Nancy-Université
Université Henri Poincaré

AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

➢ Contact SCD Nancy 1 : theses.sciences@scd.uhp-nancy.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

Nancy-Université
Université Henri Poincaré

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Contributions for Advanced Service Discovery in Ad hoc Networks

THÈSE

présentée et soutenue publiquement le 24 novembre 2011

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1
(spécialité informatique)

par

Tom Leclerc

Composition du jury

Rapporteurs:	Patrick SÉNAC	Professeur ISAE
	Marcelo DIAS de AMORIM	Chargé de recherche CNRS au LIP6
Examinateurs:	André SCHAFF	Professeur à l'ESIAL
	Jens GUSTEDT	Directeur de recherche INRIA
	Laurent CIARLETTA	MCF Ecole des Mines de Nancy
	Priyadarsi NANDA	Professeur Assistant à l'UTS
	Steffen ROTHKUGEL	Professeur Assistant à l'Université de Luxembourg
	Françoise SAILHAN	MCF au CNAM

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Loria
Laboratoire lorrain de recherche en informatique et ses applications

Mis en page avec la classe thloria.

Dedicated to my cousin Gaëtan and my grandfathers Georges and Leo,
who left us too early

Acknowledgments

First, I would like to thank the members of my jury for the time they spent to improve and comment my thesis, especially my reading committee whose advices were greatly appreciated.

I am very thankful for my two thesis advisors André Schaff and Laurent Ciarletta who gave me the opportunity to prepare and present this work. A very special thank you to Laurent, for his devotion and help throughout my thesis and with whom I had many interesting conversations about our research. He gave me the motivation, the guidelines and advices to successfully complete my work. Moreover, over time, Laurent has become a true friend with whom I also had many laughs and fun.

I would like to thank Olivier Festor for the opportunity to do this thesis in a great work and living environment that is the LORIA, INRIA Grand Est Laboratory.

Huge thanks to all the former and current members of the Madynes team who became my friends and also all the friends I found in Nancy. Special thanks to all on my favorite IRC channel #linux for the fun and also the help they provided.

This thesis was also the opportunity of many research collaborations. I would like to friendly thank Adrian Andronache, Steffen Rothkugel and Patrick Gratz from the University of Luxembourg, Julien Siebert and Vincent Chevrier from the Maia team at the LORIA, Laurent Reynaud and the members of the ANR SARAH project.

Last but not least, I would like to thank my family for always supporting me, especially my dad Patrick and my mom Martina. This won't be completed without thanking my brother Jan who spent many hours reading, correcting and improving my thesis.

Contents

Introduction **1**

Part I Foundations **7**

Chapter 1
Introduction to mobile ad hoc networks

1.1 Mobile ad hoc networks 12
1.2 Dissemination, Routing and Topology Control 14
1.2.1 Introduction to routing protocols 15
1.2.2 Routing protocols in ad hoc networks 17
1.2.3 Dissemination strategies 21
1.2.4 Clustering 21
1.2.5 Dominating Sets 24
1.3 Conclusion 26

Chapter 2
Introduction to service discovery and context in ad hoc networks

2.1 Naming Service 28
2.2 Introduction to service discovery 28
2.3 Service discovery in wired networks 30
2.3.1 SLP 30
2.3.2 UPnP 30
2.3.3 Zeroconf 30
2.3.4 Jini 33
2.4 Service discovery in ad hoc networks 34
2.4.1 Konark 34
2.4.2 Allia 34
2.4.3 Service discovery over OLSR 35
2.4.4 Service discovery using a field theoretic approach 35

2.5 Zeroconf as service discovery protocol for ad hoc networks . . . 36
2.5.1 Zeroconf and its multicast structure . . . 36
2.5.2 Why does Zeroconf use multicast? . . . 37
2.5.3 Conclusion . . . 38
2.6 Context & Metrics . . . 38
2.6.1 Social: Centrality . . . 38
2.6.2 Collaborative Filtering . . . 40
2.7 Conclusion . . . 42

Part II Contributions **43**

Chapter 3
Dissemination: SLSF - Stable Linked Structure Flooding

3.1 Big picture: NLWCA - WCPD - SLSF-R - SLSF . . . 48
3.2 Structure comparison . . . 50
3.2.1 WCPD vs. OLSR . . . 50
3.2.2 Experiments and results . . . 50
3.2.3 Conclusion . . . 52
3.3 Avoid Subheads in NLWCA . . . 53
3.3.1 Problems . . . 54
3.3.2 Solution . . . 55
3.4 Basic SLSF . . . 58
3.4.1 IRC – Inter-Cluster Relays . . . 58
3.4.2 SLSF - Broadcast . . . 62
3.4.3 Basic SLSF message Header format . . . 62
3.4.4 IP fragmentation and SLSF . . . 62
3.5 Full SLSF with Fault-recovery . . . 64
3.5.1 Acknowledgment mechanism . . . 64
3.5.2 Delayed transmission mechanism . . . 66
3.5.3 Fault Recovery Header format . . . 67
3.6 Experiments and Results . . . 68
3.7 Conclusion . . . 69

Chapter 4
Routing: SLSR - Stable Linked Structure Routing

4.1 Message types and topology gathering . . . 75
4.2 A balanced-hybrid protocol to reduce the overhead . . . 76
4.3 Routing Table and update management . . . 78

4.3.1 Choosing a route . . . 78
4.3.2 Table updates from Current Slaves messages . . . 78
4.3.3 Table updates from lost Slaves messages . . . 79
4.3.4 Table updates from Lost Route to Clusters messages . . . 80
4.4 Symbiotic behavior . . . 81
4.4.1 Higher-layer traffic . . . 81
4.4.2 Message timings and header combinations . . . 82
4.5 Message header formats . . . 83
4.5.1 Routed Message . . . 83
4.5.2 Current Slaves . . . 83
4.5.3 Lost Slaves . . . 85
4.5.4 Lost Route to Clusters . . . 85
4.6 Advanced Configuration and option messages . . . 85
4.6.1 Fault recovery activation and deactivation . . . 85
4.6.2 Extended forwarding of routing information . . . 85
4.6.3 Cross-layer design . . . 87
4.6.4 Full link-state behavior . . . 88
4.7 Conclusion . . . 89

Chapter 5
Advanced Mobility Models using multi-modeling and co-simulation

5.1 JANE and other ad hoc network simulators . . . 92
5.2 Ad hoc mobility models . . . 93
5.3 Advanced mobility models . . . 96
5.4 Case study . . . 97
5.5 Multi-modeling and co-simulation motivation . . . 97
5.6 The AA4MM meta-model and platform . . . 98
5.7 Case study: MANETs and users' behaviors . . . 100
5.7.1 Mobility modeling . . . 100
5.7.2 The multiagent paradigm . . . 100
5.7.3 User model description . . . 100
5.7.4 Modeling network aware users . . . 101
5.7.5 Synthesis . . . 101
5.8 Experiments and Results . . . 102
5.8.1 Building realistic usage scenarios . . . 102
5.8.2 Network and user behavior mutual influences . . . 103
5.8.3 Synthesis . . . 104
5.9 Conclusion . . . 105

Chapter 6
Collaborative Filtering

6.1 A Delay-Tolerant CF . . . 108
6.1.1 User Similarity . . . 108
6.1.2 Clustering Algorithm . . . 108
6.1.3 Delay-Tolerant Intra-Cluster Broadcast . . . 108
6.2 Experiments and Results . . . 111
6.2.1 Dataset . . . 111
6.2.2 Evaluation Criteria . . . 111
6.2.3 Simulation Settings . . . 111
6.2.4 Results . . . 112
6.3 Conclusion . . . 113

Chapter 7
Service Discovery in ad hoc networks using Zeroconf and SLSR

7.1 Zeroconf on top SLSF . . . 116
7.1.1 Adaptability to ad hoc networks . . . 116
7.2 Context-awareness in Service Discovery . . . 117
7.2.1 Network . . . 118
7.2.2 Node . . . 120
7.2.3 Space . . . 121
7.2.4 Service . . . 122
7.3 Collaborative filtering for service discovery . . . 122
7.4 Service Discovery Architecture . . . 123
7.4.1 Context: ANR SARAH project . . . 123
7.4.2 Scenario . . . 123
7.4.3 Service discovery architecture description . . . 124
7.4.4 Modeling our scenario using multi-agent and co-simulation . . . 125
7.5 Experimentations on real devices . . . 128
7.5.1 OLSR experiments . . . 128
7.5.2 Zeroconf on top of SLSF . . . 129
7.5.3 Zeroconf on top of SLSF and SLSR . . . 129
7.6 Conclusion . . . 131

Conclusion and perspectives **133**

Conclusion

Perspectives

Appendix A
Service discovery protocols

A.1 UPnP – Universal Plug and Play Networking . . . 139
A.1.1 Step 0: IP Address . . . 139
A.1.2 Step 1: Discovery (SSDP) . . . 139
A.1.3 Step 2: Description . . . 140
A.1.4 Step 3: Control . . . 140
A.1.5 Step 4: Eventing . . . 141
A.1.6 Step 5: Presentation . . . 141
A.2 Jini . . . 141
A.2.1 Service interactions with Lookup Service . . . 142
A.2.2 Client interactions with Lookup Service and Jini Service . . . 142

Appendix B
UPnP - XML service and device description

Appendix C
Research tools

C.1 Network and graph visualization . . . 149
C.1.1 Graphviz . . . 149
C.1.2 JUNG . . . 150
C.2 Implemented simulation tools . . . 150
C.2.1 Click-and-play mobility tools . . . 150
C.2.2 Mobility scenarios . . . 152
C.2.3 Visualization pane with Jung . . . 152

Appendix D
Implementations

D.1 OLSR . . . 155
D.1.1 OLSR service . . . 156
D.1.2 Message structures . . . 158
D.1.3 Repositories and data structures . . . 159
D.1.4 Simulation and timer utilities . . . 160
D.2 NLWCA - Subhead avoidance . . . 161
D.3 SLSF and SLSR . . . 161
D.3.1 Service events . . . 161

D.3.2 Beaconing 161
D.3.3 Fault-recovery mechanism 163
D.3.4 ICR computation 165
D.3.5 Message creation and forwarding 165
D.3.6 Routing - SLSR 168
D.4 Zeroconf - adaptations 168

Appendix E
Résumé Étendu

Appendix F
Introduction aux réseaux mobiles ad hoc et à la découverte de service

F.1 Dissémination, routage et contrôle topologique 177
F.1.1 Protocoles de routage ad hoc 178
F.1.2 Stratégies de dissémination 179
F.1.3 Sélection des nœuds relais 180
F.1.4 Hiérarchie et clustering 181
F.2 Service de nommage 183
F.3 La découverte de service 183
F.3.1 Classification des protocoles de découverte de service 184
F.3.2 Découverte de service dans les réseaux filaires 185
F.3.3 Découverte de services dans les réseaux ad hoc 186
F.4 Métriques et contexte 188

Appendix G
Dissémination: SLSF - Stable Linked Structure Flooding

G.1 Vue d'ensemble: NLWCA - WCPD - SLSF-R - SLSF 189
G.2 SLSF - Stable Linked Structure Flooding 191
G.2.1 ICR – Inter-Cluster Relays 191
G.2.2 Diffusion dans SLSF 193
G.3 SLSF avec recouvrement d'erreur 193
G.4 Résultats 194
G.5 Conclusion 196

Appendix H
Routage: SLSR - Stable Linked Structure Routing

H.1 SLSF en tant que structure de base 199
H.2 Routage des messages 200
H.3 Type de messages 201

H.4 Un protocole dit "balanced-hybrid" . . . 202
H.5 Table de routage et mise à jour . . . 202
H.5.1 Choisir une route . . . 202
H.5.2 Mise à jour de la table de routage . . . 203
H.6 Comportement symbiotique . . . 205
H.6.1 Higher-layer traffic . . . 205
H.7 Conclusion . . . 205

Appendix I
Découverte de services dans les réseaux ad hoc utilisant Zeroconf et SLSR

I.1 Pourquoi Zeroconf? . . . 208
I.2 Zeroconf sur SLSF . . . 208
I.3 *Context-awareness* pour la découverte de services . . . 210
I.3.1 Réseau . . . 210
I.3.2 Nœud/Appareil . . . 213
I.3.3 Espace . . . 213
I.3.4 Service . . . 214
I.4 Filtrage collaboratif pour la découverte de services . . . 214
I.5 Architecture de découverte de services . . . 215
I.5.1 Scénario . . . 215
I.5.2 Modélisation du scénario en utilisant les systèmes multi-agent et la co-simulation . . . 215
I.6 Cas d'étude . . . 216
I.7 Expérimentations . . . 218
I.7.1 OLSR . . . 218
I.7.2 Zeroconf au dessus de SLSR . . . 219
I.7.3 Zeroconf, SLSF et SLSR . . . 219
I.8 Conclusion . . . 220

Appendix J
Conclusion

Appendix K
Perspectives

Bibliography **229**

List of Figures

1 Representation of the contributions of this thesis. 4

1.1 Dynamic networks: Ad hoc, mesh, sensor and peer-to-peer networks. 12
1.2 Ad hoc network composed of wirelessly connected devices. 13
1.3 Unicast. 14
1.4 Broadcast. 14
1.5 Multicast. 14
1.6 a) ad hoc network. b) only active links. c) subset of links for dissemination. . . . 15
1.7 DSDV routing operations. 17
1.8 AODV route discovery operations. 18
1.9 MPR selection (Bordercasting) in OLSR. 19
1.10 Zone Routing protocol - Route discovery. 20
1.11 Example of two clusters built by NLWCA. 22
1.12 The low weight of the links avoids superfluous re-organization of the topology when for instance two clusters cross in mobile networks. 23
1.13 NLWCA beacon containing weight and designated cluster address. 23
1.14 NLWCA and WCPD Beacon. 23
1.15 WCPD message dissemination through a network. 24
1.16 Dominating sets (red nodes). 24

2.1 Service discovery protocol classification. [Ciar 02] 29
2.2 SRV record example. 32
2.3 PTR record example. 33
2.4 TXT record example. 33
2.5 Service discovery with potentials - field approach. 35
2.6 Star network. 39
2.7 Circle network. 39
2.8 Line network. 39
2.9 Representation of the contributions of this thesis. 45

3.1 Protocol hierarchy of NLWCA - WCPD - SLSF-R and SLSF: Dissemination perspective. 49
3.2 Protocol hierarchy of NLWCA - WCPD - SLSF-R and SLSF: Beacon format perspective. 50
3.3 OLSR message dissemination through a network. 51
3.4 WCPD message dissemination through a network. 51
3.5 JANE simulating the protocols on 100 devices. The mobile devices move on the streets of the Luxembourg City map. The devices move with a speed of 0.5 - 1.5 m/s. 51

3.6 Bandwidth used in order to build the topology for 100, 200 and 300 nodes. 52
3.7 Overall number of sent/forwarded messages and received for 100, 200 and 300 nodes. (smoothed with a polynomial equation of the 16th grade for visibility sake). 53
3.8 NLWCA - Node with weight 25 (Clusterhead) is designated by node with weight 15 (Subhead) which itself is designated by node with weight 14 (Slave). 54
3.9 NLWCA cluster with a sub-head chain. 54
3.10 NLWCA cluster without a sub-head. All nodes in the same cluster claim the same clusterhead. 55
3.11 NLWCA cluster with a sub-head. Nodes 9 and 14 do not claim the same clusterhead while they are in the same cluster. 55
3.12 NLWCA cluster without sub-heads. 56
3.13 Inter-cluster configuration examples where 1 and 2 are clusterheads. 57
3.14 Same topology a) with subheads and b) without subheads. Node D remains isolated in a one-node cluster in b). 57
3.15 NLWCA cluster with a subhead chain. Adding a higher weighted node in the chain. 58
3.16 NLWCA cluster without subheads. Adding a higher weighted node in the avoided-chain. 58
3.17 Inter-cluster configuration examples where 1 and 2 are clusterheads. 59
3.18 a) Cluster configuration with more than 3 hops between two clusters. b) view from node 2. c) view of node 1. d) view of node 3. 60
3.19 a) Long sequence of slaves with SLSF. b) View from each cluster of its actual nearby-clusters. 61
3.20 ICR selection with 5 clusters in SLSF. 61
3.21 MPR selection in OLSR. 61
3.22 Header of a basic broadcast message (Graphical representation). 63
3.23 Header of a basic broadcast message (Textual representation). 63
3.24 Foreign-cluster broadcast - Format of a message sent from node 1. 64
3.25 Foreign-cluster broadcast - Path of a message sent from node 1. 64
3.26 CH8 Beacon: (a)SLSF. (b)SLSF with fault recovery. 65
3.27 Node 3 Beacon: (a)SLSF. (b)SLSF with fault recovery. 65
3.28 Three clusters example. 65
3.29 Node 3 SLSF beacon. 65
3.30 Node 3 SLSF beacon with fault recovery 65
3.31 Header for Fault-recoverable broadcast message (Graphical representation). . . . 67
3.32 Header for Fault-recoverable broadcast message (Textual representation). 67
3.33 Bandwidth used in order to build the topology for 100, 200 and 300 nodes. . . . 68
3.34 Overall number of sent/forwarded messages and received for 100, 200 and 300 nodes. (smoothed with a polynomial equation of the 16th grade for visibility sake). 70
3.35 Efficiency of Bandwidth usage for 100, 200 and 300 nodes. 71
3.36 Static scenario with 100 to 300 node.s 71

4.1 Same network from two different point of views: SLSF (left) and SLSR (right). . 74
4.2 SLSR - three clusters example. SLSF view (left) and SLSR overlay view (right). . 75
4.3 Comparison of routing overhead by displaying the scope of an announcement in each routing advertisement. 77
4.4 SLSR - three clusters with a loop. SLSF view (left) and SLSR overlay view (right). 79
4.5 Example of SLSF and SLSR header combinations. 83
4.6 Header for a Routed Message (Graphical representation). 84
4.7 Header for a Routed Message (Textual representation). 84

4.8 Format of the header for a Current Slaves message (Graphical representation). . . 84
4.9 Format of the header for a Current Slaves message (Textual representation). . . . 84
4.10 Format of the header for a Lost Slaves message (Graphical representation). . . . 86
4.11 Format of the header for a Lost Slaves message (Textual representation). 86
4.12 Format of the header for lost route to nearby clusters message (Graphical representation). 86
4.13 Format of the header for lost route to nearby clusters message (Textual representation). 86
4.14 Format of the header for the fault-recovery optional message. 87
4.15 Forwarding and duplicate handling options of routing information in SLSR: left classic forwarding and right extended forwarding. 87
4.16 Format of the header for extended forwarding of routing information in SLSR. . . 88
4.17 Cross-layer classification based on the layer interactions (Figure from [Sriv 05]). . 88

5.1 Service interactions in JANE. 94
5.2 The JANE operating system. 94
5.3 Example of node movement of one group in the Reference Point Group Mobility Model from time T_0 to T_1. 95
5.4 Abstraction levels. 98
5.5 Framework model interactions. 98
5.6 Perceptions of an agent (a user) 101
5.7 Repulsive force for obstacle avoidance model. 101
5.8 Attractive force to the goal. 101
5.9 Movement computation. 101
5.10 Museum visit example in multiple environments. 102
5.11 4 source nodes: green nodes, 100 moving nodes: black when disconnected, orange when connected to a source. 103
5.12 Interactions between MASDYNE and JANE. 103
5.13 Percentage of nodes stopping when connected, remaining nodes move randomly. 104
5.14 Percentage of nodes that divide their speed by 6 when connected, remaining nodes move randomly. 104
5.15 Percentage of nodes stopping when connected, remaining nodes divide their speed by 6 when connected. 105
5.16 Percentage of nodes stopping when connected, remaining nodes divide their speed by 3 when connected. 105

6.1 Consistency table at the clusterhead. 110
6.2 Broadcasting the least common update. 110
6.3 Prediction accuracy. 112
6.4 Prediction coverage in %. 112
6.5 Bandwidth usage. 112

7.1 Zeroconf stack with Multicast IP and SLSF. 116
7.2 TXT record example containing contextual information. 118
7.3 Service discovery architecture. 124
7.4 Service discovery architecture. 126
7.5 Museum visit example in multiple environments. 126
7.6 Group gathering scenario. Evolution (from a to f) from randomly placed nodes heading towards their goal area. 127
7.8 The JANE operating system. 128

7.7 Nokia N800 used for the experimentations. . . . 128
7.9 Zeroconf on top of SLSR experiment. . . . 129
7.10 DNS Table view. . . . 130
7.11 DNS cache view. . . . 130
7.12 Zeroconf available services. . . . 130
7.13 Chat/interaction window. . . . 130
7.14 DNS Table view. . . . 132
7.15 Zeroconf available services. . . . 132
7.16 DNS cache view. . . . 132

A.1 UPnP protocol stack. . . . 139
A.2 format of a SSDP service notification message. Services are identified by their Unique Service Name (USN). . . . 140
A.3 format of a SSDP search message. ST is specifies the service type to discover. . . . 140
A.4 UPnP service description retrieval. . . . 141

B.1 UPnP device description format. . . . 146
B.2 UPnP service description format. . . . 147

C.1 Example of a DOT-language file. . . . 149
C.2 Example of a graph generated by graphviz. . . . 149
C.3 Minimum spanning tree demo from [jung]. Demonstrates JUNG's ability to show multiple views of the same graph and to extract the Minimum Spanning Trees. . . . 151
C.4 JANE (left) with Simbad (right) simulation. . . . 153
C.5 JANE (top left), Local view interface (bottom left) and Jung visualization pane (right). . . . 153

D.1 OLSR implementation in JANE. . . . 156
D.2 OLSR in the JANE operating system. . . . 157
D.3 SLSF implementation in JANE. . . . 163
D.4 Three clusters example. . . . 164
D.5 CH 5 SLSF beacon with fault recovery. . . . 164
D.6 Node 3 SLSF beacon with fault recovery . . . 164

F.1 Réseaux dynamiques: Ad hoc, mesh, capteur et pair-à-pair. . . . 178
F.2 Séléction des MPR (Bordercasting) avec OLSR. . . . 179
F.3 a) réseau ad hoc. b) lien actif uniquement. c) sous-ensemble de nœuds pour la dissémination. . . . 180
F.4 Ensemble dominant (Dominating set) (nœud en rouge). . . . 180
F.5 Exemple de deux clusters dans NLWCA. . . . 182
F.6 Deux clusters NLCWA qui se croisent. Un poids faible sur un lien indique que celui-ci n'est pas stable et n'est ainsi pas pris en compte pour la sélection du clusterhead. . . . 182
F.7 Format du beacon de NLWCA. . . . 183
F.8 Format du beacon WCPD combiné avec NLWCA. . . . 183
F.9 Dissémination d'un message avec WCPD. . . . 183
F.10 Classification des protocoles de découverte de service. [Ciar 02] . . . 185
F.11 Découverte de services utilisant l'approche par champs électrostatiques. . . . 187

G.1 Hiérarchie des protocoles NLWCA - WCPD - SLSF-R et SLSF: Dissémination. . . . 190
G.2 Hiérarchie des protocoles NLWCA - WCPD - SLSF-R et SLSF: Format des messages. 191

G.3 Exemple de configuration inter-cluster avec 1 et 2 clusterheads. 192
G.4 a) Configuration de cluster avec plus de 3 sauts entre deux clusters. b) Point de vue du nœud 2. c) Point de vue du nœud 1. d) Point de vue du nœud 3. 193
G.5 a) Longue séquence de nœuds esclaves dans SLSF. b) Vue de chacun des clusters sur son voisinage proche. 193
G.6 Sélection ICR dans un réseau avec 5 clusters. 194
G.7 JANE simulating the protocols on 100 devices. The mobile devices move on the streets of the Luxembourg City map. The devices move with a speed of 0.5 - 1.5 m/s. 195
G.8 Bande passante nécessaire à la mise en place de la topologie pour 100, 200 et 300 nœuds. 196
G.9 Nombre total de messages relayés et nombre de messages effectivement reçus pour 100, 200 et 300 nœuds. (lissé, pour une plus grande visibilité, avec une équation polynomial du16ième degré). 197
G.10 Efficacité de l'utilisation de la bande passante pour 100, 200 et 300 nœuds. 198
G.11 Scénario statique avec 100 et 300 nœuds. 198

H.1 Le même réseau avec les deux vues différentes : SLSF (gauche) et SLSR (droite). 200
H.2 SLSR - réseau avec trois clusterheads. Vue SLSF (gauche) et vue SLSR "simplifié" (droite). 201
H.3 Table de routage simplifiée pour le nœud esclave 19. 201
H.4 Table de routage simplifiée pour le clusterhead 20. 201
H.5 Table de routage simplifiée pour le clusterhead 15. 201
H.6 Comparaison du coût de routage en comparant le recouvrement des informations et la cible des annonces. 203
H.7 SLSR - trois clusters avec une boucle. Vue de SLSF (gauche) et vue "simplifié" de SLSR (droite). 204
H.8 Table de routage du clusterhead 20 204

I.1 Pile de protocoles de Zeroconf avec le multicast IP et avec SLSF. 209
I.2 Entrée TXT contentant des informations de contexte. 210
I.3 Intermédiarité 212
I.4 Architecture de découverte de services. 216
I.5 Architecture de découverte de services avec réseau mesh. 217
I.6 Interactions entre le simulateur réseau et le simulateur de mobilité. 217
I.7 Scénario de visite de musée dans différents environnements. 218
I.8 Scénario de mouvement de groupes. Evolution (de a à f) de nœuds placés aléatoirement bougeant vers leur zone cible. 221
I.9 Expérience avec Zeroconf au-dessus de SLSR. 222
I.10 Table DNS. 222
I.11 Cache DNS. 222
I.12 Services disponibles dans Zeroconf. 222
I.13 Fenêtre de chat et d'interaction. 222
I.14 Table DNS. 223
I.15 Services disponibles dans Zeroconf. 223
I.16 Cache DNS. 223

Introduction

In the last decade, the number of wireless capable devices increased drastically along with their popularity. Devices also became more powerful and affordable, attracting more users to mobile networks. Mobile networks encompass several types of networks that can be classified into three main categories: wireless networks with a managed core or infrastructure, like mesh or cellular networks, networks with a specific type of devices like sensor networks, and networks without any preexisting infrastructure and heterogeneous devices like Ad hoc networks. In this thesis we consider the latter case, Mobile Ad hoc NETworks, also called MANETs. MANETs, are a collection of devices, also called nodes, that communicate with each other over a wireless medium. Such networks are formed spontaneously whenever devices are in transmission range of each other without any preexisting infrastructure. The main characteristics of MANETs are the high dynamicity of nodes (induced by the users moving around), the volatile wireless transmissions, the user's behavior, the services and their usage. The spontaneity and dynamicity of ad hoc networks makes the use of service discovery a major advantage and challenge at the same time. Service discovery allows to find services provided by other nodes in the network in an automated way without the use of a mandatory single central control point, which is exactly what ad hoc networks are all about.

This thesis proposes a thorough solution for service discovery in ad hoc networks, from the underlying network layer up to the service discovery itself. The process of service discovery involves various underlying layers and related research domains (dissemination, routing, simulation models and tools, context, etc.). Therefore, while the initial goal of the contributions was to improve service discovery, it also led to improvement on other MANET related domains.

Following are the requirements we considered to improve service discovery. In a simplified way, service discovery is all about sending service announces or queries to all service-interested participants of the network and thereafter obtaining, by collecting the announces or receiving matching responses, the list of available services. Thus, a first requirement is an **efficient message dissemination**. In bigger networks, the need to aggregate announces or queries emerges. Therefore, for scalability reasons, service discovery protocols employ directory nodes that collect and aggregate service information for a given subset of nodes. A second requirement is the use of **reference nodes**. In a dynamic environment such as ad hoc networks, those reference nodes should also be reliable, well positioned and stay for a reasonable amount of time in the network. Hence, a third requirement for our framework is the notion of **stability** of the reference nodes or better of all nodes involved. Once the service is discovered, to be able to communicate with the node hosting the desired service or even more generally in any network, nodes need to be able to send messages to a particular node. As a fourth requirement, unicast, thereby, **routing** should be available. Moreover, because of the overall dynamicity in terms of nodes, topologies, but also usage scenarios that can occur, the framework should adapt its efforts to the current context. The fifth requirement is **context awareness** and adaptability. To be able to simulate the various usage scenarios, we need mobility models that are able to model the users and their human behavior. The sixth requirement is new **mobility models** that can model human-like behaviors. More generally to assist research in the prototyping, visualization, development and assessment effort, the seventh requirement is adapted **research tools**. Finally, to validate the overall process, real world **experiments** need to be conducted.

A first objective of this work is to build a stable structure on top of spontaneously created ad hoc networks. A major challenge is to reach a satisfactory stability by keeping the bandwidth low to create this stable structure. Our first contribution therefore, is the Stable Linked Structure Flooding (SLSF) protocol that relies on a stable based one-hop cluster structure (NLWCA) and thereby provides scalable and efficient message dissemination. The second contribution is the Stable Linked Structure Routing (SLSR) protocol that uses the SLSF dissemination structure to enable routing capabilities. Using those protocols as basis, we propose to improve service discovery by additionally considering context awareness and adaptation. Moreover, we also contributed on improving simulations by coupling simulators and models that, together, can model and simulate the variety and richness of ad hoc related usage scenarios and their human characteristics.

A guideline for all of our contributions was to be able to integrate and/or consider context and context awareness in both the proposed protocols and the related research tools and models. On one hand, the proposed protocols all have the capacity to adapt their efforts according to certain metrics, that represent the context. On the other hand, the simulator coupling architecture, permits to model and design scenarios in which the context, such as the service usages or the human behavior, has an impact and matters.

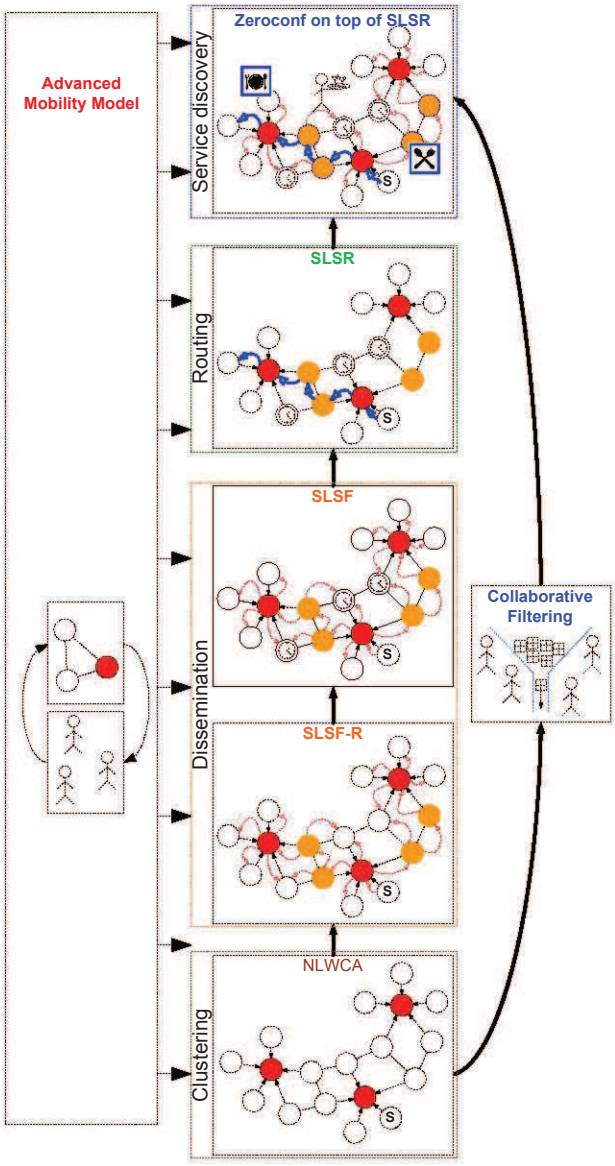


Figure 1: Representation of the contributions of this thesis.

Organization of the thesis

The first part of this dissertation establishes the fundamentals notions and domains involved throughout this thesis.

Chapter 1 introduces MANETs several dissemination and routing protocols of interest. In the following chapter 2 presents service discovery in ad hoc networks as well as the notions of context and metrics.

The second part of this thesis contains the contributions on dissemination, routing, mobility models, collaborative filtering and the final common denominator, Service discovery.

In Chapter 3 the first stone of our framework is laid with SLSF. The Stable Linked Structure Flooding protocol creates a structure based on stable clusters that provides scalable and efficient dissemination by selecting inter-cluster nodes wisely using the, so called, ICR (Inter Cluster Relay) mechanism. Moreover, SLSF also includes a fault-recovery mechanism that locally, between

two clusters, detects and repairs faulty transmissions.

Chapter 4 introduces a new routing protocol that relies on SLSF. Indeed, SLSR (Stable Linked Structure Routing) uses the dissemination structure provided by SLSF to disseminate its routing information. SLSR is a cross-layer routing protocol. SLSR adapts its efforts by being aware of the underlying SLSF structure as well as taking advantage of the overlying protocols or applications exchanges.

Chapter 5 introduces simulation of ad hoc networks, and their mobility models, and proposes a methodology and a distributed framework to design, implement and assess MANETs protocols and applications. The proposed approach enables the combined use of reference models and simulators coming from the network and sociology domains to provide a more realistic simulation. Combination is done through a simple interface implemented for each simulator (network and mobility). The presented framework eases the interactions among both mobility models and simulators.

Chapter 6 considers the overload of information that can occur with the growth of ad hoc networks and also with the multiplicity of contents (e.g. services) exchanged on such networks. Collaborative filtering deals with the issue of content overload and filtering, however existing approaches mostly require a huge centralized database. In this chapter we propose an incremental recommender system for highly dynamic mobile environments where no central global knowledge is available.

Chapter 7 shows how the previous contributions converge together to improve service discovery and enable context-awareness. As a service discovery protocol we use Zeroconf. This chapter proposes to replace the multicast in Zeroconf by the SLSF/SLSR structure. Furthermore, metrics to capture the context for service discovery are proposed. The advantage of using collaborative filtering for service related information is depicted as well as how SLSR profits and is aware of the Zeroconf messages. The chapter also presents a complete service discovery architecture proposed in the context of the ANR SARAH project where our simulation framework providing more realistic scenarios is applied. Finally, a set of real world experiments are described.

Finally, the last part of this thesis concludes and summarizes this thesis and presents the envisioned future work and perspectives.

Collaborations

During this thesis several contributions where made within collaborations with other research teams. The concept and idea of SLSF and SLSR, as well as several publications that brought our research to those protocols, where made in collaboration with the MOCCA team at the University of Luxembourg, in particular with Adrian Andronache and Steffen Rothkugel. Another collaboration with this research team, brought the contributions on collaborative filtering with Patrick Gratz and Steffen Rothkugel.

A close collaboration with Julien Siebert and Vincent Chevrier of the Maia team at the LORIA in Nancy, permitted to contribute on advanced mobility models and to propose a new concept where mobility and behavior models are coupled in a closed loop with network simulation models.

Contributions regarding the service discovery architecture and the usage scenarios where done in a collaboration within the SARAH project, in particular with Laurent Reynaud of Orange Labs.

Part I

Foundations

This part presents the foundations needed for the comprehension of this manuscript. The first chapter introduces ad hoc networks, the specificities of dissemination in such networks and how routing and more generally topology control and organization are done.

The second chapter presents service discovery, the existing, more classical, protocols for wired networks and some for MANETs specifically designed protocols. Moreover, the chapter ends with a description of solutions that handle information correlation using context awareness and information overload with collaborative filtering.

Chapter 1

Introduction to mobile ad hoc networks

Contents

1.1 Mobile ad hoc networks 12
1.2 Dissemination, Routing and Topology Control 14
1.2.1 Introduction to routing protocols 15
1.2.2 Routing protocols in ad hoc networks 17
1.2.3 Dissemination strategies 21
1.2.4 Clustering 21
1.2.5 Dominating Sets 24
1.3 Conclusion 26

Mobile Ad-hoc NETworks, also called MANETs, are composed of a collection of devices that communicate with each other over a wireless medium without a fixed pre-existing infrastructure. Such networks are formed spontaneously whenever devices are in transmission range. Potential applications of such networks can be found in vehicle communication scenarios, environmental observations, ubiquitous Internet access, and in search and rescue scenarios as described in detail in [Sant 05]. Mobile ad hoc networks are a part of the so-called dynamic networks. Dynamic networks are networks where dynamicity takes a very important role, thus a network where topology changes from frequently to constantly. Dynamicity can occur in different ways. For example in Peer-to-Peer (P2P) systems, each user is a peer that connects to other peers to exchange data without need of any infrastructure (i.e. centralized server). The dynamicity in P2P systems is directly induced by the users who come and go in the network, a phenomenon called "churn". Another example of dynamic networks are the Wireless Mesh Networks (WMNs) where wirelessly connected devices form a (wireless) meshed infrastructure to extend a wireless network coverage without having the burden and cost of wiring. In mesh networks, the dynamic mostly occurs through radio propagation changes (signal losses, interferences, etc.) or device failures. Wireless Sensor Networks (WSN) are also dynamic networks. Wireless sensor networks are spatially distributed devices that monitors their surrounding environment using sensors and report the collected information to a so called network "sink", which gathers all the data. The sensor devices are designed to have very low computation and low energy consumption as they can be functional for a long time in unreachable zones. Dynamicity here occurs at dispersion of the devices on the to-be-monitored field as the network is formed spontaneously. The second dynamic occurs due to nodes going into sleep mode to reduce their power consumption or nodes failures (e.g. battery failure) that causes changes to the currently available network nodes. Compared to WSN, nodes of an ad hoc network have much more computational capacity and

power, however their life time is therefore reduced to a few hours instead of years in some sensor networks.

Ad hoc networks include all those dynamics: nodes can come and go spontaneously, nodes move almost continuously, nodes have limited energy and wireless communication failures. All those dynamics provide ad hoc network with continuously changing network topology which become very challenging with regards to persistent communication or stability of communication between ad hoc nodes.

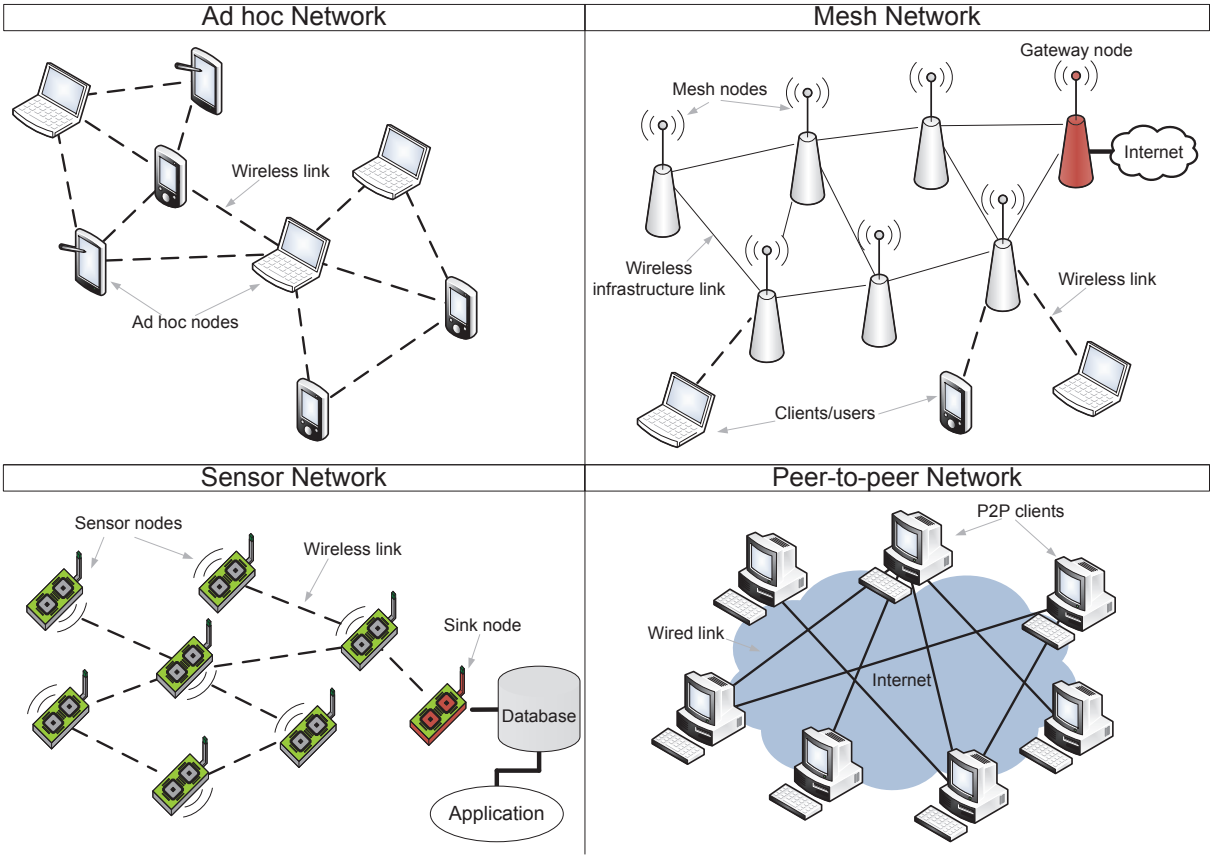


Figure 1.1: Dynamic networks: Ad hoc, mesh, sensor and peer-to-peer networks.

A first objective of this work is to build a stable structure on top of spontaneously created ad hoc networks. A major challenge is to reach a satisfactory stability by keeping the bandwidth low to create this stable structure.

1.1 Mobile ad hoc networks

MANETs are spontaneously created networks by wirelessly capable devices without any preexisting infrastructure (Figure 1.2). Nodes can only communicate directly with nodes that are in their radio transmission range. To reach distant nodes, the message is relayed by their neighbor nodes until reaching its destination. Each node may participate in the network by routing messages towards the destination. All contributions in this thesis rely on the principles that in an ad hoc network:

- Algorithms/protocols should rely mostly on locally available information.
- A node knows only with good accuracy itself and its direct (one-hop) neighbors that are inside its radio range.
- Distant information should be, when used, handled as inaccurate or unreliable.
- An ad hoc network should not rely on any infrastructure. In some specific scenarios however the presence of an infrastructure (e.g. server or router temporarily available) should benefit the existing ad hoc network (resulting in a similar network as mesh networks)

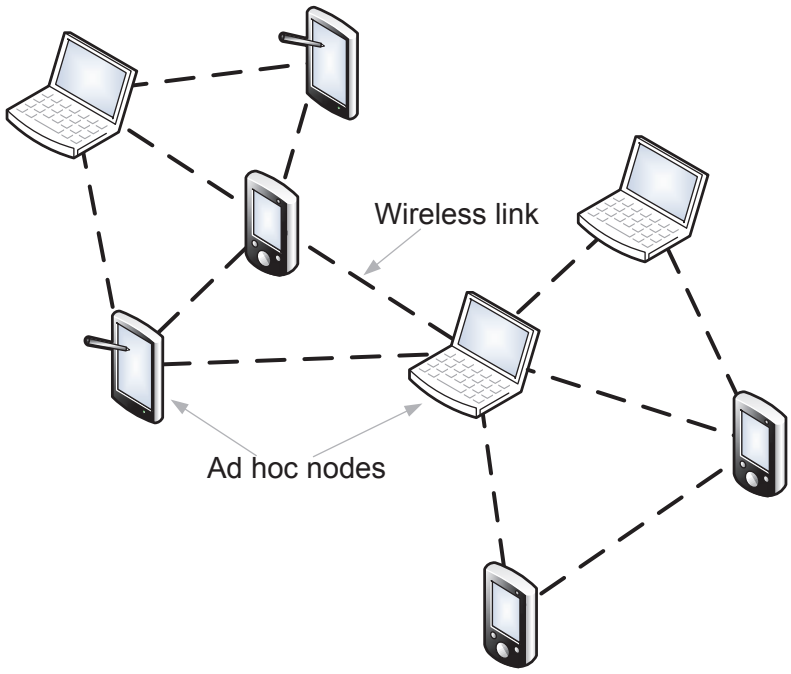


Figure 1.2: Ad hoc network composed of wirelessly connected devices.

A single protocol that is able to work optimally in all type of scenarios does not exist. Each protocol has scenarios in which it works at its best and others scenarios where it is non optimal. To obtain an optimal protocol, several protocol could be combined and appropriately selected depending on the situation. Two problems arise from this statement:

1. it is not trivial to know what protocol is best for a given situation (dynamicity, environment, etc.).
2. identifying a situation is not an easy task in the ad hoc network context: no global vision is available, only a partial spatial and temporal vision is available.

Another peculiarity of MANETs is the human part. Most devices are humanly operated on the application level but also the mobility is sometimes directly dependent on the user's behavior.

Most research done in MANETs tend to forget about the human part. This document proposes in Chapter 5 the first elements to include the human behavior in the evaluation loop and furthermore to consider the human aspect early in the design and development part of a protocol. This human behavior can be considered as a challenge, since it induces dynamicity and uncertainty, but it can also be taken as an advantage, by including and suggesting participative and helpful behaviors to the protocol (e.g. suggesting the user to move to a specific location to improve coverage).

Message transmission over wireless medium

Messages in networks can be classified by their target destination(s). A message is sent as *unicast* when only one node is designated as recipient by the emitter (Figure 1.3). A message is send as *broadcast* when all nodes should receive the message (Figure 1.4). Multicast messages are messages that have multiple destinations, called a group or multicast group (Figure 1.5). One must bear in mind that the wireless transmission of any message, regardless of their classification (unicast, broadcast or multicast), occupies the wireless medium in the same way. All devices in transmission range receive the message on their network interface disregarding the message type. The decision whether to pass the message to the higher layers is taken only based on the message type and destination fields. This is important for the design of a protocol. There is a tradeoff between one single broadcast containing information valid for all the neighbors occupying the medium for longer time and several smaller, thus shorter medium occupation, unicast containing targeted information for each neighbor. In the wired world, the situation is sightly different, a unicast is routed from the sender's machine via dedicated network switches or routers precisely (routing) to the receiver's machine. Also the wired medium is often a full duplex channel where incoming packets can be received at the same time as outgoing packets are send. In wireless transmissions the medium is shared and before any message is sent out a first phase of channel sensing is necessary to assure that the channel is free.

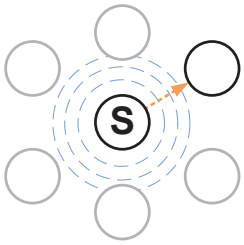


Figure 1.3: Unicast.

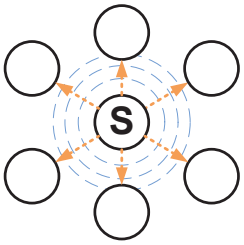


Figure 1.4: Broadcast.

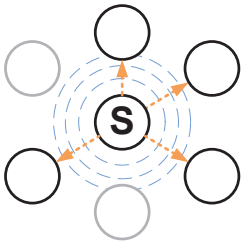


Figure 1.5: Multicast.

1.2 Dissemination, Routing and Topology Control

On a message exchange basis, service discovery essentially consists of sending advertisements or requests for services by disseminating messages through the network. Therefore, a first step towards improving service discovery is improving the underneath message dissemination. Dissemination here means distributing a message to all possible nodes or the intended group of nodes of the network in a timely manner. To do so, a message is relayed by intermediate nodes to reach distant nodes. There are various ways to achieve this. The most basic solution is flooding where each node forwards the message to its neighbors. However this behavior leads to the

well-known broadcast storm problem where nodes endlessly (re)broadcast packets [Ni 99]. To avoid this, broadcast schemes limit the number of forwarding nodes and hop count, delay and/or bandwidth used by structuring and designating the forwarding nodes.

A first example on how to achieve an improved communication are routing protocols (Sections 1.2.1 and 1.2.2). They use routing tables containing partial (e.g. a next-hop to destination list) or full path information (e.g. the list of traversed nodes to destination). To obtain this information several routing protocols structure the topology to reduce the routing overhead (e.g. the number of exchanged messages, the number of nodes forwarding routing information). So, some routing protocols propose interesting schemes to improve the dissemination of messages. In the rest of this document, we will focus on the dissemination performances of the various routing protocols rather than their routing performances.

There are also protocols dedicated to dissemination (Section 1.2.3). Unlike routing protocols they do not establish any routing tables or network wide persistent information, but instead propose strategies to reduce the overall dissemination overhead.

Basically, both, routing protocols or dissemination protocols, aim at selecting or keeping only the relevant links in the network. Relevant links can bee seen as:

- only the current active communication path, as in reactive routing protocols (Figure 1.6b), or
- a subset of minimal connected links to reach all the nodes in the network, as in dissemination protocols or proactive routing protocols (Figure 1.6c)

The problem of finding the minimal set of links or nodes to reach all the nodes in the network is a well known problem in graph theory known as Dominating Sets (Section 1.2.5)[Blum 04].

Another strategy to reduce forwarding nodes is to add a hierarchy to the network. Nodes are divided in smaller groups called clusters with, within each cluster, one node that is the central control and relay point that is called a clusterhead (Section 1.2.4).

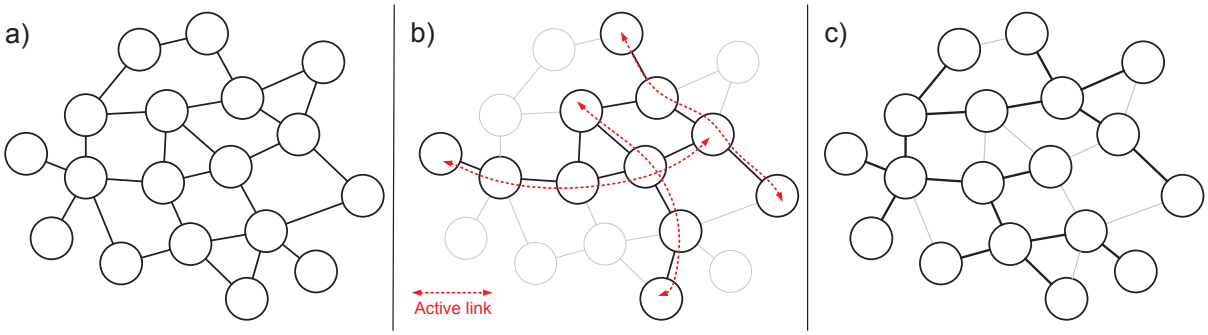


Figure 1.6: a) ad hoc network. b) only active links. c) subset of links for dissemination.

1.2.1 Introduction to routing protocols

Routing protocols construct the actual network by determining paths between nodes that are distant to each other. Discovering neighbors is straightforward since neighbor nodes can by definition communicate directly with each other. Discovering distant nodes requires a message exchange among participating nodes. The first goal of routing protocols is to obtain a path, whether it is a partial path where only the next hop is known or a full path with the complete list of nodes to be traversed.

Distance from a node: It can be computed by counting the cost of each traversed links separating a node from another node. Some protocols use weighted links to represent the cost of a link (e.g. throughput,bandwidth). In this case, the distance is a metric computed by addition of the link weights to reach a distant node. An un-weighted link is considered to have a cost of 1 (i.e. 1 is the smallest weight for a link).

Routing strategies that determine which information is exchanged with which nodes can be classified in two main categories: Distance-vector and Link-state.

Distance-vector routing: Implies two concepts: the *distance* to a destination and the *vector* (or direction) towards the destination. The vector is the next neighbor (also called next hop) to choose to reach the destination. Each node in the network exchanges the distance information about all nodes in the network only with its one-hop neighbors. As a result, a node only knows from which neighbor it learned the route but not from which node its neighbor has. Thus nodes have a one-hop visibility of the network. Only blur information (e.g. distance) is known for nodes further than one hop.

Link-State routing: Each node sends information about its neighborhood to all the nodes in the network. As a result, by overlapping the information from all the nodes, a node can obtain the entire network graph and compute the shortest-path tree for each destination. Maintaining link-state information requires more overhead than in distance-vector routing but ensures more robust and complete (i.e. complete network topology) information.

Another strategy differentiation that can be made, is the time when the information is refreshed: only on-demand as in reactive schemes, all the time as in proactive schemes, or both reactive and proactive as in hybrid schemes.

Reactive: In reactive schemes, route information is not kept up-to-date all the time but only computed on-demand when needed. The route discovery process can take a certain time, thus the route information is not available immediately. Reactive protocols are typically used in low traffic scenarios to reduce the bandwidth and energy consumption. During periods without application or data traffic, reactive protocols do not send any message.

Proactive: As opposed to a reactive protocol a proactive protocol sends periodically route information to keep information up-to-date. Route information for all the network is locally available immediately. The overall latency of a send package is reduced. Proactive protocols are well suited for high traffic and high mobility scenarios. The periodic route acquisition for all the network better cope with frequent route changes. In comparison, Reactive protocols in such scenarios, result in a constant sending of route demands and route error messages due to the constantly changing routes and topology. This leads to more control messages than the periodic updates of proactive protocol but with also more latency. On the opposite, proactive protocols in low traffic scenarios result during no traffic phases in useless sending of periodic messages and computing of routing tables.

Hybrid: Some protocols combine proactive and reactive schemes at different levels or different distances to reduce control messages. For example, by having a proactive scheme in the close neighborhood in order to have an up-to-date routing table for closer nodes and use a reactive scheme, only on demand, for the rest of the network only.

1.2.2 Routing protocols in ad hoc networks

In ad hoc networks, specially designed routing protocols are proposed to cope with dynamicity and unreliable communications. Due the fundamental differences between ad hoc networks and classical wired networks, routing protocols designed for wired networks can, most of the time, not be applied to ad hoc networks without any modifications. Following are several routing protocols that demonstrate well the kind of problems and solutions routing protocols for ad hoc networks address and propose. A more exhaustive and detailed classification of ad hoc routing protocols can be found in [Liu 05].

Destination-Sequenced Distance Vector (DSDV) routing DSDV [Perk 94] is historically one of the first ad hoc specific routing protocols. It is based on the basic Bellman-Ford algorithm [Bell 58][Jr 56] which computes shortest-paths in a weighted directed graph. Several modifications to the Bellman-Ford algorithm are done to solve problems such as routing loops or the count-to-infinity problem but also to suit the dynamicity of ad hoc networks such as topology changes.

In DSDV, each node periodically broadcasts the distance to each node it knows a path to (Example of node 2's forwarding table in Figure 1.7a). Additionally, to be able to distinguish new updates from old ones, a sequence number is attached to each entry in the routing table and update message. A node updating new information increments the sequence number of the corresponding entry by 2. On a link failure, the node detecting the failure sends out an update with hop count to the broken node set to infinite and increments the sequence number by 1 (Node 3 in Figure 1.7b). Node 2 and 5 receiving the updated route with a higher sequence number update their routing table accordingly and send out to their neighbor the updated information. Meanwhile, node 4 connects back to the network at node 8 and announces itself with a sequence number increased by 2 (Figure 1.7c). The propagating broken link update (Sequence number 21) will be overridden by the new fresher update (Sequence number 22) of node 4's new location.

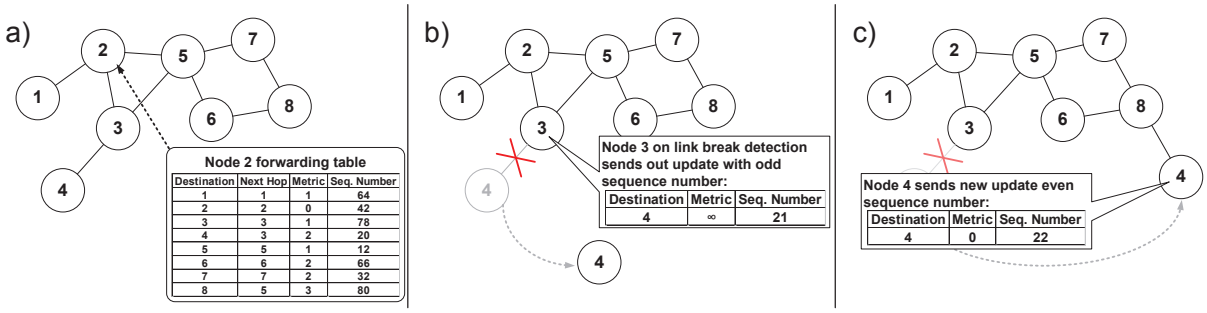


Figure 1.7: DSDV routing operations.

Ad hoc On-Demand Distance Vector (AODV) Routing AODV [Perk 03] is, as its name implies, a distance vector based reactive ad hoc routing protocol. It aims at being lightweight. To obtain a route to a destination a node sends out a broadcasted Route Request (RREQ) packet which is forwarded throughout the network until it reaches either a node that knows a path to the destination or the destination itself (Figure 1.8a). Upon reception of a RREQ, if a node already received a RREQ with the same Originator IP (IP address of the source of the message) and RREQ ID, it discards it silenlty, if it is an unseen RREQ it:

- updates the corresponding entry in its routing table if the sequence number of the RREQ is greater than the previously received RREQ sequence number.

- and sets as next hop in the routing table the node from which the RREQ was received (obtained from the IP header as source IP Address, not to be confused with the Originator IP address field of the RREQ).

This creates a reversed path along the initial RREQ (Figure 1.8b). After updating local informa-

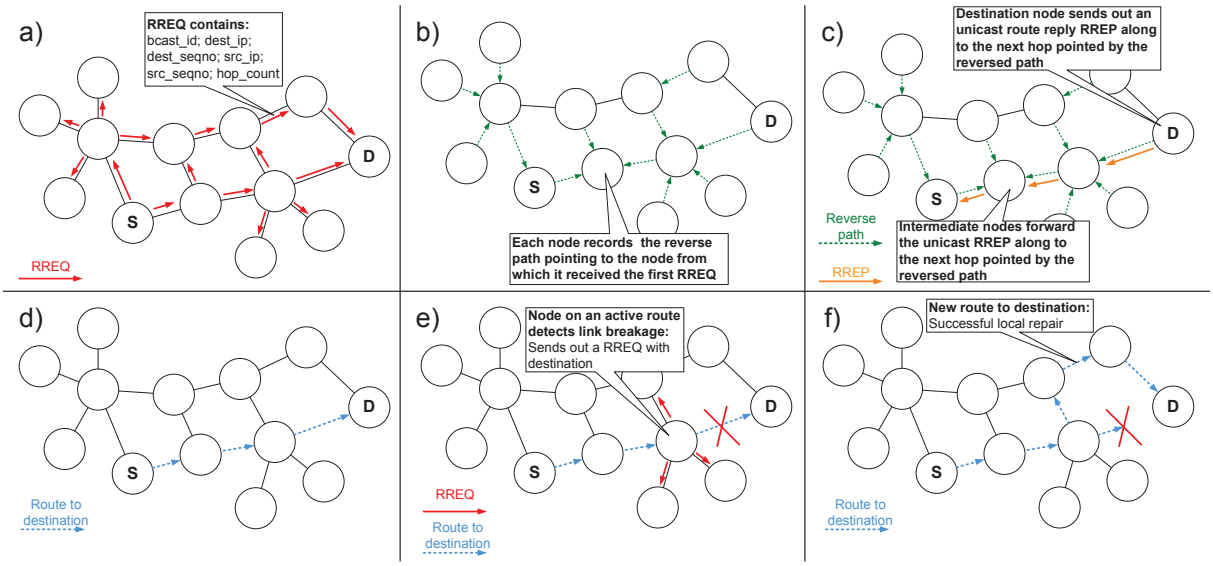


Figure 1.8: AODV route discovery operations.

tion a node that has no known route to the destination broadcasts the RREQ with the hop count value increased by 1. Eventually, the RREQ stops its progress at nodes which know a route or the destination itself which then send a unicast Route Reply (RREP) towards the source of the RREQ message (Figure 1.8c). The RREP follows the reverse path using at each passed node the next hop recorded during the RREQ transit.

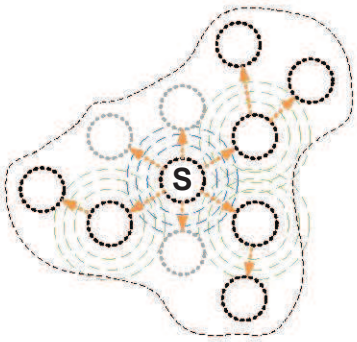
Once the RREP reaches the initial source of the route request the route is established and the communication between the source and destination begins (Figure 1.8d).

On active routes, intermediary nodes monitor their neighbors active links and react if a link on the active route breaks. At a link breakage, nodes first try a local route repair by sending out a new RREQ with a limited TTL value. During that local repair incoming data packets are buffered (Figure 1.8e). If the RREQ succeeds, the new route is notified to the upstream nodes and the communication continues (Figure 1.8f). If no RREP was received the upstream nodes will be notified with a Route Error (RERR) packet.

AODV is a very lightweight protocol with low processing needs and low memory overhead. Destination sequence numbers are used to ensure loop free message updates. It is well known in the ad hoc and sensor network community for its simplicity and also benefits of various existing implementations[1].

[1] A list of available implementations can be found here: `http://moment.cs.ucsb.edu/AODV/`

Bordercasting: is a dissemination technique where a message is send to the borders of a specific zone (e.g. a zone is given by the distance from the node). The goal of bordercasting is to reach all peripheral nodes by selecting specific relay nodes within the borders. Bordercasting can be compared with a limited multicast tree where one node communicates with a specific set of nodes.



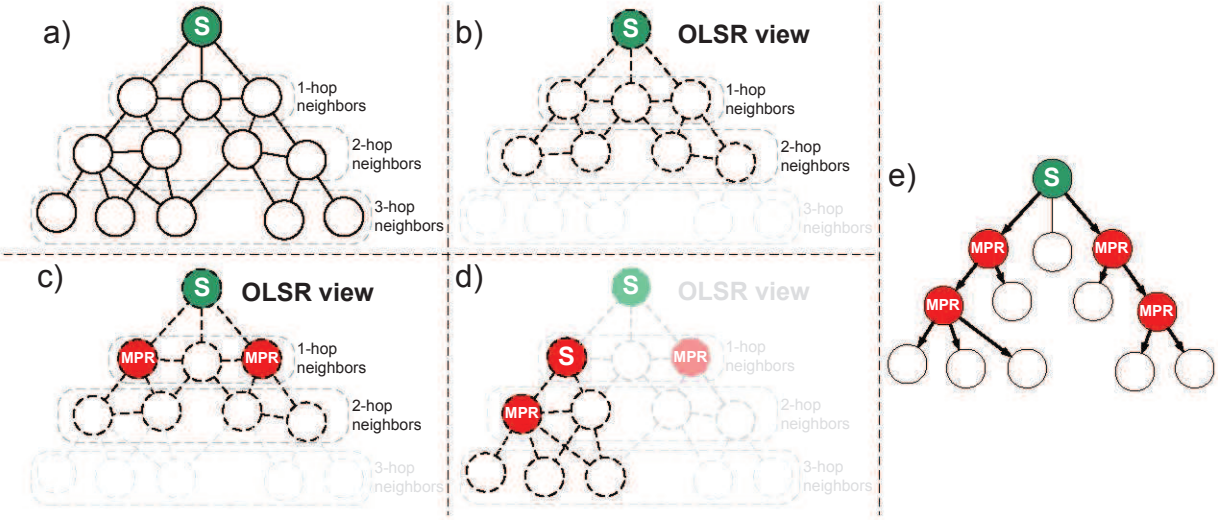


Figure 1.9: MPR selection (Bordercasting) in OLSR.

Optimized Link State Routing The Optimized Link State Routing Protocol[Clau 03] (OLSR) is a routing protocol, standardized by the IETF[2] MANET working group [Mobi], designed for ad hoc networks. It is a proactive protocol; hence it periodically exchanges topology information with other nodes of the network. One-hop neighborhood and two-hop neighborhood are discovered using HELLO Messages (similar to beacons). For example, in Figure 1.9a, the HELLO messages provide a node with a partial view of the network (Figure 1.9b). From this partial view OLSR then computes multipoint relay (MPR) nodes by selecting the smallest one-hop neighborhood set needed to reach every two-hop neighbor node (Figure 1.9c). Those MPR nodes then themselves, based on their local view, also compute MPR nodes (Figure 1.9d). To obtain global routing information each MPR node sends out, throughout the network, Topology Control (TC)

[2]Internet Engineering Task Force

messages containing its one-hop neighbors. However to reach all nodes in the network, only nodes selected as MPR forward (a.k.a. bordercasting, i.e. nodes bordercast the message to their MPR nodes) TC messages, thus reducing the number of forwarding nodes (Figure 1.9e). Eventually every node, MPR or not, possesses a routing table containing the shortest path to every node of the network. OLSR enables optimized flooding of the network by building a tree-like topology for every node from a source (Figure 1.9e). OLSR is a good candidate for data dissemination, since independently of its routing mechanism (i.e. the TC messages), the MPR mechanism provides an optimized 2-hop coverage. However, as shown later in Section 3.2, OLSR suffers from a lack of scalability. To maintain the MPRs, in OLSR each node exchanges HELLO messages with each of his neighbors containing each the list of his neighbors. Therefore with a growing network the exchanged control data increases rapidly and eventually occupies all the available bandwidth.

Zone Routing Protocol The Zone Routing Protocol (ZRP)[Haas 02] is an example of a hybrid protocol. It combines proactive routing inside a zone using bordercasting and on-demand routing outside. Each node is the center of its own zone and the zone radius is defined by a hop count to the border nodes. In a 2-hop zone radius, each node maintains node and path information to all the nodes at 2 or less hops away (Figure1.10a). Inside a zone the Intrazone Routing Protocol (IARP) proactively maintains topology using a modified distance vector scheme. A node willing to reach a destination first checks if the destination is in its own zone, if not it uses the Interzone Routing Protocol (IERP) and bordercasts the Route Request message to the border nodes of its zone using information collected by the IARP (Figure1.10b).

Bordercasting is broadcasting a message to nodes that are at a given distance k (k>1) called border nodes. In ZRP, border nodes are nodes at the zone radius distance. Several mechanisms exist in order to reduce redundancy: query detection, early termination and selective bordercasting. Selective bordercasting uses a similar mechanism as the MPR computation in OLSR and selects a subset of 1-hop nodes in charge of reaching full border node coverage.

Each border node that receives the route request, repeats the same procedure and checks if the destination is inside its zone. Thereby, the zone is centered on the current node and directed towards unexplored zones. If the destination is inside its zone it sends back a route reply packet along the path collected by the route request (Figure1.10c), if not it bordercasts it to its own border nodes.

If we consider only information dissemination in ZRP and since every zone is centered on the current node, the ZRP dissemination results in a plain bordercasting of messages towards the destination. Route maintenance inside a zone in ZRP has similar scalability issues as OLSR since it requires k-hop information (with k>1).

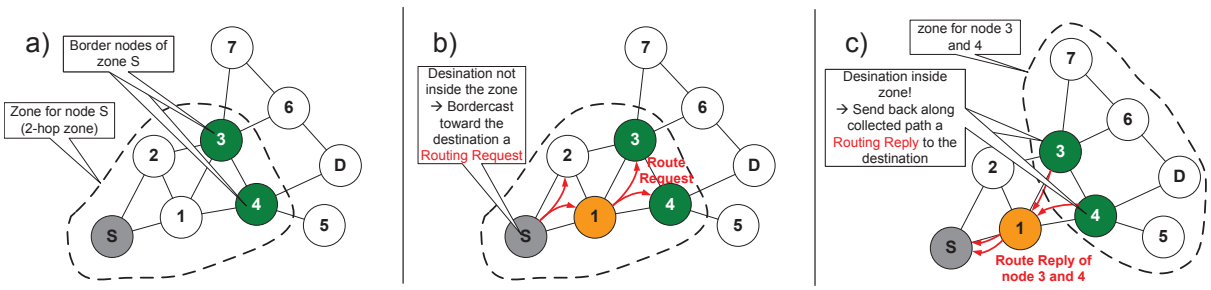


Figure 1.10: Zone Routing protocol - Route discovery.

1.2.3 Dissemination strategies

Another strategy to reduce redundant messages, used in many ad hoc protocols, is the selection of forwarding nodes. In [Peng 01] broadcast relay gateways (BRG) are selected with 2-hop knowledge. The rule is simple, every 2-hop node has to be covered by at least one BRG. BRGs are only calculated on demand and picked out along with the propagation of the broadcast. Another 2-hop knowledge approach is DCB (Double-covered Broadcast) [Lou 04] which selects gateway nodes again to cover the 2-hop neighborhood. DCB selects the forwarding nodes in such a way that every 1-hop non-forwarding node is covered by at least two nodes (double-covered), the sender and one forwarder, thus double covering the 1-hop neighborhood. DCB also provides a Re-select algorithm which, in case the source does not detect its selected forwarders retransmission signal (e.g. the forwarder moved out of range), re-selects new forwarder nodes which then deliver the broadcast to the 2-hop neighbors. Also a 2-hop knowledge based approach is the previously described OLSR protocol. We use OLSR as comparison and inspiration because it is a standardized [Mobi], widely used and deployed protocol in ad hoc networks and Mesh networks, and also because it is proved to optimize coverage of 2-hop nodes through MPRs [Jacq 01] (i.e. minimize the 1-hop forwarders to cover all 2-hop nodes), thus provide a good dissemination mechanism.

1.2.4 Clustering

In ad hoc networks, forwarding strategies should be employed to avoid broadcast storms (i.e. a message forwarded by all the nodes in the network in an endless loop. As depicted in [Ni 99], broadcast storms can be counter-measured using several schemes i.e. probabilistic, counter-based, distance-based, location-based and cluster-based. We use the latter scheme, cluster-based, since it is the only one based on network topology information which is easily and reliably available (as opposed to geo-location information for example).

Clustering is a strategy that organizes the network into groups. Each group is called a cluster with a group leader called a clusterhead. Group members are slaves or cluster members. The size of the cluster can vary depending on the algorithms. Most of the time the size is expressed in hop distance or radius from the clusterhead. Clusterheads can be decided using an election where member nodes exchange messages and elect the clusterhead. They can also be based on a designation where each node designates the node in its neighborhood (i.e. within the cluster radius) which is its clusterhead. Depending on the algorithm, clusters can also overlap, thus having two or more clusterheads within one cluster radius. A cluster can contain several clusterheads when a clusterhead selects another clusterhead, it then becomes a subhead for that selected clusterhead.

For example [Ni 99] proposes a clustering technique where the clusterhead is elected after a message exchange among the neighbors. To send a message to a destination the route is constructed on demand. The broadcasted demand is sent among all the clusters being forwarded by all the nodes reaching nearby clusters (also called gateway nodes). Every crossed cluster adds its address to the message so when finally one cluster contains the destination node, the message is sent back through the route it came from, using the information collected by the demand message.

In [Foro 05], the authors construct elected clusters based on beacon information which provides the number of neighbors and their stability represented by the number of beacons received since the node became a clusterhead. This clustering approach is similar to NLWCA (Node and Link Weighted Clustering Algorithm, detailed in the next section) but does not rely on both, a link weight and a node weight which provides NLWCA with more flexibility in terms of cluster selection. Another similarity to our approach is the forwarding node selection (named gateway

selection). They present a protocol which enables nodes to be selected as gateways to be able to relay outgoing traffic from a cluster. However the selection requires a message exchange (GATEWAY_Claim and GATEWAY_Grant) which can easily lead to loss of consistency. An important aspect is that gateways are only selected for outgoing traffic. For external (incoming) traffic they establish a Random Assessment Delay (RAD) for nodes receiving a foreign cluster message but not selected as gateway. As a consequence, unless two adjacent clusterheads elected (independently) two adjacent nodes as gateway, every single message passing from one cluster to the other will be delayed by the RAD. The main differences of our approach are (1) no message exchange except the payload broadcast itself is used to select forwarding nodes and (2) unless for error recovery no additional delay is caused by SLSF. The comparison with the protocol presented in [Foro 05] will not be further investigated since to our knowledge, we lack the description of how precisely the gateway selection is done.

NLWCA – Node and Link Weighted Clustering Algorithm

NLWCA [Andr 08b] organizes ad hoc networks in one-hop clusters by using only information available locally. Each device elects exactly one device as its clusterhead (CH). The main goal of NLWCA is to avoid superfluous re-organization of the clusters, particularly when clusters cross each other. Thus providing a certain stability (Figure 1.11).

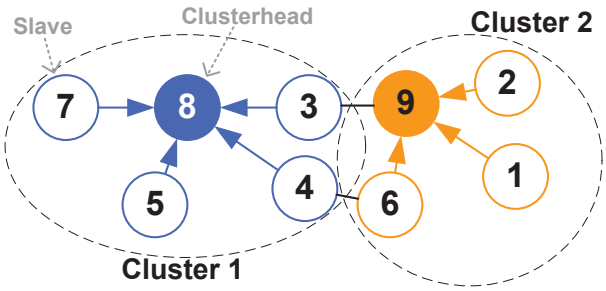


Figure 1.11: Example of two clusters built by NLWCA.

In NLWCA, each node has a *node-weight* and a *link-weight* assigned to each link connected to a one-hop network neighbor node. This link-weight is used to keep track of the connection stability of the one-hop network neighbors. When a link weight reaches a given stability threshold it is considered stable and the device is called stable neighbor device. The clusterhead, elected only from the set of stable neighbors, is the node with the highest node-weight (Note: nodes elect themselves as clusterhead if their node-weight is the highest). This avoids the re-organization of the topology when two clusters are crossing for a short period of time (Figure 1.12).

Node and Link weights: NLWCA has two sorts of weights:

- The node weight, that is a simple numerical weight representing a node. In [Andr 08b], the node weight is resulting from a formula taking as input the signal strength and the power level of the node. However, the node weight could also include much more parameters or a single one. The calculation depends on the context of the node and the protocol usage scenario.
- The link weight, represents the duration a link is active. Each nodes assigns to each of its external links a weight to keep track of the link stability.

