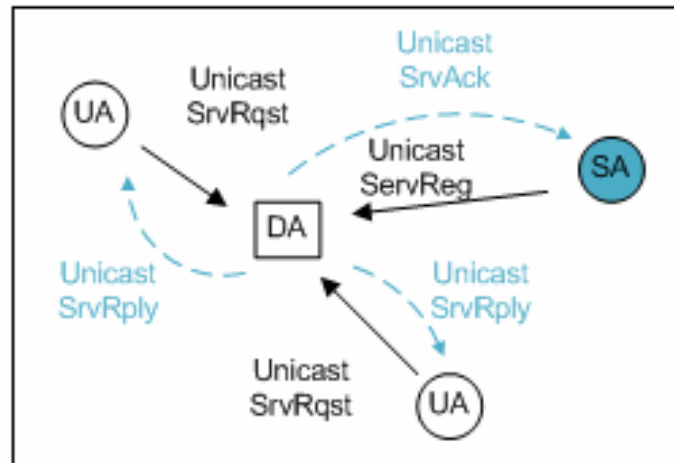
Το πλαίσιο εργασίας του Service Location Protocol επιτρέπει στο User Agent να εκπέμπει απευθείας αιτήματα στους Service Agents. Σε αυτή τη περίπτωση, το request είναι μήνυμα πολλαπλής εκπομπής. Οι Service Agents που λαμβάνουν μια αίτηση για μια υπηρεσία που διαφημίζουν, εκπέμπουν μόνο προς τον ενδιαφερόμενο κόμβο μία απάντηση που περιέχει την τοποθεσία της εν λόγω υπηρεσίας. Σχηματικά εκτελείται η εξής διαδικασία:



Σχήμα 4.3 : Εκπομπή μηνυμάτων SrvRqst και SrvRply

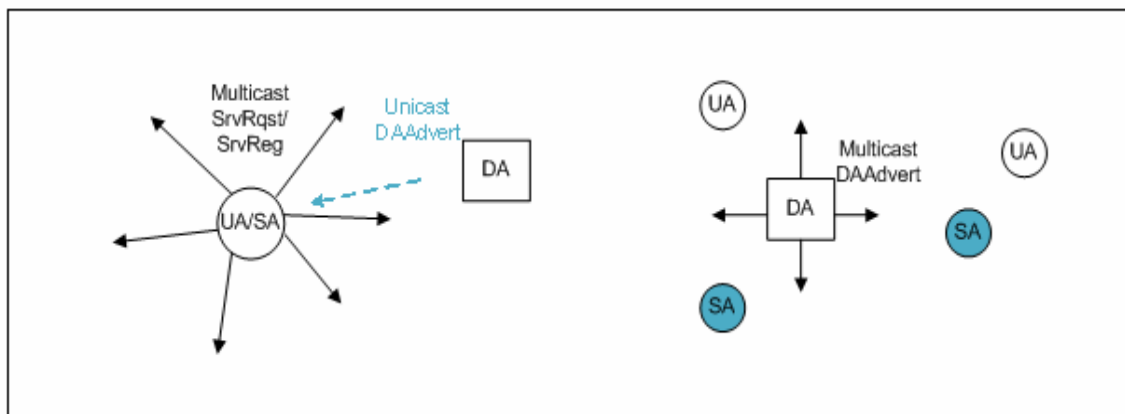
Σε μεγαλύτερα δίκτυα, κρίνεται απαραίτητη η χρησιμοποίηση ενός ή περισσότερων Directory Agents, οι οποίοι λειτουργούν ως μονάδες αποθήκευσης. Οι Service Agents στέλνουν στους DAs μηνύματα εγγραφής (SrvReg) τα οποία περιέχουν όλες τις υπηρεσίες που θέλουν να διαφημίσουν. Ως απάντηση, λαμβάνουν μηνύματα επιβεβαίωσης (SrvAck).

Τα διαφημιστικά μηνύματα πρέπει να ανανεώνονται από τον DA αλλιώς εκπνέουν. Οι User Agents εκπέμπουν αιτήματα στους Directory Agents αντί να τα στέλνουν στους Service Agents, αν βέβαια κάποιοι DAs είναι γνωστοί. Η παραπάνω λειτουργία περιγράφεται ως εξής:



Σχήμα 4.4: Αποστολή μηνυμάτων μέσω Directory Agents

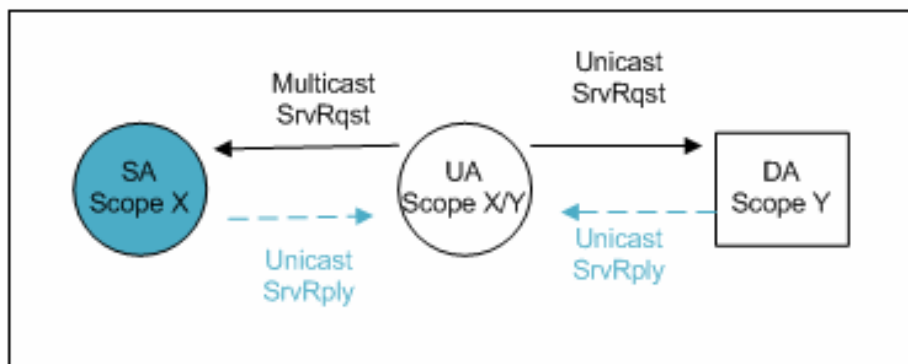
Οι User και οι Service Agents ανακαλύπτουν τους DAs με δύο τρόπους. Πρώτον, εκπέμποντας προς πολλαπλές κατευθύνσεις μηνύματα Service Request ώστε να βρουν τους Directory Agents. Δεύτερον, οι Directory Agent στέλνουν αυτόκλητα διαφημιστικά μηνύματα τα οποία μηνύματα τα ακούν οι User και Service Agents. Σε κάθε περίπτωση, οι Agents λαμβάνουν ένα μήνυμα DA Advertisement (DAAdvert).



Σχήμα 4.5: Ανακάλυψη των Directory Agents

Οι υπηρεσίες ομαδοποιούνται χρησιμοποιώντας εμβέλεις (scopes). Τα scopes είναι strings που προσδιορίζουν υπηρεσίες οι οποίες είναι διαχειριστικά αναγνωρίσιμες. Ένα scope μπορεί να υποδηλώνει μια τοποθεσία, ένα group διαχείρισης, μια γειτονική περιοχή σε μια τοπολογία δικτύου ή κάποια άλλη κατηγορία. Οι Service Agents και οι Directory Agents πάντα προσδιορίζονται από ένα scope string.

Ένας User Agent συνήθως διαθέτει ένα αλφαριθμητικό εμβέλεια (scope string). Μέσα από αυτή την εμβέλεια μόνο ο User Agent μπορεί να ανακαλύψει μια συγκεκριμένη ομάδα από παρεχόμενες υπηρεσίες. Αυτό επιτρέπει στο διαχειριστή του δικτύου να παρέχει υπηρεσίες στους χρήστες. Εναλλακτικά, ο User Agent μπορεί να διαμορφωθεί χωρίς scope. Σε αυτή τη περίπτωση, θα ανακαλύψει όλα τα διαθέσιμα scopes και θα επιτρέψει στον πελάτη να εκπέμψει αιτήματα για κάθε διαθέσιμη υπηρεσία στο δίκτυο. Το παρακάτω Σχήμα 4.6 περιγράφει την εν λόγω διαδικασία:



Σχήμα 4.6: Ανταλλαγή μηνυμάτων μέσω scopes

Στη παραπάνω διαδικασία, ο User Agent σχηματίζεται με scopes X και Y. Αν μια υπηρεσία θεαθεί στο scope X, το αίτημα εκπέμπεται προς πολλαπλούς SAs (multicast). Εάν αυτή η υπηρεσία θεαθεί στο scope Y, το μήνυμα request εκπέμπεται μοναδικά (unicast) προς το DA. Τέλος, αν το request είναι να σταλεί και στα δύο scopes, τότε πρέπει να είναι και unicast και multicast μήνυμα.

4.2.3 Τα URLs που χρησιμοποιούνται στο Service Location

Ένα Service URL υποδεικνύει τη τοποθεσία της υπηρεσίας και μπορεί να είναι της εξής μορφής:

"service:"<srvtype>:"/"<addrspec>

Το πλαίσιο ενός service URL πρέπει να αποτελείται από κάποιο standard όνομα πρωτοκόλλου το οποίο συνδέεται με τη λέξη "service:" και τη δεσμευμένη θύρα. Για παράδειγμα, το "service:ftp://myhost" δείχνει μια ftp υπηρεσία. Μια ftp υπηρεσία σε μία nonstandard θύρα θα είχε τη μορφή: "service:ftp://bad.glad.org:8080".

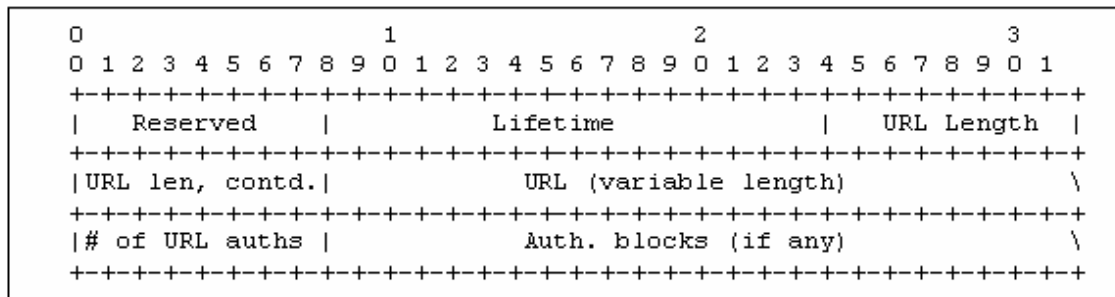
Τα Service Types πρέπει να ορίζονται από ένα "Service Template", το οποίο παρέχει τα αναμενόμενα χαρακτηριστικά, τις τιμές και τη συμπεριφορά του πρωτοκόλλου. Ένας αφηρημένος τύπος υπηρεσίας έχει την εξής μορφή:

"service:<abstract-type>:<concrete-type>".

Η συμβολοσειρά του service type "service:<abstract-type>" ταιριάζει όλες τις υπηρεσίες με τον αφηρημένο τύπο. Αν περιλαμβάνεται το concrete type, μόνο αυτές οι υπηρεσίες ταιριάζουν στα μηνύματα request. Για παράδειγμα, ένα SrvRqst ή AttrRqst το οποίο καθορίζει ως τύπο το "service:printer", θα ταιριάζει και το URL service:printer:lpr://hostname και το service:printer:http://hostname. Αν τα αιτήματα όριζαν "service:printer:http", τότε θα ταιριάζει μόνο το δεύτερο URL.

URL Entries

Το SLP αποθηκεύει τα URLs σε στοιχεία πρωτοκόλλου που καλούνται URL Entries και τα οποία συνδέουν το URL με ένα μήκος, μία διάρκεια ζωής και πιθανόν μια πληροφορία πιστοποίησης. Τα URL Entries, που ορίζονται όπως φαίνεται παρακάτω στο σχήμα, χρησιμοποιούνται στα Service Replies και στα Service Registrations.



Σχήμα 4.7 : URL Entries

4.2.4 Χρήση των Ports, του UDP, και της Multicast εκπομπής

Οι DAs πρέπει να αποδέχονται αιτήματα που στέλνονται με unicast καθώς και με multicast μηνύματα για ανακάλυψη υπηρεσιών (με service type "service:directory-agent").

Οι SAs πρέπει να αποδέχονται τόσο multicast όσο και unicast αιτήματα. Ο SA μπορεί να τα διακρίνει από το αν είναι ενεργοποιημένη η σημαία REQUEST MCAST στην επικεφαλίδα του SLP μηνύματος.

Το Service Location Protocol χρησιμοποιεί εξ' ορισμού πολλαπλή εκπομπή (multicast) για ανακάλυψη των DAs και για μετάδοση αιτημάτων στους SAs.

Η δεσμευμένη θύρα είναι η 427. Αυτή είναι η θύρα προορισμού για όλα τα SLP μηνύματα. Απαντήσεις και επιβεβαιώσεις λήψης στέλνονται στη θύρα από την οποία στάλθηκε το αντίστοιχο αίτημα. Η default μέγιστη μονάδα μεταφοράς πακέτου για τα UDP μηνύματα είναι 1400 bytes εξαιρουμένων των UDP και άλλων επικεφαλίδων.

Εάν ένα SLP μήνυμα δε χωράει σε ένα UDP datagram, πρέπει να «κατακερματιστεί» ώστε να χωράει και τότε ενεργοποιείται η σημαία OVERFLOW στο μήνυμα απάντησης. Ένας UA που λαμβάνει ένα κατακερματισμένο μήνυμα, μπορεί να ανοίξει μία TCP σύνδεση με τον DA ή τον SA και να επανεκπέμψει το αίτημά του χρησιμοποιώντας το ίδιο XID (πρόκειται για πεδίο στο SLP μήνυμα). Μπορεί επίσης, να προσπαθήσει να χρησιμοποιήσει τη κομμένη απάντηση ή να σχηματίσει ένα πιο περιορισμένο αίτημα προς αποστολή.

Το πεδίο Time-To-Live έχει default τιμή για multicast εκπομπή, τη 255. Το να θέσεις το TTL λιγότερο από 255, περιορίζει το εύρος της αναζήτησης στο δίκτυο.

4.2.5 Επαναμετάδοση SLP μηνυμάτων

Τα request μηνύματα που αποτυγχάνουν να εξάγουν κάποια απάντηση, επαναμεταδίδονται. Η αρχική επαναμετάδοση συμβαίνει μετά από μια περίοδο αναμονής CONFIG_RETRY. Οι επαναμεταδόσεις πρέπει να γίνονται με εκθετικό τρόπο, διπλασιάζοντας κάθε φορά τη περίοδο αναμονής. Αυτό βρίσκει εφαρμογή τόσο σε unicast όσο και σε multicast εκπομπές SLP αιτημάτων.

Τα unicast αιτήματα προς έναν DA ή SA πρέπει να επαναμεταδίδονται μέχρι να υπάρξει μία απάντηση (που μπορεί να είναι μήνυμα λάθους) αλλιώς να μεταδίδονται έως ότου παρέλθουν CONFIG_RETRY_MAX δευτερόλεπτα.

Τα multicast αιτήματα πρέπει να επανεκπέμπονται περισσότερο από CONFIG_MC_MAX δευτερόλεπτα μέχρι να ληφθεί κάποιο αποτέλεσμα. Οι UAs πρέπει μόνο να περιμένουν μέχρι η πρώτη απάντηση να ταιριάζει στο αίτημά τους. Έτσι, η επαναμετάδοση δεν είναι απαραίτητη εφόσον ο agent που αιτείται υπηρεσίες χρησιμοποιεί την εφαρμογή «πρώτη απάντηση» αντί της «όσες περισσότερες απαντήσεις σε ένα περιορισμένο χρονικό διάστημα».

Όταν τα SLP μηνύματα SrvRqst, SrvTypeRqst και AttrRqst είναι multicast, περιλαμβάνουν μία <PRList> από προηγούμενους αποκρινόμενους κόμβους. Αρχικά η <PRList> είναι κενή. Όταν αυτά τα μηνύματα είναι unicast, η λίστα <PRList> είναι πάντα κενή. Κάθε DA ή SA που βλέπει τη διεύθυνσή του στη <PRList> δε πρέπει να απαντήσει στο αντίστοιχο αίτημα.

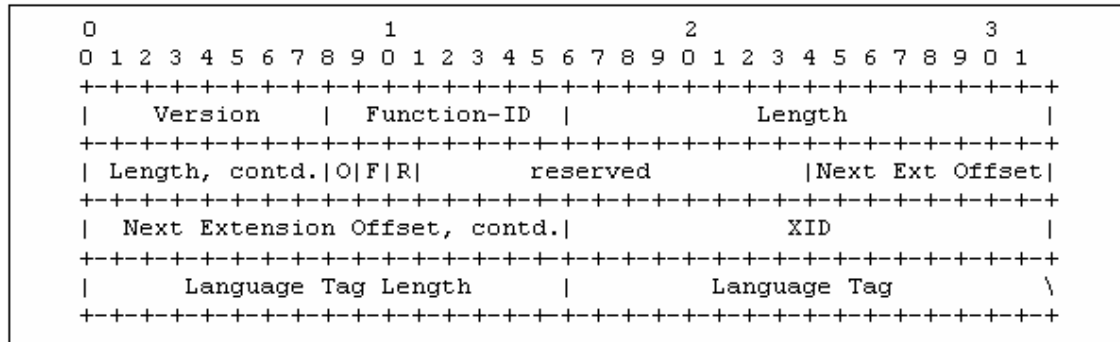
Ένα μήνυμα πρέπει να επανεκπέμπεται έως ότου η <PRList> δεν προκαλεί περεταίρω απαντήσεις ή έως ότου η προηγούμενη αποκρινόμενη λίστα και το αίτημα δε ταιριάζουν σε ένα απλό διάγραμμα ή μέχρι να περάσουν τα CONFIG_MC_MAX δευτερόλεπτα.

Οι UAs που επαναμεταδίδουν ένα αίτημα, χρησιμοποιούν το ίδιο XID. Αυτό επιτρέπει στο DA ή στον SA να αποθηκεύσει την απάντησή του στο αρχικό αίτημα και όταν καταφθάσει ένα αντίγραφο της αίτησης, να έχει έτοιμη την απάντηση και να τη στείλει πάλι. Αυτή η αποθηκευμένη πληροφορία πρέπει να κρατείται όμως για σύντομο χρονικό διάστημα. Το πεδίο XID πρέπει να επιλέγεται τυχαία ώστε να αποφεύγονται διπλότυπα XIDs στα αιτήματα στη περίπτωση που οι UAs επανεκκινούν συχνά.

4.2.6 SLP Μηνύματα

Όλα τα πεδία στα SLP μηνύματα είναι σε διάταξη ανά byte δικτύου (network byte order). Όταν καθοριστούν δυνάδες bytes, αυτά αποτελούν ακολουθίες από bytes με ακριβή σειρά τοποθετημένα και πάλι σε network byte order.

Όλα τα SLP μηνύματα ξεκινούν με την παρακάτω επικεφαλίδα:



Σχήμα 4.8 : Επικεφαλίδα SLP μηνυμάτων

Τα SLP μηνύματα διαχωρίζονται και έχουν ένα αντίστοιχο αναγνωριστικό αριθμό (ID). Ο συγκεκριμένος αριθμός αποτελεί μέρος της επικεφαλίδας του SLP μηνύματος και τοποθετείται στη θέση του πεδίου Function-ID. Οι αντίστοιχες τιμές παρουσιάζονται στο παρακάτω Σχήμα 4.9:

Message Type	Abbreviation	Function-ID
Service Request	SrvRqst	1
Service Reply	SrvRply	2
Service Registration	SrvReg	3
Service Deregister	SrvDeReg	4
Service Acknowledge	SrvAck	5
Attribute Request	AttrRqst	6
Attribute Reply	AttrRply	7
DA Advertisement	DAAdvert	8
Service Type Request	SrvTypeRqst	9
Service Type Reply	SrvTypeRply	10
SA Advertisement	SAAadvert	11

Σχήμα 4.9 : Διαχωρισμός μηνυμάτων SLP και αντίστοιχα Function-IDs

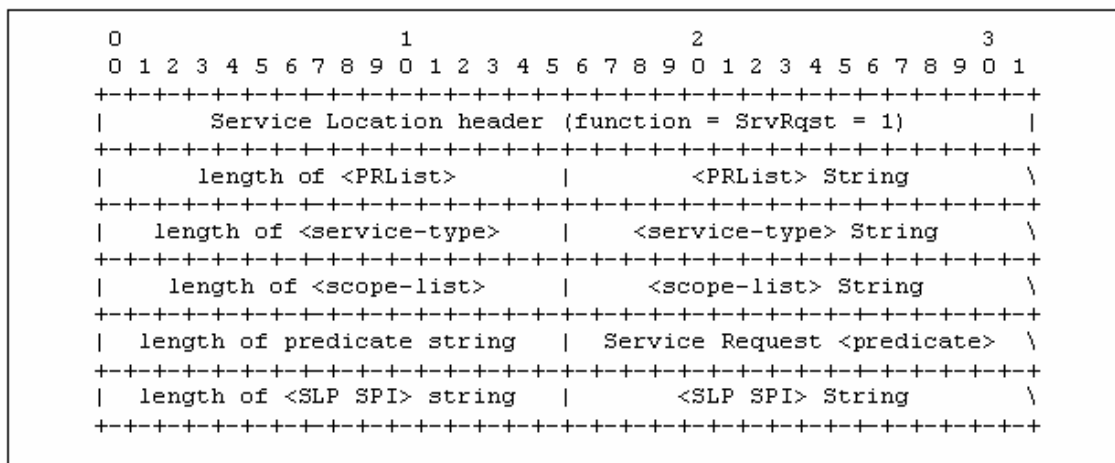
Οι SAs και οι UAs πρέπει να υποστηρίζουν μηνύματα τύπου SrvRqst, SrvRply και DAAdvert. Οι SAs πρέπει επίσης να υποστηρίζουν τα μηνύματα SrvReg, SAAadvert και SrvAck. Για τους UAs και τους SAs, η υποστήριξη άλλων μηνυμάτων είναι προαιρετική.

- Length είναι το μήκος ολόκληρου του SLP μηνύματος συμπεριλαμβανομένου και της επικεφαλίδας.
- Οι flags είναι: OVERFLOW (0x80), τίθεται στο 1 όταν το μήκος του μηνύματος υπερβαίνει το πλαίσιο. FRESH (0x40), τίθεται στο 1 κάθε νέο SrvReg. REQUEST MCAST (0x20), τίθεται στο 1 όταν υπάρχουν multicasting ή broadcasting αιτήματα. Τα Reserved bits πρέπει να είναι 0.

- Το Next Extension Offset τίθεται στο 0 εκτός αν χρησιμοποιούνται επεκτάσεις. Η πρώτη επέκταση ξεκινάει από τα 'offset' bytes, από την έναρξη του μηνύματος. Τοποθετείται μετά από τα δεδομένα του SLP μηνύματος.
- XID τίθεται σε μία μοναδική τιμή για κάθε ξεχωριστή αίτηση. Εάν η αίτηση επαναμεταδίδεται, χρησιμοποιείται το ίδιο XID. Οι απαντήσεις θέτουν το XID στην ίδια τιμή που έχει και το αντίστοιχο αίτημα. Μόνο αυτόκλητα DAAdverts στέλνονται με XID 0.
- Lang Tag Length είναι το μήκος σε bytes του πεδίου Language Tag.
- Language Tag είναι η γλώσσα των μηνυμάτων. Το Language Tag σε μια απάντηση πρέπει να είναι το ίδιο με το Language Tag στο αρχικό αίτημα. Αυτό το πεδίο κωδικοποιείται με 1*8ALPHA *("-" 1*8ALPHA).

Εάν μια επιλογή ορίζεται και δεν συμπεριλαμβάνεται στο μήνυμα, ο παραλήπτης πρέπει να απαντήσει με PARSE_ERROR.

Service Request



Σχήμα 4.10 : Service Request

Μία υπηρεσία, για να ταιριάζει ένα μήνυμα SrvRqst, πρέπει να ανήκει το λιγότερο σε ένα scope αίτησης και να υποστηρίζει το τύπο της υπηρεσίας που αναζητείται.

Η <PRList> είναι η προηγούμενη αποκρινόμενη λίστα. Αυτή η <string-list> περιέχει IP διευθύνσεις (v4) και στέλνεται επαναληπτικά με multicast εκπομπή για να συλλέξει όλα τα δυνατά αποτελέσματα. Οι UAs πρέπει να υλοποιούν αυτό τον αλγόριθμο ανακάλυψης. Οι SAs πρέπει επίσης να χρησιμοποιούν αυτή τη λίστα για να ανακαλύψουν τους διαθέσιμους DAs στην εμβέλειά τους, εάν δεν έχουν επικοινωνία με τις διευθύνσεις των DAs με κάποιο άλλο τρόπο.

Ένας SA απορρίπτει όλα τα αιτήματα τα οποία περιλαμβάνουν τη δικιά του διεύθυνση στη <PRList>. Ένας SA που έχει πολλές δικτυακές επαφές, πρέπει να ελέγχει όλες τις εγγραφές της <PRList> εάν ταιριάζουν με κάποια από τις διεπαφές του.

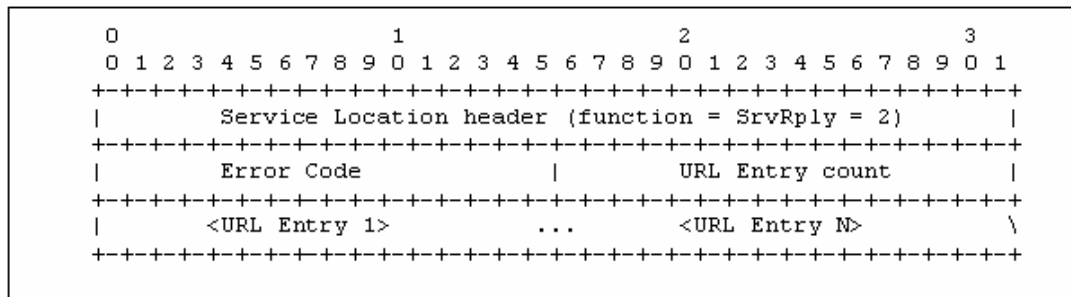
Η <scope-list> είναι μια <string-list> από διαμορφωμένα ονόματα scopes. Οι SAs και οι DAs που έχουν σχηματιστεί με όποια από τα scopes της λίστας, θα απαντήσουν στα αιτήματα.

Η συμβολοσειρά <service-type> συζητήθηκε στην ενότητα. 4.2.3. Συνήθως, ένα μήνυμα SrvRqst προκαλεί ένα μήνυμα SrvRply. Υπάρχουν δύο εξαιρέσεις: Εάν το <service-type> τεθεί στο "service:directory-agent", οι DAs αποκρίνονται στο SrvRqst με ένα μήνυμα DAAdvert. Εάν τεθεί στο "service:service-agent", οι SAs αποκρίνονται με ένα SAAdvert. Αν παραλείπεται αυτό το πεδίο, επιστρέφεται ένα PARSE_ERROR – καθώς είναι απαραίτητο πεδίο.