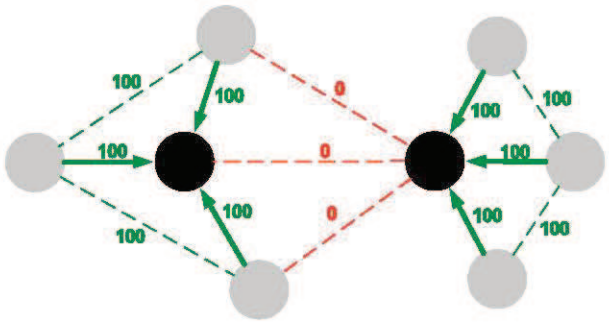


Figure 1.12: The low weight of the links avoids superfluous re-organization of the topology when for instance two clusters cross in mobile networks.



Figure 1.13: NLWCA beacon containing weight and designated cluster address.

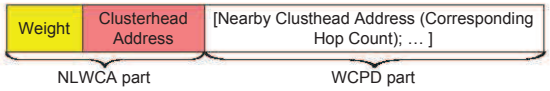


Figure 1.14: NLWCA and WCPD Beacon.

NLWCA uses periodically broadcasted messages (beacons) containing the own node weight and the designated clusterhead address (Figure 1.13). Beacons have two purposes. First, they contain information about which neighbor nodes can be considered for the clusterhead selection (i.e. all the nodes from which a beacon was received) and information to know which clusterhead a neighbor selected (can be itself). Second, the periodicity of the beacons is used to compute the link stability. Each time a beacon is received the link weight is increased. To avoid short disconnections (e.g. transmission errors) from disrupting the stability process of a neighbor, not-received beacons decrease the link weight by a given value, instead of minimizing it immediately.

WCPD – Weighted Cluster-based Path Discovery protocol

The Weighted Cluster-based Path Discovery protocol (WCPD) [Andr 08a] is designed to take advantage of the cluster topology built by the Node and Link Weighted Clustering Algorithm (NLWCA) in order to provide path discovery and broadcast mechanisms in mobile ad hoc networks.

WCPD, on top of NLWCA, discovers nearby stable-connected clusters in a pro-active fashion. For the nearby CHs discovery algorithm, WCPD uses the beacon to detect devices in communication range. NLWCA and WCPD combined provide to each node, through the beacon (Figure 1.14), the following information about each stable one-hop neighbor: its weight, its CH ID, the ID set of discovered CHs and their respective path length.

The WCPD broadcasting algorithm is simple (Figure 1.15): the broadcast source node sends the message to the CH, which stores the ID of the message and broadcasts it to the one-hop neighborhood. After that, it sends it to all nearby CHs by multi-hop unicast. The inter-cluster destination nodes repeat the procedure except that the message source clusters are omitted from further forwarding. Additionally, the information about the ID of the broadcast messages and their sources is stored for a given period of time to avoid superfluous re-sending of the message.

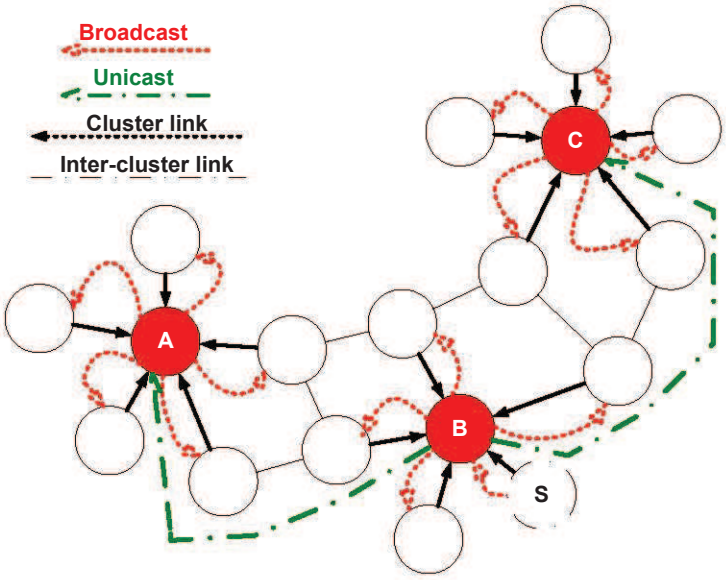


Figure 1.15: WCPD message dissemination through a network.

1.2.5 Dominating Sets

Another way to represent an ad hoc network is using graphs. An ad hoc network is there represented by a graph $G = (V, E)$ where V (Vertices) represents the set of ad hoc nodes and E (Edges) the set links. So $e = \{u, v\}$ means that u and v are in wireless communication range of each other, so e is a undirected link between u and v. G is therefore an undirected graph. Using this representation, a dominating set $D \subseteq V$ is the subset of nodes in the set V that dominates all the nodes of the graph. In other words, all the nodes not in the dominating set D have at least one adjacent node that is in the dominating set D (Figure 1.16).

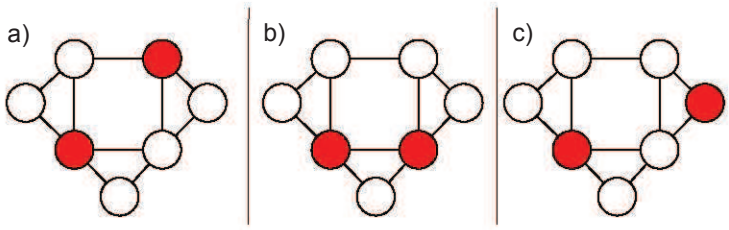


Figure 1.16: Dominating sets (red nodes).

A dominating set in an ad hoc network thus represents a specific set of nodes that are able to reach all the nodes of the network with one broadcast. Those injection points are very useful to efficiently and rapidly spread information. However, most of the time it is not possible to have several injection points. Thus, the dominating nodes additionally need to be interconnected together to be able to spread information throughout the network with one single injection point. **Connected dominating sets** (CDS) is the representation of this need of interconnection between dominating nodes. CDS have the same basic rule as a normal dominating set, but with an additional constraint that any node in D can reach any other node in D by a link that stays entirely within D. Thus, D becomes a connected subgraph of G. An example of a CDS is Figure 1.16b, both dominating nodes can communicate directly without passing through another non-

dominating node. In Figure 1.16a and c however, the dominating nodes are not directly connected and therefore they are not CDSs. Nodes in a connected dominating set form a backbone structure from which ad hoc networks can greatly benefit to efficiently disseminate information to all the nodes by using only one injection point as opposed to simple dominating sets where multiple injection points are necessary.

Clustering techniques are closely related to dominating sets. For example a one-hop clustering algorithm (i.e. NLWCA) builds a structure corresponding to a dominating set. In one-hop clustering algorithm, a node is either a clusterhead or covered by at least one clusterhead while in a dominating set each node not in the dominating set is dominated by at least one dominator. Thus a clusterhead corresponds to a dominator and a slave node is a dominated node.

An extensive survey on CDS in ad hoc and sensor networks can be found in [Blum 04]. The authors describe centralized and distributed algorithms. Centralized algorithms take advantage of their global knowledge about the graph to optimize the CDS construction. To cope with the ad hoc network paradigm, theoretically centralized solutions can be implemented in a distributed fashion, but with a higher communication overhead. Therefore distributed techniques propose to reduce the overhead without strongly impacting the resulting CDS. In the context of ad hoc networks and distributed algorithms a tradeoff between complexity (computing time), running time (converging time), stability (with respect to nodal movement) and communication overhead (message exchange) is necessary.

An example of a completely localized CDS construction algorithm is given in *Wu et al.* [Wu 02]. Here each node exchanges its local neighborhood with its immediate neighbors, thus each node knows its 2-hop neighborhood. As a first step, any node having two unconnected neighbors is marked as a dominator. The resulting set is a dominating set with a lot of redundant nodes. Therefore the set is post-processed with a general pruning rule: a node u can be taken out of the set if there exist k connected neighbors with higher IDs that can cover all u's neighbors.

Another CDS algorithm based on 2-hop knowledge is [Adji 02] which relies on the MPR mechanism of OLSR (Section 1.2.2) to construct a CDS. The structure formed by the MPR selection (i.e. selecting the smallest 1-hop neighborhood that covers all the 2-hop neighborhood) creates localized CDSs. To form a network wide CDS two algorithms are proposed: MPR flooding based and MPR-CDS. In MPR flooding the CDS is obtained by the flooding of one message through the network from one particular source. The CDS is then formed by the source itself and each node that retransmitted the message (MPR nodes). For the MPR-CDS algorithm, a node is said to be smaller than other if it has a smaller ID (e.g. IP address can be used as ID). Each node decides whether it is in the dominating set based 2 rules: (1) it is smaller than all its neighbors or, (2) it is MPR of its smallest neighbor. The advantage of the MPR-CDS algorithm is that the CDS construction is localized on each node itself while in the MPR flooding algorithm, a message has to flow through the network for the CDS to be formed. Moreover, MPR flooding algorithm can be seen as a particular case of the MPR-CDS where rule 1 only elects the source of the flooded message.

In the remaining of this thesis dominating sets and connected dominating sets are not the main matter. They will however be evoked on specific parts such as clustering algorithms and flooding optimizations (Section 3.4). Moreover, many of the work presented in Chapter 3 can be related to the dominating set research.

We define a **service** is an entity that is the representation on the network side of a piece of software or hardware that is able to execute a task or several tasks coming from an external requester. Thus a service is not necessarily a single part of software or hardware but can represent the service accomplished by a set of software and/or hardware combined together.

The service presented to the network usually also contains a human description, a set of properties and options, the interaction protocol description and its network address/port. For example a printer service is the network representation of a physical printer (hardware) and its printing software. On the user and network side the advertised printer service provides information about the printer location (human description), its properties (e.g. color printer, paper size, etc.), the printing protocol (e.g. IPP - Internet Printing Protocol).

1.3 Conclusion

This chapter presented the basics of ad hoc networks, dissemination, routing protocols and topology control. We can retain from this chapter the various techniques used to discover the neighborhood and organize the network into smaller groups or build a specific path to a set of nodes. Routing then additionally gathers paths for more distant nodes. In the remaining of this thesis, we use those techniques as root for our technical choices and contributions:

- Clustering will organize nodes into small groups, easier to handle with; We use and contribute to NLWCA.
- Neighbor discovery techniques (such as the MPR mechanism) will inspire the path setup between those small groups; We propose the SLSF protocol with the Inter Cluster Relay mechanism that wisely selects nodes among the NLWCA created clusters.
- Classic routing techniques, will provide a strong basis to improve routing efforts using the constructed structure; We propose SLSR, a routing protocol that uses both link-state and distance vector techniques to minimize the network cost.

Chapter 2

Introduction to service discovery and context in ad hoc networks

Contents

2.1 Naming Service 28
2.2 Introduction to service discovery 28
2.3 Service discovery in wired networks 30
2.3.1 SLP 30
2.3.2 UPnP 30
2.3.3 Zeroconf 30
2.3.4 Jini 33
2.4 Service discovery in ad hoc networks 34
2.4.1 Konark 34
2.4.2 Allia 34
2.4.3 Service discovery over OLSR 35
2.4.4 Service discovery using a field theoretic approach 35
2.5 Zeroconf as service discovery protocol for ad hoc networks . . . 36
2.5.1 Zeroconf and its multicast structure 36
2.5.2 Why does Zeroconf use multicast? 37
2.5.3 Conclusion 38
2.6 Context & Metrics 38
2.6.1 Social: Centrality 38
2.6.2 Collaborative Filtering 40
2.7 Conclusion 42

This chapter introduces the elements that are between the user and the network. Once the network is set up and routing is available, to render communication among nodes in an ad hoc network user friendly, service discovery allows to automatically detect nodes and services which then are displayed in a human friendly way. Furthermore, to adapt the service discovery to the user's environment, the context is taken into account.

2.1 Naming Service

From a network perspective, using dissemination and routing protocols, nodes can communicate with each other by using their IP addresses as destination address for their messages. However from a user point of view, communication is still very unfriendly to use with only IP addresses. While for Internet access, in most of the local area networks, a default gateway is designated as the outgoing IP address (e.g. IP address of the modem/router). For communication inside local networks however, one needs to know the IP address and port number of the destination. Suppose we have an application at the user level that kindly displays the IP addresses and ports available and provides a way to simply send messages to a given IP address/port. The user does not know which IP address is assigned to which device or exactly which port to use. In small networks with only a very limited number of computers (e.g. at home), the user knows the IP address and can remember them or hard code them in the configuration. Nevertheless, in bigger network or dynamic network remembering which IP address is assigned to which device is not feasible anymore. Therefore, a first improvement is to bind an IP addresses to a human friendlier name called a hostname. In managed networks (e.g. in a company, university) there usually is a DNS (Domain Name System) [Mock 87] server that keeps track of IP address/hostname pairs by receiving registration messages from the devices/machines. The DNS server ensures the uniqueness of the hostname at registration and translates hostnames to IP addresses. In smaller networks (e.g. at home) hostnames, if used, are usually configured manually on each machine. Note that in any given network the hostnames have to be unique inside this network since the translation must be unambiguous in both ways. Using hostnames with DNS or manually configured, users now, instead of having to remember an IP address, just need to remember the human-friendly hostname.

2.2 Introduction to service discovery

Even with routing protocols and human friendly hostnames available, there still is a part missing for the end user. For example lets suppose a new user is in a building is connecting to the local network. Since he is new, he does not know the hostnames of any device in the network. Even if this user could recognize the function of a device by its name (e.g. printer-210), he would probably not be able to know where it is located nor how to communicate with it. Now, if instead of 1 printer there are 10 printers with hostnames "printer-XXX", the only way to know where is printer is located, is to ask where this printer is and which printing protocol(s) it can understand. Service discovery solves exactly these situations.

Generally speaking, the goal of service discovery is to find services provided by other nodes in the network in an automated way and use them only by knowing a basic set of information.

Service discovery facilitates the location (in terms of network and physical location), usage and configuration.

Back to the example of our new user entering a building and connecting to the local network. This time, he uses service discovery and soon after his entrance, the service discovery protocol displays him the list of available services in the network. He is interested in printing a document in color and therefore only requests for services proposing a color printing service. He now has the list of color printers available in the building, however this time with location information about the printer in the building included in the service description. Eventually he chooses the printer closest to him and shortly after his device is automatically configured to communicate with this printer. Finally he can use the right printer with the right configuration without prior knowledge of the network and the devices connected to it.

In the last years, a wide range of services became popular, like music sharing, game services

or gateway services providing Internet access. Without infrastructure, as in ad hoc networks, the need to automatically, hence not manually (which would be too complicated and time-consuming with regards to the frequent changes in the topology and configuration), discover services that the network offers is even more crucial than in classical wired networks as no central information is available. Service discovery is essential especially for nodes with limited capabilities, that want to use a service while not having the capability to host or run it by themselves.

Service discovery can be categorized using two criteria, their discovery mechanism and the presence of a service directory (Figure 2.1) [Duda 02, Ciar 02]. There are two main **discovery mechanisms** in service discovery protocols, passive and active:

- Passive: the node searching a service (Client) listens passively to the service announces on the network and caches them over time. When a client searches a service it consults its local cache and communicates with the matching service. The advantage of passively discovering service is that only nodes providing services need to make announces on the network, thus keeping the communication low. However, passive discovery can miss idle services which is the case when the node proposing the service is in sleep mode.

- Active: the searching node sends a service request containing the searched service type or name and sometimes specific service parameters required by the application. All nodes matching the request send a response message back to the sender. The advantage is that active discovery finds all matching services currently available in the network. However, active discovery is often too invasive since every node in the network can request services.

Another differentiation of service discovery protocols are the use or not of **service directories**. Service directories are nodes that play the role of intermediary and ease the load of neighboring nodes. Those directory nodes store, forward and respond to incoming queries from other nodes. When a node sends a service query in the network the directory node, instead of forwarding the query can, respond directly to the source node if it knows a matching service. The other way around, directory nodes can filter and regroup service discovery announces to reduce the bandwidth usage. To be the most efficient and useful, service directories have to be placed carefully on "well positioned" nodes and "stable" nodes (what a well positioned and stable node is, will be defined later in Section 2.6.1). Some service discovery protocols use hybrid techniques combining passive and active discovery and/or use temporary directory nodes only when needed.

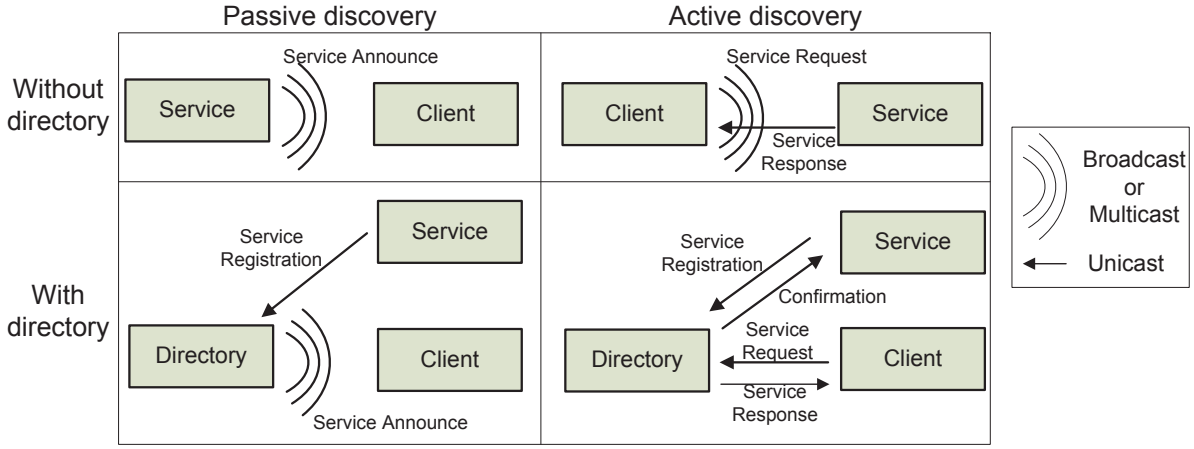


Figure 2.1: Service discovery protocol classification. [Ciar 02]

2.3 Service discovery in wired networks

Following is a short description of fundamental standardized service discovery protocols that exist in wired networks. Zeroconf is further detailed since it is the service discovery protocol used later on in the contributions.

2.3.1 SLP

Service Location Protocol [Veiz 97] uses the URL (Uniform Resource Locator) scheme to locate services within a network. The url has the following format: "`service:<srvtype>://<addr-spec>`" where "srvtype" is the type of service that is registered with the IANA (Internet Assigned Numbers Authority) and "addr-spec" the mean of access to the service (`[<user>:<password>@]<host>[:<port>]`) by the user/client. In smaller networks, the users (User Agents) send their request using the service specific multicast address. Nodes hosting this type of services (Service Agents) listen on this multicast address and respond to the user if they match the request. In bigger networks, SLP uses directory nodes (Directory Agents). Service Agents need to register their services to a Directory Agent , if present. User Agent then unicast their request to Directory Agents which respond accordingly with the corresponding service URLs.

2.3.2 UPnP

Universal Plug and Play [UPnP][UPnP doc], developed by Microsoft and IBM extends the concept of "plug and play", used in computer to detect automatically new connected hardware, to the home network. UPnP uses SSDP[3] to discover services in a local network. Service are described using XML[4] files. UPnP sends out requests using the SSDP that returns the matching URL[5] pointing to the XML file description of the service. UPnP is widely used in home networks boosted by Microsoft operating system usage. UPnP is described in further details in Appendix A.1.

2.3.3 Zeroconf

Zero configuration Networking [Ches 05, Zero] is a combination of different techniques that result in a fully usable IP network without the need of any configuration, infrastructure (e.g. DNS server, DHCP server) or network expertise. Zeroconf has three core elements: the automatic assignment of an IP address, the DNS functionalities (e.g. name resolution) without any central server (mDNS) [Ches 11] and the automatic discovery of services available in the network (DNS-SD) [Ches 06]. Zeroconf message formats are similar (even compliant) to standard DNS messages and are delivered using multicast address assigned by the IANA (224.0.0.251 for IPv4 and ff02::fb for IPv6 link-local addressing).

Zeroconf's early beginnings started with Stuart Cheshire in some postings on the Net-Thinker mailing list in 1997 and finally resulted in the IETF Zero Configuration Working Group[6] in 1999. In May 2002, Apple registered a trademark named "Rendezvous" for the Zeroconf technologies, however in April 2005, due to a conflicting name with another networking company, Apple changed the name to the current "Bonjour". Open source implementations also exist like for example Avahi[7] or JmDNS[8]. Following are the details for the three main elements that form the

[3] Simple Service Discovery Protocol[Gola 99]
[4] eXtended Markup Language[Bray 08]
[5] Uniform Resource Locator[Unif 94]
[6] `http://datatracker.ietf.org/wg/zeroconf/charter/`
[7] `http://avahi.org/`
[8] `http://jmdns.sourceforge.net/`

Zeroconf technology: link local-addressing, multicast DNS (mDNS) and DNS service discovery (DNS-SD). Note that Zeroconf technology can be seen as a guideline which contains requirements which each part should implement, but is not dependent on a specific implementation. A very good description of Zeroconf can be found in [Ches 05].

Link local-addressing

The first requirement in an IP network is an IP address. When using IPv4 one can either configure the address manually or obtain through DHCP[9] server in the network that automatically assigns them. However, when neither of those are possible or available, Zeroconf will use a self-assigned IPv4 link-local address (169.254.0.0/16) instead. The mechanism called *Link local-addressing*[10] to self-assign an IP address is as follows: The device selects a random address within the 169.254.0.0/16 range and sends ARP request to check whether the selected address is taken. If, after a timeout, no reply has been received it uses the selected IP address and replies to future ARP requests. Mechanisms to deal with duplicate IP addresses, for example when two partitioned networks join, exist. They use a simple defend and abandon principle: defend your IP address, at duplicate detection, by informing the duplicate IP owner that you are the owner. If no reaction or receiving the same message, which is legitimate since the other owner acquired the address correctly, back off and restart the random IP selection process. Bear in mind that those reaction are not humans and do not have pride. Moreover, it can be more beneficial to switch rapidly to a new IP address, while losing some possible established TCP connections, than "fighting" for an address that already affects the existing TCP connections due to routing errors induced by duplicates IP. Another advantage is that if a malicious or even manually configured device "kidnaps" an IP address the problem is quickly solved by an IP address change. Whereas IPv4 need a specific "add-on", IPv6 natively proposes the self-assigned local link address when no other configuration is available.

mDNS – Multicast DNS

Once an IP has been obtained, one needs to be able to assign a name (or hostname) that can resolve to this address. The method used to assign a name is completely independent of how the IP address was obtained, whether it was using link local-addressing, manually configured or using DHCP. Using names rather than IP addresses has advantages:

- Provided IP address may be temporary. A device using a given IP address, may at a later time, have changed its IP address. Using this given IP address to contact the device may result in an error or worse in contacting the wrong device.
- Names are human-friendly. Remembering and typing names is easier than an IP address. With IPv6, addresses become even less human-friendlier (32 hexadecimal characters) to remember or type than with IPv4 (4 decimal numbers).

All those points become even more crucial in MANETs. DNS servers that assign globally unique names on the Internet do their task very well, but when it comes down to a small network or infrastructure-less network, DNS servers do not suit in terms of configuration effort. In those networks, Multicast DNS[11] (mDNS) is a better fit since it produces the same result as a classical DNS server but without centralized authority. Using mDNS, every device answers for itself. When a device queries for name, instead of sending it to the central DNS server, it sends it using

[9] Dynamic Host Configuration Protocol[Dyna 97]
[10] `http://files.zeroconf.org/rfc3927.txt`
[11] `http://www.multicastdns.org/`

the multicast address (i.e. 224.0.0.251 for IPv4 or ff02::fb for IPv6). Using multicast, only nodes interested in that information, thus those who subscribed to this multicast group, will receive the query. Each device in the network running mDNS, listens to the multicast address and upon an incoming query, if it can respond to it, answers that query, much like a conventional DNS server would have done.

To claim a hostname, mDNS first sends 3 DNS queries (probes) and waits 250ms after each query. If no conflicting multicast DNS responds have been received, the host then moves to the announcing phase where it announces itself using an multicast DNS response to update neighboring caches. From this point on, anytime a host receives DNS query containing its name it responds to it, if it receives a DNS response with its name, thus a message that disagrees with its own record, its a conflict. Another device is claiming to be the owner of the same name. Both devices move back to the probe phase at which moment a tie breaker rule [12] will determine which device wins and keeps the name and which one has to change its name. Names assigned by mDNS are easily identifiable by their ".local" suffix.

DNS-SD – DNS Service Discovery

Up to this point, using Zeroconf, we obtained an unique IP address and an unique hostname associated with it. No need to remember IP addresses of nodes, names are sufficient. Are they? Not really since we do not necessarily know those names. Therefore Zeroconf also includes service discovery named DNS-SD[13], to be able to discover the names and descriptions of services in the network. DNS-SD is lightweight and builds on the already well established DNS standard. DNS-SD uses the "SRV" (SeRVice) type of the DNS protocol family (Figure 2.2) to advertise services. DNS-SD only adds to the standard DNS protocol the ability to display a list of available services to the user. It takes advantage of the existing PTR (PoinTeR) records of DNS, originally used to resolve an IP address to its hostname (a.k.a. reverse lookup). Here, PTR records are used to link a service description/type of service contained in the PTR record (useful for queries) to its service record (SRV record). The answer for a query asking for a service type "_ipp_tcp.local." on the PTR record of Figure 2.3 will contain all service instance names matching that service type, thus the list of all available services. PTR records contain a short description of the service, however in some cases this is not sufficient. Therefore, DNS-SD uses the DNS TXT records that can contain a series of key/value pair attributes data in the form "key=value" [14], to include additional informative data about the service. Figure 2.4 is an example of a TXT record informing about the paper format supported and a boolean value if the printer has the ability to staple pages together (if a field is present without a value it is a boolean set to true, if it is

[12] The lexicographically later authority section in the resource record wins. An example of conflict is given in [Ches 05]: *"sctibook.local. A 169.254.99.200"* is compared byte-wise to *"sctibook.local. A 169.254.200.50"* where *169.254.200.50* wins, since 200 is greater than 99.

[13] `http://www.dns-sd.org/`

[14] each key=value or only key (in case of a boolean) is preceded by the length in bytes of the string. For example on Figure 2.4 "paper=A4" is preceded by 0x08 since |p|a|p|e|r|=|A|4| is 8 bytes long, where each byte contains the ASCII (or UTF-8) codes for the indicated character (e.g. "p" is 0x70 in ASCII code)

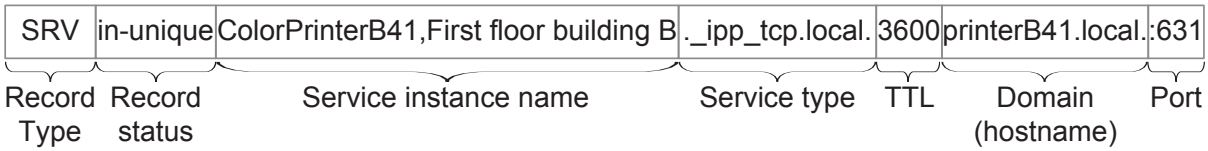


Figure 2.2: SRV record example.



Figure 2.3: PTR record example.

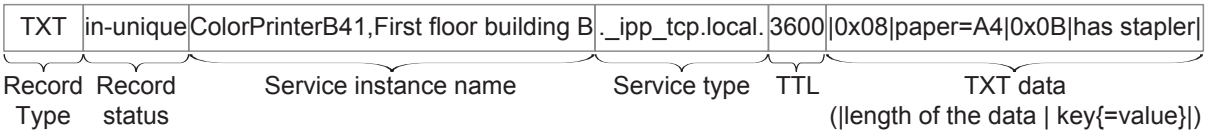


Figure 2.4: TXT record example.

not present the value is false).

Enhancements in Zeroconf

To reduce bandwidth consumption of Zeroconf, several mechanisms are proposed. First there is the Known Answer Suppression to reduce sending of redundant information. Multicast DNS queries contain a Known Answers list associated with a TTL. A responder does not respond to the query if its answer is already contained in the Known Answer list with a TTL at least half the correct value. Knowing this, a sender of a query does not include any Known Answer that has a TTL smaller than half the correct value[15]. Using multicast, messages are received by all the hosts that are in that multicast group. A host willing to transmit a query listens to passing messages, and if it detects a query containing the same question, it will not send the same query again. Likewise, a host hearing the same response that it was preparing to transmit does not transmit its redundant response.

Another enhancement, similar to having names that change less frequently than IP addresses, is the Late Binding. A user who queries for a service receives a list of service matching that query. The list is displayed to the user from which he can pick. Between the moment the query was sent and the actual selection and use of the service there time might pass. While the service might still be in the network, its IP address might have changed in between. Thus resolving the host's IP address at the last possible moment reduces the chances to obtain obsolete or invalid IP address.

2.3.4 Jini

Jini[Kuma 01], introduced in 1998 by Sun Microsystems, is a Java based technology that provides spontaneous discovery between services and users. Jini federates group of devices and software components into a single, dynamic distributed system. To do so, it uses 3 protocols: discovery, join and lookup. The communication between entities is done using RMI (Remote Method Invocation). Java objects are serialized and send over multicast IP (IANA reserved multicast address: 224.0.1.84 for jini announcement and 224.0.1.85 for jini request) or unicast IP. Jini's service interactions are presented in Appendix A.2.

[15] the sender knows the correct value from previously received records

2.4 Service discovery in ad hoc networks

Service discovery protocols for wired networks are, for most of them, not adapted to ad hoc networks. At the time of design and conception, they did not have to consider lightweight devices or lossy and limited network links. Therefore, many research was and is also done on ad hoc specific service discovery protocols. However, none of proposed protocols has gained a particular popularity or is close to standardization yet (in contrast with wired service discovery protocols). Following are ad hoc specific discovery protocols. Each of them was chosen because of its specific approach of the service discovery protocol.

2.4.1 Konark

Konark[Hela 03] is a service discovery protocol that uses XML based service descriptions and a multicast address for service announces. There are no directory nodes, every node is independent. It uses HTTP and SOAP to handle service delivery, requiring a micro-HTTP server on each device. Konark is similar to UPnP in terms of design choices. The use of XML is, from our point of view, not the best choice in such a performance constrained and resource poor environment as ad hoc networks.

2.4.2 Allia

Alliance-based service discovery for ad-hoc environments [Rats 02] is a policy-driven peer-to-peer caching based service discovery system. Service discovery relies on alliance formation between network nodes. Nodes locally form their own alliance by including other nodes to it (i.e. adding nodes to the local alliance list) while they might be added to local alliances by other nodes without any notification. Thus a node knows about its own alliance but it is not aware of other nodes' alliances in which it might be a member of. Nodes of the alliance are then used as directory nodes for service discovery. If a node is not reachable after a policy driven threshold[16], it is silently removed from the alliance. Node discovery is done by periodical advertisement messages broadcasted by every node. Service advertisement messages are only sent when the network neighborhood changes. They contain the description of the provided services of the node. Service description format is left open, xml or other formats can be used.

Nodes searching for a service first check their local cache and, if not succeeded, request the service to nodes of their alliance by sending a request message. Nodes receiving a service request can silently discard the message or check the local cache for a service matching the request. If no match was found, nodes forward the request to nodes in their own alliance. Service request navigate through multiple alliances until a match is found or the requests times out. In every decision done by a node, local policies are involved. Policies enable fine tuning of every step adapted to the user's preferences or device's capabilities.

Allia removes leader election and other problems associated with structured compound information. Every device is self-sufficient. In addition, devices with limited capabilities can run a lightweight version of the Allia protocol providing minimal functionalities to stay independent, running only one agent to run the platform It is composed of a lightweight yellow page service that registers services on the local device and a lightweight white pages that registers agents on the local device.

[16] Each device has a policy that reflects the device capabilities, user preferences, application specific settings, etc.

2.4.3 Service discovery over OLSR

Several service discovery protocols rely for the transport of their message on an underlying routing protocol. In [Jodr 06] and [Li 05] authors use the OLSR routing protocol to transport their messages by piggy-backing them into OLSR messages. Both papers propose to insert a new message type in OLSR containing service information that can benefit of the MPR mechanism to efficiently flood the network.

2.4.4 Service discovery using a field theoretic approach

In [Lend 05] authors use an approach using the analogy of an electrostatic field. The service are modeled by a "positive" point of charge and service request packets represent "negative" test charges attracted by the positive service charges. Nodes in the network calculate a potential value and route request packets (−) towards their neighbor with the highest potential, hence the services (+) (Figure 2.5). Nodes hosting service(s) announce their service(s) in the network by flooding the network. To reduce the overhead of this flooding, authors use a caching and aggregation mechanism that groups announces coming in on a certain period of time into one aggregated message. Since it was not the main goal of this work, authors did not try to use more optimized approaches.

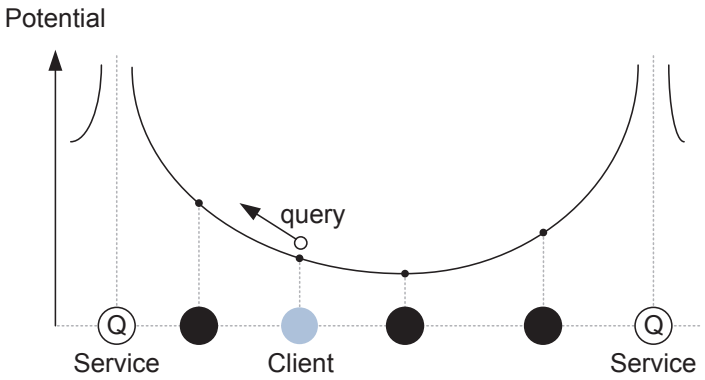


Figure 2.5: Service discovery with potentials - field approach.

Scalable Service Discovery for MANET: [Sail 05] is a distributed central-directories discovery architecture. Directories are responsible for caching the service descriptions, advertising their presence to nodes within their vicinity and handling their service requests by checking the local cache or forwarding the query to other directories. The election of the directories is done on the fly and the main election criterion is the node coverage. To exchange their directory profiles they use bloom filters and "bordercast" it in the two-hop neighborhood (using the MPR mechanism of OLSR, see Section 1.2.2). The selection of the directory nodes relies on the node coverage which can lead to problems since nearby passing node traversing the network can have a big node coverage at one particular moment but will, after being elected, disconnect shortly because of their mobility.

2.5 Zeroconf as service discovery protocol for ad hoc networks

Protocol specifically designed for ad hoc networks propose adapted techniques and algorithms, but lack, at the time of this work, in maturity and availability of exiting implementation. For those reason, we decided to use as service discovery protocols in our contributions the Zeroconf protocol. Zeroconf, is among the standardized service discovery protocols the most lightweight, modular and adaptive one. Zeroconf was not designed specifically for ad hoc networks, however, following are the reasons why using Zeroconf is a very good candidate for ad hoc networks:

- **Is based on DNS:** The approach taken by Zeroconf is to, rather than invent a new protocol with a new set of messages and structure, use a widely accepted standard. The properties provided by DNS already offer all the features needed for a service discovery protocol as described in [Ches 05]:
  - Service discovery requires an aggregation mechanism (server or directory nodes)
    DNS has servers to do this.
  - Service discovery requires a service registration protocol
    DNS provides Dynamic Update.
  - Service discovery requires a query protocol
    DNS already is a query protocol.
  - Service discovery requires a security mechanisms
    DNSSEC provides such security mechanisms.
- **Widely deployed:** Zeroconf, also due to its DNS roots, is already widely deployed in many devices and many systems have a standard implementation available. Therefore using Zeroconf in ad hoc networks removes the burden of developing a completely new protocol and, more importantly, making it widely accepted and deployed. Only adaptations or changes to ad hoc networks, for example as a plug-in, are necessary.
- **Benefits from an infrastructure:** Since Zeroconf is based on DNS, it can immediately take advantage of existing DNS servers. Most managed networks have a DNS server, DNS-SD's entries are compliant with standard DNS entries and can transparently be stored on the existing DNS server. While ad hoc networks, are infrastructure less networks, the possibility to interact transparently with an existing infrastructure, when it is available, is a great advantage for their acceptance and applicability in a real life implementation.
- **Easily adaptable to ad hoc networks:** As shown in our contributions in Section 7.1, only very few changes, if any, are necessary for Zeroconf to be adapted to ad hoc networks. The changes necessary are located at the underlying required multicast structure.
- **Extensible:** Zeroconf permits an easy addition of new features such as context information attached to a service (i.e. inside the TXT description) or at the protocol level by adapting the content of the queries (e.g. known answer suppression) or their responses with respect to the current context information. Another possible extension, is the introduction of directory nodes. Each extension is local, thus does not modify the message formats or influence implementations on other devices not implementing that extension, which makes it backwards compatible with standard Zeroconf.

2.5.1 Zeroconf and its multicast structure

As described in Section 2.3.3 Zeroconf is composed of three main parts, link local-addressing, mDNS and DNS-SD. In the Zeroconf modules itself, only very few changes need to be done.

In link-local addressing, the naming service and DNS-SD the only changes needed are at the configuration level. Since Zeroconf was designed for wired networks, the various timeouts and periods may have to be adapted to the dynamicity of ad hoc networks.

Zeroconf relies on an IP multicast structure. Multicast is a one-to-many message transmission technique that enables a single copy of a message to be delivered to multiple pre-defined receivers. Nodes willing to receive multicast messages subscribe to the multicast group corresponding to their interests. In multicast IP, on the client/host side, the subscription occurs on the local area network level (LAN) at the multicast router using the IGMP (Internet Group Management Protocol) protocol [Holb 06]. However, the core task is done by the multicast routers that construct the multicast structure. There are many different protocols that can basically be classified in, source-based tree structures or core-based tree structures. Source-based tree structures construct the multicast tree by flooding a multicast query into the network and each router interested (i.e. has received an IGMP join request from one of its hosts) in that multicast group responds with a join message. The initial flooding constructs the path to all the routers and the only routers that replied with a join response are kept in the multicast tree other routers are pruned from the tree. In core-based tree (CBT) structures, so-called Rendez-vous Point (RP) are selected to be the root of the multicast tree. Here, each router willing to join the multicast group sends a join request to the RP and thereby opens a path towards its hosts. Core-based tree structures have the advantage to regroup several multicast groups at the RPs, which reduces the multicast group management communication. However, the RPs also form a bottleneck on higher multicast data traffic loads. A detailed tutorial about IP multicast can be found in [Saha 02].

In wired networks, multicast is a good solution to limit the impact of a data-stream with multiple recipients by avoiding using several unicasts for a the same data. It is for example well adapted for applications like IPTV streaming. Zeroconf has such data as the various mDNS or DNS-SD announces. Furthermore, multicast is already widely deployed and used by other applications, therefore reusing it for Zeroconf, just as reusing the DNS structure, is taking advantage of the existing.

In ad hoc networks, building up a multicast structure similar to wired networks is not suitable because of the dynamicity that generates too much multicast structure maintenance packets. Not only do the continuous link changes require each time update messages to maintain the multicast structure, but also the lossy wireless medium and the unreliable intermediate nodes (as opposed to well managed multicast routers in wired networks) make it very difficult without a huge overhead to maintain such a structure or a loss of robustness and performances. A survey describing multicast protocols for ad hoc networks can be found in [Junh 09].

2.5.2 Why does Zeroconf use multicast?

As explained before there are two main tasks that need to be done in multicast: first, handle groups membership and second, forward data to all the members of a group. If we take a network with only Zeroconf running, the only purpose of using multicast is to reduce the complexity at Zeroconf level to be able to just send a message using the multicast group, and let the underlying multicast protocol/routers handle the dissemination of the message inside the group. Multicast is often already deployed in wired networks, so there is actually a need to handle different multicast groups for different protocols or applications needs.

In ad hoc networks, since multicast is very rarely deployed, group management is mostly useless, since only the application or protocol currently deployed, uses or would use the multicast structure. Moreover if we consider Zeroconf, the disseminated information is of interest of all the nodes in the network. Building a multicast group where nearly all the nodes of the network are participants, equals building an efficient structure that provides information dissemination

to all the nodes in the network. Moreover, one could also skip the group management part an simply send message on the multicast address, but this simply means flooding the information in the network, which is not a good option as explained in Section 1.2.

2.5.3 Conclusion

Zeroconf is a standardized and lightweight protocol, that provides the necessary mechanism and adaptability to be deployed in an ad hoc network. We discussed that the multicast structure, required by Zeroconf, is not the only challenge that is facing the use of Zeroconf in an ad hoc network. Chapter 7, will present and provide detailed description of how Zeroconf can be used in an ad hoc network and provide an adapted solution as replacement for this multicast structure using SLSF.

2.6 Context & Metrics

To enhance or optimize an application, a protocol or a service used by a node in a MANET, a good strategy is to take advantage of the context. **Context** can be defined as in [Dey 01]:

> *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."*

A situation can be characterized by many different types of information. To express this context information in computable information, so called metrics are used. Metrics are a set of measurements that quantify information. In other words, metrics are used to quantify a piece of context information that together form the nodes context. A protocol/service/application that takes into account context information is said to be context-aware. **Context-awareness** is defined in [Dey 01] as:

> *"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."*

Compared to classical wired networks, context in MANETs can be acquired from more diverse domains. Context of a MANET node can be characterized by the network as in wired networks but additionally also by the node information (e.g. power status, capacity), the geographical position. The main differing part from wired networks is the "human" entanglement. This "human" part widens drastically the context scope. One could even think of human context, emotions or mood to be taken into account if they where available. Two main challenges concerning context and context-awareness:

- Gathering context that is actually relevant and useful. One can gather a lot of information that sound interesting but does not actually help to enrich the context.
- The ability to gather a context is not straightforward. For example, human context needs to have special sensors placed on the user.

2.6.1 Social: Centrality

Structuring a network is determining which nodes plays a more important role in the network in order to manage, govern, steer or lighten the load for other nodes. A lot of research is already

done in the sociological research field about determining in a social network which nodes have more power due to their position in the network [Hann 05, Frie 91, Free 79, Bona 01]. Power is a fundamental property of social structures. Social network analysis was developed to study power, and the closely related concept of centrality. Using this network approach emphasizes the fact that power is inherently relational. A node itself has no power, but can have power if it can dominate others. Domination can occur in various ways: having a favored position, having more opportunities or fewer constrains. To be able to understand how domination occurs, 3 simple systems are considered in [Hann 05] (Figures 2.6, 2.7, 2.8).

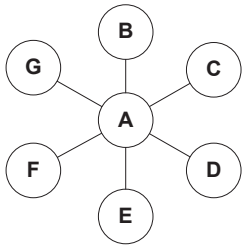


Figure 2.6: Star network.

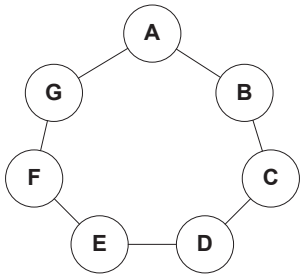


Figure 2.7: Circle network.

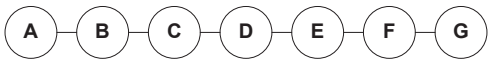


Figure 2.8: Line network.

Degree In Figure 2.6 node A has clearly a favored position in the star network. A has more opportunities and alternatives than other nodes. If one node decides to not communicate with A or to not provide A with a given resource, A still has the other nodes to exchange with. Whereas if A decides to not exchange with any other node in particular, this node will remain isolated without any exchange opportunity. The more connections a node has the more power it (may) have. Node A has a degree of 6 while all the others have a degree of 1. Nodes with a high degree are more powerful since they have more choices hence opportunities. A high degree make nodes less dependent of other nodes therefore more powerful.

In the circle network (Figure 2.7) each node have the same degree so all nodes are equally powerful.

In the line network (Figure 2.8) apparently all nodes except border nodes A and G have the same power. Nodes A and G have a degree of 1 while the other nodes have a degree of 2. However, at a closer look, one would expect that node D is more powerful than node B since it is more central to the structure. This shows that the measure of the degree alone is not always sufficient to determine the power of a node.

Closeness Power depends on direct relations and exchanges, like shown by the degree centrality. But it can also depend on how close a node is related to other nodes. If a node is close to other nodes it can be heard by more nodes. Being close to other nodes permits to be heard faster and also receive information faster. This closeness can be translated into power. In the star network (Figure 2.6) node A is at a distance of 1 hop to all the other nodes; each other node is at a distance of 2 from all other nodes (but A).

In the circle network (Figure 2.7) all nodes have a different distances to all other nodes, but the distribution of distances is the same. Therefore each node has the same power.

In the line network (Figure 2.8) node D has the most power since it is closer to all the other nodes than are the set C,E, the set B,F and the set A,G. Border nodes A and G are again at disadvantage.

Betweenness The third reason that node A is advantaged in the star network (Figure 2.6) is because node A lies between each other pairs of nodes, and no other node lies between A and other nodes. A can directly communicate with any other node, while nodes B to F need to communicate via A to contact another nodes. Hence A has the ability to decide on the communications of the other nodes and can isolate nodes from the network. Betweenness is the structural advantage of being on the communication path between two nodes.

In the circle network (Figure 2.7) each node lies between any other pair of nodes. Actually, there are two pathways connecting each pair of nodes, and each third node lies on one, but not on the other of them. Again, all nodes are equally advantaged or disadvantaged.

In the line network (Figure 2.8) border nodes are again disadvantaged since they do not lie between any pair of nodes. Similar to the closeness centrality node D lies between the most pair of nodes therefore has the highest betweenness centrality.

Extensions Based on these 3 centrality approaches namely degree, closeness and betweenness, several extensions have been proposed. For example Phillip Bonacich proposed a modification of the degree centrality approach that has been widely accepted as superior to the original measure. Bonacich argued that having the same degree does not necessarily make nodes equally important. Suppose two nodes A and B have the same degree, however A's neighbors have a very small degree, hence few neighbors while B's neighbors have a high degree. Who is more central? One would probably agree that B is, because the nodes it is connected to are better connected than A's neighbors. Bonacich argued that one's centrality is a function of how many connections one has, and how many connections the nodes in the neighborhood had. Bonacich questioned the idea that more central nodes are more likely to be powerful nodes. If we compare A and B. B is certainly more central, but not necessarily more powerful. Bonacich's argument is that since B has better connected neighbors, they are less dependent on B, so it has less power. The other way around, node A has poorly connected neighbors. A's decision to communicate or exchange affects more its neighbors hence A gains on power. Bonacich proposed that both centrality and power were a function of the connections of the nodes in one's neighborhood. The more connections the nodes in your neighborhood have, the more central you are. The fewer the connections the nodes in your neighborhood, the more powerful you are.

Conclusion Centrality and more generally social position of a node in a network can be a good metric to obtain the best positioned nodes of a network. In the context of this thesis, this metric can be used to help determine the most important node, such as in a clustering algorithm to select the clusterhead.

2.6.2 Collaborative Filtering

Dealing with information becomes an important issue when such a diverse and large quantity of information as in ad hoc networks is available. The quantity is not the only issue; a very important goal is also the quality of information; present the right information at the right time to the user. Recommender systems using collaborative filtering (CF) are a well-established technique to overcome this problem of information overload by recommending information items based on taste of like-minded users [Sarw 01]. If we take the example of service discovery, the service-responses to a query contain already a filtered part of the available services matching the

initial query. Collaborative filtering goes a step further and filters out of that response the best services, collaboratively selected by users with similar taste, based on the user's current context and profile. Most CF systems can be classified into user-based CF and item-based CF.

In **user-based CF**, most systems can be reduced in two steps:

- Look for like-minded users (i.e. users that have a similar profile or rating pattern).
- Compute a prediction based on the ratings collected in step 1.

Here the main exchange criterion is the similarity between the users, either based on their profile preferences or based on their similarity on rating the same items with the same values.

In **item-based CF** [Sarw 01], unlike user-based, only items and their ratings are compared for the prediction. The prediction process can be described in three steps:

- From the set the target user has rated, compute the similarity to the target item i (i.e. item on which the rating prediction is done).
- Select the k most similar items $\{i_1, i_2, \ldots, i_k\}$ and also compute their respective similarities $\{s_i1, s_i2, \ldots, s_ik\}$.
- Compute the prediction with a weighted average of the user's rating on those most similar items.

To avoid active user intervention, rating/ranking of items can be based on implicit observations of the user's behavior. Behaviors like which items where consumed at which time in which quantities or durations can compose the item ratings.

Currently predominant systems using CF mostly depend on huge centralized databases to store user preferences and furthermore are only available online. Different from traditional static devices, mobile devices often have no always-available cellular connection (if at all) to the Internet. Furthermore, compared to the often and freely available wireless communication, such cellular connection cause more cost and provides lesser bandwidth. Following are some existing research work about CF based recommender systems in mobile environments.

In [Cost 05] an incremental collaborative Filtering algorithm for applications, where users are occasionally connected to a central server, is introduced. The general idea is to store a subset of selected user profiles, together with a ranked list of predictions. When the user is offline, a service on the local device can still recommend items based on the previous ratings made the last time the user was connected. Each time the user supplies new ratings, the list of predictions will be recomputed, even if the user is not connected to the server. In the case that a user encounters another user, the authors suggest that they exchange their profiles and recalculate their prediction lists. The past influence of the other user should be removed from all predictions and the new influences should be added. At last this case is not evaluated or considered any further in the paper and is a part of future work.

An approach to collaborative filtering in a mobile tourist information system for visitors of a festival, based on spatio-temporal proximity in social contexts is proposed in [De S 06]. This new approach is based on the idea that users who go to the same place at the same time tend to have similar tastes. In order to keep track about the visited places each user is equipped with a portable computer coupled with a GPS unit. Furthermore, a central server provides a database with information about all the events, restaurants, venues and bars at the festival [Belo 05]. The proposed approach uses a user-based CF technique and calculates similar users via a spatio-temporal proximity measure, e.g. two users are considered as similar if they consume

the same items simultaneously; the ratings are implicitly observed. The following exchange of rating information between such similar users is done via an ad-hoc peer-to-peer interaction. However, the defined similarity measure has one drawback. Users consuming the same periodic event at different times still share interests, but are not considered as similar. In a future work, the authors intend to investigate how their CF approach can be extended in order to exchange ratings between users in spatial but not temporal proximity. Furthermore, they want to evaluate the introduced CF system at the Edinburgh Fringe festival.

In [Schi 08] an approach based on epidemic spreading of user preferences is presented and also evaluated. In their study the authors show how to reach a prediction accuracy in a mobile ad hoc scenario that is comparable to the prediction accuracy obtained in a global knowledge scenario. However, the presented evaluation lacks from several shortcomings. Firstly, no realistic mobility model has been used, but an idealized data exchange pattern with disjoint pairs per iterations. Furthermore, typical problems of mobile ad hoc networks, like unreliable connections affected by interferences or collision are not considered.

Conclusion We consider large networks with numerous nodes and each node potentially hosting one ore more services. In this context collaborative filtering techniques can be applied at the service and service discovery level to reduce the overwhelming number of services proposed or displayed at the end user.

2.7 Conclusion

This chapter presents service discovery and the existing protocols in wired and ad hoc networks. The service discovery protocol chosen for our contributions is Zeroconf. While Zeroconf was not specifically designed for ad hoc networks, it has the advantage of being lightweight, standardized, widely used and deployed, and requiring few adaptations, even none at the conceptual level, to adapt to ad hoc networks. The specifically ad hoc designed solutions are promising but lack in acceptance of the community, available implementation and more importantly detailed description.

Finally, this chapter described context and metrics available and applicable to ad hoc networks that we could later on use to improve service discovery.

Part II

Contributions

The first part described the main domains addressed in this thesis. Ad hoc networks, routing protocols, service discovery, context awareness and collaborative filtering where introduced.

This part relies on the presented various domains and proposes contributions to each of those domains. The contributions, nevertheless contribute to the main goal: service discovery in ad hoc networks.

The first chapter of this part, Chapter 3, considers dissemination in ad hoc networks and proposes the SLSF (Stable Linked Structure Flooding) protocol that relies on a stable one-hop cluster structure and thereby provides scalable and efficient dissemination by selecting inter-cluster nodes wisely. Service discovery needs dissemination to announce services.

Chapter 4 adds routing capabilities to the dissemination protocol SLSF. The proposed routing protocol is the Stable Linked Structure Routing (SLSR) protocol, a cross-layer routing protocol that adapts its efforts by being aware of the underlying and overlying layers. After the service discovery, a node must be able to reach the desired service.

Chapter 5 introduces a distributed framework that combines the use of reference models and simulators coming from the network and sociology domains to provide a more realistic simulation. The more complex scenarios to express the usage of services and its discovery, can be expressed using this framework.

The fourth chapter of this part, Chapter 6, considers the overload of information that can occur with the growth of ad hoc networks and also with the multiplicity of contents exchanged on such networks. Collaborative filtering deals with the issue of content overload and filtering. Service discovery can profit from such filtering with the growth and multiplicity of announced services.

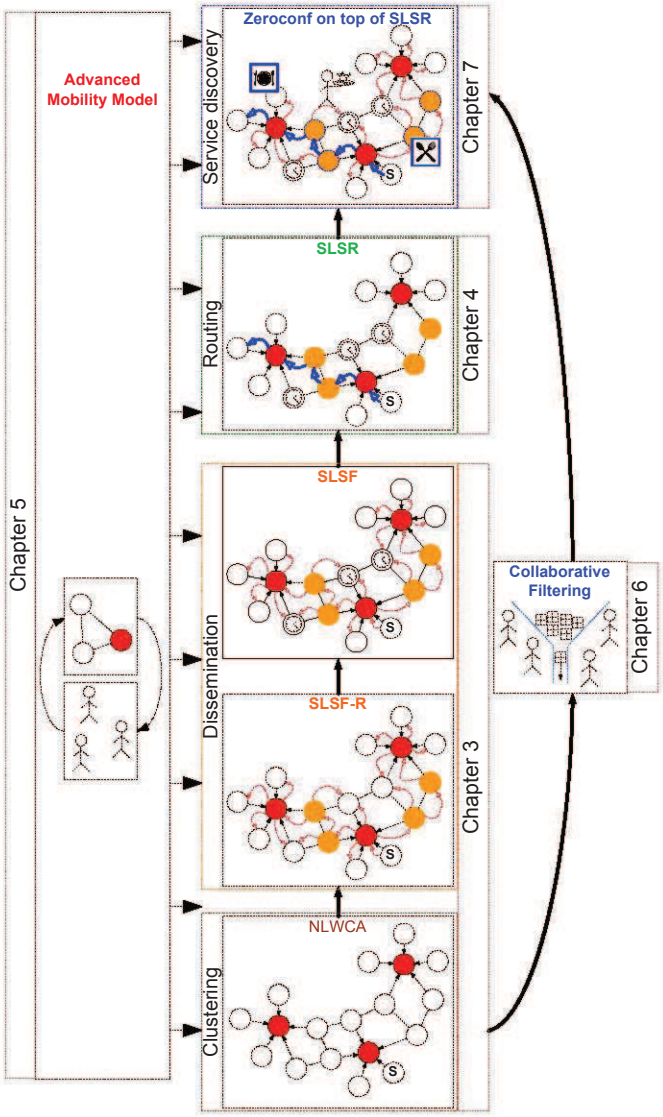


Figure 2.9: Representation of the contributions of this thesis.

Finally, Chapter 7, shows how the previous contributions converge and cooperate to improve service discovery and allow context-awareness. This chapter proposes to replace the multicast in Zeroconf by the SLSF/SLSR structure. The chapter also presents a complete service discovery architecture proposed in the context of the ANR SARAH project where our simulation framework providing more realistic scenarios is applied. To finish this chapter, a set of real world experiments are described.

Chapter 3

Dissemination: SLSF - Stable Linked Structure Flooding

Contents

3.1 Big picture: NLWCA - WCPD - SLSF-R - SLSF 48
3.2 Structure comparison 50
3.2.1 WCPD vs. OLSR 50
3.2.2 Experiments and results 50
3.2.3 Conclusion 52
3.3 Avoid Subheads in NLWCA 53
3.3.1 Problems 54
3.3.2 Solution 55
3.4 Basic SLSF 58
3.4.1 IRC – Inter-Cluster Relays 58
3.4.2 SLSF - Broadcast 62
3.4.3 Basic SLSF message Header format 62
3.4.4 IP fragmentation and SLSF 62
3.5 Full SLSF with Fault-recovery 64
3.5.1 Acknowledgment mechanism 64
3.5.2 Delayed transmission mechanism 66
3.5.3 Fault Recovery Header format 67
3.6 Experiments and Results 68
3.7 Conclusion 69

This chapter presents the Stable Linked Structure Flooding protocol (SLSF), that relies on a cluster structure, which goal is to improve dissemination of messages in an ad hoc network.

The most basic dissemination is plain flooding where each node forwards the message to its neighbors. However this behavior leads to the well-known broadcast storm problem where nodes endlessly broadcast a packet [Ni 99]. To avoid this, broadcast schemes as shown in Section 1.2 limit the number of forwarding nodes by structuring or designating the forwarding nodes. More generally, an optimal dissemination scheme would be one where:

- Each node in the network receives the message,

- The message is forwarded only by the minimal set of nodes (i.e. the minimal connected dominating set, Section 1.2.5)

However, to reach this optimal dissemination scheme in an ad hoc network, one would need to obtain first, the complete network topology, and then, compute the minimal connected dominating set. Once computed, the roles are assigned to the corresponding nodes by sending them message.

However, this would consume too much bandwidth and the forwarding-nodes set may not be relevant anymore between the moment the network topology was acquired and the moment the forwarding information was handout, due to the dynamic nature of ad hoc networks. As shown in Section 1.2.5, many research has been done in the field of distributed dominating sets. The work presented in this chapter, is related to (connected) dominating sets and therefore could also be expressed in graph theory terms. However, it will be described from a clustering point of view.

In an ad hoc network the most reliable information is the one that is nearby or locally available. Therefore, rather than focusing on the whole network at once, we propose to locally optimize dissemination in small groups linked together. Our solution, named SLSF, relies on local groups of one-hop stable-connected devices in a self-organizing manner. SLSF aims at discovering stable connections between groups, thus creating bigger stable-linked network structures. We exploit the stable-linked structures within the network topology to streamline information exchange and to minimize the overhead. The local one-hop groups are built using the NLWCA clustering protocol (Section 1.2.4). Like in WCPD (Section 1.2.4), specific beacon formats are used to detect nearby stable-connected clusters. Furthermore, to create bigger stable-connected structures we propose our SLSF protocol in Section 3.4. Finally, to add robustness, a fault recovery protocol is employed to compensate for local intermittent node failures in Section 3.5. Fault recovery can be selectively enabled or disabled on a per-packet basis (more about this in Section 3.5.3).

A part of the contributions in this chapter were done in collaboration with Adrian Andronache and Steffen Rothkugel.

3.1 Big picture: NLWCA - WCPD - SLSF-R - SLSF

This sections gives a short summary to give a clear vision of how the different protocols interact and why NLWCA, but also WCPD, are related to SLSF-R (SLSF minus Recovery) and SLSF. While SLSF and SLSF-R are detailed only later (Section 3.4), this clarification can be useful throughout the following sections to avoid mixing up the different protocols and their purposes.

- NLWCA: is a one-hop clustering algorithm. It builds one hop clusters based on link and node weights. No broadcast or dissemination mechanism is provided.

- WCPD: provides this missing dissemination mechanism to NLWCA. Basically, a disseminated message is one-hop-broadcasted inside the clusters by the clusterheads and multi-hop-unicasted by the clusterheads to the nearby clusterheads.

- SLSF-R: SLSF has two versions a complete one and a lighter version without the fault-recovery mechanism. Therefore SLSF-R is SLSF **minus** Recovery, thus the lighter version. SLSF-R is a replacement for WCPD. The dissemination in SLSF-R occurs only using broadcasts and the inter-cluster nodes forwarding messages are chosen wisely to obtain a more efficient dissemination. SLSF-R and WCPD share exactly the same beacon structure, both

have the same bandwidth usage in terms of structure building [17]. Note that result figures presented later might only display one of both acronyms (WCPD or SLSF-R) when speaking of their structure bandwidth consumption for the sake of readability. The difference between WCPD and SLSF-R is at the dissemination level and in the way the information, gathered using the beacons, is better taken advantage of.

- SLSF: adds a Fault-recovery mechanism to SLSF-R that is only active on failure detection. The dissemination in SLSF occurs as in SLSF-R (except in case of delivery failures). The beacon format has additional sequence-number fields and differs from SLSF-R (thus also WCPD).

Figure 3.1 shows the relation between the protocols from a dissemination point of view. On Figure 3.2 the beacon formats of the four protocols are depicted.

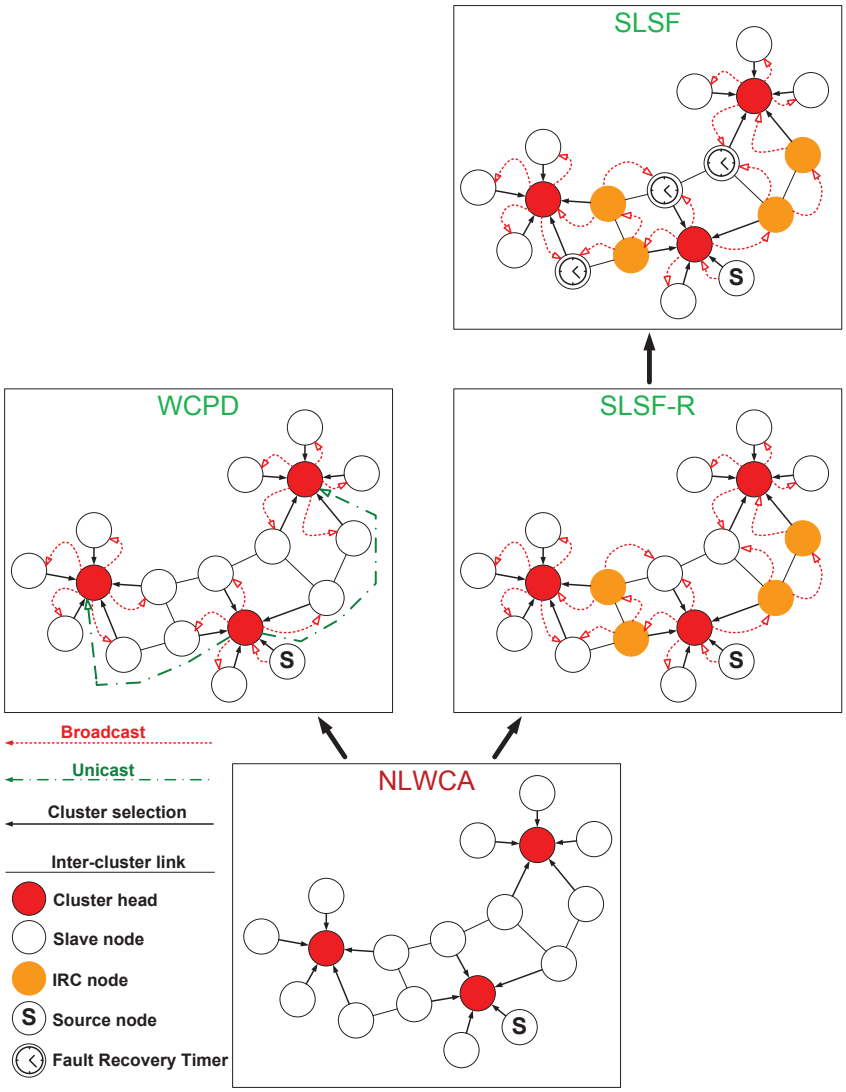


Figure 3.1: Protocol hierarchy of NLWCA - WCPD - SLSF-R and SLSF: Dissemination perspective.

[17] the bandwidth is the same only to set up the cluster structure, but in terms of dissemination, the bandwidth between WCPD and SLSF-R is different

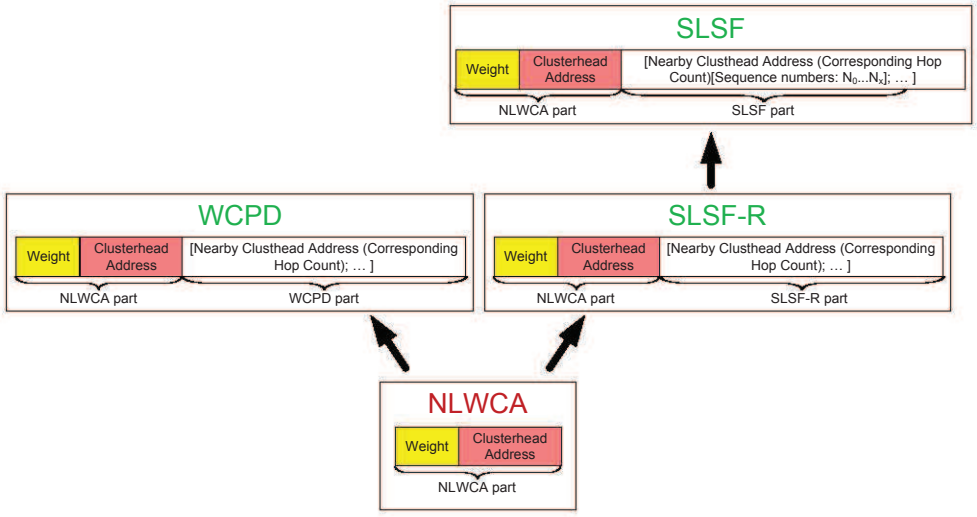


Figure 3.2: Protocol hierarchy of NLWCA - WCPD - SLSF-R and SLSF: Beacon format perspective.

3.2 Structure comparison

3.2.1 WCPD vs. OLSR

This section compares the dissemination performances of two existing protocols, OLSR against WCPD. We chose OLSR as a reference for comparison, because it optimizes the number of 1-hop distant nodes needed as forwarders to reach the 2-hop neighbors. WCPD instead is especially lightweight and is based on one-hop clusters built by NLWCA and discovers nearby (adjacent) clusters, using beacon information.

As our comparison relies on the information dissemination of both OLSR and WCPD, we furthermore compare both message dissemination mechanisms [Lecl 08, Lecl 09]. When following the flow of a disseminating message, the topological structures, tree and star shaped, of both protocols are highlighted. With OLSR, dissemination occurs along the route formed by MPR nodes (Figure 3.3). A broadcast message sent from a source traverses the network by being forwarded only by MPR nodes and the messages reaches every node in the network. With WCPD, a message from a source S (in this case a slave node) is first sent to its clusterhead B (Figure 3.4). Clusterhead B then one-hop broadcasts the messages to its slaves and multi-hop unicasts it to the nearby clusterheads A and C. Upon receiving the messages, clusterheads A and C start the same procedure; Send a broadcast to its slaves and send unicasts to nearby clusters (omitting the source cluster). Therefore, eventually, every node in the network receives the message.

3.2.2 Experiments and results

For the conducted experiments we used the Restricted Random Way Point mobility model [Blav 02], whereby the devices move along defined streets on the map of Luxembourg City for 5 minutes (Figure 3.5). For each device the speed was randomly varied between [0.5;1.5] units/s. At simulation startup, the devices are positioned at random selected crossroads and the movement to other crossroads is determined by the given random distribution seed. For the experiments a number of 10 different random distribution seeds were used in order to feature results from different topologies and movement setups.

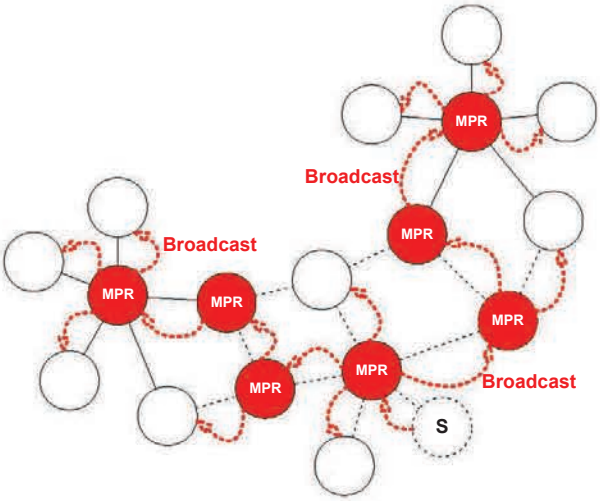


Figure 3.3: OLSR message dissemination through a network.

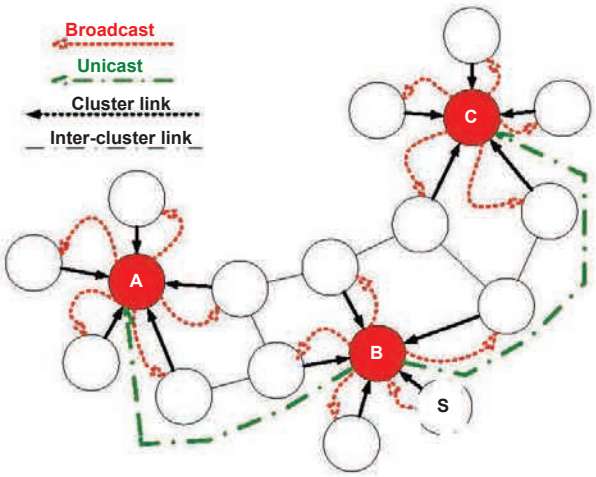


Figure 3.4: WCPD message dissemination through a network.

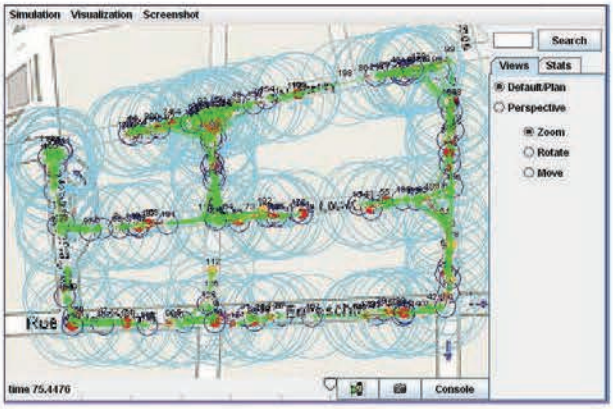


Figure 3.5: JANE simulating the protocols on 100 devices. The mobile devices move on the streets of the Luxembourg City map. The devices move with a speed of 0.5 - 1.5 m/s.

For the used mobile environment where nodes move with low speeds between 1.8 and 5.4 km/h the NLWCA link-stability threshold is set on 2 [Andr 08b]. Simulations were done to determine both the used bandwidth in order to build the topologies and the information dissemination performance of broadcasting on top of the two constructed topologies.

To build the MPR topology, OLSR exchanges the sets of one-hop neighbor nodes with every node in the communication range. Similar to OLSR, WCPD use the beacon to exchange the list of the discovered nearby-clusterheads with the one-hop neighbor nodes. To find out the network load produced during this phase, the size of both the one-hop neighbor sets and the size of discovered clusterheads were tracked every second of the simulation. In order to monitor the information dissemination performance and network load of the broadcasting mechanisms, a node was chosen to broadcast a message every 10 seconds during different simulation runs using different distribution seeds. The number of sent messages (i.e. broadcasts and unicasts) during the dissemination and the number of reached network nodes were tracked.

Results

The results in figure 3.6 are illustrating the size of the exchanged node-ID lists at the respective points in the timeline.

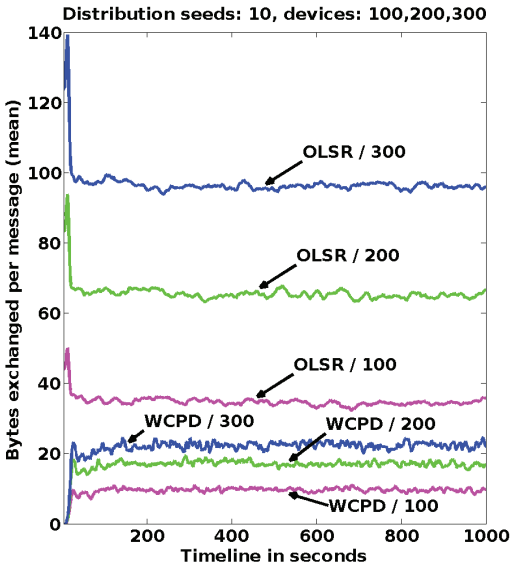


Figure 3.6: Bandwidth used in order to build the topology for 100, 200 and 300 nodes.

To calculate the bandwidth used by the protocol, one needs to take into consideration the time interval used to periodically send the exchange messages (i.e. hello messages or beacons) and the size of the used node IDs (e.g. 32 bits for IPv4 addresses) plus headers.

The results illustrated in figure 3.6 show that OLSR uses a higher bandwidth in both sparse and dense networks. This situation was expected since OLSR is exchanging the set of one-hop neighbors needed for the MPR nodes election.

In contrast to OLSR, WCPD only exchange the set of local discovered nearby clusterhead and sub-heads in order to discover stable paths between clusters in the network vicinity. The NLWCA protocol elects one clusterhead/sub-head in each one-hop neighborhood, which means that the number of clusterheads is a fractional amount of the number of nodes in the network.

The tracking results regarding the message dissemination performance and network load of the broadcasting protocols are presented in figure 3.7. The overall results show that the broadcasting on top of the OLSR topology performs much better in terms of message dissemination than on top of the WCPD topology. The denser the network, the higher is the difference between both the number of sent messages and the number of receiver nodes.

3.2.3 Conclusion

The simulation results show that OLSR highly outperforms WCPD in terms of broadcast receivers. On the other side, the network load produced by OLSR to build the topology is much higher compared to the one of WCPD. The OLSR dissemination approach has the advantage of reaching a much higher number of nodes than WCPD but at the cost of high network overload for the topology maintenance.

Based on this result we propose to combine the two protocols in a synergetic way by building clusters of stable connected nodes and using on top of the cluster topology an MPR-inspired

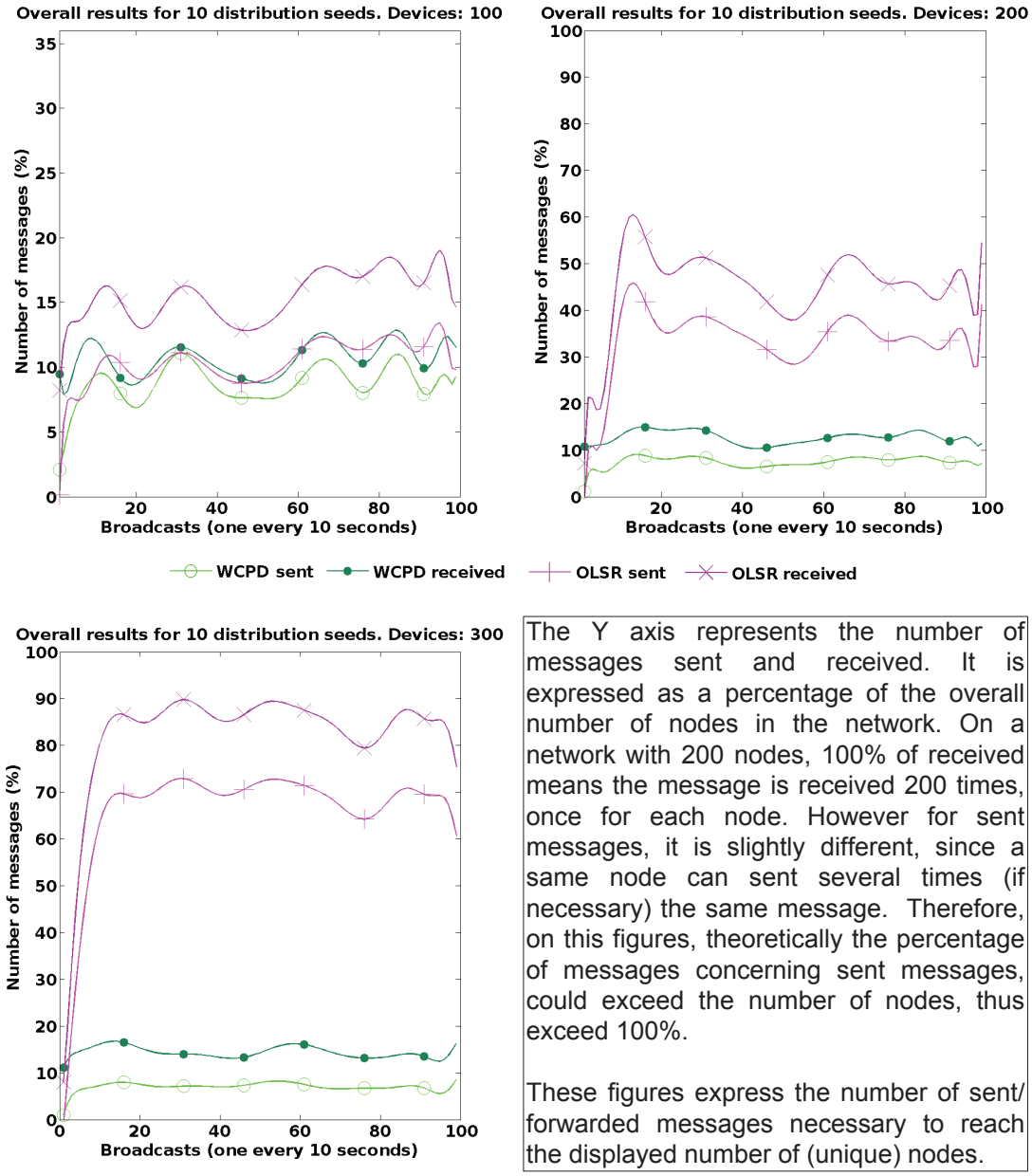


Figure 3.7: Overall number of sent/forwarded messages and received for 100, 200 and 300 nodes. (smoothed with a polynomial equation of the 16th grade for visibility sake).

mechanism. Thus, a better inter-cluster path discovery and loop-free broadcasting mechanism is provided at a low network cost used for topology maintenance. This will enable the service discovery protocol to take advantage of stable paths to service hosts in the vicinity and at the same time to reach a high number of network nodes by broadcast.

3.3 Avoid Subheads in NLWCA

Before presenting the SLSF protocol, a small but important change is proposed to be done to the NLWCA protocol. During the early design phase of SLSF we noticed an issue about NLWCA

and its cluster formations. In NLWCA, the elected clusterhead is always, among the stable neighbors, the node with the highest weight. Thus a node that designates a node as clusterhead can itself be elected by another neighbor node as clusterhead. A node that elects a neighbor node as clusterhead but at the same time is elected as clusterhead by some other one-hop neighbor node is called a subhead (3.8). Following sections describe the problems induced by subheads but also the solution to avoid them.

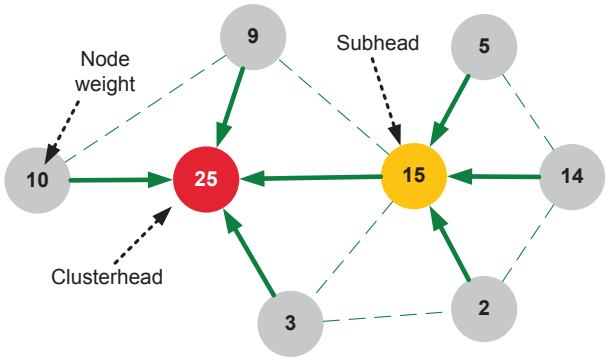


Figure 3.8: NLWCA - Node with weight 25 (Clusterhead) is designated by node with weight 15 (Subhead) which itself is designated by node with weight 14 (Slave).

3.3.1 Problems

While subheads are not an issue in NLWCA or WCPD, they can lead to several complications when using NLWCA as basis for another protocol (i.e. SLSF).

Subhead chains

Formations of subheads can lead to topological chains (Figure 3.9). Subhead chains can lead to a long multi-hop path inside one cluster. Long chains inside a single cluster increase the complexity of information exchange. A subhead at the end of the chain has to synchronize with its clusterhead on the other side of the chain. The scalability advantage of clusters can be lost if long chains of subheads occur.

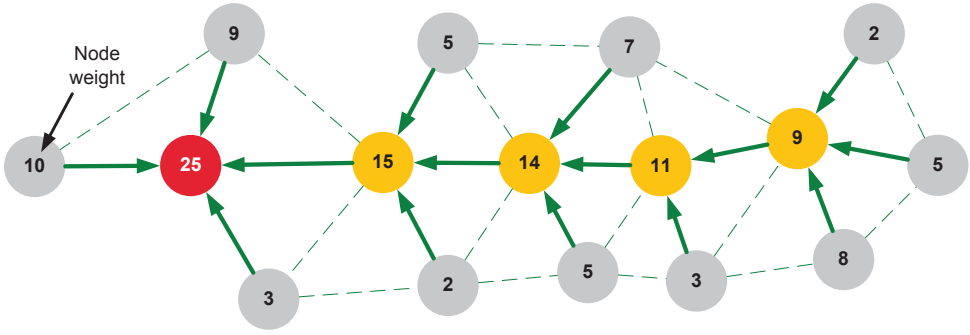


Figure 3.9: NLWCA cluster with a sub-head chain.

Cluster identity

Even when only one subhead exists in the network, when the identity of a cluster matters, problems occur. As shown on Figure 3.10, without subhead, all the nodes in the same cluster claim the same clusterhead, thus the identity of a cluster is the same from all the points of view. With subheads however, Figure 3.11, nodes 9 and 14 do not claim the same clusterhead while they are in the same cluster. Different points of view are here possible depending on which slave node is asked for its clusterhead. This can lead to problems when routing tables or any use of identity of a cluster is done since 9 and 14 would announce different clusterheads while they are actually in the same cluster. A solution for that would be to exchange messages so that 14 knows that the main clusterhead is 25 and not 15. However, besides requiring to update correctly the information for all cluster participants, message exchange will intensify with longer subhead chains.

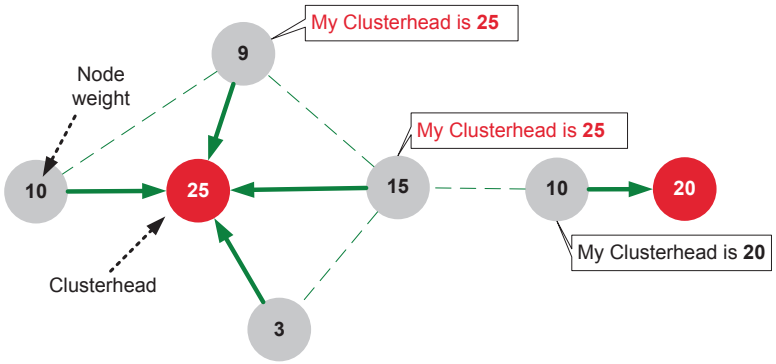


Figure 3.10: NLWCA cluster without a sub-head. All nodes in the same cluster claim the same clusterhead.

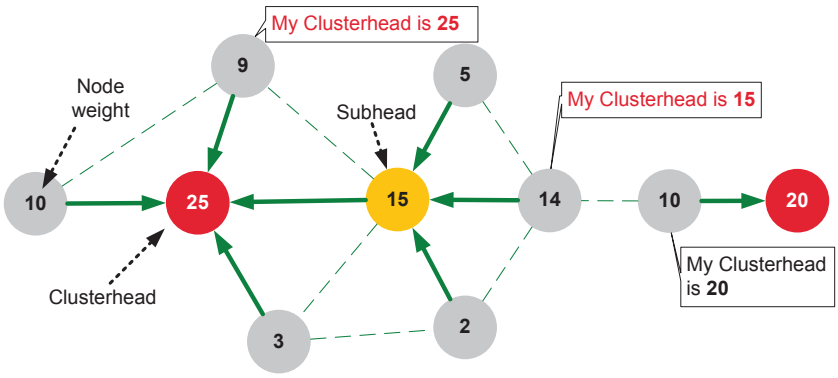


Figure 3.11: NLWCA cluster with a sub-head. Nodes 9 and 14 do not claim the same clusterhead while they are in the same cluster.

3.3.2 Solution

To avoid subheads from occurring we propose an additional simple rule to the NLWCA clusterhead election [Grat 09]: a node that already elected a foreign node as clusterhead is not eligible

to be elected by another node as clusterhead. As a result if a node would elect the highest neighbor node which already elected a foreign clusterhead, the next highest neighbor node that has not elected a foreign clusterhead is elected as clusterhead. For illustration with the subhead prevention rule the big cluster formed in Figure 3.9 would result in 3 clusters (nodes with weight 25, 14 and 9) shown in Figure 3.12. Here, node 3 cannot select node 11 as clusterhead and therefore picks the next highest neighbor which is node 9.

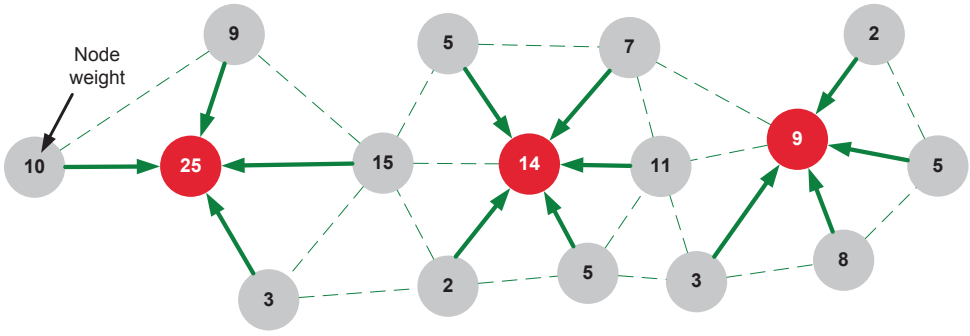


Figure 3.12: NLWCA cluster without sub-heads.

Simple Management

By avoiding subheads, a new characteristic appears in NLWCA: The clusters are now strict one-hop clusters. This will ensure that:

- the clusterhead has all the information available at one-hop distance concerning its cluster-members.
- there is no need to synchronize inner-cluster related information with other clusterheads or subheads.
- there is one single identity for the cluster. Each cluster-member announces the same identity, the same clusterhead.
- one single broadcast emitted by the clusterhead reaches all cluster-members.
- cluster members can reach the clusterhead in one hop.

Topology restrictions

Another new characteristic of NLWCA is that possible configurations between two nearby clusters ("neighbor" clusters) are very restricted. As shown on Figure 3.13 the maximum hop-count between two nearby clusterheads is three. This, by design, topology restriction has the advantage to bound and limit scopes of protocols. Thus, this simplifies protocol design when using NLWCA as underlying cluster topology. It also helps when considering timeouts and time to live (TTL) for messages (Section 3.5).

Drawbacks

The subhead avoidance does not have obvious drawbacks such as consuming more bandwidth or needing new messages to be exchanged since it is a simple rule introduced locally during

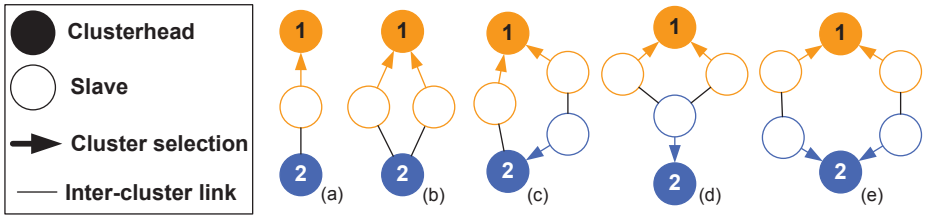


Figure 3.13: Inter-cluster configuration examples where 1 and 2 are clusterheads.

clusterhead selection. However, subhead avoidance induces topology changes that can represent drawbacks depending on situations:

- **Isolated clusterheads:** For example, Figure 3.14 shows the isolation of node D in a new cluster 4 while it was inside cluster 2 before subheads where avoided. Note that node A is also "kicked out" of cluster 2 but therefore joins cluster 1. Isolated clusters increase the overall number of clusters in the network. This increases the number of entries contained in a routing or state table, however the problems induced by subheads presented in Section 3.3.1 yet still make avoidance of subheads more viable.

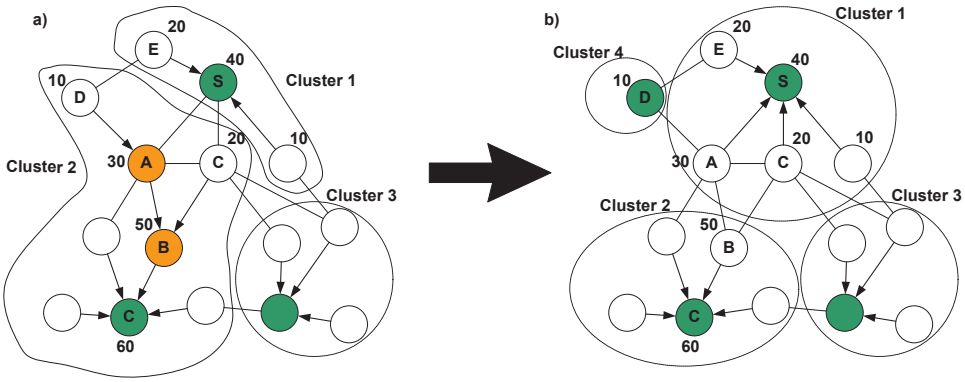


Figure 3.14: Same topology a) with subheads and b) without subheads. Node D remains isolated in a one-node cluster in b).

- **Clusterhead chain switching:** In certain cases, in the presence of (possible) subhead chains, adding a node with a higher weight to the cluster triggers a chain reaction of cluster switching. Compare for example Figure 3.9 with Figure 3.15, when subheads are present only node 25 and some of its slaves are affected by the venue of node 32. Whereas, if we compare Figure 3.12 with Figure 3.16, the venue of node 32 in the network affects, in a chain reaction, all the nodes in the network. Node 2 remains isolated, as explained in the previous paragraph. At first, here the subhead chain seems to be at its advantage since only the first node in the chain is affected, but actually if identity matters (Section 3.3.1) all the nodes will be affected, since the cluster changed its main identity (its clusterhead). Nevertheless, this clusterhead switching is still undesirable. Strategies to avoid this could be related to a node-weight adaptation. The first sketch would be: the longer a node is clusterhead the stronger it becomes and increases its node-weight, thus stopping the clusterhead chain switching.

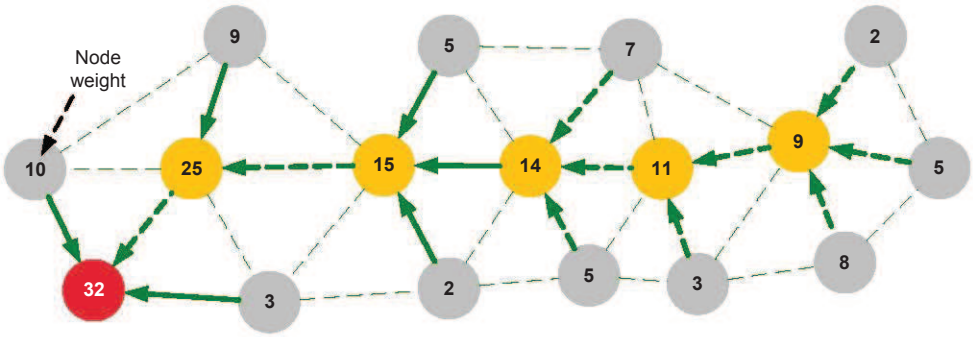


Figure 3.15: NLWCA cluster with a subhead chain. Adding a higher weighted node in the chain.

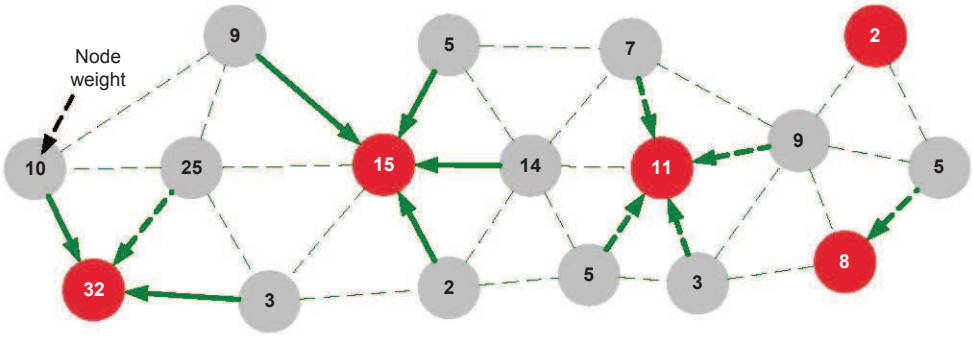


Figure 3.16: NLWCA cluster without subheads. Adding a higher weighted node in the avoided-chain.

3.4 Basic SLSF

NLWCA and WCPD provide a stable-connected cluster architecture. However as shown in Section 3.2 the broadcasting algorithm of WCPD needs improvement on dissemination performances. Our SLSF (Stable Linked Structure Flooding) protocol [Lecl 10b, Lecl 10c] replaces the inefficient broadcasting mechanism of WCPD with the Inter-Cluster Relay (ICR) mechanism. SLSF combines the advantages of all the protocols NLWCA, WCPD and OLSR: scalability, stability, reachability, while keeping the drawbacks low (i.e. the bandwidth usage). SLSF forms a first level of hierarchy by building a dominating set using the cluster algorithm NLWCA. Considering the dominant nodes of the underlying level (NLWCA), SLSF then forms a locally-optimal connected dominating set with the ICR mechanism.

The first level reduces the network to its dominant nodes and the second level insures local shortest-path connectivity and minimal relay nodes among dominant nodes of the first level.

3.4.1 IRC – Inter-Cluster Relays

Multiple paths to reach a given CH requires choosing one path rather than another. We use a next-hop selection inspired by the MultiPoint Relay (MPR) mechanism of OLSR to select the forwarding neighbors. We name Inter-Cluster-Relays (ICR) the nodes selected as next-hops. The goal of ICR is to reach all nearby CHs with the minimal set of 1-hop neighbors while optimizing the hop-count. The ICR nodes are calculated by selecting the smallest one-hop neighborhood set (directly connected nodes) needed to reach every nearby CH. ICR selection remains straightforward because the possible inter-cluster configurations are restricted by the

underlying one-hop cluster topology (Section 3.3.2).

SLSF, on top of NLWCA, discovers the nearby clusters (similar to WCPD) by reading the neighbor beacons. The improvement and novelty relies on the ICR selection which avoids superfluous network communication overhead without any additional message exchange. SLSF keeps the last beacon of every one-hop neighbor in cache. Hence every node has the following information locally available about each stable 1-hop neighbor: its weigth, its CH ID, the ID set of discovered CHs and their respective path length. ICR selection occurs as follows:

i. Select as ICR, neighbors that are the only one reaching a given nearby CH.

ii. Remove the now covered clusters from the list.

iii. Remove for every neighbor from the announced CH-list the entries with a worse hop count than the best one (i.e. keep only shortest path entries).

1. Calculate the cluster reachability for every one-hop neighbor (i.e. number of foreign CHs the neighbor announces in its beacon).
2. Select the neighbor with the best reachability.
3. Else if equivalent: select the neighbor with the highest weight.
4. Else if equivalent: select the node with the biggest IP address.
5. Remove the now covered clusters from the list.
6. While there is a not-covered CH, go back to 1.

Duplicate messages in SLSF:
A message is considered, by a given recipient, as a duplicate message if the message has the same source address and the same source sequence number. Duplicate information is kept in memory for a given period of time.

3-hop Inter-cluster case

NLWCA builds one-hop clusters, thus it permits up to three hops (two slave nodes) between CHs. ICR selection with two hops (one slave node) between CHs (Figure 3.17b) is a normal selection by the CH, however an additional hop (Figure 3.17d) requires additional attention.

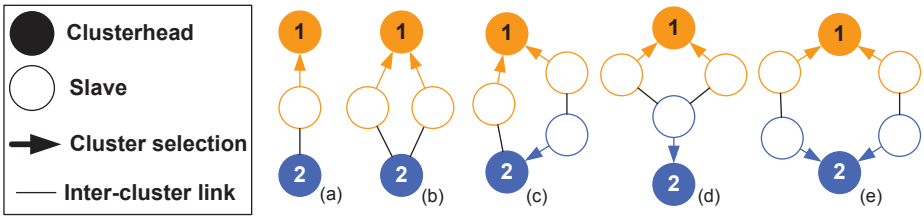


Figure 3.17: Inter-cluster configuration examples where 1 and 2 are clusterheads.

A further hop involves an additional forward of the message to reach the nearby CH. For example on Figure 3.13d, the source CH2 designates a node as ICR (here the blue slave neighbor of CH2). The designated ICR has to make a choice between one of the two (orange) slaves of CH1. This choice is computed by using ICR selection using the list containing only 1 hop distant(from the blue slave: here the orange CH) clusterheads.

More than 2 slaves between 2 clusters case

In the previous paragraph we saw that there can be no more than 2 slaves between two nearby clusterheads. However, between not-nearby clusterheads there can be more. Figure 3.18a) contains an example of such a configuration: between CH1 and CH3 there are 3 slaves. The important point here is that CH1 and CH3 are not considered as nearby clusterheads. Therefore, as shown in Figures 3.18b, c and d, CH2 has 2 nearby clusterheads CH1 and CH3 while both CH1 and CH3 only have one nearby clusterhead which is CH2.

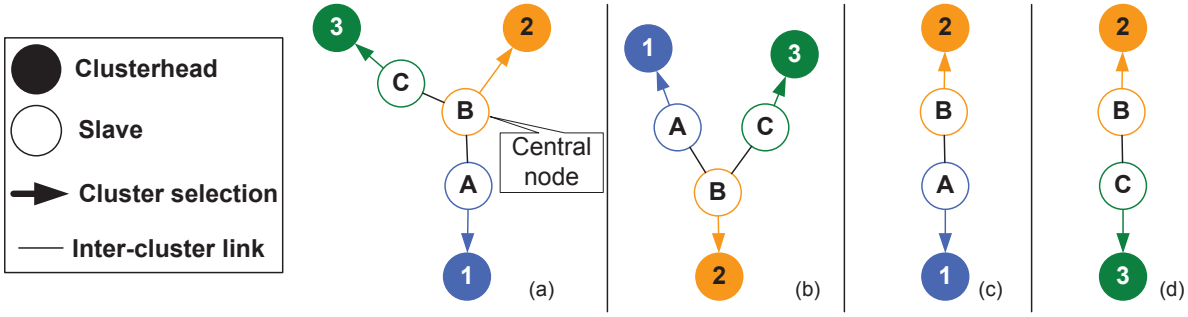


Figure 3.18: a) Cluster configuration with more than 3 hops between two clusters. b) view from node 2. c) view of node 1. d) view of node 3.

This topology has no impact in the way ICR nodes are computed or a fundamental change of processing the message at the central node (i.e. the node between those 3 clusters). Suppose CH1 sends a broadcast message using SLSF. CH1 selects its slave A as ICR. A computes the ICR (3-hop cluster case) and selects node B (central node). B then forwards the message to its CH2. Here node C receives the message but silently discards it since the message traveled 3 hop and C is not a clusterhead, the message therefore should go further. CH2, since it is a broadcast, forwards the message to all its nearby clusters except the one from where it came from, here it forwards it to CH3. To do so, CH2 selects slave B as its ICR. However, B has already seen this message and should discard it according to the rule in SLSF[18]. Here, an exception to this rule has to be added, since we want node B to forward the message a second time in order to reach CH3. The exception should however be tailored specifically to this scenario to avoid unnecessary message transmissions for other topologies. The duplicate forwarding exception rule is:

On reception of an already seen message m, a node n should consider it as new:

- **If** the message m traveled 2-hops the first (or last) time it was considered as new
- **and** the message m designates node n as an ICR node
- **and** the sender of m is a Clusterhead
- **and** the message m only traveled 1 hop (i.e. the sender is an immediate neighbor)
- **then** message m should be processed and forward as if it was never seen before.

Following this rule, node B forwards the message to node C (and selects it as ICR), which then forwards it to CH3.

[18] In SLSF, to avoid infinite message loops, nodes that already received a message discard it silently.

One can argue that messages, in topologies with more than 2 slaves between 2 clusters, are not forwarded along the shortest path. However, using this rule and thereby preventing complex topologies from appearing at the SLSF level provides a strictly bounded simple topology that is considered between two nearby clusters. For example, a topology like shown in Figure 3.19a) with a long sequence of slaves connected together could highly increase the complexity. Although, since we consider only as nearby clusterheads clusters that are at most at a 2 slaves (3 hops) distance and using the duplicate-forward exception to be able to reach all clusters, we simplify this complex topology to its nearby cluster view as depicted on Figure 3.19b).

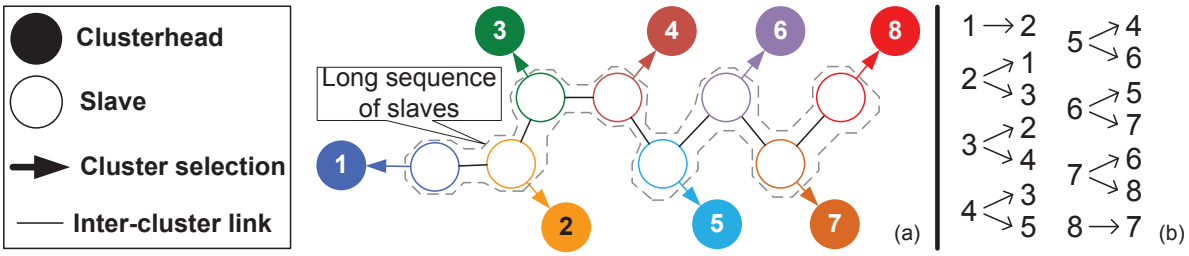


Figure 3.19: a) Long sequence of slaves with SLSF. b) View from each cluster of its actual nearby-clusters.

When to select ICR nodes?

ICR selection is done based on events. Every time a change in the stable neighborhood that influences the ICR calculation occurs, the ICR selection is re-calculated. Thus broadcasting or forwarding a message using ICRs is immediate: replace the ICR set in the message with the one locally pre-calculated. Further detail on broadcasting in SLSF in section 3.4.2.

ICR: The big picture.

To highlight the gain of ICR selection, Figure 3.20 shows an example with 5 clusters where the message source CH S sends a broadcast containing the ICR selected neighbors 1,2,3,4 and 5.

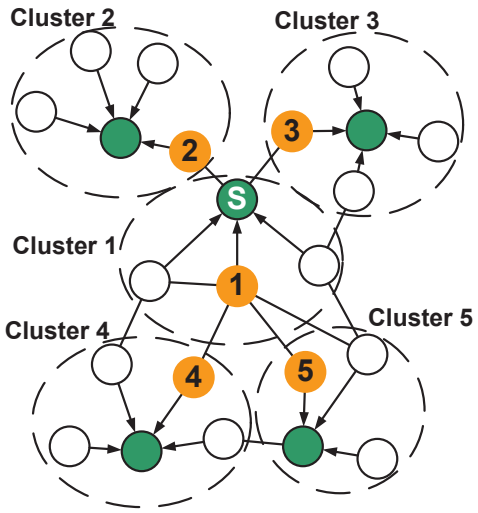


Figure 3.20: ICR selection with 5 clusters in SLSF.

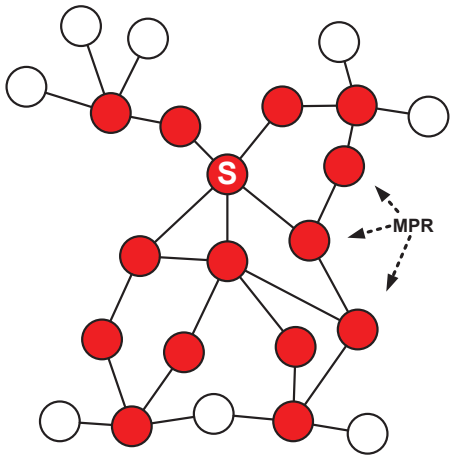


Figure 3.21: MPR selection in OLSR.

On reception of this broadcast only nodes 1, 2 and 3 will forward the message, while the other neighboring nodes process the message silently. Note that node 1 selects 4 and 5 as ICR according to section 3.4.1 "3-hop Inter-cluster case".

We see that ICR selection reduces a lot the number of forwarding nodes. As an example, on Figure 3.20 there are 23 nodes in the network and only 10 nodes (including the CHs) are emitting to reach all the nodes in the network. Every CH will emit the message once, in order for their slaves to receive it and if necessary include their local ICR selection for further forwarding in the network (see section 3.4.2). In comparison, there would be 15 nodes forwarding the message using OLSR, as shown on Figure 3.21. This is due to OLSR using only 2-hop information while SLSF uses 1-hop cluster information which represents information from up to 3-hops away. While 3-hop knowledge usually increases the amount of information to collect, using clusters reduces drastically the nodes to keep track of for ICR selection.

3.4.2 SLSF - Broadcast

At this point, every communication occurs between one cluster and its nearby clusters. To enable communication with foreign clusters, we propose a simple broadcast mechanism.

The broadcast mechanism is simple now that we only need to deal at cluster level. Every node, clusterhead or slave, that has a message to broadcast, adds its address, a sequence number and a time to live value. Additionally the number of hops is tracked. So, a message ready for broadcast has the following fields: Source Address, Source Sequence Number, Source hop count and TTL (Time to live). If the node willing to broadcast a message is a slave, it simply unicasts it to its clusterhead. Clusterheads, for their own messages or from messages received from slaves, compute the ICRs and add related information to the messages. The information concerning ICRs is the following: the list of nodes selected as ICR, the number of nodes selected as ICR (used for message parsing), the last crossed CH (to exclude the CH from where the message comes from the ICR selection) and the number of hops since the last CH (to avoid out of bound ICR selection). The following section shows the complete message header formats for SLSF messages.

3.4.3 Basic SLSF message Header format

Following is the format of the header of a SLSF broadcast message (Figures 3.22, 3.23). The fields Source, Source sequence Number, from Source and TTL apply during the overall path taken by the message. Whereas, the fields ICR set, Last Crossed Clusterhead and HLC (hop-count from last clusterhead) are only valid between two clusterheads and change at each clusterhead passed by the message.

The following example uses only the relevant fields to simplify the figure. On Figure 3.24 Node 1 sends a message by unicasting (since 1 is a slave) the message to CH9. CH9 computes the ICRs and adds the additional information to the packet. Nodes then follow the ICR mechanism. Note that Node 7 is in the "3-hop inter-cluster case" (section 3.4.1), and computes ICRs by excluding the CH8 (last crossed CH) from the selection. The message reaches all the nodes in the network following the path depicted on Figure 3.25.

3.4.4 IP fragmentation and SLSF

The previous section showed the format of the basic SLSF header. The header has a fixed part of 16 bytes and a variable part of 4 bytes per selected ICR node. IPv4 packets have a payload capacity of maximum 1480 bytes[19]. Payloads bigger than 1480 bytes are fragmented on several

[19] Ethernet maximum payload length 1500 - IPv4 header length 20

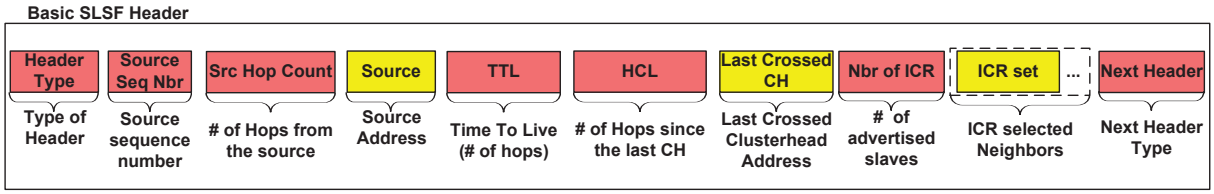


Figure 3.22: Header of a basic broadcast message (Graphical representation).

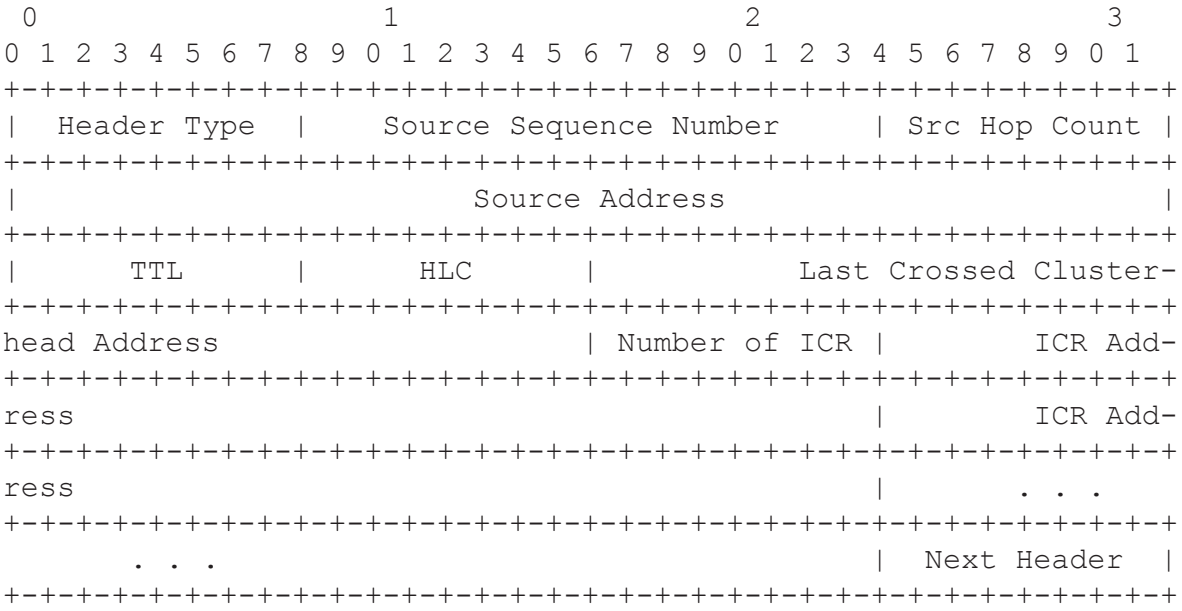


Figure 3.23: Header of a basic broadcast message (Textual representation).

IP packets with each their own IP header. SLSF adds a new header to each IP packet, the remaining payload capacity is maximum 1500-16=1484 bytes when no ICR is selected (e.g. a border clusterhead node). Each selected ICR decreases the payload capacity by 4 bytes. With 2 ICRs, the maximum payload capacity is 1476 bytes. A node selects at most one ICR per nearby clusterhead. However, the number of nearby clusterheads, hence also the ICRs, can vary along the route of a message. This leads to a problem with fragmented payloads.

For example, a large payload of 2000 bytes is sent using SLSF. The source clusterhead has 3 nearby clusters. The SLSF header has the size of $16 + (3 \times 4) = 28$ *bytes*. The payload is then fragmented in 2 parts: the first of $1480 - 28 = 1452$ *bytes* and the second the remaining payload of $2000 - 1452 = 508$ *bytes*. The first IP packet has its maximum capacity of 1500 bytes and the second IP packet has 536 bytes (SLSF header + payload) of payload. However, if another clusterhead that receives the message has more than 3 ICR nodes to select, the message would have to be fragmented again since the SLSF header plus the 1452 bytes payload would exceed the 1480 bytes limit. Therefore, to avoid a message to be refragmented during its journey a new rule is introduced: ICR selection has a validity time of 2 seconds and payloads that are more than 1424 bytes[20] will not have any ICR selection added. A source node sending a payload of more

[20] $1480\ bytes - (16\ bytes + 10 \times 4\ bytes) = 1424\ bytes$, where 16 is the minimum SLSF header size and 10 is the maximum theoretical number of nearby clusterheads (never reached in simulations)

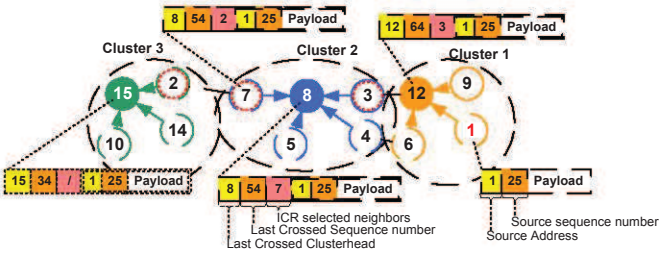


Figure 3.24: Foreign-cluster broadcast - Format of a message sent from node 1.

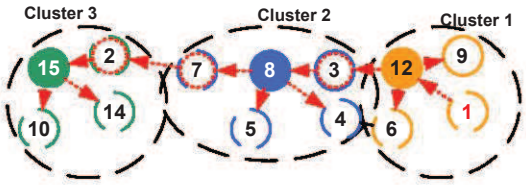


Figure 3.25: Foreign-cluster broadcast - Path of a message sent from node 1.

than 1424 bytes checks if it has sent a broadcast with ICR information in the last 2 seconds. If true: send the message, the ICR selection is still valid. If false: send a first message containing only headers (thus the ICR selection) to setup up the ICR path throughout the network and thereafter send the message.

Traffic scenarios of broadcasts with a lot of big payloads are probably applications such as video streams. Those specific scenarios should be handled more specifically. Nevertheless, if only big payloads are sent with a high rate, the process of sending a payload-less message every 2 seconds to set up the ICR path isn't much of an overhead. Moreover, any scenario with a high traffic rate benefits from the ICR validity time. By inserting the specific ICR selection only in a small percentage of the message, by maintaining a periodicity of at most 2 seconds, the overhead of SLSF can be reduced.

3.5 Full SLSF with Fault-recovery

In [Lecl 10c] we add fault recovery to SLSF-R (basic SLSF). The goal of fault-recovery is to be able to transmit the message even if the ICR path fails without having the source to re-emit the message. To do so we propose two mechanisms, the first is the acknowledgement mechanism, enabling broadcasts to be acknowledged by nearby CHs and the second is the delayed transmission detecting transmission errors and handling them.

3.5.1 Acknowledgment mechanism

The following acknowledgement mechanism has two advantages. The first, for which it was actually designed, is permitting a delayed transmission to compensate local intermittent node failures. The second is classic acknowledgment of messages but only between adjacent CHs as opposed to acknowledgements from one end of the network to the other. The classic cluster-to-cluster acknowledgement is a consequence of the initial design. Further consideration of end-to-end acknowledgements would be out of scope for this paper but is an open interest for future work.

Our acknowledgement mechanism works using sequence numbers. Every node puts inside its beacon, sequence numbers to acknowledge messages. However, only CHs acknowledge the messages while inter-cluster nodes forward merely the acknowledgement information coming from nearby CHs to their neighborhood. As consequence, beacons of CHs have a different format than inter-cluster node beacons. Following is an example illustrating how the acknowledgement mechanism works. Using Figure 1.11 as reference, we suppose node 9 sends a broadcast with sequence number N9. Nodes 1, 2 and 3 receive the message. Only node 3 will forward it to its neighborhood as it is the designated ICR (since it is the only inter-cluster node). Node 1 and 2 process silently the message without forwarding it. CH8 receives the message from node 3,

processes the messages, and puts the acknowledged sequence number N9 for CH9 in its beacon. The beacon of CH8 contains now its weight, its CH-address (here its own), its nearby CHs with the corresponding hop count and the acknowledged sequence number. So, the SLSF beacon (Figure 3.26a) is only extended with one sequence number (Figure 3.26b).

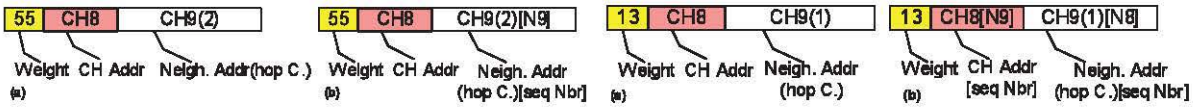


Figure 3.26: CH8 Beacon: (a)SLSF. (b)SLSF with fault recovery.

Figure 3.27: Node 3 Beacon: (a)SLSF. (b)SLSF with fault recovery.

Node 3 reads the CH8 beacon (Figure 3.26b) and includes the new sequence number acknowledged by CH8 for the message source CH9 in its beacon. In order to reduce the beacon size for inter-cluster nodes, we compact acknowledgment information into the basic SLSF beacon by just adding sequence numbers in the right order and place. If we consider the basic SLSF beacon (Figure 3.27a) for the slave node 3 we integrate additionally the information that CH8 acknowledges sequence number N9 for CH9 and that CH9 acknowledges N8 for CH8 inside the beacon (Figure 3.27b). Note the inversion of the sequence numbers compared to the beacon of CH 8.

To really point out this inversion an example where three nearby clusters are inter-connected is necessary (Figure 3.28). In this example, node 3 is connected to three clusters. As a matter of fact, node 3 will forward any message exchange between those clusters. The beacon of node 3 has to contain all the acknowledgement information for the three CHs. The basic SLSF beacon of node 3 of Figure 3.28 is shown in Figure 3.29.

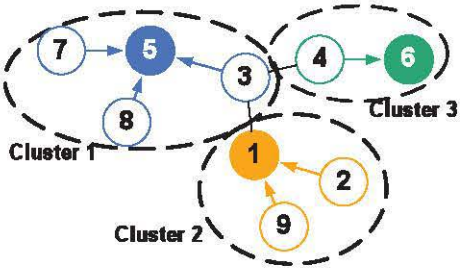


Figure 3.28: Three clusters example.

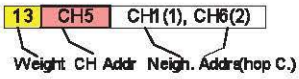


Figure 3.29: Node 3 SLSF beacon.

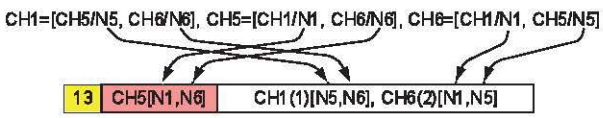


Figure 3.30: Node 3 SLSF beacon with fault recovery .

Node 3's beacon contains its own clusterhead, CH5, and two nearby clusterheads, CH1 and CH6, on 1 and 2 hop distance respectively. Now we consider we want the following information into that beacon, like for instance after some broadcasts were sent:

- CH1 acknowledges N5 for CH5 and N6 for CH 6
- CH5 acknowledges N1 for CH1 and N6 for CH 6
- CH6 acknowledges N5 for CH5 and N1 for CH 1

Where Nx corresponds to the acknowledged sequence number.
Writing this differently gives us:

- CH1 [CH5/N5, CH6/N6]

- CH5=[CH1/N1, CH6/N6]
- CH6=[CH5/N5, CH1/N1]

The original SLSF beacon already contains the three CH-addresses (typically the IP addresses of the CHs); the own CH-address and two nearby CH-addresses. First sort all the acknowledgements CH-addresses in ascending order. For example CH6=[CH5/N5, CH1/N1] is sorted as CH6=[CH1/N1, CH5/N5]. For each CH-address in the beacon attach the corresponding sequence numbers and omit the superfluous CH-addresses inside the list, as shown on Figure 3.30. Doing so, putting in the right order the right sequence numbers, results in a new beacon with all the needed information just by adding the needed sequence numbers without additional addresses.

The constructed beacon (Figure 3.30) is received by the neighbors. For instance CH1 can read in the beacon [N5,N6] and by ordering the announced CHs (CH1, CH5, CH6) and omitting itself (CH5, CH6) it can read that CH5 acknowledges N5 and CH6 acknowledges N6.

The presented acknowledgement mechanism permits acknowledgements between a cluster and its nearby clusters. While the acknowledgements of the CHs using beacons are straightforward, the inter-cluster beacons of the nodes are constructed using inversion and re-ordering to avoid redundant information inside the beacon and still enabling the source cluster to distinguish which cluster acknowledges which messages without any other message exchange than beacons.

3.5.2 Delayed transmission mechanism

The ICR selection enables the communication among nearby clusters in an optimized way. To keep the inter-cluster communication reliable in case the selected ICR path failed, we introduce a delayed transmission mechanism. Thus, we have an immediate communication path formed by the ICRs and we keep a backup, although delayed, path in case of failures.

Delayed transmission occurs as follows: If a broadcasted message is emitted from a source, all neighbor nodes NOT selected as ICR in the message will keep the message in cache and only in case of a failure transmit the message. While the ICR nodes forward the messages, the nodes pending for delayed transmission observe the neighboring beacons for acknowledgements of nearby CHs. On reception of a pending acknowledgement, the delayed transmission is aborted. If the pending time for delayed transmission from a given CH times out, the message is broadcasted with all the neighbors that announced the given CH, to maximize the chances of reception, in the ICR set. Nodes that already received the message will discard it silently while others forward it immediately (since they are ICR nodes). The message arrives at destination in case of failure without re-emission of the message from the source. Thus, having only nodes that did not try forwarding the message yet, effectively (re)transmitting the message.

Following discusses the delay chosen for timeout to occur. If we consider the beacon-interval as bI, the number of nodes the message already passed through as $hopCount$ and 4 the maximal number of hops for a message to go back and forth from the first slave to a cluster on (maximal) 2-hop distance. Then the transmission delay td is calculated as follows: $td = (\frac{4}{hopCount}) \times bI$. If the beacon-interval is 1 second and the hop-count is 1 then the transmission delay is set to 4 seconds. If the hop-count is 2 then the transmission delay is set to 2 seconds.

Observing acknowledgments

As mentioned above, nodes not selected as ICR observe the acknowledgments to check whether the message arrived successfully at all nearby clusterheads. As it is a dynamic network and one wants to keep the bandwidth low, the messages emitted from the source clusterhead do

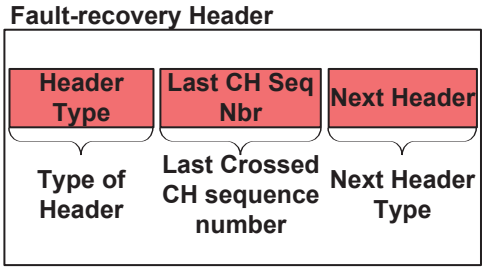


Figure 3.31: Header for Fault-recoverable broadcast message (Graphical representation).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Header Type  |Last CH Crossed Sequence Number|  Next Header  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.32: Header for Fault-recoverable broadcast message (Textual representation).

not contain which clusterheads are aimed for reception (i.e. the clusterheads that are nearby-clusterheads of the source). While the ICR calculation optimizes the inter-cluster nodes needed to reach all nearby cluster, the inter-cluster nodes can detect, by knowing only the local information acquired from neighbor beacons, which nearby clusterheads should receive the message. As consequence, the NOT ICR observe all neighbor beacons. If the neighbor beacon contains an earlier acknowledgement from a clusterhead for the given message source then it is included in the awaiting acknowledgment list. For bootstrap reasons, if it is the first message emitted from a source, the nearby clusterhead will have -1 as first acknowledged sequence number and still be able to observe correctly.

3.5.3 Fault Recovery Header format

To be able to be delayed-retransmitted, additional information is added to a basic SLSF message. Fault recovery is only operated between two nearby clusters. Nodes between two nearby clusters do not know about foreign nodes. Therefore, all fault-recoverable messages, when entering between nearby clusters, need a local identity which is provided by the *last crossed CH* field (Section 3.4.3) and also need a local unique identification which is obtained by adding a *last crossed sequence number* field. So, a message, at each clusterhead, receives a new local identity and local unique identification. Those are the information that are acknowledged via beacons. The fault-recovery header format is shown on Figures 3.31, 3.32.

Using a separate header for fault recovery permits to selectively enable or disable it on a per-packet basis. A message that should not be fault recovered simply does not include this header. This can be very useful with delay sensible traffic, where once a message is delayed, it is not of any use anymore. This reduces the size of the message and the functionality of the network but does not reduce the beacon, since other messages or nearby clusterheads might still use fault recovery. A complete deactivation of fault recovery is proposed in an optional header presented together with other optional headers in the next chapter in Section 4.6.1.

3.6 Experiments and Results

To evaluate the performances of our SLSF protocol, we implemented the three protocols (OLSR, NLWCA/WCPD and SLSF) on top of the JANE simulator [Gorg 07] and performed several simulations. For those experiments we used the Restricted Random Way Point mobility model[21] [Blav 02], whereby the devices move along defined streets on the map of Luxembourg City for 1000 seconds. For each device the speed was randomly varied between [0.5;1.5] units/s with a transmission range of 25 units. For each experiment 10 different random distribution seeds were used in order to feature results from different topologies and movement setups. For the used mobile environment where nodes move with low (walking) speeds between 1.8 and 5.4 km/h, the NLWCA link-stability threshold is set on 2 [Andr 08b].

Simulations were done to determine the bandwidth used by the protocols in order to build the topologies and the information dissemination performance of broadcasting on top of the different topologies. Then we compared the efficiency of the protocols and finally made a static evaluation to compare information dissemination solely on MPR and ICR performances.

OLSR exchanges the sets of one-hop neighbor nodes with every node in communication range. Similar to OLSR, SLSF exchanges the list of the discovered nearby-CHs with the one-hop neighbor nodes. For our experiments we distinguished two different SLSF configurations. the first is SLSF without fault recovery referred as SLSF-R (minus fault recovery), thus only ICR selection with basic SLSF beacon (same format as the WCPD beacon, thus same bandwidth usage). The second is the full SLSF protocol with ICR selection and fault-recovery mechanism with added sequence numbers in the beacon. To find out the network load produced during this phase, the size of the exchanged data sets were tracked every second of the simulation: for OLSR the size of the one-hop neighbor sets, for SLSF-R and WCPD the size of the discovered CHs and for SLSF additionally to the discovered CHs the sequence numbers.

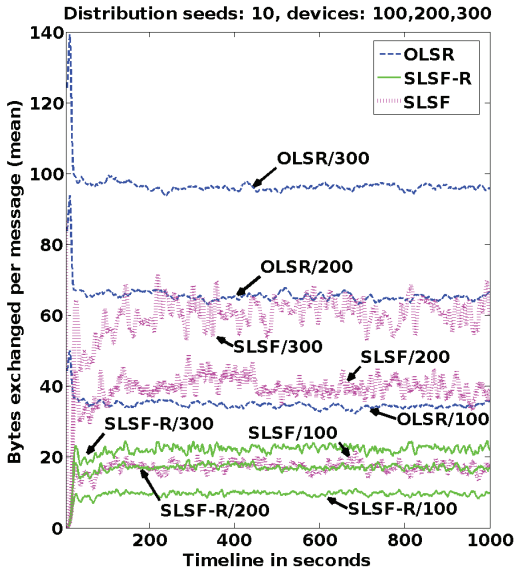


Figure 3.33: Bandwidth used in order to build the topology for 100, 200 and 300 nodes.

In order to monitor the information dissemination performance (Reachability) a node was chosen to broadcast a message every 10 seconds during different simulation runs. The number of

[21] at the time those experimentations were made, we did not have the advanced mobility models proposed in Chapter 5

sent messages (i.e. broadcasts and unicasts) during the dissemination and the number of reached network nodes were tracked.

As shown in Figure 3.33, OLSR uses a higher bandwidth in both sparser (100 nodes) and denser networks (300 nodes). This was expected since OLSR is exchanging the set of one-hop neighbors, while SLSF only exchanges the set of locally discovered nearby CHs which is a fractional amount of the total number of nodes. SLSF-R uses exactly the same bandwidth (80% less than OLSR) as WCPD since they share the same beacon structure. SLSF uses more bandwidth than SLSF-R, as sequence numbers were added to the beacon, but still uses about 40% less bandwidth than OLSR.

The dissemination performance results (Figure 3.34, NOTE: periodicity in curves is induced by the smoothing) show that SLSF performs the best for all densities. For 300 nodes SLSF performs only slightly better than OLSR, but uses on average 10 to 15% less forwarders with 40% less bandwidth usage. WCPD performs the worst and uses accordingly lesser forwarding nodes. Whereas SLSF-R uses approximately the same amount of forwarders than WCPD, it reaches from 10% to 20% more nodes. This is the pure gain of ICR selection which optimizes the forwarding nodes.

Subsequently we calculated a "quality-cost" ratio extracted from the results of Figures 3.33 and 3.34. We calculated the percentage of nodes reached, divided by the bandwidth used. We see in Figure 3.35 that SLSF and SLSF-R are in average two to three times more efficient than OLSR. The poor performances of WCPD highlight the need for improvement that SLSF brings.

SLSF relies on stable structures built by NLWCA: only nodes considered as stable will receive the message. So finally, to compare the performances on equal levels, we experimented OLSR and SLSF in a static scenario where all nodes are considered stable connected. The experiments where done on a 300x300 units surface with 100 to 300 nodes randomly positioned using 100 different topology seeds. Again, the number of forwarding and receiving nodes using MPR and ICR selection where tracked. The results on Figure 3.36 show that SLSF outperforms OLSR in terms of ratio of forwarding nodes over receiving nodes. With increasing density on average with OLSR about 85% of the receivers are also forwarders, whereas in SLSF this amount decreases from 60% towards 30%.

3.7 Conclusion

We presented SLSF, a flooding protocol which selects Inter-Cluster Relays to optimize the communication among the stable-connected cluster architecture. To deal with intermittent message loss a fault recovery mechanism was added.

The goal of the ICR (Inter-Cluster Relay) selection is to reach all nearby clusterheads with the minimal set of 1-hop neighbors while optimizing the hop-count. ICR selection on top of the stable-cluster architecture reduces substantially the number of forwarding nodes. Generally, as shown by the experiment results, SLSF performs well in high density networks while keeping the used bandwidth very low.

SLSF offers a very good basis for applications or protocols in which dissemination is the main messaging scheme. We will use SLSF in two further Chapters. Chapter 4, where SLSF is used as basis for a routing protocol and Chapter 7 where Zeroconf uses SLSF as replacement for the required multicast layer.

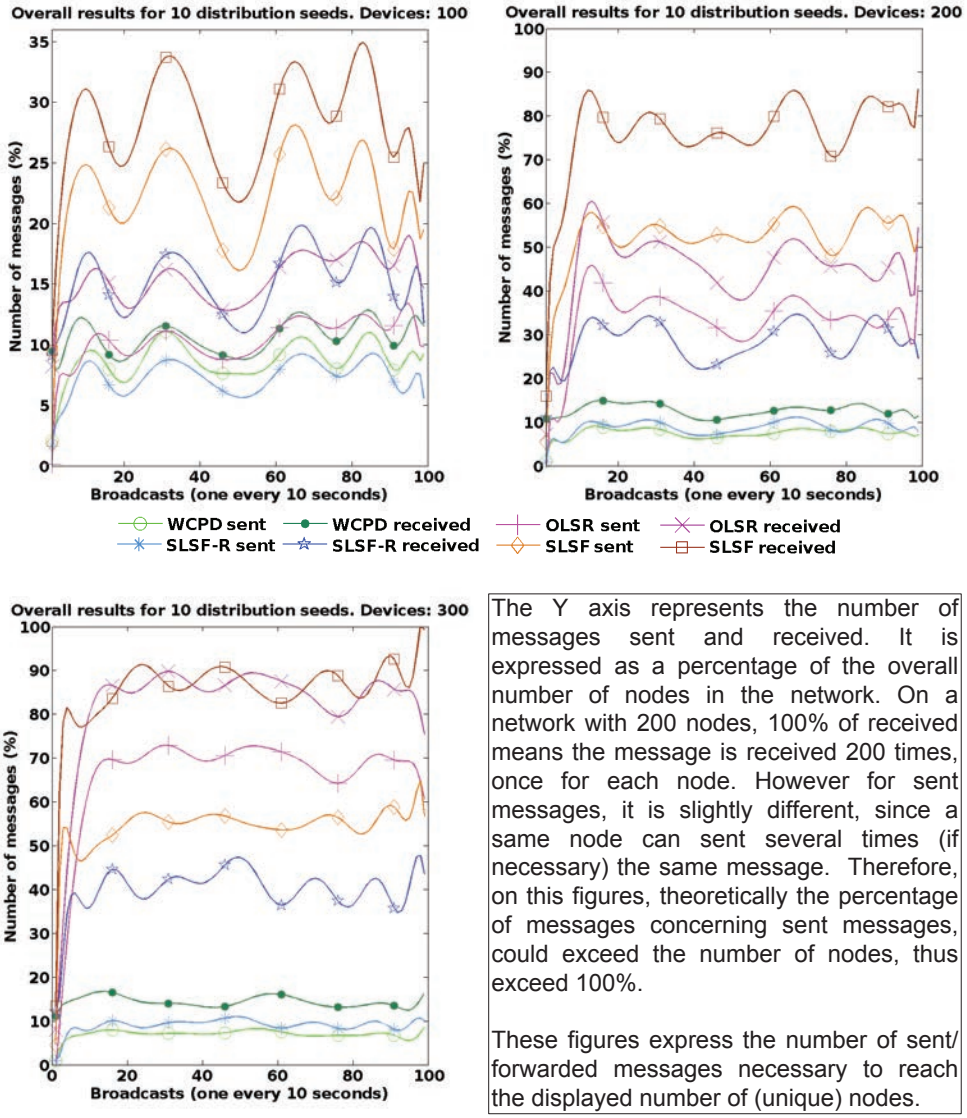


Figure 3.34: Overall number of sent/forwarded messages and received for 100, 200 and 300 nodes. (smoothed with a polynomial equation of the 16th grade for visibility sake).

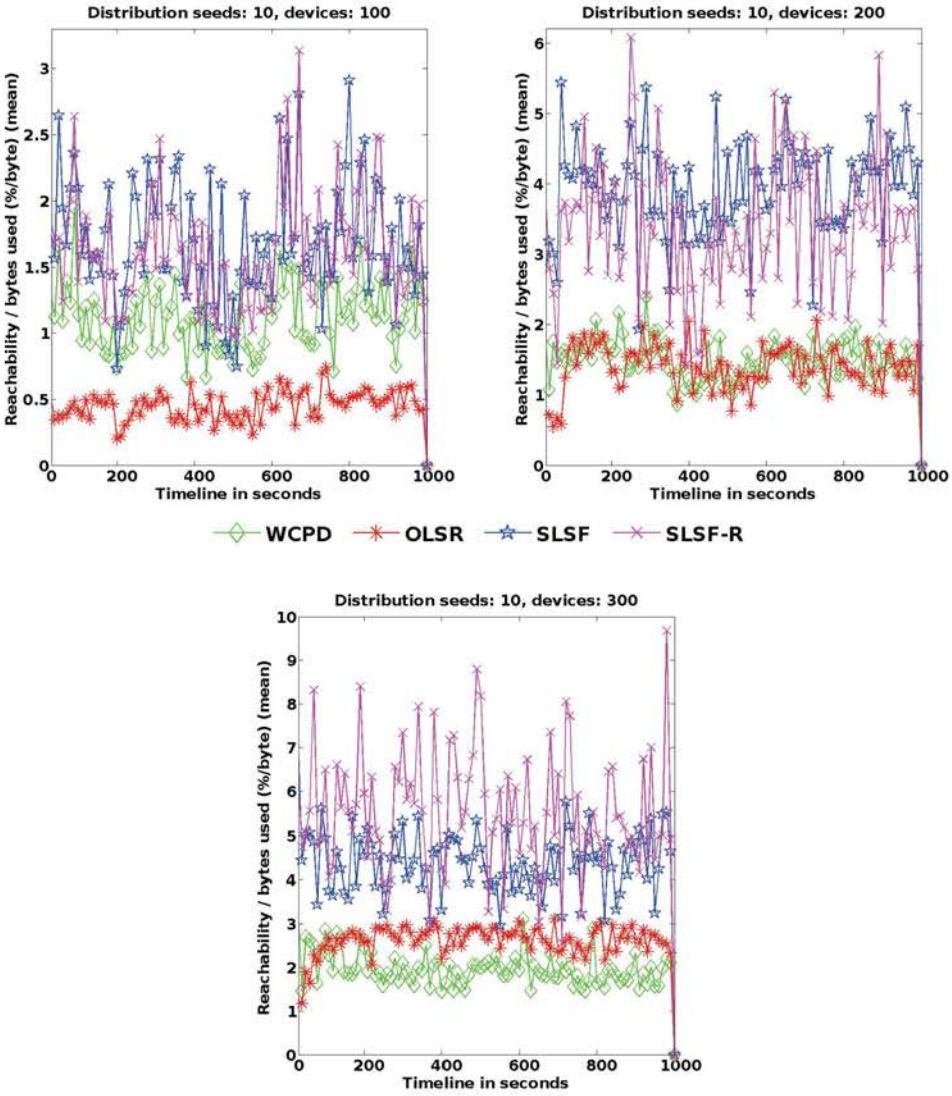


Figure 3.35: Efficiency of Bandwidth usage for 100, 200 and 300 nodes.

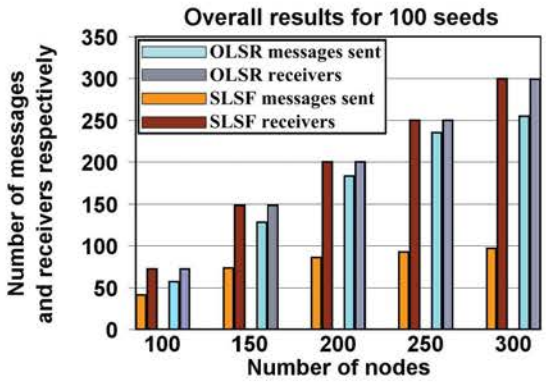


Figure 3.36: Static scenario with 100 to 300 node.s

Chapter 4

Routing: SLSR - Stable Linked Structure Routing

Contents

4.1 Message types and topology gathering 75
4.2 A balanced-hybrid protocol to reduce the overhead 76
4.3 Routing Table and update management 78
4.3.1 Choosing a route 78
4.3.2 Table updates from Current Slaves messages 78
4.3.3 Table updates from lost Slaves messages 79
4.3.4 Table updates from Lost Route to Clusters messages 80
4.4 Symbiotic behavior 81
4.4.1 Higher-layer traffic 81
4.4.2 Message timings and header combinations 82
4.5 Message header formats 83
4.5.1 Routed Message 83
4.5.2 Current Slaves 83
4.5.3 Lost Slaves 85
4.5.4 Lost Route to Clusters 85
4.6 Advanced Configuration and option messages 85
4.6.1 Fault recovery activation and deactivation 85
4.6.2 Extended forwarding of routing information 85
4.6.3 Cross-layer design 87
4.6.4 Full link-state behavior 88
4.7 Conclusion 89

This chapter presents SLSR (Stable Linked Structure Routing), a routing protocol that takes advantage of the SLSF dissemination structure. It can be classified as a balanced-hybrid protocol which has both link-state and distance vector features. It is a proactive protocol with a symbiotic behavior: it uses periodic or event-based messages to keep the routes up-to-date, but when available, profits from the higher-layer traffic in a symbiotic way by adding (i.e. piggy-backing) small routing information to the original message. The goal of SLSR is to profit from existing under and overlying protocols and structures, to reduce the routing overhead, the number of messages and their size. SLSR creates a simplified view of the network which facilitates the routing process. Clusterheads play a key role. They manage and keep the topology information up-to-date while slaves merely forward information.

SLSF as base structure

Using SLSF as base structure permits to simplify the routing procedure and to focus only on clusterheads and inter-cluster paths. Thereby, from a structure built by SLSF, SLSR creates a simplified overlay view (Figure 4.1). This has several advantages:

- The routing information can be efficiently disseminated using the SLSF dissemination.
- The routing operations can be concentrated at the clusterheads:
  - Only clusterheads need to send out routing advertisements.
  - Routing operations and decisions are done at clusterhead nodes. Slave nodes merely forward packets.
- Pruning of routes can be aggregated to entire clusters (i.e. an announce of a lost nearby cluster will remove the cluster itself but also all its members from the routes). The pruning messages can be reduced to the concerned clusterhead address.

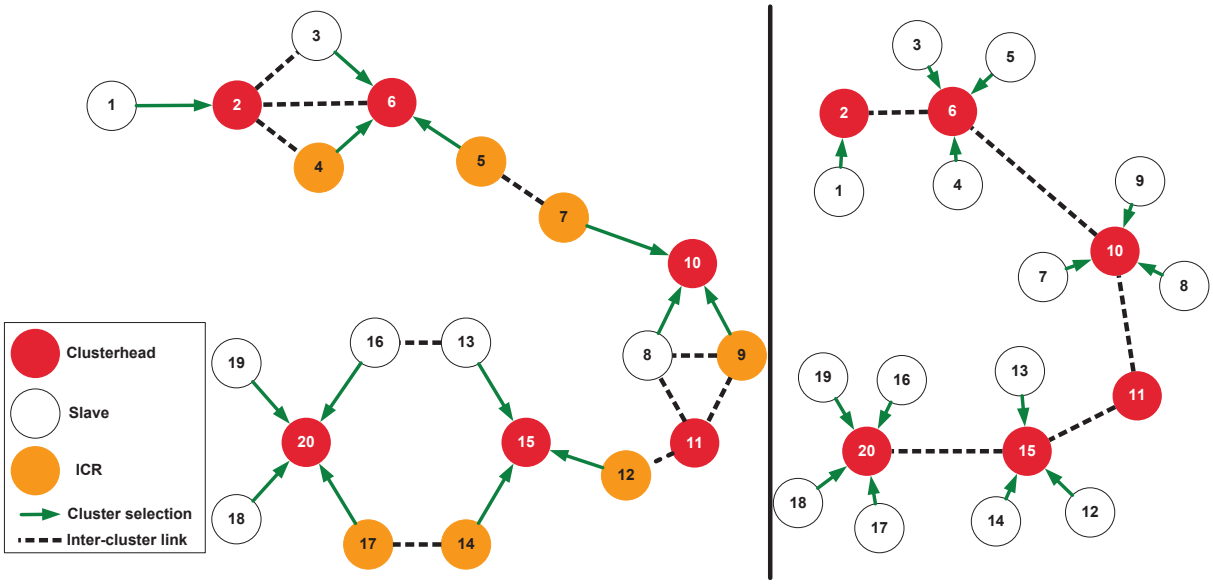


Figure 4.1: Same network from two different point of views: SLSF (left) and SLSR (right).

Message routing and forwarding

In SLSR, instead of forwarding a message to the next hop as in most routing protocols, messages are forwarded to the next clusterhead. The path taken by the message between two clusters is entirely managed by SLSF with its ICR selection and fault-recovery mechanism. Therefore, all routing operations are done at clusterhead nodes and all routes mention only clusterheads. For example, Figure 4.2 shows 3 clusters and their ICR paths formed by SLSF on the left and the overlay view from the SLSR perspective on the right. The routing table for slave node 19 (Table 4.1) is very simple, and contains its own clusterhead address for all destinations as "next clusterhead". Similarly, for the routing tables of clusterheads 20 and 15 (Tables 4.2 and 4.3) which contain as "next clusterhead" for each destination the clusterhead from which the information about this destination came from. A message traveling across the network is, at each clusterhead,

Dst	Next CH
16	20
17	20
18	20
20	20
9	20
11	20
12	20
13	20
14	20
15	20

Table 4.1: Simplified routing table for slave node 19.

Dst	Next CH
16	20
17	20
18	20
19	20
9	15
11	15
12	15
13	15
14	15
15	15

Table 4.2: Simplified routing table for clusterhead node 20.

Dst	Next CH
12	15
13	15
14	15
11	11
9	11
20	20
16	20
17	20
18	20
19	20

Table 4.3: Simplified routing table for clusterhead node 15.

forwarded to the next clusterhead towards the destination. Slaves, are transparent at the SLSR level and only intervene at the SLSF level for ICR forwarding. Following sections describe how the routing table is populated and kept up-to-date.

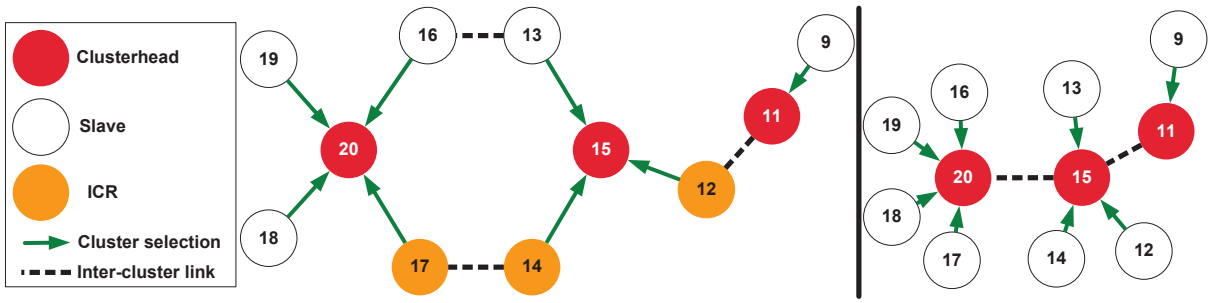


Figure 4.2: SLSR - three clusters example. SLSF view (left) and SLSR overlay view (right).

4.1 Message types and topology gathering

Each message sent by SLSR is encapsulated in a SLSF broadcast message header that will take care of the correct dissemination in the network (Sections 3.4.3 and 3.5.3). SLSR introduces 4 new types of message headers: *Routed-message*, *Current Slaves*, *Lost Slaves* and *Lost Route to Clusters*. The detail format and fields of these messages can be found in Section 4.5.

The first, **Routed-message**, adds unicast and routing capabilities to a SLSF message. It contains simply the destination address, to unicast a message, and the next clusterhead address, to route that message from cluster to cluster towards its destination.

The other 3 types of message headers are sent (SLSF-broadcasted) only by clusterheads to build and keep the topology information and routing tables up-to-date:

- **Current Slaves:** Clusterheads announce their current cluster members to the rest of the network. Those messages contain the list of current slaves of the announcing clusterhead. They are sent periodically but can also be triggered by events (i.e. arrival of a new member). *Current Slaves* messages already suffice to populate fully the routing tables of all the nodes

in the network. They have a validity time, stale routes automatically expire. However, to be able to be more reactive on topology changes and quickly update the topology information, the following two other header types are proposed.

- **Lost Slaves:** Clusterheads announce the slaves they lost. Those messages contain the list of lost slaves and are sent only based on events (e.g. lost of a slave). A node receiving such a message simply removes each entry matching the announcing clusterhead and the lost slave from its routing tables.

- **Lost Route to Clusters:** Clusterheads also announce the loss of clusterheads when they lose a link to a nearby clusterhead. In other words, they announce the lost clusterheads that they were able to reach using the route through the lost nearby clusterhead, before the loss. Those messages contain the list of lost clusterheads (including the nearby clusterhead itself) and the number of remaining routes; Along, with each announced lost cluster, each clusterhead forwarding this message (including the source of the message) puts in the message how many routes he has left to this clusterhead. The number of routes is used to avoid pruning a route while there is still a path available. More on this in Section 4.3.4.

4.2 A balanced-hybrid protocol to reduce the overhead

SLSR can be classified as a balanced-hybrid protocol. This term was introduced in the Enhanced Interior Gateway Routing Protocol (EIGRP) [EIGRP 92], a Cisco proprietary protocol. Balanced-hybrid protocols take features from both link-state and distance vector algorithms. However, except sharing the same classification, SLSR has no similarity with EIGRP, therefore we are not going to detail it.