Το πεδίο <predicate> είναι ένα φίλτρο αναζήτησης, ένα Lightweight Directory Access Protocol Version 3 (LDAPv3) και είναι προαιρετικό. Οι υπηρεσίες ανακαλύπτονται απλά από τον τύπο και το scope τους. Αλλιώς, ανακαλύπτονται όταν ικανοποιούν το πεδίο <predicate>. Οι DAs, όπως αναφέραμε και πιο πάνω, ανακαλύπτονται όταν σταλθεί ένα SrvRqst με τον τύπο της υπηρεσίας να είναι "service:directory-agent". Στη περίπτωση όμως που στο μήνυμα SrvRqst περιλαμβάνεται το predicate, ο DA απαντά μόνο αν τα χαρακτηριστικά του DA ικανοποιούν το φίλτρο.

Τέλος, το string <SLP SPI> υποδηλώνει ένα SLP Security Parameter Index (SPI) που ο αιτών έχει διαμορφωθεί με αυτό. Αν παραλείπεται αυτό το string, ο αποκρινόμενος κόμβος δεν περιλαμβάνει κάποιο block πιστοποίησης στην απάντησή του. Εάν πάλι περιλαμβάνεται, ο αποκρινόμενος πρέπει να επιστρέψει απάντηση με κάποιο block πιστοποίησης που να σχετίζεται με το SLP SPI του SrvRqst. Σε περίπτωση που δεν επιστρέφονται απαντήσεις λόγω μη υποστήριξης του SLP SPI, ο αποκρινόμενος επιστρέφει ένα σφάλμα AUTHENTICATION_UNKNOWN.

Service Reply



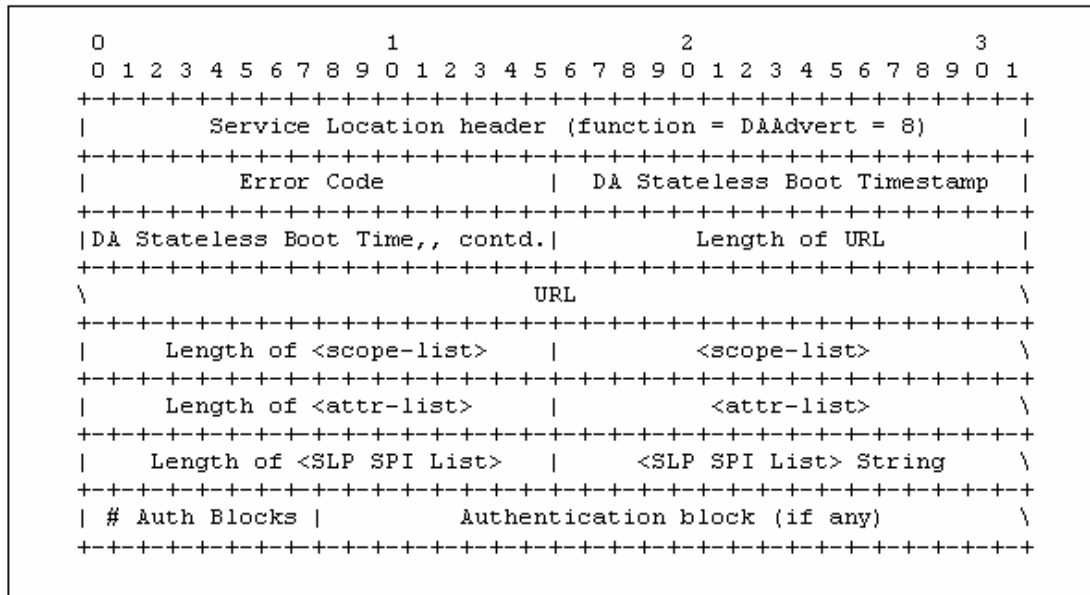
Σχήμα 4.11 : Service Reply

Το μήνυμα service reply περιέχει μηδέν ή περισσότερα URL entries. Ένα service reply με 0 URL entries πρέπει να στέλνεται ως απάντηση σε ένα unicast μήνυμα Service Request, εάν κανένα URL δε ταιριάζει. Ένα service reply με 0 URL entries δεν πρέπει να στέλνεται ως απάντηση σε multicast ή broadcast αιτήματα (αντιθέτως, αν δεν βρεθεί ταιριασμα των στοιχείων, το μήνυμα service reply δεν πρέπει να δημιουργείται καν).

Όταν ένας UA λάβει μια απάντηση, αποθηκεύει την πληροφορία της κάθε URL εγγραφής για όσο χρόνο ορίζει είτε το Lifetime είτε ένας URL Authenticator block.

Στη περίπτωση που ένα SrvRply στέλνεται με UDP, το URL Entry δεν πρέπει να συμπεριλαμβάνεται εκτός και αν το μήνυμα χωράει ολόκληρο χωρίς κατακερματισμό.

Directory Agent Advertisement



Σχήμα 4.12 : Directory Agent Advertisement

Το πεδίο Error Code τίθεται στο 0 όταν το μήνυμα DAAdvert στέλνεται multicast. Εάν το DAAdvert επιστρέφεται εξαιτίας ενός unicast SrvRqst (π.χ. ένα αίτημα χωρίς να έχει ενεργοποιημένη τη σημαία REQUEST MCAST), ο DA επιστρέφει τα ίδια λάθη που θα επέστρεφε και το SrvRply.

Ο DA μπορεί να περιλάβει μια λίστα των χαρακτηριστικών του στο μήνυμα DAAdvert. Αυτή η λίστα πρέπει να κρατηθεί μικρή καθώς το DAAdvert πρέπει να χωράει σε ένα πλαίσιο αν σκοπεύει να εκπεμφθεί multicast.

Το URL είναι "service:directory-agent://" <addr> of the DA, όπου το <addr> είναι η διεύθυνση του DA. Η <scope-list> του DA δεν πρέπει να είναι κενή.

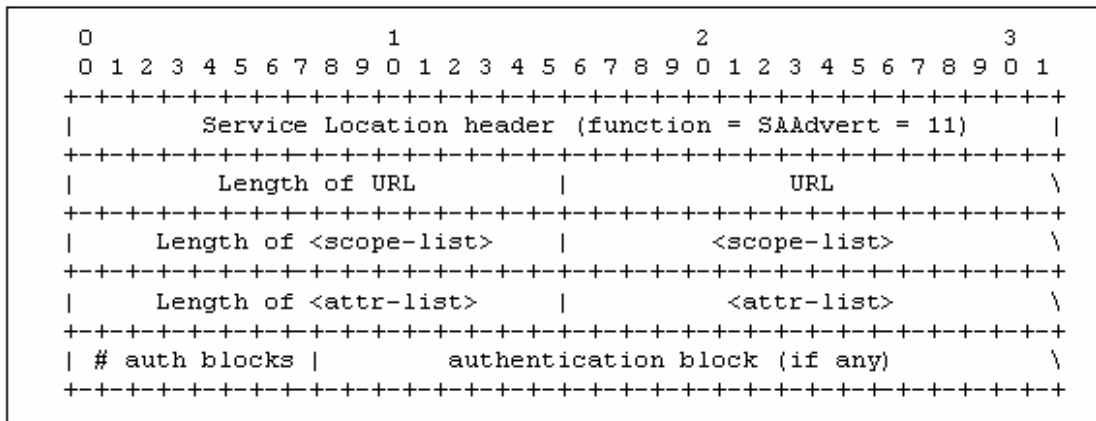
Η SLP SPI List είναι λίστα με τα SPIs τα οποία ο DA είναι ικανός να επικυρώσει. Οι SAs δεν πρέπει να εγγράφουν υπηρεσίες για τα SLP SPIs που δεν είναι στην παραπάνω λίστα. Οι DAs θα απορρίπτουν τις εγγραφές που δεν μπορούν να επικυρώσουν, επιστρέφοντας ένα μήνυμα λάθους AUTHENTICATION_UNKNOWN.

Εάν ένα DAAdvert στέλνεται ως απάντηση σε ένα SrvRqst, το DAAdvert περιέχει μόνο το block πιστοποίησης με το SLP SPI του SrvRqst. Σε περίπτωση που το SrvRqst στέλνεται unicast προς τον DA και περιλαμβάνεται ένα μη υποστηριζόμενο SLP SPI, ο DA απαντά με ένα μήνυμα DAAdvert που έχει το πεδίο Error Code με τιμή AUTHENTICATION_UNKNOWN error.

Οι UAs πρέπει να σχηματίζονται έχοντας SLP SPIs τα οποία θα τους επιτρέπουν να επικυρώνουν DA Advertisements, αλλιώς να απορρίπτουν τέτοια διαφημιστικά μηνύματα.

Service Agent Advertisement

Οι User Agents δεν πρέπει να προκαλούν την αποστολή SA Advertisements εάν έχουν σχεδιαστεί να χρησιμοποιούν ένα συγκεκριμένο DA, ή εάν χρησιμοποιούν μια <scope-list> ή αν έχουν ανακαλύψει κάποιον DA. Οι UAs ζητούν να τους αποστέλλονται SA Advertisements μόνο όταν είναι ρητά σχεδιασμένοι να χρησιμοποιούν το User Selectable scopes ώστε να ανακαλύψουν scopes που υποστηρίζουν οι SAs.



Σχήμα 4.13 : Service Agent Advertisement

Το SAAadvert μπορεί να περιλαμβάνει μια λίστα με τα χαρακτηριστικά που υποστηρίζει ο SA. Αυτή η λίστα πρέπει να είναι μικρού μήκους ώστε το μήνυμα SAAadvert να μην υπερβαίνει το MTU.

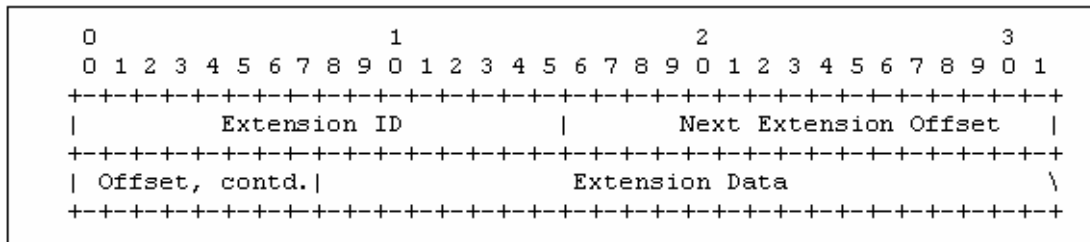
Το URL που χρησιμοποιείται είναι το "service:service-agent://" <addr> of the SA, όπου <addr> είναι η διεύθυνση του SA. Όπως και στους DAs, η <scope-list> δεν πρέπει να είναι κενή.

Τέλος, το SAAadvert περιέχει ένα SAAadvert Authentication block για κάθε SLP SPI. Αν ο UA μπορεί να επικυρώσει το block πιστοποίησης του SAAadvert αλλά το SAAadvert αποτυγχάνει τελικά να επικυρωθεί, τότε ο UA το απορρίπτει.

4.2.7 Service Location Protocol Extensions

Οι επεκτάσεις δεν είναι υποχρεωτικές στα απλά IP δίκτυα. Είναι όμως χρήσιμες σε περιπτώσεις διαλογικής χρήσης του πρωτοκόλλου (interactive use) όπου ο χρήστης κυρίως, και όχι κάποιο πρόγραμμα, επιλέγει την υπηρεσία μέσα από περιβάλλοντα φυλλομέτρησης (browsing interface). Επιπλέον, οι επεκτάσεις προσδίδουν μια κλιμάκωση στο SLP.

Η μορφή ενός Service Location Extension είναι η εξής:



Σχήμα 4.14 : Service Location Extension

Τα Extension IDs προσδιορίζονται με τον παρακάτω τρόπο:

- 0x0000-0x3FFF Τυποποιημένο. Προαιρετικό για υλοποίηση. Αγνοείται αν δεν αναγνωρίζεται.
- 0x4000-0x7FFF Τυποποιημένο. Υποχρεωτικό για υλοποίηση. Ένας UA ή SA που λαμβάνει μια απάντηση και έχει αυτή την επιλογή την οποία δεν καταλαβαίνει, πρέπει να απορρίπτει το μήνυμα. Ένας DA ή SA που λαμβάνει ένα αίτημα με αυτή την επιλογή εντός, την οποία δεν τη καταλαβαίνει, πρέπει να στέλνει μήνυμα λάθους OPTION_NOT_UNDERSTOOD.
- 0x8000-0x8FFF Για ιδιωτική χρήση (μη τυποποιημένο). Προαιρετικό για υλοποίηση. Αγνοείται αν δεν αναγνωρίζεται.
- 0x9000-0xFFFF Δεσμευμένο.

Το πεδίο, μήκους τριών byte, «offset to next extension», δηλώνει τη θέση της επόμενης επέκτασης ως offset από την αρχή του SLP μηνύματος. Η τιμή offset είναι 0 αν δεν υπάρχουν επεκτάσεις που να ακολουθούν την παρούσα επέκταση.

Εάν το offset είναι 0, το μήκος του πεδίου Extension Data καθορίζεται από το συνολικό μήκος του SLP μηνύματος, όπως αυτό δίνεται στην επικεφαλίδα SLP, μείον το offset της τρέχουσας επέκτασης.

4.3 SLP ΣΕ AD HOC ΔΙΚΤΥΑ

Το Service Location Protocol, όπως έχει οριστεί στο RFC2608, δε μπορεί να λειτουργήσει αποδοτικά στα Mobile Ad Hoc Network (MANET). Χρειάζεται κάποιες τροποποιήσεις και προσθήκες διατηρώντας ωστόσο βασικά στοιχεία του πρωτοκόλλου.

Παρακάτω, θα παρουσιάσουμε το Service Location Protocol για MANET (SLPManet), που είναι το γνωστό μας SLP προσαρμοσμένο για λειτουργία στα Ad Hoc δίκτυα. Το SLPManet υλοποιεί όλα τα απαραίτητα στοιχεία του SLP Version 2. Όλα τα στοιχεία που δεν υλοποιούνται στο SLPManet, περιγράφονται ως προαιρετικά στο RFC του SLP και έμειναν εκτός επειδή δε ταιριάζουν σε περιβάλλοντα κινητών δικτύων. Τα πιο αξιοσημείωτα σημεία που έχουν εξαιρεθεί είναι:

- *Directory Agents (DAs)*: Οι Directory Agents, όπως τους έχουμε περιγράψει, είναι κεντρικές μονάδες αποθήκευσης πληροφοριών των υπηρεσιών του δικτύου. Οι

Directory agents υπάρχουν σε δίκτυα LANs ώστε να επαυξήσουν την απόδοση και να προσδώσουν μια κλιμάκωση στο πρωτόκολλο. Στα MANETs δεν υπάρχουν συνήθως DAs καθώς τέτοια δίκτυα λειτουργούν χωρίς την ύπαρξη κόμβων που προϋπάρχουν ή είναι μόνιμα παρόντες. Σαν συνέπεια αυτής της απουσίας, μόνο active discovery επιτελείται.

- *Authentication Blocks (Blocks Πιστοποίησης)*: Οι Agents ούτε είναι διαμορφωμένοι ώστε να παράγουν blocks πιστοποίησης ούτε τα επικυρώνουν. Αυτό το προαιρετικό στοιχείο χρειάζεται για να αποτρέπει των έλεγχο υπηρεσιών από εχθρικούς κόμβους. Στο SLPManet δεν υλοποιείται καθώς η ασφάλεια δεν είναι πρωταρχικός στόχος.

- *Optional SLP messages*: Μηνύματα όπως τα Attribute Request και Service Type Request δεν υλοποιούνται στο SLPManet. Και αυτό γιατί τέτοια επιπλέον στοιχεία προσθέτουν πολυπλοκότητα και καταναλώνουν αρκετές πηγές ενέργειας του δικτύου.

Να τονίσουμε ξανά ότι σε άλλες προσαρμογές του SLP για κινητά δίκτυα, τα παραπάνω στοιχεία μπορεί να υλοποιηθούν. Απλά εδώ παρουσιάζουμε μία τροποποίηση του SLP, το SLPManet. Στη συνέχεια, θα ξανασυζητήσουμε τα βασικά στοιχεία του SLP όπως τα παρουσιάσαμε στην ενότητα 4.2 υποδεικνύοντας τις αλλαγές.

Protocol Overview

Λόγω της απουσίας των DAs, η διαδικασία αποστολής μηνυμάτων αίτησης και απάντησης γίνεται απευθείας μεταξύ των User Agents και των Service Agents. Ο UA εκπέμπει ένα μήνυμα 'Service Request' (SrvRqst) για λογαριασμό του πελάτη, ορίζοντας τα χαρακτηριστικά της υπηρεσίας που αναζητά ο πελάτης. Οι Service Agents που λαμβάνουν μια αίτηση για μια υπηρεσία την οποία διαφημίζουν, εκπέμπουν μόνο προς τον ενδιαφερόμενο κόμβο μία απάντηση που περιέχει την τοποθεσία της εν λόγω υπηρεσίας. Εκτελείται δηλαδή η διαδικασία του σχήματος 4.3.

URLs, χρήση των Ports, Επαναμετάδοση SLP μηνυμάτων

Η χρησιμοποίηση των URL Entries στα MANETs γίνεται με τον ίδιο ακριβώς τρόπο που περιγράψαμε για τα IP δίκτυα. Το ίδιο συμβαίνει και για την χρήση των θυρών και της UDP ή TCP μεταφοράς δεδομένων. Επίσης, η περίπτωση της επαναμετάδοσης μηνυμάτων λειτουργεί χωρίς παραλλαγές.

SLPManet Μηνύματα

Διατηρώντας την ίδια επικεφαλίδα, τα μηνύματα που χρησιμοποιεί η υλοποίηση SLPManet είναι τρία:

Message Type	Abbreviation	Function-ID
Service Request	SrvRqst	1
Service Reply	SrvRply	2
SA Advertisement	SAAdvert	11

Σχήμα 4.15 : SLP Μηνύματα σε ad hoc δίκτυα

Η ανάλυση των πεδίων αυτών των μηνυμάτων έγινε στην ενότητα 4.2.6. Η μόνη διαφορά είναι πως δεν έχουμε μηνύματα DAAdvert ως απάντηση σε κάποιο μήνυμα SrvRqst.

4.4 ΕΠΙΛΟΓΗ ΛΕΙΤΟΥΡΓΙΑΣ SLP ΓΙΑ AD HOC ΔΙΚΤΥΑ

Παραπάνω παρουσιάστηκε η λειτουργία του πρωτοκόλλου και για τη centralized και για τη distributed μορφή του. Στο επόμενο κεφάλαιο, θα συγκριθούν αυτές οι δύο μορφές όταν εφαρμοστούν σε ad hoc δίκτυα.

ΚΕΦΑΛΑΙΟ 5

5 ΣΥΓΚΡΙΣΗ CSLP ΚΑΙ DSLP

5.1 ΕΙΣΑΓΩΓΗ

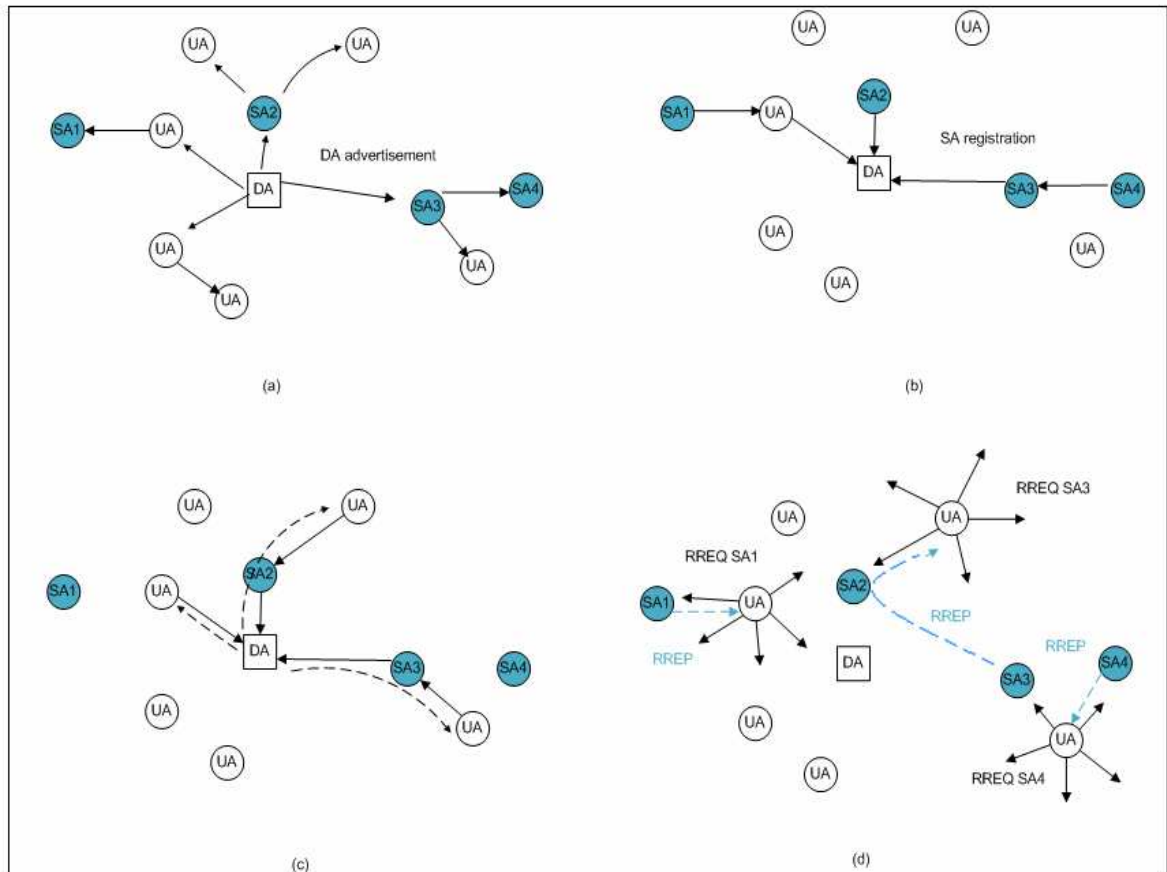
Στο πέμπτο αυτό κεφάλαιο θα συγκρίνουμε τα αποτελέσματα της υλοποίησης του SLP στις περιπτώσεις της κεντρικής (centralized) και της κατανεμημένης (distributed) εκδοχής. Η προτεινόμενη προς υλοποίηση αρχιτεκτονική του SLP, χρησιμοποιεί τον Ad hoc On Demand Distance Vector (AODV) [17], [18] αλγόριθμο δρομολόγησης στο επίπεδο δικτύου. Για την αποτίμηση της απόδοσης της αρχιτεκτονικής και για τις δύο εκδοχές της, εξετάστηκαν κριτήρια όπως η αναλογία των υπάρχοντων SA σε σχέση με το σύνολο των κινητών κόμβων (service hit ratio), ο χρόνος ανακάλυψης υπηρεσίας και το κόστος των μηνυμάτων ελέγχου (control message overheads).

5.2 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΓΙΑ ΑΝΑΚΑΛΥΨΗ ΥΠΗΡΕΣΙΑΣ ΣΕ MANET

Η παρακάτω αρχιτεκτονική για ανακάλυψη υπηρεσίας σε MANET, βασίζεται όπως αναφέραμε σε on-demand αλγόριθμους δρομολόγησης. Η αρχιτεκτονική ενσωματώνει τόσο centralized όσο και distributed δομές.

5.2.1 Centralized Service Location Scheme (CSLP)

Η κεντρική έκδοση της αρχιτεκτονικής (CSLP) αποτελείται και από τις τρεις οντότητες, User Agent (UA), Service Agent (SA) και Directory Agent (DA) όπως τις περιγράψαμε στο Κεφ. 4. Το Σχήμα 5.1 δείχνει την όλη διαδικασία ανακάλυψης στο CSLP.



Σχήμα 5.1 : Διαδικασία ανακάλυψης υπηρεσίας στο CSLP

Όπως φαίνεται στο Σχήμα 5.1.a, ο DA αποθηκεύει πληροφορίες σχετικά με τις διαθέσιμες υπηρεσίες των εγγεγραμμένων SAs και «διαφημίζει» τον εαυτό του περιοδικά στέλνοντας πολλαπλά (multicast) ή καθολικά (broadcast) μηνύματα. Σε αυτή την αρχιτεκτονική, υιοθετούμε τη broadcast εκπομπή αντί της multicast ώστε να μειώσουμε το overhead της multicast πληροφορίας. Κάθε κινητός κόμβος, (UA ή SA) καταγράφει την πληροφορία για τους διαθέσιμους DAs και παράλληλα διατηρεί σε πίνακες δρομολόγησης τα μονοπάτια προς αυτούς τους DAs.

Οι SAs «ακούνε» στα broadcasted διαφημιστικά μηνύματα των DAs. Οι UAs, όταν λάβουν διαφημιστικό μήνυμα από κάποιον DA, μπορούν να επιλέξουν τον τρόπο ανακάλυψης υπηρεσίας είτε σε centralized είτε σε distributed μορφή. Εάν ο UA επιλέξει το centralized scheme, τότε εκπέμπει ένα μήνυμα SrvRqst για να ρωτήσει απευθείας τον συγκεκριμένο DA για συγκεκριμένες υπηρεσίες. Έτσι δε χρειάζεται να στείλει καθολικά το SrvRqst μήνυμα σε μια broadcast περιοχή. Στη συνέχεια ο DA ελέγχει τη μνήμη του με τις εγγεγραμμένες πληροφορίες για την συγκεκριμένη υπηρεσία και θα απαντήσει στον UA με ένα unicast SrvRply μήνυμα.