The routing table of SLSR looks like a distance-vector protocol routing table, however the gathering of information to fill this routing table is done based on link-state principles: A node (here a clusterhead) sends to all nodes in the network, information about its neighbors (here its slaves only). SLSR is also not a link-state algorithm since the routing table does not contain the complete topology. The difference from classic link-state or distance-vector based protocols is that the information that clusterheads advertise does not overlap. However, a recipient of SLSR routing information cannot deduct that two clusters are neighbors, as it is the case in a classic link-state algorithm, but only knows from where the information came from (more like distance vector algorithms).

Figure 4.3 compares the reach of routing information in link-state (here OLSR), distance-vector and SLSR. In link-state and distance vector algorithms, the advertised information highly overlaps thereby creates a high routing overhead. In SLSR, information never overlaps due to the use of clusterheads, thereby the routing overhead is minimized. Of course, this comes at a cost of less topology knowledge. A node only knows the direction in which to send information (this is where the routing table is similar to distance-vector routing tables) and not the entire topology. However, in a dynamic configuration knowing the topology may be wasted effort since it might already have changed during the journey of a message, next (clusterhead) hop information is all that matters from a node's point of view. Note, that SLSR can also, if necessary, work as a full link-state protocol as described in Section 4.6.4.

SLSR does not come with the classic problems of distance-vector such as count-to-infinity (see framed paragraph hereafter) or routing loops. Why? Because any information that is send is only send by a node that is the owner of that piece of information. No one speaks for another node. Current Slaves and Lost Slaves messages are only send by the source (i.e. the clusterhead) of the information, the other nodes only forward it without altering it. Lost Route to Clusters messages, are send only by nodes that effectively lost a route. Additionally,

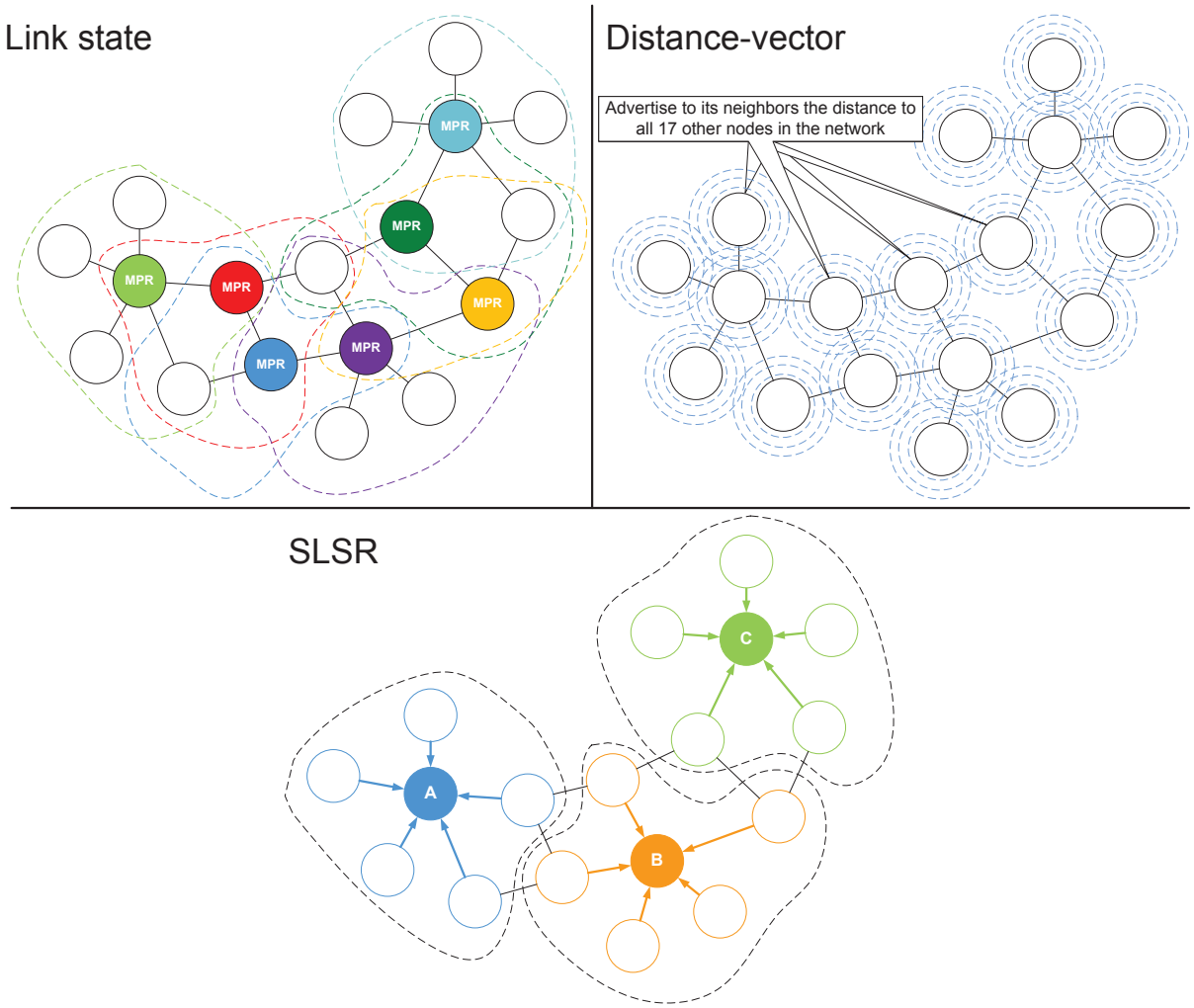


Figure 4.3: Comparison of routing overhead by displaying the scope of an announcement in each routing advertisement.

messages are always send onward, never backwards (i.e. from where the message came from). Also, duplicate messages are silently discarded, thereby loops will automatically be absorbed by nodes that already received the message from the other side of the loop.

Count-to-infinity: suppose A-B-C are connected in a row. B has a distance of 1 to A and C a distance of 2 to A. If using distance vector, only distances are know, so B does not know that C accesses A through B. If A goes down and C is not yet aware of it, B sees that it can reach A through C in 3 hops (C can reach A in 2). Thereafter, C sees that B reaches now A in 3 hops instead of 1, and updates its distance information. B then again updates its distance to A to 4 and so on. This only stops until the maximum ("infinity") distance is reached.

4.3 Routing Table and update management

Before explaining how the routing table entries are filled, next are the fields of the routing table of SLSR (Table 4.4):

- **Dst:** Destination Address.
- **Next CH:** Next clusterhead address. The next CH hop towards the destination.
- **metric:** in our example, the number of hops to the destination. Any other metric to represent the path cost can be used.
- **CH metric:** Here, the number of clusters to cross to reach the destination.
- **Dst CH addr:** Clusterhead address of the destination.
- **Expire Time:** Expiration time of the routing table entry. It is calculated on update or creation of a routing entry by adding the current time to the announced *validity time* of the message.
- **Seq. Nbr:** Sequence number of the last message that updated this entry.

4.3.1 Choosing a route

Once the routing table is filled (the following sections show how this is done) the routing table contains all the known routes for each destination. There can be more than one next clusterhead available to reach a destination. For example, in Table 4.4, there are two entries to destination node 12 (lines *4* and *6*). The choice of route is done as follow:

1. select the route with the smallest *CH metric*, or if equal,
2. select the route with the smallest *metric*, or if equal,
3. select the route with the latest *expire time*, or if equal,
4. finally select the route with the smallest *Next CH* address.

4.3.2 Table updates from Current Slaves messages

Nodes can fill their routing table by using the received *Current Slaves* messages. Figure 4.4 shows an example of a network where 3 clusters form a loop. Each clusterhead receives *Current Slaves* messages from 3 other clusterheads. Here, for example, CH 20 receives messages from CH 11, CH 15 and CH 10. Those 3 messages contain for CH 11 its slave 9, for CH 15 its slaves 12, 13 and 14 and for CH 10 no slave. Since there is a loop, messages arrive from 2 different directions.

> **Duplicate handling:** To handle more than one different path, a special treatment on the duplicate message handling of SLSF (Section 3.4.1) is made : if the message concerns topology information, consider it to populate the routing table, however without forwarding it further.

Suppose the routing table is empty, here is how the routing table of CH 20 on Table 4.4 is filled[22]:

[22] The routing table is extracted from a JANE simulation of SLSR. The *Expire time* therefore reflects a simulation time.

#	Dst	Next CH	metric	CH metric	Dst CH addr	Expire time	Seq. Nbr
1	15	15	3	0	15	125.02	225
2	14	15	4	0	15	125.02	225
3	13	15	4	0	15	125.02	225
4	12	15	4	0	15	125.02	225
5	15	11	5	1	15	125.14	225
6	12	11	6	1	15	125.14	225
7	13	11	6	1	15	125.14	225
8	14	11	6	1	15	125.14	225
9	11	11	3	0	11	129.02	31
10	9	11	4	0	11	129.02	31
11	11	15	5	1	11	129.14	31
12	9	15	6	1	11	129.14	31
13	10	10	2	0	10	131.47	12

Table 4.4: Routing table for clusterhead 20

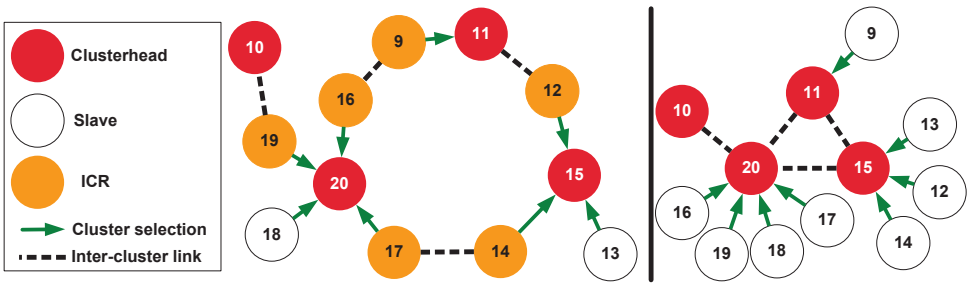


Figure 4.4: SLSR - three clusters with a loop. SLSF view (left) and SLSR overlay view (right).

- On reception of the first message from CH 11 (via 9 and 16): add the lines *9* and *10*.
- On reception of the first message from CH 15 (via 14 and 17): add the lines *1-4*.
- On reception of the second message from CH 11 (via 12, CH 15, 14 and 17): add the lines *11* and *12*.
- On reception of the second message from CH 15 (via 12, CH 11, 9 and 16): add the lines *5-8*.
- On reception of the first (and unique) message from CH 10 (via 19): add the line *13*.

4.3.3 Table updates from lost Slaves messages

Suppose node 13 on Figure 4.4 leaves the vicinity of its CH 15. CH 15 sends out a *Lost Slaves* message containing the lost node 13. Upon reception, nodes remove the corresponding entry linked to the announcing CH from their table. For example, CH 20 removes the entries number *3* and *7* from its routing table (Table 4.4), since they both contain 13 as destination and 15 as destination clusterhead address. If, in between, node 13 already joined a new cluster and its clusterhead sends out a new *Current Slaves* message, those new entries will not be affected by the *Lost Slaves* message sent by its former clusterhead.

4.3.4 Table updates from Lost Route to Clusters messages

This time, on Figure 4.4, suppose the link between node 17 and node 14 breaks. CH 20 lost the nearby CH 15 and along with it the route to CH 11 via CH 15. It can easily extract this information from its routing table (Table 4.4): On the loss of a nearby CH (here, CH 15), a CH announces as "lost route", every unique CH mentioned in the *Dst CH Addr* (here, CH 11 and CH 15) where the *Next CH* is the lost nearby clusterhead (CH 15)[23]. Thus, routes to CH 11 and CH 15 are lost however only via CH 15. Therefore, additionally, CH 20 adds the number of remaining routes for each lost destination CH. Here, for each lost destination, CH 15 and CH 11, there is still one route remaining via CH 11 as shown in the routing table. Similarly, CH 15, announces lost routes to CH 20 and CH 11 with each one route remaining (via CH 11).

Upon reception of a *Lost Route to Clusters* message from a nearby CH, two decisions are taken by a clusterhead[24]: first, to keep or not the corresponding entry in the routing table and second, to forward or not the piece of information. The decision algorithm is shown in Algorithm 1. Back to our example with CH 20. CH 20, has 1 other route for each of both lost routes (CH 15 and CH 11), therefore it removes the lines *1,2,3,4,11* and *12* of its routing table and forwards the message (here unchanged) containing CH15/1 and CH11/1. Slaves of CH 20 receive the *Lost Route to Clusters* message. Since, slaves only have one route available, via their own clusterhead, and all the lost routes announced have at least one other route, no changes in the routing table is necessary. Additionally, slaves always leave the packet unchanged and forward according to the SLSF rules (ICR nodes). CH 10 receives the message and also leaves its routing table unchanged, but does not forward the packet (i.e. the list l in Algorithm 1 becomes empty in the end) any further (even if there were more downstream clusters). Slave 9 receives the message from 16 and forwards it, without processing it, to its CH 11. CH 11, which has two routes to CH 15 but "no route" to itself, removes the information about CH 11 from the message and forwards it, containing only CH15/1 as lost, to slaves and the remaining nearby CH 15. Here, again, CH 15 finally removes CH 15/1 from the message which is not forwarded anymore since it is empty.

Note that, for each lost cluster that is removed from a routing table, all the entries of the slaves of that cluster are also removed. Thereby, SLSF reduces drastically the size of pruning messages when clusters leave vicinity. In, our example, the loss of nearby CH 15 causes the removal of the lines *1,2,3,4* of the routing table.

Another important design choice here is that, downstream nodes are completely unaware of any changes in the upstream topology. When a clusterhead decides not to update its routing table on the announce of a lost route, and not to forward the message, the downstream node are not informed. This, is a good feature since the actual local topology, which is the Next CH pointing to the given destination, did not change. However, there is a tradeoff: with the quiet discard of a lost route, the downstream node does not know, hence update immediately, the new hop count necessary to reach that destination. The tradeoff is, nevertheless, very limited. First, the algorithm only discards *Lost Route to Clusters* if a route is unchanged and available, thus messages arrive at destination, just a little later than expected. Second, as soon as the next *current cluster* message from the corresponding CH arrives, the routing tables will be updated

[23] Routing table entries number *1-4* and *11-12*

[24] Only clusterheads take decisions, slave nodes only consider routing information incoming from their clusterhead. Other message are only forward without being processed

with the new correct hop count.

Algorithm 1: Routing table update and forwarding algorithm.

```
Input: message m contains list l of (cluster/number of routes left) tuples from last
       crossed CH; Routing table RT;
Output: message m now contains updated list l
1  foreach lost-tuple in l do
2      set lost-route=lost-tuple.lost cluster;
3      set nbr-of-route-left=lost-tuple.number of routes left;
4      set iCH = last crossed CH;
5      // if the routing table has a corresponding entry
6      if RT contains entry where (Dst == lost-route AND Next CH == iCH) then
7          // if the route through iCH is the only route available
8          if count-Other-Entries(lost-route, iCH) == 0 then
9              // if iCH says it has another route
10             if nbr-routes-left > 0 then
11                 // keep the entry,if CH remove this tuple from the message
12                 if is-clusterhead then l.remove(lost-tuple);
13             else
14                 // remove the entries from RT
15                 removeAllCorrespondingEntries(lost-route,iCH);
16                 // keep this tuple in the message
17             end
18         else
19             // else there are other routes available
20             // if iCH has 1 or 0 routes left (1, is the route through ourselves)
21             if nbr-routes-left < 2 then
22                 // remove the entries from RT
23                 removeAllCorrespondingEntries(lost-route,iCH);
24                 // Keep this tuple in the message
25             else
26                 // else iCH has at least one other route than through ourselves available
27                 // keep the entry,if CH remove this tuple from the message
28                 if is-clusterhead then l.remove(lost-tuple);
29             end
30         end
31     else
32         // else information is not interesting for downstream nodes
33         // no change in RT,if CH remove this tuple from the message
34         if is-clusterhead then l.remove(lost-tuple);
35     end
36 end
```

4.4 Symbiotic behavior

4.4.1 Higher-layer traffic

SLSR profits from the underlying SLSF dissemination, but also profits from the higher-layer traffic. The idea is to piggy-back routing related information with the payload data of the higher

Header Type	Fixed part	Variable part		
		multiplication factor	×	size per item
Basic SLSF header	16 bytes	nbr of ICRs	×	4 bytes
Fault Recovery header	4 bytes			
Routed Message header	10 bytes			
Current slaves header	9 bytes	nbr of slaves	×	4 bytes
Lost Slaves header	7 bytes	nbr of lost slaves	×	4 bytes
Lost Route to Clusters header	3 bytes	nbr of lost routes	×	5 bytes

Table 4.5: SLSF and SLSR Header size calculations (for 32 bits IP address sizes).

layers in a non intrusive manner for the performances. The payload messages given by the higher layers are received at the SLSR layer, here a decision is taken depending on the type of message using the Table 4.5:

If the message is a unicast: Route the message using SLSR. The message is composed of its payload, a *Routed message* header (Section 4.5) and the SLSF header (composed of both headers: *basic message* and *fault recovery*, see Sections 3.4.3 3.5.3). The maximum payload size that can be transported in one SLSR packet (with fault-recovery) is $1480-(16+1\times4)-4-10 = 1466 bytes$[25]. Note that for *Routed Messages*, the number of ICR nodes is always 1, so the SLSF header has a fixed size of 20 bytes. Payloads of more than 1466 bytes will be fragmented in two or more messages.

If the message is a broadcast: Routing is not necessary, thus SLSF suffices, however SLSR can profit from it. The message is composed of its payload and the SLSF header (composed of both headers: *basic message* and *fault recovery*). To profit from it, SLSR adds its own routing headers to a message only if the routing headers to be added plus the payload size is smaller than 1420 bytes: 1424 bytes (as shown in Section 3.4.4) - 4 bytes for the fault-recovery header. This ensures that adding SLSR headers in traffic messages does not induce additional messages and fragmentation.

4.4.2 Message timings and header combinations

Piggy-backing routing information in higher layer traffic reduces the number of messages in the network. A periodic routing information, such as *Current Slaves* sent every 5 seconds, included in a payload message resets the timer for 5 more seconds. Thus, if the broadcasted payload traffic is constant, routing information can be inserted every 5 seconds and thereby no additional message has to be sent. Similarly for event based messages, if at the moment the event occurs (e.g. lost a slave) higher layer traffic is available the routing header can be included in the traffic without and additional message.

Moreover, SLSR can use its own traffic to reduce the number of messages. The headers of SLSR, as detailed in Section 4.5, are independent from each other. The only dependency of SLSR headers is the basic SLSF header (Section 3.4.3). Therefore, if no higher-layer broadcast traffic is available, SLSR can combine more than one header per message. For example, a message with a *Lost Route to Clusters* header that has to be sent on the detection of a loss of route to a nearby clusterhead (event based) can include a *Current Slaves* header (periodic based). As

[25]for IPv4: (Ethernet maximum payload length 1500 - IPv4 header length 20)= 1480 bytes and for IPv6, without options and without the use of jumbo packets: (1500-40)=1460 bytes

explained in Section 4.3.4, *Lost Route to Clusters* messages are not forwarded throughout the network but only along routes that are impacted by this new information. In contrast, *Current Slaves* messages are to be sent on all possible routes to keep the routes up-to-date and valid (i.e. to avoid routing entry timeouts) for each clusterhead and slave. As a consequence, a message containing both *Current Slaves* and *Lost Route to Clusters* headers, can, if the *Lost Route to Clusters* part is not of any interest anymore, be truncated to a single header message with only *Current Slaves*. This is valid as well for headers piggy-backed in payload messages; deprecated headers are simply removed along the journey of the message. Figure 4.5 shows example of header combination of SLSR messages. Figure 4.5a) is a *Current Slaves* message with fault recovery. Figure 4.5b), shows an example of piggy-backed SLSR headers, here *Current Slaves* and *Lost Route to Clusters* headers are piggy-backed on a, upper-layer, payload message disseminated with SLSF. Finallt, Figure 4.5c) is an example of a routed payload message in SLSR.

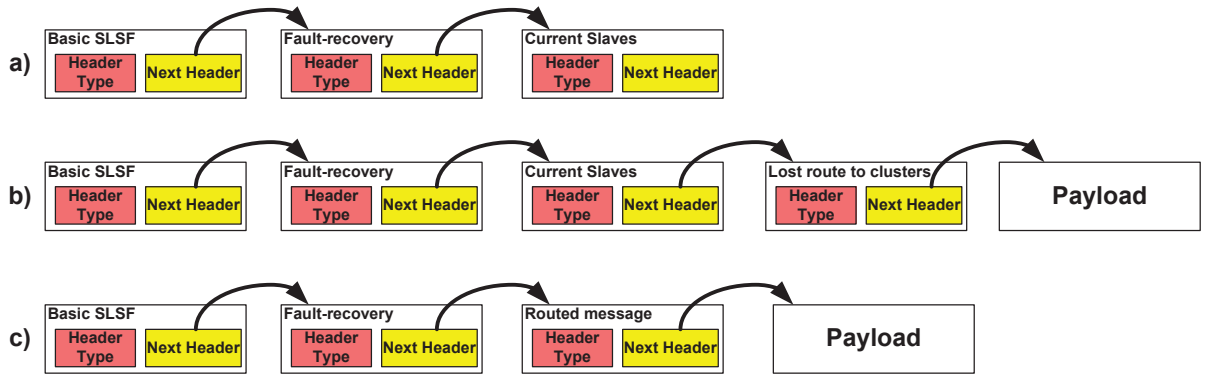


Figure 4.5: Example of SLSF and SLSR header combinations.

4.5 Message header formats

Following are the detailed header formats for SLSR. Each header is independent and can be combined with any of the other headers. SLSR headers rely on the basic SLSF header (Section 3.4.3). One basic SLSF header can be followed by several SLSR headers each profiting once processed of the information contained in the SLSF part.

4.5.1 Routed Message

The format of a header for a SLSR *Routed Message* is shown on Figures 4.6,4.7. This header is used by any node to unicast a message (using the SLSF structure) to a given destination. It contains the destination of the message and the next clusterhead towards the destination to which the message should be forwarded.

4.5.2 Current Slaves

Figures 4.8, 4.9 show the format of the *Current Slaves* advertisement message. The slave list can be exhaustive or only partial. If a list is too long for one message it can be dispatched in several smaller messages, this also is useful in the symbiotic usage of the bandwidth. As long as each slave is mentioned in a message within the *Message Validity Time* period the entries will remain valid. Slaves not advertised within the *Message Validity Time* will be deleted.

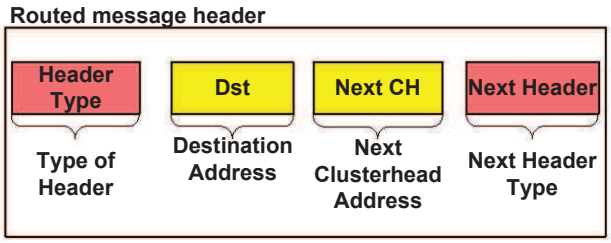


Figure 4.6: Header for a Routed Message (Graphical representation).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Header Type  |             Destination Address
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                |          Next Clusterhead Address
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                |  Next Header  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4.7: Header for a Routed Message (Textual representation).

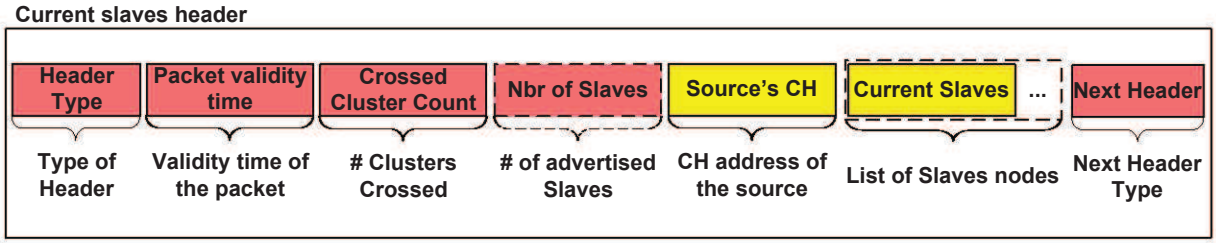


Figure 4.8: Format of the header for a Current Slaves message (Graphical representation).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Header Type  |Packet V. Time |Nbr Crossed CH | Nbr of Slaves |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Source Clusterhead Address                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Slave Address                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Slave Address                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              . . .                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |
+-+-+-+-+-+-+-+-+
```

Figure 4.9: Format of the header for a Current Slaves message (Textual representation).

4.5.3 Lost Slaves

Entries have a validity time obtained from the *Message Validity Time*, they automatically expire at the end of this duration. However, to be able to be more reactive on topology changes and quickly update the topology information, a message explicitly announcing the loss of a slave is proposed. Figures 4.10, 4.11 show the format of the *Lost Slaves* advertisement message.

4.5.4 Lost Route to Clusters

The *Lost Route to Clusters* message has two purposes. First, announce the loss of a cluster and thereby also invalidate all slave entries for that cluster. Second, prune routes to this clusters. Those messages are sent by nearby clusters that detect this loss. Figures 4.12, 4.13 depict the format of the *Lost Route to Clusters* message.

4.6 Advanced Configuration and option messages

SLSF and SLSR can be extended very easily with new headers. For example, to configure certain parameters such as the ICR validity time, or thresholds such as the stability threshold in NLWCA.

Configurations can be targeted to a specific path, destination or on a given distance from the source. Thereby simply using a configuration header combined with a *Routed Message* header applies the configuration only along the path to a specific destination. Similarly, configuring the *TTL* field of the *basic SLSF* header to a short distance provides a change of configuration only on a limited distance. To illustrate how extensions can be added to SLSF and SLSF, we propose in the following two extensions.

4.6.1 Fault recovery activation and deactivation

Fault recovery is an additional feature in SLSF. It is possible to enable or disable it completely (i.e. also the case for beacon fields concerning sequence numbers). To do so, an authoritative node sends an optional header containing the new configuration. This header would be used in very specific scenarios where an authoritative node is or can be designated to perform network configurations. Configuration messages emitted from this node are authenticated using, for example, a pre-shared key. The header would then contain: options for fault recovery or maybe only a flag whether or not to disable it, a specific sequence number to avoid replays of the message and the corresponding signature authenticating the other fields. An example of the format of such a header is given in Figure 4.14.

4.6.2 Extended forwarding of routing information

SLSF and SLSR handle duplicates considering only the source address and source sequence number (Section 3.4.1). Duplicates are never forwarded, except to discover several paths for a given destination, SLSR adds an exception to the handling rule as shown in Section 4.3.2. This behavior results in a propagation of routing information, such as a *Current Slaves* message, from a source S as shown in Figure 4.15. For any data except routing information, duplicate message are useless. For routing purposes, discovering the same information from a different clusterhead means another route is available. This situation only occurs when topological loops are formed. Limiting the routing information by discarding duplicates is good, but it can happen at the cost of unseen routes. Therefore, only for headers concerning routing information, duplicates could also be handled differently. Instead of only considering the source address and the sequence number, to

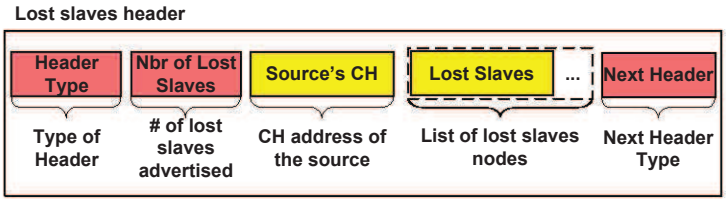


Figure 4.10: Format of the header for a Lost Slaves message (Graphical representation).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Header Type  |Nbr Lost Slaves|                 Source Cluster-
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
head Address                    |                 Lost Slave Add-
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
ress                            |                 Lost Slave Add-
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
ress                            |                    . . .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 . . .                          |  Next Header  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4.11: Format of the header for a Lost Slaves message (Textual representation).

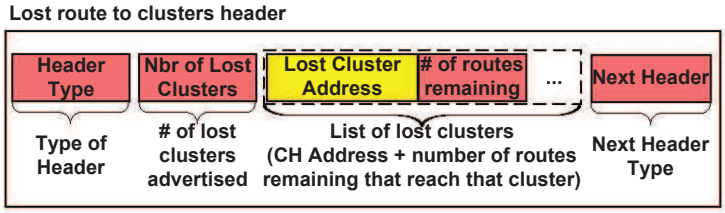


Figure 4.12: Format of the header for lost route to nearby clusters message (Graphical representation).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Header Type  |  Nbr Lost CH  |                     Lost CH Add-
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
ress                            |Nbr remaining R|    Lost CH Add-
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
ress                                            |Nbr remaining R|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                . . .                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     . . .     |  Next Header  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4.13: Format of the header for lost route to nearby clusters message (Textual representation).

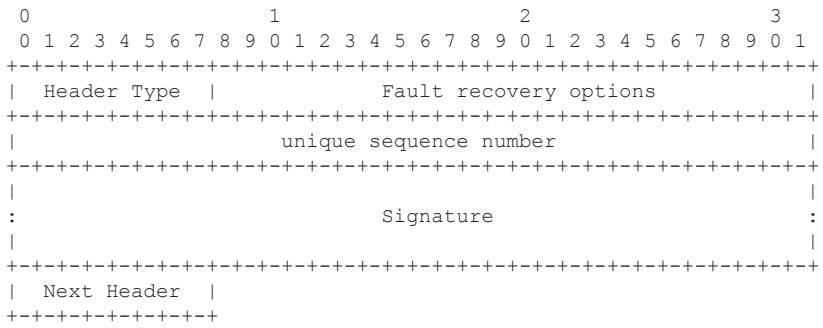


Figure 4.14: Format of the header for the fault-recovery optional message.

distinguish the same message incoming from different directions, we should additionally consider the Last CH field. Routing messages are now forwarded all along the loop as shown on Figure 4.15.

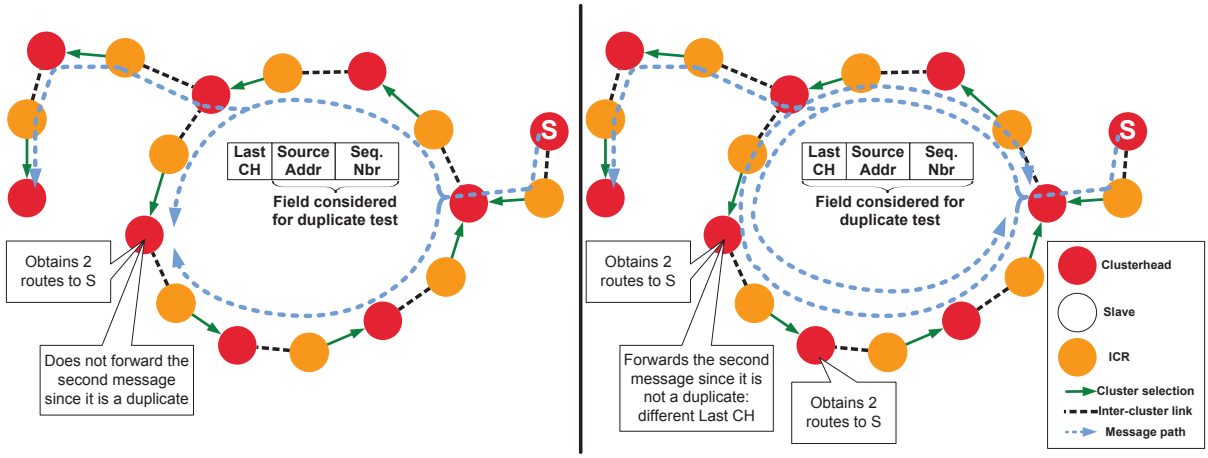


Figure 4.15: Forwarding and duplicate handling options of routing information in SLSR: left classic forwarding and right extended forwarding.

Depending on the context one could turn on or off this extended forwarding for routing information message. Unlike the Fault recovery activation/deactivation message, the extended forwarding behavior can be selectively activated or deactivated per source. The format of the header of such a message is proposed in Figure 4.16. Where F is an "on or off" flag, I is for infinity (i.e. until announced differently) and validity time is the time the behavior should be applied if not infinity.

4.6.3 Cross-layer design

The layered open systems interconnect (OSI) architecture divides the network protocol stack in 7 layers. Each layer can interact only with its adjacent layers through a given interface. Cross-layer designs are about making those separated OSI layers interact more and/or with non-adjacent layers. [Sriv 05] describes and classifies the different types of cross-layer designs that exist in 6 categories as shown on Figure 4.17. SLSR could be classified in the third category (Figure 4.17c) since it uses both upward and downward information.

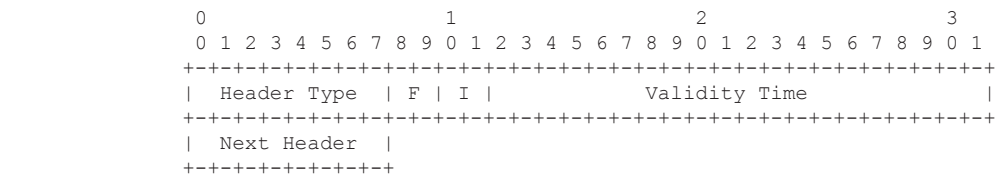


Figure 4.16: Format of the header for extended forwarding of routing information in SLSR.

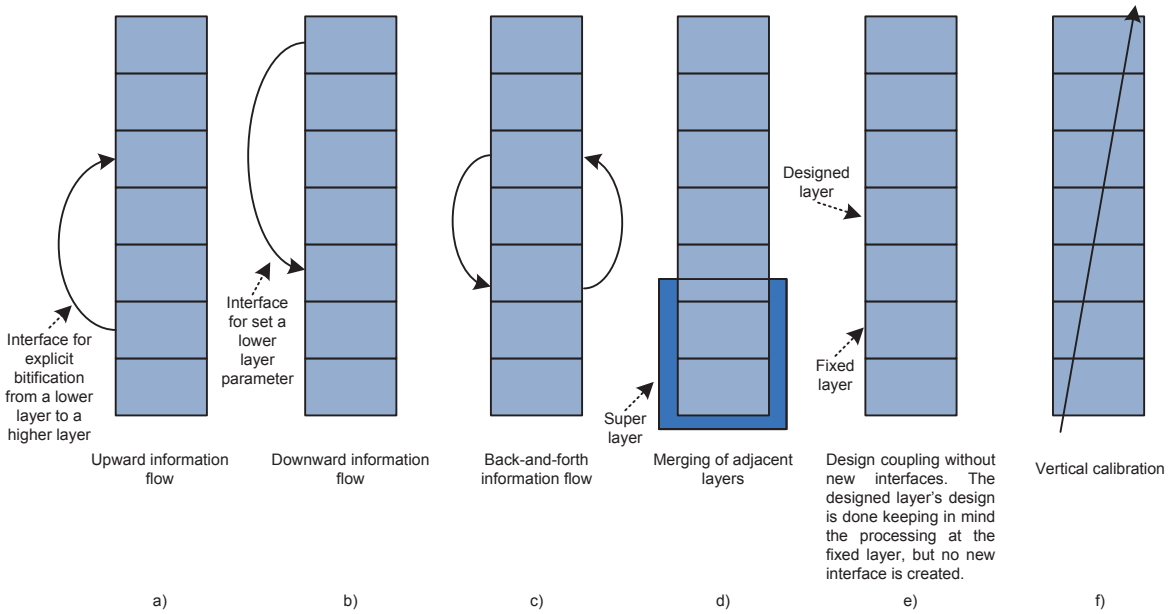


Figure 4.17: Cross-layer classification based on the layer interactions (Figure from [Sriv 05]).

In this paragraph, we shortly describe some of the advantages of using cross-layer downward information flow applied to SLSR. Higher layers could inform SLSR of the type of payload traffic that flows in the network. Doing so, SLSR can adapt its decisions and timings to this given context. For example, as shown in Chapter 7, Zeroconf profits of SLSF to propagate its service advertisements or other service related information. Service discovery traffic is not just any traffic and is to some extends flexible to delays and has a noticeable periodicity. SLSR could propose a lightweight message management mechanism to higher layers. Here, an application, protocol or any higher layer could pass its payload and specify a given maximum sending delay. In service discovery protocols, such as Zeroconf, advertisements are sent periodically. In Zeroconf they are sent at 80% of the record validity time. Zeroconf, could therefore also provide the information at already, for example, 70% of the validity time by informing SLSR that it is delay tolerant for 10% of the validity time. Thereby, SLSR, on a high traffic load, can schedule its own routing headers to be included and sent in the Zeroconf message at a later point in time and possibly when the traffic load is lower.

4.6.4 Full link-state behavior

If a full topology is required, a clusterhead in SLSR can be configured to advertise more than only its slaves and itself. By simply extending the scope of the routing advertisements to the

nearby clusterheads, SLSR becomes a link-state routing protocol. Doing so, SLSR obtains the complete cluster topology and the member nodes for each cluster. The topology information is complete at the cluster level but fuzzy at the slave level. The cluster-path is known but the inter-cluster paths are unknown.

To enable this features various possibilities exist. A first one would be to enable it for the whole network similarly as for fault recovery activation in Section 4.6.1. A second possibility would be to enable it locally depending on the context. Each clusterhead decides, based on its current context, to advertise nearby clusters or not. A third possibility could be to enable it only on clusterheads that are on an active or more important route. Here, a header with a validity time could be added similarly as in the extended forwarding Section 4.6.2.

4.7 Conclusion

SLSR provides a routing layer on top of SLSF. It takes advantage of the underlying cluster structure and provides the routing and unicast capabilities missing in SLSF. SLSR, instead of routing towards the next hop, routes messages towards the next clusterhead. SLSR paths remain unchanged, even if nodes on paths between clusters change while the clusters remain stable.

To obtain the routing paths, SLSR uses routing headers, in addition with the SLSF headers, that can use the higher-layer traffic or even its own traffic in a symbiotic way by including routing information in this traffic. When the broadcast traffic rate is high enough, SLSR can include its headers in the payload traffic, thus reduce the overhead at the cost of few additional bytes per message from time to time.

Moreover SLSR can be classified as a balanced-hybrid protocol and thereby reduces the routing overhead by using the features from both distance-vector and link-state. Overhead reduction is simply due to the fact that routing information shared in SLSR does not overlap. A node is only advertised once by its clusterhead.

SLSR has been implemented in the JANE simulator and tested in small simulation settings (as shown in Section 7.5.3). In the near future, we plan to evaluate and compare the routing performances and the routing overhead of SLSR with other routing protocols such as OLSR.

Chapter 5

Advanced Mobility Models using multi-modeling and co-simulation

Contents

5.1 JANE and other ad hoc network simulators 92
5.2 Ad hoc mobility models 93
5.3 Advanced mobility models 96
5.4 Case study 97
5.5 Multi-modeling and co-simulation motivation 97
5.6 The AA4MM meta-model and platform 98
5.7 Case study: MANETs and users' behaviors 100
5.7.1 Mobility modeling 100
5.7.2 The multiagent paradigm 100
5.7.3 User model description 100
5.7.4 Modeling network aware users 101
5.7.5 Synthesis 101
5.8 Experiments and Results 102
5.8.1 Building realistic usage scenarios 102
5.8.2 Network and user behavior mutual influences 103
5.8.3 Synthesis 104
5.9 Conclusion 105

Even with all the intensive research done in the domain of mobile ad hoc networks, ad hoc networks have not yet made their breakthrough with the average consumers. Only a small number of research projects or specific applications actually use ad hoc networks.

Therefore, to evaluate MANET protocols, applications or services researchers are left with two choices: testbed or simulation. A testbed is a real ad hoc network composed of real devices set up by researchers for their experimentation. Testbeds usually contain a limited number of devices but have the advantage of reflecting real world conditions. The limited number of devices is due to cost and management issues. Mobile ad hoc network require mobility of nodes. With 100 devices one needs 100 people or robots (to be able to replay scenarios) that move the devices around like the users would move according to the usage scenario. The inflexibility, often non reproducibility, lack of scalability and cost of testbeds make simulators a widely chosen alternative.

In this chapter we start by briefly presenting existing network simulators as well as the simulator used for our analysis. We present the existing mobility models that are used in most of those simulators. The existing approaches however lack in flexibility and in producing realistic user movements. Therefore, we propose advanced mobility models that can, using a multi-model and co-simulation framework, produce realistic mobility patterns grounded on sociological research.

5.1 JANE and other ad hoc network simulators

Several MANET simulators were considered for the experimentations presented in this document (Table 5.1). A detailed survey about MANET simulators can be found in [Hogi 06]. The choice of simulator was not only based on their popularity, the number of existing available protocols or their performance. The significant choice criteria were: to be able to easily change, create and design new protocols while also having the opportunity to add new features to the simulator itself (e.g. mobility models). As a result, we chose the Java Ad hoc Network Emulator (JANE) [Gorg 07]. Besides having those advantages, we had already some experience using JANE and a direct contact with a JANE expert (close to the developing team). Also, a significant advantage of JANE is that the simulation code (e.g. implemented protocols) can be carried directly onto a real device (e.g. PDA, Nokia N800). Thus, it drastically facilitates the step from simulation to testbed experiments.

JANE - Java Ad hoc Network Emulator

As described in [Gorg 07] JANE is a Java based development platform in which software development uses a bottom-up approach. Basic functionalities are implemented as elementary components that can be combined to more complex ones using well-defined interfaces. Reusability in different execution contexts is done by providing an appropriate machine abstraction.

Figure 5.1 shows the interactions between the simulated devices during the simulation. JANE distinguishes three service types: runtime service, simulation service, global service.

- **Runtime services:** Runtime services have one instance assigned to each mobile device. Those are usually used for protocols since they only can use information available locally on the device as a real device would.
- **Simulation services:** For design or test purposes there are simulation services. They also have one instance assigned per mobile node but additionally have the ability to communicate directly with neighboring devices by generating events on them. Simulation services have access to global simulation knowledge (e.g. exact device positions). They are useful when services are evaluated and some necessary services should not influence simulation results.
- **Global services:** The last type of service are global services where only one instance exists for all the simulation using a centralized approach rather than a distributed one, as in the simulation service. Those services also have access to global simulation knowledge. Purposes of global service are for visualization, global statistics, logging, etc.

JANE is composed of three layers (Figure 5.2).

- The first layer is composed of the platform core, the hybrid core, and the simulation core that provide JANE its versatility. It can function in classic simulation mode, using only virtual devices, in hybrid mode where real devices are connected to simulated devices and in platform mode using only real devices that then form a testbed.

Name	Interface Language	License	Popularity	Modularity	Comments
Jane [Gorg 07]	Java	Open source	+	+++	has a platform mode
ns-2 [ns 2]	C++ / OTCL	Open source	++++	++	well documented, implemented protocols, TQ (scripts)
ns-3 [ns 3]	C++ / Python Bindings	Open source	++	++++	scripts and modifications can be done in C++ or using Python bindings
MadHoc [Madh]	Java	Open source	+	+++	focused on mobility of nodes rather than protocols
Omnet++ [OMNE]	C++	Free for academic and educational use	++	+	very big and complex simulator
J-Sim [J SIM]	Java	Open source	+	++	provides the same possibilities as the Simula language
Opnet [Opne]	C	Commercial	++	+	proposes simulation and analysis of a broad range of wireless networks
Qualnet [Qual]	Parsec	Commercial	++	++	based on the GloMoSim simulator
GloMoSim [Zeng 98]	Parsec	Open source	++	++	uses the parallel discrete-event simulation capability provided by Parsec

Table 5.1: MANET simulators

- The second layer is composed of the JANE operating system, providing the timer system, the service manager, the service interaction, the execution manager and the simulation knowledge which are not used in platform mode.
- The last layer are the services developed in JANE (e.g. routing protocol).

5.2 Ad hoc mobility models

To simulate the mobile part of ad hoc networks, simulators use mobility models that define the movement pattern of the nodes during the simulation. Mobility models provide the simulator with the location (e.g. two dimensional coordinates) and the speed (e.g. meters per second) of each node in the simulation. Similar to the reason why using a simulator instead of a testbed, mobility models are mostly computed rather than extracted from real mobility trace since no significant number of traces are available due to the very low number of MANET deployments. Thus, the mobility models are computed using different techniques that can be classified

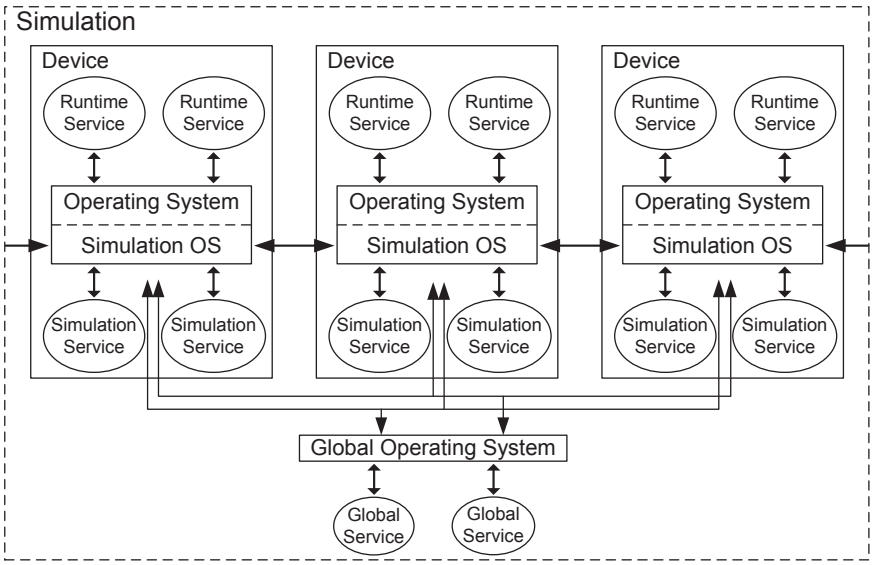


Figure 5.1: Service interactions in JANE.

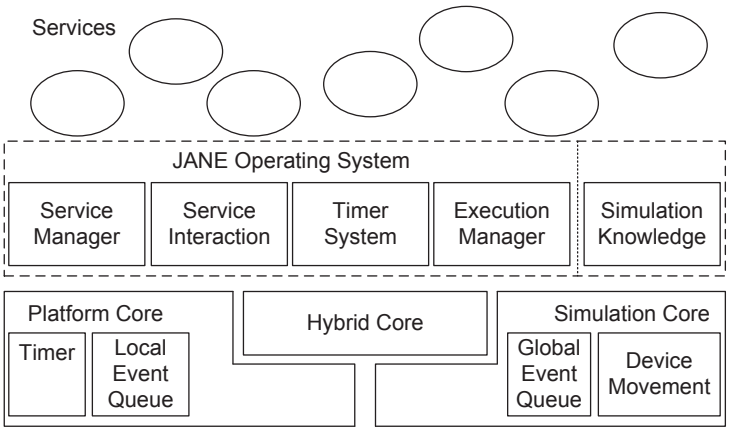


Figure 5.2: The JANE operating system.

according to [Bai 04] in four categories: Random models, models with temporal dependency, models with spatial dependency and models with geographical restriction. To those categories an emerged fifth one can be added: models inspired from sociological research. Comprehensive details about the different mobility models for ad hoc networks can be found in [Bai 04] and [Camp 02]. Furthermore, in this chapter, we propose a completely new sixth category: mobility models with a closed interaction loop between the behavior (mobility) model and the network model. In our approach, the mobility or behavior model can react to network events (e.g. connections, service availability, etc.) and the network events themselves depend from the mobility of the nodes, thus creating a closed loop.

- **Random models**: The simplicity of random based models has made them very popular for evaluation of ad hoc networks. A lot of different random model variations exist. Most of them are a variation of the well-known Random Waypoint mobility model [Broc 98]. In the random waypoint model each node randomly selects a destination inside the simulation field and randomly selects a uniformly distributed speed between $[minspeed, maxspeed]$ at

which it travels towards the destination. As a result, each node in the simulation independently selects a destination and a travel speed. Each time a node reaches a destination, it waits for a specified pause time (e.g. a pause time of zero provides a continuous movement). When the pause time expires, the same process starts again.

The random waypoint suffers from a lack of stability, since as shown in [Yoon 03], the average nodal speed consistently decreases over time. This speed decaying problem is due to the fact that nodes that chose a long travel distance with a very slow speed (depending on *minspeed* value the speed can be 0) tend to stay "trapped" on those long journeys and might not arrive at destination before the simulation ends. Since the speed and destination are chosen randomly after each pause time, more and more nodes during the simulation can be "trapped" on those long journeys, thus decreasing the average speed value. To solve this speed decaying problem, authors in [Lin 04] propose to change the speed distribution function for all the steps following the first one.

- **Models with temporal dependency**: To obtain more realistic mobility of nodes, temporal dependency proposes to make the next change of direction and speed dependent on the previous direction and speed. The memoryless nature of random models makes them unable to provide smooth and more realistic transitions between each steps. An example of such a temporal dependent mobility model is the Gauss-Markov mobility model. Here, each node selects at constant time intervals the next speed and direction based on the current speed, direction and location with an α tuning parameter that determines the randomness of this decision (i.e. if $\alpha = 0$: results in a completely random movement (Brownian motion) and if $\alpha = 1$: results in a linear motion).

- **Models with spatial dependency**: To provide more realistic movements, spatial dependent models are able to model groups of nodes moving together based on their relative positions in space. For example the Reference Point Group mobility model[Hong 99] divides the nodes in groups. Each group has a center, either a spatial center or a group leader node center. Each member of a group then travels relative to the group center with a uniformly distributed random motion vector that selects the next deviation from that center position while staying inside the group boundaries (Figure 5.3).

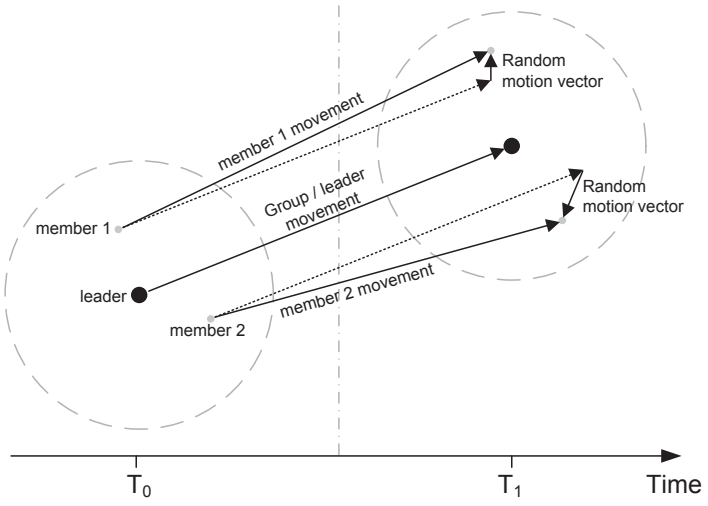


Figure 5.3: Example of node movement of one group in the Reference Point Group Mobility Model from time T_0 to T_1.

- **Models with geographical restrictions**: To take into account the physical environment of the nodes into the mobility model, models propose to add restrictions on the movement possibilities of the nodes in their simulation environment. Those restrictions are, for example, in the Pathway mobility model or in the Restricted Random waypoint [Blav 02] modeled by a graph where vertices represent the buildings and edges the streets connecting those buildings together. Nodes travel following the streets (edges) towards their destination building (vertice) through the shortest path along the edges. On arrival at their destination a node pauses for a specified time and randomly selects the next destination building. Another geographically restricted mobility model is the obstacle mobility model where obstacles are represented by rectangular boxes on the simulation area. Nodes adapt their trajectories accordingly to avoid them. Trajectories to avoid obstacles are pre-calculated, thus the obstacle model is very similar to the pathway model.

- **Models inspired from sociological research**: In those models, sociological research or observations are considered to generate mobility models. In [Muso 06], research based on social networks is used to produce a two stage process. At the first stage the social connections are defined by an interaction matrix extracted from social network studies. The second stage consists in dividing the nodes at the topological level (hence the position in the simulation) into groups. Nodes then follow their group similar to the Reference Point Group Mobility Model. However, each time a goal or destination is reached the nodes decide based on the interaction matrix whether to stay in the group, to change group or to move out of any group.

  Another mobility model based on social observations is [Maed 05]. They extract the behavior patterns for specific target zones by observing, through fix points (e.g. a webcam), the number of pedestrians flows and densities for that zone. Using linear programming techniques they calculate the average flow per unit per time in the target zone. With the derived flow rates, they generate a UPF (Urban Pedestrian Flow) scenario as mobility model for network simulators.

  Another approach in [Nguy 11], abstracts the spatio-temporal correlation of human mobility using preferential attachment to favorite locations. The model is able to reflect similar routing performances than the one obtained from real mobility traces.

The lack of existing ad hoc deployment and real user traces motivated the research on mobility models. The first ones where very basic like the random waypoint and very easy to compute. The following up added new dependency to obtain more realistic movements. Finally, a model inspired by sociological research compensate that lack of ad hoc traces by using the social interactions as basis. This is exactly the right way to go since sociological research is all about pedestrian movement analysis, social interactions, etc.

5.3 Advanced mobility models

Several scientific domains (network, AI, physics, sociology, ...) are involved in the field of MANETs and more generally Ubiquitous Computing [Weis 95]. Each domain has its own vocabulary, habits, needs and culture. No single universal model exists, to deal with the interacting complex models of ubiquitous computing. Experience in ubiquitous systems demonstrates that advanced research in such a complex topic cannot be pursued by only broadening an initial domain with, unavoidably, partial knowledge from others. A typical example is the design of mobile services where the user carries devices that contribute to the delivery of data to other users. In this case, the behavior of the users (e.g. their mobility patterns) in a crowd highly

impacts the overall operation of the service and thus need to be considered early in the service design.

Therefore, in a close collaboration with Julien Siebert and Vincent Chevrier, we propose a methodology and the use of a novel distributed framework to design, implement and assess MANET communication related technologies [Lecl 10d]. Our solution is built on two key elements: model interaction and multi-simulation engine.

First, our approach enables the combined use of reference models and simulators coming from different specific domains (figure 5.4). Through a simple interface implemented for each simulator, the used framework eases the interactions among both models and simulators. This significantly improves the initial design of the EXiST (EXperImental Simulation Tool [Ciar 01]) co-simulator by both providing decentralization support and a better formalization.

5.4 Case study

In the domain of ad hoc networks and more generally wireless technologies, mesh networks, routing protocols, or ubiquitous services are often studied (designed, experimented, assessed) using network simulators. Indeed, real world experimentations with a representing set of devices are excessively time and money consuming, especially in the case of ad hoc networks or large scale peer-to-peer environments. It is even of little scientific relevance since reproducing a scenario/an experiment is not possible due to the ever changing experimental conditions. Therefore, a lot of models and simulators have been developed in the field of ubiquitous computing over time [Naic 06, Kurk 05]. They aim at simulating the network layers in (more or less) details and indeed most of them are not designed for doing more, like for example advanced node dynamics, or users goals. In fact, in ubiquitous computing, one key element of the equation is "the human" and more specifically his behavior.

As a case study, to demonstrate our approach we focus on mobility in MANETs. As shown in the previous section, most mobility models are computed by merely considering the user as a random walker without goal or decision process, and without any knowledge of how the network actually behaves. Unfortunately, this is generally considered sufficient to give the system its "dynamic" characteristic and, therefore, is used to prove the validity and demonstrate the performance of protocols which later diverge when deployed in the real world. Our approach circumvents this limitation.

As a proof of concept, we combine two existing simulators: a mobility simulator (based upon a multiagent model) and a network simulator (JANE). By doing this, we combine sociological research achieved in urban simulation community with network research. Our experiments in Section 5.8 show the possibilities the framework offers and also the importance of the mutual influences between the network and user behavior. We believe that the originality of our approach is to allow to close the loop between the users behaviors and their mobile ubiquitous environment.

5.5 Multi-modeling and co-simulation motivation

Our approach builds on the use of multi-modeling and co-simulation in order to take into account both users' behaviors and network performance within an integrated study. The used framework offers a way for protocol and service designers to get a "bigger picture" early in the design phase.

As stated in the introduction, we argue that the study and the design of mobile ubiquitous applications cannot be achieved efficiently by taking into account only one point of view. By point of view, we mean the physical medium aspects, the network aspects (protocols, services, messages, topology...), the users' behavior aspect (mobility, sharing resources ...), etc. (figure 5.4). Depending on the study (and questions asked), omitting some of these points of view

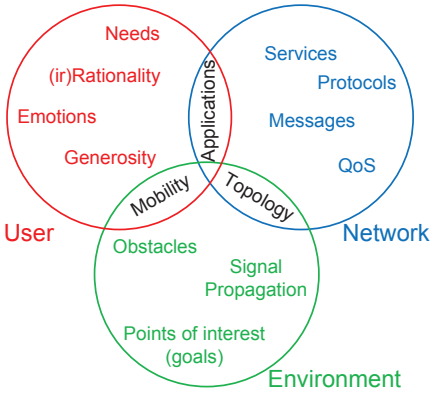


Figure 5.4: Abstraction levels.

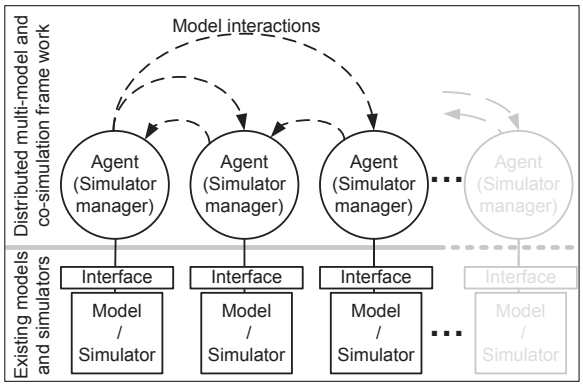


Figure 5.5: Framework model interactions.

may lead to non-significant simulation results. For example the authors of [Hami 09] show the impact of taking different physical medium models for the wireless communication into account. Moreover as many models and simulators already exist and have been validated, reusing them is the best approach.

We propose to use multi-modeling (e.g. using different interacting models) and co-simulation in order to represent all the different aspects (or point of view) needed for the simulation to be more significant. We rely upon a meta-model and a framework, proposed by Julien Siebert, called AA4MM (Agent and Artefact for Multiple Models [Sieb 10]), which allows us to couple different existing models and simulators in order to build a more complex and more accurate simulation. These simulations are used, on one hand, to evaluate protocols and services against different usage scenarios and, on the other hand, to design new protocols and services by taking into account some global usage scenarios (e.g. the user behavior and the environment parameters). By co-simulation or multi-simulation, we mean the ability to combine multiple simulators and/or real implementations (prototype, software and/or hardware) at the same time.

The main advantage of this approach is to achieve a good separation of concerns. Computer network scientist and designers only focus on the network aspect (protocol and services definitions, network parameters: radius, bandwidth, latencies...) and cognitive and human scientists focus on the user behavior modeling (mobility, user needs, (ir)rationality). The whole simulation efficiency is our main limitation. Reusing existing simulators - that may not have been designed for distributed simulations - may be less efficient in terms of execution times than a single multi-model implemented in a natively distributed simulator. However, we consider that the advantages brought by the separation of concerns are conceptually more important and that the simulation efficiency is a technical question that can be targeted later.

5.6 The AA4MM meta-model and platform

We rely on a multi-modeling platform called AA4MM (Agent and Artefact for Multiple Models [Sieb 10]). Its design is inspired by HLA (High Level Architecture)[Elec 00] which main goal is re-usability and interoperability of different simulators, but uses the principles behind simpler tools such as EXiST or MSI (the Multi Simulation Interface [26]).

AA4MM is based upon the work made in complex systems modeling and multiagent community. It allows us to take existing simulators and make them interact in a decentralized and distributed fashion, much like the RTI (Run Time Infrastructure) in HLA, but completely dis-

[26] `http://msi.sourceforge.net/`

tributed. The platform's role is (also) to manage the synchronization of simulators (causality constraints) and the semantic coherence between models. It inherits also from the component-based software engineering world in order to be the more transparent possible.

As a consequence, it is relatively simple to add a model and its simulator with a very limited set of modifications (the least set of the following functions should be implemented, Table 5.2). The interactions problematic (e.g. causality constrains, simulators synchronization and semantic coherence, model compatibilities) are managed by the AA4MM platform itself by creating entities external to the original tools. This presents some advantages: it is easy to reuse existing models and simulators (no need of decentralized simulations knowledge). The few modifications brought to the models and simulators allow us to use them either as a stand-alone application or inside the multi-simulation.

	Functions	Description
1	Initialization	Initialize, passing parameters, etc.
2	Model execution	Execute 1 step of the model (e.g. execute 1 simulation step, 1 simulation event or a given time interval)
3	Get simulation time	Obtain current simulation time
4	Data input	Provide data to the simulator (e.g. input information from another simulator)
5	Data output	Retrieve data from the simulator (e.g. output information going to another simulator or for logging purposes)
6	(Optional) Finalize simulation	Finalize simulation after last step (e.g. retrieve logs from / execute logging scripts)

Table 5.2: Interface to define for a simulator to work within the AA4MM platform.

Figure 5.5 describes the composition of the AA4MM framework. Each simulator is controlled by a simulator manager (formally an agent) which is an autonomous entity. All these manager agents cooperate in order to run the whole simulation and to take care about the interaction problematics. To make different simulators interact (in the AA4MM platform), the following steps are required:

- define a simulator interface (one for each simulator): implement 6 basic functions as described in table 5.2 directly from the source code or from the api, or more laborious by extracting an api if only binaries are available; this is the only modification to make.

- create the specific AA4MM entities (outside the simulator and model):

  - for each simulator, create an entity (called an agent (Simulator Manager)) in order to manage the simulator (input/output data flows, model execution and simulation time management).

  - for each link between the simulators, create an entity in charge of the data flow exchange.

The whole platform is sustained by a distributed simulation algorithm (a conservative one) that allows discrete events, step by step or continuous time based simulators to interact (and stay synchronized [Sieb 09]).

The framework uses a series of XML configuration files that allow for the simple description and tweaking of the different simulators involved and of the global simulation.

5.7 Case study: MANETs and users' behaviors

5.7.1 Mobility modeling

As shown in Section 5.2 there are many ways to model the different types of mobility, each with their own specificities. There is no formal model combining some of those classical models. And most critically, none ever considered any feedback (for example, closing the loop between the users' and the underlying network behavior). Our work allows both, by proposing to use the agent paradigm as a unique tool for modeling the largest and various sort of mobility.

5.7.2 The multiagent paradigm

The multiagent paradigm is a way to model sets of autonomous interacting entities within an environment. It is a well-known paradigm used in human sciences, ecology or in robotics. It describes the systems into (at least) these different components: agents, environment, interactions. The agents are autonomous and proactive entities, situated in an environment. They only have a partial (local) view of it and decide which action to take dealing with their own perceptions and reasoning.

MABS (Multiagent Based Simulation) [Davi 00] offers us the right level of description when we want to model users' behavior, goals and actions. Instead of using a global equation to model users' trajectories, we can, via the agent based model, re-create the way users move. It means that we can directly model behaviors such as "if an obstacle is present in front of you, then avoid it" or "reach a goal, stay nearby during five minutes and then go".

More generally, with this approach, we can model more complex behaviors such as willingness to use and share a service depending on the bandwidth consumed or the generosity of a user ; or the reaction to unpredictable events.

Mobility has already been studied and modeled via the multiagent paradigm. Here an agent can describe human, animal or robots. In [Reyn 87], Craig Reynolds worked on bird flocks modeling where each agent tries to stay inside the flocks only by computing a small set of forces (Boids). Individual-based pedestrian modeling is also used in urban simulations ([Helb 05, Tekn 00, Gaud 07]). This paradigm is also used to model crowd scenes in movies (as battlefields in Lord of the Ring) and implemented in video animation software such as MASSIVE[27].

5.7.3 User model description

Our agent-based mobility model is inspired from urban research and pedestrian modeling [Gaud 07, Helb 05], but can also model classic mobility behaviors (e.g. random waypoint). Each mobile node (a user) is represented by an agent (named a_i). The agent behavior can be seen as a combination of simple behaviors resulting in a complex one. For example random waypoint is implemented as the simple following rule: every time period, each agent changes its direction. More complex behaviors such as obstacle avoidance, flocking or goal attraction are modeled as a function, a sum of forces, resulting in a node movement. Each force/behavior describes an

[27] `http://www.massivesoftware.com`

interaction of the agent with its environment and the other agents. The agent has a limited perception (figure 5.6): these interactions are only effective on the neighborhood of the agent. In our case, the movements of the agents are computed by applying laws of mechanics: namely point kinematics. These models are easily extensible, easy to implement and can express a large set of behaviors by weighing each force. The examples below (figures 5.7,5.8 and 5.9) depict force oriented behaviors.

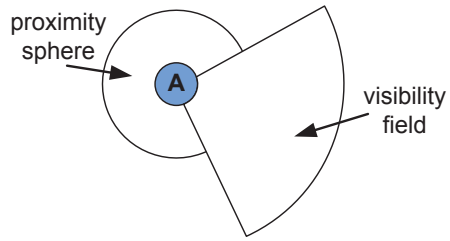


Figure 5.6: Perceptions of an agent (a user)

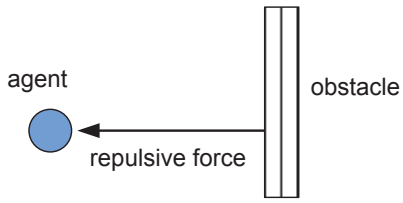


Figure 5.7: Repulsive force for obstacle avoidance model.

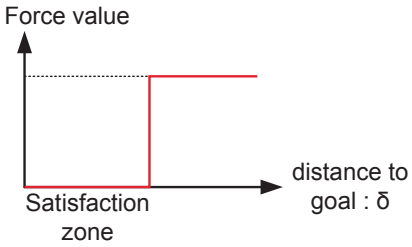


Figure 5.8: Attractive force to the goal.

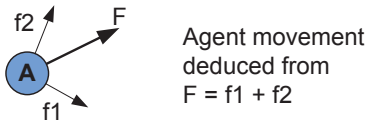


Figure 5.9: Movement computation.

5.7.4 Modeling network aware users

Integrating network aspects into the agent decision process is achieved easily and is straightforward. Indeed, once the agent perceived the network information (e.g. connectivity presence/loss, quality of services), a simple rule defines its reaction. For example in Section 5.8 we describe users that slow their speed or stop moving when they perceive good connectivity.

5.7.5 Synthesis

This model respects the constraints cited in [Bai 04]: temporal dependency, spatial dependency and geographical dependency. Describing sophisticated movements is straightforward: for example from our two simple movements we have nodes avoiding obstacles and following multiple succeeding goals. Moreover, we can easily model mobility of groups of people just by adding a force that attracts agents that go in the same direction (as shown in Section 5.8).

We have developed a set of mobility models, from simple random waypoint or restricted random waypoint to advanced particles engine, flocking or explorer behavior, that are fully parameterized. Using multiagent based simulation (MABS) to simulate basic behaviors, such as random waypoints, seems probably overkill at first.

However, since this modeling approach is individual-based, we can easily tune each behavior and describe heterogeneous ones. Indeed, the highest level of granularity can be reached by implementing a different model of behavior per agent. Thus, we can describe, for example,

different kinds and mixes of populations. Finally, with our approach, a user can dynamically switch from one behavior to another.

5.8 Experiments and Results

As a proof-of-concept of our vision and framework we coupled a user behaviors simulator, that we extended with an implementation of ad hoc protocols, named MASDYNE (MultiAgent Simulator of DYnamic Network usErs) with the JANE simulator.

The goals of the following experiments are on one hand, to show the simplicity of a realistic usage scenario design and implementation, and on the other hand, to show the effects of having interactions between the user behavior model and the network model.

5.8.1 Building realistic usage scenarios

The goal of the first experiment is to obtain a mobility scenario (Figure 5.10): A group of students visiting a museum. This scenario fits with our goal to test and deploy, in the future, an ad hoc network within a museum (ANR SARAH project).

This mobility model is based upon force-oriented behaviors: the users' behavior, the interaction between the users and the environment are represented by simple forces. For this scenario, we use four simple force oriented behaviors: *goal force, avoid walls, repulsive force, attraction force*. This provides us the following scenario: Agent 1, a tour guide, follows goals unknown to the students. The other agents, the students, follow agent 1. The combination of these simple behaviors, done by summing the forces, results in a complex and more realistic behavior.

- All Agents have a Goal: e.g. the students follow Agent 1, if visible.
- All Agents avoid walls: Repulsive force from the walls.
- Agents have a repulsive force from each other (comfort zone).
- Agents are attracted by other agents that go in the same direction.

Figure 5.10 show this usage scenario in different environments (e.g. corridor, crossroad, doors, museum). Parameterization of the model is done according to [Helb 05]. We observe that, in different environments, the student group is clearly following the Tour guide even when walls are involved.

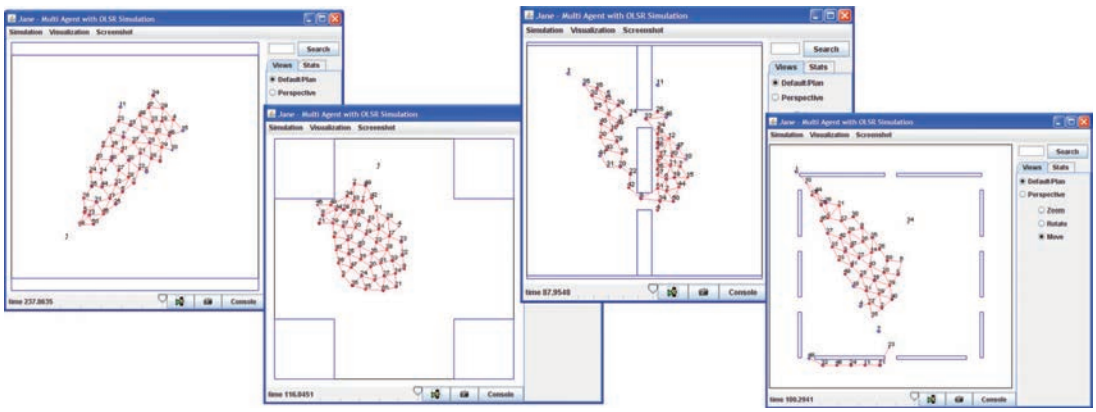


Figure 5.10: Museum visit example in multiple environments.

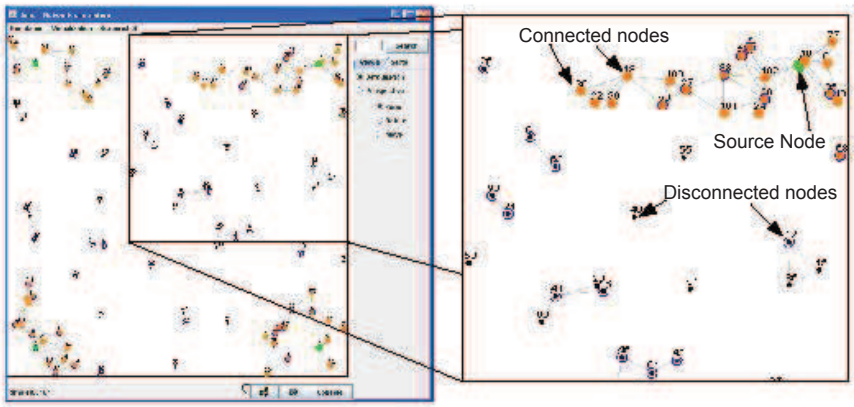


Figure 5.11: 4 source nodes: green nodes, 100 moving nodes: black when disconnected, orange when connected to a source.

5.8.2 Network and user behavior mutual influences

Even the simplest network protocol such as a broadcast obviously would perform very well if the students seriously follow the rules. However, what happens if not all the students follow the rules and for example react to network events (e.g. network connectivity). The AA4MM framework allows us to respond to this kind of questions. To show the effects of these mutual influences, the second experiment is a scenario with network feedback (Figure 5.11). The aim of this experiment is not so much about being realistic but more about showing the possibilities offered by our approach.

Experimental protocol:

The usage scenario is the following: 4 source nodes (access points) are placed in every corner of a place. 100 nodes/users want to connect to a source. To keep it simple to explain and to avoid errors or bias induced by the algorithm of a protocol, we used a basic flooding algorithm that simply rebroadcasts every non seen message. Messages already seen are silently discarded.

The users' behavior is the following. At the beginning, the user moves randomly. He is aware of its connection status: connected or not connected. Then we propose 3 basic behaviors: continue to move randomly (user is not aware of network feedback or does not care), slowly speed down (user continues walking but only slowly), stop (user sits down to enjoy the connection). Figure 5.12 depicts the interactions between both simulators.

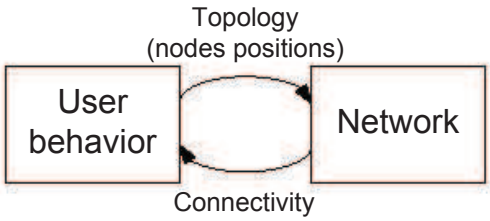


Figure 5.12: Interactions between MASDYNE and JANE.

We measured the evolution of the percentage of connected nodes. For each experiment we used 2 behaviors. We vary the percentage of agents having each behavior. For example, in figure

5.13 the curve marked 40% means that 40% of the agents implement the stop behavior while the remaining 60% implement the random behavior. Each experiment was done using 50 distribution seeds.

Results:

We observe that the more the users slow down or stop, the better the connectivity rate is. The stopping nodes create a sort of backbone for the other nodes while the backbone created by slowing nodes is only temporary until the nodes move out of range. In other words, the more users participate and follow the "stopping" and/or "slowing" guidelines, the better the network is, even if the protocol itself has not changed at all.

In figure 5.13n we increased the percentage of nodes stopping, while the remaining ones continue to move randomly. With 100% stopping nodes, after 60 seconds all nodes reached an access point. With already 60% of nodes having stopped, 80% of the nodes were connected. In figure 5.14, the nodes divide their speed by 6, when connected. Again we increased the percentage of nodes slowing down while remaining nodes move randomly. Compared to stopping nodes, performances were worse. In figure 5.15 and 5.16 every node reacts on connection. We varied the percentage of stopping nodes while the remaining nodes divided their speed. We observed that the results significantly differed when using random waypoint model or more complex usage scenario.

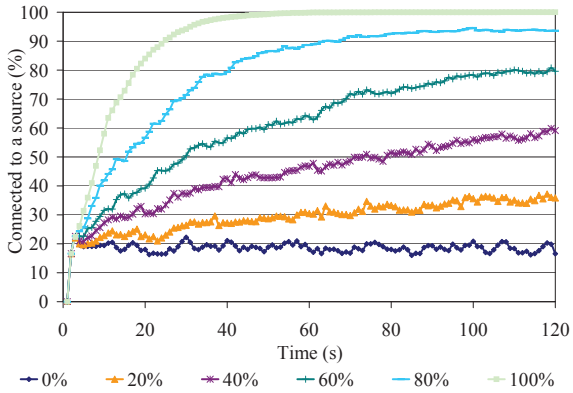


Figure 5.13: Percentage of nodes stopping when connected, remaining nodes move randomly.

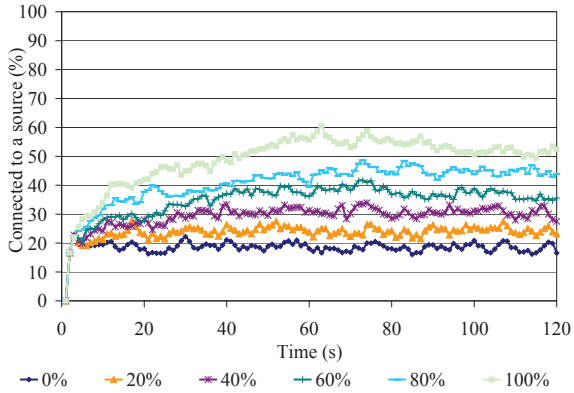


Figure 5.14: Percentage of nodes that divide their speed by 6 when connected, remaining nodes move randomly.

5.8.3 Synthesis

These results show that our approach has: 1) the ability to take mutual influences of users' behaviors and network performances into account; 2) the ability to design usage scenarios with heterogeneous users behaviors; 3) the ability to benchmark a network protocol against a wide range of usage scenarios.

In order to consider this work from a higher standpoint, we do not assume that the users behaviors will always be predictable. But, with this approach, we are able to predict that if only a percentage of the users behave like we predict, then the network performances will be better or worse.

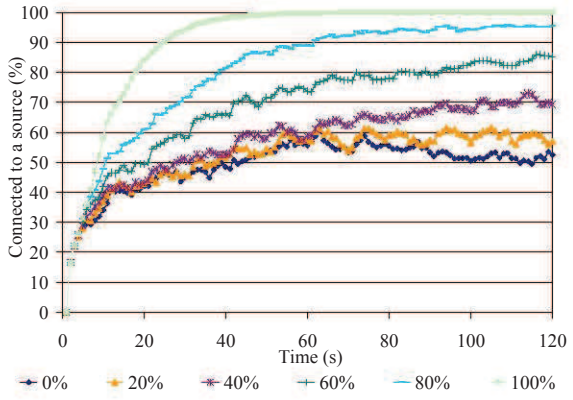


Figure 5.15: Percentage of nodes stopping when connected, remaining nodes divide their speed by 6 when connected.

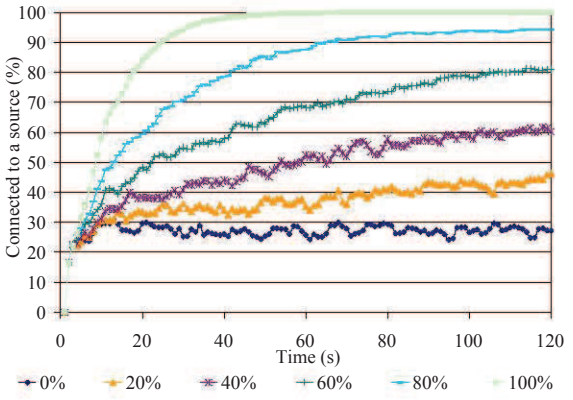


Figure 5.16: Percentage of nodes stopping when connected, remaining nodes divide their speed by 3 when connected.

5.9 Conclusion

Mobility modeling is a key point in evaluating wireless technologies and services. This chapter showed the lack of classical modeling approaches in their ability to take into account the users' behaviors and their interactions with the network performances. The conceptual framework and a prototype implementation where presented. Using our multi-modeling approach, different existing models can easily be coupled in a loose and generic way.

We argued and showed that multiagent system not only provides the usual mobility models, but also that it is very simple to design, fine-tune or design new ones. Multiagent allows the description of heterogeneous behaviors. The new mobility models can take into account networks or, more generally, environment inputs, thereby having a closed-loop system where something closer to the "human behavior and real-life" is considered.

Our approach offers a basis for valid comparison of wireless technologies and services but can also be extended to any dynamic environment, such as P2P networks for example. It fits very well for situations with interactions between the users, the networking and the physical environments.

Our experiments demonstrate that closing the loop leads to new ways of evaluating technologies. Even a basic protocol, such as our flooding example, can have strong performances if the users follow their directives. Some can be difficult to enforce (complete stop when detecting a connection), but other could be reasonable in a real world (slowing down).

Chapter 6

Collaborative Filtering

Contents

6.1 A Delay-Tolerant CF 108
6.1.1 User Similarity 108
6.1.2 Clustering Algorithm 108
6.1.3 Delay-Tolerant Intra-Cluster Broadcast 108
6.2 Experiments and Results 111
6.2.1 Dataset 111
6.2.2 Evaluation Criteria 111
6.2.3 Simulation Settings 111
6.2.4 Results 112
6.3 Conclusion 113

This chapter presents a delay tolerant collaborative filtering algorithm. The evaluations of this chapter are based on a movie data set that contains 100,000 ratings for 1682 movies by 943 users. We used this data set because it is recognized by the community and has consistent data. However, as described later on in section 7.3, we plan to apply collaborative filtering techniques on service information. As consequence, instead of rated movies, we have services rated by users. This benefits us especially if the number of services and users in a network grows.

As presented in Section 2.6.2, recommender systems using collaborative filtering are a well-established technique to overcome information overload in today's digital society. Currently, predominant collaborative filtering systems mostly depend on huge centralized databases to store user preferences and furthermore are only available when connected to the Internet. In a collaboration with Patrick Gratz [Grat 09], we considered an incremental recommender system for highly dynamic mobile environments where no central global knowledge is available and communication links are rather unreliable in comparison to static networks. The algorithm aims to reach a reasonable prediction coverage and accuracy while keeping the amount of additional network overhead as small as possible, maximizing the performance of our system. For this purpose, the algorithm is based on a delay-tolerant broadcasting mechanism on top of a weighted cluster topology, namely NLWCA, presented in Section 1.2.4.

Evaluation results show that in terms of accuracy and coverage, the results of the presented algorithm converge with those obtained from a global knowledge scenario, even in the case of message loss.

6.1 A Delay-Tolerant CF

Based on the work introduced in [Grat 08], our collaborative filtering system can be roughly divided into three phases: (1) requesting similar neighbor information, (2) updating the local recommender model and (3) a subsequent prediction calculation based on this local model. In addition, on each clusterhead, there are two preceding phases: (p1) the retrieval of unknown profile information and (p2) the subsequent aggregation and provision of similar neighbor profiles. In the following section we introduce a suitable metrics to determine the similarity between two user profiles and describe how the underlying cluster topology is created. Afterwards, we present our delay-tolerant intra-cluster based broadcasting mechanism for the retrieval, aggregation and provision of similar neighbor information.

6.1.1 User Similarity

In order to calculate the similarity w between an active user a and a neighbor u we used the Pearson correlation coefficient which is defined as follows:

$$w_{a,u} = \frac{\sum_{i=1}^{m}(r_{a,i} - \bar{r}_a) * (r_{u,i} - \bar{r}_u)}{\sigma_a * \sigma_u}.$$

Where $r_{u,i}$ is the rating for item i given by user u, $\bar{r}$ is the average rating and σ the covariance. However, one of the issue of this metric is that highly correlated neighbors are often based on a tiny number of co-rated items. To overcome this issue we applied a correlation significance weighting as introduced and discussed in [Herl 99].

6.1.2 Clustering Algorithm

Our algorithm is based on stable weighted cluster topologies generated by the Node and Link Weighted Clustering Algorithm (NLWCA) as presented in Section 1.2.4. The subhead avoidance rule as described in Section 3.3 ensures the clusters to be strictly one-hop clusters. This drastically simplified the algorithm compared to [Grat 08].

6.1.3 Delay-Tolerant Intra-Cluster Broadcast

On top of the above mentioned topology, our algorithm works as follows. Each clusterhead communicates exclusively with its stable slaves assigned by the described clustering algorithm. In order to detect and exchange only relevant profiles, additional beacon information is used. For this purpose, we extended the beacon of a slave device by the following information: a time stamp (pTime) with respect to the current profile information, the similarity of the least similar neighbor (LSN) stored in the local recommender model and an identifier of the last received clusterhead message (mID). If a clusterhead detects a new or updated profile of a slave device, he initiates a request procedure (Algorithm 2) to get the corresponding information. In order to avoid sending separate request messages to subsequently detected devices, the request procedure uses a delayed broadcast mechanism. Thus, each time a clusterhead initiates a request to a detected device, a specified timeout period is started. If during this timeout, further relevant devices are detected, the timeout period will be restarted and the corresponding devices will be noticed in a request-list. Otherwise (once the timer elapsed successfully), the clusterhead sends a broadcast containing the request-list.

Algorithm 2: Request procedure (pseudo code)

```
1  foreach received beacon from a stable slave do
2  |  addr = slaveAddress;
3  |  time = beacon.pTime;
4  |  lsn = beacon.LSN;
5  |  if profileCache.contains(addr, time) then
6  |  |  lsnList.add(addr,lsn);
7  |  |  if updateID == beacon.mID then
8  |  |  |  consistTable.set(addr,0);
9  |  |
10 |  else
11 |  |  if requestTimer is running then
12 |  |  |  requestTimer.reset();
13 |  |  requestList.add(addr);
14 |  |  requestTimer.start();
15 |  end
16 end
17 foreach elapsed requestTimer do
18 |  reqM = new requestMessage();
19 |  reqM.addReceivers(requestList);
20 |  sendBroadCast(reqM);
21 end
```

Algorithm 3: Update procedure (pseudo code)

```
1  foreach received profile from a stable slave do
2  |  addr = slaveAddress; requestList.remove(addr);
3  |  profileCache.add(addr, profile);
4  |  consistTable.set(addr,k+1);
5  |  if updateTimer is running then
6  |  |  updateTimer.reset;
7  |  updateTimer.start();
8  end
9  foreach elapsed updateTimer do
10 |  profiles = recently received profiles;
11 |  matrix.update(profiles);
12 |  currentDelta = matrix.getDelta();
13 |  consistTable.add(currentDelta);
14 |  update = consistTable.getUpdate();
15 |  updateID = update.getID();
16 |  sendBroadCast(update);
17 |  consistTable.increaseLevel();
18 end
```

After receiving a request from the elected clusterhead, the slave devices contained in the request-list start to send their current profile information to their clusterhead. In order to avoid simultaneous responses, each slave waits a random timeout before sending the profile. After receiving the first requested profile information, the clusterhead initiates an update procedure.

As shown in Algorithm 3, the procedure starts with a further time-out phase and each received profile during this phase causes a restart of the timer.

Once the timeout period has successfully elapsed, the clusterhead starts to calculate the corresponding profile similarities and manages the resulting values in a similarity matrix. Subsequently, the clusterhead aggregates relevant similar neighbor information for each slave and broadcasts the result to the cluster participants. Because of the limited storage capacity and in order to avoid the dissemination of non-relevant information, for each slave, the clusterhead considers only neighbor profiles with a similarity value higher than the LSN from the recently received beacon. Once the recommender model at a slave device reaches its maximum capacity, this value is set to the similarity value of the least similar neighbor stored in the local model, otherwise the value is set to -1.

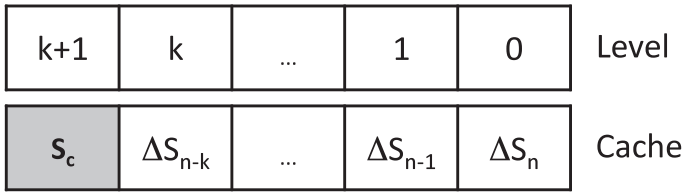


Figure 6.1: Consistency table at the clusterhead.

Due to possible communication errors, it its very likely that some devices are temporarily not able to receive updates. For this reason, each clusterhead caches a limited number of recently sent updates and maps each one-hop member to a corresponding consistency level (Figure 6.1) depending on whether he received an according acknowledgment (mID) via a beacon of this slave device or not. In the case that there are one-hop members at different consistency levels, the clusterhead calculates and broadcasts the least common update (Figure 6.2).

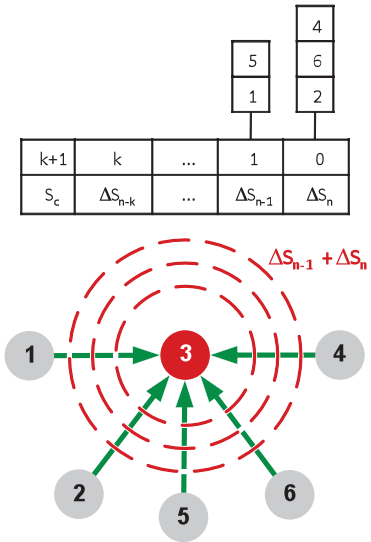


Figure 6.2: Broadcasting the least common update.

6.2 Experiments and Results

In order to evaluate our algorithm, we implemented it on top of JANE [Gorg 07] and performed several experiments via an existing rating database. The following sub-sections describe the used data set, the evaluation criteria as well as the results under certain simulation settings.

6.2.1 Dataset

We used the MovieLens[28] Data Set that consists of 100,000 ratings for 1682 movies by 943 users, where each user has at least rated 20 movies. For our experiments, the data has been split into 5 different training and test sets for a five-fold cross validation. Each training set contains 80% of the data and each test set the remaining 20%, where each set maintains part of the ratings for both users and items.

6.2.2 Evaluation Criteria

In our experiments we used the Mean Absolute Error (MAE) metric as the criterion to evaluate the accuracy of the calculated predictions. Thus, after each simulation run we compare the predicted votes with the corresponding votes in the test set and calculate the MAE, as follows:

$$MAE = \frac{\sum_{i=0}^{M} |pred(i) - r(i)|}{|M|}.$$

If a predicted item did not have an adequate entry in the test set, it was eliminated from the evaluation. In the case that there is no corresponding prediction for an item in the test set, we used the average user rating as prediction value. Note that we used the MAE only to compare how accurately our algorithms predict a randomly selected item rather than evaluating the user experience of generated recommendations. As second criterion we measured the prediction coverage. We compute the coverage as the percentage of items for which our system can provide a prediction relatively to the number of items in the test set. As further criteria, we measured the used bandwidth in Kbytes.

6.2.3 Simulation Settings

For each experiment we used the Restricted Random Way Point mobility model[29] [Blav 02] with 300 mobile devices moving along predefined streets on the map of Luxembourg city for 10 minutes. For each device, the speed randomly varied between [0.5;1.5] units/s. While, every time a device reaches a crossroad, it randomly selects a street to turn in at next.

At startup, the devices are positioned at random selected crossroads and initialized with the given votes from the training set in order to calculate an initial user profile. In order to avoid a data exchange at this point, where the devices are already strongly clustered at the crossroads, we delay the startup of our algorithm via a timeout of one minute. After this timeout the devices begin to exchange their profiles in order to determine the $\underline{k}$-most similar neighbors. For all experiments we performed a five-fold cross-validation with 25 different topologies per training set and $\underline{k}$ limited to 20. All results are shown with 95% confidence intervals.

[28] `http://www.grouplens.org`

[29] at the time those experimentations were made, we did not have the advanced mobility models proposed in Chapter 5

6.2.4 Results

The following figures show the expansion of the measured criteria in discrete steps of 30 seconds. Overall, five different experiments were performed. One without communication errors, two experiments with 15% and two with 30% (uniformly distributed) message loss. For each error rate two different algorithms – the introduced delay-tolerant intra-cluster broadcast (DICB) and a best-effort variant (BE) – were applied. The best-effort variant does not keep any state information about its stable neighborhood and sends each message only once, while DICB uses 3-level consistency table. In addition to the five mentioned different experiments, we also measured the maximum achievable accuracy and coverage in an idealized global knowledge scenario, where each node already knows its k-most similar neighbors (constant vertical line in Figure 6.3 and 6.4).

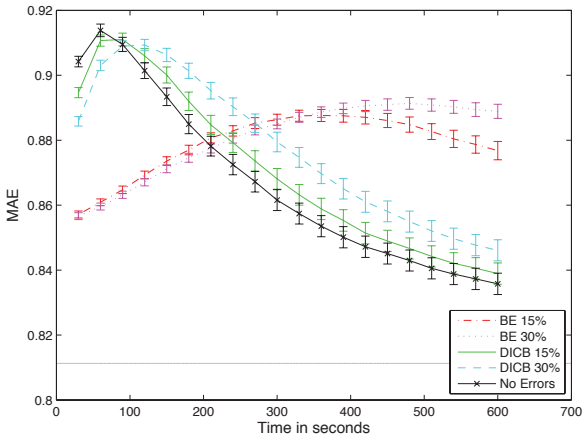


Figure 6.3: Prediction accuracy.

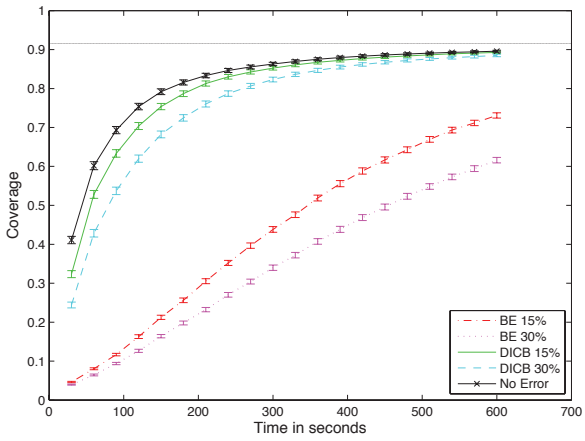


Figure 6.4: Prediction coverage in %.

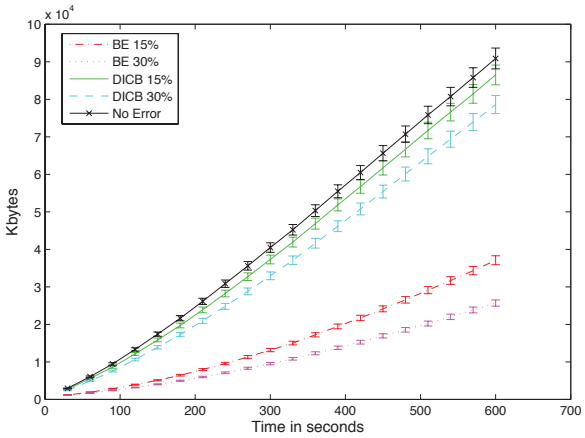


Figure 6.5: Bandwidth usage.

As the figures show, the general performance of the presented algorithm converges quite well towards the achieved values from the global knowledge scenario. Although, the measured coverage (Figure 6.4) converges faster towards the optimal value than the measured accuracy (Figure 6.3). This is due to the fact, that the coverage depends more on the number of retrieved similar neighbors. The specified limitation to 20 similar neighbors causes the corresponding caches to already reach their maximum capacity after the first third of simulation time. However, the

accuracy depends more on the quality of the retrieved similar neighborhood and therefore shows a different convergence acceleration. Furthermore, the results show that the CF performance is significantly reduced in a scenario with 15% or 30% message loss, when the best-effort variant (BE) is applied. While the difference in accuracy constitutes only around 5% and 6% respectively, the achieved coverage after 10 minutes is still around 20% and 30% lower than in the loss-free setting. However, if DICB is applied the achieved performance is nearly as good as in the loss-free case, while the used bandwidth constantly remains lower than in the error-free scenario as shown in Figure 6.5. This lower usage in bandwidth results from the fact, that with a lost message further induced messages are also omitted.

6.3 Conclusion

This chapter presents a delay-tolerant algorithm for a CF based recommender system in highly dynamic mobile environments. The actual CF algorithm runs on top of a weighted cluster topology generated by NLWCA. In order to exchange profile information with its stable cluster participants, the delay-tolerant broadcast algorithm operates on each clusterhead. Evaluation results show that the proposed algorithm achieves a reasonable prediction accuracy and coverage even in scenarios with a relatively high message loss.

In the conducted experiments we used the MovieLens data set that contained movie ratings, however any kind of information can be used for collaborative filtering. Therefore in Section 7.3 we propose to use collaborative filtering to rate services.

Chapter 7

Service Discovery in ad hoc networks using Zeroconf and SLSR

Contents

7.1 Zeroconf on top SLSF 116
7.1.1 Adaptability to ad hoc networks 116
7.2 Context-awareness in Service Discovery 117
7.2.1 Network 118
7.2.2 Node 120
7.2.3 Space 121
7.2.4 Service 122
7.3 Collaborative filtering for service discovery 122
7.4 Service Discovery Architecture 123
7.4.1 Context: ANR SARAH project 123
7.4.2 Scenario 123
7.4.3 Service discovery architecture description 124
7.4.4 Modeling our scenario using multi-agent and co-simulation 125
7.5 Experimentations on real devices 128
7.5.1 OLSR experiments 128
7.5.2 Zeroconf on top of SLSF 129
7.5.3 Zeroconf on top of SLSF and SLSR 129
7.6 Conclusion 131

This chapter presents our contributions regarding service discovery. Moreover, it shows how each of the previous chapters contribute to service discovery in ad hoc networks. We chose the standardized and well-known Zeroconf protocol as service discovery protocol. The reasons and advantages of using Zeroconf in this ad hoc context and also how SLSF/SLSR contribute to Zeroconf, to make suitable to ad hoc networks, are presented. Thereafter, a first draft for context aware service discovery is proposed. A further extension of this awareness using collaborative filtering techniques to select the best services is described. Finally we present a service discovery architecture, proposed for the French ANR[30] project SARAH[31], that is capable of including those various features.

[30] Agence Nationale de Recherche
[31] Services Avancés pour Réseaux Ad hoc

7.1 Zeroconf on top SLSF

We propose to replace the multicast IP structure needed by Zeroconf with our SLSF dissemination structure for the following reasons:

- Zeroconf uses multicast to disseminate information efficiently to other zeroconf nodes in the network: SLSF provides efficient dissemination and reduces the number of forwarding nodes.
- Group management is not necessary in our context, since only Zeroconf uses multicast in our network: SLSF can replace multicast and could in a future work handle groups at the cluster level using group membership messages. Until there is an efficient and widely accepted multicast-protocol providing group management and thereby attracting a wider use of the multicast structure in ad hoc networks, group membership management can be skipped.
- Zeroconf's information, in our scenario, may interest every node. Even if this was not the case, it probably passed through or has been heard by most of them at some point. Therefore, there is no interest in using multicast (i.e. selectively broadcasting) in adhoc networks in the cases we envision.
- SLSF provides additional advantages such as stability, context-awareness and scalability.

As shown on Figure 7.1 Zeroconf is not affected at all by this change. Multicast is only a structure for easy dissemination of messages, SLSF transparently replaces it.

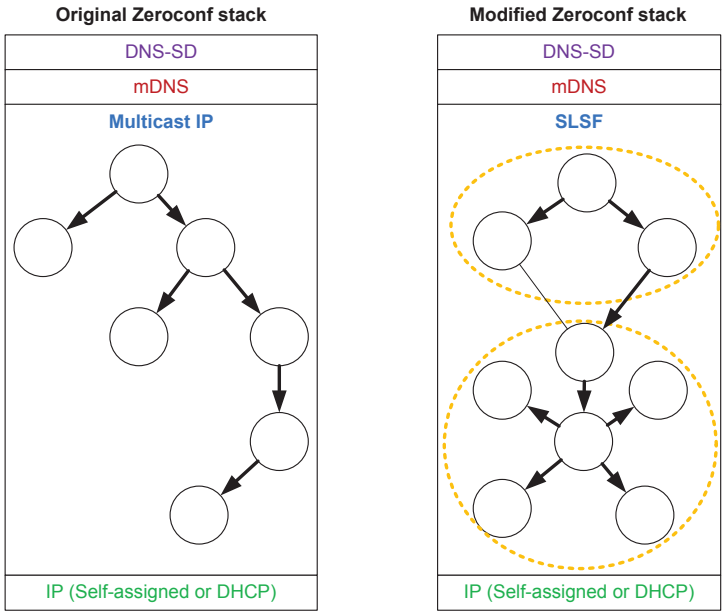


Figure 7.1: Zeroconf stack with Multicast IP and SLSF.

7.1.1 Adaptability to ad hoc networks

Zeroconf was designed for wired networks and therefore timer and timeout values recommended in the mDNS RFC[Ches 11] are adapted to networks where changes occur rather rarely. The

RFC recommends a TTL value of 120 seconds (Chapter 10 - page 32 [Ches 11]) for DNS resource records with a host name or a host name contained in the resource record's data.

At first, one could think that 2 minutes in a MANET is far from reactive enough due to its high dynamicity. But actually, regarding services and their discovery, it is. Indeed, routing protocols and service discovery protocols execute very similar operations such as disseminating messages to all nodes in the network; queries or responses for service discovery and routing information for routing protocols (i.e. in link-state protocols). However, service discovery is at the application level, therefore service information has a very different purpose and life cycle. Routing protocols are about tracking the topology while service discovery is only about services, not about the underlying topology. Moreover, in MANETs, topology changes are much more frequent than service related changes. SLSR provides us with the topology information (i.e. how to reach each node) and additionally a dissemination structure (provided by SLSF). Thus, Zeroconf on top of SLSR, only needs to worry about services and not about how to reach them. Therefore, periodicity of service advertisements can be very low compared to the overall dynamicity of MANETs.

Even if nodes hosting services move around and are not reachable at a given point in time because they are out of the network, their service related information is likely to remain valid once they come back in the network. Therefore, we can use the standard Zeroconf protocol values of 120 seconds of resource records TTL value. The services of a node that only temporarily leaves the network remain valid until the TTL times out. This has two advantages. A node leaving a network in an unplanned fashion (i.e. not a clean shutdown) does not need to react to it. When the node comes back, it does not need to send a particular message except the planned periodic advertisements. Thus, services might remain in other nodes' local services caches, for a maximum time of 120 seconds, even if the node is already unreachable. However, these situations only happen for inactive services and unrequested services. If a node requests a service during its temporary disconnection and does not receive a response, it deletes it from the cache, and so will all the nodes listening to the passing messages. Moreover, the information, whether the node hosting the service is reachable is already available and provided by SLSR in its routing table. So, if a node checks its local service cache for a requested service, it also checks its SLSR routing table to verify that the node is still reachable. However, it will not necessarily delete this node from its cache, but just not use it or ignore it as a potential service until it comes back in the network or the entry times out. Doing so, a service is not "rejected" when it disconnects only for a short time, but it is only not considered during its disconnection time[32].

A node that proposes a very attractive and reliable service but is very mobile and often disconnects shortly should be avoided as a first choice service. The following section describes how to do this using context-awareness applied to Zeroconf.

7.2 Context-awareness in Service Discovery

To be able of more than just finding a matching service, this section presents how and where to make the discovery and the usage of services context aware. In our architecture with Zeroconf, context-awareness can occur at two different levels:

- at resource records level: TXT records associated to SRV records can include additional context data. The example on Figure 7.2 depicts a TXT record of a lightweight-device Internet gateway service that adapts Internet content to the user's device. This records

[32] a temporary unavailable service due to a topological disconnection could be notified to the zeroconf user by graying out the corresponding service entry. The user could, based on this knowledge, decide to try again later to see if the service is reachable or use another service.

contains two data fields. The served time that indicates the total time this service has provided its services and the flag that indicates that the service is free (i.e. no fees to pay). A querier can, based on these information, pick between two services and decide which one is the best.

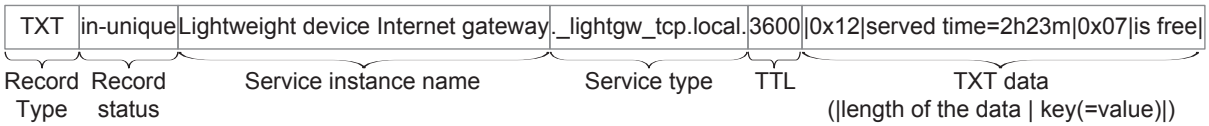


Figure 7.2: TXT record example containing contextual information.

- at dissemination level: intermediate or directory nodes filter out superfluous information from to-be-forwarded queries or responses depending on the context. Additionally to the proposed extension in Zeroconf as shown in Section 2.3.3, we propose to filter queries or responses in a hierarchical manner. Clusterheads can lighten and reduce the choice for their slaves. For example, among all the received responses to a query, a clusterhead can keep, only the best or the top five services. If the number of nodes hosting a similar service in the network is big, slaves instead of receiving the complete list, only receive the best service(s) among all the services available.

Using context-awareness, a query can be filtered early by intermediate nodes in order to reach only the relevant nodes (e.g. directory nodes) in the network. Then, upon reception, the recipients match the query with the DNS records and additionally refine the results using the additional context information contained in the TXT record. Once the responses are sent out, intermediate nodes can aggregate some of the replies to reduce the traffic load (as planned in the Zeroconf extensions). The original requester obtains a list of services matching his query and his/the context.

To be able to acquire the context for those levels, we are using **metrics**. Metrics are different kind of quantifiable information gathered through calculation, sensors or user interaction. They that can characterize context information by themselves (e.g. geographical position) or be coupled with one or more other metrics (e.g. temperature sensor and humidity sensor to provide weather information) to assess the situation.

We classify metrics in 4 categories: Network, Node, Space, and Service. Following are the metrics we see as the most relevant [Lecl 07]:

7.2.1 Network

Metrics related to the network, that can be computed or gathered directly from the network, are:

Hop count: Number of intermediate nodes that need to be crossed to reach a destination. Also called hop distance. It provides a distance metric used for routing decisions. Based on hop count, the shortest path between two nodes can be calculated to decide which route to take.

Throughput: The amount of data that can be carried from one point to another in a given time laps. Throughput estimation or calculation are used to regulate the package sending rates to avoid network congestions.

Bandwidth: Maximum capacity of throughput on a given connection. It is the maximum theoretical value of possible throughput. This metric is generally used in network management to distribute the network load and predict link capacities.

Latency: Latency, also referred as lag, is the time elapsed between the packet sending and the reception by the end-receiver node. This time, is composed of the sum of different latencies: the processing time of a message at the end-points communication equipments, the time to travel through the physical medium on the given distance (e.g. for optical fiber: about 1.5 times slower than light = 4.9 microseconds per kilometer) and the processing times in the various communication equipments (e.g. routers, signal amplifiers/repeaters). The latency is a specially important issue in real-time applications. An access on both end-points is required to be able to calculate the latency between those two end-points.

Jitter: Deviation in time or displacement of data packets during transmission between two nodes. Mostly caused on relay nodes or repeaters in the network. Unordered packets or irregular packet rates can become an issue in real-time applications.

Round Trip Time: RTT, is the time/latency for a data packet to arrive at its destination and come back to the source. The distinction between the Round Trip Time and the Latency is made because the RTT can easily be obtained by sending a packet awaiting a response (typically an ICMP[33] echo request packet) and recording the departure and arrival times.

Expected Transmission Count: ETX [De C 04] is an expected number of (re)transmissions needed for a packet to be received correctly at its destination. The best value for ETX is 1 (1 single transmission reaches its destination) and the worst is infinity. ETX is used in multi-hop wireless transmissions as a route metric to determine the less erroneous path. ETX is an expected value, thus it is a probability forecast. This metric is used, for example, in OLSR.

Situation: As more detailed in Section 2.6.1 centrality measurements provide the situation of nodes in the network. Example of centralities are Degree (number of neighbor nodes), Closeness (hop distance to all nodes in the network) and Betweenness (number of time the nodes is on the shortest path of two other communicating nodes).

Link stability: The link stability metric quantifies the stability of a link in an ad hoc network. Link stability can be computed from individual link metrics, but also from a combination of different metrics: signal strength information, GPS positioning, speed information, link uptime, beacon count (e.g. as presence reference, affinity, associativity), etc. [Pari 09]. In *Associativity-Based-Routing* (ABR) [Toh 97], authors use speed information, transmission range and beacon count(ticks) to determine a stability threshold. Considering a node's speed, they determine the number of ticks required to differentiate a passing node from a more stable one. In [Andr 08b] clusters are formed only among nodes for which links are considered stable. Similarly as in ABR, stability is based on the number of beacons received during a specific time period. Above a given threshold the link is considered as stable. In [Gerh 02], the link stability is computed based on link durations analysis by statistically evaluating and predicting link durations. They show that selecting the oldest link is not always the best way to go since the chances for this link to fail become higher from a certain age on.

[33] Internet Control Message Protocol

Lost Routed Packet: LRP [Bado 05] is a combination of two basic packet metrics: Received Routed Packets (RRP), number of received to-be-routed packets and Send Routed Packets (SRP), number of sent to-be-routed packets. LRP = RRP − SRP, is the number of to-be-routed packets that where not forwarded. The Lost Routed Packet metric represents the willingness to contribute and participation of nodes in the network.

7.2.2 Node

This section describes metrics related to the node itself, mostly the characteristics of the device. Metrics related to the node are easy to obtain, they represent a basic set of values to improve decisions based on the capacity of the node.

CPU performance: Processing power of the device. A high CPU performance indicates the ability of the node to process relatively large amounts of incoming messages or even to host certain, more resource consuming, services.

Memory: Similar to CPU performance, more memory indicates the ability for a node to host larger quantities of information (e.g. routing table, service directory).

Signal Strength: Signal strength of the radio signal. A large signal strength indicates that the node is able to reach more nodes within one local broadcast. However, nodes with lower signal strength might not be able to reach the source node to respond to this broadcast.

Charging state: We propose 3 states: Charging, on battery, and no battery available. This metric gives an indication whether the node may move in the next time period.

1. Charging: The node is probably not moving until the charging process finishes. A charging node also has, momentarily, no energy concern. Charging nodes should be considered in priority to execute a task.
2. On battery: The node is running on batteries, the most common state. No information about stability can be inferred from this state.
3. No battery available: This is the best case. The device has no battery available which means it is a fixed node (e.g. a computer tower participating in the network using a wireless adapter). The Node can be considered as very stable and reliable (e.g. no energy shortage).

Device proper acceleration: Numerous devices (e.g. smartphones) are nowadays equipped with accelerometers. They provide the ability to detect magnitude and direction of the acceleration as a vector quantity and can be used to sense vertical orientation, acceleration, vibration, shock, and falling. While this information is mostly used on the application basis (e.g. rotate the display correctly), it also provides information about stability. We propose to differentiate 3 states of movement:

1. Sensor is detecting zero movement: This is the simplest case, the device is not moving, thus it is currently geographically stable.
2. Sensor is detecting strong/intense/large movement: Completely at the opposite of the first case, the device is moved largely and constantly. We consider those movements to happen mostly when the device is currently not operated, as for example when the device is inside a pocket.

3. Sensor is detecting moderate movement, small movements are detected. This is the typical movement pattern when the user is operating the device, holding it in his hands. It is not necessarily possible to differentiate between a user moving (i.e. walking) or just standing. However information that a user is operating the device already is an useful information since we could (in most of the cases) deduce that the user will be moving slower to operate it correctly and comfortably or even simply read the screen.

Using those 3 differentiations, one can tell about a node that it is "stable", "unstable and probably in movement/walking" and "unstable but moving slower".

Application: Applications can provide numerous information about the context of a node. For example, additionally to the sensor movements, applications could provide to the lower layers whether or not they are currently operated by the user or not. The information used at the lower layer highly depends on the applications and also on whether it is possible to use this information in a useful way.

7.2.3 Space

Metrics related to the space/environment surrounding the node. This category of metrics are the most difficult to obtain since they link the computer world to the physical world.

Geo-localization: Is the position of the node in space. Depending on the localization technique, one can know the absolute position (e.g. using the GPS[34]) or the relative (to a reference point) position (e.g. using trilateration or triangulation techniques).

Relative Mobility: The mobility of a node relative to its surrounding neighbor-nodes. Relative mobility can be gathered using a geo-localization system (e.g. GPS), requiring nodes to exchange their position information and then compute their relative mobility. However techniques using only network or locally available information exist. In ABR [Toh 97] where relative mobility is expressed as associativity based on the beacon count or as in [Basu 01] where authors compute relative mobility by measuring the signal strength of successive message receptions from neighboring nodes. Doing so, a node is able to distinguish a neighbor moving away, moving closer and staying at the same distance from it.

Speed: Moving speed of the node, it can be measure, for example, using a GPS or a podometer. Speed information is useful, especially to set thresholds for stability computations.

Moving Frequency/Tendency: It provides a movement profile of the user and helps on movement prediction. Long term profiles provide the movement habits of the user. Doing so, prediction based on habits can propose a movement probability for the next time period, for example the next hour. Short term profiles reflect the current mobility situation and only provides short term mobility tendency. Moving tendency can be refined using the previous *Charging state* metric.

Cardinal Direction: It is available using a compass (often available on current smartphones). The cardinal directions provide roughly the next location of the node. It can be very useful, for example in a cluster structure, to prepare migration of one node to its new cluster.

[34] Global Positioning System

7.2.4 Service

Metrics related to the services available in the network. Service metrics are used to enhance the process of service discovery by proposing more relevant services, better localized services and more stable services and available services.

Service proposition: The number of services the nodes provides to the network. A node proposing numerous services can be considered as more powerful, but can also be considered as more likely to be overloaded since it attracts more service requests.

Service knowledge: The number of foreign services a node knows. A node knowing about a lot of service is not necessarily hosting any service, but it indicates that it is probably well positioned and stable in the network. This metric can be used to decide which node(s) should be considered for becoming a *Service directory*.

Service discovery context: The context in which a given service was discovered is composed of:

- Service discovery type: Discovery of a service, can be either passive or active.
- Discovery time: Time since the service was discovered.
- Service distance: Hop distance needed to communicate with the service.
- Service usage: Gives usage information such as the duration of the last service usage, the last time the service was used or the usage frequency of that service.

7.3 Collaborative filtering for service discovery

Collaborative filtering presented in Chapter 6 uses the ratings acquired from similar users (users with similar taste/profiles) to propose and predict ratings on unrated items. Similar users collaborate together and exchange their ratings and after a computation the unrated items obtain a predicted rating. In the conducted experiments we used the MovieLens data set that contained movie ratings.

However, any kind of information can be used for collaborative filtering. We propose to use collaborative filtering to improve service discovery.

In this thesis we consider relatively large ad hoc networks, therefore we also consider to deal with large amounts of available services. The hand-held devices today are becoming more and more powerful, offering a wide range of possible applications. Each of those applications, on the network, could possibly be represented by a service on the network. In a network of 200 nodes, if each of the devices proposes several services, this represents a considerable large amount of available services. A device, hence also its user, could therefore face easily a list of 100 available services corresponding to its request. Collaborative filtering can help to avoid the user the tedious task to chose in a list of 100 services.

In this thesis, in Chapter 6, we presented a delay tolerant collaborative filtering algorithm based on NLWCA. The advantages of this algorithm are that:

- it exchanges ratings only among members of the same cluster. Information is therefore only exchanged locally inside the cluster.
- it is delay tolerant. A loose synchronization among participant is used.

- it takes advantage of the natural mobility of the nodes to distribute information among different clusters.
- it is based on NLWCA. This makes the algorithm fully compatible with our architecture and protocols (SLSF/SLSR).
- it performs well. As shown in the results, the performances of the algorithm are good even under lossy conditions.

As a result, using this collaborative filtering algorithm permits to present the user a list only with the best services available. Those services are elected as best by the votes of like-minded users. Furthermore, the filtering can also reduce the network load. For example, services with an overall very bad rating could be left out in the passing response or their announces could be limited in distance to avoid unnecessary transmissions.

Collaborative filtering does not replace other metrics that determine the objective quality of a service, it however adds a social dimension to discovery process, hence the context.

7.4 Service Discovery Architecture

This section proposes a service discovery architecture that supports context-aware service discovery in a hybrid scenario. Spontaneously connected mobile ad-hoc nodes can take advantage of a fixed infrastructure and but still be able to work without it when it is not available. In order to adapt to the availability and dynamicity of the network and services, we use metrics that provide contextual information that allows selecting the best service in the network, among the matching services.

7.4.1 Context: ANR SARAH project

One of the funding partner for my thesis was the French ANR[35] project SARAH[36]. SARAH is a joint project, started in February 2007, between Université Pierre et Marie Curie (LIP6), Université de Paris Sud - XI (LRI), INRIA Lorraine - Madynes, INRIA - Hipercom, GET INT (LOR: équipe de sécurité), France Telecom R&D (Orange Labs) and Ucopia.

The SARAH project aims at deploying advanced multimedia services in an hybrid ad hoc network architecture attached to an infrastructure network. The hybrid architecture permits to efficiently extend applications and services offered by the infrastructure to the mobile nodes via the ad hoc network at a low cost. We contributed on the work packages SP3 and SP5 of the SARAH project, which emphasizes on Service Discovery Protocols (SDPs) and architecture for those dynamic networks [Ciar 07, Ciar 09, Reyn 10, Lecl 10a]. In SP3, an architecture for Service Discovery has been designed that is able to function with services that are being described or developed in other work-packages: SP2 (routing / multicast, geo-localization) and SP4 (Security), but to not rely on such services. The main goal is to be functionally integrated with the SP5, i.e. the demonstrator. Therefore our solution has been tailored to fit the SP5 scenarios (firstly the "Musée des Telecoms"), but remains generic enough to provide a solution for other settings.

7.4.2 Scenario

The SARAH project describes, within the SP5 work package, a demonstrator scenario based on a visit in a museum. Each visitor participating in the experiment receives a PDA type of device.

[35] Agence Nationale de Recherche
[36] Services Avancés pour Réseaux Ad hoc

Visitors can follow a guided visit or freely visit the museum. Using the PDA, visitors obtain their geographical position inside the museum and additional media or text information about nearby items and points of interest. Moreover, a geo-localized quiz is proposed in which visitors are displayed questions, based on their position, for which the answer can be found in their surroundings by reading or watching nearby items. Service discovery, in this scenario, permits to automatically detect and find the additional information, on the network, related to those items and points of interest.

7.4.3 Service discovery architecture description

In order to permit service discovery in the SARAH scenario, we propose an architecture that fits it, but also that is generic enough to be applied to other scenarios and situations [Ciar 09, Lecl 10a]. The goal of this service discovery architecture is to be able to make different types of networks interact in a loosely-coupled kind of way. The core of the architecture is formed by, spontaneously connected, ad hoc devices with a static (fixed) infrastructure composed of access points and servers around them(Figure 7.3).

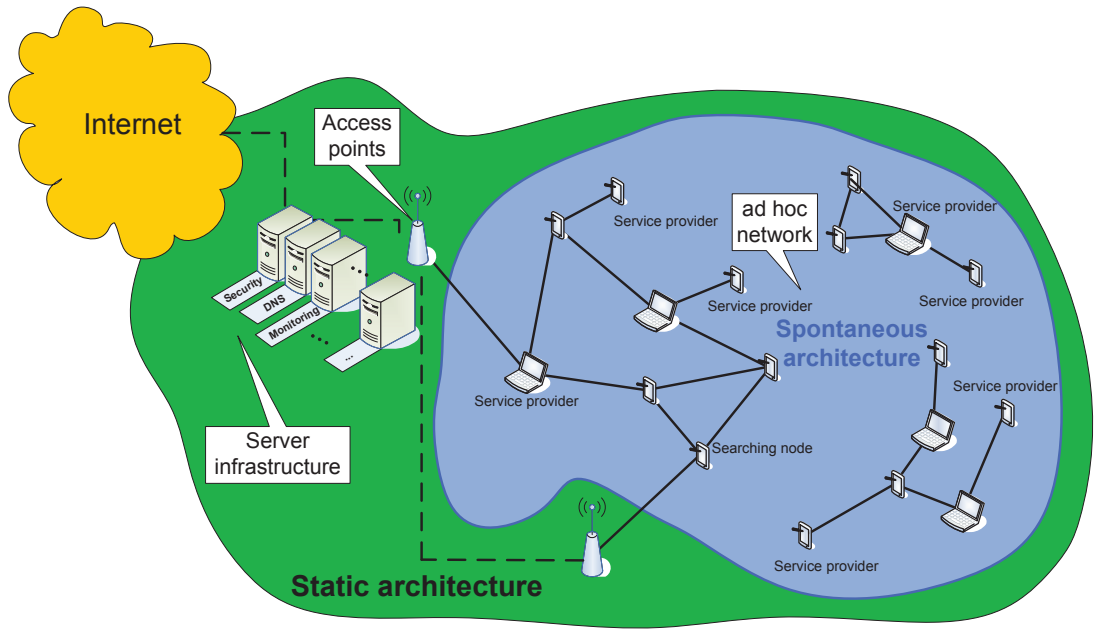


Figure 7.3: Service discovery architecture.

One of the requirements was to be able to transparently use the infrastructure while also being able to continue to function properly in a pure ad hoc mode. In the core of the network, for the ad hoc part, we use the SLSR/SLSF protocol as routing and dissemination protocol with Zeroconf on top of it. The fixed infrastructure uses classic routing schemes. Having Zeroconf in the ad hoc network and a DNS server in the fixed infrastructure, service discovery can seemlessly pass from infrastructure-less to server-based discovery. This is a feature provided natively by Zeroconf.

The transition from infrastructure to ad hoc network at the Zeroconf level is transparent. At the routing level, however, packets formats and protocols are not the same between the infrastructure and the ad hoc side. Therefore, gateway nodes need to provide the necessary translation functionalities. Ad hoc nodes, that are not connected directly to an access point,

need to be able to detect or discover the existence of such gateways. Therefore, the gateways advertise their additional functionality as a service. Using Zeroconf, nodes discover the gateway nodes and reach them using the route obtained by SLSR. Gateway nodes have three tasks:

- Advertise themselves using Zeroconf as new service type to designate their gateway capability.
- Unpack data payloads going towards the fixed infrastructure from their SLSR/SLSF headers.
- Encapsulate data payloads going inside the ad hoc network in SLSF/SLSR messages.

Those gateway nodes can be either designated nodes that are part of the ad hoc network or the access points connected to the ad hoc network. The designated nodes would be ad hoc nodes that are in direct communication range with an access point. However, those node could and most probably will often move, so new nodes would need to be designated each time a node moves out of range of the access point. Therefore, gateway nodes should be the access points connected to the ad hoc network, because of their reliability and also their static position.

To become a gateway, the access points need to have SLSR with Zeroconf up running, thereby acting, from the ad hoc point of view, as a normal ad hoc node. The access points advertise their gateway capability as a Zeroconf service. Ad hoc nodes discover those gateway services and can select the "best" gateway among several, if available. The route to the gateway service is provided by SLSR. Any payload towards the infrastructure is then simply SLSR-unicasted towards a node that provides such a gateway service. Any payload from outside the ad hoc network with a destination inside the ad hoc network can be routed using the SLSR routing table in the access points towards the correct access points and then inside the ad hoc network.

Zeroconf announces that are SLSF-broadcasted messages are simply unpacked at the gateways and forwarded as a normal Zeroconf/DNS message to the DNS server. In the same way, Zeroconf queries sent by nodes from inside the ad hoc network are also unpacked and forwarded to the DNS server, which then can reply to them. This transition from ad hoc to infrastructure is implicit, the ad hoc node, source of the query, is not necessarily aware of the fact that its query might be replied by an actual DNS server. Upon reception of the replies from the different nodes, it can decide to give more credit to the DNS server's response due to its longevity in the network. An ad hoc node can also do an explicit "outside" query by SLSR-unicasting the query to the DNS server.

Using, a standardized and well-known protocol such as Zeroconf, provides the necessary flexibility to easily adapt the architecture to new scenarios. For example, as shown in Figure 7.4, we can extend the scenario with a third type of network, a mesh network. Here, the mesh network could use exactly the same protocol stack as the ad hoc network, namely Zeroconf on top of SLSR, by adapting the settings to the more stable nature of mesh networks (i.e. manually set cluster weights to optimize the clusterhead positions). However, it could also use any other routing protocol as long as it supports the Zeroconf layer (i.e. DNS).

7.4.4 Modeling our scenario using multi-agent and co-simulation

We use the AA4MM architecture presented in Chapter 5 to model a simplified version of the SARAH scenario. In our first model, we have 4 different groups of nodes that are randomly positioned in a museum-like environment. Each of the 4 groups has a different meeting point. Very similar as in Section 5.8, the behavior of the nodes (agents) is the result of the combination of 3 simple forces:

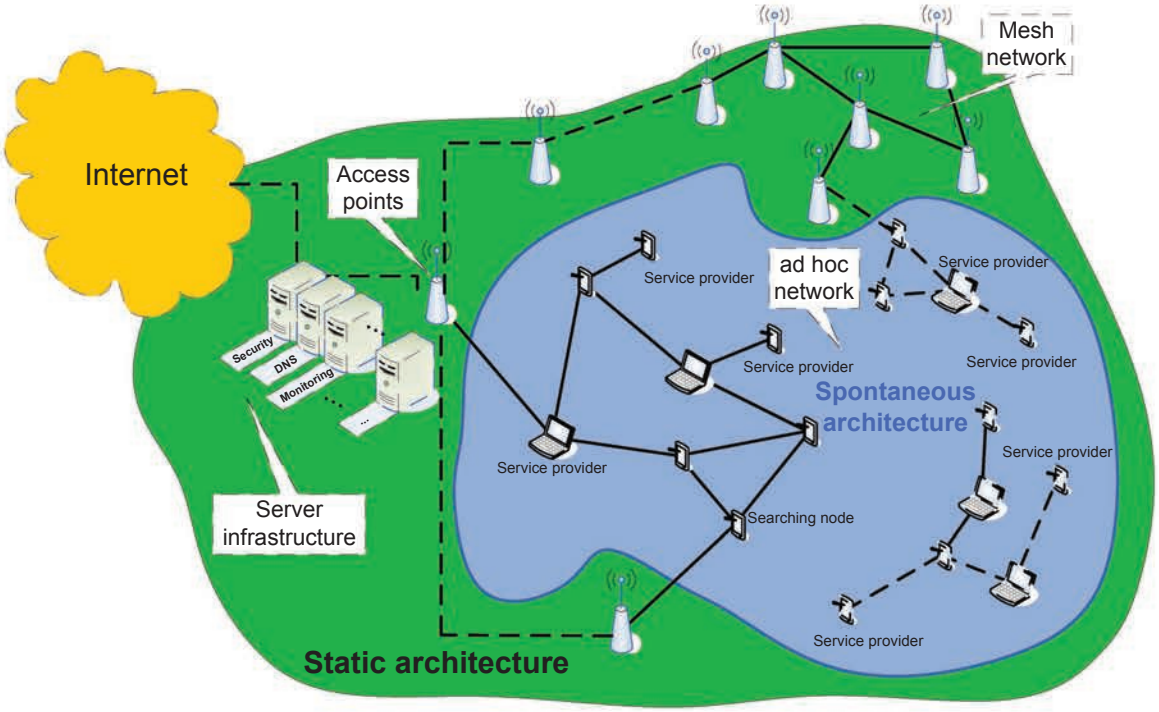


Figure 7.4: Service discovery architecture.

- All nodes have a different goal depending on their group membership.
- All nodes avoid walls by applying a repulsive force from the walls.
- Nodes have a repulsive force from each other (comfort zone).

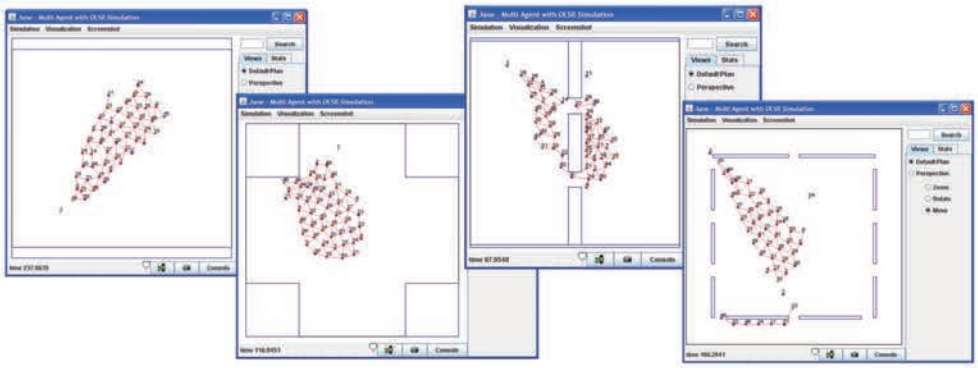


Figure 7.5: Museum visit example in multiple environments.

The resulting behavior is that each node will, wherever it starts from, head towards its goal area (here at the 4 corners), thereby gather together with other group members, as shown on Figure 7.6. Another mobility scenario, on Figure 7.5, was already presented in Section 5.8: A group of students follow their guide at a visit of a museum.

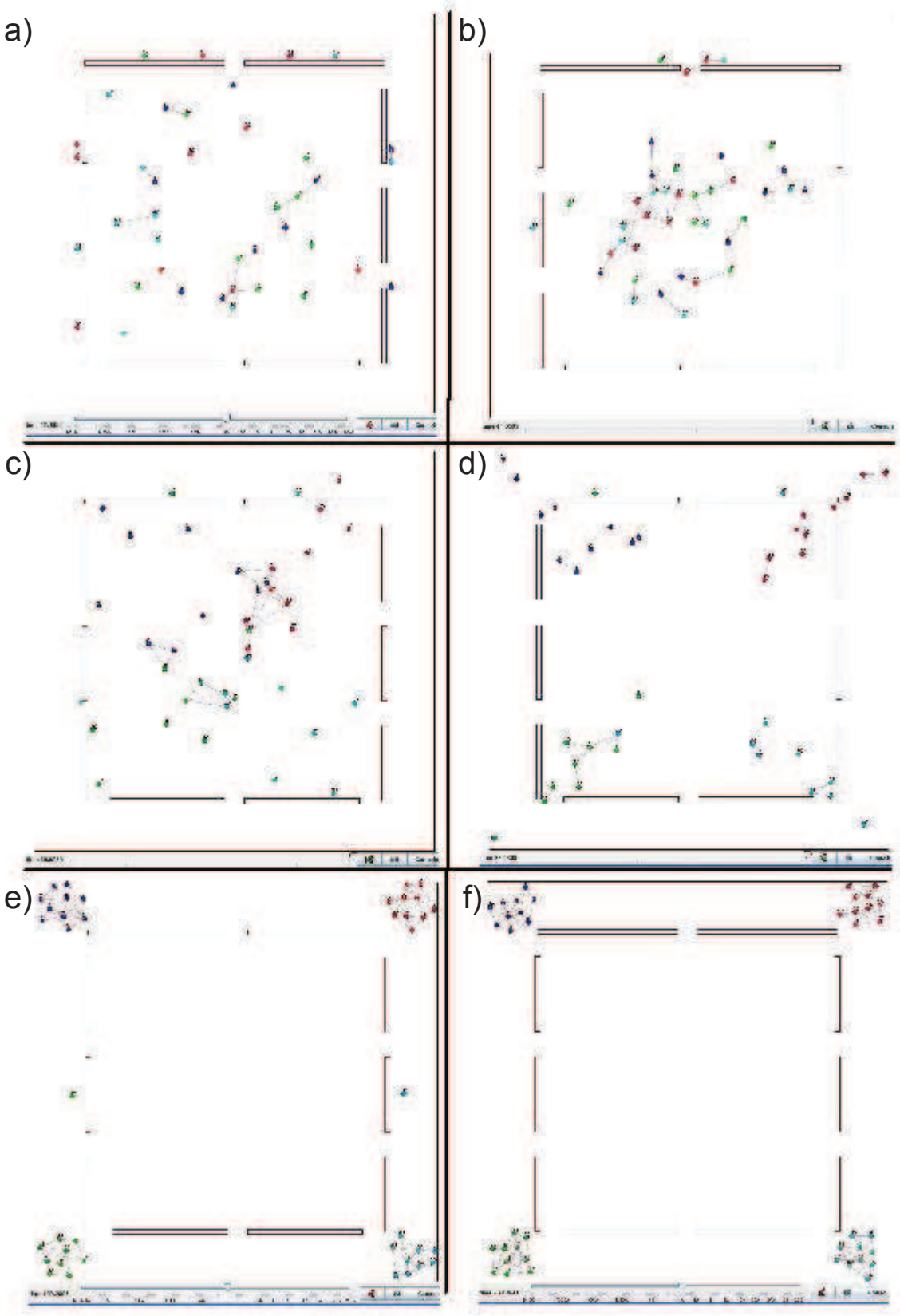


Figure 7.6: Group gathering scenario. Evolution (from a to f) from randomly placed nodes heading towards their goal area.

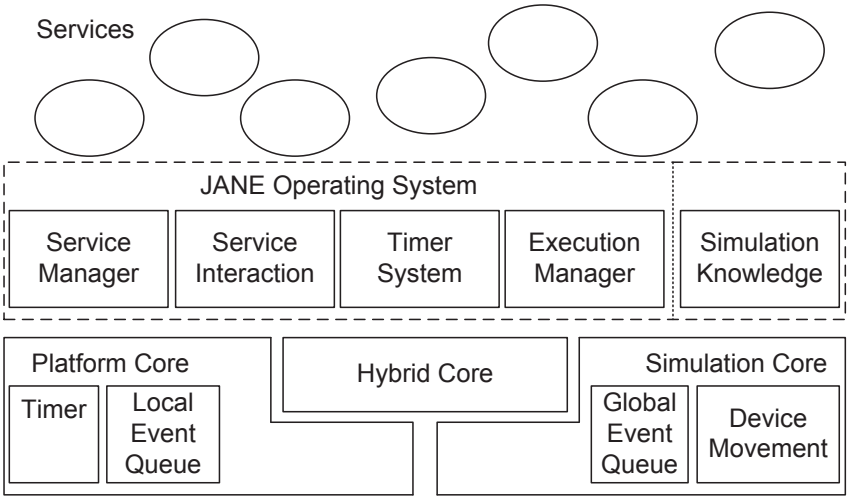


Figure 7.8: The JANE operating system.

7.5 Experimentations on real devices

During my thesis, several real world experiments were done. Those experiments were composed only of a limited number of devices and are to be considered as a proof of concept rather than performance tests. As test devices, we used Nokia N800 handheld devices. They have the great advantage to be linux based and provide the necessary wireless and debugging tools for our experiments.



Figure 7.7: Nokia N800 used for the experimentations.

All the protocols presented in the contributions of this thesis have been implemented in Java for the JANE simulator (described in Section 5.1). For the experimentations on real devices we used the, so called, platform mode provided by JANE. This mode permits a rapid cycle between the design and the prototype of protocols and permits, similarly as the prototype environment in [Abde 07], a synthetic implementation, here the JANE implementation, in real network conditions. The same implementations as in the simulations can be used without any modification on real devices. In platform mode, a JANE node/device uses standard Java network sockets for communication instead of the simulation core simulated interfaces in simulation mode. Figure 7.8 depicts the 3 cores of JANE, Simulation, Platform and Hybrid, and how the services (i.e. the implemented protocols), transparently use either one of them through the JANE Operating system.

To use the JANE platform mode on a handheld device (such as a Nokia N800 as we used) or any other devices, the only requirement is the presence of a Java virtual machine [37].

7.5.1 OLSR experiments

We executed experiments with up to 5 devices. The first experiment was our OLSR implementation on several hops at the INRIA building. To augment the number of participating nodes, we used a laptop additionally to the 3 Nokia N800. The experimentation worked up to 4 hops,

[37] Java is not supported officially on the N800, but the Jalimo project `https://evolvis.org/plugins/mediawiki/wiki/jalimo/index.php/Jalimo` offers a free Java stack for mobile Linux devices.

however the interferences induced by the local wireless access points made the connections rather unstable so that connections and routes dropped off from time to time and came back after few seconds.

To avoid any interferences with existing access points during one of the first SARAH demos at the Orange Labs building, we used ethernet and ethernet-over-usb as replacement for the actual ad hoc connections. Using cables, we showed that the protocol (here OLSR) worked well and behaved properly on disconnections and reconnections of cables. A quick scan of the wireless environment showed more than 15 access points in the area. Any ad hoc connection would have been severely interfered by the access points and due to the lower emission power of ad hoc nodes.

7.5.2 Zeroconf on top of SLSF

Another experiment, done with 3 Nokia N800 devices, was performed with Zeroconf on top of SLSF. Figure 7.9 shows the experiment where the node Bob provides a *http* service "service-Bob" using Zeroconf. The figure shows the DNS table containing DNS names and http service "serviceBob" discovered using Zeroconf.

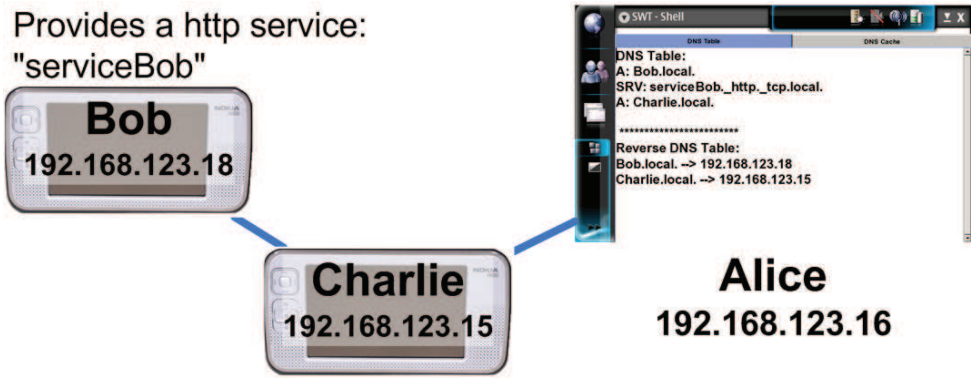


Figure 7.9: Zeroconf on top of SLSR experiment.

Figure 7.10 illustrates the graphical user interface and is a screenshot from a Nokia N800 communicating with 2 other N800 devices using Zeroconf on top of SLSF. The interface shows, the local DNS table of the node containing the two other devices in the network (*Nokia_16.local* and *N18.local*). The routing table shown here is not the one from SLSR, but is a preliminary routing table extracted only from the incoming Zeroconf messages. Here, node IP address 192.168.2.16 is the next hop for all destination since it is the CH address.

The second tab in Figure 7.11 shows the DNS cache containing the DNS records. The third tab (Figure 7.12) shows the underlying NLWCA clustering information. Finally, the last tab in Figure 7.13, permits the user to interact with Zeroconf and SLSF. Here the user can register a new service to be announced in the network or send chat messages. Chat messages are a simple demo of messages that are send and forwarded using SLSF and each node in the network receives them.

7.5.3 Zeroconf on top of SLSF and SLSR

Another experiment example was with SLSR, thus with SLSF underneath, and Zeroconf. The experiment is shown on Figure 7.14. Here, we show a simulation scenario, instead of a real experimental setup, so we are able to provide an example with 20 nodes. The Clusterheads are

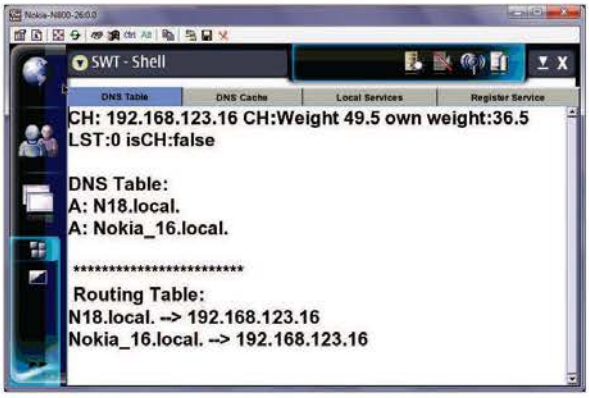


Figure 7.10: DNS Table view.

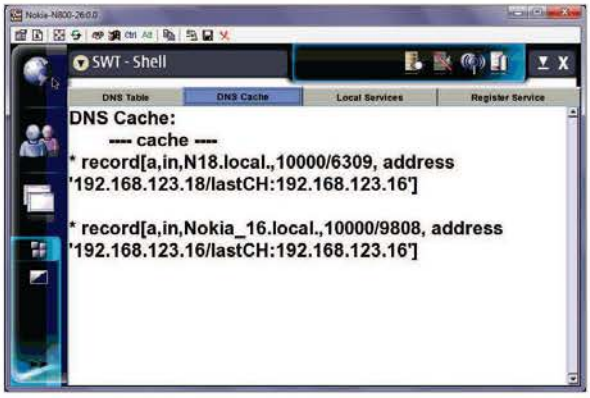


Figure 7.11: DNS cache view.



Figure 7.12: Zeroconf available services.



Figure 7.13: Chat/interaction window.

marked red and the ICR selected nodes are marked orange. In this network, nodes 3, 9 and 18 provide a *http* service and nodes 6, 7 and 15 provide a *video* service. This simulation also includes a simplified service metric that provides a rating or quality of the service that influences the service discovery process. Using this metric, slaves obtain only the best service per type from their clusterhead, instead of all the services. Thus, clusterheads have the complete set of available services and slave nodes only a list of the best service, defined and given by the clusterhead. This metric service quality is here inserted in the TXT record of the Zeroconf service description. For example, following is the http service (DNS name "web-15") DNS records of node 15 (DNS name "myDevice15.local"):