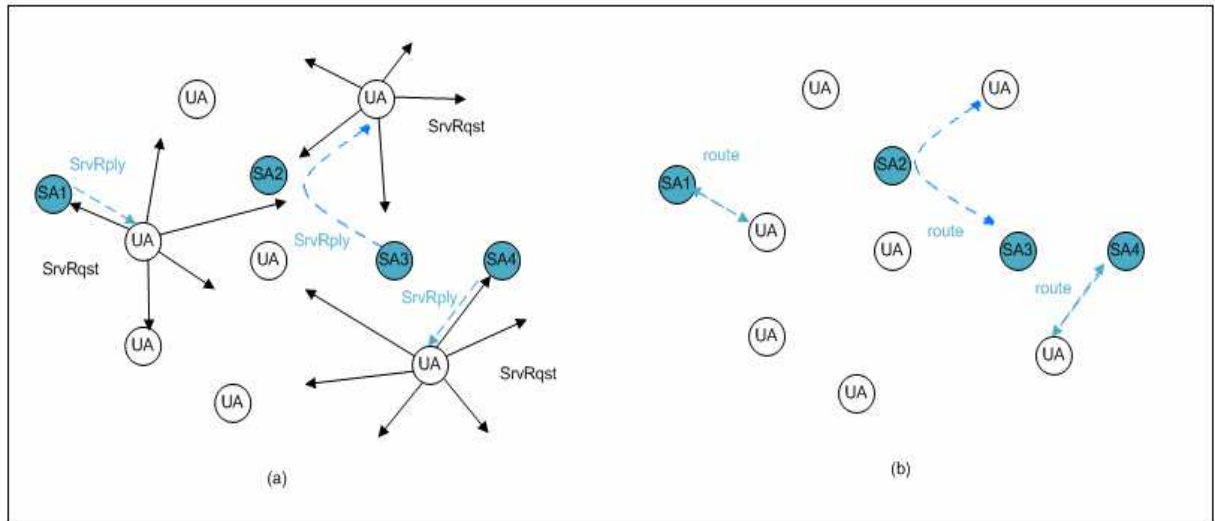
Στο Σχήμα 5.1.b, ένας κόμβος SA στέλνει μηνύματα SrvReg για να εγγραφεί την τοποθεσία του και τα χαρακτηριστικά των υπηρεσιών που παρέχει. Η εγγραφή αυτή στον DA ανανεώνεται περιοδικά. Τα μηνύματα SrvReg περιέχουν πληροφορία όπως η IP διεύθυνση και ο τύπος της υπηρεσίας.

Στο Σχήμα 5.1.c βλέπουμε πως όταν ένας UA χρειάζεται μια εφαρμογή – υπηρεσία, δημιουργεί ένα μήνυμα αίτησης SrvRqst και το στέλνει στον DA για να ανακαλύψει την τοποθεσία αυτή της εφαρμογής. Τότε ο DA ανταποκρίνεται

στέλνοντας μήνυμα SrvRply που περιλαμβάνει τις διευθύνσεις των SAs των οποίων οι υπηρεσίες ταιριάζουν στην αίτηση του UA. Τώρα ο UA μπορεί να έχει πρόσβαση σε μία από αυτές τις υπηρεσίες που τον ικανοποιούν. Να σημειώσουμε ότι ο UA γνωρίζει μόνο τη διεύθυνση των SAs και όχι το μονοπάτι προς αυτούς. Συνεπώς, πρέπει να εγκαθιδρύσει μονοπάτια δρομολόγησης προς τους SAs. Στο Σχήμα 5.1.d φαίνεται αυτή η διαδικασία πρόσβασης των SAs. Αυτή η διαδικασία είναι αναπόφευκτη στα MANETs που βασίζονται σε on-demand αλγόριθμους δρομολόγησης. Σε αυτούς τους αλγόριθμους, πρέπει να ανακαλυφθεί πρώτα μία διαδρομή πριν αρχίσει μία ανταλλαγή δεδομένων. Να υπενθυμίσουμε ότι στα σταθερά δίκτυα, αλλά ακόμα και στα MANETs, με proactive αλγόριθμους δρομολόγησης, δεν χρειάζονται αυτά τα βήματα ανακάλυψης διαδρομής. Αυτή η επιπλέον διαδικασία σπαταλά πολύτιμο εύρος ζώνης και ενεργειακών αποθεμάτων των κόμβων ενώ παράλληλα προκαλεί παραπάνω κίνηση στο δίκτυο.

5.2.2 Distributed Service Location Scheme (DSLP)

Στο DSLP, μόνο οι δύο οντότητες, οι UAs και SAs, περιλαμβάνονται. Όταν ένας UA επιχειρεί να χρησιμοποιήσει μια υπηρεσία, της οποίας ο σχηματισμός και η παρουσία δεν είναι ακόμα γνωστή, ξεκινά μια διαδικασία ανακάλυψης υπηρεσίας. Ο UA εκπέμπει broadcast ένα μήνυμα SrvRqst στο οποίο ορίζει τον τύπο της επιθυμητής υπηρεσίας. Το μήνυμα αυτό πλημμυρίζεται σε μια προκαθορισμένη broadcast περιοχή. Όταν ένα SrvRqst καταφθάσει σε έναν SA, που βρίσκεται εντός της broadcast περιοχής, και ο οποίος να παρέχει την υπηρεσία που αναζητείται, τότε απαντάει στον χρήστη UA με ένα μήνυμα SrvRply που περιέχει την πληροφορία για τη διαμόρφωση της υπηρεσίας. Ο UA μπορεί να λάβει πολλά SrvRplies από διαφορετικούς SAs και εξαρτάται πλέον από αυτόν να διαλέξει το καλύτερο μήνυμα. Ένας ενδιάμεσος κόμβος που λαμβάνει ένα SrvRply, διατηρεί στη μνήμη του το μονοπάτι προς τον SA για μετέπειτα χρήση της υπηρεσίας από τον UA. Στο Σχήμα 5.2.a φαίνεται η διαδικασία καθολικής εκπομπής ενός SrvRqst και η λήψη απαντήσεων SrvRply(s). Το Σχήμα 5.2.b δείχνει τα μονοπάτια δρομολόγησης μεταξύ UAs και SAs.



Σχήμα 5.2 : Διαδικασία ανακάλυψης υπηρεσίας στο DSNP

5.3 ΑΠΟΤΙΜΗΣΗ ΑΠΟΔΟΣΗΣ ΠΡΩΤΟΚΟΛΛΟΥ

Στην ενότητα αυτή θα παρουσιάσουμε τα αποτελέσματα για τις δύο εκδοχές του SLP κατά τη διαδικασία ανακάλυψης υπηρεσίας.

5.3.1 Προσομοίωση με χρήση του NS-2

Στις προσομοιώσεις χρησιμοποιήθηκε ο Network Simulator Version 2 (NS-2) που έχει αναπτυχθεί από το project Virtual InterNetwork Testbed (VINT) στο πανεπιστήμιο Berkley της Καλιφόρνιας [19]. Ο NS-2 είναι ένας ιδιαίτερος προσομοιωτής γεγονότων ανοιχτού κώδικα που στοχεύει στην έρευνα δικτύων. Παρέχει πλήρη υποστήριξη για προσομοιώσεις του πρωτοκόλλου Transmission Control Protocol (TCP), δρομολόγησης, πρωτοκόλλων πολλαπλής εκπομπής σε ενσύρματα και ασύρματα (τοπικά και δορυφορικά) δίκτυα. Εργαλεία που υποστηρίζουν ασύρματα δίκτυα πολλαπλών αλμάτων με μοντέλα για το φυσικό επίπεδο, το επίπεδο συνδέσμου μετάδοσης δεδομένων καθώς και για το επίπεδο Media Access Control (MAC), παρέχονται από την ομάδα ερευνών Monarch του πανεπιστημίου Carnegie Mellon.

Ο Network Simulator 2 είναι ένας αντικειμενοστραφής προσομοιωτής δικτύων γραμμένος σε C++ και OTcl (η αντικειμενοστραφής έκδοση της γλώσσας Tcl). Στον NS-2 είναι ενσωματωμένα τα γνωστότερα δικτυακά πρωτόκολλα όπως για παράδειγμα τα πρωτόκολλα TCP και UDP, πρωτόκολλα εφαρμογής (FTP, Telnet, Web, CBR), προσομοιώσεις διαχείρισης ουρών δρομολογητών (DropTail, RED), δρομολόγηση κ.λ.π.

Οι προγραμματιστές του NS-2 προκειμένου να ισορροπούν ανάμεσα στην υψηλού επιπέδου υλοποίηση και στον εύκολο προγραμματισμό, χώρισαν τον προσομοιωτή σε δύο τμήματα. Το ένα τμήμα (κυρίως για τη διασύνδεση με το χρήστη) είναι γραμμένο στη γλώσσα OTcl ώστε να μη χρειάζεται ο απλός χρήστης να γνωρίζει τη δυσκολότερη αλλά πανίσχυρη C++ προκειμένου να τον χρησιμοποιήσει. Ο υπόλοιπος κώδικας του NS-2 είναι γραμμένος στη γλώσσα C++ ώστε το αποτέλεσμα να είναι υψηλού επιπέδου και να είναι αποδοτικός σε σχέση με την ταχύτητα. Ο προσομοιωτής λοιπόν, υποστηρίζει μία ιεραρχία τάξεων στη C++ και μία παρόμοια ιεραρχία τάξεων στην OTcl. Οι δύο ιεραρχίες λειτουργούν αρμονικά μεταξύ τους. Από την πλευρά του χρήστη, υπάρχει μία απευθείας αντιστοιχία μιας κλάσης στην OTcl ιεραρχία και μιας κλάσης στη C++ ιεραρχία.

Δηλαδή ένα δικτυακό πρόβλημα μπορεί να εκφραστεί μόνο με την OTcl, μόνο με τη C++ ή και με συνδυασμό των δύο. Συνήθως η παραμετροποίηση του προβλήματος γίνεται με την OTcl για μεγαλύτερη ευκολία και ταχύτητα ενώ το υπόλοιπο πρόβλημα χρησιμοποιεί κώδικα C++. Οι χρήστες δημιουργούν νέα αντικείμενα προσομοίωσης χρησιμοποιώντας σενάρια OTcl (scripts). Τα αντικείμενα της C++ έχουν αναπαρασταθεί και σαν αντικείμενα της OTcl ώστε να μπορεί αν εκφράσει κανείς το πρόβλημα και με τους δύο τρόπους.

5.3.2 Περιβάλλον προσομοίωσης

Στα πειράματα που έγιναν, παράχθηκαν διάφορα σενάρια σε κινητά ad hoc δίκτυα. Στο ΠΑΡΑΡΤΗΜΑ Α, παρατίθενται οι αλλαγές στα αρχεία του NS-2 ώστε να υποστηρίζεται το πρωτόκολλο SLP. Στο ΠΑΡΑΡΤΗΜΑ Β, παρατίθενται αναλυτική περιγραφή και ο κώδικας σε γλώσσα TCL για τα σενάρια που δοκιμάστηκαν. Το πρωτόκολλο επιπέδου MAC που χρησιμοποιείται είναι το IEEE 802.11 Distributed Coordination Function (DCF). Κάθε κόμβος διατηρεί μια ουρά προτεραιότητας που ενταμιεύει πακέτα έως ότου το MAC στρώμα τα μεταδώσει. Το μήκος της ουράς είναι 50 πακέτα. Πακέτα δρομολόγησης έχουν υψηλή προτεραιότητα και εισάγονται στη κορυφή της ουράς.

Στον NS-2, κάθε κινητός κόμβος σε επίπεδο έδαφος, χρησιμοποιεί μία ομοιοκατευθυντική κεραία με μοναδιαίο κέρδος εκπομπής και λήψης. Οι κεραίες εκπομπής και λήψης έχουν 1.5 μ. ύψος. Η ασύρματη μετάδοση γίνεται στα 914 MHz χρησιμοποιώντας το Lucent WaveLAN Direct - Sequence Spread Spectrum (DSSS). Το μοντέλο διάδοσης του σήματος συνδυάζει την διάδοση στον ελεύθερο χώρο με τη διάδοση σε επίπεδη επιφάνεια. Όταν ο πομπός είναι εντός της απόστασης αναφοράς του δέκτη, χρησιμοποιείται το μοντέλο ελεύθερου χώρου με εξασθένηση σήματος αντιστρόφως ανάλογη του τετραγώνου της απόστασης. Ειδιάλλως, χρησιμοποιείται το μοντέλο επίπεδης επιφάνειας, όπου η εξασθένηση του σήματος είναι αντιστρόφως ανάλογη της τέταρτης δύναμης της απόστασης. Η ισχύς του εκπεμπόμενου σήματος είναι περίπου 0.2818 W ενώ το κατώφλι για τη λαμβανόμενη ισχύ είναι περίπου $3.652 \cdot 10^{-10}$ W. Συνεπώς, η απόσταση εμβέλειας είναι περίπου 250 μ.

Οι κόμβοι κινούνται εντός της περιοχής υπακούοντας στο μοντέλο κινητικότητας το Random Waypoint Mobility Model [20]. Σε αυτό το μοντέλο, κάθε κόμβος κινείται μπροστά προς τυχαίο προσανατολισμό με μία μέγιστη ταχύτητα. Αφού φτάσει στον προορισμό, κάνει μια παύση προτού συνεχίσει ξανά για νέο προορισμό.

Κάθε κόμβος συμπεριφέρεται ως ένας από τους τρεις πράκτορες, UA, SA ή DA. Ο αριθμός των SAs στο δίκτυο ποικίλει ώστε να δούμε την επίδραση της

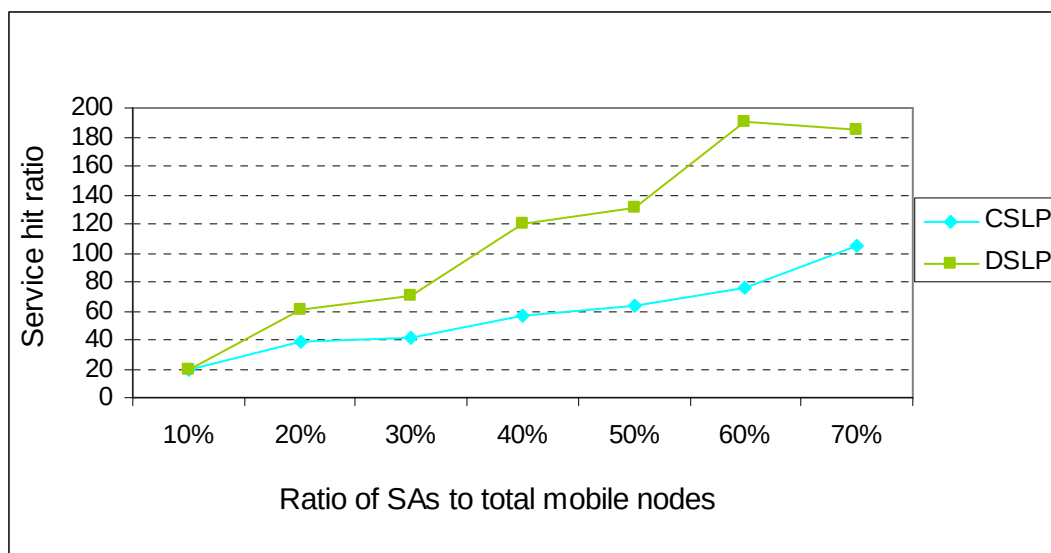
διαθεσιμότητας των υπηρεσιών στην απόδοση του service discovery. Για παράδειγμα, επιλέγοντας 5 κόμβους (από σύνολο 50) σαν SAs, η αναλογία είναι 10%. Ο συνολικός αριθμός των υπηρεσιών που καθορίζονται κατά τη διάρκεια της προσομοίωσης είναι 10, ενώ κάθε κόμβος παρέχει δύο διαφορετικές υπηρεσίες. Αν η αναλογία είναι 20% (10 κόμβοι), συνολικά 20 υπηρεσίες παρέχονται και η ίδια υπηρεσία μπορεί να προσφερθεί από διαφορετικούς SAs. Η επιλογή μιας υπηρεσίας μεταξύ πολλών στιγμιότυπων επαφίεται στον UA. Κάθε UA αιτείται μια υπηρεσία κάθε 3 δευτερόλεπτα. Οι DAs διαφημίζονται κάθε 5 δευτερόλεπτα στο CSLP.

Και για το CSLP και για το DSLP, διεξήχθησαν δύο ειδών προσομοιώσεις. Η πρώτη είναι η σύγκριση της αποδοτικότητας των δύο αρχιτεκτονικών μεταβάλλοντας την αναλογία των SAs από 10% (5 από τους 50 π.χ. κόμβους) σε 70% (35 από τους 50 π.χ. κόμβους). Η δεύτερη προσομοίωση μας δίνει την απόδοση ενώ μεταβάλλουμε το χρόνο παύσεων ώστε να δούμε την επίδραση της κινητικότητας.

5.3.3 Αποτελέσματα προσομοιώσεων

Συγκρίνουμε την απόδοση του DSLP και του CSLP μεταβάλλοντας το λόγο του αριθμού των SAs προς το συνολικό αριθμό των κινητών κόμβων. Τα αποτελέσματα της προσομοίωσης είναι ο μέσος όρος 3 ίδιων μοντέλων κίνησης αλλά διαφορετικών, τυχαία παραγόμενων, σεναρίων κινητικότητας. Τα σημεία σε κάθε γραφική παράσταση αποτελούν το μέσο όρο των 3 σεναρίων.

Τα αποτελέσματα φαίνονται στα σχήματα 5.3 – 5.6. Το Σχήμα 5.3 παρουσιάζει το service hit ratio.

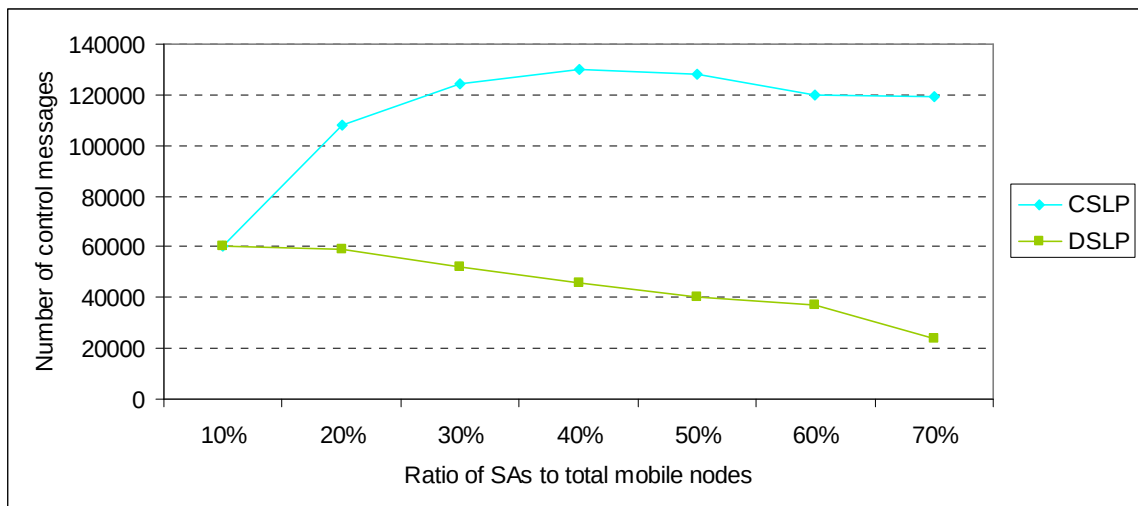


Σχήμα 5.3 : Service hit ratio

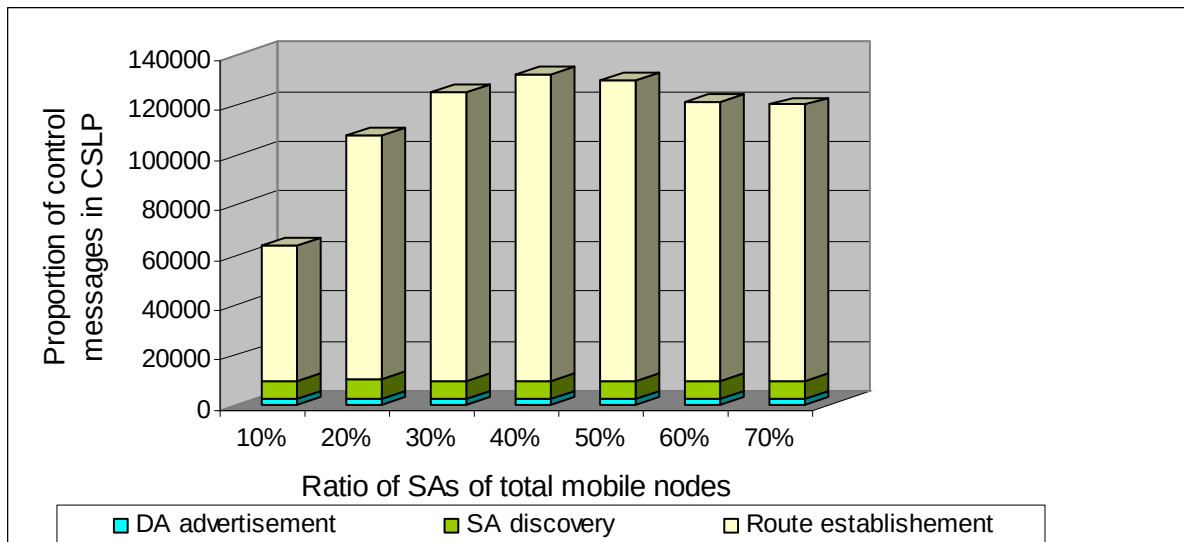
Το service hit ratio είναι ευθέως ανάλογο της αναλογίας των SAs. Αυτό συμβαίνει καθώς η πιθανότητα να υπάρχει μία ζητούμενη υπηρεσία είναι ανάλογη του λόγου των SAs. Όσοι πιο πολλοί οι πάροχοι, τόσο πιο πιθανό να βρει ο χρήστης την υπηρεσία που επιθυμεί. Αρκεί να θυμηθούμε, πως κάθε SA παρέχει δύο

διαφορετικές υπηρεσίες και ο συνολικός αριθμός των τύπων υπηρεσίας είναι 10. Ένα άλλο ενδιαφέρον σημείο είναι ότι στο DSLP, το service hit ratio γίνεται 190% για λόγο SAs 70%. Αυτό υποδηλώνει ότι κάθε UA ανακάλυψε 2 πανομοιότυπες υπηρεσίες από διαφορετικούς παρόχους. Το DSLP δείχνει σχεδόν διπλάσια απόδοση από το CSLP.

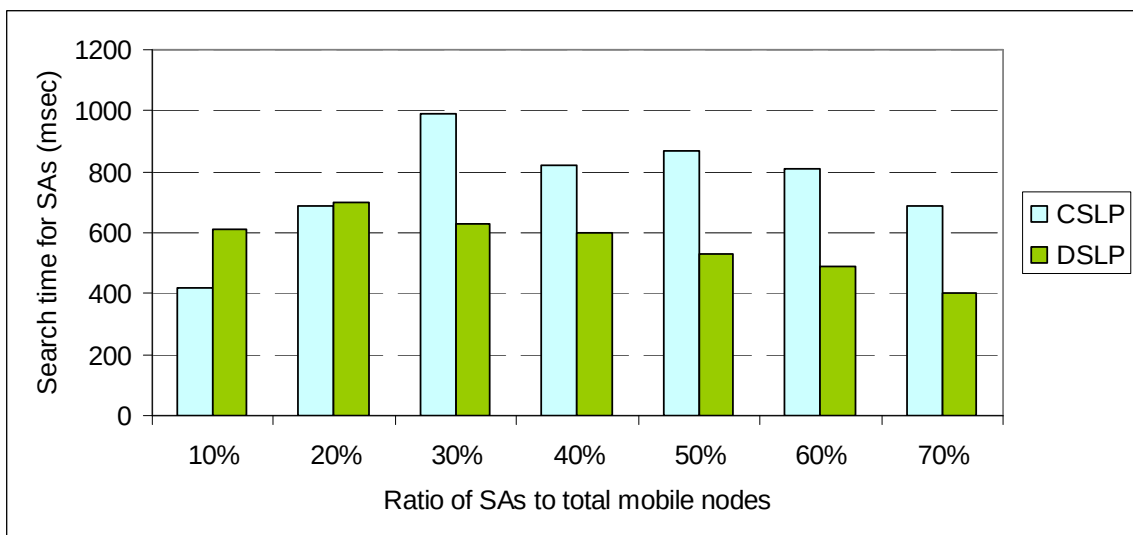
Η ανάλυση της διαφοράς στην απόδοση μπορεί να εξαχθεί κοιτώντας και τα σχήματα 5.4 – 5.6. Το σχήμα 5.4 μας δείχνει τον αριθμό των μηνυμάτων ελέγχου (control messages) που παράγονται κατά την ανακάλυψη μιας υπηρεσίας. Το CSLP παράγει σχεδόν 5 φορές πιο πολλά μηνύματα από το DSLP για ποσοστό των SAs 70%. Στο σχήμα 5.5 φαίνεται πιο λεπτομερώς η ανάλυση των μηνυμάτων ελέγχου στο CSLP. Τα μηνύματα ελέγχου στο CSLP αποτελούνται από τους εξής τρεις τύπους: διαφημιστικά μηνύματα των DAs, μηνύματα εγγραφής και απαντήσεων των SAs (SrvReg και SrvRplies) και εγκαθίδρυση διαδρομής προς τον SA μετά από την ανακάλυψη του παρόχου που προσφέρει την επιθυμητή υπηρεσία. Τα περισσότερα από τα control messages είναι για την εγκαθίδρυση των διαδρομών προς τους SAs. Και όπως αναφέραμε και στην ανάλυση της αρχιτεκτονικής, ο DA απλά δίνει στο χρήστη τη διεύθυνση του κατάλληλου παρόχου και όχι τη διαδρομή προς αυτόν. Ο UA πρέπει να βρει μόνος του τη διαδρομή προς τον SA και αυτή η διαδικασία προκαλεί αύξηση μηνυμάτων στο δίκτυο.



Σχήμα 5.4 : Αριθμός μηνυμάτων ελέγχου (control messages)



Σχήμα 5.5 : Ανάλυση μηνυμάτων ελέγχου στο CSLP



Σχήμα 5.6 : Χρόνος ανακάλυψης των SAs

Το Σχήμα 5.6 δείχνει το χρόνο που παρέρχεται για την ανακάλυψη υπηρεσίας και για τα δύο πλαίσια. Το DSLP παρέχει καλύτερη απόδοση CSLP όσο αυξάνει ο αριθμός των SAs. Αυτό συμβαίνει επειδή η δύο – φάσεων διεκπεραίωση του CSLP διαρκεί περισσότερο από τη διεκπεραίωση της διαδικασίας σε μία φάση στο DSLP. Συνεπώς, στη κατακεκομμένη αρχιτεκτονική, είναι πιο σύντομο να ανακαλύψουμε την υπηρεσία που επιθυμούμε.

5.4 ΣΥΜΠΕΡΑΣΜΑΤΑ - ΑΝΑΓΚΗ ΤΡΟΠΟΠΟΙΗΣΕΩΝ ΣΤΟ SLP

Οι παραπάνω γραφικές παραστάσεις και γενικά η σύγκριση του CSLP και του DSLP κατέδειξε πως είναι καλύτερο να εφαρμόζουμε μια αποκεντρωμένη δομή για τα κινητά περιβάλλοντα

Το Service Location Protocol, όπως έχει παρουσιαστεί για σταθερά δίκτυα, είναι μια αρχιτεκτονική ανακάλυψης υπηρεσιών που δεν παρέχει αποδοτικότητα σε κινητά ad hoc δίκτυα. Αυτό οφείλεται κυρίως στους ενεργειακούς περιορισμούς και στα φαινόμενα κινητικότητας των κόμβων. Οι centralized μηχανισμοί λειτουργούν ανασταλτικά στα MANETs, καθώς η παρουσία των Directory Agents εισάγει δυσκολίες. Η περιοδική αποστολή μηνυμάτων στους DAs για ανανέωση πληροφοριών, οδηγεί σε επιπρόσθετη κατανάλωση ενέργειας των κόμβων. Έτσι, οι κόμβοι βγαίνουν εκτός λειτουργίας του δικτύου και προκαλούν αλλαγές στη τοπολογία και στις διαδικασίες ανακάλυψης και παράδοσης των υπηρεσιών.

Κρίνεται έτσι αναγκαία η υλοποίηση του SLP για λειτουργία σε MANETs με μια αποκεντρωμένη (decentralized) προσέγγιση χωρίς την παρουσία DAs. Στο επόμενο κεφάλαιο, θα παρουσιάσουμε την πρότασή μας για το SLP που θα λειτουργεί αποδοτικά σε MANETs περιβάλλοντα.

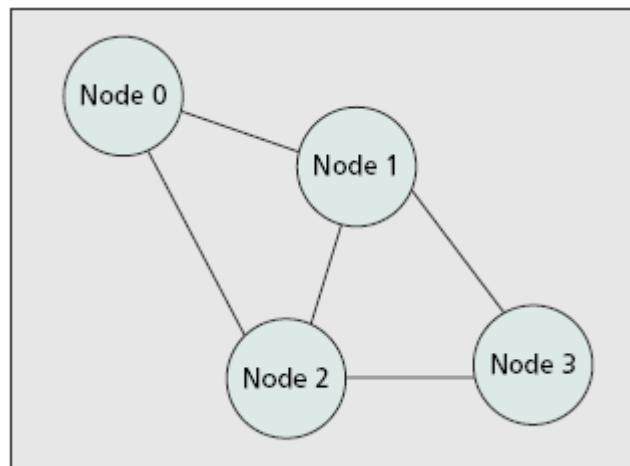
ΚΕΦΑΛΑΙΟ 6

6 ΕΠΕΚΤΑΣΗ ΤΟΥ SERVICE LOCATION PROTOCOL

6.1 ΕΙΣΑΓΩΓΗ

Στο κεφάλαιο αυτό θα αναλύσουμε τον αλγόριθμο που προτείνουμε ως λύση για διαδικασίες ανακάλυψης και παράδοσης υπηρεσίας σε κινητά ad hoc δίκτυα. Η συγκεκριμένη αρχιτεκτονική παρέχει τη δυνατότητα να επιλέξουμε έναν πάροχο υπηρεσιών λαμβάνοντας υπόψη διάφορες μετρικές όπως τα αποθέματα ενέργειας των κόμβων και το ακολουθιακό μονοπάτι από την πηγή στον προορισμό. Ο παρακάτω αλγόριθμος είναι αποκεντρωμένος και βασίζεται στις επεκτάσεις του Service Location Protocol version 2 (SLPv2).

Η συνοχή ενός δικτύου είναι σημαντικός παράγοντας στα κινητά περιβάλλοντα που μελετάμε και η οποία μπορεί να αλλάξει λόγω της κατανάλωσης ενέργειας όπως αναλύεται στο [21]. Εάν ένας κόμβος τεθεί εκτός δικτύου, εξαιτίας χαμηλής μπαταρίας, ολόκληρο το μονοπάτι από τον κόμβο πηγή στον κόμβο προορισμό πιθανόν θα αλλάξει προκαλώντας έτσι δυσκολίες στην ανακάλυψη υπηρεσιών καθώς και στην εγκαθίδρυση μιας σύνδεσης μεταξύ του παρόχου υπηρεσίας και του πελάτη που την απαιτεί. Ένα παράδειγμα που απεικονίζει αυτές τις δυσκολίες φαίνεται στο Σχήμα 6.1.



Σχήμα 6.1 : Παράδειγμα συνοχής δικτύου

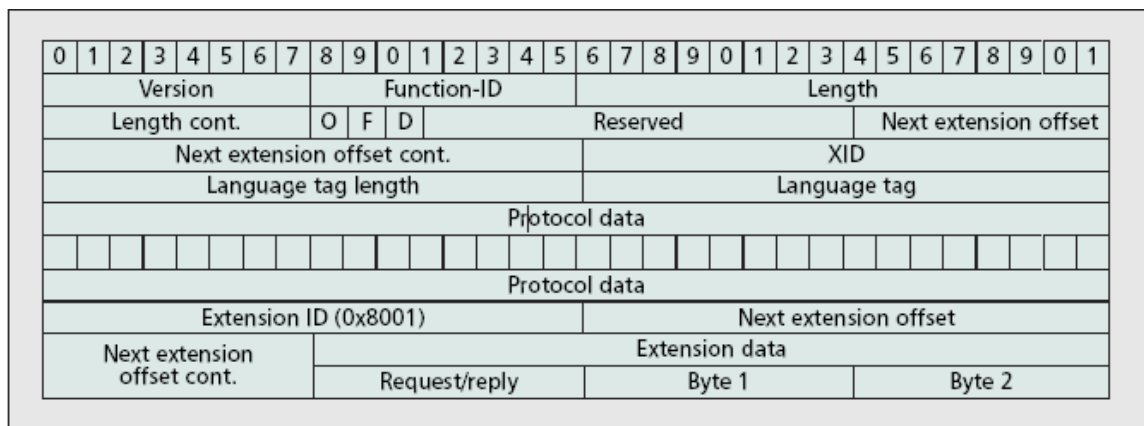
Το κινητό ad hoc δίκτυο αποτελείται από τέσσερις κόμβους και ας υποθέσουμε ότι ο κόμβος 0 αναζητεί μια υπηρεσία που μόνο ο κόμβος 3 μπορεί να την παράσχει. Εάν η υπηρεσία είναι για παράδειγμα μεταφορά βίντεο ή μια εφαρμογή πολυμέσων, η διαχείριση της καταναλισκόμενης ενέργειας είναι πολύ σημαντική, ώστε να αυξηθεί η διάρκεια ζωής του δικτύου. Έτσι, ο κόμβος 0 που αναζητά την υπηρεσία από τον 3,

πρέπει να ξέρει την υπολοιπόμενη ενέργεια όχι μόνο του κόμβου-παρόχου αλλά και όλων των ενδιάμεσων κόμβων, ώστε να επιλέξει την κατάλληλη διαδρομή μέσω της οποίας θα του παραδοθεί η υπηρεσία.

6.2 ΠΡΟΤΕΙΝΟΜΕΝΟΣ ΑΛΓΟΡΙΘΜΟΣ

Το πρόβλημα των ενεργειακών περιορισμών μπορεί να επιλυθεί εισάγοντας μια νέα αρχιτεκτονική ανακάλυψης και παράδοσης υπηρεσιών. Σε αυτή, κρίσιμη πληροφορία, όπως οι εναπομένοντες ενεργειακοί πόροι του κόμβου – παρόχου καθώς και των ενδιάμεσων κόμβων, είναι γνωστή στο πελάτη.

Ο μηχανισμός επέκτασης στο SLPv2 παρέχεται από συγκεκριμένα τμήματα στο τέλος των SLP μηνυμάτων. Η ακριβής θέση της πρώτης επέκτασης στο SLP μήνυμα καθορίζεται από ένα συγκεκριμένο πεδίο στην επικεφαλίδα του μηνύματος ως offset μετρημένο σε bytes από την αρχή του μηνύματος. Το πεδίο αυτό είναι το “Next Extension Offset” που είναι δύο bytes και περιέχει έναν ακέραιο σε διάταξη ανά byte δικτύου (network byte order). Κάθε τμήμα επέκτασης στο τέλος του μηνύματος ορίζει την ακριβή θέση της επόμενης επέκτασης ως offset από την αρχή του μηνύματος. Πάλι, το όνομα του πεδίου είναι “Next Extension Offset”, το μήκος του είναι δύο bytes και περιέχει έναν ακέραιο. Εάν δεν υπάρχει άλλη επέκταση, το αμέσως προηγούμενο πεδίο “Next Extension Offset” πρέπει να έχει τιμή μηδέν. Στο Σχήμα 6.2 περιγράφεται αυτή η διευσθέτηση. Δείχνει τη γενική διάταξη ενός SLP μηνύματος, το μήκος, καθώς και τη θέση των διάφορων πεδίων. Η επικεφαλίδα καταλαμβάνει τις τέσσερις πρώτες γραμμές του πίνακα ακολουθούμενη από τα δεδομένα του πρωτοκόλλου και το πεδίο επέκτασης που καταλαμβάνει τις τελευταίες 2 γραμμές.



Σχήμα 6. 2: Δομή SLP μηνύματος

Η προτεινόμενη επέκταση για τη διαχείριση ενέργειας καθορίζεται στο Extension ID 0x8001, όπως περιγράφεται στο RFC 3224 [22] για πειραματικές επεκτάσεις. Το πεδίο “Extension Data” του προτεινόμενου αλγορίθμου έχει μήκος τριών bytes και περιέχει ένα πεδίο ενός byte που ορίζει εάν πρόκειται για αίτηση ή απάντηση με ενσωματωμένη την πληροφορία για την ενεργειακή κατάσταση του κόμβου. Επίσης

αποτελείται από άλλα δύο bytes που περιέχουν έναν ακέραιο, σε network byte order, που αναπαριστά το χρόνο σε λεπτά που η αντίστοιχη διαδρομή προς έναν πάροχο υπηρεσίας θα είναι διαθέσιμη. Αυτό εκφράζεται με την εναπομένονσα διάρκεια ζωής του ενδιαμέσου άλματος για τη διαδρομή που έχει τους λιγότερους ενεργειακούς πόρους. Πρόκειται για μία απαίτηση της υλοποίησης όπου κάθε κόμβος που λαμβάνει μέρος στο μοντέλο διαχείρισης ενέργειας είναι ικανός να αναφέρει τη διάρκεια ζωής του σε λεπτά. Η δρομολόγηση βασισμένη σε αυτή τη μετρική, που καλείται Minimum Battery Cost Routing (MBCR), έχει προταθεί στο [23], όπου συγκρίνεται με άλλα σχήματα βασισμένα σε ενεργειακή δρομολόγηση όπως το Total Transmission Power Routing (TTPR) και το Min-Max Battery Cost Routing (MMBCR).

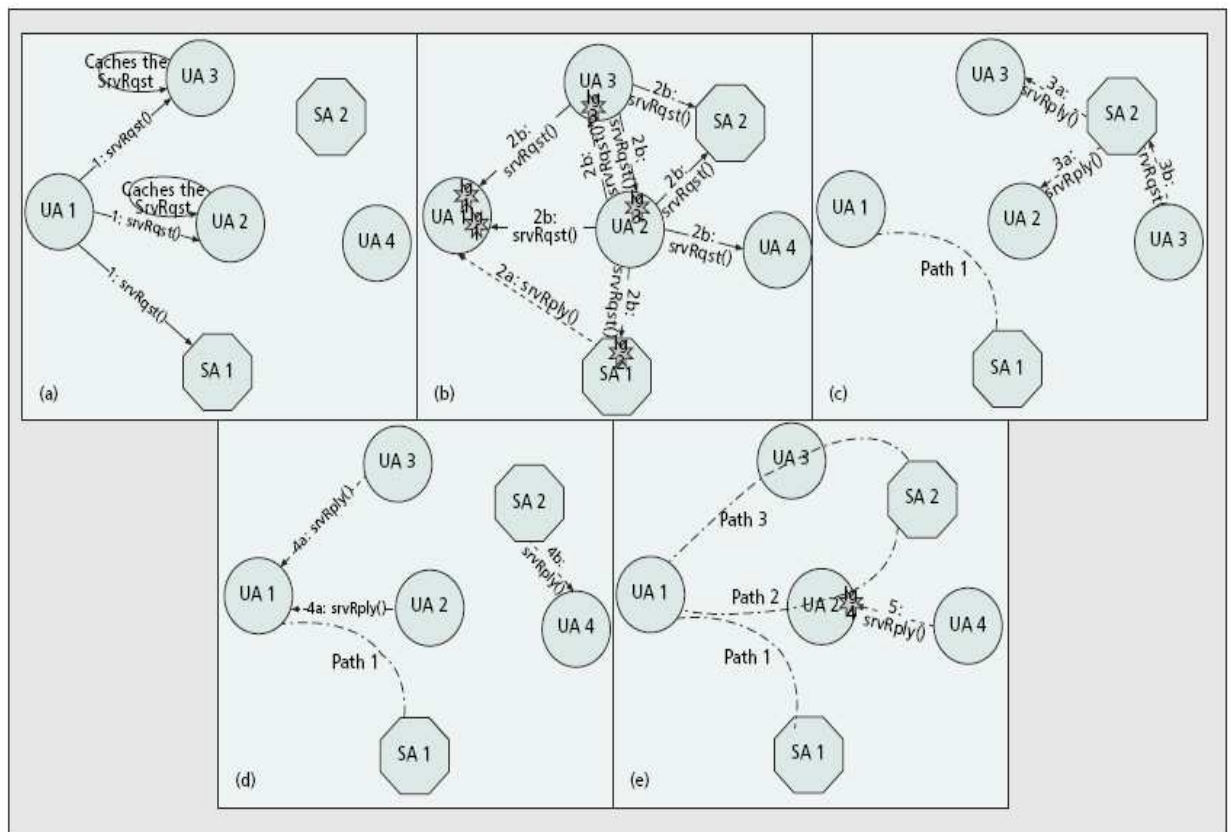
Το πεδίο Next Extension Offset παίρνει δύο πιθανές τιμές

- Μηδέν, όταν είναι η τελευταία επέκταση στο μήνυμα
- Την τιμή του πεδίου “Extension Offset” της επικεφαλίδας του μηνύματος συν οκτώ, για τα οκτώ bytes της προτεινόμενης επέκτασης, όταν υπάρχουν επακόλουθα τμήματα επέκτασης στο SLP μήνυμα.

Αντίστοιχα, το πεδίο Rqst/Rply έχει τιμή

- 0x00, όταν το τμήμα επέκτασης εδρεύει σε ένα SLP SrvRqst μήνυμα
- 0xF0, όταν το τμήμα επέκτασης εδρεύει σε ένα SLP SrvRply μήνυμα.

Η προτεινόμενη ενσωμάτωση των αρχών του MBCR στην ανακάλυψη υπηρεσιών βασισμένη στο SLP περιγράφεται με τον παρακάτω αλγόριθμο. Χάριν απλότητας, η λειτουργικότητα του αλγορίθμου, που λαμβάνει υπόψη του το SLP με επεκτάσεις, περιγράφεται με ένα παράδειγμα και απεικονίζεται στο Σχήμα 6.3.



Σχήμα 6.3 : Βήματα αλγορίθμου

Οι υποθέσεις που αφορούν την ακόλουθη περιγραφή, είναι ότι ο κόμβος UA1 εισέρχεται στο MANET και αναζητεί μία υπηρεσία την οποία και ο κόμβος SA1 και ο SA2 μπορούν να παρέχουν, ενώ οι κόμβοι UA2, UA3 και UA4 δεν μπορούν να προσφέρουν. Οι τελευταίοι, μπορούν μόνο να ενεργούν ως ενδιάμεσοι κόμβοι δρομολόγησης κατά τη φάση διάθεσης της υπηρεσίας. Επίσης, υποθέτουμε ότι κάθε Service Agent που λαμβάνει ένα μήνυμα SrvRqst, απαντάει με το αντίστοιχο μήνυμα SrvRply επεκτείνοντάς το και προσθέτοντας το παραπάνω πεδίο που περιγράψαμε την τιμή της εναπομένουσας ενέργειάς του. Επιπλέον, υποθέτουμε ότι κάθε κόμβος χειρίζεται το ληφθέν εκτεταμένο SrvRply μήνυμα όπως περιγράφεται πιο κάτω: πρώτα ελέγχει το νέο πεδίο στο SrvRply μήνυμα που αφορά τους εναπομείναντες ενεργειακούς πόρους του προηγούμενου κόμβου, και αν οι δικοί του πόροι είναι λιγότεροι, αντικαθιστά το πεδίο με τη μικρότερη τιμή (τη δική του). Ειδικά, αποστέλλει το μήνυμα στον επόμενο κόμβο χωρίς αλλαγή στο μήνυμα.

Όπως φαίνεται στο σχήμα 6.3.a, ο πελάτης UA1 εκπέμπει ένα εκτεταμένο μήνυμα SrvRqst προς όλους τους υπόλοιπους κόμβους της εμβέλειάς του (π.χ. SA1, UA2 και UA3), αναζητώντας μία συγκεκριμένη υπηρεσία. Εάν ο κόμβος που λαμβάνει το SrvRqst μήνυμα δε μπορεί να παρέχει την υπηρεσία (π.χ. UA2 και UA3), αποθηκεύει στη προσωρινή μνήμη του το μήνυμα με το αρχικό προφίλ του χρήστη-πελάτη (π.χ. IP διεύθυνση της πηγής και αριθμό XID). Το XID είναι μια μοναδική τιμή στην επικεφαλίδα κάθε SLP μηνύματος που διαφοροποιεί τα SrvRqst μηνύματα. Εισάγεται σε κάθε μήνυμα ώστε ο κόμβος να αντιλαμβάνεται εάν το λαμβανόμενο αίτημα υπηρεσίας είναι διαφορετικό από κάποιο άλλο.

Όπως απεικονίζεται στο σχήμα 6.3.b, εάν ένας πράκτορας υπηρεσίας (SA1) υπάρχει στην εμβέλεια του αιτούντος κόμβου (UA1), τότε θα απαντήσει με ένα SrvRply μήνυμα στον πελάτη. Μέσω της απάντησης, ο UA1 λαμβάνει την κατάλληλη πληροφορία που αφορά τους εναπομείναντες ενεργειακούς πόρους του SA1 και το πρώτο δυνατό μονοπάτι μεταφοράς της υπηρεσίας (π.χ. το μονοπάτι 1 στο σχήμα 6.3.c). Οι άλλοι κόμβοι, που δε μπορούν να παρέχουν στο κόμβο – πηγή τη ζητούμενη υπηρεσία, αναπαράγουν και στέλνουν το SrvRqst σε όλους τους υπόλοιπους κόμβους της εμβέλειάς τους. Ένα ληφθέν SrvRqst μήνυμα μπορεί να αγνοηθεί από ένα κόμβο στις 3 εξής περιπτώσεις:

- ο Αν ο κόμβος είναι UA και είναι ο πραγματικός εναρκτήριο κόμβος της αίτησης (Ig 1 στο σχήμα 6.3.b)
- ο Αν ο κόμβος είναι ένας SA (π.χ. SA1) και έχει ήδη λάβει ένα μήνυμα SrvRqst απευθείας από τον εναρκτήριο κόμβο (UA1). Αυτό οφείλεται στο γεγονός ότι το απευθείας μονοπάτι μεταξύ SA και UA είναι πάντα βέλτιστο συγκρινόμενο με όλα τα υπόλοιπα δυνατά μονοπάτια (που περιλαμβάνουν τον SA), ακόμα και αν ο SA έχει την ελάχιστη ενέργεια από όλους τους κόμβους των μονοπατιών (Ig 2 στο σχήμα 6.3.b)
- ο Αν ο κόμβος είναι UA και έχει ήδη λάβει ένα SrvRqst με το ίδιο XID, για τον ίδιο λόγο με παραπάνω, (Ig 3 στο σχήμα 6.3.b)

Σε περίπτωση που ένας SA λάβει περισσότερα του ενός SrvRqst μηνύματα, με το ίδιο XID αλλά από διαφορετικούς UAs (π.χ. SA2 στο σχήμα 6.3.c), απαντά με ένα SrvRply σε όλους αυτούς τους UAs (UA2 και UA3 στο 6.3.c και UA4 στο 6.3.d). Οι UAs, στέλνουν τα μηνύματα SrvRply πίσω στη πραγματική πηγή (UA1) ακολουθώντας την αντίστροφη διαδρομή από το προκαθορισμένο μονοπάτι. Ένας UA (όπως ο UA2 στο 6.3.e) μπορεί να αγνοήσει και να καταστρέψει ένα SrvRply μήνυμα προερχόμενο από άλλον UA (για παράδειγμα από τον UA4 στο 6.3.e) και το

οποίο δημιουργήθηκε από έναν SA, σε περίπτωση που έχει ήδη λάβει ένα SrvRply μήνυμα απευθείας από τον ίδιο SA (π.χ. SA2 στο σχήμα 6.3.c). Αυτό συμβαίνει για τον ίδιο λόγο που αφορά το βέλτιστο προς τα πίσω (backward) μονοπάτι. Έτσι, χρησιμοποιώντας τον προτεινόμενο αλγόριθμο, τα τελικά μονοπάτια που ανακαλύπτονται είναι τα βέλτιστα όσον αφορά τα ενδιάμεσα άλματα (hops). Ταυτόχρονα, ο πραγματικός κόμβος – πελάτης (UA1) διατηρεί την πληροφορία για την εναπομένουσα ενέργεια κάθε μονοπατιού. Βασισμένος σε αυτή, ο πελάτης (UA1) επιλέγει το κατάλληλο μονοπάτι προς τον SA (Path 1 προς τον SA1 ή Path2/Path3 προς τον SA2 στο σχήμα 6.3.e).

Η τελική επιλογή του μονοπατιού μπορεί να γίνει στη πλευρά του πελάτη χρησιμοποιώντας τυπικές και απλές συναρτήσεις με βάρη που έχουν ως ανεξάρτητες παραμέτρους τον αριθμό των αλμάτων και τους ενεργειακούς πόρους που απομένουν σε κάθε μονοπάτι. Συνεπώς, ο προτεινόμενος αλγόριθμος πληρεί τις βέλτιστες συνθήκες όσον αφορά την παροχή υπηρεσιών, ενώ παράλληλα επιδιώκει να διατηρήσει την τοπολογία και τη διάταξη του MANET.

6.3 ΣΥΜΠΕΡΑΣΜΑ

Στο κεφάλαιο αυτό αναπτύξαμε την πρότασή μας για την ανακάλυψη υπηρεσίας σε κινητά ad hoc δίκτυα. Αυτή η αρχιτεκτονική υλοποιείται στο επίπεδο εφαρμογής χωρίς να λαμβάνει υπόψη της διαδικασίες στο επίπεδο δικτύου. Η βασική ιδέα είναι βασισμένη στις επεκτάσεις του SLP σε συνδυασμό με έναν αλγόριθμο που λαμβάνει υπόψη του την εναπομένουσα ενέργεια των κόμβων καθώς και τον ελάχιστο αριθμό των αλμάτων σε κάθε δυνατό μονοπάτι παροχής υπηρεσίας. Οι νέες μετρικές «περνιούνται» μέσω μιας διαδικασίας «σηματοδοσίας στο στρώμα εφαρμογής». Το ελαφρύ φορτίο στην επικεφαλίδα που προκαλείται από αυτή τη σηματοδοσία (signaling) μπορεί να παρακαμφθεί εάν τα καίρια πλεονεκτήματά του – η διατήρηση της διάρκειας ζωής των ad hoc δικτύων και η διατήρηση της συνοχής των κόμβων – πραγματοποιούνται.

ΚΕΦΑΛΑΙΟ 7

7 ΣΥΜΠΕΡΑΣΜΑΤΑ

7.1 ΣΥΝΕΙΣΦΟΡΑ

Η παρούσα διπλωματική ανέπτυξε τη διαδικασία ανακάλυψης υπηρεσιών και συγκεκριμένα με τη χρήση του πρωτοκόλλου Service Location Protocol. Αναλύθηκε η λειτουργία, αρχικά για σταθερά IP δίκτυα υπολογιστών αλλά κυρίως για κινητά ad hoc δίκτυα. Καθώς τέτοια περιβάλλοντα έχουν αρκετούς περιορισμούς, το SLP χρειαζόταν τροποποιήσεις ώστε να λειτουργήσει αποτελεσματικά.

Με την σύγκριση των 2 αρχιτεκτονικών υλοποίησης του SLP, την centralized και την distributed, δείξαμε ότι είναι πιο αποδοτικό να μην υπάρχουν ενδιάμεσοι κόμβοι στα MANETs παρά μόνο οι χρήστες UAs που ζητούν υπηρεσίες και οι εξυπηρετητές SAs που τις παρέχουν.

Επιπλέον, στη διπλωματική αυτή, αναπτύχθηκε και ένας αλγόριθμος που λαμβάνει υπόψη του ενεργειακούς περιορισμούς κατά τη διαδικασία ανακάλυψης και παράδοσης υπηρεσιών. Είναι μια αρχιτεκτονική που θα προσδώσει αξιοπιστία και αποτελεσματικότητα στα ad hoc δίκτυα.