```
record[srv,in-unique,web-15._http._tcp.local.,120000/103165,myDevice15.local.:888]
record[txt,in-unique,web-15._http._tcp.local.,120000/103165, metric=10]
record[ptr,in,_http._tcp.local.,120000/100165,web-15._http._tcp.local.]
```

Figure 7.15, shows the DNS and SLSR routing table for clusterhead node 2[38]. All the available services are listed in the DNS table. At node 1 (Figure 7.16), the DNS table only contains the considered-as-best services received by its CH node 2. The metric values for the three available http service at node 2 are:

- web-15 with metric=10

[38]The routing table of clusterheads do not include nodes from inside the cluster. Therefore, the routing table of CH 2 only contains 18 entries out of 20 nodes.

- web-7 with metric=26
- web-6 with metric=85

Therefore, node 1, only sees the best available service, in our example web-6, in its list.

7.6 Conclusion

In this chapter we showed how each of the previous contributions work together to form our service discovery architecture. The choice of Zeroconf in ad hoc networks is discussed. Furthermore, the combination of our SLSF/SLSR protocol as replacement of the multicast structure in Zeroconf is presented. Metrics that permit context-awareness in service discovery are proposed and we described a further step, we want to make in a future work, towards social context by applying collaborative filtering techniques to service discovery.

We describe our service discovery architecture and its ability to transparently use the existence of an infrastructure when it is available or be independent when it is not. Finally, we present the conducted real world experiments using Zeroconf on top of SLSR.

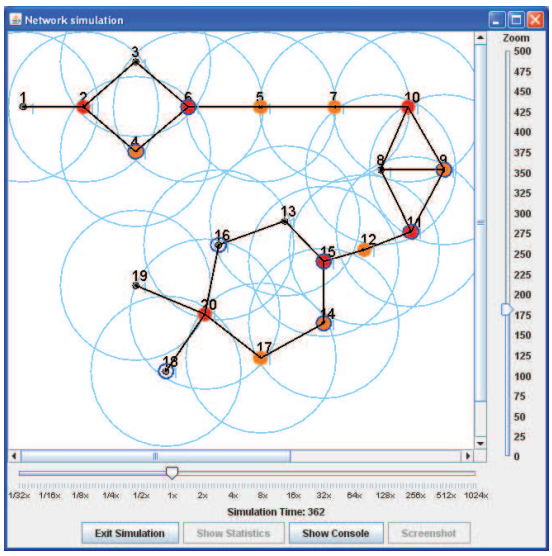


Figure 7.14: DNS Table view.

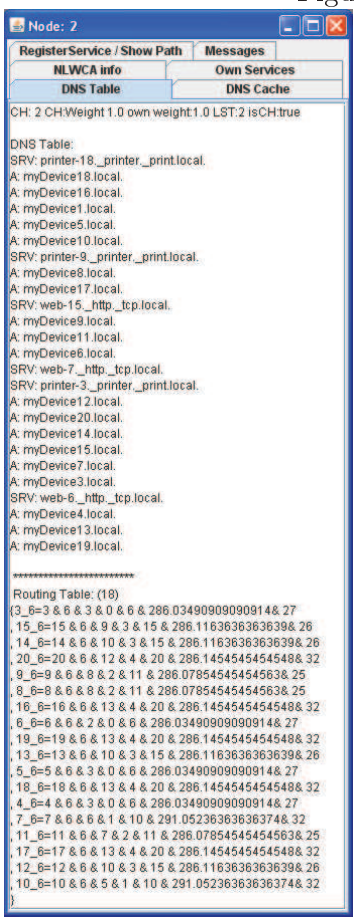


Figure 7.15: Zeroconf available services.

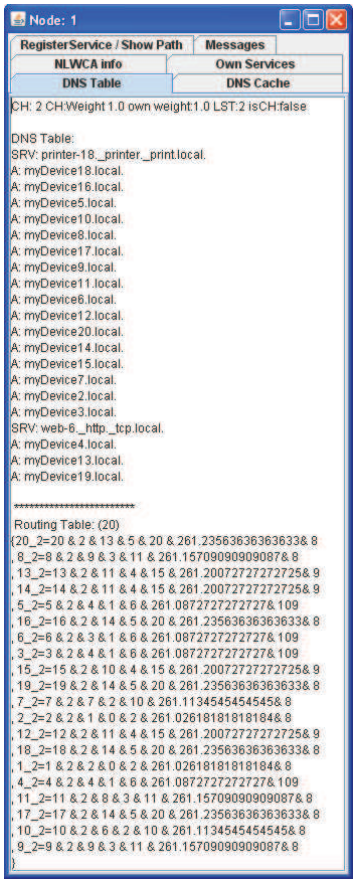


Figure 7.16: DNS cache view.

Conclusion and perspectives

Conclusion

Conclusion

In the first part of this thesis we introduced the various domains that address, and are related to, service discovery in ad hoc networks. Our approach was to consider service discovery as the target application to improve, but while keeping in mind the bigger picture and all the elements that are involved in the service discovery process. The goal of this thesis was to:

> Provide service discovery specifically adapted to ad hoc networks with an architecture that can deal with high mobility, numerous available services and heterogeneous devices by adapting itself to the surrounding context.

To address this problematic, we propose contributions in several domains. They each improve service discovery at a different level. An important goal we had when designing those solutions, was to obtain a symbiotic behavior between the various layers of protocols or applications. A protocol or application should be able to profit from the efforts an underlying protocol already did, however, without degrading the latter's operations.

This thesis was the opportunity to collaborate with several universities and research teams on several domains, in particular, within the ANR SARAH project on service discovery, with the University of Luxembourg on dissemination and collaborative filtering, and with the Maia team on the multiagent domain. The contributions are the following:

- Dissemination:
  - We propose a dissemination structure, Stable Linked Structure Flooding, that is based on an one hop clustering structure. Compared to other dissemination protocols, using clustering and wisely selecting the intermediate forwarding nodes, we are able to reduce the overhead needed to set up this structure and reduce the number of forwarding nodes. Additionally, we add a fault-recovery mechanism that locally provides robustness between two clusters.
- Routing:
  - Taking advantage of the underlying SLSF structure, we propose a routing protocol that can profit from network traffic to piggy-back the routing announces. The proposed routing protocol, SLSR, can adapt its effort to the context, by reducing the spreading of certain routing information when not needed or augmenting it when the complete network topology is requested.
- Collaborative filtering:
  - Relying on the same cluster structure as SLSF, we proposed a delay-tolerant collaborative filtering algorithm that provides good results on a data movie set. In this

algorithm, nodes only exchange user profiles with nodes within the same cluster. Thereby, they obtain a prediction rate for movies rated by similar users. We propose to use collaborative filtering for service discovery. Thereby, in a service rich environment, users filter out only the services that are recommended by other like-minded users.

- Simulation tools:
  - To assist research for service discovery, we propose a distributed framework that builds on the use of multi-modeling and co-simulation that is able to take into account both users behaviors and network performances. Our approach has: 1) the ability to take mutual influences of users behaviors and network performances into account; 2) the ability to design usage scenarios with heterogeneous users behaviors; 3) the ability to benchmark a network protocol against a wide range of usage scenarios.
  - at the same time, we developed several smaller tools that visually assist simulations in order to ease the analysis of a protocol or application.
- Service discovery in ad hoc networks profits from each of these contributions in a different manner:
  - Dissemination: Service discovery disseminates its service announces through the network. Thus, an improved dissemination also improves the overlying service discovery in terms of network overhead and effort.
  - Routing: In some cases during discovery, nodes need to be contacted individually (e.g. obtain additional information form a particular service/node) and also once the service is discovered, to use it. Routing provides the path(s) to reach the desired node using unicast.
  - Collaborative filtering: In a dense and crowded network where numerous services are available, collaborative filtering can provide only the best subset of services, rated by similar users, instead of an unmanageable long service list.
  - Simulation tools: The tools developed and proposed do not help service discovery directly, but help more generally the research on service discovery. In particular for ad hoc networks those tools provide user scenarios and new, more adapted and realistic, mobility models.
  - Zeroconf: finally we used Zeroconf, a standardized and widely used service discovery protocol, and proposed to adapt it to ad hoc networks by replacing the multicast with our SLSF dissemination structure.

Perspectives

Perspectives

Context and metrics for service discovery

In this thesis, we propose a service discovery architecture that is able to incorporate context and metrics at each decision level. In a future work, we plan to incorporate additional metrics and more importantly to analyze which metrics are the most relevant. Our goal is to capture the situation in which the user, hence also the device itself, is and adapt the protocol configurations or even select another, better suited, protocol.

Collaborative Filtering

In another future work, we will investigate how much we can gain using collaborative filtering on services. If we create an ad hoc network using today's smart-phones, with their recent tremendous popularity, we would have a large number of relatively powerful devices offering services. To cope with this overload of information, collaborative filtering could select the service(s) recommended by similar users. Furthermore, similar users could be determined based on their social network profile. Doing so, ad hoc networks could alleviate the cellular network for local tasks. Social networks could also work based only on the locally created ad hoc network, thus the natural proximity.

SLSR evaluation

In a very near future work, we will evaluate in detail the proposed SLSR protocol. We plan to use the mobility models that include a closed-loop between the user and the network in order to investigate the limits and performances of SLSR. As a further analysis, we specifically focus on the gain of the symbiotic behavior.

Towards standard mobility models

In the short term, we plan to show the disruptive effect of a non conforming behavior and to extend our experiments to more advanced protocols and scenarios. In the longer term, the framework will be enriched with a set of mobility models and a set of environmental models. We also plan to have a real setup (a typical existing room or building or city modeled in 3D from real data for example). These sets can serve as references that could be used to assess the performances and applicability of a solution, and validate it in certain contexts. This could be a good basis to provide the ubiquitous computing community with a benchmarking evaluation toolkit.

Experimentation

To assess our simulations, we plan to experiment our protocols SLSF and SLSR, with Zeroconf on top, on a larger scale network (10-20 users). It will be especially interesting to compare the experimental results with the simulations done with our proposed mobility models. To go even further, we could then fine-tune our users' behavior using the results from our experiments to obtain more realistic simulation results.

Appendix A

Service discovery protocols

A.1 UPnP – Universal Plug and Play Networking

UPnP is a set of protocols steered by the "UPnP Forum", composed of many different industry members, to enable, mainly in home/residential networks, the transparent interconnection of devices and services. UPnP defines a protocol stack largely inspired by web application design (Figure A.1). Two types of devices are defined: controlled devices and control points. Controlled devices, shortly called devices, propose services or capabilities (such as media display) and respond to incoming requests from control points. Control points whereas, can be seen as the remote control for the devices, usually the client of the service. They send out requests and service controls and receive events regarding the controlled services from controlled devices. UPnP is defined as 5 steps (6 with IP) from no networking at all to full service usage and control.

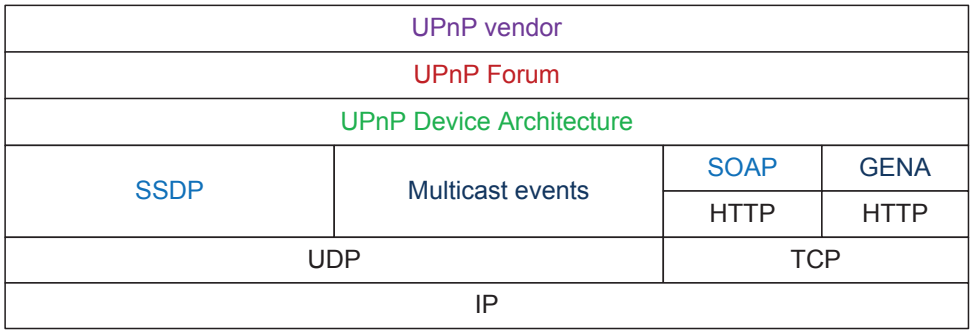


Figure A.1: UPnP protocol stack.

A.1.1 Step 0: IP Address

As in any IP based network the first operation before being able to use the network is obtaining an IP Address. This step is the same as the link-local addressing in Zeroconf (Section 2.3.3). Three possibilities: the presence of a DHCP server, manual configuration or Auto-assignment of an IP address. Strategies and address range remain the same as for Zeroconf.

A.1.2 Step 1: Discovery (SSDP)

Once an IP address is obtained the first actual UPnP step is discovery. UPnP uses SSDP (Simple Service Discovery Protocol) to discover devices or services. A (controlled) device entering

a network advertises itself and the proposed services via multicast/UDP using an SSDP advertisement message (Figure A.2) to the control points. The other way around, a control point entering a network searches for services of interest using a search message (Figure A.3). Search target can be from all services "ssdp:all" to specific types of services defined by the UPnP forum "urn:schemas-upnp-org:service:serviceType:ver" or specific vendor defined types "urn:domain-name:device:deviceType:ver". Messages are sent in multicast/UDP. All matching controlled devices respond to this search query with an SSDP response which contains the Location URL for the service.

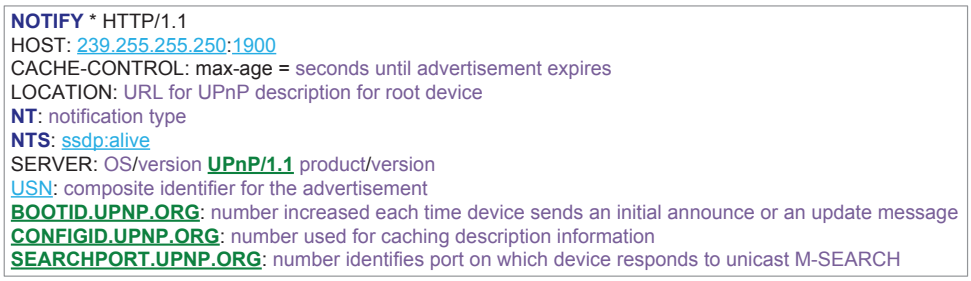


Figure A.2: format of a SSDP service notification message. Services are identified by their Unique Service Name (USN).

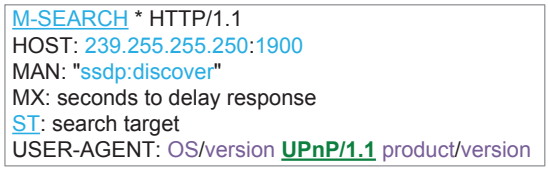


Figure A.3: format of a SSDP search message. ST is specifies the service type to discover.

A.1.3 Step 2: Description

At this stage, the control point discovered the services matching its request but does not know how their specific options or how to interact with them. To obtain further information about the service, the control point uses the URL obtained in the response message and retrieves an XML formatted file using HTTP from the distant device hosting the service (Figure A.4). The returned XML file provides the necessary information to interact with the device or service. Formats of the XML description for a device and a service can be found in Appendix B.

A.1.4 Step 3: Control

The control point is now aware of the device/service capabilities and the necessary information to use and control the service. Similar to a remote procedure call, services are controlled using the SOAP[39] protocol. Messages are transferred via HTTP over TCP over IP. SOAP defines the use of XML and HTTP for remote procedure calls. and UPnP 1.1 uses HTTP to deliver SOAP 1.1 encoded control messages to devices and return results or errors back to control points.

[39]Simple Object Access Protocol[Simp 00]

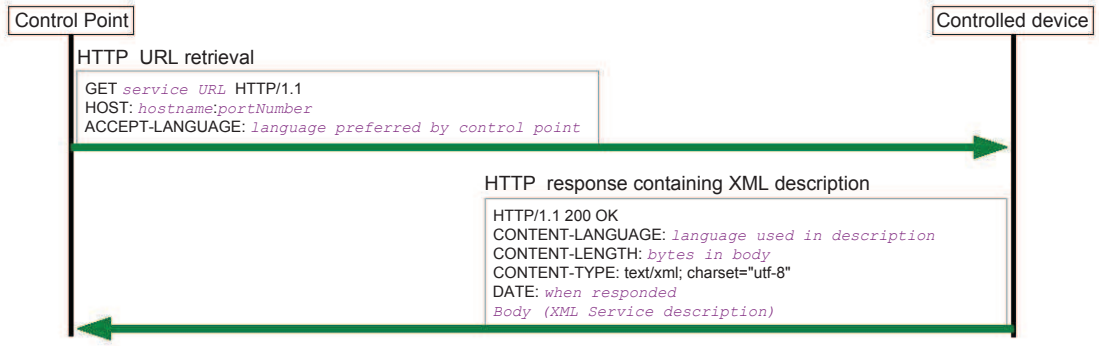


Figure A.4: UPnP service description retrieval.

A.1.5 Step 4: Eventing

In order to receive state information for services of interest or currently used, control points can subscribe to events, using a subscribe message, on controlled devices. Each time the state on a controlled device changes, regardless the reason of the change (e.g. response to an action, running service evolution), it notifies the new state information in an event message. Event messages contain the names of one or more state variables and the current value of those variables expressed in XML. Eventing messages can be sent over TCP in unicast, as found in UPnP 1.0, or over UDP in multicast to reach multiple control points with one message.

A.1.6 Step 5: Presentation

The initial service discovery response, additionally to the service URL, may contain a "Presentation URL". This url is simply the URL of an HTML page that can be retrieved. The content of this presentation page is completely vendor specific. The degrees to which a control point is able to display the presentation page depend on its capabilities. The page must be an HTML page and it is recommended to be based upon XHTML-Basic.

A.2 Jini

Jini is one of the main service discovery protocols that where widely deployed and used. Also, Jini has a very different approach compared to Zeroconf or UPnP by using RMI and the Java environment. For these reasons we describe the service operations in detail. Jini distinguishes 3 types of entities:

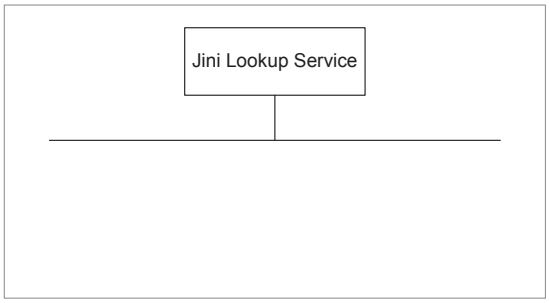
- The Jini Client: entity using a service.
- The Jini Service: entity that provides the service.
- The Jini Lookup Service: entity that provides the information about services in the network (directory)

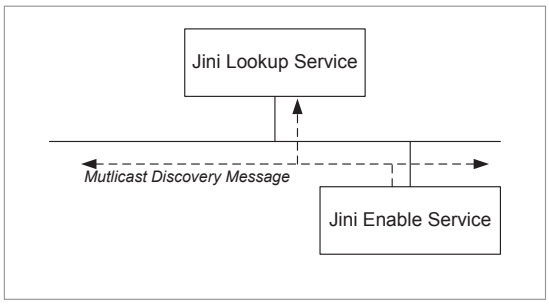
Following sections show the process of discovery, join and lookup for a client and a service [Oliv 00].

A.2.1 Service interactions with Lookup Service

The following 4 steps show the interactions between a Jini enabled service entering a network and a lookup service.

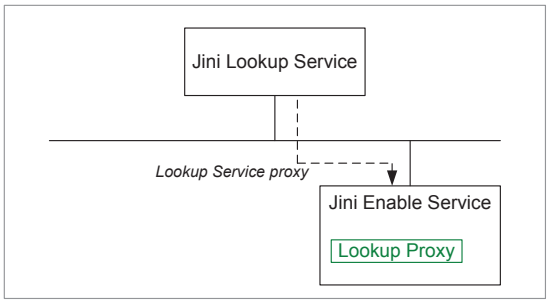
1) A Jini Lookup Server is set up on the network.

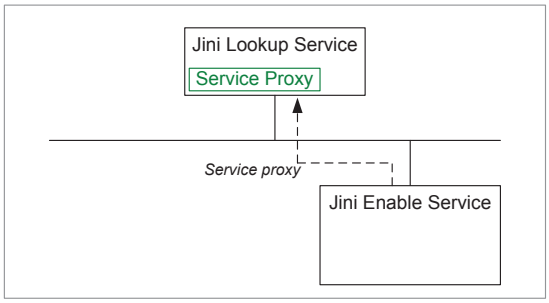




2) Upon arrival on the network the Jini Service sends a multicast Lookup Discovery message.

3) Any Jini Lookup Server in range of that multicast message responds to the service with a unicast message containing a proxy to connect to itself.



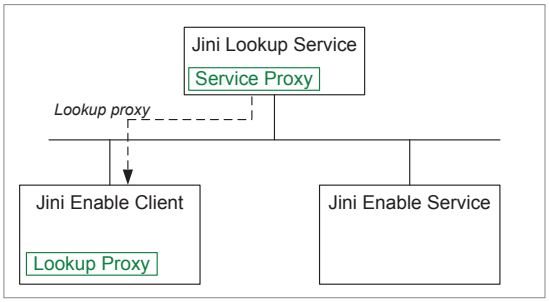


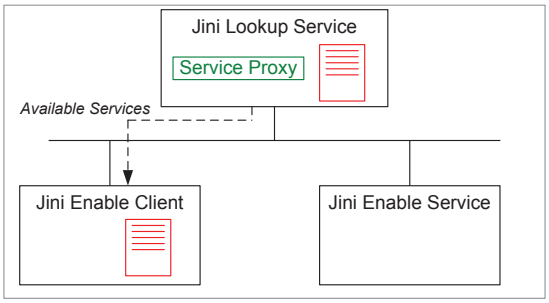
4) The service joins the Jini Lookup Service by sending back a unicast registration message using the Lookup proxy. The message contains a Service Proxy pointing to the joining service. A copy of that proxy is later on sent to Jini clients so they can reach the service directly. Additionally to the proxy, the registration message also contains the service attributes (e.g. human readable description, physical location).

A.2.2 Client interactions with Lookup Service and Jini Service

Steps 1 to 3 are the same for the Jini Client side to discover the lookup service on the network and obtain its lookup proxy. Following steps show the interactions from service discovery to service usage.

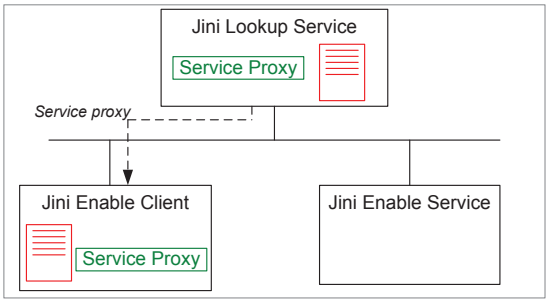
1-3) A client joining the network proceeds to the same steps 1, 2 and 3 as for the Jini Service. Send a multicast lookup message and receive a Lookup Proxy as response.

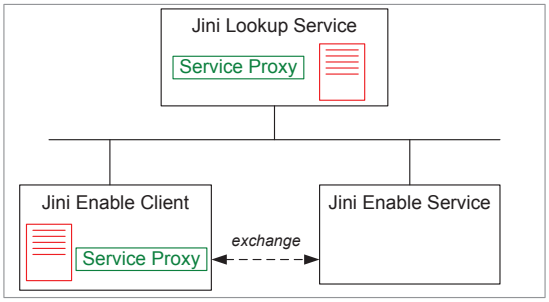




4) The Jini Client sends a unicast service request to the lookup service which responds with the matching available service list.

5) The client selects the desired service from the list and receives from the Lookup Service the corresponding Service Proxy.





6) Using the provided service proxy the client contacts the service directly, the exchange and service usage starts.

Services can represent hardware or software. Everything has to be able to represent itself as a software entity with an interface known to any client. The Discovery process in Jini is not the discovery of services in the network but is the discovery of the Lookup services in the network. For redundancy on network partitions, several Lookup services can be in the same network. They are responsible for listing the available service and pass, to requesting clients, the necessary means (a service proxy) to contact the desired service. Services registered to the Lookup service have a lease time after which the lease has to be renewed by the Jini Service. The lease can be canceled by the service on a graceful shutdown or on a time out (e.g. due to device crash or network errors), which erases the service from the available services on the Lookup service. Jini also provides the ability to register to remote events. By doing so the client is informed by the service about specific events. Remote events also have a given lease time that times out if not renewed. Transactions in Jini maintain distributed data consistency among clients and services. Transactions are groups of operations where each of them reports back the success or failure to a Transaction Manager. If operations do not complete within the transaction lease period, a roll back command is issued to all participants.

Appendix B

UPnP - XML service and device description

```
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0"
 configId="configuration number">
     <specVersion>
        <major>1</major>
        <minor>1</minor>
     </specVersion>
<device>
        <deviceType>urn:schemas-upnp-org:device:deviceType:v</deviceType>
        <friendlyName>short user-friendly title</friendlyName>
        <manufacturer>manufacturer name</manufacturer>
        <manufacturerURL>URL to manufacturer site</manufacturerURL>
        <modelDescription>long user-friendly title</modelDescription>
        <modelName>model name</modelName>
        <modelNumber>model number</modelNumber>
        <modelURL>URL to model site</modelURL>
        <serialNumber>manufacturer's serial number</serialNumber>
        <UDN>uuid:UUID</UDN>
        <UPC>Universal Product Code</UPC>
        <iconList>
            <icon>
               <mimetype>image/format</mimetype>
               <width>horizontal pixels</width>
               <height>vertical pixels</height>
               <depth>color depth</depth>
               <url>URL to icon</url>
            </icon>
            <!-- XML to declare other icons, if any, go here -->
        </iconList>
        <serviceList>
            <service>
               <serviceType>urn:schemas-upnp-org:service:serviceType:v</serviceType>
               <serviceId>urn:upnp-org:serviceId:serviceID</serviceId>
               <SCPDURL>URL to service description</SCPDURL>
               <controlURL>URL for control</controlURL>
               <eventSubURL>URL for eventing</eventSubURL>
            </service>
            <!-- Declarations for other services defined by a UPnP Forum working committee
               (if any) go here -->
            <!-- Declarations for other services added by UPnP vendor (if any) go here -->
            </serviceList>
            <deviceList>
            <!-- Description of embedded devices defined by a UPnP Forum working committee
               (if any) go here -->
            <!-- Description of embedded devices added by UPnP vendor (if any) go here -->
            </deviceList>
            <presentationURL>URL for presentation</presentationURL>
 </device>
</root>
```

Figure B.1: UPnP device description format.

```
<?xml version="1.0"?>
<scpd
 xmlns="urn:schemas-upnp-org:service-1-0"
 xmlns:dt1="urn:domain-name:more-datatypes"
 <!-- Declarations for other namespaces added by UPnP Forum working committee (if any) go
     here -->
 <!-- The value of the attribute must remain as defined by the UPnP Forum working committee.
     -->
 xmlns:dt2="urn:domain-name:vendor-datatypes"
 <!-- Declarations for other namespaces added by UPnP vendor (if any) go here -->
 <!-- Vendors must change the URN's domain-name to a Vendor Domain Name -->
 <!-- Vendors must change vendor-datatypes to reference a vendor-defined namespace -->
 configId="configuration number">
  <specVersion>
        <major>1</major>
        <minor>1</minor>
  </specVersion>
  <actionList>
        <action>
          <name>actionName</name>
          <argumentList>
                <argument>
                  <name>argumentNameIn1</name>
                  <direction>in</direction>
                  <relatedStateVariable>stateVariableName</relatedStateVariable>
                </argument>
                <!-- Declarations for other IN arguments defined by UPnP Forum working
                    Committee (if any) go here -->
                <argument>
                  <name>argumentNameOut1</name>
                  <direction>out</direction>
                  <retval/>
                  <relatedStateVariable>stateVariableName</relatedStateVariable>
                </argument>
                <argument>
                  <name>argumentNameOut2</name>
                  <direction>out</direction>
                  <relatedStateVariable>stateVariableName</relatedStateVariable>
                </argument>
                <!-- Declarations for other OUT arguments defined by UPnP Forum working
                    committee (if any) go here -->
                </argumentList>
        </action>
        <!-- Declarations for other actions defined by UPnP Forum working committee
          (if any)go here -->
        <!-- Declarations for other actions added by UPnP vendor (if any) go here -->
  </actionList>
  <serviceStateTable>
        <stateVariable sendEvents="yes"|"no" multicast="yes"|"no">
          <name>variableName</name>
          <dataType>basic data type</dataType>
          <defaultValue>default value</defaultValue>
          <allowedValueRange>
          <minimum>minimum value</minimum>
          <maximum>maximum value</maximum>
          <step>increment value</step>
          </allowedValueRange>
        </stateVariable>
        <stateVariable sendEvents="yes"|"no" multicast="yes"|"no">
          <name>variableName</name>
          <dataType type="dt1:variable data type">string</dataType>
          <defaultValue>default value</defaultValue>
          <allowedValueList>
                <allowedValue>enumerated value</allowedValue>
                <!-- Other allowed values defined by UPnP Forum working committee
                  (if any) go here -->
                <!-- Other allowed values defined by vendor (if any) go here -->
          </allowedValueList>
        </stateVariable>
        <stateVariable sendEvents="yes"|"no" multicast="yes"|"no">
          <name>variableName</name>
          <dataType type="dt2:vendor data type">string</dataType>
          <defaultValue>default value</defaultValue>
        </stateVariable>
        <!-- Declarations for other state variables defined by UPnP Forum working committee
          (if any) go here -->
        <!-- Declarations for other state variables added by UPnP vendor (if any) go here -->
  </serviceStateTable>
</scpd>
```

Figure B.2: UPnP service description format.

Appendix C

Research tools

C.1 Network and graph visualization

In addition to available visualization of simulators that usually show the nodes moving on the simulation field a more topological view can be very useful. Graph visualization is not only useful to represent the network topology itself but also the protocol interactions, the different hierarchical views (e.g. cluster view), the message flows or any other interesting data about the nodes. Two tools were used for visualizing such graphs during my thesis, Graphviz and JUNG.

```
Graph g {
"node 1" -- "node 9";
"node 1" -- "node 8";
"node 8" -- "node 7";
"node 9" -- "node 7";
"node 8" -- "node 5";
"node 8" -- "node 3";
"node 9" -- "node 2";
}
```

Figure C.1: Example of a DOT-language file.

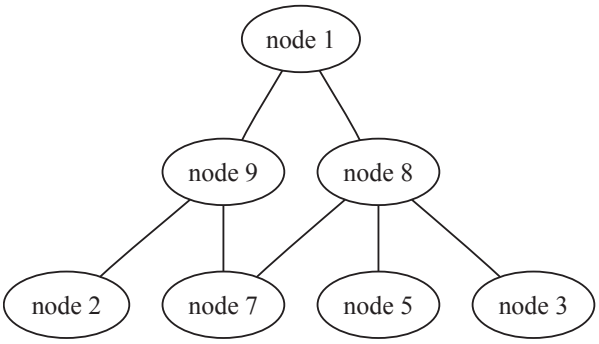


Figure C.2: Example of a graph generated by graphviz.

C.1.1 Graphviz

Graphviz[graphviz] is a very popular open source graph visualization software. Graphviz uses a simple graph definition language named DOT (Figure C.1). Several different graph layouts automatically arrange the nodes on the graph (Figure C.2). Those layouts can handle very large graphs, sub layouts can be displayed and many more other options. Graphviz can also be used as a library integrated in another visualization tool. However, graphviz has a static approach and graphs can not be updated on the fly. Ad hoc networks and their evolution are very dynamic data that need to be represented also dynamically.

C.1.2 JUNG

JUNG for Java Universal Network/Graph Framework [jung] is an open-source visualization tool for any data that can be represented as a graph or network[40]. It supports dynamic graphs using filters or by adding or removing nodes (also called vertices) or links (also called edges) from the network/graph. It provides great flexibility on the representation of the vertices and edges in the graph. Vertices and edges can be any JAVA object with any attributes. A vertex is then represented by one type of JAVA Object A and an edge is another JAVA Object B that simply links two vertices of type A. The attributes and functions of the objects can be completely independent of JUNG. To provide very easy and flexible customization of the graph (e.g. shape of a node, color, size, lines, etc.), numerous so called *Transformer* functions are proposed.

A Transformer for a vertex is simply a function that takes as input attribute the vertex Object A and gives as output result the attribute to change. For example to change the color of a vertex a simple Transform function as depicted in Algorithm 4 sets the vertex color to computed result of the object's function *compute_Color_From_Attributes*. Of course it is dynamic and the color of the vertex in JUNG changes if the result of *compute_Color_From_Attributes* changes. Exactly the same is possible for edges, where a transformer function taking as attribute edge Object B is used.

Algorithm 4: Jung transformer function to set the color of a vertice of type A.

```
1 public Paint transform(A vertex) {
2    return vertex.compute_Color_From_Attributes();
3 }
```

Any data or result of a computation can, using such transformer functions, be displayed by JUNG in any configurable way. For example, the more a node has neighbors the bigger the size of the node. Or the color of a node depends on its status (e.g. clusterhead, slave, etc.).

Nevertheless, JUNG also provides automatic positioning of the vertices and edges of the graph. Some of those layouts provide only a static layout (graph layout is only computed once at startup) as it is the case with graphviz, others provide a dynamic layout to permit nodes and edges to be added dynamically to the graph.

JUNG also provided graph/network algorithms that such as shortest path, minimum spanning tree and centrality measures. One single graph can also be represented in several views. Figure C.3 show both, multiple views of one single graph and the minimum spanning tree on one of those views.

C.2 Implemented simulation tools

To facilitate and assist the process of simulations and their analysis we implemented tools that improve the simulation or the usage of the simulator.

C.2.1 Click-and-play mobility tools

As a first enhancement of the JANE simulator concerning the early simulation process we implemented an improved *click and play mobility model*. The click and play mobility model is a simple mobility model that is driven by the user interacting with the simulation nodes with its mouse. This model is very useful for first tests on a protocol by controlling every node's movement. However, the JANE *ClickAndPlayMobilitySourceSimple* provides only very limited

[40] The descriptions made here corresponds to JUNG version 2

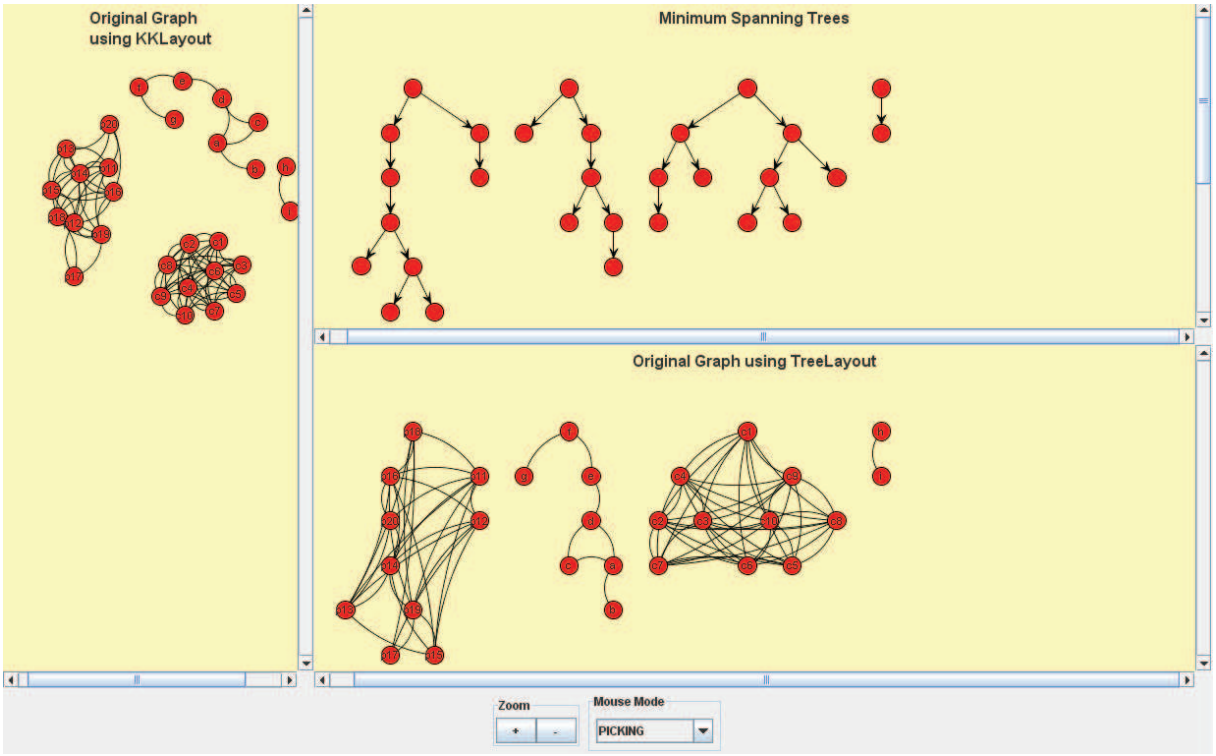


Figure C.3: Minimum spanning tree demo from [jung]. Demonstrates JUNG's ability to show multiple views of the same graph and to extract the Minimum Spanning Trees.

interactions: a node can only be moved to a given location and once its movement started it can not be interrupted or changed until the node arrives at destination. To provide more flexible movement controls in the click and play mode, we add following features:

- The ability to interrupt nodes during their movement.
- Select nodes with left click and de-select nodes with right click.
- Pressing shift while selecting a destination moves the node *towards* the destination. The nodes moves in the direction given by the vector between the initial coordinate and the clicked position. The node then only stops until it reaches the border of the simulation area.
- Middle-click while a node is in movement stops the node from moving.

The new mobility source *Advanced_ ClickAndPlayMobilitySource* [41], provides those features by changing the way the movement are internally executed. In the original *ClickAndPlayMobilitySourceSimple* a movement is composed of the initial position, the destination position and the node's speed. Once the movements is executed internally by JANE it cannot be interrupted until the node has reached the destination position. To be able to interrupt a movement and change the destination, our *Advanced_ ClickAndPlaySimulationFrame*) implementation splits the initial movement into several smaller steps. The initial distance is then divided into s steps given by the following:

[41] composed of two classes: the *Advanced_ ClickAndPlayMobilitySource* and the *Advanced_ ClickAndPlaySimulationFrame*

$$s = \frac{1}{(distance(intit_pos, dest_pos))/node_speed}$$

For example, an initial position of $x, y = 100, 100$ and a destination $200, 100$ ($distance(init, dest) = 100$) with a speed of 20 units/per second divides the 100 unit distance into 10 steps of 10 units each. At each of these steps, the movement can be interrupted and a new destination can be assigned where new steps are computed.

C.2.2 Mobility scenarios

Before the AA4MM framework was available, we implemented a first version of a simulation externally controlled by an other simulator providing mobility patterns (Figure C.4). As mobility simulator we used the Simbad simulator [simbad] [Hugu 06]. Simbad is a java 3d robot simulator that was kept voluntarily simple. The simulation has an environment where objects, such as walls, can be placed. The robots operate inside this simulation area and are equipped with sensors (e.g. vison sensors, range sensors, contact Sensor). Robots can have simple behaviors, using this sensor information, and adapt their movement to the environment (e.g. by avoiding walls or other robots).

To link the JANE simulator with the Simbad simulator together we created a new mobility model in JANE named *AWR_ RobotMobilitySourceSimple*. The connection between the two simulators is simple, each time JANE awaits a new position for a given device in the network simulation [42], the Simbad simulator is queried for the position of the device at the robot simulation. As a consequence, the nodes in the JANE simulator obtain their position directly from Simbad. However, the Simbad simulation time is not strongly connected or synchronized with the JANE simulation time. Therefor, if one of both simulators is under heavy computation, it might happen that their time values get out of synchronization. JANE takes the current position of the given device, regardless the time values. This timer issue is one of the contribution that is solved in Chapter 5 by the AA4MM framework.

C.2.3 Visualization pane with Jung

During the design and debugging of a protocol, one must often read the information contained at each individual node to comprehend the problem or behavior correctly. In JANE however, except for a console output, there is no such feature. To fill this gap, we propose a simple additional local view interface that contains a list of all the node (Figure C.5). A user can than display for each node the current state or any other information of the currently simulated protocol. For example, when simulating OLSR, the user can display the local routing table of the selected node.

To provide a global of view of the network, we propose a second interface that displays the network connections. To do so, we use the Jung visualization tool. The interface, on the right on Figure C.5 is composed of three parts. The first is the parameter panel where the user can for example select two vertices in a list and display the shortest path available between those two vertices. The second, is the left pane that shows the current status of the network with coloring and dimensions adapted to the network information (here colors display the number of neighbors). The third pane is the same view as the second pane except that one can navigate through the history of network connections.

[42] JANE requests new positions at the mobility model (here *AWR_ RobotMobilitySourceSimple*) in the inner class *DeviceEntity* with the *getNextArrivalInfo* method.

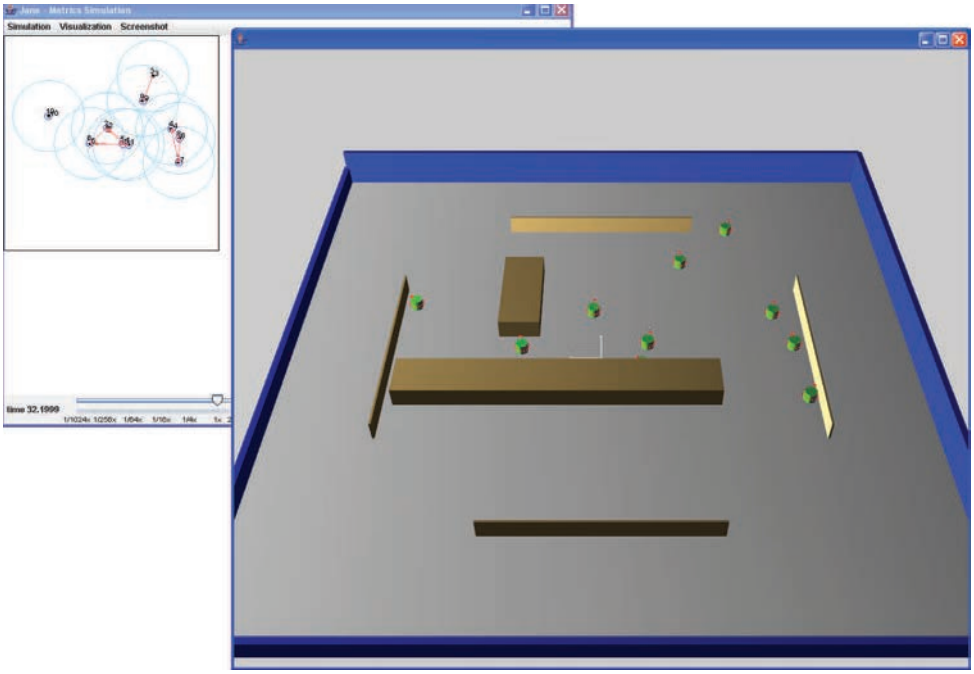


Figure C.4: JANE (left) with Simbad (right) simulation.

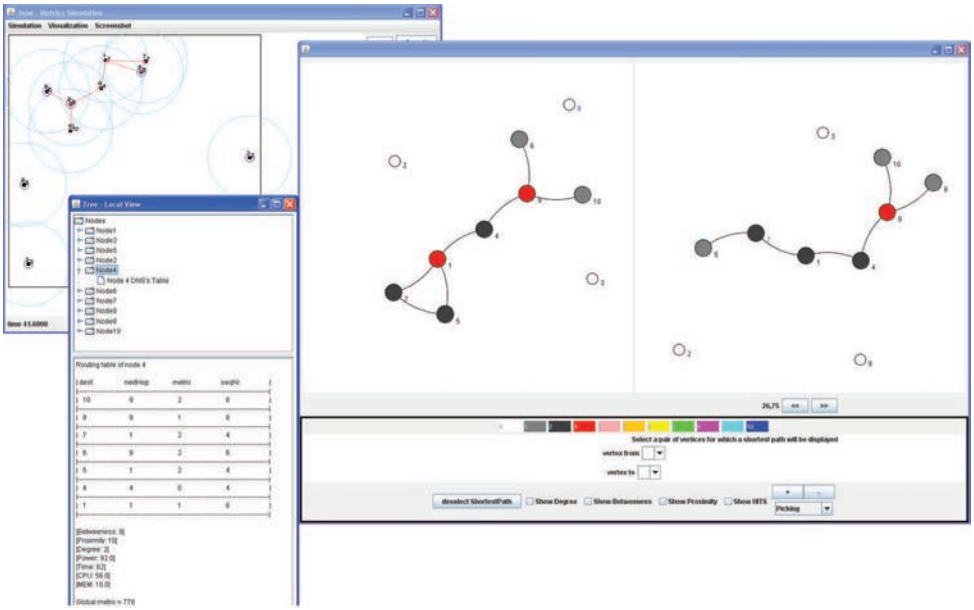


Figure C.5: JANE (top left), Local view interface (bottom left) and Jung visualization pane (right).

Appendix D

Implementations

D.1 OLSR

We implemented the OLSR (version 1) protocol as a reference implementation in the SARAH project. The implementation was done by following the rfc [Clau 03] and keeping only the features relevant to our simulated environment. The implementation is done for the JANE simulator. It is integrated in the JANE environment as a Runtime Service as shown in Figure D.1. The operating system, local to each device, provides the basic communication paradigms such as sending messages (here a broadcast provided by the *sendBroadcast* method) and its opposite, receiving messages (here implemented in form of a handle<Type of message> method; e.g. *handleHelloMessage*). The local operating system, referred in the implementation as RuntimeOperatingSystem, is also present when using JANE in platform mode (i.e. on real devices). On the contrary, the Simulation OS is only present when using JANE in simulation mode, likewise for the Global Operating System. Attached to the global operating system are here two global services: CollisionFreeNetwork and LogToFileService.

The first, CollisionFreeNetwork, is a global service provided by JANE that simulates a network where no collision between network packets occur. This is configurable and not actually part of the OLSR implementation, but more as a simulation parameter. The second, LogToFileService, is a logging service to gather the simulation results acquired on all the JANE simulated devices and log them into a file for further processing. The figure illustrates well the separation of the protocol implementation part and the simulation part. When using, here, the implemented OLSR protocol on real devices, JANE cuts its architecture between the Operating System and the Simulation OS and plugs in, among others, the real world networks sockets and timer values. Figure D.2, depicts the operating system view of JANE where both simulation and platform Core are separated. From a implementation point of view, there is absolutely no difference, the transition from simulation is transparent. Hence, for example, the simulation time is replaced by the real device's local time, the simulation id/address is replaced by the device's network IP address and the communications are now through the real network interfaces (e.g. the *sendBroadcast* method now reach a JAVA socket instead of the simulation OS).

The implementation is divided in four parts:

- the main part with the OLSR service, containing all the protocol logic and interactions
- message structures
- repositories and data structures
- simulation and timer utilities

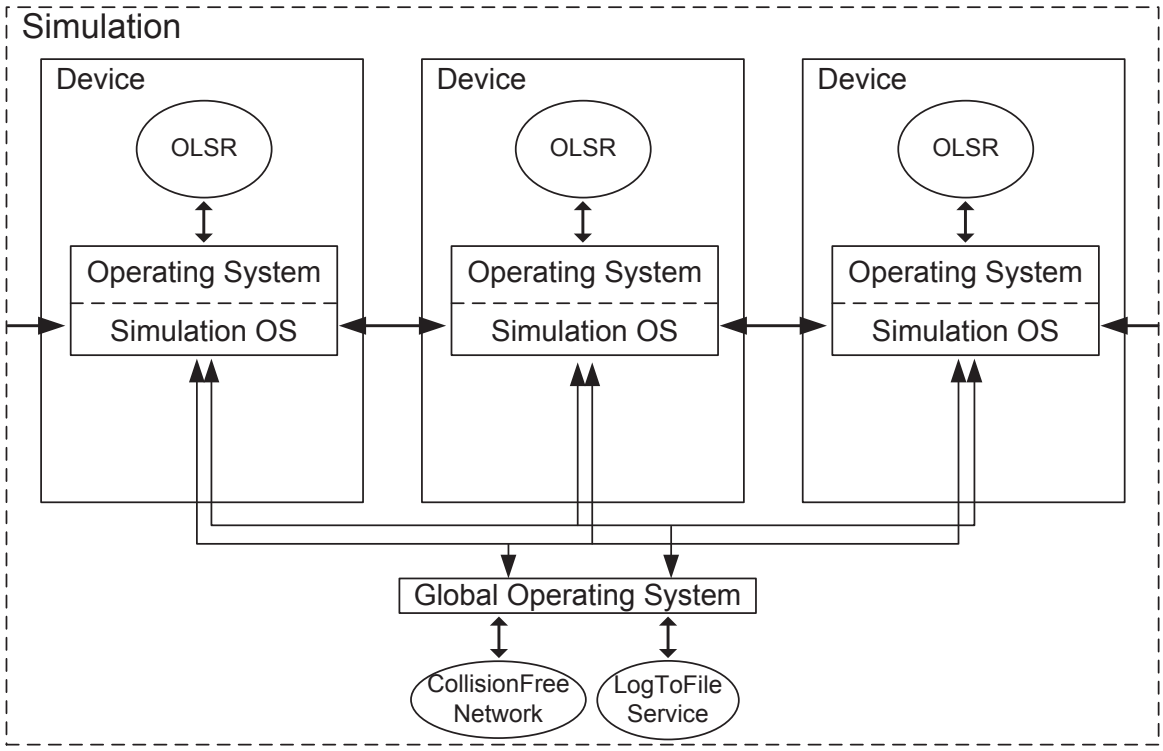


Figure D.1: OLSR implementation in JANE.

D.1.1 OLSR service

In JANE, each protocol is represented by a runtime service. JANE provides to such services, basic functionalities such as: sending and receiving messages, sensing the neighborhood (provided by the *OneHopNeighborDiscoveryService*), initialization and termination functions, providing runtime information (such as the simulation time) and timer functionalities.

The *OLSRservice* is in our case a JANE local service (i.e. one service instance per simulated node) that provides all the protocol functionalities:

- Send and receive HELLO messages, mainly used to obtain the two hop neighborhood, provided by two functions:
  - *generateHelloMessage*: triggered by a JANE timer every second to send a HELLO message containing the list of one-hop neighbors.
  - *handleHelloMessage*: triggered by JANE at each reception of a received HELLO message from a neighbor. This function will, among other functionalities, gather and update the two hop neighbors set and trigger a new MPR computation if necessary.
- Compute the MPR set. This functionality is composed of one main function and several helper functions:
  - *mpr_computeMPRset*: contains the main MPR computation logic and uses the following helper functions to make the code more readable and understandable.
  - *mpr_clean2HopNeighbors*: Performs cleaning of the 2-hop neighbors by excluding: the nodes only reachable by members of N with willingness WILL_NEVER, the node performing the computation and all the symmetric neighbors: the nodes for which there exists a symmetric link to this node on some interface.

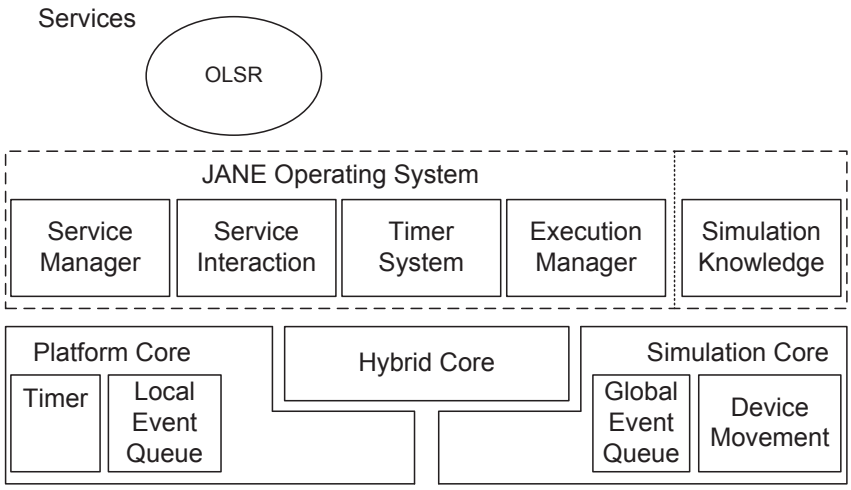


Figure D.2: OLSR in the JANE operating system.

- *mpr_selectWillAlwaysNodesAsMPR*: Selects as MPR, neighbors that advertise their willingness as WILL_ALWAYS.
- *mpr_calculateDy*: Calculates the D(y) for each neighbor. D(y) is defined as the number of 1-hop neighbors of node y that are: not neighbors of the node performing the computation, not the node performing the computation itself and not already selected as MPR.
- *mpr_computeTwoHopNeihgborsReachability*: Computes the 2-hop neighbors reachability. In other words: counts, for every 2-hop neighbor x, how many 1-hop neighbors can reach the given x 2-hop neighbor.
- *mpr_selectMPRwithReachabilityOfOne*: Adds to the MPR set the , which are the 1-hop neighbors that are the **only** nodes to provide reachability to a 2-hop node. For example, if 2-hop node b can be reached only through one 1-hop node a, then add node a to the MPR set.
- *mpr_removeCoveredNeighbors*: Removes the 2-hop nodes from the *twoHopNeighborsReachability* set which are now covered by a node in the MPR set.
- *mpr_updateNeighborTypes*: Updates the type of neighbors based on the received HELLO messages. Neighbors types are:
  NOT_NEIGH, SYM_NEIGH and MPR_NEIGH.

- Send and receive Topology Control (TC) messages. MPR nodes use TC messages to advertise their 1-hop neighborhood to all the network. Provided by three functions:
  - *generateTopologyControlMessage*: triggered by a JANE timer every 5 seconds on MPR nodes only.
  - *handlePayloadMessage*: triggered by JANE at each reception of a received TC message from a neighbor. Basically, decides whether or not the received TC message should be processed or not (using *processTopologyControlMessage*) , depending if it is a duplicate message, and also whether it should be forwarded or not, depending if the deciding node is selected as MPR or not.
  - *processTopologyControlMessage*: Processes the incoming TC messages and prepares the data for the routing table computation.

- Send and receive Payload messages. Can contain any payload message to be send using OLSR. Provided by two functions:

- *generatePayloadMessage*: to send any payload. In our simulations used to generate network traffic and gather performance information such as reachability.

- *handlePayloadMessage*: triggered by JANE at each reception of a received Payload message from a neighbor. In our simulations, this function logs the gathered simulation information.

D.1.2 Message structures

To communicate using JANE, there are two main type of methods: the send methods (e.g. *sendBroadcast* and *sendUnicast*) and the handle methods. While the send methods are located in the main JANE service, the actual handle method that is called by JANE is located in each message type. For example, the HelloMessage class contains a *handle(LinkLayerInfo info, SignalListener listener)* method that is called by JANE each time a message is received. This method then calls the previously mentioned *handleHelloMessage* in the OLSRservice where the processing of the message occurs. Each JANE message must implement the *LinkLayerMessage* class which requires the *handle* method. Among the actual fields of the message, there are two very important methods that should be implemented carefully. The first is the *copy()* method. The copy method is automatically called by JANE each time the message is "send" to another node. If the copy method is not correctly implemented and since it is a simulation, the message is not really send but rather a reference to the Java object is given to all recipients. If the first recipients that receives the message, modifies any field, it is actually modifying the common Jave object. Therefore, to avoid any concurrent modifications, JANE calls the copy method that should provide as output the complete copy of the object. Here, the word complete is very important since when, using for example Vectors in Java, passing a vector to a constructor, does not copy the vector but only copies the reference to the vector. To make a complete copy of a vector or data structure, Java often provides a *clone()* method.

In our implementation we have 6 message structures. OLSR uses a unified packet format for communication of any data related to the protocol. Each packet, can contain several OLSR base messages. And finally those base messages contain the protocol messages such as the HELLO and Topology Control. Additionally, HELLO messages contain an inner message structure called Link message. For our simulations tests and logging purposes we also use a generic Payload message that uses the OLSR default forwarding algorithm.

- *olsrBasePacket*: rfc defined base packet structure that contains one or more OLSR base messages. Contains the fields: packet length and packet sequence number.

- *olsrBaseMessage*: rfc defined base message structure that contains one or more OLSR base messages. Contains the fields: message type, validity time, message size and originator address, time to live, hop count, message sequence numbers and the message itself.

> **Sequence numbers wrap-around:** (Here as described in the OLSR RFC [Clau 03], but also valid for other protocols) The sequence numbers are represented by a limited number of bits, so there is a maximum possible sequence number. To overcome this limitation a wrap-around occurs: the incremented sequence number after the maximum possible value is zero. To be able to determine, even in case of a wrap-around, which sequence number is the latest (newest) one, the following algorithm is used:

> MAXVALUE designates the largest possible value for a sequence number. The sequence number S1 is said to be "greater than" the sequence number S2 if:
> S1 > S2 AND S1 - S2 <= MAXVALUE/2 OR
> S2 > S1 AND S2 - S1 > MAXVALUE/2

- *HelloMessage*: Implements the LinkLayerMessage from JANE, thus it is a final message. This message is as the rfc defined HELLO message structure that is send every second and advertises the one-hop neighbors only to the one-hop neighborhood. Each one-hop neighbor thereby obtains the two-hop neighborhood of the sending node. Contains the fields: reserved, hTime, willingness and the linkMessageSet.
  - *LinkMessage*: Is a sub-message of the HelloMessage. It contains the set of one-hop neighbors. Contains the fields: neighborType, linkType, linkMessageSize and neighborInterfaceAddress.
- *ToplogyControlMessage*: Also implements the LinkLayerMessage from JANE. This message is as the rfc defined Topology Control (TC) message structure that is send every five seconds by MPR nodes to advertise their neighborhood to the rest of the network. This permits each node in the network to discovery distant nodes and their neighborhood. Once a TC message is received from each MPR, OLSR can build its routing table containing the complete network topology. Contains the fields: ansn, reserved and advertisedNeighborMainAddress.
- *PayloadMessage*: Also implements the LinkLayerMessage from JANE. This message is not an rfc defined message structure. We use it only used for simulation and logging purposes. For our simulation, to measure reachability and follow the path of a single packet that is disseminated, we send such a Payload message that will trigger the logging on its reception by each recipient. In our simulations in Chapter 3 where a payload message is send every 10 seconds. Contains only inherited fields from the OLSR packet and OLSR base message structures.

D.1.3 Repositories and data structures

Protocol information and other data structures useful for the protocol functioning are stored in repositories. All OLSR protocol necessary repositories are contained in the class *OLSRInformationSets*. The repositories are represented in Java using *Vectors* or *Hashtables*. Each Vector or Hashtables contains then one of the helper Java objects. The repositories contained in the *OLSRInformationSets* class are as follows:

- *linkSet*: Is a Hashtable that contains as key the Address and as value the related link information in a *LinkTuple* object with the following fields: l_local_iface_addr, l_neighbor_iface_addr, l_SYM_time, l_ASYM_time, l_time, linkType and l_Lost_Link_time.
- *neighborSet*: Is a Hashtable that contains as key the Address of the neighbor and as value a *NeighborTuple* that contains the neighbor information with the following fields: n_neigbor_main_addr, n_status, n_willingness and neighborType. Note, that the neighbor information contains a n_neigbor_main_addr field that is the main interface used by the node in case of multiple interfaces. Although this feature is not implemented, the data structures contain such elements to for future use.

- *twoHopMeighborSet*: Is a Vector that contains a *TwoHopNeighborTuple* object with the fields: n_neigbor_main_addr, n_2hop_neigbor_main_addr and n_time. The object contains, the one-hop neighbor (main) address and its two-hop neighbor that was discovered from the HelloMessages. The n_time field simply gives an expiration time for information.

- *mprSet*: Is a Vector that contains the one-hop neighbor addresses that are selected as MPR.

- *mprSelectorsSet*: Is a Hashtable that contains as key the address of neighbors that selected this node as MPR and as value a Double containing the validity time for that entry.

- *topologySet*: Is a Vector containing a *TopologyInformation* object. A *TopologyInformation* object is created or updated at each reception of an TC message and is used for the routing table computation. It contains the fields: t_dest_addr, t_last_addr, t_seq and t_time.

- *duplicateSet*: Is a Vector containing a *DuplicateTuple* object. The *DuplicateTuple* object is used to verify if a message has already been seen or not, it contains the fields: d_addr, d_seq_num, d_retransmitted, d_iface_list, d_Time and messageType. At each message reception, the ducplicateSet is checked and updated with new messages. Each entry also expires and is removed at time set in d_Time.

- *routingTable*: Is a Vector containing a *RoutingTableEntry* object. The *RoutingTableEntry* object represents one entry in the routing table and has the following fields: Address r_dest_addr, Address r_next_addr, int r_dist and Address r_iface_addr.

D.1.4 Simulation and timer utilities

To facilitate implementations and readability we used some utilities that provide basic functionalities.

JANE only provides a basic timeout feature such as: set a timeout for a given period and it will trigger an event once the timeout period is reached. However, in many protocols a periodic timer is necessary (e.g. trigger a HelloMessage every second). Another frequent task in protocols is waiting for a event to occur during a given time (e.g. waiting for a acknowledgement message from another node). For all those features we use the *XTimer* class that provides all those functionalities. XTimer provides several constructors that permits for both *Global* and *Local* services (they use different JANE operating systems; GlobalOperatingSystem and RuntimeOperatingSystem respectively):

- Fire an event at every given time span: Constructor takes as argument *initTick*: The time span in milliseconds the timer will fire a tick event.

- Run the timer for a given time: Constructor takes additionally as argument: *initSec*: The number of milliseconds the timer will run.

- If no *initSec* is provided: The timer runs endless and triggers a tick every *initTick* milliseconds.

A timer *id* can be assigned to each *XTimer* to be able to distinguish incoming ticks by using their id. Each timer can at every moment, be stopped, reseted or started.

In the OLSR implementation all timers and message triggering are managed by the *MessageTicker* class. This class simply starts the timer for each defined OLSR message (i.e. HELLO, TC and Payload). HELLO messages are triggered every second, TC messages every 5 seconds and Payload messages are triggered every *SIMCONFIG.broadcast_interval* (a simulation parameter).

The *LogToFileService* class, is a simple class that is used to log the simulation result in a file whose filename is given as a simulation parameter in *SIMCONFIG.fileName*.

D.2 NLWCA - Subhead avoidance

To implement the subhead avoidance mechanism (Section 3.3) into the existing NLWCA implementation. The implementation existing implementation was already realized for JANE environment. The changes to implement a switch to be able to turn of or on the subhead avoidance where very small. The NLWCA code affected by the change is only the method where the clusterhead election occurs. The method providing this is named *electLocalLeader* and is located in the *NLWCAService* class. The Listing D.1 shows a simplified version of the Java code for readability. To be able to disable or enable subheads in NLWCA, the lines 26, 27 and 30 where added and use the simulation parameter *SIMCONFIG.USE_SUBHEADS*.

D.3 SLSF and SLSR

The SLSF, hence also the SLSR protocol, rely on the NLWCA clustering protocol. JANE permits a clean separation between two protocols by providing a *ServiceEvent* mechanism that enables protocols to send events that can be catched by other protocols. Since SLSR is an improvement, with routing capabilities, of SLSF, the implementation of SLSR replaces the SLSF implementation. However, in a near future, the SLSR implementation will be implemented as an separated module instead of being completely integrated.

As basis for the SLSF/SLSR implementation, we used the existing WCPD implementation. The beacon structure is the same but SLSF has a new for the forwarding mechanism with the ICR selection and also a fault-recovery mechanism.

D.3.1 Service events

As shown on Figure D.3, NLWCA communicates with SLSR using ServiceEvents. Those events contain cluster status information. The event send out by NLWCA is of type *NLWCAEvent* and informs the recipient about following fields: clusterhead address, isClusterhead flag, isSubhead flag, list of stable neighbors, node weight and the linkStabilityTreshhold. SLSR, on reception of such an NLWCAEvent, simply updates its local status to reflect the new NLWCA status. Doing so, the NLWCA implementations remains completely independent of any above lying protocol. In the code, to be receive event from another JANE service the service must implement the *EventListener* that requires a *handle()* method and declare its existence in the start method using following line *runtimeOperatingSystem.registerEventListener(new NLWCAEvent(), this);*.

D.3.2 Beaconing

NLWCA and SLSF both use beacons for their communications (Section 1.2.4 and 3.4). JANE provides beacon functionalities through the *OneHopNeighborDiscoveryService*. This service provides through its listener (*NeighborDiscoveryListener*) 3 basic methods:

- setNeighborData: called to receive the first (new) neighbor information (beacon) (from a previously not neighbor node).
- updateNeighborData: called to receive at each period the information from neighbor nodes.
- removeNeighborData: called to inform a loss (beacon timeout) of a neighbor.

Listing D.1: electLocalLeader(). NOTE: contains pseudo code parts for readability and size

```
/**
 * Elects a clusterhead between the one hop neighbors. The node with the
 * best weight is elected as clusterhead. Also check if the own device is
 * subhead (when subheads are activated).
 *
 * @return True if a new clusterhead was elected, false else.
 */
public boolean electLocalLeader() {
        // first of all check if have neighbor devices
          if (no neighbor devices){
                  // set the own device as clusterhead
                  setSelfClusterhead();
          }
        // initialize variables
        initalize_variavles();
        // elect the best of the neighbors!
        // ------------------------------------------------------
        // for each neighbor in stableNeighbors
        for (Iterator it = this.stableNeighbors.keySet().iterator(); it
                        .hasNext();) {
                // get the the Address and deviceData of the neighbor.
                Address neighborAddress = (Address) it.next();
                DeviceData neighborData = (DeviceData) this.stableNeighbors.get(
                    neighborAddress);
                // compare the weights
                if (neighborData != null) {
                        // if neighbor is its own clusterhead OR if using SUBHEADS
                        if (neighborData.getClusterheadAddress().equals(
                            neighborAddress) || SIMCONFIG.USE_SUBHEADS) {
                                // elect best Neighbor as new leader
                                electBestNewLeader();
                        }
                }
        }
        // ------------------------------------------------------
        // check if the own device is the best compared to best neighbor
        checkAgainstSelf();
        // check if some neighbor is using the own device as clusterhead.
        // also get the number of slaves in this iteration.
        // NOTE: if SIMCONFIG.USE_SUBHEADS = false, subheads do not happen but the
            code can remain unchanged here.
        checkIfSubhead_and_getNeighbors();
        // if the old clusterhead is still leading
        if (oldCHAddress != null)
                if (oldCHAddress.equals(this.deviceData.getClusterheadAddress()))
                        return false;
        // there is a new clusterhead
        return true;
```

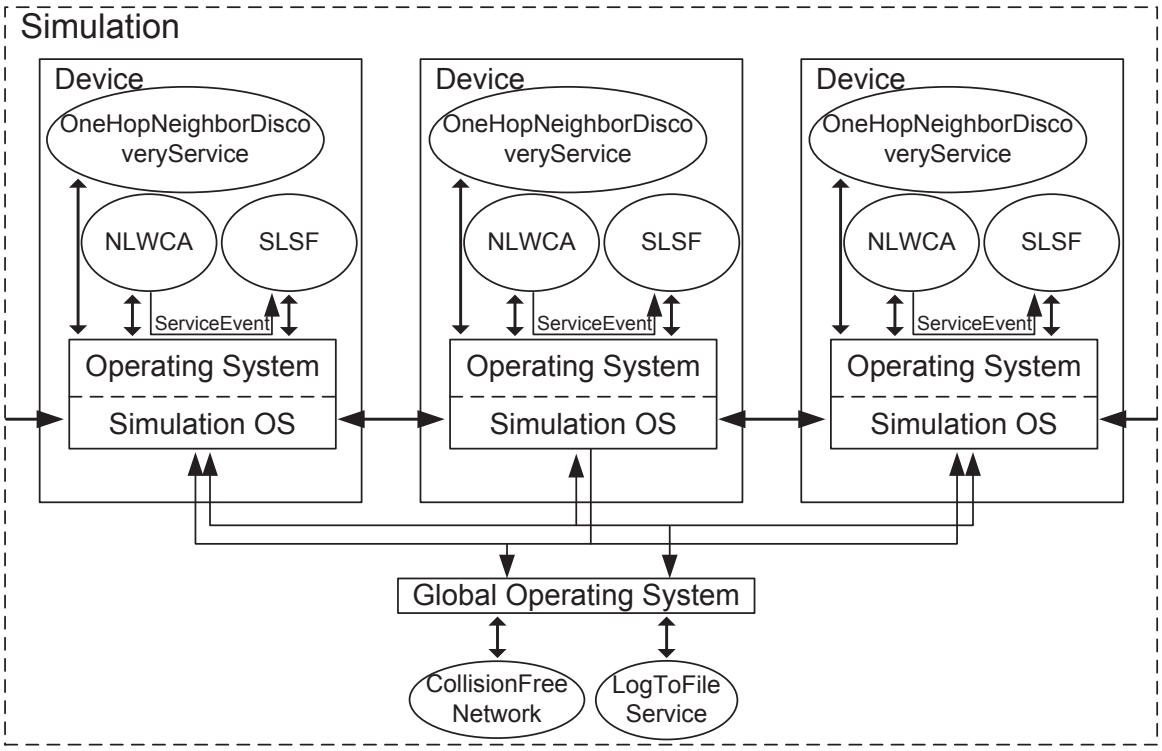


Figure D.3: SLSF implementation in JANE.

To be able to set specific information inside the beacon, the *OneHopNeighborDiscoveryService* also provides a *setOwnData* method that can contain any data structure that inherits from the JANE *Data* class. However, since NLWCA and SLSR are using the same *OneHopNeighborDiscoveryService*, they must use 2 different data structures to put their information in the beacon. NLWCA uses *DeviceData* as beacon structure and SLSR uses *DiscoveryACKData* as data structure. On one hand, NLWCA can use its *DeviceData* structure and remain unaware of any other beacon structure used by SLSF/SLSR. On the other hand, SLSR can use both, *DeviceData* (in a read-only fashion) and its own *DiscoveryACKData*. Note, that the data structures are differentiated using the *DATA_ID* field.

D.3.3 Fault-recovery mechanism

The fault-recovery mechanism presented in Section 3.5 needs two interactions with the beacons. The first, is to put sequence numbers according to the acknowledgement (reminder Figures D.4,D.5 and D.6). The second, is to obtain the beacon information from the neighbors containing acknowledgements (sequence numbers).

Set information in own beacon

To set the data inside the beacon we use, as described in the previous paragraph, the *setOwnData* method with the SLSF specific beacon data structure *DiscoveryACKData*. SLSF has different beacon formats whether the node is a clusterhead (Figure D.5) or a simple slave node (Figure D.6). Therefore, as shown in Table D.1, there are two different methods that translate the local stored data into the correct beacon structure depending on the node's status.

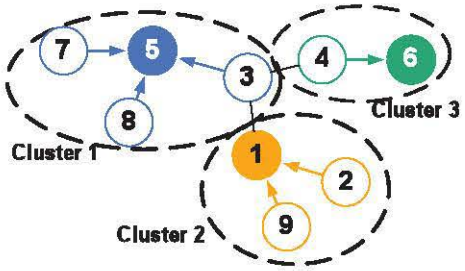


Figure D.4: Three clusters example.

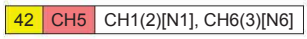


Figure D.5: CH 5 SLSF beacon with fault recovery.



Figure D.6: Node 3 SLSF beacon with fault recovery .

Node is CH	dataToClusterBeacon
Node is not CH	dataToNodeBeacon

Table D.1: Two different methods to build the beacon information.

Obtain information from neighbor beacons

The neighbor beacons arrive at the *OneHopNeighborDiscoveryService* methods: *setNeighborData* and *updateNeighborData*. However we only process beacons coming from the updateNeighborData method, since the beacons incoming at the setNeighborData come from yet "unstable" neighbors that are processed at the NLCWA level and not at the SLSR level. At the *updateNeighborData* the beacon information is first compared to the old stored information (if any) and then passed to the *handleNeighborhoodChange* method if the information is of interest/new. A specific method that processes the sequence numbers out of the beacon information is the *handleSequenceNumbers*.

The *handleSequenceNumbers* method dispatches the information for processing depending on 2 parameters: is the node processing a clusterhead or not and is the neighbor (source of the beacon) a clusterhead or not.

	Beacon source is CH	**Beacon source is not CH**
Node is CH	clusterheadBeaconToClusterhead	nodeBeaconToClusterhead
Node is not CH	clusterheadBeaconToNodeData	nodeBeaconToNodeData

Table D.2: handleSequenceNumbers dispatches to corresponding method depending on the beacon source and node status.

Message retransmission

The fault recovery mechanism retransmits message if no corresponding acknowledgement is received in time. Only slave nodes that have not been designated as ICR retransmit messages. On reception of a message to be forwarded (in the *handlePayloadMessage* method), the retransmission is only triggered if the node is a slave and not an ICR. The Listing D.2 shows how a message that is to be considered for retransmission is added to the retransmission queue. In line *13* to *22*, the timer of the retransmission is set according to the hop count the message has traveled since the last clusterhead. If the hop-count is 1 then the transmission delay is set to 4 seconds. If the hop-count is 2 then the transmission delay is set to 2 seconds. Once the timeout is set, the message is added to the retransmission queue at line *26* using a *RetransmissionOb-*

ject. The *RetransmissionObject* is filled with the time the message should be retransmitted if no acknowledgement is received, the last crossed clusterhead address and its message sequence number, the number of awating acknowledgement, the sender of the message and a complete copy of the message.

The number of awaiting acknowledgements is simply the number of nearby cluster that can receive and acknowledge the message. In other words, from a slave node's (considering a message for retransmission) perspective, the number of nearby clusterhead that are in reach of the message and itself (i.e. maximum 2 hops away). The method to calculate the number of pending acknowledgements for a given clusterhead source (i.e. the last crossed clusterhead) is shown in Listing D.3.

The retransmission queue is checked at every beacon reception using the *checkRetransmissionTimeout()* method in Listing D.4. Note, that each time a beacon is processed and an acknowledgement is received at the *clusterheadBeaconToNodeData* and *nodeBeaconToNodeData* (only slave node do retransmissions), the acknowledged message is removed from the retransmission queue.

D.3.4 ICR computation

Unlike described in Section 3.4.1, the ICR computation in the implementations is not yet event based. In the implementation, the ICR are recomputed each time a message is forwarded to a neighbor using the *icr_computeICR* method detailed in the Listing D.5. Future, implementation should be adapted to make this computation based on neighborhood changes rather than message forwarding.

D.3.5 Message creation and forwarding

In SLSR, to create a message with a payload, the *generatePayloadMessage(Address, Object, int)* method is used. Once this message is received by a neighbor node, and after it content is processed, the message is forwarded. Forwarding occurs differently whether the node is a clusterhead or a slave.

- **Clusterheads** have 2 different cases of forwarding:
  - Message source is one of its slaves (i.e. the message is unicasted from the slave to its CH). CH forwards this message as a broadcast and inserts its ICR selection. (method name: *forwardSlavePayloadMessage*)
  - Message source is a foreign node: insert/replace ICR selected in the message and forward it. (method name: *interClusterPayloadMessageRetransmission*).
- **Slave nodes** have also 3 different cases of forwarding (contained in the *handlePayloadMessage* method).:
  - Message designates node as ICR: forward the message and insert updated ICR selection.
  - Message does not designate node as ICR: do not forward it, but consider it for retransmission.
  - Message is *delayed_retransmitted*: forward it immediately. ICR selection is not considered for retransmitted messages.

Listing D.2: retransmission code part in handlePayloadMessage(). NOTE: contains pseudo code parts for readability and size

```
1
2  public void handlePayloadMessage(LinkLayerInfo info,
3                  PayloadMessage payloadMessageOriginal) {
4          // ...
5          /**
6           * IF
7           * > message is new (not in duplicate list) and passed various validity
                checks
8           * > node is not CH and not ICR
9           * > node has other nearby clusterheads
10          * > message is not already a delayed retransmitted message
11          * THEN put the message in the retranmission queue
12          */
13         // ...
14         // init field
15         double retransmission_time = 0;
16         // if the message is coming directly from a clusterhead
17         if (payloadMessage.getHopCountFromLastCH() == 1) {
18                 // set the timeout to: current time + RETRANSMISSION_DELAY  = 4
                       seconds from now
19                 retransmission_time = runtimeOperatingSystem.getTime()+
                       RETRANSMISSION_DELAY;
20         // if the message already traveled 2 hops (coming from a slave node)
21         } else if (payloadMessage.getHopCountFromLastCH() == 2) {
22                 // set the timeout to: current time + RETRANSMISSION_DELAY/2  = 2
                       seconds from now
23                 retransmission_time = runtimeOperatingSystem.getTime()+
                       RETRANSMISSION_DELAY / 2;
24         }
25         // if the retransmission time was set
26         if (retransmission_time != 0) {
27                 // put the message in the transmission queue
28                 retransmissionQueue.put(new RetransmissionObject(
                       retransmission_time, payloadMessage.
                       getLastNearbyClusterheadAddress(), payloadMessage.
                       getLastCHMessageSeqNbr(),
                       calculateNumberOfPendingAcknowledgment(payloadMessage.
                       getLastNearbyClusterheadAddress()), payloadMessage.getSender()
                       ), (PayloadMessage) payloadMessage.copy());
29         }
30 }
```

Listing D.3: calculate the number of pending acknowledgements.

```
1
2  // Calculate the number of pending acknowledgement for a given
       sourceClusterAddress
3  public int calculateNumberOfPendingAcknowledgment(Address sourceClusterAddress) {
4          int numberOfPendingAcks = 0;
5          // for each nearby clusterhead
6          for (Object currentCluster : this.notClusterNodeAcknowledgementsInfo
7                          .keySet()) {
8                  // retrieve the sequence numbers that nearby clusterhead announced
9                  Vector<SequenceNumberTuple> sequenceNumberTuples = (Vector)
                       notClusterNodeAcknowledgementsInfo.get((Address)
                       currentCluster);
10                 // for each of the announced sequence number
11                 for (SequenceNumberTuple sequenceNumberTuple :
                       sequenceNumberTuples) {
12                         // if the sourceClusterAddress is mentioned. In other
                               words, if the nearby clusterhead has acknowledged
                               messages for the sourceClusterAddress.
13                         if (sequenceNumberTuple.getAcknowledgedClusteraddress() ==
                               sourceClusterAddress) {
14                                 // if there exists an discovery data about
                                       sourceClusterAddress
15                                 if (discoveryACKData.getForeignClusterheads().
                                       containsKey(
16                                                         sourceClusterAddress)) {
17                                         // ignore hopcounts greater than 2 (out of
                                               reach)
18                                         if (((DiscoveryTuple) discoveryACKData
19                                                         .getForeignClusterheads().
                                                               get(
                                                               sourceClusterAddress))
                                                               .getHopCount() > 2) {
20                                                 // increase count
21                                                 numberOfPendingAcks++;
22                                         }
23                                 } else {
24                                         // increase count
25                                         numberOfPendingAcks++;
26                                 }
27                         }
28                 }
29         }
30         return numberOfPendingAcks;
31 }
```

Listing D.4: checkRetransmissionTimeout method.

```
1
2 private void checkRetransmissionTimeout() {
3         // for each message in the retransmission queue
4         for (Object retransmissionObject0 : retransmissionQueue.keySet().toArray()
              ) {
5                 RetransmissionObject retransmissionObject = (RetransmissionObject)
                      retransmissionObject0;
6                 // check if the retransmission timed out
7                 if (retransmissionObject.getTimeout() < runtimeOperatingSystem.
                      getTime()) {
8                     //if so, prepare the message
9                     PayloadMessage payloadMessage = retransmissionQueue.get(
                          retransmissionObject);
10                    payloadMessage.setIcrSet(new Vector<Address>());
11                    payloadMessage.setDelayed_retransmitted(true);
12                    payloadMessage.setTimeToLive(payloadMessage.getTimeToLive
                          () - 1);
13                    payloadMessage.incrementHopCount();
14                    // send (retransmit) the message
15                    sendBroadcast(payloadMessage);
16                    // remove the retranmitted message from the queue
17                    retransmissionQueue.remove(retransmissionObject);
18                }
19        }
20 }
```

D.3.6 Routing - SLSR

SLSR adds routing capabilities to the SLSF protocol. In the implementation, routing can be divided in 4 tasks:

- Add new headers/fields messages as described in Section 4.5 (here fields are added in the *payloadMessage* class).
- Send out new routing information/messages based on events triggered on neighborhood changes (task done by *sendImmediateRouteUpdate* and *sendImmediateLostCHUpdate* methods).
- Build the routing table by extracting the routing information from incoming messages (done at the *buldRoutingTable* method).
- Route messages towards the destination using the information contained in the routing table (method *getNextClusterheadHop* returns the next clusterhead hop for a given destination).

D.4 Zeroconf - adaptations

As described in Chapter 7, we use Zeroconf as service discovery protocol. Moreover, we also propose to replace the multicast part of mDNS with our SLSF/SLSR protocol. As reference implementation for Zeroconf we use the available *jmDNS* implementation [jmDNS].

Listing D.5: icr_computeICR method.