7.2 ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Στη κατεύθυνση αυτής της διπλωματικής, η μελλοντική εργασία που μπορεί να γίνει συνοψίζεται στα εξής:

- Το SLPManet που αναλύσαμε στο τέταρτο κεφάλαιο, μπορεί να τροποποιηθεί ώστε να λαμβάνει υπόψη του και την passive κατάσταση του service discovery. Για παράδειγμα, κάθε SA μπορεί να «πλημμυρίζει» το δίκτυο με ανακοινώσεις υπηρεσιών όταν εισέρχεται σε αυτό. Το τροποποιημένο πρωτόκολλο θα συνδυάζει την active και την passive λειτουργία.
- Απαιτείται η υλοποίηση του αλγορίθμου που προτείναμε στο έκτο κεφάλαιο. Προσομοιώνοντας διάφορα σενάρια, παρμένα από τις εφαρμογές των ad hoc δικτύων, θα διαπιστώσουμε κατά πόσον θα λυθεί το πρόβλημα των ενεργειακών περιορισμών. Και φυσικά αυτό δε σημαίνει ότι δε θα εξαντλούνται οι μπαταρίες των κόμβων, αλλά θα υπάρχει μια αξιόπιστη επικοινωνία και μετάδοση δεδομένων γνωρίζοντας εκ των προτέρων τα ενεργειακά αποθέματα.
- Ο προτεινόμενος αλγόριθμος πρέπει να εισάγει επιπλέον κριτήρια για την ανακάλυψη υπηρεσιών. Η απόσταση μεταξύ των κόμβων είναι για παράδειγμα πολύ σημαντικό ζήτημα. Ποιοι κόμβοι είναι εντός εμβέλειας ενός συγκεκριμένου host, πόσα άλματα (hops) θα χρειαστούν για

επικοινωνία με άλλους κόμβους, είναι μετρικές που πρέπει να συνυπολογίζονται στον αλγόριθμο. Άλλωστε η ενέργεια που δαπανάται σε ένα ad hoc δίκτυο και πιο συγκεκριμένα σε ένα κόμβο που συμμετέχει στο μονοπάτι είναι άμεσα συσχετισμένη και με την απόσταση αποστολής μηνύματος. Συγκεκριμένα, η δαπανώμενη ενέργεια κατά την αποστολή μηνύματος είναι αντιστρόφως ανάλογη της απόστασης μεταξύ δύο κόμβων.

- Επιπλέον κριτήρια που αφορούν τις ενεργειακές ανάγκες πρέπει να εισαχθούν. Στόχος είναι η κατανάλωση όσο το δυνατόν λιγότερης ενέργειας για τη μετάδοση πακέτων καθώς και η μεγιστοποίηση της διάρκειας ζωής των κόμβων. Και επειδή η ταυτόχρονη επίτευξη αυτών των σκοπών μπορεί να είναι δύσκολη, επιβάλλεται ένα καλό trade – off. Διάφοροι αλγόριθμοι έχουν αναπτυχθεί οι οποίοι μπορούν να υιοθετηθούν από τον αλγόριθμο που προτείναμε στο έκτο κεφάλαιο.

- Minimum Battery Cost Routing (MBCR) [23]: Υποθέτουμε ότι c_i^t είναι η χωρητικότητα της μπαταρίας ενός κόμβου n_i κατά τη χρονική στιγμή t και η οποία κυμαίνεται από 0 έως 100. Ορίζοντας ως f_i μια συνάρτηση κόστους της μπαταρίας, μπορούμε να υποθέσουμε ότι ισούται με $f_i = \frac{1}{c_i^t}$. Καθώς η χωρητικότητα

μειώνεται, η συνάρτηση κόστους αυξάνει. Το συνολικό κόστος R_j για μια διαδρομή i που αποτελείται από D κόμβους είναι :

$$R_j = \sum_{i=0}^{Dj-1} f_i(c_i^t). \text{ Συνεπώς, για να βρούμε τη διαδρομή με τη}$$

μεγαλύτερη χωρητικότητα μπαταριών, πρέπει να επιλέξουμε μια διαδρομή i που να έχει το μικρότερο κόστος $R_i = \min\{R_j \mid j \in A\}$, όπου A το σύνολο των δυνατών διαδρομών.

Το μειονέκτημα ωστόσο είναι πως μπορεί η επιλεγθείσα διαδρομή να έχει ένα κόμβο με μεγάλο κόστος f_i και συνεπώς οι διαδικασίες θα τον εξαντλήσουν γρήγορα.

- Min-Max Battery Cost Routing (MMBCR) : Για να ξεπεραστεί το παραπάνω πρόβλημα, η σχέση που δίνει το συνολικό κόστος R_j μπορεί να τροποποιηθεί, όπως υποδεικνύεται στο [24]. Εκεί το κόστος R_j ορίζεται ως $R_j = \max_{i \in \text{route}_j} f_i(c_i^t)$. Παρόμοια, το επιθυμητό μονοπάτι θα επιλεγεί από την εξίσωση $R_i = \min\{R_j \mid j \in A\}$. Ωστόσο, αυτή η μετρική θα κάνει πιο δίκαιη τη χρήση των κόμβων που έχουν ελάχιστη απομένουσα ενέργεια.

- Conditional Max-Min Battery Capacity Routing (CMMBCR) : Ο συγκεκριμένος αλγόριθμος, επιλέγει το συντομότερο μονοπάτι εάν όλοι οι κόμβοι σε όλες τις πιθανές διαδρομές έχουν επαρκή ενεργειακά αποθέματα. Όταν η χωρητικότητα της μπαταρίας κάποιων κόμβων πέσει κάτω από μια προκαθορισμένη τιμή κατωφλίου γ , οι διαδρομές μέσω αυτών των κόμβων θα αποφευχθούν. Εδώ χρησιμοποιούμε τον όρο της χωρητικότητας μιας μπαταρίας και όχι τη συνάρτηση κόστους της. Ορίζουμε

ως $R_j^c = \min_{i \in \text{route}_j} c_i^t$ την χωρητικότητα τη χρονική στιγμή t . Εάν A είναι το σύνολο των δυνατών διαδρομών μεταξύ οποιονδήποτε δύο κόμβων, τότε πρέπει να ικανοποιείται η εξίσωση $R_j^c \geq \gamma$, για καθε διαδρομή $j \in A$. Προσαρμόζοντας την τιμή του γ , μπορούμε να μεγιστοποιήσουμε είτε το χρόνο που κάποιος κόμβος θα πέσει, είτε τη διάρκεια ζωής των περισσότερων κόμβων στο δίκτυο.

Για σύγκριση αυτών των αλγορίθμων, αναφορά γίνεται στο [25]. Να σημειώσουμε ότι ως μελλοντική εργασία θεωρείται και η υλοποίηση συνδυασμού των παραπάνω αλγορίθμων κατά την χρήση της στην επέκταση του SLP.

- Τα κριτήρια απόστασης και ενέργειας μπορούν να συνδυαστούν κατάλληλα ορίζοντας νέες, αντίστοιχες μετρικές που θα βελτιστοποιήσουν την επιλογή διαδρομών κατά την ανακάλυψη υπηρεσιών.
- Όπως αναφέραμε στο έκτο κεφάλαιο, η προτεινόμενη υλοποίηση γίνεται στο επίπεδο εφαρμογών (application layer). Ωστόσο, κρίνεται σημαντική η υλοποίηση μιας cross-layer προσέγγισης στη διαδικασία ανακάλυψης υπηρεσιών. Αυτή η αρχιτεκτονική θα συγχωνεύει πληροφορίες ενέργειας και αριθμό αλμάτων με αντίστοιχες πληροφορίες απόστασης προερχόμενες από το επίπεδο δικτύου. Τέτοιες αρχιτεκτονικές αναλύονται στο [26], [27] και [28]. Τα κριτήρια που εφαρμόστηκαν στο application layer, πρέπει να ενσωματωθούν σε αλγόριθμους δρομολόγησης (π.χ. AODV, DSDV). Τυπικά, η συγχώνευση του service discovery με διαδικασίες δρομολόγησης (routing), μπορεί να οδηγήσει σε καλύτερη επίδοση στα ad hoc δίκτυα. Το στενό δέσιμο των δύο επιπέδων, μπορεί να αποδειχθεί η καταλληλότερη λύση.
- Στο SLPManet που είδαμε, πρέπει να λάβουμε υπόψη μας και την περίπτωση που μια υπηρεσία δε μπορεί να προσπελαστεί. Τότε, θα πρέπει ο χρήστης να μπορεί να ξανά ανακαλύψει υπηρεσίες χωρίς να χρειαστεί να περιμένει εκπνοή των αποθηκευμένων URLs.
- Για καλύτερα και πιο αξιόπιστα αποτελέσματα, πρέπει οι διάφορες υλοποιήσεις να εξετάζουν πολλές περιπτώσεις. Έτσι πρέπει να εισαχθούν νέα μοντέλα κίνησης στις προσομοιώσεις ώστε να αποτυπώνουν πιο πιστά τη πραγματικότητα. Για παράδειγμα, στο [29], προτείνεται ο σχεδιασμός ενός μοντέλου που εισάγει εμπόδια τα οποία περιορίζουν την κίνηση αλλά και τη διάδοση του σήματος. Επιπλέον, νέες μετρικές πρέπει να εκτιμηθούν. Μία από αυτές είναι το ποσοστό των ανακαλυπτόμενων υπηρεσιών που είναι προσβάσιμες από το χρήστη.
- Τέλος, όσον αφορά το περιβάλλον προσομοίωσης, στην υλοποίηση του SLP στον NS-2, κάθε κόμβος δρα είτε ως SA είτε ως UA. Μελλοντική εργασία πρέπει να αφορά την συνύπαρξη αυτών των δύο agent σε έναν κόμβο.

8 ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1] S. Vinoski, “Service Discovery 101,” *IEEE Internet Comp.*, vol. 7, no. 1, Jan. 2003, pp. 69–71.
- [2] http://en.wikipedia.org/wiki/Service_discovery
- [3] E. Guttman, C. Perkins, J. Veizades, and M. Day, “Service Location Protocol, Version 2,” Request for Comments Standards Track 2608, Internet Engineering Task Force, June 1999.
- [4] Sun Microsystems, Jini Device Architecture Specification, Version 2.0. <http://www.sun.com/software/jini/whitepapers/jini-datasheet0601.pdf>
- [5] Salutation Consortium, “Salutation Architecture Overview.” White Paper, 1998. <http://www.salutation.org/whitepaper/originalwp.pdf>
- [6] Universal Plug and Play Forum, Universal Plug and Play Device Architecture, Version 1.0, June 2000. <http://www.upnp.org/resources/documents.asp>
- [7] Bluetooth SIG, Specification of the Bluetooth System, Volume 1, 2001. <http://www.bluetooth.com>.
- [8] M. Nidd, “Service Discovery in DEAPspace,” *IEEE Personal Comm.*, pp. 39–45, August 2001
- [9] M. Storey, G. Blair, and A. Friday, “MARE: resource discovery and configuration in ad hoc networks,” *Mobile Networks and Applications*, vol. 7, pp. 377 – 387, 2002.
- [10] F. Zhu, M. Mutka, and L. Ni, “Splendor: A secure, private, and location aware service discovery protocol supporting mobile services,” in *Proceeding of the 1st IEEE Annual Conference on Pervasive Computing and Communications*, pp. 235–242, IEEE Computer Society Press, March 2003.
- [11] S. Helal, N. Desai, V. Verma, and C. Lee, “Konark - A Service Discovery and Delivery Protocol for Ad-hoc Networks,” in *Proceedings of the Third IEEE Conference on Wireless Communication Networks (WCNC)*, vol. 3, pp. 2107–2113, March 2003.
- [12] <http://uddi.org/pubs/uddi-v3.0.2-20041019.htm>
- [13] J. Macker and I. Chakeres, “Mobile Ad-hoc Networks (manet) IETF Charter”, <http://www.ietf.org/html.charters/manet-charter.html>
- [14] D. Chakraborty, A. Joshi, Y. Yesha, T. Finin, “GSD: A Novel Group-based Service Discovery Protocol for MANETS”, *Computer Science And Electrical Engineering*, University of Maryland, Baltimore County.

- [15] http://en.wikipedia.org/wiki/Ad_hoc_protocol_list#Proactive_Routing_.28T.20able-driven.29
- [16] <http://www.faqs.org/rfcs/rfc2608.html>
- [17] C. E. Perkins and E. Belding-Royer, "Ad Hoc On-Demand Distance Vector (AODV) Routing," IETF MANET Working Group, RFC3561, July 2003.
- [18] <http://moment.cs.ucsb.edu/AODV/aodv.html>
- [19] K. Fall and K. Varadhan, The ns Manual. VINT Project - UC Berkley, December 2003. <http://www.isi.edu/nsnam/ns/ns-documentation.html>
- [20] W. Navidi and T. Camp, "Stationary Distributions for the Random Waypoint Mobility Model," IEEE Transactions on Mobile Computing, vol. 3(1), pp. 99–108, January/March 2004.
- [21] A. Ephremides, "Energy Concerns in Wireless Networks" IEEE Wireless Commun., vol. 9, no. 4, Aug. 2002, pp. 48–59.
- [22] <http://www.faqs.org/rfcs/rfc3224.html>
- [23] C.-K. Toh, "Maximum Battery Life Routing to Support Ubiquitous Mobile Computing in Wireless Ad Hoc Networks," IEEE Commun. Mag., June 2001, pp. 138–47.
- [24] Singh, M. Woo, and C. S. Raghavendra, "Power-Aware Routing in Mobile Ad Hoc Networks," Proc. MobiCom '98, Dallas, TX. Oct. 1998.
- [25] <http://ieeexplore.ieee.org/iel5/35/20027/00925682.pdf?arnumber=925682>.
- [26] A. Varshavsky, B. Reid, and E. de Lara, "The Need for Cross-Layer Service Discovery in MANETs," Tech. Rep. CSRG-492, UofT Comp. Sci., 2004
- [27] L. Li and L. Lamont, "A Lightweight Service Discovery Mechanism for Mobile Ad Hoc Pervasive Environment Using Cross-Layer Design," in Proceeding of the Third IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'05), pp. 55–59, March 2005.
- [28] A. Varshavsky, B. Reid, and E. de Lara, "Cross-Layer Support for Timely and Efficient Service Selection in MANETs." Department of Computer Science, University of Toronto.
- [29] A. Jardish, E. Belding-Royer, K. Almeroth, and S. Suri, "Real-World Environment Models for Mobile Network Evaluation," IEEE Journal on Selected Areas in Communications (ISACEM), vol. 23(3), pp. 622–632, March 2005.

- [30] T. Kunz, “Multicasting in mobile ad-hoc networks: Achieving high packet delivery ratios,” in Proceedings of the 2003 Center for Advanced Studies Conference (CAS), (Toronto, Canada), pp. 156–170, IBM Canada Ltd. Laboratory, Center for Advanced Studies, October 2003.
- [31] T. Kunz, “Public Domain Code,” March 2004.
<http://www.sce.carleton.ca/wmc/code.html>.

ΠΑΡΑΡΤΗΜΑ Α

ΑΛΛΑΓΕΣ ΣΤΟΝ NS-2

A.1 Προσθήκη αρχείων για υποστήριξη του SLP

Για να προσομοιώσουμε τα σενάρια στον Network Simulator – 2, πρέπει να προσθέσουμε κάποια αρχεία του ώστε να υποστηρίζουν το πρωτόκολλο SLP. Η έκδοση που χρησιμοποιήσαμε είναι η ns-allinone-2.30.

Τα αρχεία που πρέπει να προστεθούν σε έναν φάκελο μέσα στον \ns-allinone-2.30\ns-2.30\ είναι τα εξής:

- 1) udp-slp.h
- 2) udp-slp.cc
- 3) slp-sa.h
- 4) slp-sa.cc
- 5) slp-ua.h
- 6) slp-ua.cc
- 7) slp-common.h

Ο κώδικας κάθε αρχείου φαίνεται παρακάτω:

```
#include "slp/slp-common.h"

// UdpSlpAgent Class definition
class UdpSLPAgent : public UdpAgent {
public:
    UdpSLPAgent();
    UdpSLPAgent(packet_t);
    virtual int supportSLP() { return 1; }
    virtual void enableSLP() { support_slp_ = 1; }
    virtual void sendto(int nbytes, const char* flags, ns_addr_t
dst);
    void recv(Packet*, Handler*);
protected:
    int support_slp_; // set to 1 if above is SLApp
};
```

A.1.1 udp-slp.h

```
#include "slp/udp-slp.h"

int hdr_slp::offset_;

// SLP Header Class
static class SLPHeaderClass : public PacketHeaderClass {
public:
    SLPHeaderClass() :
PacketHeaderClass("PacketHeader/SLP", sizeof(hdr_slp)) {
```

```

        bind_offset(&hdr_slp::offset_);
    }
} class_slphdr;

// UdpSlpAgent OTcl linkage class
static class UdpSLPAgentClass : public TclClass {
public:
    UdpSLPAgentClass() : TclClass("Agent/UDP/UDPSlp") {}
    TclObject* create(int, const char*const*) {
        return (new UdpSLPAgent());
    }
} class_udpslp_agent;

// Constructor (with no arg)
UdpSLPAgent::UdpSLPAgent() : UdpAgent()
{
    support_slp_ = 0;
    type_ = PT_SLP;
}

UdpSLPAgent::UdpSLPAgent(packet_t type) : UdpAgent(type)
{
    support_slp_ = 0;
    type_ = PT_SLP;
}

// Add Support of SLP Application to UdpAgent::sendto
void UdpSLPAgent::sendto(int nbytes, const char* flags, ns_addr_t
dst)
{
    Packet *p;
    int n, remain;

    if (size_) {
        n = (nbytes/size_ + (nbytes%size_ ? 1 : 0));
        remain = nbytes%size_;
    }
    else
        printf("Error: UDPSlp size = 0\n");

    if (nbytes == -1) {
        printf("Error: sendmsg() for UDPSlp should not be -1\n");
        return;
    }
    double local_time =Scheduler::instance().clock();
    while (n-- > 0) {
        p = allocpkt();
        if(n==0 && remain>0)
            HDR_CMN(p)->size() = remain;
        else
        {
            //hdr_cmn::access(p)->size() = size_;
            printf("Error: SLP message (%d bytes) could not fit UDP
datagram (%d bytes)\n",
                nbytes,size_);
            return;
        }
    }
    hdr_rtp* rh = HDR_RTP(p);

```

```

    rh->flags() = 0;
    rh->seqno() = ++seqno_;
    HDR_CMN(p)->timestamp() = (u_int32_t)(SAMPLERATE*local_time);
    // to eliminate recv to use SLP fields for non SLP packets
    hdr_slp* sh = HDR_SLP(p);
    /*
    sh->function_id = 0;
    sh->length = 0;
    sh->length_contd = 0;
    sh->reserved = 0;
    sh->next_ext_offset = 0;
    sh->next_ext_offset_contd = 0;
    sh->xid = -1;
    sh->language_tag_length = 0;
    sh->language_tag = 0;
    */
    // slp udp packets are distinguished by setting the ip
    // priority bit to 14.
    if(support_slp_) {
        hdr_ip* ih = HDR_IP(p);
        ih->prio_ = 14;
        if(flags) // SLP Seq Num is passed as flags
        {
            if (((hdr_slp *)flags)->function_id
== SLPTYPE_SRVRQST) {
                hdr_slp_srvrqst* srvrqst = HDR_SLP_SRVRQST(p);
                ih->dst()=dst;
                memcpy(srvrqst, flags, sizeof(hdr_slp_srvrqst));
            }
            else if (((hdr_slp *)flags)->function_id ==
SLPTYPE_SRVRPLY) {
                ih->dst()=dst;
                hdr_slp_srvrply* srvrply = HDR_SLP_SRVRPLY(p);
                memcpy(srvrply, flags, sizeof(hdr_slp_srvrply));
            }
            else if (((hdr_slp *)flags)->function_id ==
SLPTYPE_SAADVERT) {
                ih->dst()=dst;
                hdr_slp_saadvert* saadvert = HDR_SLP_SAADVERT(p);
                memcpy(saadvert, flags, sizeof(hdr_slp_saadvert));
            }
            else {
                printf("Error: UdpSlpAgent::sendmsg ... slp message
(%d) not supported\n",
                    ((hdr_slp *)flags)->function_id);
                return;
            }
        }
#ifdef DEBUGUDP
        printf("n%d: ---->UDP agent sending %d bytes message to
%x:%d\n",addr(), nbytes, ih->daddr(),ih->dport());
#endif
    }
    // add "beginning of talkspurt" labels (tcl/ex/test-rcvr.tcl)
    if (flags && (0 ==strcmp(flags, "NEW_BURST")))
        rh->flags() |= RTP_M;

    target_->recv(p);
}
idle();
}

```

```
// Support SLP Application
void UdpSLPAgent::recv(Packet* p, Handler*)
{
    hdr_ip* ih = HDR_IP(p);
    int bytes_to_deliver = HDR_CMN(p)->size();
#ifdef DEBUGUDP
    printf ("n%d: <---UDP agent receving message from %x:%d\n",addr(),
ih->saddr(),ih->sport());
#endif
    // if it is a SLP packet (data or ack)
    if(ih->prio_ == 14) {
        if(app_) { // if SLP Application exists
            hdr_slp* sh = HDR_SLP(p);
            if (sh->function_id == SLPTYPE_SRVRQST) {
                hdr_slp_srvrqst* srvrqst = HDR_SLP_SRVRQST(p);
                hdr_slp_srvrqst srvrqst_buf;
                memcpy(&srvrqst_buf, srvrqst,
sizeof(hdr_slp_srvrqst));
                app_->recv_slpmsg((ns_addr_t)ih->src(),
srvrqst_buf.size(), (char*) &srvrqst_buf);
            }
            else if (sh->function_id == SLPTYPE_SRVRPLY) {
                //printf ("n%d: <---UDP agent receving message from %x:%d to
%d\n",addr(), ih->saddr(),ih->sport(), ih->daddr());
                hdr_slp_srvrply* srvrply = HDR_SLP_SRVRPLY(p);

                hdr_slp_srvrply srvrply_buf;
                memcpy(&srvrply_buf, srvrply,
sizeof(hdr_slp_srvrply));
                app_->recv_slpmsg((ns_addr_t)ih->src(),
srvrply_buf.size(), (char*) &srvrply_buf);
            }
            else if (sh->function_id == SLPTYPE_SAADVERT) {
                hdr_slp_saadvert* saadvert = HDR_SLP_SAADVERT(p);

                hdr_slp_saadvert saadvert_buf;
                memcpy(&saadvert_buf, saadvert,
sizeof(hdr_slp_saadvert));
                app_->recv_slpmsg((ns_addr_t)ih->src(),
saadvert_buf.size(), (char*) &saadvert_buf);
            }
            else {
                printf("Error: UdpSlpAgent::recv ... slp message not
supported\n");
                return;
            }
        }
        Packet::free(p);
    }
    // if it is a normal data packet (not SLP data or ack packet)
    else {
        if (app_) app_->recv(bytes_to_deliver);
        Packet::free(p);
    }
}
```

A.1.2 udp-slp.cc

```
#include "slp/slp-common.h"
class SLPsa;
class URLnode;
class urlExpireTimer : public TimerHandler {
public:
    urlExpireTimer(SLPsa* s, URLnode* n, double timeout) :
TimerHandler(), s_(s), n_(n) {
        sched(timeout);
    }
    inline virtual void expire(Event*);

protected:
    SLPsa* s_;
    URLnode * n_;
};

class URLnode {
public:
    hdr_slp_url      data;
    URLnode*        next;
    urlExpireTimer   expireTimer;
    URLnode (SLPsa* s, double timeout) : expireTimer(s, this,
timeout) {}
};

// SLP SA Class Definition
class SLPsa : public Application {
    friend class URLnode;
    friend class urlExpireTimer;
public:
    SLPsa();
    void send_srvrply_pkt(u_int16_t url_count, const char
*service_type, u_int16_t xid, ns_addr_t dest);
    void send_saadvert_pkt(u_int16_t xid, ns_addr_t dest);

protected:
    int command(int argc, const char*const* argv);
    void start();          // Start sending data packets (Sender)
    void stop();           // Stop sending data packets (Sender)

private:
    void init();
    virtual void recv_slpmsg(ns_addr_t src, int nbytes, const char
*msg = 0);
    void add_scope(const char *newscope);
    void remove_scope(const char *removescope);
    void add_service(const char *newservice, int timeout, int
lifetime);
    void remove_service(URLnode *removeservice_url);
    bool checkscope(const char* scope_list, u_int16_t
length_scope);
    u_int16_t checkservice(const char* service_type);
    char *get_urls (u_int16_t &no_urls, const char *serv_type, int
size_so_far);
    char * SLPsa::get_local_scopes() const;
    int    running_;
    int    pktsize_;
    char * local_services_[SLP_MAX_SERVICE];    //array implementation
    int    number_of_scopes_;
    char * local_scopes_[SLP_MAX_SCOPE];        //array implementation
```

```
bool checkprlist(const char* pr_list, u_int16_t length_pr_list);
URLnode * url_head;
};
```