```
1  public void icr_computeICR(Address sourceCH) {
2          icrSet.clear(); // init
3          HashMap<Address, HashMap> discoveredClusterList = (HashMap<Address,
               HashMap>) this.discoveredClusterheads.clone();
4           // Remove the source clusterhead from the icr selection
5          if (sourceCH != null) discoveredClusterList.remove(sourceCH);
6           // Remove one hop Clusters
7          icr_removeDirectConnectedClusters(discoveredClusterList);
8          // keep only shortest path
9          HashMap<Address, HashMap> oneHopDiscoveredClusterList =
               icr_keepBestHopCounts(discoveredClusterList);
10         // First elect nodes that are the only reaching a particular cluster
11     icr_selectNodesWhichAreTheOnlyReachingACluster(oneHopDiscoveredClusterList);
12         // remove covered clusters from the list
13         icr_removeCoveredClusters(oneHopDiscoveredClusterList);
14         // init reachabilityList
15         HashMap<Address, Integer> reachabilityList = null;
16         // while there a clusters to be covered
17         while(!oneHopDiscoveredClusterList.isEmpty()) {
18                 // compute reachability
19                 reachabilityList = icr_calculateReachabilityOfOneHopNodes(
                       oneHopDiscoveredClusterList);
20                 Address selected_ICR = null;// init parameters
21                 int bestReachability = -1;  // .
22                 double bestWeight = -1;     // .
23                 // for each cluster in the list
24                 for(Address address : reachabilityList.keySet()){
25                         // select as ICR the node that reaches the most clusters
26                         if(reachabilityList.get(address)>bestReachability){
27                                 bestReachability = reachabilityList.get(address);
28                                 bestWeight = ((DeviceData) stableNeighbors.get(
                                       address)).getWeight();
29                                 selected_ICR = address;
30                         } else if(reachabilityList.get(address)==bestReachability)
                               {
31                                 DeviceData deviceData = (DeviceData)
                                       stableNeighbors.get(address);
32                                 // select as ICR the node that has the best weight
33                                 if(deviceData.getWeight()>bestWeight){
34                                         bestWeight = deviceData.getWeight();
35                                         selected_ICR = address;
36                                 } else if(deviceData.getWeight()==bestWeight){
37                                         // select as ICR the node that has the
                                               biggest IP address
38                                         if(address.compareTo(selected_ICR)>0)
                                               selected_ICR = address;
39                                 }
40                         }
41                 }
42                 // add the newly select ICR to the icrSet
43                 icrSet.add(selected_ICR);
44                 // remove covered clusters from the list
45                 icr_removeCoveredClusters(oneHopDiscoveredClusterList);
46         }
47 }
```

In order to transform the jmDNS implementation into a JANE service, the necessary JANE methods where added to the *JmDNSImpl* class. The *JmDNSImpl* class, obtains the basic JANE methods as *start*, *finish*, *sendBroadcast* and *sendUnicast*[43].

In section 2.5.1 we presented the advantages of using SLSF/SLSR as a replacement for the multicast layer in Zeroconf. In our implementation, to transport and forward Zeroconf messages using SLSF/SLSR we replaced all output and input points (Java sockets) from the jmDNS implementation with new outputs and inputs from JANE. Those new out/input points are not JANE standard communication ways (e.g. *linkLayer.sendBroadcast(LinkLayerMessage)* and *LinkLayerMessage handle(LinkLayerInfo)* methods), but use *serviceEvents* to communicate its with the underlying SLSF/SLSR protocol. Listing D.6 and D.7 show the jmDNS original outputs and the JANE implementation outputs where the socket is replaced bby a serviceEvent dispatch. Similarly, input messages received in the original implementation at the *SocketListener* by the socket in Listing D.8 are now received aat the JANE *EventListener* by the handle method from service events (send by SLSF/SLSR).

Listing D.6: Zeroconf message output in the jmDNS implementation (pseudo code).

```
1  public class JmDNSImpl extends JmDNS {
2      private MulticastSocket socket;
3      /**
4       * Send an outgoing multicast DNS message.
5       */
6      public void send(DNSOutgoing out) throws IOException {
7          if (!out.isEmpty()) {
8              DatagramPacket packet = new DatagramPacket(out.data, out.off, group,
                   DNSConstants.MDNS_PORT);
9              // send packet using multicast socket
10             socket.send(packet);
11         }
12     }
13 }
```

Listing D.7: Zeroconf message output in the JANE/SLSR adapted implementation (pseudo code).

```
1  public class JmDNSService extends JmDNS implements RuntimeService, EventListener,
       MenuSignal {
2          /**
3           * Send an outgoing multicast DNS message.
4           */
5          public void send(DNSOutgoing out) throws IOException {
6                  if (!out.isEmpty()) {
7                          // build packet
8                          DatagramPacket packet = new DatagramPacket(out.data, out.
                               off,group, DNSConstants.MDNS_PORT);
9                          // send packet using JANE service events (here a listener
                               is the SLSRService class that will send the message)
10                         this.runtimeOperatingSystem.sendEvent(new JmDNSEvent(
                               packet, 0));
11                 }
12         }
13 }
```

[43] sendBroadcast and sendUnicast are not used here since the outgoing message are send via SLSF/SLSR

Listing D.8: Zeroconf message input in the jmDNS implementation (pseudo code).

```
class SocketListener implements Runnable {
        public void run() {
                // retrieve packet from the socket
                this.jmDNSImpl.getSocket().receive(packet);
                // extract mmessage from datagramm packet
                DNSIncoming msg = new DNSIncoming(packet);

                if (msg.isQuery()) {
                        this.jmDNSImpl.handleQuery(msg,packet,...);
                } else {
                        this.jmDNSImpl.handleResponse(msg, packet,...);
                }
        }
}
```

Listing D.9: Zeroconf message input in the JANE/SLSR adapted implementation (pseudo code).

```
public class JmDNSService extends JmDNS implements RuntimeService,EventListener,
    MenuSignal {
        // handle method is called by JANE upon event reception from another
            service
        public void handle(de.uni_trier.jane.service.event.ServiceEvent e) {
                // filter SLSR events
                if ("class slsr.SLSREvent".equals(e.getClass().toString())) {
                        // check if payload is a Zeroconf DatagramPacket
                        if (slsrEvent.getPayload() instanceof DatagramPacket) {
                                // handle data gramm packets
                                handleIncomingDatagramPackets((DatagramPacket)
                                    slsrEvent.getPayload(),...);
                        }
                }
        }

        public void handleIncomingDatagramPackets(DatagramPacket packet, ...) {
                // extract message from datagramm packet
                DNSIncoming msg = new DNSIncoming(packet);