A.1.3 slp-sa.h

```
#include "slp/slp-sa.h"

// SLPsa OTcl linkage class
static class SLPsaClass : public TclClass {
public:
    SLPsaClass() : TclClass("Application/SLPsa") {}
    TclObject* create(int, const char*const*) {
        return (new SLPsa);
    }
} class_slp_sa;

void urlExpireTimer::expire(Event *)
{
    s_>remove_service(n_);
}

// Constructor (also initialize instances of timers)
SLPsa::SLPsa() : running_(0)
{
    bind("pktsize_", &pktsize_);
    number_of_scopes_ = 0;
    url_head = NULL;
}

// OTcl command interpreter
int SLPsa::command(int argc, const char*const* argv)
{
    Tcl& tcl = Tcl::instance();

    if (argc == 3) {
        if (strcmp(argv[1], "attach-agent") == 0) {
            agent_ = (Agent*) TclObject::lookup(argv[2]);
#ifdef DEBUGSA
            printf("new server agent=%d\n", agent_);
#endif
            if (agent_ == 0) {
                tcl.resultf("no such agent %s", argv[2]);
                return(TCL_ERROR);
            }

            // Make sure the underlying agent support MM
            if(agent_>supportSLP()) {
                agent_>enableSLP();
            }
            else {
                tcl.resultf("agent \"%s\" does not support SLP Application",
                    argv[2]);
                return(TCL_ERROR);
            }
        }
    }
}
```

```

        agent_->attachApp(this);
        return(TCL_OK);
    }
    else if (strcmp(argv[1],"add-scope") == 0)
    {
        add_scope(argv[2]);
        return(TCL_OK);
    }
    else if (strcmp(argv[1],"remove-scope") == 0)
    {
        remove_scope(argv[2]);
        return(TCL_OK);
    }
}
else if (argc == 5) {
    if (strcmp(argv[1],"add-service") == 0)
    {
        add_service(argv[2], atoi(argv[3]), atoi(argv[4]));
        return(TCL_OK);
    }
}
return (Application::command(argc, argv));
}

void SLPsa::init()
{
    //add_scope("DEFAULT");
}

void SLPsa::add_scope(const char* newscope) {
    char *scope=small_string(newscope);
    for (int i=0; i<number_of_scopes_; i++)
    {
        if (strcmp(local_scopes_[i],scope)==0)
        { //already in list
            return;
        }
    }
    if (number_of_scopes_==SLP_MAX_SCOPE)
    {
        printf("Can't add anymore scopes, MAX=%d scopes per
SA\n",SLP_MAX_SCOPE);
        return;
    }
    local_scopes_[number_of_scopes_]=scope;

#ifdef DEBUGSA
    printf("SA:Added %s(%d), total=%d Scope
\n",local_scopes_[number_of_scopes_], strlen(newscope),
number_of_scopes_+1);
#endif
    number_of_scopes_++;
    return;
}

void SLPsa::remove_scope(const char *removescope) {
    int i,j;

```

```

char *scope = small_string(removescope);
for (i=0; i<number_of_scopes_; i++)
{
    if (strcmp(local_scopes_[i],scope)==0)
        break;
}
number_of_scopes_--;
for (j=i; j<number_of_scopes_; j++)
{
    local_scopes_[j]=strdup(local_scopes_[j+1]);
}
}

void SLPsa::add_service(const char* newservice, int timeout, int
lifetime) {

    /*adding new node*/
    URLnode *c;
    c = new URLnode(this, (double)timeout);
    (c->data).lifetime=lifetime;
    (c->data).url=strdup(newservice);
    (c->data).url_length=strlen(newservice);
    (c->data).auth_no=0;
    c->next = url_head;
    url_head = c;

#ifdef DEBUGSA
    printf("SA:Added %s %d\n", (url_head->data).url, (url_head-
>data).auth_no);
#endif
}

void SLPsa::remove_service(URLnode *removeservice_url) {
    URLnode *p=NULL, *c=url_head;
    while ((c!=NULL) && (c != removeservice_url)) {
        p=c; c=c->next;
    }
    if (c==NULL)
        return;
    if (p==NULL)
        url_head = c->next;
    else
        p->next = c->next;
    delete c;
}

void SLPsa::start()
{
    init();
    running_ = 1;
    if (number_of_scopes_==0)
        add_scope("DEFAULT");
}

void SLPsa::stop()
{
    running_ = 0;
}

```

```
// Send application data packet
void SLPsa::send_srvrply_pkt(u_int16_t url_count, const char
*service_type, u_int16_t xid, ns_addr_t dest)
{
    #ifdef BCAST
        dest.addr_=dest.addr_+MULTICAST_ADDR+1;
    #endif

    hdr_slp_srvrply srvrply_buf;
    if (running_) {
        // the below info is passed to UDPslp agent, which will write it
        // to SLP SRVRPLY header after packet creation.
        srvrply_buf.version = SLP_VERSION;
        srvrply_buf.function_id = SLPTYPE_SRVRPLY;
        srvrply_buf.flags=0;
        srvrply_buf.next_ext_offset=0;
        srvrply_buf.xid = xid;
        srvrply_buf.language_tag_length=strlen(SLP_LANGUAGE);
        srvrply_buf.language_tag=SLP_LANGUAGE;
        srvrply_buf.error_code=0;
        //first assume no urls, to get size so far
        srvrply_buf.url_entry_count=0;
        int size_so_far = srvrply_buf.size();
        //now assume packet is big enough to include all urls
        srvrply_buf.url_entry_count=url_count;
        //adjust the url_entry_count if packet not big enough, and return
        max possible string of urls
        srvrply_buf.url=get_urls(srvrply_buf.url_entry_count,
        service_type, size_so_far);
        srvrply_buf.length=srvrply_buf.size();
        agent_>sendto(srvrply_buf.length, (char*) &srvrply_buf, dest);
    // send to UDP

        #ifdef DEBUGSA
            printf("n%d: SA sending service reply (xid=%d) to
            %x:%d\n",agent_>addr(), xid, dest.addr_,dest.port_);
            printf("reply urls(%d) ptr: %d %s\n",srvrply_buf.url_entry_count,
            srvrply_buf.url,((hdr_slp_url *)
            srvrply_buf.url)->url);
        #endif
    }
}

// Send saadvert packet
void SLPsa::send_saadvert_pkt(u_int16_t xid, ns_addr_t dest)
{
    hdr_slp_saadvert saadvert_buf;
    char url_[30] = "service:service-agent://";
    char urla_[5];
    sprintf(urla_, "%d", agent_>addr());
    strcat (url_,urla_);
    char * scopes = get_local_scopes();
    if (running_) {
        // the below info is passed to UDPslp agent, which will write it
        // to SLP SAADVERT header after packet creation.
        saadvert_buf.version = SLP_VERSION;
        saadvert_buf.function_id = SLPTYPE_SAADVERT;
        saadvert_buf.flags=0;
        saadvert_buf.next_ext_offset=0;
        saadvert_buf.xid = xid;
        saadvert_buf.language_tag_length=strlen(SLP_LANGUAGE);
    }
}
```

```

saadvert_buf.language_tag=SLP_LANGUAGE;

saadvert_buf.length_url=strlen(url_);
saadvert_buf.url=strdup(url_);
saadvert_buf.length_scope_list=strlen(scopes);
saadvert_buf.string_scope_list=scopes;
saadvert_buf.length_attr_list=0;
saadvert_buf.auth_no=0;
saadvert_buf.length=saadvert_buf.size();
agent_->sendto(saadvert_buf.length, (char*) &saadvert_buf, dest);
// send to UDP

#ifdef DEBUGSA
printf("n%d: SA sending service advertisement (xid=%d) to
%x:%d\n",agent_->addr(), xid, dest.addr_,dest.port_);
printf("url: %s\n",saadvert_buf.url);
#endif
}
}

// Receive message from underlying agent
void SLPsa::recv_slpmsg(ns_addr_t src, int nbytes, const char *msg)
{
    u_int16_t no_urls;
    bool scopeexists;
    bool nopreexists;
#ifdef DEBUG
printf("n%d: SA receiving message from %x:%d\n",agent_->addr(),
src.addr_, src.port_);
#endif
    if(msg) {
        hdr_slp* slp_buf = (hdr_slp*) msg;
        // If received packet is a service request packet
        if (slp_buf->function_id == SLPTYPE_SRVRQST)
        {
            hdr_slp_srvrqst* slp_srvrqst_buf = (hdr_slp_srvrqst*) msg;

            scopeexists=checkscope(slp_srvrqst_buf->string_scope_list,
slp_srvrqst_buf->length_scope_list);
            nopreexists=checkprlist(slp_srvrqst_buf->string_prlist,
slp_srvrqst_buf->length_prlist);

            if (strcmp(slp_srvrqst_buf-
>string_service_type,"service:service-agent")==0)
            {
                if ((!scopeexists) && (slp_srvrqst_buf-
>length_scope_list!=0))
                    return;
                if (!nopreexists)
                    return;
                send_saadvert_pkt(slp_srvrqst_buf->xid, src);
            }
            else
            {
                if (!scopeexists)
                    return;
                if (!nopreexists)
                    return;
                if ((no_urls=checkservice(slp_srvrqst_buf-
>string_service_type)) < 1)
                    return;
            }
        }
    }
}

```

```

        send_srvrply_pkt(no_urls, slp_srvrqst_buf-
>string_service_type, slp_srvrqst_buf->xid, src);
    }
}
}

bool SLPsa::checkscope(const char* scope_list, u_int16_t
length_scope)
{
    if (length_scope==0)
        return false;

    char *delimiters = ",";
    char *token;
    char scope[length_scope];
    strcpy(scope,scope_list);

    token=strtok(scope, delimiters);
    while (token!=NULL)
    {
#ifdef DEBUGSA
        printf("Token is %s\n",token);;
#endif
        for(int i=0; i<number_of_scopes_; i++)
        {
            if (strcmp(local_scopes_[i],token)==0)
                return true;
        }
        token=strtok(NULL, delimiters);
    }

    return false;
}

static bool match(const char *url, const char *service)
{
    int position;
    char *token;
    char *temp1;
    char *temp2;
    temp1=strdup(url);
    token=strstr(temp1,"://");
    position = token - temp1;
    temp2=strndup(temp1,position);
    delete temp1;

    if (strstr(temp2,service) != NULL)
    {
        delete temp2;
        return true;
    }
    else
    {
        delete temp2;
        return false;
    }
}

u_int16_t SLPsa::checkservice(const char* service_type)

```

```

{
    URLnode *s = url_head;
    u_int16_t no_urls=0;

    while (s!=NULL)
    {
        #ifdef DEBUGSA
            printf("SLPsa: comparing %s with %s service\n", (s->data).url,
service_type);
        #endif

        if (match((s->data).url, service_type) == true)
            no_urls++;
        s=s->next;
    }
    return no_urls;
}

bool SLPsa::checkprlist(const char* pr_list, u_int16_t
length_pr_list)
{
    if (length_pr_list==0)
        return true;

    char *delimiters = ",";
    char *token;
    char prlist[length_pr_list];
    strcpy(prlist, pr_list);
    token=strtok(prlist, delimiters);
    while (token!=NULL)
    {
        #ifdef DEBUGSA
            printf("prToken is %s, comparing with n%d\n", token, agent_-
>addr());
        #endif
        if (agent_->addr()==atoi(token))
            return false;

        token=strtok(NULL, delimiters);
    }
    return true;
}

char * SLPsa::get_urls (u_int16_t &no_urls, const char *serv_type,
int size_so_far)
{
    URLnode *c = url_head;
    char *buff;
    //maximum memory that needs to be allocated
    buff = (char *)malloc(sizeof(hdr_slp_url)*no_urls);
    int i=0;
    int size=size_so_far;
    u_int16_t new_no_urls=0;
    while (c!=NULL)
    {
        if (match((c->data).url, serv_type)==true)
        {
            //if next url will make pkt overflow, then don't add anymore
            if ( (size += (c->data).size()) > pktsize_)

```

```

        break;
        memcpy(buff+i,&(c->data), sizeof(hdr_slp_url));
        i+=sizeof(hdr_slp_url);
        new_no_urls++;
    }
    c=c->next;
}
//adjust no_urls
no_urls = new_no_urls;
//shrink memory already allocated if necessary (i.e. if new_no_urls
< original no_urls)
buff = (char *) realloc(buff, sizeof(hdr_slp_url)*no_urls);

#ifdef DEBUGSA
i=0;
for (int j=0; j<no_urls; j++) {
    printf("okay %d %d %s\n",((hdr_slp_url *) (buff+i))-
>lifetime,((hdr_slp_url *) (buff+i))->auth_no,
        ((hdr_slp_url *) (buff+i))->url);
    i+=sizeof(hdr_slp_url);
}
#endif

return buff;
}

char * SLPsa::get_local_scopes() const
{
    char buf[21];
    char str[SLP_MAX_SCOPE*20]="";
    for (int i=0; i<number_of_scopes_; i++)
    {
        if (i==0) {
            sprintf(buf,"%s",local_scopes_[i]);
            strcat(str,buf);
        }
        else {
            sprintf(buf,"%s",local_scopes_[i]);
            strcat(str,buf);
        }
    }
    return strdup(str);
}

```

A.1.4 slp-sa.cc

```

#include "slp/slp-common.h"

extern u_int16_t G_Xid;
extern RNG G_arrival;
class SLPua;
class rqstHashTable;
class rqstNode;
class urlNode;

class reReqTimer : public TimerHandler {

```

```

public:
    reReqTimer(SLPua* s, double i, nsaddr_t a, const char* r, int m)
: TimerHandler() {
    s_=s; i_=i; a_=a; r_=strdup(r); m_=m;
    if (m==REG)
        resched(i);
    else if (m==EXP)
        resched(G_arrival.exponential(i));
}
inline virtual void expire(Event*);

protected:
    SLPua*      s_;
    double      i_;
    nsaddr_t    a_;
    const char* r_;
    int         m_;
};

class urlCacheTimer : public TimerHandler {
public:
    urlCacheTimer(SLPua* s, urlNode* n, double timeout) :
TimerHandler(), s_(s), n_(n) {
        sched(timeout);
    }
    inline virtual void expire(Event*);

protected:
    SLPua*      s_;
    urlNode *   n_;
};

class urlNode {
public:
    hdr_slp_url    data;
    urlNode*       next;
    urlCacheTimer  expireTimer;
    urlNode (SLPua* s, double timeout) : expireTimer(s, this,
timeout) {}
};

class retxTimer : public TimerHandler {
public:
    retxTimer(rqstHashTable* h, rqstNode* n) : TimerHandler(), h_(h),
n_(n) {}
    inline virtual void expire(Event*);
protected:
    rqstHashTable* h_;
    rqstNode *   n_;
};

class rqstNode
{
public:
    u_int16_t xid;
    int      number_of_pr;
    int      local_pr[SLP_MAX_PRLIST];
    char     msg[sizeof(hdr_slp_srvrqst)]; // buffered pkt (used for
retx)
    ns_addr_t dest;
    double timeout; // retransmission timeout
}
    
```

```

double expire;    // retransmissions end here
double cache_lifetime; // cache lifetime
rqstNode *next;    // next in the bucket
bool  retxok;    // enable retransmissions
bool  gotreply;
retxTimer retx_timer_;
rqstNode(rqstHashTable *h) : retx_timer_(h, this) {}
};

class rqstHashTable {
private:
    int tablesize;
    rqstNode* *table;
    SLPua* s_;
    int hash(u_int16_t xid) const;
    void  add_local_pr(rqstNode* c, int newaddress);
    char *  get_local_pr(const rqstNode* c) const;
public:
    rqstHashTable(SLPua* s);
    void  cancel_rettx(u_int16_t xid);
    void  add_pr(u_int16_t xid, int newaddress);
    char *  get_pr(u_int16_t xid) const;
    void  add_resched_send(u_int16_t xid, int size, const char* mssg ,
ns_addr_t dst);
    void  update_resched_resend(rqstNode* n);
    void  remove_from_table (u_int16_t xid);
};

// SLP UA Class Definition
class SLPua : public Application {
    friend class rqstHashTable;
    friend class urlCacheTimer;
    friend class reReqTimer;

public:
    SLPua();
protected:
    int command(int argc, const char*const* argv);
    void start();    // Start sending data packets (Sender)
    void stop();    // Stop sending data packets (Sender)
private:
    void init();
    virtual void recv_slpmsg(ns_addr_t src, int nbytes, const char
*msg = 0);
    void add_scope(const char * newscope);
    void remove_scope(const char * removescope);
        void xid_seed(void);
        u_int16_t xid_generate(void);
    void printReply(hdr_slp_srvrply* rply);
    void send_srvrqst_pkt(const char* service_type, nsaddr_t
destaddr);
    void cache_url(u_int16_t url_entry_count, const char* url);
    bool search_cached_urls(const char* service_type);
    void remove_cached_url(urlNode *removeurl);
    void discover_scopes(const char* scope_list, u_int16_t
length_scope);
    char * get_local_scopes() const;
    int  pktsize_;
    int32_tdstport_;
    int  maxSearch_;

```

```
    int    running_;
    int    number_of_scopes_;
    char * local_scopes_[SLP_MAX_SCOPE];
    rqstHashTable rqstTable_;
    urlNode * url_head;
};
```

A.1.5 slp-ua.h

```
#include "slp/slp-ua.h"

u_int16_t G_Xid = 0;
int rx_count = 0;
int rx_advertcount = 0;
RNG G_arrival;

// SLPua OTcl linkage class
static class SLPuaClass : public TclClass {
public:
    SLPuaClass() : TclClass("Application/SLPua") {}
    TclObject* create(int, const char*const*) {
        return (new SLPua);
    }
} class_slp_ua;

void reReqTimer::expire(Event *)
{
    s_->send_srvrqst_pkt(r_, a_);
    if (m_==REG)
        resched(i_);
    else
        resched(G_arrival.exponential(i_));
}

void urlCacheTimer::expire(Event *)
{
    s_->remove_cached_url(n_);
}

void retxTimer::expire(Event *)
{
    if (n_->retxok || !n_->gotreply)
        h_->update_resched_resend(n_);
    else //no further responses caused by the PR_LIST
        h_->remove_from_table (n_->xid);
}

// Constructor (also initialize instances of timers)
SLPua::SLPua() : running_(0) , rqstTable_(this)
{
    bind("pktsize_",&pktsize_);
    bind("dstport_",&dstport_);
    bind_bool("maxSearch_",&maxSearch_);
    number_of_scopes_ = 0;
    url_head = NULL;
}

// OTcl command interpreter
int SLPua::command(int argc, const char*const* argv)
```

```

{
    Tcl& tcl = Tcl::instance();

    if (argc == 3)
    {
        if (strcmp(argv[1], "attach-agent") == 0) {
            agent_ = (Agent*) TclObject::lookup(argv[2]);
            if (agent_ == 0) {
                tcl.resultf("no such agent %s", argv[2]);
                return(TCL_ERROR);
            }
            // Make sure the underlying agent support MM
            if(agent_>supportSLP()) {
                agent_>enableSLP();
            }
            else {
                tcl.resultf("agent \"%s\" does not support SLP Application",
                argv[2]);
                return(TCL_ERROR);
            }
            agent_>attachApp(this);
            return(TCL_OK);
        }
        else if (strcmp(argv[1], "add-scope") == 0)
        {
            add_scope(argv[2]);
            return(TCL_OK);
        }
        else if (strcmp(argv[1], "remove-scope") == 0)
        {
            remove_scope(argv[2]);
            return(TCL_OK);
        }
    }
    else if (((argc==4)|| (argc==6))&&(strcmp(argv[1], "rqst-service") ==
    0))
    {
        if (argc==6)
        {
            if (strcmp(argv[4], "-reg")==0)
                reReqTimer *resend=new reReqTimer(this, atof(argv[5]),
                (nsaddr_t)strtol(argv[3], (char **)NULL, 16), argv[2], REG);
            if (strcmp(argv[4], "-exp")==0)
                reReqTimer *resend=new reReqTimer(this, atof(argv[5]),
                (nsaddr_t)strtol(argv[3], (char **)NULL, 16), argv[2], EXP);
        }
        send_srvrqst_pkt(argv[2], (nsaddr_t)strtol(argv[3], (char **)NULL,
        16));

        return(TCL_OK);
    }
    return (Application::command(argc, argv));
}

void SLPua::init()
{
    //add_scope("DEFAULT");
    xid_seed();
}

```

```

void SLPua::add_scope(const char* newscope) {
    char *scope = small_string(newscope);
    for (int i=0; i<number_of_scopes_; i++)
    {
        if (strcmp(local_scopes_[i],scope)==0)
        { // already in list
            return;
        }
    }
    if (number_of_scopes_==SLP_MAX_SCOPE)
    {
        printf("Can't add anymore scopes, MAX=%d scopes per
SA\n",SLP_MAX_SCOPE);
        return;
    }
    local_scopes_[number_of_scopes_]=scope;
#ifdef DEBUGUA
    printf("UA:Added %s, total=%d Scope
\n",local_scopes_[number_of_scopes_], number_of_scopes_+1);
#endif
    number_of_scopes_++;
}

void SLPua::remove_scope(const char *removescope) {
    int i,j;
    char *scope = small_string(removescope);
    for (i=0; i<number_of_scopes_; i++)
    {
        if (strcmp(local_scopes_[i],scope)==0)
            break;
    }
    number_of_scopes_--;
    for (j=i; j<number_of_scopes_; j++)
    {
        local_scopes_[j]=strdup(local_scopes_[j+1]);
    }
}

void SLPua::cache_url(u_int16_t url_entry_count, const char* url) {

    urlNode *c;
    hdr_slp_url *addurl;
    int index=0;
    for (int i=0; i<url_entry_count; i++)
    {
        addurl=(hdr_slp_url *)(url + index);
        if (addurl->lifetime != 0)
        {
            #ifdef DEBUGUA
            printf("%f: Expiring after
%f\n",CURRENT_TIME(),(double)(addurl->lifetime)-CURRENT_TIME());
            #endif
            c = new urlNode(this, (double)(addurl->lifetime));
            (c->data).lifetime=addurl->lifetime;
            (c->data).url=strdup(addurl->url);
            (c->data).url_length=addurl->url_length;
            (c->data).auth_no=0;
            c->next = url_head;
            url_head = c;
        }
    }
}

```

```

        #ifdef DEBUGUA
        printf("UA:Cached %s %d\n", (url_head->data).url, (url_head->data).auth_no);
        #endif
    }
    index+=sizeof(hdr_slp_url);
}

static bool match(const char *url, const char *service)
{
    int position;
    char *token;
    char *temp1;
    char *temp2;
    temp1=strdup(url);
    token=strstr(temp1, "://");
    position = token - temp1;
    temp2=strndup(temp1, position);
    delete temp1;

    if (strstr(temp2, service) != NULL)
    {
        delete temp2;
        return true;
    }
    else
    {
        delete temp2;
        return false;
    }
}

bool SLPua::search_cached_urls(const char* service)
{
    bool matched=false;
    urlNode *c = url_head;
    int offset;
    Tcl& tcl = Tcl::instance();
    char wrk[5000];

    while (c!=NULL)
    {
        if (match((c->data).url, service)==true)
        {
            matched=true;
            tcl.evalf("puts $tracefd \"C n%d: %f****Cached service urls matches %s\\\"", agent_>addr(), CURRENT_TIME(), (c->data).url);
            return matched;
        }
        c=c->next;
    }
    return matched;
}

void SLPua::remove_cached_url(urlNode *removeurl) {
    urlNode *p=NULL, *c=url_head;
    while ((c!=NULL) && (c != removeurl)) {
        p=c; c=c->next;
    }
}

```

```

    if (c==NULL)
        return;
    if (p==NULL)
        url_head = c->next;
    else
        p->next = c->next;
    delete c;

    //we assume that a cached service doesn't timeout before
    MC_CONFIG_MAX, so we do
    //not care to remove it from the prlist if the request
    retransmission are still active
}

void SLPua::start()
{
    init();
    running_ = 1;
}

void SLPua::stop()
{
    running_ = 0;
}

char * SLPua::get_local_scopes() const
{
    char buf[21];
    char str[SLP_MAX_SCOPE*20]="";
    for (int i=0; i<number_of_scopes_; i++)
    {
        if (i==0) {
            sprintf(buf,"%s",local_scopes_[i]);
            strcat(str,buf);
        }
        else {
            sprintf(buf,"%s",local_scopes_[i]);
            strcat(str,buf);
        }
    }
    return strdup(str);
}

// Send application data packet
void SLPua::send_srvrqst_pkt(const char* service_type, nsaddr_t
destaddr)
{
    //first check if request service is cached
    if (search_cached_urls(service_type)==true)
        return;

    #ifdef DEBUG
        printf("No cached match...sending request\n");
    #endif

    //Otherwise, send a new service request
    ns_addr_t dest;
    dest.addr_=destaddr;
    dest.port_=dstport_;
    char * scopes = get_local_scopes();
    //printf("Dest is %d:%d\n",dest.addr_,dest.port_);
    hdr_slp_srvrqst srvrqst_buf;

```

```

if (running_) {
    // the below info is passed to UDPslp agent, which will write it
    // to SLP SRVRQST header after packet creation.
    srvrqst_buf.version = SLP_VERSION;
    srvrqst_buf.function_id = SLPTYPE_SRVRQST;
    srvrqst_buf.flags = SLP_REQUEST_MCAST;
    srvrqst_buf.next_ext_offset=0;
    srvrqst_buf.xid = xid_generate();
    srvrqst_buf.language_tag_length=strlen(SLP_LANGUAGE);
    srvrqst_buf.language_tag=SLP_LANGUAGE;
    srvrqst_buf.length_prlist=0;
    srvrqst_buf.length_service_type=strlen(service_type);
    srvrqst_buf.string_service_type=strdup(service_type);
    srvrqst_buf.length_scope_list=strlen(scopes);
    srvrqst_buf.string_scope_list=scopes;
    srvrqst_buf.length_predicate=0;
    srvrqst_buf.length_spi=0;
    srvrqst_buf.length=svrqst_buf.size();
#ifdef DEBUG
    printf("n%d: UA Sending Service Request (xid=%d) to
    %x:%d\n",agent_>addr(), srvrqst_buf.xid,dest.addr_,dest.port_);
#endif
    rqstTable_.add_resched_send(srvrqst_buf.xid, srvrqst_buf.length,
    (char*) &svrqst_buf, dest);
}
}

// Receive message from underlying agent
void SLPua::recv_slpmsg(ns_addr_t src, int nbytes, const char *msg)
{
#ifdef DEBUGUA
    printf("n%d: UA receiving message from %x:%d\n",agent_
    >addr(),src.addr_, src.port_);
#endif

    if(msg) {
        hdr_slp* slp_buf = (hdr_slp*) msg;

        if (slp_buf->function_id == SLPTYPE_SRVRPLY) {
            // If received packet is a request packet
            hdr_slp_srvrply* slp_srvrply_buf = (hdr_slp_srvrply*) msg;
            //print service reply here
#ifdef DEBUG
            printReply(slp_srvrply_buf);
#endif
            cache_url(slp_srvrply_buf->url_entry_count,slp_srvrply_buf-
            >url);
            if (maxSearch_)
                rqstTable_.add_pr(slp_srvrply_buf->xid, src.addr_);
            else
                rqstTable_.cancel_ret(x(slp_srvrply_buf->xid);
#ifdef DEBUGUA
            printf("n%d:String is %s\n",agent_
            >addr(),rqstTable_.get_pr(slp_srvrply_buf->xid));
#endif
        }
        else if (slp_buf->function_id == SLPTYPE_SAADVERT) {
            hdr_slp_saadvert* slp_saadvert_buf = (hdr_slp_saadvert*) msg;
            printf("n%d: ****Received a service advert xid=%d scopes=%s
            url=%s (adv count=%d)\n",
            agent_>addr(), slp_saadvert_buf->xid, slp_saadvert_buf-

```

```

>string_scope_list,slp_saadvert_buf->url,++rx_advertcount);
    if (maxSearch_)
        rqstTable_.add_pr(slp_saadvert_buf->xid, src.addr_);
    else
        rqstTable_.cancel_retx(slp_saadvert_buf->xid);

    //configure SA with scopes from the SAAdvert
    discover_scopes(slp_saadvert_buf->string_scope_list,
slp_saadvert_buf->length_scope_list);
    }
}

void SLPua::discover_scopes(const char* scope_list, u_int16_t
length_scope)
{
    if (length_scope==0)
        return;

    char *delimiters = ",";
    char *token;
    char scope[length_scope];
    strcpy(scope,scope_list);
    token=strtok(scope, delimiters);
    while (token!=NULL)
    {
        add_scope(token);
        token=strtok(NULL, delimiters);
    }
}

void SLPua::printReply (hdr_slp_srvrply* rply)
{
    printf("n%d: ****Received a service reply xid=%d url=%d
(count=%d)\n",agent_->addr(), rply->xid,
rply->url_entry_count,++rx_count);
    int index=0;
    for (int i=0; i<rply->url_entry_count; i++)
    {
        printf("%s*",((hdr_slp_url *)((rply->url) + index))->url);
        index+=sizeof(hdr_slp_url);
    }
    printf("\n");
}

void SLPua::xid_seed(void)
{
    G_Xid=0;
    //G_Xid = (u_int16_t)Random::uniform(65535);
}

u_int16_t SLPua::xid_generate(void)
{
    G_Xid++;
    return G_Xid;
    //return ((u_int16_t)Random::uniform(65535));
}

rqstHashTable::rqstHashTable(SLPua *s) {

```

```

s_ = s;
tablesize = SLP_MAX_OUTSTANDING*3;
table = new rqstNode* [tablesize];
for (int i=0; i<tablesize; i++) {
    table[i] = NULL;
}
}

int rqstHashTable::hash (u_int16_t xid) const
{
    return (xid%tablesize);
}

void rqstHashTable::add_pr(u_int16_t xid, int newaddress)
{
    rqstNode *c;
    int key=hash(xid);
    for (c=table[key]; c!=NULL; c=c->next) {
        if (c->xid == xid) {
            add_local_pr(c, newaddress);
            return;
        }
    }
}

void rqstHashTable::cancel_retx(u_int16_t xid)
{
    rqstNode *c;
    int key=hash(xid);
    for (c=table[key]; c!=NULL; c=c->next) {
        if (c->xid == xid) {
            //cancel retransmissions
            c->retx_timer_.cancel();
            //remove node from hash table
            remove_from_table (xid);
            return;
        }
    }
}

void rqstHashTable::add_local_pr(rqstNode* c, int newaddress)
{
    for (int i=0; i<c->number_of_pr; i++)
    {
        if (c->local_pr[i]==newaddress)
        { // already in list
            return;
        }
    }
    if (c->number_of_pr==SLP_MAX_PRLIST)
    {
        printf("Can't add anymore prev. responders, MAX=%d pr per
SA\n", SLP_MAX_PRLIST);
        return;
    }
    c->local_pr[c->number_of_pr]=newaddress;
    c->number_of_pr++;
    c->retxok=true;
    c->gotreply=true;
}

char * rqstHashTable::get_pr(u_int16_t xid) const
{

```

```

    rqstNode *c;
    int key = hash(xid);
    for (c=table[key]; c!=NULL; c=c->next) {
        if (c->xid == xid)
            return (get_local_pr(c));
    }
    return ("");
}

char * rqstHashTable::get_local_pr(const rqstNode* c) const
{
    char buf[21];
    char str[SLP_MAX_PRLIST*21]="";
    for (int i=0; i<c->number_of_pr; i++)
    {
        if (i==0) {
            sprintf(buf, "%d", c->local_pr[i]);
            strcat(str, buf);
        }
        else {
            sprintf(buf, ",%d", c->local_pr[i]);
            strcat(str, buf);
        }
    }
    return strdup(str);
}

void rqstHashTable::add_resched_send(u_int16_t xid, int size, const
char *mssg, ns_addr_t dst)
{
    /* if size exceeds SLP pkt size, then do nothing */
    if (size > s_>pktsize_) {
        #ifdef DEBUGUA
            printf("Will not send request, size=%d, max=%d\n", size, s_
>pktsize_);
        #endif
        return;
    }
    /*adding new node*/
    rqstNode *c;
    int key=hash(xid);
    c = new rqstNode(this);
    c->xid = xid;
    c->number_of_pr = 0;
    c->next = table[key];
    memcpy(&c->msg, mssg, sizeof(hdr_slp_srvrqst));
    c->dest=dst;
    c->timeout=SLP_INITIAL_TIMEOUT;
    c->expire=CURRENT_TIME()+SLP_CONFIG_MC_MAX;
    table[key]=c;

    /*rescheduling packet*/
    c->retx_timer_.resched(c->timeout);
    c->retxok=false;
    c->gotreply=false;
    /*sending packet*/
    s_>agent_>sendto(size, c->msg, c->dest); // send to UDP

    #ifdef DEBUGUA
        printf("(xid=%d) Current Time=%f, next retx=%f, expire=%f\n", c-
>xid, CURRENT_TIME(), c->timeout+CURRENT_TIME(),

```

```

        c->expire);
    #endif
}
void rqstHashTable::update_resched_resend(rqstNode *n)
{
    //retransmit until CONFIG_MC_MAX is reached
    if (CURRENT_TIME() >= n->expire)
    {
        //remove this request history from hashtable
        remove_from_table (n->xid);
        return;
    }

    hdr_slp_srvrqst* srvrqst_buf = (hdr_slp_srvrqst*) n->msg;

    //adding the prlist to srvrqst packet before its resent
    char *pr=get_local_pr(n);
    srvrqst_buf->length_prlist=strlen(pr);
    srvrqst_buf->string_prlist=pr;
    srvrqst_buf->length=srvrqst_buf->size();

    n->retxok=false;

    //retransmit until request will no longer fit in a single datagram
    if (srvrqst_buf->size() > s_>pktsize_) {
        #ifdef DEBUGUA
            printf("Will not send request, size=%d, max=%d\n",srvrqst_buf->size(),s_>pktsize_);
        #endif
        return;
    }

    #ifdef DEBUGUA
        printf("R(xid=%d) Current Time=%f, next retx=%f\n",n->xid,
CURRENT_TIME(), n->timeout+(CURRENT_TIME()*2.0));
    #endif

    /*Rescheduling*/
    n->retx_timer_.resched(n->timeout*=2.0);

    ///*Re-sending*/
    s_>agent_>sendto(srvrqst_buf->length,
                    n->msg,
                    n->dest);
}
void rqstHashTable::remove_from_table (u_int16_t xid)
{
    int key=hash(xid);
    rqstNode *p=NULL, *c=table[key];
    while ((c!=NULL) && (xid!=c->xid)) {
        p=c; c=c->next;
    }
    if (c==NULL) {
        printf("Error: Didn't find the node to be deleted from hash
table\n");
        return;
    }

    //now adjust pointers
    if (p==NULL) //Node to be deleted is the first node

```

```
    table[key] = c->next;
else
    p->next = c->next;

delete c;
}
```

A.1.6 slp-ua.cc

```
//#define DEBUG
//#define DEBUGSA
//#define DEBUGUA
//#define DEBUGUDP

#define BCAST
#define MULTICAST_ADDR 0xE0000000

#ifndef ns_udp_slp_h
#define ns_udp_slp_h

#include "udp.h"
#include "ip.h"
#include "random.h"
#include "packet.h"
#include "app.h"
#include "timer-handler.h"
#include "rtp.h"
#include <string.h>
#include <ctype.h>

#define HDR_SLP(p) ((struct hdr_slp*)hdr_slp::access(p))
#define HDR_SLP_SRVRQST(p) ((struct hdr_slp_srvrqst*)hdr_slp::access(p))
#define HDR_SLP_SRVRPLY(p) ((struct hdr_slp_srvrply*)hdr_slp::access(p))
#define HDR_SLP_SAADVERT(p) ((struct hdr_slp_saadvert*)hdr_slp::access(p))

#define CURRENT_TIME Scheduler::instance().clock

#define SLP_MAX_LANGUAGE 1
#define SLP_MAX_PRLIST 10000
#define SLP_MAX_OUTSTANDING 1000 // number of outstanding requests
#define SLP_MAX_TYPE 1
#define SLP_MAX_SCOPE 10
#define SLP_MAX_SERVICE 10
#define SLP_MAX_PREDICATE 1
#define SLP_MAX_SPI 1
#define SLP_MAX_URL 1

#define SLP_INITIAL_TIMEOUT (double)2.0
#define SLP_CONFIG_MC_MAX (double)15.0

#define SLP_VERSION 2
#define SLP_LANGUAGE "en"
#define SLP_REQUEST_MCAST 0x2000 //bit mask of 00100000 00000000
#define SLP_OVERFLOW 0x8000 //bit mask of 10000000 00000000
```

```

#define REG          1
#define EXP          2

//packet formats
#define SLPTYPE_SRVRQST 1
#define SLPTYPE_SRVRPLY 2
#define SLPTYPE_SAADVERT 11

static char *small_string(const char *string)
{
    char *newstring = (char *)malloc(sizeof(char)*strlen(string));
    for (int i=0; i<strlen(string); i++)
        newstring[i]=(char)tolower((int)string[i]);
    newstring[strlen(string)]='\0';
    return newstring;
}

//SLP common Header
struct hdr_slp {
    u_int8_t      version;
    u_int8_t      function_id;
    u_int32_t     length : 24;
    u_int16_t     flags;
    u_int32_t     next_ext_offset : 24; //24 bits = 3 byte field
    u_int16_t     xid;
    u_int16_t     language_tag_length;
    const char*   language_tag;

    // Header access methods
    static int offset_; // required by PacketHeaderManager
    inline static int& offset() { return offset_; }
    inline static hdr_slp* access(const Packet* p) {
        return (hdr_slp*) p->access(offset_);
    }
};

// srvRqst header
struct hdr_slp_srvrqst {
    /*common slp header*****/
    u_int8_t      version;
    u_int8_t      function_id;
    u_int32_t     length : 24;
    u_int16_t     flags;
    u_int32_t     next_ext_offset : 24;
    u_int16_t     xid;
    u_int16_t     language_tag_length;
    const char*   language_tag;
    /*common slp header*****/
    u_int16_t     length_prlist;
    const char*   string_prlist;
    u_int16_t     length_service_type;
    const char*   string_service_type;
    u_int16_t     length_scope_list;
    const char*   string_scope_list;
    u_int16_t     length_predicate;
    const char*   string_predicate;
    u_int16_t     length_spi;
    const char*   string_slp;

    inline int size() {
        int sz = (2*sizeof(u_int8_t) + 8*sizeof(u_int16_t) +

```

```

2*3
        +((language_tag_length+length_prlist+length_service_type
+length_scope_list+length_predicate+length_spi)*sizeof(char)));
        return sz;
    }
};

//url entries
struct hdr_slp_url {
    u_int8_t      reserved;
    u_int16_t     lifetime;
    u_int16_t     url_length;
    const char*   url;
    u_int8_t      auth_no;
    const char*   auth;

    inline int size() {
        int sz = (2*sizeof(u_int8_t) + 2*sizeof(u_int16_t) +
((url_length+auth_no)*sizeof(char)));
        return sz;
    }
};

// srvRply header
struct hdr_slp_srvrply {
    /*common slp header*****/
    u_int8_t      version;
    u_int8_t      function_id;
    u_int32_t     length : 24;
    u_int16_t     flags;
    u_int32_t     next_ext_offset : 24;
    u_int16_t     xid;
    u_int16_t     language_tag_length;
    const char*   language_tag;
    /*common slp header*****/
    u_int16_t     error_code;
    u_int16_t     url_entry_count;
    const char*   url;

    inline int url_size()
    {
        int sz=0;
        char *ptr = (char *)url;
        int index=0;
        for (int i=0; i<url_entry_count; i++)
        {
            sz += (((hdr_slp_url *) (url+index))->size());
            index += sizeof(hdr_slp_url);
        }
        return sz;
    }
    inline int size() {
        int sz = (2*sizeof(u_int8_t) + 5*sizeof(u_int16_t) + 2*3
+url_size());
        return sz;
    }
};

```

```
// SAdvert header
struct hdr_slp_saadvert {
    /*common slp header*****/
    u_int8_t      version;
    u_int8_t      function_id;
    u_int32_t     length : 24;
    u_int16_t     flags;
    u_int32_t     next_ext_offset : 24;
    u_int16_t     xid;
    u_int16_t     language_tag_length;
    const char*   language_tag;
    /*common slp header*****/
    u_int16_t     length_url;
    const char*   url;
    u_int16_t     length_scope_list;
    const char*   string_scope_list;
    u_int16_t     length_attr_list;
    const char*   string_attr_list;
    u_int8_t      auth_no;
    const char*   auth;

    inline int size() {
        int sz = 3*sizeof(u_int8_t) + 6*sizeof(u_int16_t) + 2*3
                +(
(language_tag_length+length_url+length_scope_list
+length_attr_list+auth_no)*sizeof(char));
        return sz;
    }
};

// for size calculation of header-space reservation
union hdr_slp_all {
    hdr_slp      sh;
    hdr_slp_srvrqst  srvrqst;
    hdr_slp_srvrply  srvrply;
    hdr_slp_saadvert saadvert;
};

#endif
```

A.1.7 slp-common.h

Για να υλοποιήσουμε το Service Location Protocol για MANET, είναι απαραίτητο να χρησιμοποιήσουμε ένα πρωτόκολλο δρομολόγησης για MANET που θα υποστηρίζει πολλαπλή (multicast) ή καθολική (broadcast) εκπομπή καθώς και μοναδική εκπομπή προς έναν μόνο κόμβο (unicast). Γι' αυτό το σκοπό, χρησιμοποιήσαμε ένα BroadCAST πρωτόκολλο δρομολόγησης για κινητά ad hoc δίκτυα (BCAST), που αναπτύχθηκε από τον Kunz [30]. Μία υλοποίηση του BCAST στον NS-2 είναι διαθέσιμη στο [31].

Ωστόσο, το BCAST δεν υποστηρίζει unicast εκπομπές. Το SLP για MANETs απαιτεί ένας Service Agent να εκπέμπει μοναδικά μηνύματα SrvRply προς τον αιτούντα User Agent. Για να ξεπεράσουμε αυτό το εμπόδιο, κάναμε κάθε UA να ενσωματώνεται σε ένα BCAST γκρουπ της επιλογής του. Ο SA θα στέλνει έτσι τις

απαντήσεις του στο γκρουπ, που αποτελείται από έναν μόνο κόμβο, του αιτούντος UA. Για παράδειγμα, ένας UA που είναι προσαρτημένος στον κόμβο 4 θα ενσωματωθεί σε ένα BCAST γκρουπ της επιλογής του και η διεύθυνση του γκρουπ θα είναι 0xE000004. Όταν κάποιος SA λάβει μηνύματα Service Requests (SrvRqsts) από τον κόμβο 4, θα εκπέμψει broadcast τις απαντήσεις του (SrvRplys) στον κόμβο-γκρουπ 0xE000004. Το μειονέκτημα αυτής της υλοποίησης είναι η ανταλλαγή επιπλέον μηνυμάτων δρομολόγησης από τους πράκτορες BCAST.

Το SLP για τα κινητά ad hoc περιβάλλοντα που μελετάμε, απαιτεί τα πακέτα να μεταδίδονται μέσω UDP. Από τη στιγμή που ένας UDP agent κατανέμει και στέλνει πακέτα δικτύου, όλη η πληροφορία που χρειάζεται για την επικοινωνία των εφαρμογών πρέπει να χειρίζεται στον UDP agent σαν ροή δεδομένων. Δυστυχώς, η UDP υλοποίηση στον NS-2 κατανέμει μόνο τα πακέτα που περιέχουν επικεφαλίδα. Έτσι, έπρεπε να τροποποιήσουμε την υλοποίηση UDP ώστε να χειρίζεται σωστά τα δεδομένα εφαρμογών που στέλνονται από και προς τους slp πράκτορες.

A.2 Τροποποίηση αρχείων για υποστήριξη του SLP

Τα αρχεία που θέλουν προσθήκες είναι τα εξής:

Όνομα αρχείου	Προορισμός
packet.h	\ns-allinone-2.30\ns-2.30\common
agent.h	\ns-allinone-2.30\ns-2.30\common
app.h	\ns-allinone-2.30\ns-2.30\apps
ns-packet.tcl	\ns-allinone-2.30\ns-2.30\tcl\lib
ns-default.tcl	\ns-allinone-2.30\ns-2.30\tcl\lib
cmu-trace.h	\ns-allinone-2.30\ns-2.30\trace
cmu-trace.cc	\ns-allinone-2.30\ns-2.30\trace

Πίνακας A.2.1 Αρχεία προς τροποποίηση

Οι αλλαγές είναι οι εξής:

```
.
.
PT_DSR,
PT_AODV,
PT_IMEP,

/* Multicast protocols */
PT_BCAST,

// RAP packets
PT_RAP_DATA,
.
.

// SMAC packet
PT_SMAC,
```

```
// SLP packet
PT_SLP,

// insert new packet types here
PT_NTYPE // This MUST be the LAST one
.
.
// BCAST
name_[PT_BCAST] = "BCAST";
.
.
// SLP
name_[PT_SLP]= "SLP";
.
.
```

A.2.2. packet.h

```
.
.
void send(Packet* p, Handler* h) { target_->recv(p, h); }
virtual void timeout(int tno);

// SLP
virtual int supportSLP() { return 0; }
virtual void enableSLP() {}
.
.
```

A.2.3. agent.h

```
.
.
virtual void resume();

// SLP
virtual void recv_slpmsg(ns_addr_t src, int nbytes, const char *msg = 0) {};
.
.
```

A2.4. app.h

```
.
.
foreach prot {
    AODV
    // BCAST
    BCAST
    ARP
.
.
```

```
SCTP
Smac
// SLP
SLP
NV
}
.
.
```

A.2.5. ns-packet.tcl

```
.
.
# SLP
Agent/UDP/UDPSlp set ttl_ 255
Agent/UDP/UDPSlp set packetSize_ 1400
Application/SLPsa set pktsize_ 1400
Application/SLPua set pktsize_ 1400
Application/SLPua set dstport_ 18
Application/SLPua set maxSearch_ false
.
.
```

A.2.6. ns-default.tcl

```
.
.
#define DROP_MAC_BUSY "BSY"
#define DROP_MAC_INVALID_DST "DST"
#define DROP_MAC_SLEEP "SLP" // smac sleep state
#define DROP_RTR_NO_ROUTE "NRTE" // no route
.
.
/* trace formats for multicast protocols... */
// BCAST
void format_bcast(Packet *p, int offset);
// SLP
void format_slp(Packet *p, int offset);
};
.
.
```

A.2.7. cmu-trace.h

```

.
.
/* packet headers for multicast protocols */
#include <bcast/bcast_packet.h> //BCAST

// SLP
#include <slp/slp-common.h>
.
.
void
CMUTrace::format_aadv(Packet *p, int offset)
{
.
.
    if (pt_>tagged()) {
        sprintf(pt_>buffer() + offset,
            "-aadv:t %x -aadv:h %d -aadv:b %d -aadv:d %d "
            "-aadv:ds %d -aadv:s %d -aadv:ss %d "
            "-aadv:c REQUEST ",
            rq->rq_type,
            rq->rq_hop_count,
            rq->rq_bcast_id,
.
.
    } else {
        sprintf(pt_>buffer() + offset,
            "[0x%x %d %d [%d %d] [%d %d]] (REQUEST)",
            rq->rq_type,
            rq->rq_hop_count,
            rq->rq_bcast_id,
.
.
void
CMUTrace::format_bcast(Packet *p, int offset)
{
    struct hdr_bcast *bh = HDR_BCAST(p);

    switch(bh->type) {

    case BCASTTYPE_HELLO:
    case BCASTTYPE_DATA:
    case BCASTTYPE_NACK:

        if (pt_>tagged()) {
            // Need to determine tag names for this data
            //sprintf(pt_>buffer() + offset,
            //    "",
            //    );
        } else if (newtrace_) {
            sprintf(pt_>buffer() + offset,
                "-P bcast -Pt 0x%x -Ph %d -Pd %d -Pds %d -Ps
%d -Pss %d -Pl %f -Pc %s ",
                bh->type,
                bh->hop_count,
                bh->dst,
                bh->dst_seqno,
                bh->src,
                bh->src_seqno,
                bh->lifetime,
                bh->type == BCASTTYPE_DATA ? "DATA" :

```

```

(bh->type == BCASTTYPE_HELLO ? "HELLO" :
"NACK"));
        } else {
            sprintf(pt_>buffer() + offset,
                "[0x%x %d [%d %d] [%d %d] %f] (%s)",
                bh->type,
                bh->hop_count,
                bh->dst,
                bh->dst_seqno,
                bh->src,
                bh->src_seqno,
                bh->lifetime,
                bh->type == BCASTTYPE_DATA ? "DATA" :
                (bh->type == BCASTTYPE_HELLO ? "HELLO" :
                "NA"));
        }
        break;

    default:
#ifdef WIN32
        fprintf(stderr,
            "CMUTrace::format_bcast: invalid BCAST packet
type\n");
#else
        fprintf(stderr,
            "%s: invalid BCAST packet type\n", __FUNCTION__);
#endif
        abort();
    }
}
.
.
void
CMUTrace::format_slp(Packet *p, int offset)
{
    struct hdr_slp *sh = HDR_SLP(p);

    if (newtrace_) {
        sprintf(pt_>buffer() + offset,
            "-P slp -Px %d",
            sh->xid);
    }
    else {
        sprintf(pt_>buffer() + offset,
            "%d ",
            sh->xid);
    }

    offset = strlen(pt_>buffer());

    switch(sh->function_id) {

    case SLPTYPE_SRVRQST:
        sprintf(pt_>buffer() + offset, "%s ", "srvRqst");
        break;
    case SLPTYPE_SRVRPLY:
        sprintf(pt_>buffer() + offset, "%s ", "srvRply");
        break;
    case SLPTYPE_SAADVERT:

```

```
        sprintf(pt_>buffer() + offset, "%s ", "srvAdvert");
        break;

    default:
        fprintf(stderr,
            "CMUTrace::format_slp: invalid SLP packet
type\n");
        abort();
    }
}
.
.
void CMUTrace::format(Packet* p, const char *why)
{
    .
    .
    if (pt_>namchannel())
        nam_format(p, offset);
    offset = strlen(pt_>buffer());
    switch(ch->ptype()) {
    case PT_MAC:
    case PT_SMAC:
        break;
    case PT_ARP:
        format_arp(p, offset);
        break;
    default:
        format_ip(p, offset);
        offset = strlen(pt_>buffer());
        switch(ch->ptype()) {
            case PT_SLP:
                format_slp(p, offset);
                break;

            case PT_AODV:
                format_aodv(p, offset);
                break;
            case PT_BCAST:
                format_bcast(p, offset);
                break;
        }
    }
    .
    .
}
```

A.2.8. cmu-trace.cc

Επίσης, στη λίστα μεταγλώττισης του αρχείου Makefile.in, πρέπει να προστεθούν τα εξής:

```
slp/udp-slp.o,
slp/slp-sa.o,
slp/slp-ua.o
```


ΠΑΡΑΡΤΗΜΑ Β

ΣΕΝΑΡΙΑ ΠΡΟΣΟΜΟΙΩΣΗΣ

B.1 Περιγραφή – Παράμετροι

Αρχικά, ορίζουμε τις παραμέτρους που αφορούν κάθε ένα από τα τρία σενάρια που υλοποιήσαμε καθώς και περιγράφουμε που μπορούν να βρουν πρακτική εφαρμογή. Τα στοιχεία που απαιτούνται για την υλοποίηση είναι:

- Περιοχή προσομοίωσης. Πρόκειται για το μέγεθος του χώρου στο οποίο οι κόμβοι επιτρέπονται να κινούνται (σε μέτρα).
- Μοντέλο κινητικότητας. Υλοποιείται το Random Waypoint Mobility Model [20].
- Μέγεθος δικτύου (Network Size – NS). Ο συνολικός αριθμός των κόμβων που απαρτίζουν το MANET.
- Αναλογία αντιπροσώπων υπηρεσιών (Service Agent Ratio – SR). Το κλάσμα των συνολικών κόμβων που είναι service agents.
- Αναλογία αντιπροσώπων χρηστών (User Agent Ratio – UR). Το κλάσμα των συνολικών κόμβων που είναι user agents.
- Υπηρεσίες ανά πάροχο υπηρεσιών (services per server node – SP). Ο αριθμός των ξεχωριστών υπηρεσιών που προσφέρει ο κόμβος – πάροχος.
- Διάρκεια ζωής διαφήμισης υπηρεσίας (Service Advertisement Lifetime – SL). Πρόκειται για τη διάρκεια (σε δευτερόλεπτα) για την οποία είναι έγκυρο (και αποθηκεύεται προσωρινά) το URL της υπηρεσίας, το οποίο διαφημίζεται σε ένα μήνυμα SrvRply.
- Ταυτόχρονες αιτήσεις (Simultaneous Requests - SQ). Το ποσοστό των UAs που ζητούν υπηρεσίες ταυτόχρονα.
- Αναλογία πανομοιότυπων υπηρεσιών (Service Duplication ratio – SD). Το κλάσμα των SAs που προσφέρουν πανομοιότυπες υπηρεσίες.

Σενάριο 1 – Συνεδρίαση σε εταιρεία

Πρόκειται για ένα κινητό ad hoc δίκτυο που θα καλύπτει τις ανάγκες μιας συνεδρίασης σε κάποια εταιρεία. Ο παρουσιαστής που θα κάνει κάποια ανάλυση, θα μπορεί να μεταδίδει τις διαφάνειές του σε πολλούς συμμετέχοντες που είτε βρίσκονται μέσα στην αίθουσα είτε θα μπορούσαν να βρίσκονται στα γραφεία τους σε άλλο όροφο. Επιπλέον, με χρήση video, οι εταίροι θα μπορούν να κάνουν ερωτήσεις και να λαμβάνουν το λόγο. Ακόμη, τους δίνεται η δυνατότητα να ανταλλάσουν γρήγορα ηλεκτρονικά αρχεία και να χρησιμοποιούν κοινόχρηστους πόρους της αίθουσας.

Από άποψη κινητικότητας, οι συμμετέχοντες είναι σχεδόν στατικοί. Όταν κινούνται έχουν πολύ χαμηλές ταχύτητες και υπάρχουν μεγάλες παύσεις μεταξύ των κινήσεών τους. Συνεπώς, το μοντέλο Random Waypoint περιγράφει ικανοποιητικά αυτή τη κίνηση.

Ορίζοντας ως $\bar{v}$, τη μέση ταχύτητα σε m/s και ως Δv την απόκλιση από τη μέση τιμή. Αντίστοιχα, ορίζουμε $\bar{\pi}$ τη μέση τιμή παύσεων μεταξύ των κινήσεων και ως $\Delta \pi$ την απόκλιση των παύσεων (σε sec).

Συγκεκριμένα, στο σενάριο μας, θεωρούμε τη μέση ταχύτητα $\bar{v}$ είναι ίση με 0.60 ενώ ορίζουμε την απόκλιση $\Delta v = 0.50$. Η διάρκεια των παύσεων θα είναι 60 sec με άλλα 60 sec απόκλιση.

Ως περιοχή προσομοίωσης θέτουμε μία έκταση $20m \times 10m$, το μέγεθος του δικτύου είναι 10 κόμβοι, ορίζοντας 3 από αυτούς ως παρόχους υπηρεσιών, 5 ως χρήστες και τους υπόλοιπους απλά να μεταδίδουν ότι πακέτα παίρνουν χωρίς να συμμετέχουν στη διαδικασία ανακάλυψης υπηρεσίας. Σε κάθε πάροχο αντιστοιχούμε 1 υπηρεσία, η διάρκεια ζωής της διαφήμισης τίθεται στα 60 sec ενώ θέτουμε 2 χρήστες να αιτούνται ταυτόχρονα μία υπηρεσία. Τέλος, 2 από τους service agents, παρέχουν πανομοιότυπες υπηρεσίες.

Σενάριο 2 – Δίκτυο αυτοκινήτων

Στη περίπτωση αυτή θα προσομοιώσουμε ένα δίκτυο που έχει στηθεί σε οδικό δίκτυο μεταξύ σταθερών σταθμών βάσης και κινούμενων αυτοκινήτων κατάλληλα εξοπλισμένων. Οι σταθμοί βάσης ενημερώνουν τα διερχόμενα αυτοκίνητα σχετικά με την κυκλοφοριακή συμφόρηση ή σε πόση απόσταση υπάρχει πρατήριο καυσίμων ή κάποιο άλλο κατάσταση που τυχόν ζητήσει ένας οδηγός. Επιπλέον, υπάρχει και η δυνατότητα επικοινωνίας μεταξύ των αυτοκινήτων. Τέλος, υποθέτουμε ότι ένας σταθμός ή ένα σταθερό όχημα, λειτουργεί ως «office car» παρέχοντας εκτυπωτές, fax, τηλέφωνα για τους διερχόμενους οδηγούς.

Μοντελοποιώντας αυτό το σενάριο, παραθέτουμε τις παραμέτρους κίνησης, καθώς και τις τιμές των άλλων δικτυακών παραμέτρων, στους παρακάτω πίνακες 6.1 και 6.2, όπου φαίνονται οι παράμετροι για όλα τα σενάρια.

Σενάριο 3 – Mesh networks

Χωρίς ανάγκη καλωδίωσης, αναπτύσσονται τα mesh networks όπου και ολόκληρη κοινότητα μπορεί να επικοινωνεί ταχύτερα και ευκολότερα.

Ο αριθμός των κόμβων που απαρτίζουν ένα τέτοιο δίκτυο είναι μεγάλος, ωστόσο οι κόμβοι είναι σχεδόν στατικοί (κάτοικοι μιας κοινότητας), κινούνται με πολύ μικρές ταχύτητες και σταματούν για πολύ μεγάλα χρονικά διαστήματα

Συγκεντρώνοντας τις τιμές των παραμέτρων που αφορούν τα τρία παραπάνω σενάρια, παραθέτουμε τους εξής πίνακες:

Εφαρμογή	Χαρακτηριστικά	Μοντέλο κινητικότητας	Μέση ταχύτητα $\bar{v}$ (m/s)	Δv (m/s)	Μέση διάρκεια παύσεων $\bar{\pi}$ (sec)	$\Delta \pi$ (sec)
(1) Συνεδρίαση	Ανθρώπινες ταχύτητες, μεγάλες παύσεις	Random Waypoint	0.60	0.50	60	60
(2) Δίκτυο αυτοκινήτων	Μέτριες ταχύτητες οχημάτων, μικρές παύσεις	Random Waypoint	12.00	11.00	10	10
(3) Mesh networks	Σχεδόν στασιμότητα, μεγάλες παύσεις	Random Waypoint	0.30	0.25	250	0

Πίνακας B.1.1 Χαρακτηριστικά κίνησης σεναρίων

Εφαρμογή	Περιοχή Προσομοίωσης (m×m)	NS	SR	UR	SP	SL (sec)	SQ	SD
(1) Συνεδρίαση	20×10	10	3	5	1	60	2	2
(2) Δίκτυο αυτοκινήτων	1200×1200	70	25	30	1	10	10	20
(3) Mesh networks	3000×3000	300	100	100	1	100	10	100

Πίνακας B.1.2 Δικτυακοί παράμετροι σεναρίων

B.2 Κώδικες σεναρίων

Παραθέτουμε τους κώδικες των σεναρίων που υλοποιήσαμε στον NS-2.

```

/*****
**
*   File Name: mobgen.c
*   Purpose: Random Waypoint mobility model generator.
*****/

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>

#undef rand48

#define true 1
#define false 0

```

```

int main(int argc, char *argv[])
{
    int i;
    int numNodes=0;
    int nextNode=0;
    double maxX=0.0, maxY=0.0;
    double endTime=0.0, lowest;
    double speedMean=0.0, speedDelta=0.0;
    double pauseMean=0.0, pauseDelta=0.0;
    int *moving;
    double *nextEvent;
    double *xLoc, *yLoc;
    double newX, newY, speed, dist;
    double speedLow, pauseLow;
    double speedRange, pauseRange;
    char output;

    if (argc == 10)
    {
        numNodes    = atoi(argv[1]);
        maxX        = atof(argv[2]);
        maxY        = atof(argv[3]);
        endTime     = atof(argv[4]);
        speedMean   = atof(argv[5]);
        speedDelta  = atof(argv[6]);
        pauseMean   = atof(argv[7]);
        pauseDelta  = atof(argv[8]);
        output      = argv[9][0];

        fprintf(stdout, "#~~~~~\n");
        fprintf(stdout, "#\tnumNodes    = %6d\n", numNodes);
        fprintf(stdout, "#\tmaxX      = %9.2f\n", maxX);
        fprintf(stdout, "#\tmaxY      = %9.2f\n", maxY);
        fprintf(stdout, "#\tendTime   = %9.2f\n", endTime);
        fprintf(stdout, "#\tspeedMean = %9.2f\n", speedMean);
        fprintf(stdout, "#\tspeedDelta = %9.2f\n", speedDelta);
        fprintf(stdout, "#\tpauseMean = %9.2f\n", pauseMean);
        fprintf(stdout, "#\tpauseDelta = %9.2f\n", pauseDelta);
        fprintf(stdout, "#\toutput    = %6c\n", output);
        fprintf(stdout, "#~~~~~\n\n");
    } else {
        fprintf(stdout, "Usage:  mobgen <number of nodes>\n");
        fprintf(stdout, "        <max-x> <max-y> <end time>\n");
        fprintf(stdout, "        <speed mean> <speed delta>\n");
        fprintf(stdout, "        <pause time> <pause time\n");
        fprintf(stdout, "        <'N' or 'G'>\n");
        fprintf(stdout, "        'N' implies NS2 mobility\n");
        fprintf(stdout, "        'G' implies gnuplot path\n");
        return -1;
    }

    if (output == 'N')
    {
        fprintf(stdout, "# output format is NS2\n");
    } else if (output == 'G') {
        fprintf(stdout, "# output format is gnuplot\n");
        if (numNodes != 1)

```

```

{
    fprintf(stderr, "Gnuplot output is only possible with one
mobile node.\n");
    return -1;
} else {
    fprintf(stdout, "plot \'-\' notitle with linespoints\n");
}
} else {
    fprintf(stderr, "Unknown output type requested\n");
    return -1;
}

#ifdef rand48
    srand48((int)time(NULL));
#else
    srand((int)time(NULL));
#endif
speedLow = speedMean-speedDelta;
pauseLow = pauseMean-pauseDelta;
speedRange = 2*speedDelta;
pauseRange = 2*pauseDelta;

moving = (int*)malloc(sizeof(int)*numNodes);
nextEvent = (double*)malloc(sizeof(double)*numNodes);
xLoc = (double*)malloc(sizeof(double)*numNodes);
yLoc = (double*)malloc(sizeof(double)*numNodes);

fprintf(stdout, "#\tInitial positions:\n");
for (i=0; i<numNodes; i++)
{
    #ifdef rand48
        moving[i] = (drand48())>=0.5?true:false;
        xLoc[i] = drand48()*maxX;
        yLoc[i] = drand48()*maxY;
    #else
        moving[i] = (rand())>=(RAND_MAX/2)?true:false;
        /*
        fprintf(stdout, "$node_(%d) is ", i);
        if (moving[i])
        {
            fprintf(stderr, "moving\n");
        } else {
            fprintf(stderr, "not moving\n");
        }
        */
        xLoc[i] = ((double)rand())/((double)RAND_MAX)*maxX;
        yLoc[i] = ((double)rand())/((double)RAND_MAX)*maxY;
    #endif
    /*fprintf(stdout, "%d:  %s moving\n", i, ((moving[i])?"is
not":"is"))*/
    if (output == 'N')
    {
        fprintf(stdout, "$node_(%d) set X_ %.12f\n", i, xLoc[i]);
        fprintf(stdout, "$node_(%d) set Y_ %.12f\n", i, yLoc[i]);
        fprintf(stdout, "$node_(%d) set Z_ %.12f\n", i, 0.0);
    } else if (output == 'G') {
        xLoc[i] = maxX/2.0;
        yLoc[i] = maxY/2.0;
        fprintf(stdout, "%.12f %.12f\n", xLoc[i], yLoc[i]);
    }
}

```

```

    nextEvent[i] = 0.0;
}
if (output == 'N')
{
    fprintf(stdout, "\n\n#\tMovements:\n");
}

lowest = endTime; /* initialize high so all starting movements are
                    scheduled and output */
while (lowest <= endTime)
{
    /* schedule the next event */
    if (moving[nextNode])
    {
        moving[nextNode] = false;
        speed = 0.0;
        if (output == 'N')
        {
            fprintf(stdout, "$ns_ at %.12f \"$node_(%d) setdest %.12f
%.12f %.12f\n",
                    nextEvent[nextNode], nextNode,
xLoc[nextNode], yLoc[nextNode], speed);
        }
        #ifdef rand48
            nextEvent[nextNode] += drand48()*pauseRange + pauseLow;
        #else
            nextEvent[nextNode] +=
((double)rand())/((double)RAND_MAX)*pauseRange + pauseLow;
        #endif
    } else /* not moving */ {
        moving[nextNode] = true;
        #ifdef rand48
            speed = drand48()*speedRange + speedLow;
            newX = drand48()*maxX;
            newY = drand48()*maxY;
        #else
            speed = ((double)rand())/((double)RAND_MAX)*speedRange +
speedLow;
            newX = ((double)rand())/((double)RAND_MAX)*maxX;
            newY = ((double)rand())/((double)RAND_MAX)*maxY;
        #endif
        if (output == 'N')
        {
            fprintf(stdout, "$ns_ at %.12f \"$node_(%d) setdest %.12f
%.12f %.12f\n",
                    nextEvent[nextNode], nextNode, newX, newY,
speed);
        } else if (output == 'G') {
            fprintf(stdout, "%.12f %.12f # node %d at %.10f
speed=%.10f\n", newX, newY, nextNode, nextEvent[nextNode], speed);
        }

        dist = sqrt((newX-xLoc[nextNode])*(newX-xLoc[nextNode]) +
(newY-yLoc[nextNode])*(newY-yLoc[nextNode]));
        xLoc[nextNode] = newX;
        yLoc[nextNode] = newY;
        nextEvent[nextNode] += dist/speed;
    } /* if (moving[nextNode]) */

    /* find new lowest */
    lowest = endTime + 1.0;
}

```

```

for (i=0; i<numNodes; i++)
{
    if (nextEvent[i] <= lowest)
    {
        lowest = nextEvent[i];
        nextNode = i;
    } /* if (nextEvent[i] <= lowest) */
} /* for (i=0; i<numNodes; i++) */
} /* while (lowest <= endTime) */

if (output == 'N')
{
    fprintf(stdout, "\n\n\n");
} else if (output == 'G') {
    fprintf(stdout, "e\n\n");
}

free(nextEvent);
free(xLoc);
free(yLoc);

return 0;
}

```

Πίνακας B.2.1 Random Waypoint mobility model (mobgen.c)

```

# Common parameters

set val(a_sn)      [expr $val(sn)-$val(dup)]

set val(chan)      Channel/WirelessChannel
set val(prop)      Propagation/TwoRayGround
set val(netif)     Phy/WirelessPhy
set val(mac)       Mac/802_11
set val(ifq)       Queue/DropTail/PriQueue
set val(ll)        LL
set val(ant)       Antenna/OmniAntenna
set val(ifqlen)    50          ;# max packet in ifq

set val(adhocRouting)  BCAST
set val(SLPport)  18
set val(MulticastAddr)  0xE000000
set val(extra)    20.00        ;# wait for that amount after simulation
ends
set val(motionFile)
    "slp_tools/slpMotion/$val(runDirectory)/$val(scenario).motion"
set val(outputFile)
    "slp_tools/slpTraces/$val(runDirectory)/original/$val(scenario).t
r"
#####
# minimize memory requirements by removing all irrelevant packet
headers
remove-packet-header PGM PGM_SPM PGM_NAK Pushback NV LDP MPLS
rtProtoLS Ping
remove-packet-header TFRC TFRC_ACK Diffusion RAP TORA IMEP MIP
remove-packet-header IPinIP Encap HttpInval MFTP SRMEXT SRM aSRM
remove-packet-header mcastCtrl CtrMcast rtProtoDV GAF Snoop TCPA TCP

```

```

IVS
remove-packet-header Resv UMP Src_rt

# selectively allow only one of the following: AODV, SR, BCAST
remove-packet-header AODV
remove-packet-header SR
# remove-packet-header BCAST
#####

# Initialize Global Variables and Procedures

#Set up Random Number Generators

#seed the defaultRNG
$defaultRNG seed 9999

set arrivalRNG [new RNG]
set nnodeRNG [new RNG]
set serviceRNG [new RNG]

for {set j 1} {$j < $val(seed)} {incr j} {
    $arrivalRNG next-substream
    $nnodeRNG next-substream
    $serviceRNG next-substream
}
set arrival_ [new RandomVariable/Exponential]
$arrival_ set avg_ $val(exp_avg)
$arrival_ use-rng $arrivalRNG

set nnode_ [new RandomVariable/Uniform]
$nnode_ set min_ 0
$nnode_ set max_ [expr $val(nn)-1]
$nnode_ use-rng $nnodeRNG

set service_ [new RandomVariable/Uniform]
$service_ set min_ 0
if {$val(dup) == 0} {
    $service_ set max_ [expr $val(a_sn)-1]
} else {
    $service_ set max_ $val(a_sn)
}
$service_ use-rng $serviceRNG

# Initialize array of nodes to zero (no nodes used yet)
for {set i 0} {$i < $val(nn)} {incr i} {
    set array($i) 0
}

# Procedure gets a random node number from the list of nodes
# and set its array position to used, so it won't be polled again
proc get_node {nnode_ name} {
    upvar $name array
    while (1) {
        set i [expr round([$nnode_ value])]
        if {$array($i) == 0} {
            set array($i) 1
            return $i
        }
    }
}

```

```

# Procedure gets an exponential arrival time (for request)
proc get_time {arrival_ stop} {
    set time [$arrival_ value];
    while {$time >= $stop || $time<1000} {
        set time [$arrival_ value];
    }
    return $time;
}

# create simulator instance

set ns_ [new Simulator]

# setup topography object

set topo [new Topography]

# create trace object for ns

set tracefd [open $val(outputFile) w]

# $ns_ use-newtrace
$ns_ trace-all $tracefd

# define topology
$topo load_flatgrid $val(x) $val(y)

#
# Create God
#
set god_ [create-god $val(nn)]

# Global node setting

$ns_ node-config -adhocRouting $val(adhocRouting) \
                -llType $val(ll) \
                -macType $val(mac) \
                -ifqType $val(ifq) \
                -ifqLen $val(ifqlen) \
                -antType $val(ant) \
                -propType $val(prop) \
                -phyType $val(netif) \
                -channelType $val(chan) \
                -topoInstance $topo \
                -agentTrace ON \
                -routerTrace OFF \
                -macTrace OFF

# Create the specified number of nodes [$val(nn)]

for {set i 0} {$i < $val(nn)} {incr i} {
    set node_($i) [$ns_ node]
    $node_($i) random-motion 0      ;# disable random motion
}

# Define node movement model

puts "Loading connection pattern..."
source $val(motionFile)

```

```
#####
# Set up SLP servers (nos: 10-19)

for {set i 0} {$i < $val(sn) } {incr i} {
    set j [get_node $nnode_ array]
    set udp_sa_($i) [new Agent/UDP/UDPSlp]
    $ns_ attach-agent $node_($j) $udp_sa_($i)
    $node_($j) attach $udp_sa_($i) $val(SLPport)
    set slpsa_($i) [new Application/SLPsa]
    $slpsa_($i) attach-agent $udp_sa_($i)
    $ns_ at 0.0100000000 "$node_($j) bcast-join-group
$val(MulticastAddr)"
    $ns_ at 0.0 "$slpsa_($i) start"
}
#####
# Set up SLP clients (nos: 0-9)

for {set i 0} {$i < $val(cn) } {incr i} {
    set j [get_node $nnode_ array]
    #Setup a SLP UDP connection
    set udp_ua_($i) [new Agent/UDP/UDPSlp]
    $ns_ attach-agent $node_($j) $udp_ua_($i)
    $node_($j) attach $udp_ua_($i) $val(SLPport)
    set slpua_($i) [new Application/SLPua]
    $slpua_($i) set maxSearch_ $val(maxSearch)
    $slpua_($i) set dstport_ $val(SLPport)
    $slpua_($i) attach-agent $udp_ua_($i)
    $slpua_($i) add-scope DEFAULT
    $ns_ at 0.0100000000 "$node_($j) bcast-join-group [expr
$val(MulticastAddr)+$j+1]"
    $ns_ at 0.0 "$slpua_($i) start"
}
#####
# Tell nodes when the simulation ends

for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at [expr $val(stop)+$val(extra)] "$node_($i) reset";
}

$ns_ at [expr $val(stop)+$val(extra)+0.002] "puts \"NS
EXITING...$val(scenario)\" ; $ns_ halt"

# Output simulation parameters first to trace file

puts $tracefd "# Scenario: $val(scenario)"
puts $tracefd "# Nodes: $val(nn)"
puts $tracefd "# Server Agents: $val(sn)"
puts $tracefd "# Duplicate Servers: $val(dup)"
puts $tracefd "# Client Agents: $val(cn)"
puts $tracefd "# Total Requests: $val(reqs)"
puts $tracefd "# Simultaneous Requests: $val(simult)"
puts $tracefd "# Max Search: $val(maxSearch)"
puts $tracefd "# Service lifetime: $val(lifetime) s"
puts $tracefd "# Interarrival exponential avg: $val(exp_avg) s"
puts $tracefd "# Area: $val(x)m X $val(y)m"
puts $tracefd "# Routing: $val(adhocRouting)"
puts $tracefd "# Simulation Time: $val(stop) s (first 1000.0 s
discarded)"

#####
```

```
# Add service to server agents
for {set i 0} {$i < $val(a_sn)} {incr i} {
    $slpsa_($i) add-service service:ServiceNum$i://134.117.61.$i
$val(stop) $val(lifetime)
}
for {set i $val(a_sn)} {$i < $val(sn)} {incr i} {
    $slpsa_($i) add-service
service:ServiceNum$val(a_sn)://134.117.61.$val(a_sn) $val(stop)
$val(lifetime)
}
#####

# Clients send requests
set j 0
set simultaneous [get_time $arrival_ $val(stop)]
for {set i 0} {$i < $val(reqs)} {incr i} {
    set sno [expr round([$service_ value])]
    if {$j<$val(simult)} {
        $ns_ at $simultaneous "$slpua_($j) rqst-service
service:ServiceNum$sno $val(MulticastAddr)"
    } elseif {$j==$val(simult)} {
        set simultaneous [get_time $arrival_ $val(stop)]
        $ns_ at [get_time $arrival_ $val(stop)] "$slpua_($j) rqst-service
service:ServiceNum$sno $val(MulticastAddr)"
    } else {
        $ns_ at [get_time $arrival_ $val(stop)] "$slpua_($j) rqst-service
service:ServiceNum$sno $val(MulticastAddr)"
    }
    incr j
    if {$j==$val(cn)} {
        set j 0
    }
}

#####
# Start the Simulation
puts "Starting Simulation..."
$ns_ run
```

B.2.2 original_common.tcl

```
#####
# Scenario: Collaborative Conference
#####

# SLP scenario-particular parameters

set val(scenario) conf

set val(x)          20      ;# X dimension of the topography
set val(y)          10      ;# Y dimension of the topography
set val(nn)         10      ;# how many nodes are simulated
set val(sn)         4       ;# how many of the nodes are servers
set val(cn)         5       ;# how many of the nodes are clients
set val(stop)       2000.0   ;# simulation time + 10
seconds
set val(exp_avg)    1360.0    ;# Exponential Distribution average for
iia-times
set val(maxSearch)  "false"   ;# Allow maximum search
"true/false"
```

```

set val(lifetime) 60      ;# Lifetime of a service advertisement
set val(reqs)      1000    ;# Number of service requests
set val(simult)    2       ;# Number of clients requesting
simultan.
set val(dup)       3       ;# Number of duplicated servers
#####

if {($argc < 2) || ($argc > 3)} {
    error "usage: scenario runSubdirectory seed <-cache>"
}
set val(runDirectory) [lindex $argv 0]
set val(seed) [lindex $argv 1]
if {($argc==3) && ([lindex $argv 2] == "-cache")} {
    source slp_tools/cache_common.tcl
} else {
    source slp_tools/original_common.tcl
}

```

B.2.3 conf.tcl

```

#####
# Scenario: Vehicle-Network
#####

# SLP scenario-particular parameters

set val(scenario) vr

set val(x)          1200      ;# X dimension of the topography
set val(y)          1200      ;# Y dimension of the topography
set val(nn)         70        ;# how many nodes are simulated
set val(sn)         25        ;# how many of the nodes are servers
set val(cn)         30        ;# how many of the nodes are clients
set val(stop)       2000.0    ;# simulation time + 10
seconds
set val(exp_avg)    1360.0     ;# Exponential Distribution average for
iaa-times
set val(maxSearch)  "false"    ;# Allow maximum search
"true/false"
set val(lifetime)  10         ;# Lifetime of a service advertisement
set val(reqs)      1000       ;# Number of service requests
set val(simult)    10         ;# Number of clients requesting
simultan.
set val(dup)       20         ;# Number of duplicated servers

#####

if {($argc < 2) || ($argc > 3)} {
    error "usage: scenario runSubdirectory seed <-cache>"
}
set val(runDirectory) [lindex $argv 0]
set val(seed) [lindex $argv 1]
if {($argc==3) && ([lindex $argv 2] == "-cache")} {
    source slp_tools/cache_common.tcl
} else {
    source slp_tools/original_common.tcl
}

```

B.2.4 vehnet.tcl

```

#*****
# Scenario: Mesh Network
#*****

# SLP scenario-particular parameters

set val(scenario) mesh

set val(x)          3000      ;# X dimension of the topography
set val(y)          3000      ;# Y dimension of the topography
set val(nn)         300       ;# how many nodes are simulated
set val(sn)         100       ;# how many of the nodes are servers
set val(cn)         100       ;# how many of the nodes are clients
set val(stop)       2000.0    ;# simulation time + 10 seconds
set val(exp_avg)    1360.0    ;# Exponential Distribution average for
iaa-times
set val(maxSearch)  "false"    ;# Allow maximum search
"true/false"
set val(lifetime)   100.0      ;# Lifetime of a service advertisement
set val(reqs)       1000      ;# Number of service requests
set val(simult)     10         ;# Number of clients requesting
simultan.
set val(dup)        100       ;# Number of duplicated servers

#####

if {($argc < 2) || ($argc > 3)} {
    error "usage: scenario runSubdirectory seed <-cache>"
}
set val(runDirectory) [lindex $argv 0]
set val(seed) [lindex $argv 1]
if {($argc==3) && ([lindex $argv 2] == "-cache")} {
    source slp_tools/cache_common.tcl
} else {
    source slp_tools/original_common.tcl
}

```

B.2.5 mesht.tcl