                if (msg.isQuery()) {
                        this.handleQuery(msg,packet,...);
                } else {
                        this.handleResponse(msg, packet,...);
                }
        }
```

Résumé Étendu: Contributions pour une découverte de services avancée dans les réseaux ad hoc

Appendix E

Résumé Étendu: Contributions pour une découverte de services avancée dans les réseaux ad hoc

Lors de la dernière décennie, le nombre d'appareils possédant des capacités sans fil a très fortement augmenté. Ces appareils sont aussi devenus plus puissants et abordables, attirant ainsi le grand public vers les réseaux mobiles sans fil.

Les réseaux mobiles regroupent plusieurs types de réseaux que l'on peut classer en trois catégories: les réseaux sans fil avec infrastructure tel que les réseaux mesh ou cellulaire, les réseaux sans fil avec des appareils spécifiques tel que les réseaux de capteurs et les réseaux sans fil sans aucune infrastructure, et des appareils hétérogènes, tel que les réseaux ad hoc. Dans cette thèse nous considérons le cas des réseaux mobiles ad hoc aussi connus sous le nom de MANET (Mobile Ad hoc NETworks). Les MANETs sont des réseaux constitués d'appareils, que l'on appelle aussi nœuds, qui communiquent entre eux en utilisant les ondes radio. Les réseaux ad hoc se forment spontanément dès lors que des nœuds se trouvent dans leurs rayons de transmissions respectifs, formant ainsi un réseau sans infrastructure. Les caractéristiques principales des MANETs sont la grande dynamicité des nœuds (induite par le mouvement des utilisateurs), la propriété volatile des transmissions sans fil, le comportement des utilisateurs, les services et leurs utilisations. La spontanéité et la dynamique des réseaux ad hoc fait de l'usage de la découverte de services un atout majeur et représente, en même temps, un défi important. La découverte de services permet de trouver des services fournis par d'autres nœuds du réseau de manière automatisée sans la nécessité d'un point central de contrôle.

Cette thèse propose une solution complète pour la découverte de services dans les réseaux ad hoc, de la couche réseau sous-jacente à la découverte de services à proprement dite. Le processus de découverte de services implique tous les niveaux sous-jacents et leurs domaines de recherche respectifs. Ainsi, bien que notre but soit d'améliorer la découverte de services, les domaines liés sont aussi impliqués et pris en compte pour nos contributions.

Un premier objectif de cette thèse, est de construire une structure stable au-dessus du réseau ad hoc spontanément formé. Un défi majeur est d'atteindre une stabilité satisfaisante tout en gardant un niveau bas du coût, en terme de bande passante, pour la mise en place cette structure stable. Notre première contribution est ainsi le protocole Stable Linked Structure Flooding (SLSF) qui est basé sur une structure de cluster à un saut (NLWCA) et permet d'obtenir une diffusion efficace des messages et qui passe à l'échelle. La seconde contribution est le protocole de routage Stable Linked Structure Routing (SLSR) qui utilise SLSF pour la dissémination

des messages de routage. En utilisant ces protocoles de dissémination et de routage comme base, nous proposons d'améliorer la découverte de services en prenant en compte le contexte et en adaptant cette découverte à celui-ci. De plus, nous avons également contribué aux outils de simulation. Nous proposons de coupler et de combiner les simulateurs et les modèles, qui, ensemble, permettent de simuler une grande variété de scénarios et notamment des scénarios capables de reproduire le comportement humain.

Une ligne directrice pour toutes nos contributions a été de pouvoir intégrer la prise en compte du contexte dans les protocoles ainsi que dans les outils de recherches proposés.

Appendix F

Introduction aux réseaux mobiles ad hoc et à la découverte de service

Contents

F.1 Dissémination, routage et contrôle topologique 177
F.1.1 Protocoles de routage ad hoc 178
F.1.2 Stratégies de dissémination 179
F.1.3 Sélection des nœuds relais 180
F.1.4 Hiérarchie et clustering 181
F.2 Service de nommage 183
F.3 La découverte de service 183
F.3.1 Classification des protocoles de découverte de service 184
F.3.2 Découverte de service dans les réseaux filaires 185
F.3.3 Découverte de services dans les réseaux ad hoc 186
F.4 Métriques et contexte 188

Les réseaux mobiles ad hoc sont aussi connus sous le nom de MANET (Mobile Ad hoc NETworks). Ce sont des réseaux constitués d'appareils, également appelés nœuds, qui communiquent entre eux en utilisant les ondes radio. Les réseaux ad hoc se forment spontanément dès lors que ces nœuds se trouvent dans leurs rayons de transmissions respectifs, formant ainsi un réseau sans aucune infrastructure préexistante. Ce type de réseau peut être utilisé dans des scénarios tel que la communication inter-véhicules, les observations environnementales, l'accès à Internet ubiquitaire ou encore lors d'opérations de secours [Sant 05]. Les MANETs font partie d'une famille de réseaux appelée les réseaux dynamiques. Les réseaux dynamiques peuvent être classés en quatre catégories représentées dans la figure F.1.

Parmi ces quatre catégories, les réseaux ad hoc sont ceux dont la dynamicité est la plus grande et surtout provenant de plusieurs domaines variées. En effet, la dynamicité peut venir de la simple mobilité des nœuds du réseau, de la ressource énergétique limitée causant des déconnexions, de la nature volatile des transmissions sans fil ou encore du comportement humain de l'utilisateur. Obtenir une communication durable et stable dans de telles conditions représente un des défis majeurs des réseaux ad hoc.

F.1 Dissémination, routage et contrôle topologique

Si l'on observe seulement les échanges de messages, la découverte de services peut se résumer en une suite d'envois de messages d'annonce ou de requêtes que l'on diffuse dans le réseau.

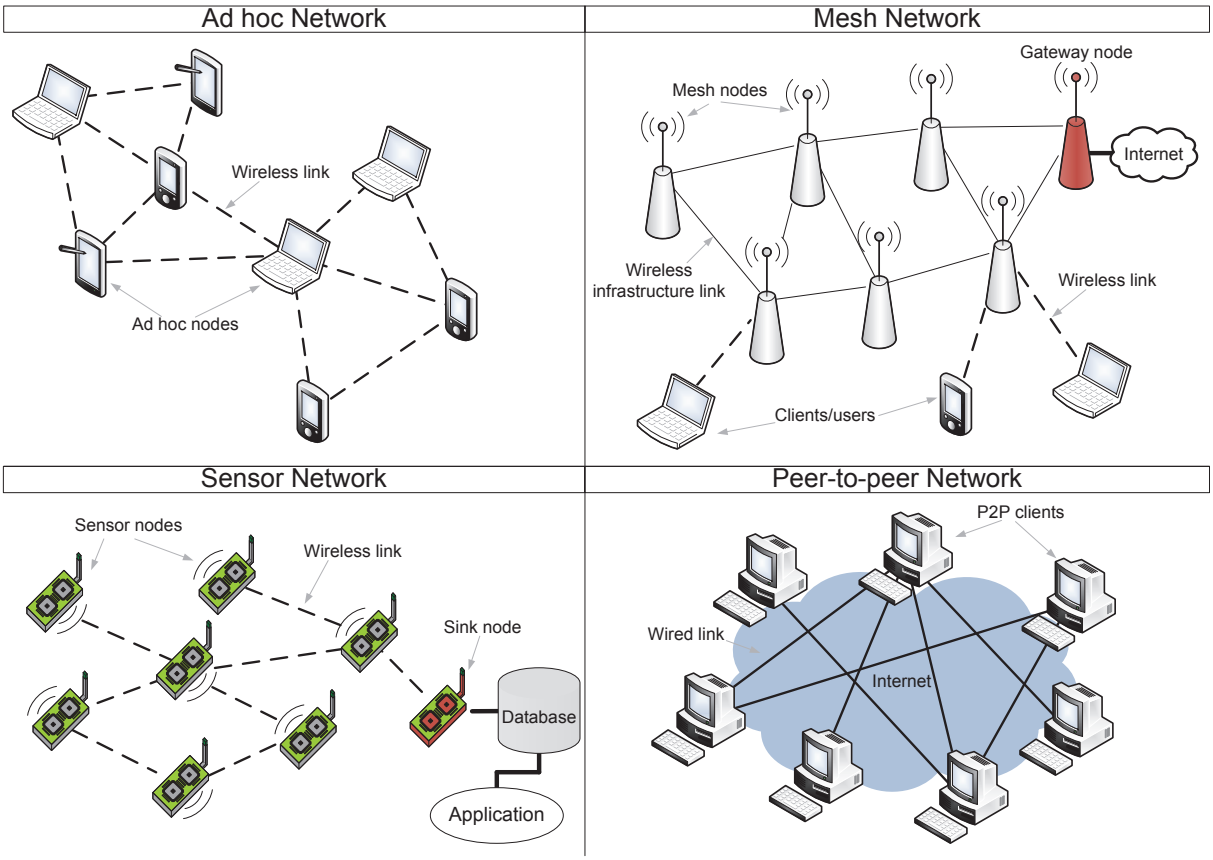


Figure F.1: Réseaux dynamiques: Ad hoc, mesh, capteur et pair-à-pair.

Ainsi, une première étape vers l'amélioration de la découverte de services est l'amélioration de la dissémination sous-jacente. On entend ici par dissémination, la distribution d'un message à tous les nœuds (ou à un groupe en particulier) joignables du réseau dans un temps imparti. Pour faire ceci les nœuds doivent relayer les messages pour d'autres nœuds distants. Il y a plusieurs moyens pour y parvenir. Le plus basique est le flooding ou l'inondation, où chaque nœud relaie le message pour ses voisins. Cependant, ce comportement entraîne un problème bien connu sous le nom de *broadcast storm*, lorsque les nœuds relaient sans cesse les messages [Ni 99]. Afin d'éviter ce problème, les protocoles de dissémination et de routage, structure ou désigne de façon propice les nœuds qui vont être chargés de relayer les messages.

F.1.1 Protocoles de routage ad hoc

Afin de rendre possible la communication avec des nœuds distants dans le réseau ad hoc, les protocoles de routage sont chargés de déterminer le chemin à suivre pour que le message arrive à destination. Pour faire cela, le plus souvent, ils établissent une table de routage, qui peut être plus ou moins complète. Pour obtenir cette table de routage, il existe plusieurs stratégies, le but étant de minimiser le coût en termes de bande passante et de nombre de messages échangés. Cela dit, dans ce manuscrit, nous nous intéressons plus aux performances de dissémination atteint par ces protocoles de routage en utilisant les informations recueillies qu'à la qualité (en terme de chemins choisis, exhaustivité, etc.) de la table de routage.

Optimized Link State Routing (OLSR)

Le protocole de routage OLSR [Clau 03] conçu pour les réseaux ad hoc a été standardisé au sein de l'IETF[44] par le groupe de travail MANET [Mobi]. C'est un protocole de routage proactif qui échange périodiquement des messages HELLO. Ces messages permettent à chaque nœud de découvrir son voisinage direct, mais aussi son voisinage à deux sauts. Il obtient ainsi une vue à deux sauts du réseau comme le montre la figure F.2.

Le principal atout d'OLSR est sa sélection de nœuds MPR (MultiPoint Relay), qui est une forme de bordercast à deux sauts. En effet, en utilisant l'information à deux saut de son voisinage, OLSR sélectionne les nœuds de telle sorte à avoir le plus petit nombre de nœuds à un saut qui relaient les messages tout en atteignant tous les nœuds à deux sauts (figure F.2c). Ces nœuds intermédiaires sont alors des nœuds MPR. Chaque nœud exécute alors le même processus localement et désigne à son tour les MPR dont il a besoin pour atteindre tous les nœuds à deux sauts (figure F.2d). Au final on obtient une structure de nœuds où en utilisant uniquement les MPR pour relayer un message, on atteint tous les nœuds du réseau (figure F.2e). OLSR utilise alors cette structure de dissémination pour diffuser des messages Topology Control (TC) qui contiennent la liste des voisins à 1 saut de l'émetteur d'un tel message. Seuls les MPR émettent des messages TC et en même temps seuls les MPR relayent les messages.

Une fois les messages TC de chaque MPR du réseau reçus, chaque nœud possède une table de routage contenant la topologie complète du réseau. Nous nous sommes tout particulièrement intéressés à OLSR, à cause de la structure de dissémination qu'il établit avec la sélection des nœuds MPR.

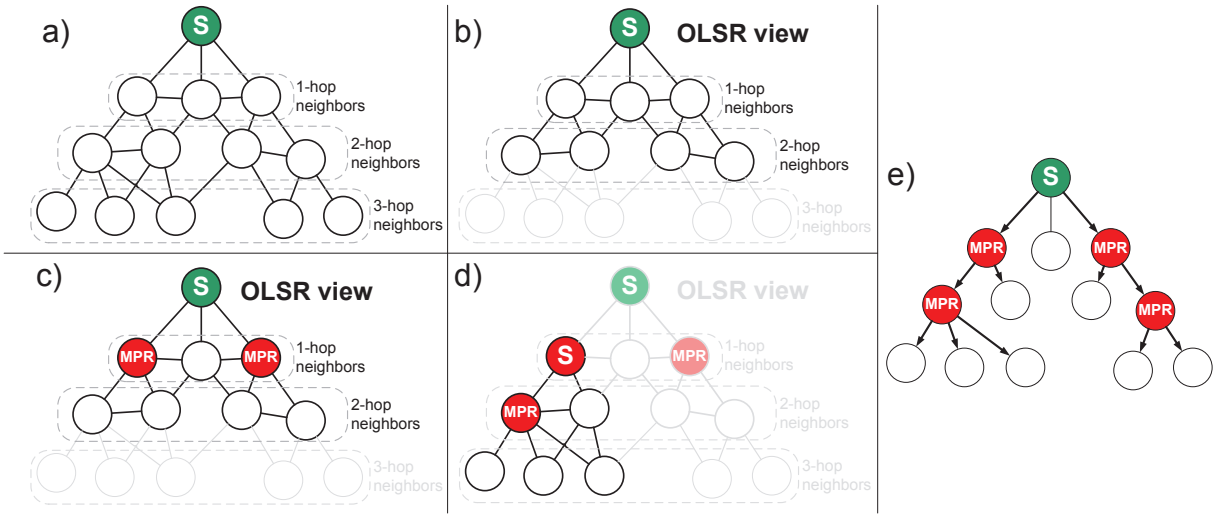


Figure F.2: Séléction des MPR (Bordercasting) avec OLSR.

F.1.2 Stratégies de dissémination

Il existe aussi des protocoles spécifiquement dédiés à la dissémination. Contrairement aux protocoles de routage, ils n'établissent pas de table de routage ou de chemin vers un nœud en particulier. Leur but est simplement de proposer des stratégies pour réduire le coût de la dissémination d'un message (envoi du message à tous les nœuds) dans le réseau.

[44] Internet Engineering Task Force

D'un certain point de vue, les protocoles de routage et de dissémination partagent un but commun : atteindre tous les nœuds du réseau en ne gardant que les liens pertinents. Un lien pertinent peut être:

- seulement les liens où une communication est en cours, comme l'on peut observer dans les protocoles réactifs (figure F.3b).
- un sous ensemble de liens minimaux qui permet d'atteindre tout les nœuds du réseau. Tel est le cas des protocoles de dissémination et les protocoles de routage proactifs (figure F.3c).

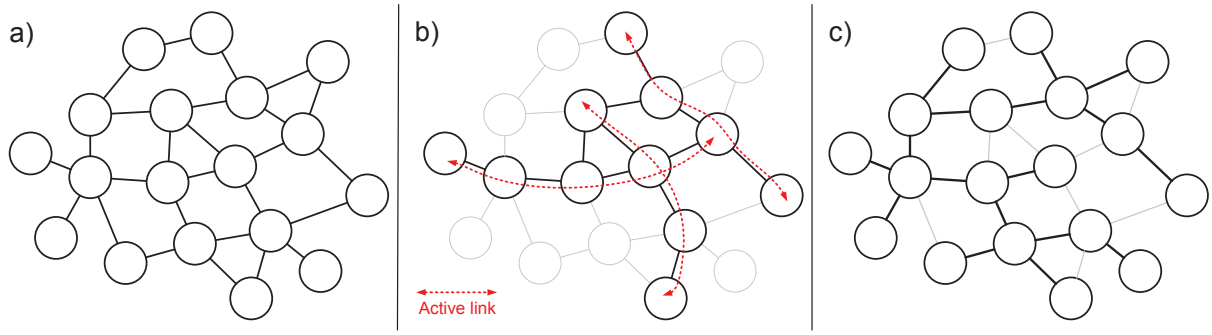


Figure F.3: a) réseau ad hoc. b) lien actif uniquement. c) sous-ensemble de nœuds pour la dissémination.

Le problème qui consiste à trouver le plus petit ensemble de liens ou nœuds qui permet de couvrir tous les nœuds du réseau est un problème bien connu de la théorie des graphes sous le nom d'ensembles dominants (*dominating sets*). En utilisant la théorie des graphes, on peut représenter un réseau ad hoc par un graphe $G = (V, E)$ où V (Vertices) représente les nœuds et E (Edges) les liens les reliant. En utilisant cette représentation, un ensemble dominant $D \subseteq V$ est un sous ensemble de V qui domine tous les nœuds du graphe G. En d'autres termes, chaque nœud qui n'est pas dans l'ensemble dominant D a au moins un nœud dans son voisinage direct qui fait partie de l'ensemble D (figure F.4). Une étude complète sur les ensembles dominants et notamment ceux qui sont construits de manière distribuée se trouve dans [Blum 04] et [Wu 02].

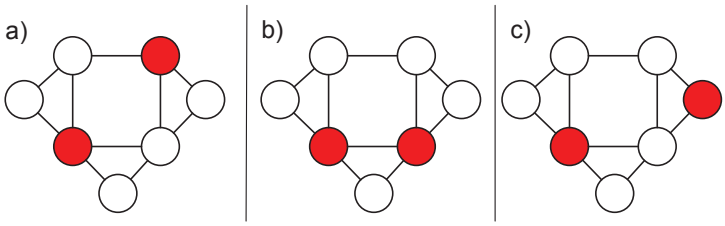


Figure F.4: Ensemble dominant (Dominating set) (nœud en rouge).

F.1.3 Sélection des nœuds relais

Afin de réduire le nombre de transmission de messages, beaucoup de protocoles ad hoc utilisent la sélection de nœuds relais. Dans [Peng 01], des " broadcast relay gateways " (BRG) sont sélectionnés de tel manière à couvrir chaque nœud à 2 sauts par au moins un BRG. Une autre

approche, DCB (Double-covered Broadcast) [Lou 04], sélectionne des nœuds relais de telle sorte à couvrir chaque nœud à 2 saut par deux nœuds relais. Les nœuds à 2 sauts sont ainsi doublement couverts. Comme nous l'avons précédemment décrit, le mécanisme des MPR de OLSR forme une structure de dissémination basée sur la sélection des nœuds relais pour couvrir avec le nombre minimal de nœud à 1 saut tous les nœuds à 2 sauts [Jacq 01].

F.1.4 Hiérarchie et clustering

Une autre stratégie pour réduire le coût de dissémination est l'organisation du réseau en hiérarchie ou " clusters ".

Le *clustering* est une technique qui organise le réseau en groupes de nœuds appelés des clusters. Chaque cluster a un " meneur ", appelé *clusterhead*, qui est en charge de la gestion du cluster. Les autres participants du cluster sont appelés des *slaves*. L'étendu d'un cluster peut dépendre par exemple du nombre de sauts depuis le clusterhead ou encore de la distance géométrique. Le clusterhead peut être élu par vote coordonné, mais aussi par simple désignation individuelle de chaque nœud. Ainsi, selon l'algorithme, les clusters peuvent se recouvrir partiellement et avoir plusieurs clusterhead dans un même rayon. Un clusterhead peut, lui aussi, élire un autre nœud que lui même en tant que son clusterhead et devenir un " sous-clusterhead " de ce dernier appelé *subhead*.

Par exemple les auteurs de [Ni 99] proposent un algorithme de clustering qui élit les clusterhead après un échange de messages dans le voisinage. Dans [Foro 05], les clusters sont sélectionnés individuellement sur la base des informations contenues dans les messages périodiques échangés, les beacons. Il n'y a donc pas d'élection mais simplement une annonce individuelle de chaque nœud du clusterhead choisi. De plus, une notion de stabilité des nœuds y est proposée pour sélectionner en tant que clusterhead le nœud le plus stable. Cette approche, est très similaire au protocole NLWCA que nous avons choisi. L'avantage de NLWCA par rapport à cette approche est qu'il dispose de cette même notion de stabilité des nœuds mais en plus d'une notion de stabilité des liens entre les nœuds.

NLWCA – Node and Link Weighted Clustering Algorithm

NLWCA [Andr 08b] organise le réseau en clusters à 1 saut en n'utilisant que des informations disponibles localement. Chaque nœud choisit un nœud en tant que son clusterhead, formant ainsi des clusters comme ceux de la figure F.5. Le choix de clusterhead est basé sur les poids annoncés par chaque nœud du voisinage. Le nœud avec le poids le plus grand est alors choisi comme clusterhead. Le but principal de NLWCA est d'éviter les réorganisations superflues des clusters lorsque ceux-ci se croisent. NLWCA possède une notion de stabilité des liens, représentée par un poids pour chaque lien, qui permet d'éviter de prendre en compte un nœud (voire même un cluster entier) dans son organisation. L'exemple sur la figure F.6 montre les liens inter-cluster dont la stabilité est à 0 et évite ainsi une réorganisation des deux clusters qui ne se croisent qu'un court instant.

Les informations de poids d'un nœud et du choix de clusterhead sont transmises dans les beacons périodiquement émis par chacun des nœuds. Les beacons de NLWCA, sur la figure F.7, ont deux utilités:

- Permettre la sélection des clusterheads en se basant sur les informations qu'ils contiennent (poids du nœud annoncé et adresse de clusterhead).
- Calculer la stabilité d'un lien en utilisant la périodicité des annonces. C'est ainsi, que les nœuds dont le nombre de beacons reçus dans une période donnée est inférieur à un seuil de stabilité (*stability threshold*) sont considérés comme instables.

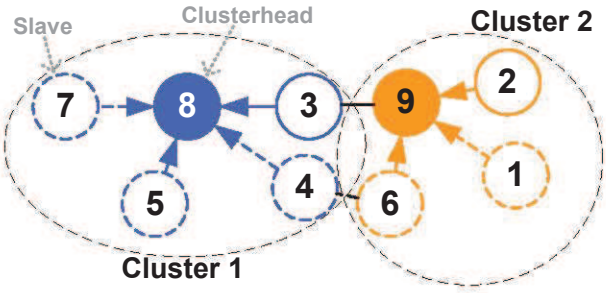


Figure F.5: Exemple de deux clusters dans NLWCA.

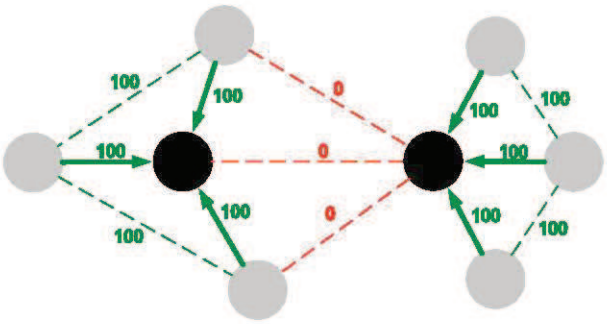


Figure F.6: Deux clusters NLCWA qui se croisent. Un poids faible sur un lien indique que celui-ci n'est pas stable et n'est ainsi pas pris en compte pour la sélection du clusterhead.

WCPD – Weighted Cluster-based Path Discovery protocol

Le Weighted Cluster-based Path Discovery protocol (WCPD) [Andr 08a] se base sur les clusters formés par NLWCA et permet la communication entre ces clusters. WCPD ajoute des champs au beacon (figure F.8) qui vont permettre de découvrir les clusterheads voisins. Chaque nœud possède alors les informations suivantes à propos de chacun de ses voisins à un saut: son poids, son clusterhead, les clusterheads voisins qu'il a découverts et la distance pour atteindre ceux-ci.

Pour diffuser un message dans le réseau avec WCPD, le nœud source envoi tout d'abord le message à son clusterhead (si ce n'est pas lui même). Celui-ci va alors envoyer une première fois le message en broadcast à son voisinage à 1 saut, les membres du cluster. Ensuite il l'envoi en unicast multi-saut vers tout les clusters voisins (figure F.9). Chaque clusterhead qui reçoit ce message exécute la même procédure ce qui, au final, diffusera le message dans tout le réseau.

Un **service** est une entité qui est la représentation, sur le réseau, d'un appareil ou d'un programme qui est capable d'exécuter une ou plusieurs tâches pour un autre nœud ou utilisateur. Un service peut être la combinaison d'une suite d'appareils ou de programmes variés qui, ensemble, accomplissent une tâche. Ces services possèdent typiquement une description détaillée des propriétés et des options disponibles destinée aux utilisateurs ainsi que les protocoles compatibles, l'adresse IP et le port utilisé.



Figure F.7: Format du beacon de NLWCA.

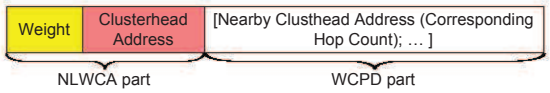


Figure F.8: Format du beacon WCPD combiné avec NLWCA.

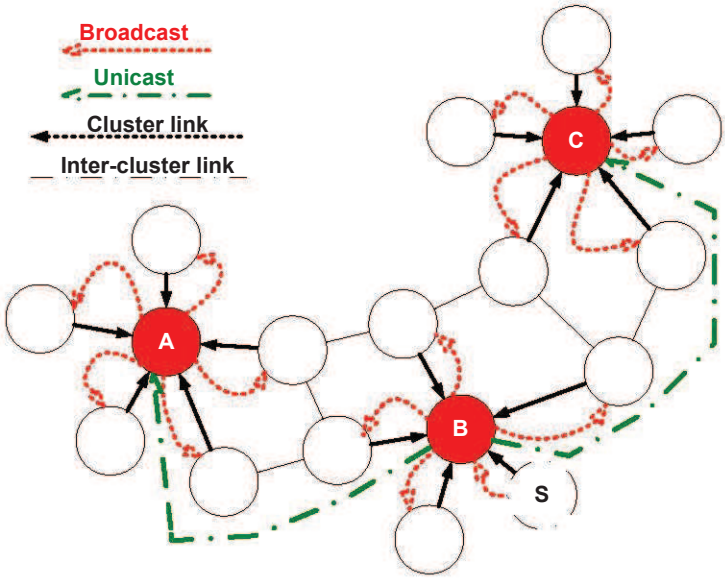


Figure F.9: Dissémination d'un message avec WCPD.

F.2 Service de nommage

D'un point de vue réseau, les différents protocoles, tel que le routage ou autre, utilisent le plus souvent les adresses IP pour la communication. Du point de vue d'un utilisateur cependant, utiliser les adresses IP pour identifier un appareil, un service ou une autre entité d'un réseau n'est pas du tout naturel et pratique. Pour ces raisons, on attribut un nom à chaque appareil, appelé *hostname*, qui donne une identité plus compréhensible et simple qu'une série de chiffres. Dans un réseau domestique, l'attribution des noms peut être configurée manuellement et la gestion de l'unicité de ceux-ci est facile. Cependant dans des réseaux plus grand, tel qu'un réseau d'entreprise ou bien plus grand encore le réseau Internet, ceci n'est plus gérable manuellement. C'est alors qu'intervient le DNS (Domain Name System), [Mock 87] qui est chargé d'enregistrer et de garantir l'unicité des noms dans un réseau ou domaine dont il a la charge. Il sera aussi en charge de convertir les hostnames en adresse IP lorsqu'il reçoit une requête dite de résolution de nom. Un utilisateur n'a alors qu'à retenir ces noms humainement compréhensibles pour accéder à une machine distante. La résolution du hostname en adresse IP se fait de manière transparente pour l'utilisateur.

F.3 La découverte de service

Même en ayant des noms de machines humainement compréhensibles, il manque cependant toujours un élément important. En effet, si le nombre de machine devient trop important, il devient difficile de retenir tous les noms. De plus, si un utilisateur entre dans un réseau inconnu, il ne connaît pas les noms des différentes machines. C'est là où la découverte de service intervient

et permet à un utilisateur de découvrir les noms et machines qui hébergent un service sur le réseau.

Le but de la découverte de services est de trouver les services, fournis par d'autres nœuds, présents dans le réseau de manière automatisée, en ne requérant de la part de l'utilisateur de ne fournir quel type de service est désiré. La découverte de services facilite ainsi la localisation (physique et sur le réseau), l'usage et la configuration des services.

Un utilisateur qui entre sur un réseau inconnu peut alors, grâce à la découverte de services, obtenir la liste des services disponibles. Une fois un service choisi, la configuration est automatique aussi bien au niveau des protocoles de communication qu'au niveau des applications nécessaires au bon fonctionnement.

Durant ces dernières années, une large variété de services est apparu, tel que le partage multimédia, les services de jeux ou encore l'accès à Internet au travers d'une passerelle distante. Dans les réseaux ad hoc, sans infrastructure, le besoin de trouver automatiquement aussi bien les services que leur position dans le réseau devient encore plus crucial.

F.3.1 Classification des protocoles de découverte de service

Les protocoles de découverte de services peuvent être classés selon deux critères, le mécanisme de découverte et l'utilisation de répertoires de services (figure F.10) [Duda 02, Ciar 02].

Il existe deux mécanismes de découverte de services, la découverte active ou passive:

- Passive: les nœuds hébergeant des services s'annoncent périodiquement dans le réseau tandis que les autres écoutent et notent ces annonces. Ainsi, lorsque qu'un nœud désire obtenir un service, il n'a qu'à vérifier dans la liste des services obtenus lors des annonces et choisir le service désiré.

  L'avantage d'utiliser une découverte de services passive est que seuls les nœuds proposant un service font des annonces dans le réseau. Cependant, la découverte passive peut parfois rater des services dont les nœuds sont inactifs ou bien venant d'arriver sur le réseau.

- Active: les nœuds désirant un service initient une recherche de services dans le réseau. Ainsi, ils envoient une requête dans le réseau contenant le type de service désiré. Tous les nœuds proposant un ou plusieurs services correspondant à la demande répondent par un message. L'avantage par rapport à la découverte passive est que chaque service est découvert. Par contre, la découverte active peut se révéler trop coûteuse en termes de bande passante si le nombre de requêtes et de réponses devient trop important.

L'autre caractéristique qui différencie les protocoles de découverte de services est l'utilisation de répertoire de services. Les répertoires de services (*Service directories* sont des nœuds qui jouent le rôle d'intermédiaire et allègent ainsi la charge pour les nœuds aux alentours. Ils ont pour mission d'enregistrer, relayer et répondre aux différents messages de requêtes ou d'annonces provenant du réseau. Lorsqu'une requête de service circule sur le réseau, un nœud répertoire qui a enregistré et donc connaît la réponse, va intercepter cette requête et y répondre directement. Ainsi, le message n'aura, peut être, parcouru que quelques sauts au lieu de tout le réseau pour obtenir une réponse. De plus, les nœuds répertoire peuvent regrouper, voire même filtrer, les annonces de services et ainsi relayer une information plus condensée pour son voisinage. Pour être le plus efficace possible, ces répertoires doivent être soigneusement placés sur des nœuds bien positionnés et aussi de préférence stables en termes de durée de vie et de visibilité dans le réseau.

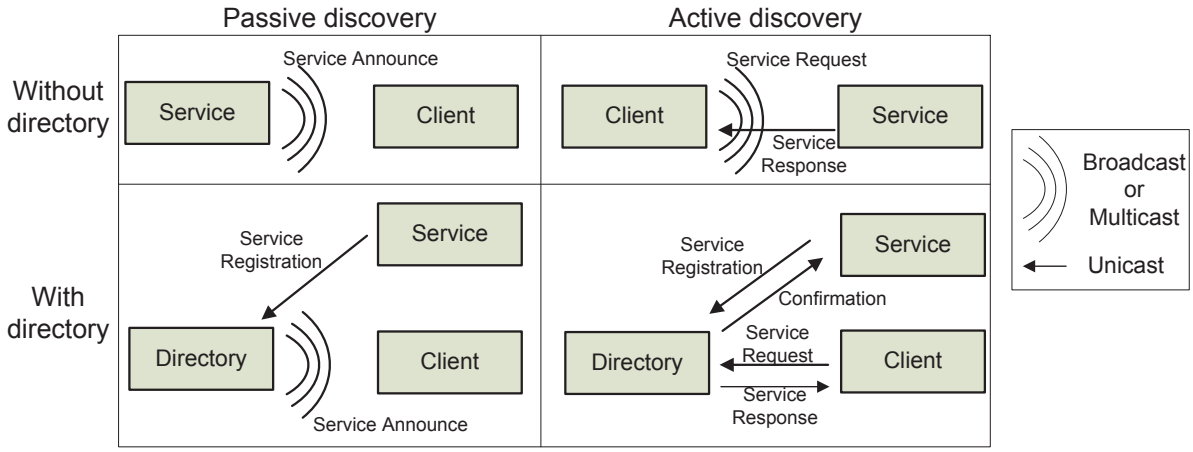


Figure F.10: Classification des protocoles de découverte de service. [Ciar 02]

F.3.2 Découverte de service dans les réseaux filaires

Dans les paragraphes suivants nous décrivons les protocoles de découverte de services les plus significatifs des réseaux filaires.

SLP: Le Service Location Protocol [Veiz 97] utilise des URLs (Uniform Resource Locator)[Unif 94] pour localiser les services dans le réseau. Ces URLs ont le format suivant: "`service:<srvtype>://<addr-spec>`" où "srvtype" est le type de service parmi une liste de services enregistrés auprès de la IANA (Internet Assigned Numbers Authority) et "addr-spec" est le moyen d'accès au service (`[<user>:<password>@]<host>[:<port>]`) par le client. Dans des réseaux de petite taille, SLP fonctionne simplement en utilisant les adresses mulitcast pour la communication. Dans des réseaux plus grands, SLP utilise des nœuds répertoires appelés *Directory Agents* qui répliquent l'information dans les différentes portions du réseau.

UPnP: Universal Plug and Play [UPnP][UPnP doc], développé par Microsoft et IBM, étend le concept du "plug and play" existant dans le domaine du matériel informatique qui permet sa reconnaissance automatique par les machines. UPnP utilise SSDPP[45][Gola 99] pour la découverte de service. La description des services se fait à l'aide de fichiers XML[46][Bray 08]. Une requête de service UPnP est émise en utilisant SSDP. Les réponses correspondantes contiennent l'URL[47][Unif 94] qui pointe vers le fichier XML décrivant le service et sa mise en œuvre. UPnP est très populaire dans les réseaux domestiques, aidé par sa mise en avant par Microsoft dans ses systèmes d'exploitation. Cependant, aussi bien le SSDP et l'utilisation de fichiers XML ne sont pas adaptés aux réseaux à faible fiabilité et à faible bande passante tel que les réseaux ad hoc.

Zeroconf: Zero configuration Networking [Ches 05, Zero] est un regroupement de plusieurs techniques qui ensemble permettent l'établissement d'un réseau IP fonctionnel sans configuration et sans nécessiter une infrastructure tel qu'un serveur DNS ou DHCP. Zeroconf est composé de trois éléments:

[45] Simple Service Discovery Protocol
[46] eXtended Markup Language
[47] Uniform Resource Locator

- un module qui permet l'attribution automatique d'une adresse IP (sans l'utilisation de DHCP).
- un module qui permet d'avoir les fonctionnalités de DNS mais sans l'utilisation d'un serveur. Ce module appelé multicast DNS (mDNS) [Ches 11] utilise le multicast pour diffuser les annonces et les requêtes dans le réseau. L'adresse multicast réservée auprès de la IANA est 224.0.0.251 pour IPv4 et ff02::fb pour IPv6.
- un module de découverte de services nommé DNS Service Discovery (DNS-SD) [Ches 06]. DNS-SD, propose un format spécifique d'entrée DNS pour l'enregistrement et la description des services. Ces entrées sont également compatibles avec les serveurs DNS classiques.

Jini: Jini[Kuma 01], introduit en 1998 par Sun Microsystems, est une technologie basée sur Java qui permet la découverte de services et d'utilisateurs dans un réseau. La communication entre les entités se fait à l'aide de RMI (Remote Method Invocation). Jini utilise exclusivement des nœuds répertoires, appelés *Jini Lookup Service*, pour la découverte et l'enregistrement des services. La découverte se fait en trois étapes:

- "Discovery": La première phase est réalisée par les nœuds *Lookup Service*. Ceux-ci vont s'annoncer dans le réseau, en utilisant le mulitcast, afin de diffuser des objets Java appelés *Lookup Proxy*. Le Lookup Proxy permettra alors aux nœuds de contacter le Lookup Service correspondant.
- "Join": Chaque nœuds proposant un service enregistre son service auprès d'un nœud Lookup Service (en utilisant le Lookup Proxy). Lors de cet enregistrement, le nœud y dépose un objet Java appelé *Service proxy*, qui va permettre par la suite au client de savoir comment et où contacter le service.
- "Lookup": Un nœud recherchant un service envoie une requête au Lookup Service et obtient, en réponse, la liste des services correspondants à sa requête. Il signale alors le service qu'il a choisi au Lookup Service et obtient le *Service Proxy* du service.

F.3.3 Découverte de services dans les réseaux ad hoc

Il existe aussi des protocoles de découverte de services spécifiquement développés pour les réseaux ad hoc. Cependant, aucun d'entre eux, à ce jour, n'a réussi à se démarquer des autres ou à être reconnu par la communauté. Les protocoles suivants ont été choisis pour leurs approches intéressantes et variées.

Konark: [Hela 03] est un protocole de découverte de services qui utilise XML pour la description des services et le multicast pour la diffusion des annonces de services. Il ressemble très fortement à UPnP. Il n'y a pas de nœud répertoire dans Konark, chaque nœud est indépendant. Pour l'interaction avec les services, il utilise HTTP et SOAP[48], ce qui nécessite la présence d'un serveur micro-HTTP.

Allia: *Alliance-based service discovery* est un protocole dont les décisions sont basées sur des politiques définies à partir d'un profil d'utilisateur. Chaque nœud ajoute des nœuds de son voisinage dans son alliance. Cette alliance est local, chaque nœud a sa propre alliance et ne sait pas dans quelles alliances il a été ajouté. Les nœuds appartenant à une alliance sont alors utilisés en tant que nœud répertoire. Les nœuds peuvent à la fois jouer le rôle de répertoire

[48] Simple Object Access Protocol[Simp 00]

pour un nœud et utiliser d'autres nœuds en tant que répertoire. Les liens créés par les alliances forment ainsi une structure pair-à-pair décentralisée. Lorsqu'un nœud ne répond plus après un temps donné, ce nœud est simplement et silencieusement retiré de l'alliance et un autre nœud est sélectionné pour le remplacer. Les nœuds qui hébergent un service, l'annonce périodiquement sur le réseau. Ainsi, pour trouver un service dans le réseau, un nœud procède par étape. En premier, il vérifie dans sa liste des services enregistrés lors des annonces. S'il ne le trouve pas, il questionne un des nœuds dans son alliance. Si ce dernier ne l'a pas non plus dans sa liste locale, il va lui même contacter des nœuds dans sa propre alliance. Ainsi, une requête peut traverser plusieurs alliances avant de trouver une réponse. Pour chaque décision, Allia fait appel aux politiques locales définies par les configurations de l'utilisateur. Ces politiques permettent un réglage fin, adapté aux préférences de l'utilisateur et aux capacités de l'appareil.

Découverte de service basé sur OLSR: Certains protocoles de découverte de services reposent, pour le transport des messages de contrôle, sur un protocole de routage sous-jacent. Dans [Jodr 06] et [Li 05] les auteurs utilisent le protocole OLSR pour transporter les informations de découverte de services. En ajoutant des champs dans les messages OLSR, ils profitent ainsi de la dissémination optimisée par les MPRs.

Découverte de services en utilisant une approche par champs d'attraction: L'article [Lend 05] propose une approche qui s'inspire des champs électrostatiques. Un service est représenté par une charge "positive" et une requête est représentée par une charge, opposée, "négative" attirée par la charge positive. Les nœuds du réseau calculent alors un potentiel électrique pour une requête. La requête circule, à chaque fois, vers le voisin avec le plus grand potentiel et avance vers sa destination (Figure F.11). Les nœuds hébergeant des services diffusent périodiquement des messages d'annonce de services. Cette approche est intéressante dans sa vision et son analogie avec les champs électriques mais ne comporte pas d'optimisation ou d'amélioration de la diffusion des messages.

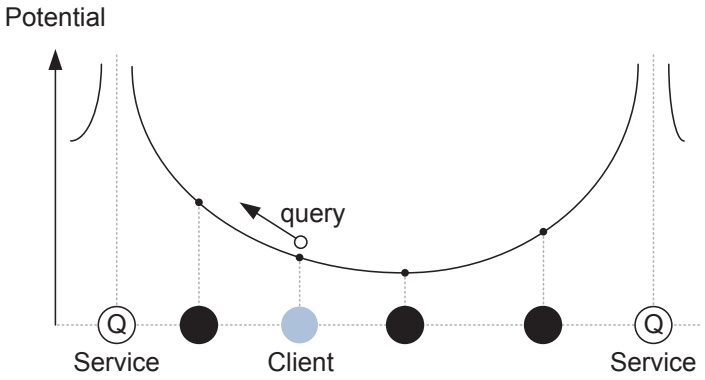


Figure F.11: Découverte de services utilisant l'approche par champs électrostatiques.

Scalable Service Discovery for MANET: [Sail 05] est une architecture de découverte de services à répertoires distribués. L'architecture se base sur OLSR, en utilisant les nœuds MPR comme premier niveau de hiérarchie. Les répertoires sont sélectionnés à la volée en prenant comme critère principal la couverture des nœuds du voisinage. Les nœuds répertoires s'échangent entre eux les profils et informations à l'aide de filtres de bloom et en utilisant la diffusion de OLSR

(sous-jacent). Comparé à notre approche, décrite plus tard dans la thèse, le désavantage ici est que même les nœuds qui ne sont que de passage, pour un court instant, peuvent être sélectionnés très momentanément comme nœud répertoire. Ceci induit une surcharge du réseau due aux nombreux changements de nœuds répertoires.

F.4 Métriques et contexte

Pour améliorer ou optimiser une application, un protocole ou un service dans un réseau ad hoc, une bonne stratégie est de tirer avantage du contexte. Le contexte peut être défini comme dans [Dey 01]:

> *"Le contexte est toute information qui peut être utilisée pour caractériser la situation d'une entité. Une entité est une personne, un endroit ou un objet qui est considéré comme pertinent lors d'une interaction entre un utilisateur et une application, y compris l'utilisateur ou l'application eux-mêmes."*

Une situation peut être caractérisée par de nombreux types d'informations. Pour exprimer ce contexte en information calculable et traitable automatiquement, des métriques sont utilisées. Une métrique est une mesure qui permet de quantifier une information. Un/une protocole/application/service qui est capable de prendre en compte le contexte est dit *context-aware*.

Être **context-aware** est défini dans [Dey 01] :

> *"Un système est dit context-aware s'il utilise le contexte pour apporter une information plus pertinente à l'utilisateur ou au service, où la pertinence dépend directement de l'activité de l'utilisateur."*

Comparé au réseau filaire classique, le contexte dans les MANETs peut provenir de sources beaucoup plus diverses. Un nœud dans un réseau ad hoc peut être caractérisé par le réseau, les caractéristiques techniques de l'appareil, la position géographique, mais aussi et surtout par l'aspect humain qui caractérise et différencie le plus les réseaux ad hoc des autres réseaux. Le contexte humain des réseaux ad hoc élargit grandement les possibilités et aussi le contexte. Ainsi, le contexte humain comme les émotions et les humeurs peut influencer le réseau. Obtenir et prendre en compte le contexte dans les réseaux ad hoc relève les défis suivants:

- Définir ce qu'est un contexte pertinent et utile. Une information de contexte n'est que intéressante si l'on peut en tirer avantage.
- Obtenir ce contexte n'est pas toujours évident. Le contexte qui concerne l'aspect humain ou encore les données du monde physique (tel que la position géographique) ne sont souvent pas des informations facile à obtenir.

Appendix G

Dissémination: SLSF - Stable Linked Structure Flooding

Contents

G.1 Vue d'ensemble: NLWCA - WCPD - SLSF-R - SLSF 189
G.2 SLSF - Stable Linked Structure Flooding 191
G.2.1 ICR – Inter-Cluster Relays 191
G.2.2 Diffusion dans SLSF 193
G.3 SLSF avec recouvrement d'erreur 193
G.4 Résultats 194
G.5 Conclusion 196

Ce chapitre présente le protocole SLSF (Stable Linked Structure Flooding), qui repose sur une structure de cluster crée par NLWCA. Le but de SLSF est d'améliorer la dissémination des messages en termes de nombre de nœuds relais et de bande passante utilisés pour sa mise en place.

SLSF part du même principe que NLWCA qui est que l'information obtenu dans le voisinage très proche est la plus fiable. La stratégie de SLSF est d'optimiser au mieux la dissémination localement pour des petits groupes, au lieu de s'intéresser au réseau en entier d'un seul coup. SLSF découvre et établit des liens stables entre les clusters créés par NLWCA. En reliant les clusters entre eux, SLSF construit une structure de dissémination qui diminue le coût de diffusion d'une information dans le réseau. Le point clé de SLSF réside dans sa façon de sélectionner les nœuds intermédiaires entre les clusters qui vont jouer le rôle de relais. De plus, pour pallier à la fiabilité des transmissions, un mécanisme de recouvrement des erreurs local est ajouté.

G.1 Vue d'ensemble: NLWCA - WCPD - SLSF-R - SLSF

Afin de bien différencier et d'éviter les confusions entre les protocoles NLWCA, WCPD, SLSF-R et SLSF, leurs objectifs ainsi que leurs interactions sont résumés dans ce paragraphe. SLSF-R et SLSF, font partie des contributions et sont détaillés plus tard.

- NLWCA: est un protocole de clustering. Il établi des clusters à 1 saut basés sur le poids des nœuds mais aussi des liens. Il ne dispose pas d'un mécanisme de diffusion de message.
- WCPD: répond au manque de NLWCA et propose un mécanisme de dissémination. Dans WCPD, les messages sont broadcastés à 1 saut à l'intérieur des clusters et unicastés sur plusieurs sauts entre les clusters.

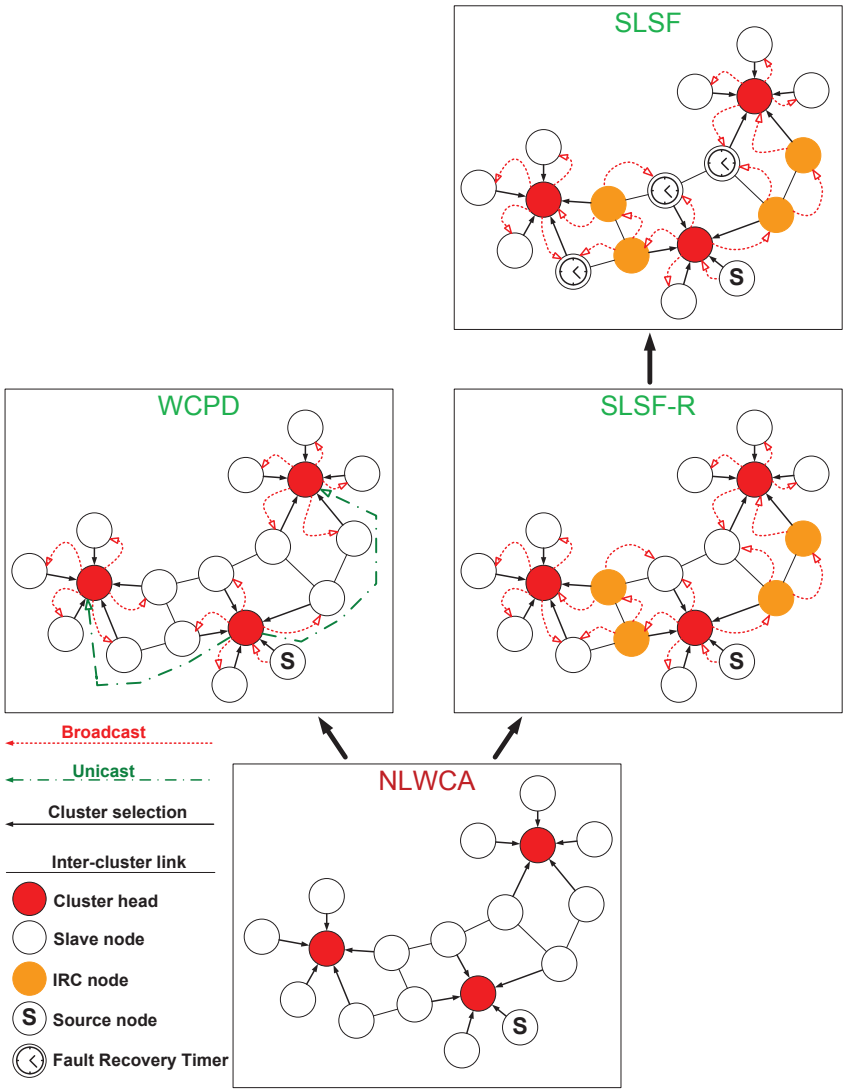


Figure G.1: Hiérarchie des protocoles NLWCA - WCPD - SLSF-R et SLSF: Dissémination.

- SLSF-R: SLSF a deux versions, une version complète et une version "allégée" sans mécanisme de recouvrement des erreurs appelée SLSF-R (lire, SLSF moins R pour recouvrement). SLSF-R remplace le protocole WCPD existant par une dissémination plus efficace. A noter que SLSF-R et WCPD partagent le même format de beacon et consomment ainsi la même bande passante pour la mise en place de la dissémination.

- SLSF: est la version complète avec mécanisme de recouvrement. Le format de beacon n'est plus le même, il faut ajouter des numéros de séquence pour rendre possible le recouvrement des erreurs. Les performances de dissémination sont nettement améliorées.

La figure G.1 montre la relation entre les protocoles du point de vu de la dissémination, tandis que la figure G.2 la montre du point de vue des formats des beacons.

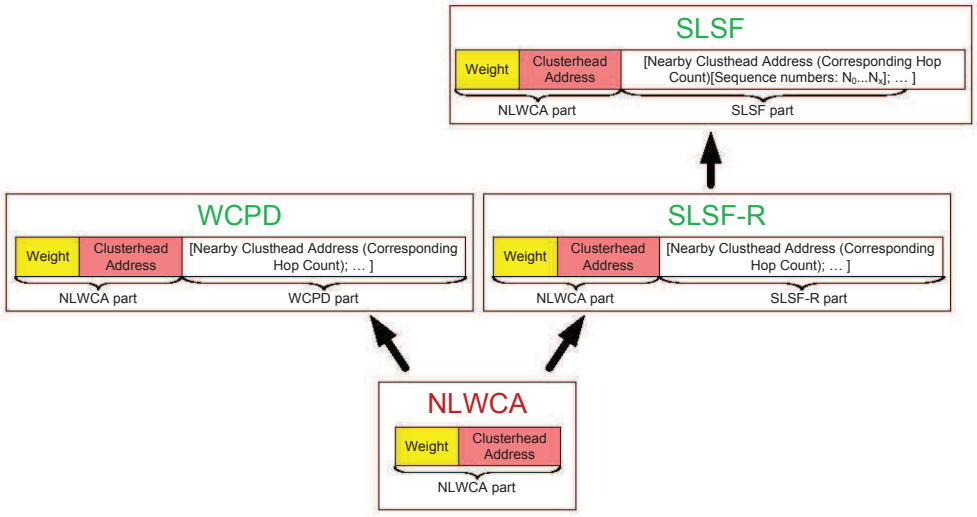


Figure G.2: Hiérarchie des protocoles NLWCA - WCPD - SLSF-R et SLSF: Format des messages.

G.2 SLSF - Stable Linked Structure Flooding

Le protocole SLSF (Stable Linked Structure Flooding)[Lecl 10b, Lecl 10c] établit une structure basée sur des clusters stables créés pas NLWCA, qui permet d'obtenir une dissémination efficace et qui passe à l'échelle. SLSF obtient ce résultat en sélectionnant de manière propice les nœuds intermédiaires entre les différents clusters. Ces nœuds sont alors appelés ICR (*Inter-Cluster Relay*). De plus, SLSF comprend aussi un mécanisme de recouvrement d'erreurs qui permet de détecter et de réparer, localement entre deux clusters, les erreurs de transmission. SLSF est une très bonne structure de base pour d'autres protocoles, ceux-ci pouvant en tirer avantage et ainsi réduire le coût de leur dissémination.

G.2.1 ICR – Inter-Cluster Relays

NLWCA crée des clusters à un saut strict. Il s'agit maintenant de relier et de diffuser, de la meilleure manière possible, ces clusters entre eux. En utilisant le même format de beacon que WCPD, nous découvrons les clusterheads voisins (chaque nœud annonce les clusterheads auquel il est connecté). En utilisant l'information des clusters voisins et la distance (en nombre de sauts) pour les atteindre, nous proposons le mécanisme d'Inter-Cluster Relay. Le but des ICR est d'atteindre tous les clusterheads voisins en minimisant les nœuds relais intermédiaires. Ces nœuds relais sont appelés ICRs.

Le choix des nœuds intermédiaires reste simple de part la nature des clusters formés qui restreint les possibilités de formation des nœuds entre ces clusters, comme démontré sur la figure G.3.

Le processus de sélection des ICRs est le suivant:

i. Choisir en tant que ICR, les nœuds qui sont les seuls à atteindre un clusterhead voisin.

ii. Retirer les clusterheads désormais couverts pas ces nouveaux ICRs.

iii. Retirer dans le choix de nœuds intermédiaires pour un clusterhead, les nœuds qui n'atteignent pas le clusterhead avec le nombre de saut minimal disponible (on ne garde que les chemins les plus courts).

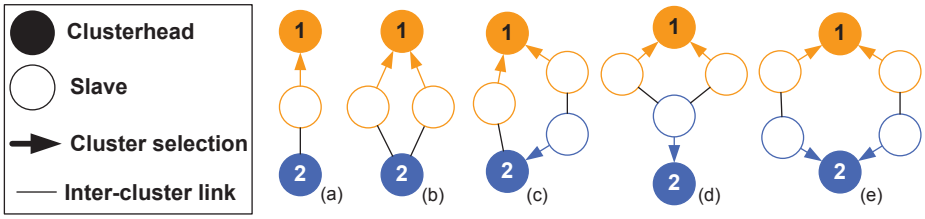


Figure G.3: Exemple de configuration inter-cluster avec 1 et 2 clusterheads.

1. Pour chaque intermédiaire potentiel, on calcule le nombre de clusterheads voisins qu'il peut atteindre.
2. Ensuite on choisit celui qui atteint le plus grand nombre.
3. En cas d'ex-æquo, on choisit celui avec le plus grand poids.
4. En cas d'ex-æquo, on choisit celui avec la plus grande adresse IP.
5. Retirer les clusterheads désormais couvert par ces nouveaux ICRs.
6. Tant qu'il y a des clusterheads non couverts par un ICR, on recommence à l'étape 1.

Les messages dupliqués dans SLSF:
Un message est considéré comme " déjà reçu ", donc un duplicata, si celui-ci a la même adresse source et le même numéro de séquence qu'un précédent message. Les informations des messages pour la vérification de duplicata sont conservées en mémoire pendant un temps donné.

Voisinage de cluster

Les configurations au niveau des nœuds intermédiaires sont restreintes par le fait qu'il ne peut y avoir au maximum que deux nœuds esclave entre deux clusterheads. Ceci a un impact aussi sur le voisinage qu'un clusterhead perçoit. Sur le réseau de la figure G.4a, les 3 clusterheads n'ont pas la même vue du voisinage. En effet, si l'on se met du point de vue du clusterhead 2, sur la figure G.4b, il y perçoit les deux clusters voisins, 1 et 3. Par contre, le clusterhead 1, figure G.4c, et le clusterhead 3, figure G.4d, ont un point de vue différent et ne voient que le clusterhead 2 dans leur voisinage. Ceci est tout simplement dû au fait qu'entre le clusterhead 1 et 3 il y a 3 nœuds esclaves A, B et C, ils ne peuvent donc pas être considérés comme voisins. Aux premiers abords, ceci peut paraître comme un désavantage ou une limitation, mais en fait c'est un grand avantage car ceci limite grandement la complexité. Prenons l'exemple sur la figure G.5a, on y voit une suite de clusterheads avec, au centre, une longue suite de nœuds esclave. Si l'on autorisait plus de 2 nœuds entre les clusterheads, on aurait alors le clusterhead 1 qui serait voisin avec les sept autres clusterheads de la figure. On aurait alors des problèmes de gestion de voisinages trop étendus. En restreignant le voisinage à ce maximum de 2 nœuds entre les clusterhead, on obtient une vue locale pour chaque clusterhead de son voisinage tel que sur la figure G.5b. On a alors, de petits voisinages allant au maximum à 3 sauts. Le prix de ce choix est que la dissémination d'un message allant du clusterhead 1 vers le clusterhead 8, par exemple, passe par chacun des clusterheads de 2 à 7, alors qu'il pourrait théoriquement passer seulement par la longue séquence de nœuds esclaves.

Un exemple de réseau avec 5 clusters est donné sur la figure G.6. Cette exemple montre bien le gain que l'on peut obtenir en utilisant SLSF, puisque seuls les clusterheads (marqués en vert)

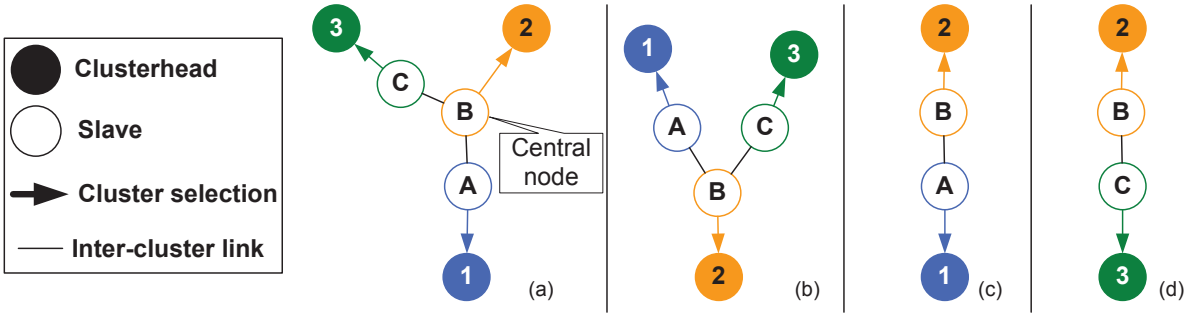


Figure G.4: a) Configuration de cluster avec plus de 3 sauts entre deux clusters. b) Point de vue du nœud 2. c) Point de vue du nœud 1. d) Point de vue du nœud 3.

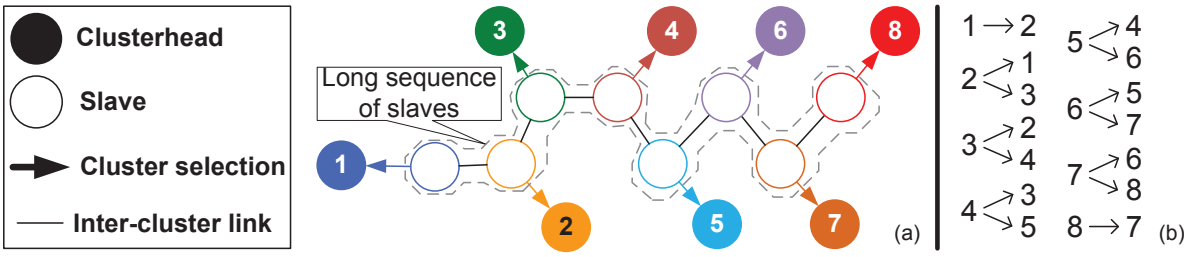


Figure G.5: a) Longue séquence de nœuds esclaves dans SLSF. b) Vue de chacun des clusters sur son voisinage proche.

et les nœuds ICRs (nœuds 1, 2, 3, 4 et 5) relayent les messages tout en atteignant tous les nœuds du réseau. Ainsi, 10 nœuds suffisent pour atteindre les 23 nœuds du réseau. Par comparaison, sur ce même exemple, OLSR utilise 15 nœuds relais pour atteindre ces mêmes 23 nœuds.

G.2.2 Diffusion dans SLSF

La diffusion de messages dans SLSF est simple, seuls les nœuds qui sont clusterhead ou ICR relayent les messages. A chaque relai, le nœud, le clusterhead ou ICR, va inclure sa sélection de nœuds ICRs dans l'entête du message. Ainsi, les nœuds recevant ce message sauront, une fois cet entête extrait, s'ils doivent relayer ce message ou pas. Le message atteint ainsi tous les nœuds du réseau tout en n'ayant été relayé que par un sous-ensemble des nœuds.

G.3 SLSF avec recouvrement d'erreur

Nous ajoutons, à SLSF, un mécanisme de recouvrement d'erreur publié dans [Lecl 10c]. Nous distinguerons à l'avenir les deux par, SLSF-R (moins le recouvrement) la version sans recouvrement et SLSF la version complète. Le but du recouvrement d'erreur et de pouvoir corriger les erreurs de retransmission au niveau local. Le recouvrement d'erreur est capable de retransmettre un message, dont la diffusion a échoué, sans que la source du message n'ait à réémettre ce message. L'erreur est détectée et pris en compte localement par les nœuds non-ICR. Deux mécanismes permettent le recouvrement d'erreur entre deux clusterheads:

- Le mécanisme d'accusé de réception: Afin de permettre de savoir si un message arrive correctement au prochain clusterhead, nous ajoutons un numéro de séquence qui sera repris

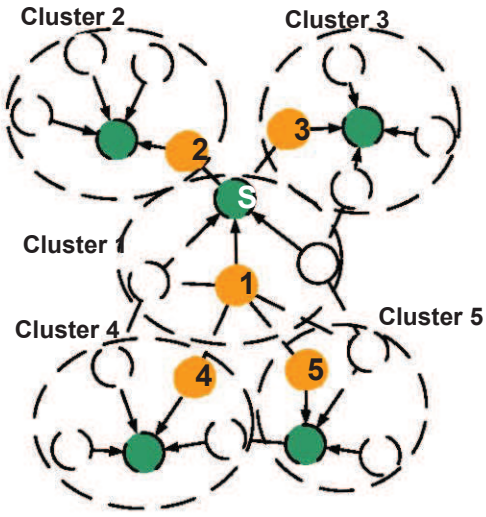


Figure G.6: Sélection ICR dans un réseau avec 5 clusters.

dans les beacons de chaque nœud.

- Le mécanisme de retransmission retardée: les nœuds non-ICR, donc ceux qui n'ont pas encore relayé le message, vont retransmettre les messages qui n'avaient pas été relayés correctement. Pour cela, ils écoutent les messages, à l'aller, et attendent les accusés de réception correspondants, au retour. Si un accusé de réception pour un message n'est pas reçu après un temps donné, le message est alors retransmis.

Le choix de garder local le mécanisme de retransmission est important. Il ne s'agit pas d'assurer la transmission sur toute sa longueur, mais plutôt de garder les informations et leur traitement local. Les beacons permettent de faire remonter les accusés de réception localement, ce qui ne serait pas possible si l'accusé de réception devait traverser tout le réseau car trop coûteux.

Aussi, retransmettre les messages par les nœuds non-ICR est un choix stratégique. Puisque la transmission par le chemin initial, les ICRs, a échoué, la retransmission se fait par des nœuds n'ayant pas encore essayé la transmission. Le message peut alors arriver à destination en utilisant un chemin alternatif.

G.4 Résultats

Pour évaluer les performances de notre protocoles SLSF, nous avons implémenté les trois protocoles (OLSR, NLWCA/WCPD et SLSF) dans le simulateur [Gorg 07]. Les simulations en ont été faites avec le modèle de mobilité Restricted Random Way Point [Blav 02], où des nœuds se déplacent sur des routes prédéfinies d'une carte du Luxembourg (figure G.7) et décident à chaque croisement de la nouvelle route à suivre. Pour chaque variante de simulation nous avons utilisé 10 graines aléatoires différentes afin d'obtenir des résultats avec des topologies et des mouvements à chaque fois différents. Le temps de simulation est de 1000 secondes pour chaque simulation et la vitesse des nœuds variait entre [0.5;1.5] unités/s avec une portée de transmission sans fil de 25 unités.

Le but de ces simulations est de mesurer la bande passante utilisée par les protocoles pour établir la structure de dissémination (MPR pour OLSR, cluster pour WCPD et cluster+ICR pour

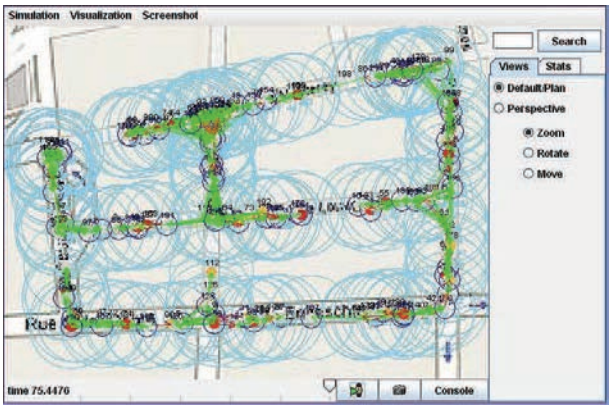


Figure G.7: JANE simulating the protocols on 100 devices. The mobile devices move on the streets of the Luxembourg City map. The devices move with a speed of 0.5 - 1.5 m/s.

SLSF) et les performances de dissémination. De plus nous comparons l'efficacité des protocoles et les performances de dissémination lors d'un scénario statique sans mouvement.

Pour mesurer la bande passante utilisée pour la mise en place de la topologie par chaque protocole, nous avons, pour OLSR, mesuré la bande passante utilisée par les messages HELLO, et pour WCPD et SLSF, la bande passante utilisée par les beacons échangés. Les résultats sur la figure G.8 montrent que OLSR consomme plus de bande passante dans toutes les configurations : 100, 200 et 300 nœuds. SLSF-R, quant à lui consomme le moins de bande passante pour la mise en place de la structure. SLSF, lui, consomme plus de bande passante que SLSF-R, à cause de l'ajout de numéros de séquences dans le beacon pour le recouvrement d'erreurs. Tous deux, SLSR et SLSR-R, consomment dans tous les cas moins de bande passante qu'OLSR. On s'attendait à un tel résultat, puisque là où OLSR doit échanger la liste de tout son voisinage à un saut, SLSF lui ne doit échanger que la liste des clusterheads de son voisinage, qui est plus petite dans la grande majorité des cas et au maximum égale.

Les résultats des performances de dissémination sont présentés sur la figure G.9. Pour chaque protocole on y retrouve deux courbes, l'une, *messages sent*, représente le nombre de fois qu'un même message est envoyé (ou encore le nombre de nœuds ayant relayé le message) et l'autre, *receives*, représente le nombre de nœuds qui ont reçu le message. Une bonne performance est un protocole qui minimise la courbe *messages sent* et qui maximise celle des *receivers*.

Si l'on se concentre sur la figure représentant les résultats pour 300 nœuds simulés, on peut voir que SLSF et OLSR obtiennent un nombre de *receivers* très similaire. Cependant, pour atteindre ceci, OLSR utilise/nécessite en moyenne 10 à 15% de *messages sent* (nœuds relais) en plus que SLSF. De plus, OLSR nécessite en moyenne 40% de bande passante en plus pour la mise en place des MPRs que SLSF. En comparant maintenant, SLSF-R et WCPD, on peut observer que SLSF-R atteint de 10 à 20% de nœuds en plus tout en ayant un nombre de messages relayés très proche. La bande passante consommée pour la mise en place de la structure étant strictement la même, ce gain est attribuable complètement au mécanisme des ICRs.

Nous avons également évalué le rapport qualité/prix des protocoles en combinant les deux résultats précédents. Ce rapport qualité/prix représente l'efficacité d'un protocole et est représenté par le pourcentage de nœuds atteints divisé par la bande passante utilisée. Sur la figure G.10 on voit que SLSF et SLSF-R sont dans les 3 cas (100, 200 et 300 nœuds) environ 2 à 3 fois plus efficaces qu'OLSR. Les performances médiocres de WCPD, mettent en avant l'amélioration qu'apporte SLSF et son mécanisme des ICRs.

Pour évaluer les protocoles sur un pied d'égalité, nous avons également réalisé un scénario

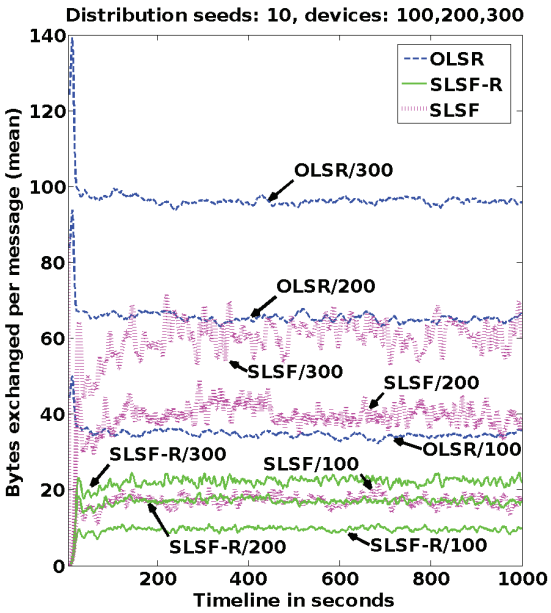


Figure G.8: Bande passante nécessaire à la mise en place de la topologie pour 100, 200 et 300 nœuds.

statique sans mouvements de nœuds. En effet, puisque NLWCA utilise la stabilité des liens pour prendre en compte ou non un nœud dans son calcul, les nœuds qui bougent trop vite peuvent se retrouver exclus de ce calcul. Ainsi, là où OLSR utilise tous les liens, NLWCA n'utilise que les liens qu'il considère comme stables. OLSR peut dans certains cas (très instables) obtenir de meilleures performances de dissémination au travers de ces liens instables, mais il le paye au niveau de la bande passante pour maintenir à jour cette structure (instable). Les résultats de la figure G.11 montrent que, dans un scénario statique où tous les liens sont stables, SLSF est beaucoup plus performant que OLSR. Pour atteindre le même nombre de nœuds, OLSR nécessite 85% de ces nœuds en tant que relais tandis que SLSF fait chuter le nombre de relais entre 60 et 30%.

G.5 Conclusion

Ce chapitre a décris SLSF, un protocole de dissémination qui sélectionne des Inter-Cluster Relays afin d'optimiser la communication entre des clusters connectés de manière stable. De plus, un mécanisme de recouvrement d'erreurs a été ajouté pour pallier les erreurs de transmission intermittentes.

Le but des ICRs (Inter-Cluster Relay) est d'atteindre tous les clusterheads voisins en utilisant un ensemble minimal de nœuds à un saut tout en gardant le plus court chemin. La sélection des ICRs réduit nettement le nombre de nœuds relais nécessaire à la diffusion. Comme démontré lors des simulations, SLSF atteint de bonnes performances tout en consommant peu de bande passante.

SLSF offre une très bonne structure de base pour des applications ou des protocoles pour lesquelles la dissémination est l'une des composantes principales. Nous allons, utiliser SLSF comme structure de dissémination à deux reprises. Dans le chapitre H nous utilisons SLSF comme base pour le protocole de routage SLSR et, dans le chapitre I, comme structure de dissémination pour la découverte de service.

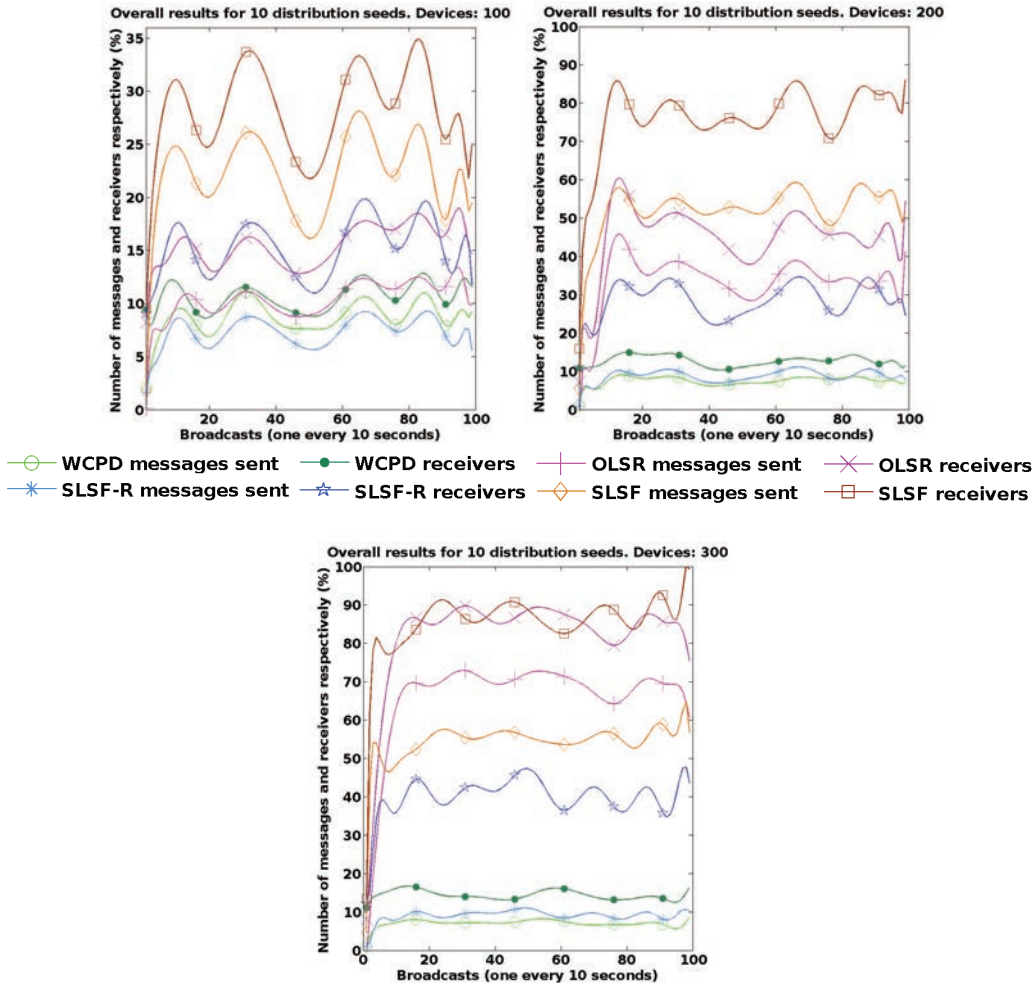


Figure G.9: Nombre total de messages relayés et nombre de messages effectivement reçus pour 100, 200 et 300 nœuds. (lissé, pour une plus grande visibilité, avec une équation polynomial du16ième degré).

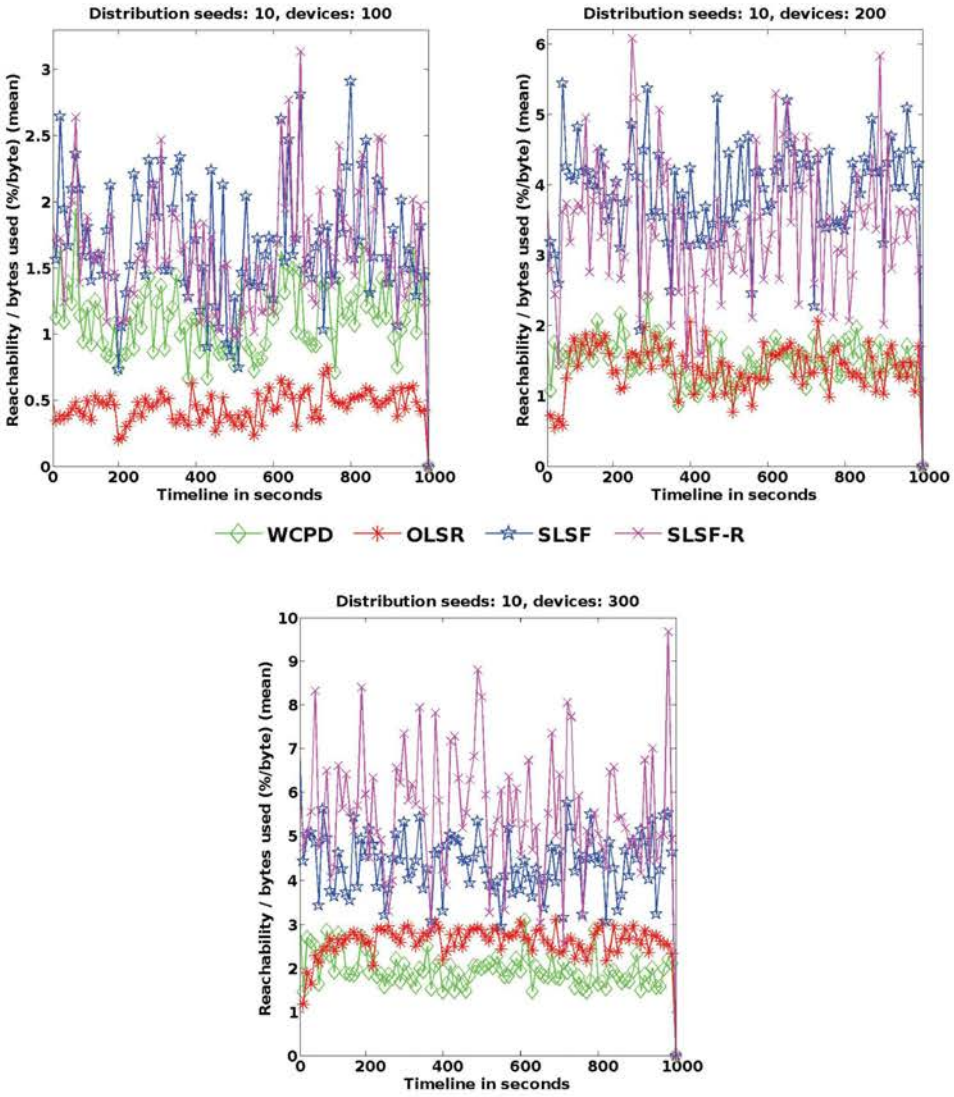


Figure G.10: Efficacité de l'utilisation de la bande passante pour 100, 200 et 300 nœuds.

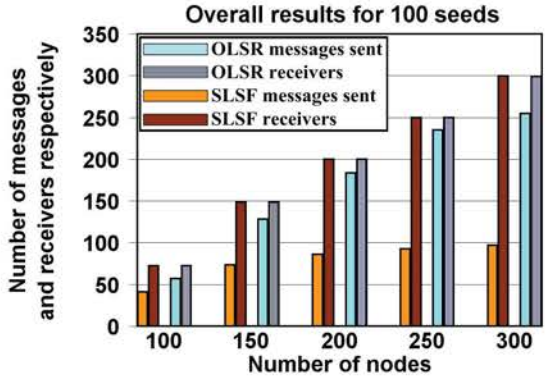


Figure G.11: Scénario statique avec 100 et 300 nœuds.

Appendix H

Routage: SLSR - Stable Linked Structure Routing

Contents

H.1 SLSF en tant que structure de base 199
H.2 Routage des messages 200
H.3 Type de messages 201
H.4 Un protocole dit "balanced-hybrid" 202
H.5 Table de routage et mise à jour 202
H.5.1 Choisir une route 202
H.5.2 Mise à jour de la table de routage 203
H.6 Comportement symbiotique 205
H.6.1 Higher-layer traffic 205
H.7 Conclusion 205

Ce chapitre présente SLSR (Stable Linked Structure Routing), un protocole de routage basé sur la structure de dissémination SLSF. On peut classer notre protocole de routage parmi les protocoles dit hybride-balancés qui ont les fonctionnalités des deux types: ceux à état de lien et ceux à vecteur de distance. C'est un protocole proactif, il maintient à jour les routes en envoyant des messages d'annonce soit périodiquement soit suite à des évènements.

Dans un souci de réduction du coût du routage, SLSR, prend en compte le trafic déjà présent dans le réseau. SLSR ce comporte alors de façon symbiotique avec les applications ou protocoles supérieurs en analysant le trafic pour y ajouter, lorsque cela est propice, ses informations de routage. SLSR peut alors, au lieu d'envoyer un message complet avec les informations de routage, ajouter un simple entête contenant les informations de routage aux messages diffusés par d'autres applications ou protocoles.

Comme dans SLSF, les clusterheads jouent un rôle clé dans SLSR. Les nœuds esclaves ont un rôle très simple, ils écoutent seulement les informations que leur donne leur clusterhead. Les clusterheads eux vont faire le travail du routage des messages vers leurs destinations.

H.1 SLSF en tant que structure de base

L'utilisation de SLSF en tant que structure de base permet de simplifier la procédure de routage et de concentrer les efforts sur les clusterheads et les chemins inter-cluster (ICR). Ainsi SLSR

crée une vue simplifiée du réseau au dessus de la structure de SLSF, comme on peut le voir sur la figure H.1. Les avantages sont:

- Les informations de routage sont disséminées efficacement par SLSF.
- Les opérations de routage sont concentrées que sur les clusterheads:
  - Seul les clusterheads envoient des annonces de routages
  - Les décisions de routage ne sont faites que par les clusterheads. Les nœuds esclaves ne font que relayer des informations.
- La révocation des routes peut être agrégée au cluster entier. L'annonce de la perte d'un lien vers un clusterhead va non seulement révoquer celui-ci mais aussi révoquer tous les nœuds esclaves de ce clusterhead. Le message de révocation ne contient ainsi que l'adresse du clusterhead du cluster à révoquer.

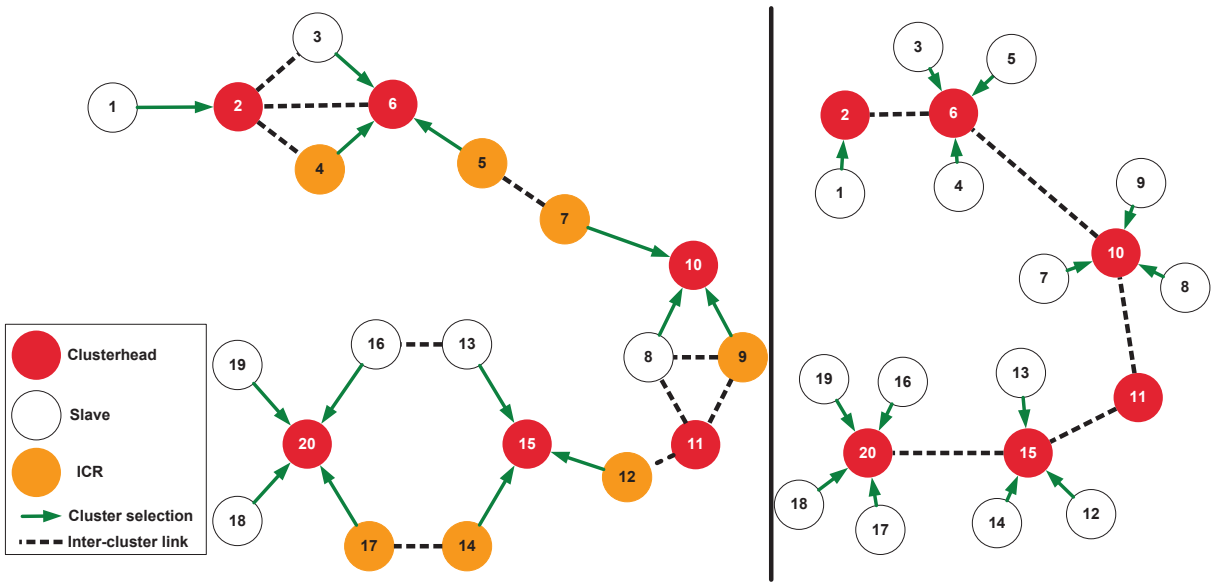


Figure H.1: Le même réseau avec les deux vues différentes : SLSF (gauche) et SLSR (droite).

H.2 Routage des messages

Le routage dans SLSR se différencie des protocoles de routage classiques. Au lieu de router les messages vers le prochain nœud voisin, SLSR envoie ses messages vers le prochain clusterhead voisin. La table de routage contient le *next clusterhead* au lieu du *next hop*. La figure H.2 donne l'exemple d'un réseau avec 3 clusters, ainsi que la vue qu'a SLSR sur ce réseau (partie droite). La différenciation entre clusterheads et esclaves se fait aussi au niveau de la table de routage. Ainsi, la table de routage du nœud 19 (figure H.3) ne contient comme seul next CH (ClusterHead), son propre clusterhead, le CH 20. Par contre, les tables de routage des clusterheads 20 et 15, sur les figures H.4 et H.5, contiennent les clusterheads vers lesquelles les messages doivent être envoyées pour atteindre leur destination.

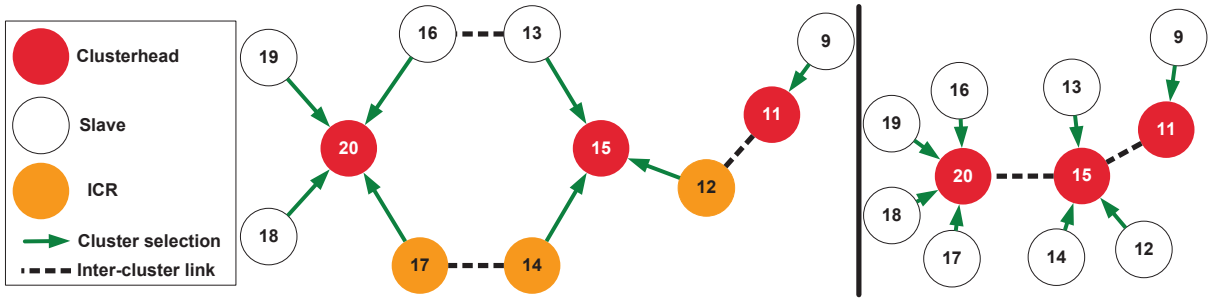


Figure H.2: SLSR - réseau avec trois clusterheads. Vue SLSF (gauche) et vue SLSR "simplifié" (droite).

Dst	Next CH
16	20
17	20
18	20
20	20
9	20
11	20
12	20
13	20
14	20
15	20

Figure H.3: Table de routage simplifiée pour le nœud esclave 19.

Dst	Next CH
16	20
17	20
18	20
19	20
9	15
11	15
12	15
13	15
14	15
15	15

Figure H.4: Table de routage simplifiée pour le clusterhead 20.

Dst	Next CH
12	15
13	15
14	15
11	11
9	11
20	20
16	20
17	20
18	20
19	20

Figure H.5: Table de routage simplifiée pour le clusterhead 15.

H.3 Type de messages

Chaque message (ou entête) de SLSR est encapsulé dans un message SLSF. Les messages de SLSR peuvent aussi être vus comme des entêtes dans le cas d'un ajout dans un message venant d'une application ou d'un protocole du niveau supérieur. SLSR propose 4 types d'entêtes:

- **Routed-message**: Permet le routage des messages dans SLSR.
- **Current slaves**: Envoyé seulement par les clusterheads, cet entête est diffusé dans tout le réseau pour annoncer les membres du cluster et permet d'obtenir le coût du parcours du message.
- **Lost slaves**: Entête envoyé par un clusterhead lorsque celui-ci perd un de ses membres. Ces messages ne sont pas envoyés périodiquement mais déclenchés par des événements. Ils peuvent être considérés comme facultatifs puisque la périodicité des messages *current slaves* permet de mettre à jour les membres d'un cluster. Selon la vitesse de mise à jour souhaitée, on peut activer ou désactiver l'envoi de ces messages.
- **Lost route to clusters**: Entête, lui aussi, envoyé seulement par les clusterheads, qui sert à annoncer la perte d'une route vers un ou plusieurs clusters voisins. Le message contient la liste des clusterheads vers lesquelles une route a été perdue. Il est alors diffusé dans tout le réseau, mais la diffusion peut être arrêtée par un nœud possédant une route alternative.

H.4 Un protocole dit "balanced-hybrid"

SLSR peut être classé en tant que protocole "balanced-hybrid". Ce terme, introduit avec le protocole Enhanced Interior Gateway Routing Protocol (EIGRP) [EIGRP 92], désigne les protocoles qui possèdent des propriétés des deux algorithmes : à état de lien et à vecteur de distance. Cela dit, SLSF n'est en rien similaire à EIGRP.

La table de routage est proche de celle d'un protocole à vecteur de distance, par contre la façon de récupérer les informations pour remplir celle-ci est similaire aux algorithmes à état de lien. Dans SLSR, les clusterheads diffusent la liste de leurs membres (les nœuds esclaves) dans tout le réseau, à la façon d'un protocole à état de lien. La table de routage, elle, ne contient que des distances et des directions pour chaque destination, à la façon d'un algorithme de vecteur de distance. Le point clé réside dans la couverture des informations diffusées et aussi dans les nœuds choisis pour ces annonces. Si l'on compare SLSR avec un protocole à état de lien et à vecteur de distance, on peut voir, sur la figure H.6, que les informations dans ces deux types d'algorithmes se recouvrent. Les informations diffusées dans SLSR ne se recouvrent pas, chaque information est propre à son cluster et est seulement diffusée par ce dernier.

SLSR est aussi capable de devenir, si nécessaire, un protocole à état de lien à part entière en étendant les informations diffusées dans les annonces. Au lieu d'annoncer seulement les membres du cluster et lui-même, un clusterhead pourrait également annoncer ses clusterheads voisins. Ainsi, à la lecture de ces annonces, on obtiendrait la topologie complète du réseau.

H.5 Table de routage et mise à jour

La figure H.8 est la table de routage du clusterhead 20 du réseau de la figure H.7. Les champs de la table de routage sont les suivants:

- **Dst:** Adresse de destination.
- **Next CH:** Adresse du prochain clusterhead menant vers la destination.
- **metric:** Métrique de coût du chemin, dans notre exemple il s'agit du nombre de sauts vers la destination.
- **CH metric:** Métrique de coût du chemin de clusters, dans notre exemple il s'agit du nombre de clusters à traverser vers la destination
- **Dst CH addr:** Adresse du clusterhead de la destination.
- **Expire Time:** Temps d'expiration d'une entrée dans la table de routage. Il est calculé en ajoutant, au moment de la mise à jour de l'entrée, le temps de validité annoncé dans le message reçu (*validity time*) et le temps courant.
- **Seq. Nbr:** Numéro de séquence du dernier message ayant causé une mise à jour de l'entrée.

H.5.1 Choisir une route

La table de routage contient toutes les routes connues pour chaque destination. Il peut y avoir plusieurs *next clusterhead* pour la même destination. Par exemple, dans le figure H.8, il y a deux entrées pour la route vers le nœud 12 (lignes *4* et *6*). Le choix d'une route pour une destination donnée est le suivant:

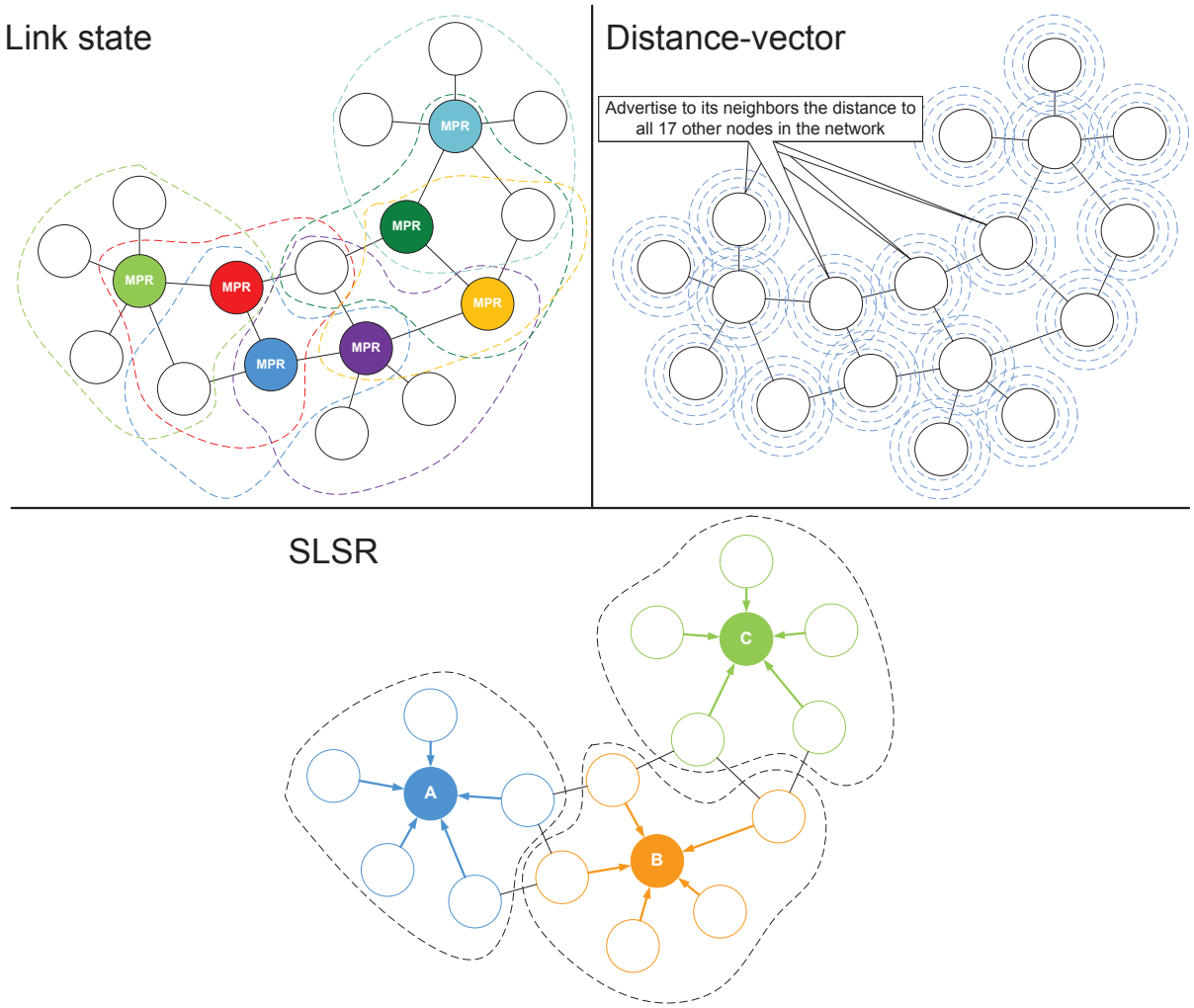


Figure H.6: Comparaison du coût de routage en comparant le recouvrement des informations et la cible des annonces.

1. Choisir la route qui a la métrique *CH metric* la plus petite.
    i. Si identique.
2. Choisir la route qui a la métrique *metric* la plus petite.
    i. Si identique.
3. Choisir la route avec le *expire time* le plus éloigné.
    i. Si identique.
4. Finalement, choisir la route dont l'adresse *Next CH* est la plus petite.

H.5.2 Mise à jour de la table de routage

Les 3 types de messages d'annonces de SLSR (*current slaves*, *lost slaves* et *lost route to clusters*) permettent de remplir et de mettre à jour la table de routage.

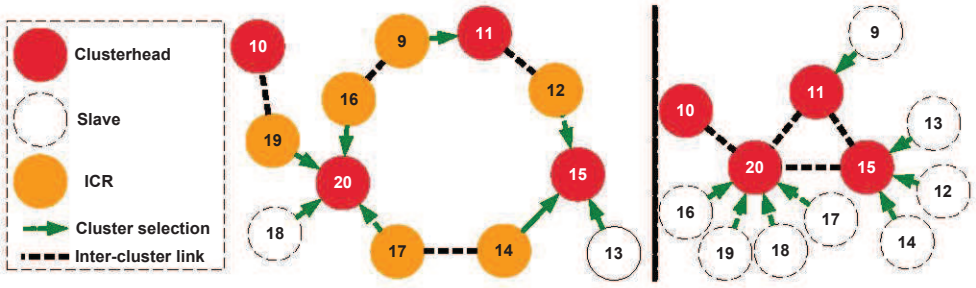


Figure H.7: SLSR - trois clusters avec une boucle. Vue de SLSF (gauche) et vue "simplifié" de SLSR (droite).

#	Dst	Next CH	metric	CH metric	Dst CH addr	Expire time	Seq. Nbr
1	15	15	3	0	15	125.02	225
2	14	15	4	0	15	125.02	225
3	13	15	4	0	15	125.02	225
4	12	15	4	0	15	125.02	225
5	15	11	5	1	15	125.14	225
6	12	11	6	1	15	125.14	225
7	13	11	6	1	15	125.14	225
8	14	11	6	1	15	125.14	225
9	11	11	3	0	11	129.02	31
10	9	11	4	0	11	129.02	31
11	11	15	5	1	11	129.14	31
12	9	15	6	1	11	129.14	31
13	10	10	2	0	10	131.47	12

Figure H.8: Table de routage du clusterhead 20

- **Current slaves**: permet, à lui seul, de remplir les tables de routage. Chaque message *current slaves*, va mettre à jour (ou créer) des entrées pour le clusterhead, et pour chacun des membres annoncés dans le message (le champs *Dst*), comme *Next CH*, le dernier clusterhead par lequel est passé le message. Les entrées de la table de routage sont effacées automatiquement une fois que celles-ci expirent.

- **Lost slaves**: permet d'effacer, immédiatement, les entrées des membres qui ne sont plus associés au clusterhead émetteur du message.

- **Lost route to clusters**: permet d'effacer, immédiatement, les routes (*Next CH*) par lesquelles l'on ne peut plus joindre un clusterhead donné. Cependant, la diffusion de ces messages peut être interrompue, à tout moment, par un clusterhead dont l'annonce ne modifie pas de route vers la destination. Si une information reçue ne modifie pas le *Next CH*, alors rien ne sert de relayer cette information plus loin puisque la direction reste inchangée. Par contre, le nombre de sauts vers la destination peut avoir changé, mais ceci est acceptable puisque qu'il sera mis à jour correctement dès la prochaine réception d'un message *current slaves* de la destination concernée.

Header Type	Fixed part	Variable part		
		multiplication factor	×	size per item
Basic SLSF header	16 bytes	nbr of ICRs	×	4 bytes
Fault Recovery header	4 bytes			
Routed Message header	10 bytes			
Current slaves header	9 bytes	nbr of Slaves	×	4 bytes
Lost slaves header	7 bytes	nbr of lost Slaves	×	4 bytes
Lost route to clusters header	3 bytes	nbr of lost routes	×	5 bytes

Table H.1: Tailles des entêtes SLSF et SLSR (pour des adresses IP de 32 bits).

H.6 Comportement symbiotique

H.6.1 Higher-layer traffic

SLSR profite de la dissémination fournie par SLSF, mais il profite également du trafic réseau venant des niveaux supérieurs. L'idée est d'ajouter les informations de routage dans le trafic des niveaux supérieurs, mais seulement lorsque cela n'affecte pas le message original en lui même. Cet ajout est réalisé au moment où le niveau supérieur confie le message à SLSR. Ce message peut être destiné à être routé vers une destination particulière ou encore à être diffusé dans tout le réseau. Dans le cas de la diffusion, la couche SLSR ne fait que passer le message à SLSF. Cependant, SLSR peut profiter de ce passage pour décider s'il peut insérer un entête, contentant des informations de routage, dans le message. Le critère principal, pour cette insertion, est que l'ajout des entêtes de routage ne doit pas affecter le message original. Le message ne doit pas être scindé en deux suite à un ajout d'entête. Par exemple, si le message avec l'entête de routage vient à dépasser la taille maximale d'un paquet IP, le message devrait alors être réparti sur deux paquets IP distincts. Dans le cas où un entête ne peut être inséré dans un message cet entête sera inséré dans un message suivant, si les délais d'envoi sont flexibles. En dernier recours l'entête devient alors un message à part entière et est envoyé dans le réseau.

Le tableau H.1 donne les différentes tailles des entêtes calculées sur une base d'adresse IP de 32 bits (4 bytes).

H.7 Conclusion

SLSR fourni une couche de routage au dessus de SLSF. Il tire avantage de la structure sous-jacente et permet le routage unicast, la pièce manquante de SLSF. SLSR fait du routage vers le prochain clusterhead, contrairement aux protocoles de routage classiques qui routent vers le prochain voisin. Les routes de SLSR restent inchangées même lorsque les nœuds intermédiaires changent tant que les clusters voisins restent stable.

Pour l'obtention des routes, SLSR utilise des entêtes, qui viennent s'ajouter aux entêtes de SLSF. Ces entêtes peuvent être intégrés de manière symbiotique dans les messages/données venant d'un niveau supérieur. SLSR réduit ainsi son empreinte réseau, en diminuant le coût d'acheminement des messages au travers de l'agrégation d'informations.

De plus, on peut classer SLSR parmi les protocoles dit "balanced-hybrid" qui emploient les caractéristiques des algorithmes à vecteur de distance combinées à ceux à état de lien. La réduction des coûts de routage vient aussi du fait que les informations diffusées, pour l'annonce des routes, par SLSR ne se recouvrent pas. Seuls les clusterheads émettent des annonces et celles-ci concernent uniquement leur propre cluster. Chaque nœud n'est alors annoncé qu'une

seule fois, l'annonce de son clusterhead.

Appendix I

Découverte de services dans les réseaux ad hoc utilisant Zeroconf et SLSR

Contents

I.1 Pourquoi Zeroconf? 208
I.2 Zeroconf sur SLSF 208
I.3 *Context-awareness* pour la découverte de services 210
I.3.1 Réseau 210
I.3.2 Nœud/Appareil 213
I.3.3 Espace 213
I.3.4 Service 214
I.4 Filtrage collaboratif pour la découverte de services 214
I.5 Architecture de découverte de services 215
I.5.1 Scénario 215
I.5.2 Modélisation du scénario en utilisant les systèmes multi-agent et la co-simulation 215
I.6 Cas d'étude 216
I.7 Expérimentations 218
I.7.1 OLSR 218
I.7.2 Zeroconf au dessus de SLSR 219
I.7.3 Zeroconf, SLSF et SLSR 219
I.8 Conclusion 220

Ce chapitre présente les contributions concernant la découverte de services. Il décrit comment chacun des chapitres précédents contribue à la découverte de services. Nous avons choisi comme protocole de découverte de services le protocole Zeroconf. Nous présentons les avantages qu'apporte Zeroconf dans notre contexte et, aussi, comment SLSF et SLSR contribuent à l'adaptation de Zeroconf dans le milieu des réseaux ad hoc. Ensuite nous décrivons les premières notions d'un protocole "context aware". De plus, nous proposons d'étendre la prise en compte du contexte en utilisant les techniques du filtrage collaboratif afin de choisir les meilleurs services. Enfin, nous présentons l'architecture de découverte de services, proposée dans le cadre du projet ANR[49] SARAH[50].

[49] Agence Nationale de Recherche
[50] Services Avancés pour Réseaux Ad hoc

I.1 Pourquoi Zeroconf?

Les raisons du choix de Zeroconf et pourquoi nous pensons qu'il a beaucoup d'avantages pour les réseaux ad hoc sont les suivantes:

- **Il est basé sur DNS:** L'approche de Zeroconf est d'utiliser un standard existant largement reconnu tel que l'est le DNS. Les propriétés du DNS fournissent déjà un grand nombre de fonctionnalités utiles et requises pour la découverte de services, tel que décris dans [Ches 05]:
  - La découverte de services requiert un mécanisme d'agrégation (un serveur ou des nœuds répertoires).
    Les serveurs et mécanismes d'agrégation existent déjà dans DNS.
  - La découverte de services requiert un protocole d'enregistrement des services.
    DNS possède les mises à jour dynamiques (Dynamic Updates).
  - La découverte de services requiert un protocole de requêtes.
    DNS a déjà un protocole de requêtes.
  - La découverte de service requiert un moyen de sécurisation.
    DNS propose le DNSSEC.
- **Largement déployé:** Utiliser un standard très largement déployé tel que DNS, fait que Zeroconf est, du coup, lui aussi largement déployé au sein des réseaux filaires. La compatibilité DNS des appareils permet une comptabilité aussi pour Zeroconf. Pour cela, Zeroconf nous parait être parfaitement adapté aux réseaux ad hoc.
- **Capacité à profiter d'une infrastructure:** Un autre critère important en faveur de Zeroconf, est sa capacité à utiliser une infrastructure existante lorsque celle-ci est présente. Zeroconf peut utiliser un serveur DNS de manière transparente pour ce dernier. En plus de cette capacité, l'avantage d'utiliser les serveurs DNS est que ceux-ci sont souvent déjà déployés dans les réseaux filaires. Il suffit alors pour un réseau ad hoc d'avoir une connexion à un serveur DNS local pour profiter des avantages d'une infrastructure, sans toutefois en être dépendant.
- **Très peu de changements nécessaires pour être adapté aux réseaux ad hoc:** Pour adapter Zeroconf aux réseaux ad hoc, seule la structure pour la diffusion des messages de Zeroconf doit être adaptée aux réseaux ad hoc.
- **Extensible:** Zeroconf permet facilement l'ajout de nouvelles fonctionnalités (tel que dans les champs des entrées DNS TXT) ou encore au niveau du protocole lui-même, en adaptant les requêtes et les réponses au contexte (par exemple, en supprimant des réponses connues ou inadaptées au contexte). En plus des serveurs DNS, la mise en place de nœuds répertoires est aussi possible. Un nœud Zeroconf classique peut très bien interagir avec un nœud qui, lui, est capable de prendre en compte le contexte dans son fonctionnement.

I.2 Zeroconf sur SLSF

Zeroconf repose sur le multicast IP pour disséminer les messages dans le réseau. En pratique, il s'agit simplement de l'envoi d'un message avec en tant qu'adresse de destinataire l'adresse du groupe multicast IP réservée pour Zeroconf auprès de l'IANA (adresse multicast 224.0.0.251 pour IPv4 et ff02::fb pour IPv6). Dans un réseau filaire la diffusion de ces messages est faite

automatiquement par les switch dans le même sous réseau (comme pour les messages broadcast). Pour passer d'un sous-réseau à l'autre, les routeurs construisent un arbre multicast entre eux pour diffuser les messages dans tous les sous-réseaux ayant des nœuds intéressés par ces messages multicast. Les différentes techniques pour la constructions de ces arbres multicast sont décrites dans [Saha 02].

Cependant, dans un réseau ad hoc, il n'y a pas de switch ou de routeur qui relaye les messages dans le réseau. Les messages multicast doivent alors être relayés par les nœuds eux même, tout comme c'est le cas pour les messages broadcast.

Nous proposons de remplacer le multicast IP par notre protocole de dissémination SLSF, dont voici les avantages:

- Zeroconf utilise le multicast pour la dissémination efficace d'informations vers d'autres nœuds Zeroconf: SLSF fourni cette diffusion efficace et réduit le nombre de nœuds relais nécessaires.

- La gestion de groupe multicast n'est pas nécessaire dans notre contexte, tous les nœuds du réseau sont intéressés par les messages de Zeroconf. La propriété du multicast de restreindre la diffusion à un groupe donné n'est donc pas utilisée dans notre cas. Cependant, si cela devenais nécessaire, SLSF pourrait même gérer les abonnements et appartenances aux groupes multicast au niveau des clusterheads en échangeant des messages d'abonnement de groupes.

- SLSF, de plus, propose des avantages tels que la stabilité, la prise en compte du contexte et le passage à l'échelle.

Comme l'on peut le voir sur la figure I.1 Zeroconf reste inchangé. Seule la diffusion multicast est remplacée par une diffusion SLSF de manière transparente pour Zeroconf.

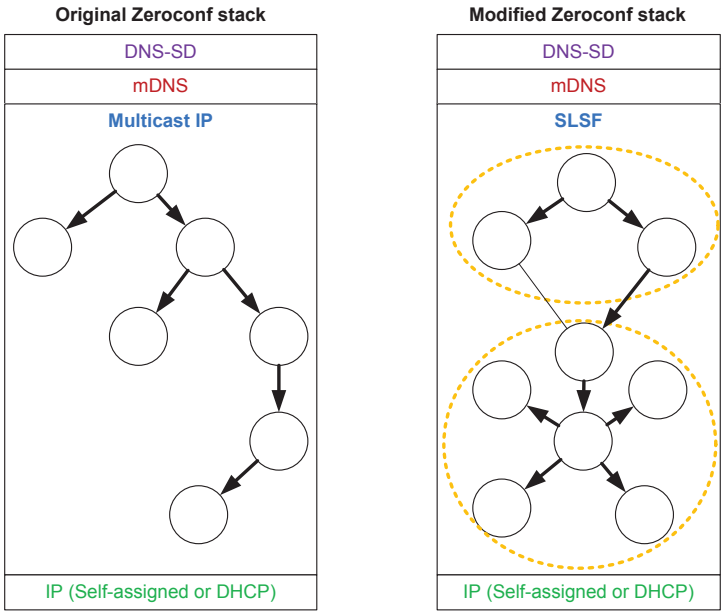


Figure I.1: Pile de protocoles de Zeroconf avec le multicast IP et avec SLSF.

I.3 *Context-awareness* pour la découverte de services

Pour pouvoir trouver plus que les services correspondants simplement à une requête, cette section présente comment et où le contexte peut intervenir pour améliorer la découverte de services. Dans notre architecture utilisant Zeroconf, le contexte peur intervenir à deux niveaux:

- au niveau des entrées DNS: Les entrées TXT associées aux entrées SRV, peuvent contenir des données de contexte. L'exemple sur la figure I.2 montre une entrée TXT d'un service proposant une passerelle Internet pour appareils à faibles capacités en adaptant le contenu des pages Internet à l'appareil. Cette entrée TXT contient deux champs de données: Le champ *served time*, représente le temps total de service fourni et le champ *isFree*, représente un booléen indiquant par sa présence que le service est gratuit.

  Un nœud recherchant un service peut alors affiner sa recherche et son choix en utilisant les informations supplémentaires contenues dans les champs de données des entrées TXT.

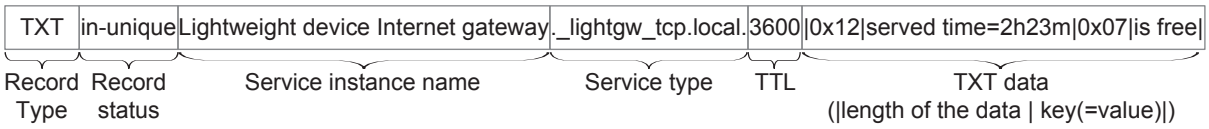


Figure I.2: Entrée TXT contentant des informations de contexte.

—

- au niveau de la dissémination: Les nœuds intermédiaire/répertoire peuvent filtrer les messages, ou des parties des messages, en transit contenant des informations considérées comme superflues d'après le contexte courant. En tirant avantage de la structure de cluster, les clusterheads peuvent diminuer et alléger le choix pour leurs membres. Si dans un réseau il y a beaucoup de services du même type, les clusterheads peuvent filtrer les services et ne proposer à leurs membres que les meilleurs services parmi la multitude de services annoncés.

En utilisant le contexte, une requête peut ainsi être écartée en amont par les nœuds intermédiaires, permettant ainsi que seules les informations pertinentes soient diffusées aux nœuds. De plus, chaque nœud peut affiner son choix de service en utilisant les informations de contexte contenues dans les entrées TXT.

Les **métriques** permettent l'acquisition du contexte. Nous classons les métriques que nous considérons pour caractériser le contexte pour la découverte de services et les réseaux ad hoc en 4 catégories [Lecl 07]:

I.3.1 Réseau

Métriques liées au réseau qui peuvent être calculées ou obtenues directement à partir des messages échangés.

Nombre de sauts: Nombre de nœuds intermédiaires nécessaires pour atteindre une destination. Appelée aussi nombre de hop ou distance en hop, cette métrique est souvent utilisée car elle est facile à obtenir et permet de calculer le chemin le plus court vers une destination.

Débit: Le débit est la quantité d'informations transmises via un canal de communication dans un intervalle de temps donné. Le débit d'une connexion Internet s'exprime généralement en kbps (kilobit par seconde). L'estimation du débit permet de réguler le flux des paquets émis afin d'éviter ou de limiter la congestion du réseau.

Bande passante: La bande passante est la capacité maximale de débit sur une liaison donnée, déterminée par les technologies de transmission mises en œuvre à l'aide des équipements situés à chaque extrémité de cette liaison. C'est la valeur maximale (théorique) de débit possible. Cette métrique est le plus souvent utilisée à des fins de gestion du réseau.

Latence: La latence est le temps écoulé entre l'instant où l'unité de commande déclenche l'envoi et le moment ou les données sont traitées par la destination. La latence est la somme des temps de traitement et de transfert d'un message de l'émetteur jusqu'au destinataire. La mesure de la latence requiert un accès aux deux parties, l'émetteur et le destinataire.

Gigue: La gigue est la fluctuation de courte durée d'un signal. Elle est la dispersion temporelle ou le glissement de phase qui se produit sur une ligne de transmission d'informations à la suite de l'utilisation de répéteurs ou de régénérateurs. Elle peut causer des erreurs de transmissions, particulièrement à grande vitesse, tel que le dé-ordonnancement des paquets.

Round Trip Time: RTT ou encore temps de rotation est le temps que met un paquet pour aller de la source vers la destination et revenir. L'avantage du RTT par rapport à la latence est qu'il ne requiert, pour son calcul, qu'un accès à l'émetteur du paquet. Typiquement, le RTT est mesuré en utilisant un paquet ICMP[51] avec lequel on mesure la différence entre le temps de départ et l'arrivée de sa réponse.

Expected Transmission Count: ETX[De C 04] est une métrique qui définit le nombre de retransmissions estimé pour envoyer un paquet d'une source vers sa destination. La meilleure valeur pour ETX est 1 (une seule transmission pour atteindre la destination) et la pire valeur est "infini". ETX est utilisé lors des transmissions multi-sauts dans les réseaux sans fil pour déterminer le chemin le moins erroné. ETX est notamment implanté dans le protocole OLSR.

Situation: La situation d'un nœud peut être caractérisée par les mesures de centralité. La centralité donne l'importance structurelle d'un nœud dans un graphe, Ici le graphe représente les connexions réseau entre les nœuds. Un nœud est dit " central " lorsque l'on considère qu'il a une forte influence sur les autres nœuds. La centralité donne la position " sociale " du nœud dans le réseau.

Nous considérons trois centralités différentes: le degré, la proximité et l'intermédiarité. Parmi les centralités existantes, ce sont celles que nous considérons comme les plus abordables en termes d'informations nécessaires et de complexité d'acquisition.

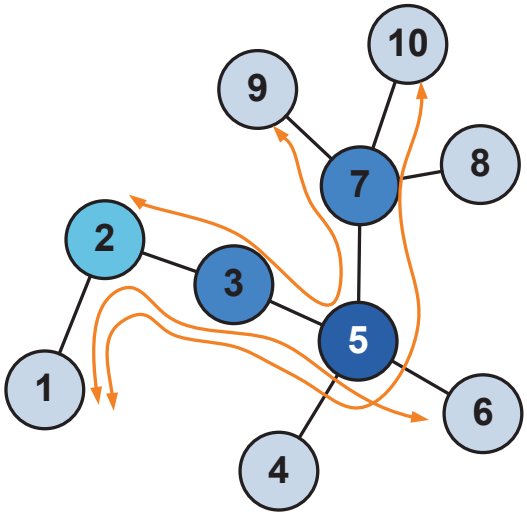
- **Degré:** On considère que plus un nœud est directement lié à un grand nombre de nœuds plus il est important. La mesure de cette métrique se fait simplement en comptant le nombre de connexions directes d'un nœud avec les autres nœuds, à savoir le nombre de voisins directs. Cette information est très facile à obtenir puisque la plupart des protocoles de routage nécessitent la liste des voisins.

[51] Internet Control Message Protocol

- **Proximité:** On considère que plus un nœud est proche des autres nœuds plus il est important. La mesure se fait en calculant la valeur inverse de la somme des longueurs des plus courts chemins vers un nœud du graphe ou, en d'autres termes, la valeur inverse du nombre de sauts séparant un nœud de tous les autres nœuds du réseau. Comme pour le degré, cette valeur est souvent nécessaire aux protocoles de routage.

- **Intermédiarité:** On considère que plus un nœud sert d'intermédiaire pour d'autres nœuds plus il est important. On mesure la somme des ratios des plus courts chemins entre deux nœuds qui passent par le nœud dont on calcule l'intermédiarité.

  La mesure réelle de cette métrique est difficile, car il faut soit connaître toute la topologie du réseau, soit la calculer à partir des messages échangés. Si l'on peut considérer que le réseau ne comporte pas de boucle, alors on peut simplifier le calcul et calculer l'intermédiarité à partir d'une table de routage comportant seulement le nombre de sauts qui séparent le nœud de tous les autres. Pour le nœud *5* de la figure I.3 on calcule l'intermédiarité en prenant le nombre de voisins sur chacune des 4 branches (par exemple la branche [*1,2,3*], [*4*], [*6*] et [*7,8,9,10*])) et en le multipliant avec le nombre de voisins cumulés des autres branches. Pour la branche *1,2,3* on multiplie 3 fois 6 (*4,6,7,8,9,10*). En cumulant le résultat pour chaque branche on obtient $18+8+8+20=54$. Pour avoir l'intermédiarité, qui est réciproque, il suffit de diviser par 2. L'intermédiarité du nœud *5* est 27.



node	Intermediarity
5	27
7	21
3	14
2	8
1	0
4	0
6	0
8	0
9	0
10	0

Figure I.3: Intermédiarité

Stabilité des liens: La stabilité d'un lien peut être calculée à partir de métriques individuelles, ou d'une combinaison tels que: la puissance du signal, la position GPS, la vitesse, la durée de vie du lien, le nombre de beacons reçus (représente une information de présence, d'affinité ou encore associativité), etc. [Pari 09].

Lost Routed Packet: LRP [Bado 05] est une combinaison de deux métriques: Received Routed Packets (RRP), le nombre de paquets à relayer reçus et Send Routed Packets (SRP), le nombre de paquets à relayer envoyés. Ainsi on obtient le LRP = RRP − SRP : le nombre de paquets à relayer qui n'ont pas été relayés. Ceci caractérise la participation et la contribution d'un nœud dans le réseau.

I.3.2 Nœud/Appareil

Métriques qui concernent le nœud lui-même. L'obtention de ces métriques est simple puisqu'elles sont disponibles localement.

Puissance de calcul: La puissance de calcul permet d'estimer la capacité d'un nœud à traiter les messages ou encore à héberger un service gourmand en ressources.

Mémoire: Deux types : Mémoire restante et mémoire initiale. Plus de mémoire, signifie plus de capacité à traiter de large quantité d'informations (ex: table de routage, répertoire de services).

Puissance du signal: La puissance du signal peut par exemple indiquer que le nœud est atteignable en émission direct et donner une mesure de la qualité de la communication.

État de charge: On entend par état de charge, l'état dans lequel la batterie se trouve: "en charge", "en utilisation", "pas de batterie". Cette métrique peut informer sur la stabilité (puissance et mobilité) d'un nœud. Ainsi, un nœud "en charge" aura tendance à ne pas bouger. Il peut éventuellement bouger, mais à des petites distances et il n'aura pas de problème d'énergie. Par contre, un nœud "en utilisation" aura beaucoup plus tendance à bouger et, en plus, un risque de manque d'énergie. Enfin, "pas de batterie" signifie que c'est un nœud fixe non mobile et à énergie, à priori, infinie.

Accélération et mouvement des nœuds: De plus en plus d'appareils (e.g. smartphones) sont équipés de capteurs de mouvements. Nous proposons d'utiliser les informations de mouvements de l'appareil pour estimer la stabilité et l'activité de l'utilisateur. Nous proposons 3 états:

1. Aucun mouvement détecté: l'appareil est stable et est sans doute géographiquement stable.
2. De fort mouvements détectés: L'utilisateur est en mouvement et sans doute pas en train de manipuler ou utiliser l'appareil (e.g. il marche et a l'appareil dans sa poche).
3. Mouvements modérés: L'utilisateur manipule l'appareil, ce qui diminue ses mouvements et même parfois sa vitesse de déplacement pour pouvoir par exemple lire l'écran.

Applications: Les applications peuvent fournir de nombreuses informations intéressantes pour le contexte.

I.3.3 Espace

Métriques qui concernent l'espace/l'environnement dans lequel le nœud est. Ces métriques sont souvent difficiles à obtenir car elles proviennent du monde physique.

Géo-localisation: Position géographique du nœud.

Mobilité relative: La mobilité d'un nœud par rapport aux nœuds qui l'entourent.

Vitesse: Vitesse de mouvement du nœud, elle peut-être obtenue en utilisant un GPS ou encore un podomètre.

Fréquence/Tendance au mouvement: Cette métrique établie un profil de mouvement du nœud afin de prédire les prochains mouvements.

Direction cardinal: Fournie par une boussole (souvent présente sur les smartphones de nos jours), la direction cardinale permet de donner une estimation grossière de la prochaine position. Par exemple, on peut essayer de prédire le prochain cluster vers lequel un nœud se dirige.

I.3.4 Service

Métrique concernant les services du réseau. Ils ont pour but d'améliorer la découverte de services ou leur fonctionnement.

Proposition de services: Nombre de services qu'un nœud propose.

Connaissance de services: Nombre de services qu'un nœud connaît. Un nœud répertoire est un nœud qui connaît beaucoup de services ou à l'inverse si un nœud connaît beaucoup de services, il est judicieux de le choisir comme nœud répertoire.

Contexte de découverte de services: Contexte dans lequel un service a été découvert. Composé de:

- Type de découverte: Active ou passive.
- Temps depuis la découverte: Temps depuis lequel le service a été découvert.
- Distance du service: Distance, en nombre de saut, depuis le service.
- Utilisation du service: Donne un profil d'utilisation du service tel que la durée d'utilisation du service ou encore la fréquence d'utilisation du service.

I.4 Filtrage collaboratif pour la découverte de services

Le filtrage collaboratif (*collaborative filtering*) a pour but d'éviter la surcharge d'information. Pour cela il utilise les avis et les notes d'utilisateurs, ayant un profil similaire, pour filtrer les informations et ne garder que celles qui sont pertinentes. Appliqué à la découverte de services, le filtrage collaboratif, va permettre de réduire la liste des services affichés/reçus au niveau d'un nœud en ne gardant que les services qui sont les plus pertinents pour lui. Les utilisateurs ont des profils exprimant leurs préférences et leur historique. Nous avons proposé dans [Grat 08] un protocole de filtrage collaboratif qui repose sur la structure de cluster de NLWCA. Dans ce protocole, les profils ne sont échangés que parmi les membres d'un même cluster. Ce sont les mouvements et les migrations des nœuds d'un cluster à l'autre qui permettent la circulation des informations. Dans [Grat 08], nous avons évalué l'algorithme sur la prédiction et suggestion de film, dont les notations proviennent de la base de données MovieLens[1] qui contient 100.000 notations de 1682 films faites par 943 utilisateurs, où chaque utilisateur a noté au moins 20 films. Les résultats sont très encourageants et montrent que l'algorithme donne de bonnes prédictions. L'algorithme proposé à l'avantage d'utiliser la même structure de cluster que SLSF/SLSR, il est ainsi immédiatement compatible et intégrable à l'architecture de découverte de services.

[1] http://www.grouplens.org

I.5 Architecture de découverte de services

Cette section décrit l'architecture de découverte de services que nous proposons pour permettre une découverte de services contextuelle dans un scénario hybride mêlant réseau ad hoc et infrastructure fixe. L'architecture repose sur le réseau ad hoc et tire avantage de l'infrastructure fixe lorsque celle-ci est disponible. L'architecture, au travers des protocoles tel que SLSF, SLSR ou encore Zeroconf, permet d'intégrer et de prendre en compte le contexte à différents niveaux.

I.5.1 Scénario

Le projet SARAH décrit un scénario de démonstrateur basé sur une visite dans un musée. Chaque visiteur participant à cette expérience reçoit un appareil, de type PDA. Les visiteurs peuvent alors suivre une visite guidée ou visiter librement le musée. Le PDA leur fournit leur localisation géographique à l'intérieur du musée ainsi que des informations multimédia complémentaires sur les œuvres des environs. De plus, des questionnaires géo-localisés sont proposés aux visiteurs, les réponses se trouvant sur les œuvres environnantes. La découverte de services intervient dans ce scénario pour la découverte automatique, et la récupération depuis le réseau, des informations supplémentaires des œuvres environnantes.

L'architecture de découverte de services sur la figure I.4 montre le cœur du réseau composé de nœuds ad hoc et d'une architecture statique composée de points d'accès sans fil et de serveurs. Certains nœuds du réseau ad hoc ont aussi une connexion à un des points d'accès de l'infrastructure fixe.

Au niveau de l'implantation il y a deux possibilités.

- Les nœuds ayant les deux connexions, ad hoc et infrastructure fixe, jouent le rôle d'intermédiaire et de relai. Ils font également les éventuelles adaptations nécessaires (e.g. adapter les messages) pour faire la transition du réseau ad hoc vers le réseau d'infrastructure fixe.

- Les points d'accès ont la même pile de protocoles que les nœuds du réseau ad hoc et apparaissent alors comme des nœuds ad hoc normaux. Ils sont alors en charge de faire les éventuelles adaptations protocolaires nécessaires.

Le dernier choix, sur les points d'accès, est le meilleur, mais peut ne pas être possible à cause de limitations d'accès au matériel ou encore à des incompatibilités logicielles.

Dans tous les cas, la présence de l'infrastructure fixe peut être simplement signalée et découverte en utilisant la découverte de services. L'infrastructure apparaît alors comme un service supplémentaire.

On peut aussi considérer un scénario comportant, en plus, un réseau mesh. La figure I.5 montre l'architecture avec une partie du réseau qui est connecté par réseau mesh. Tout comme pour les points d'accès, les nœuds mesh peuvent utiliser la même pile de protocole que celle utilisée dans le réseau ad hoc et ainsi interagir de manière transparente avec les nœuds ad hoc.

I.5.2 Modélisation du scénario en utilisant les systèmes multi-agent et la co-simulation

Pour modéliser notre scénario de visite de musée dans un simulateur, nous avons utilisé l'architecture AA4MM (Agent and Artefact for Multiple Models [Sieb 10]). Cette architecture permet le couplage de différents simulateurs et modèles venant de domaines différents. Dans notre cas, nous couplons un simulateur réseau avec un simulateur de mobilité de personnes.

Le choix de séparer la simulation réseau de la simulation de la mobilité, vient du constat que les réseaux de communications et leur simulation est un domaine très éloigné de la mobilité des humains et de sa simulation.

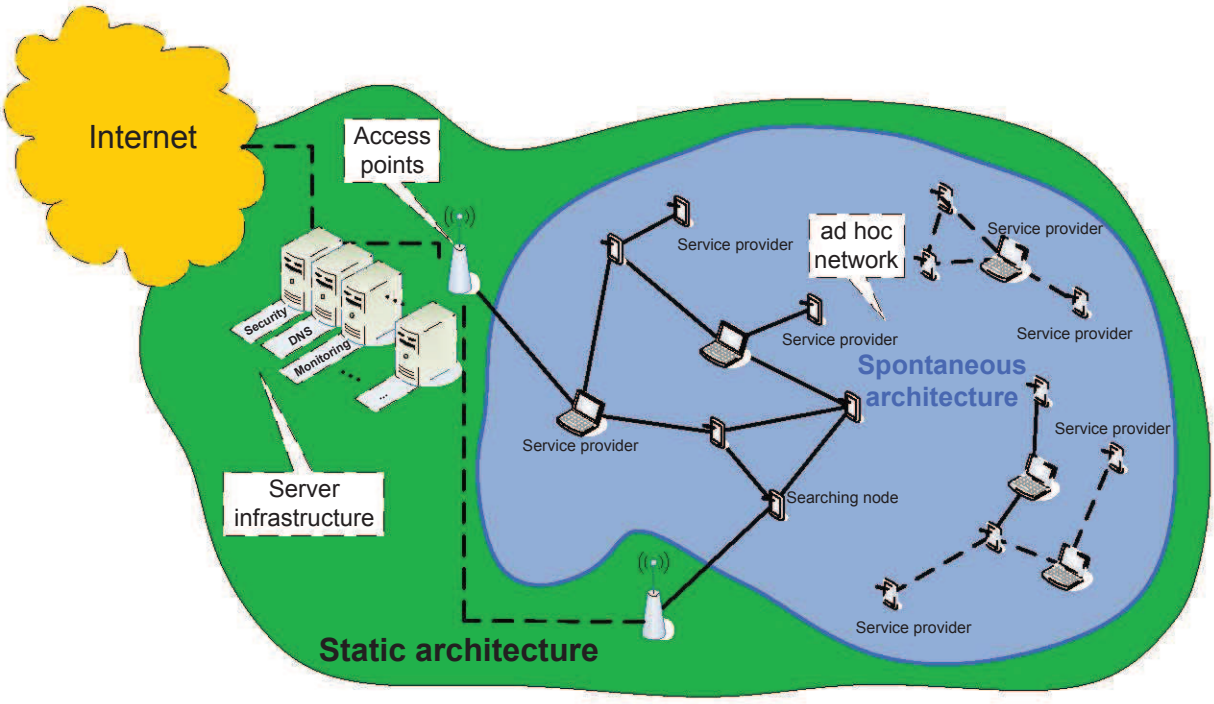


Figure I.4: Architecture de découverte de services.

I.6 Cas d'étude

Dans les domaines des réseaux ad hoc, bien souvent on fait appel à la simulation pour évaluer les protocoles. En effet, les expérimentations avec des appareils réels, avec un nombre représentatif d'appareils, sont très difficiles à mettre en place pour des raisons de coût, de temps de mise en œuvre et aussi d'analyse des résultats. Les expériences sont difficilement reproductibles puisque les résultats sont souvent influencés par des évènements extérieurs (p. ex. interférences des ondes sans fil) ou encore le facteur humain.

De nombreux simulateurs et modèles ont été développés dans le domaines des réseaux dynamiques [Naic 06, Kurk 05]. Le but d'un simulateur est de simuler le réseau et ses couches de manière plus ou moins détaillée selon les besoins. Pour simuler des réseaux ad hoc les simulateurs ont aussi des modèles de mobilité. Ceux-ci sont bien souvent très basiques, car ce n'est pas le but principal. Pourtant, un facteur primordial pour la mobilité et pour le comportement d'un nœud dans un réseau ad hoc est le facteur humain.

Nous proposons de coupler différents simulateurs et modèles venant de domaines différents en utilisant l'architecture AA4MM (Agent and Artefact for Multiple Models [Sieb 10]). Ainsi, nous prenons l'outil qui est le plus adapté et reconnu pour chacun des domaines. De plus, les efforts de recherche d'un domaine peuvent ainsi être mis à contribution facilement pour d'autres domaines. Dans notre cas, nous combinons un simulateur réseau avec un simulateur de mobilité multi-agent. Ceci réuni la recherche du domaine des réseaux avec la recherche du domaine sociologique sur les mouvements de piétons dans des environnements urbains.

AA4MM permet de réunir ces deux simulateurs, tout en gérant la synchronisation, de manière décentralisée, des informations échangées entre les simulateurs. Pour aller plus loin, le comportement (et la mobilité) des utilisateurs, peuvent être influencés par un retour sur la qualité du réseau ou d'autres évènements provenant du réseau. Avec notre solution, nous pouvons facile-

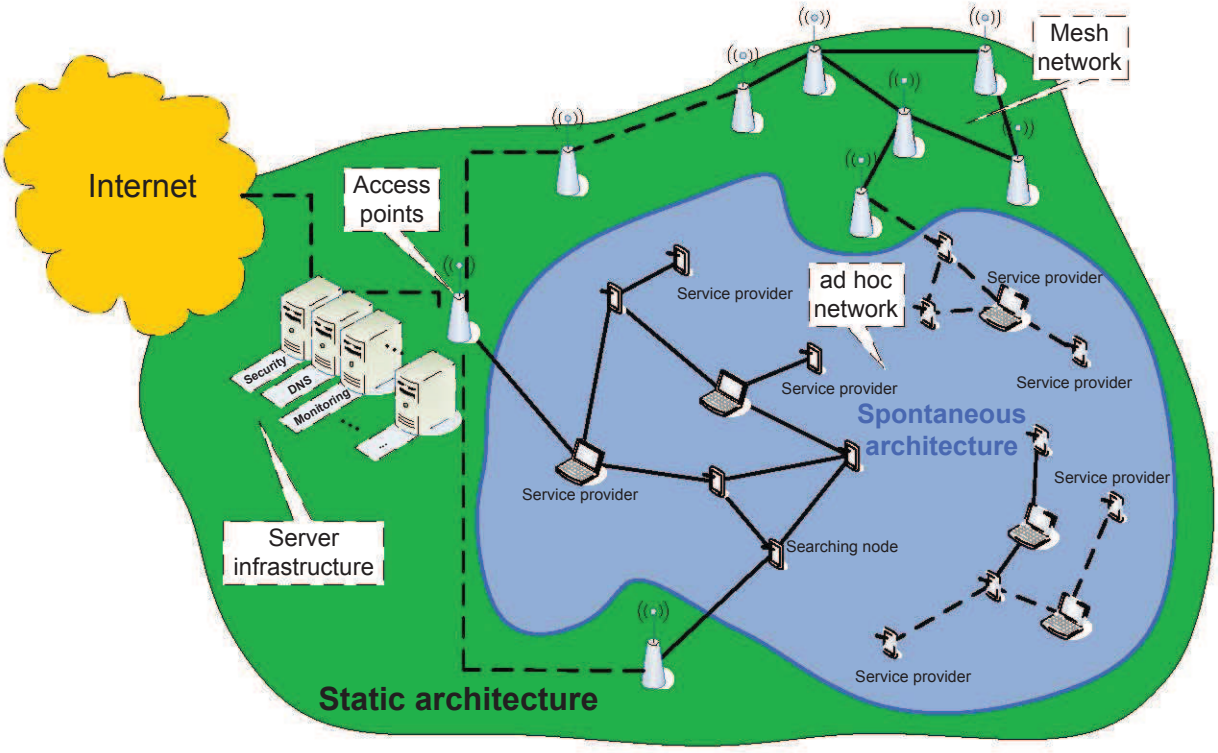


Figure I.5: Architecture de découverte de services avec réseau mesh.

ment réaliser un comportement qui dépend d'informations provenant de l'extérieur, tel que le simulateur réseau. Puisque la mobilité fait, elle aussi. varier les performances du réseau, une influence mutuelle entre le réseau et la mobilité est créée (figure I.6).

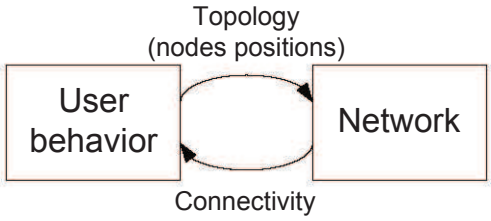


Figure I.6: Interactions entre le simulateur réseau et le simulateur de mobilité.

Nous avons utilisé l'architecture AA4MM pour modéliser une version simplifiée du scénario du projet SARAH. Notre premier modèle comporte 4 groupes de nœuds différents, initialement positionnés aléatoirement, dans un environnement proche de celui d'un musée. Chaque groupe a un point d'arrivée final (but géographique) différent. Le comportement, calculé par le simulateur de mobilité mutli-agent, est le résultat de la combinaison de plusieurs comportements simples:

- Chaque nœud se dirige vers un but géographique qui est différent selon le groupe auquel il appartient.
- Chaque nœud évite les murs en appliquant une force répulsive qui va le faire s'éloigner du mur.

- Les nœuds exercent aussi des forces répulsives les uns sur les autres, créant ainsi une " bulle " de distance entre chaque nœud.

Le résultat de la combinaison de ces trois comportements est présenté sur la figure I.8. On y voit l'évolution des mouvements des nœuds en fonction de leur couleur/groupe auquel ils appartiennent. Le but de chacun des groupes se situe dans un coin différent.

Dans un autre scénario nous avons introduit un retour d'information de la part du réseau qui influence le comportement. Ici, le comportement des nœuds va dépendre, en plus des comportements du scénario précédent, de la connectivité du nœud. Le scénario, sur la figure I.7, est celui d'une visite de musée, où un guide (le nœud 1) est suivi par des visiteurs. Le guide possède ses propres buts géographiques, définis à l'avance, qui représentent le parcours de la visite. Les visiteurs eux ont pour but de garder une connexion au niveau du réseau ad hoc avec le guide. Ainsi, lorsque le guide avance, la connexion avec celui-ci sera coupée et les nœuds vont alors le suivre jusqu'à être à nouveau connectés. Si le guide s'arrête les nœuds, de nouveau connectés, ne bougent plus.

I.7 Expérimentations

Lors de ma thèse, j'ai effectué plusieurs expériences avec des vrais appareils. Ces expériences ont un nombre limité de nœuds et sont surtout à considérer comme des preuves de concept.

Pour exécuter les protocoles sur des appareils réels, nous avons utilisé le mode *platform* de JANE qui permet d'exécuter directement le code de simulation sur les appareils. Le seul prérequis est la présence d'une machine virtuelle Java sur les appareils [52].

I.7.1 OLSR

Les expériences ont comporté au maximum 5 appareils. La première expérimentation fût réalisée avec le protocole OLSR. Nous avons ainsi obtenu un réseau de plusieurs sauts utilisant OLSR dans le bâtiment de l'INRIA à Nancy. Pour augmenter le nombre de sauts et d'appareils participant, nous avons, en plus de 3 Nokia N800 (appareils portatifs proche d'un smartphone), utilisé un laptop. Nous avons obtenu un réseau de 4 sauts, mais bien souvent, les connexions étaient momentanément coupées à cause des interférences des réseaux sans fils environnants.

[52] Java n'est pas supporté officiellement sur les Nokia N800 (appareils utilisés pour les expériences. Nous avons donc utilisé la version Java du projet Jalimo `https://wiki.evolvis.org/jalimo/index.php/Hauptseite`

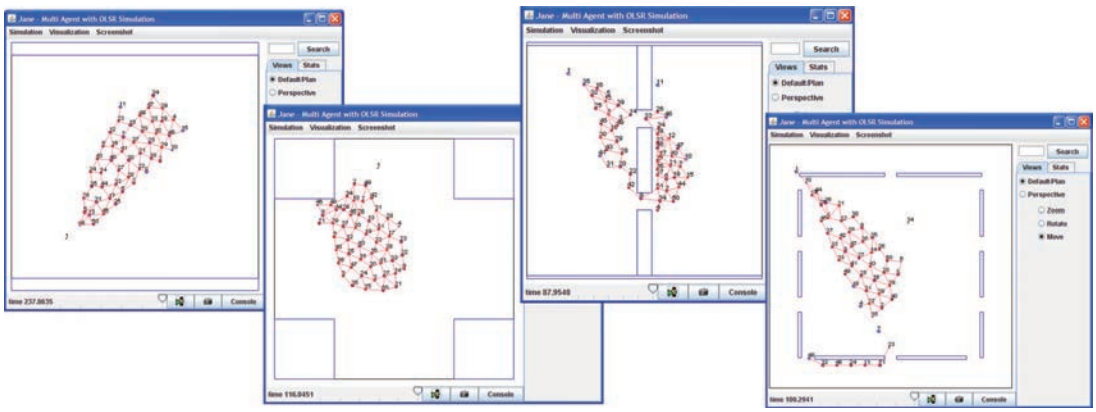


Figure I.7: Scénario de visite de musée dans différents environnements.

I.7.2 Zeroconf au dessus de SLSR

Nous avons également effectué des expériences de Zeroconf au-dessus de SLSR. Les expériences comportent 3 Nokia N800. Sur la figure I.9 on peut voir un nœud nommé " Bob " qui propose un service nommé " serviceBob " en utilisant Zeroconf. La figure montre la table DNS contenant les noms et les services *http* " serviceBob " découvert par Alice.

Une autre expérimentation avec les Nokia N800, plus récente, est présentée sur les figures I.10, I.11, I.12 et I.13. On y voit les interfaces graphiques des différentes informations visibles sur les appareils:

- La table DNS, figure I.10.
- Le détail des entrées DNS, figure I.11.
- Les informations concernant NLWCA et SLSF, figure I.12.
- L'interface pour interagir et envoyer des messages vers d'autres nœuds du réseau, figure I.13.

I.7.3 Zeroconf, SLSF et SLSR

Un exemple d'expérience avec les 3 protocoles, SLSF, SLSR et Zeroconf, est présenté sur la figure I.14. Il s'agit d'une capture de simulation, afin de permettre de montrer un exemple avec 20 nœuds. Les clusterheads sont marqués en rouge et les nœuds ICR en orange. Dans ce réseau, les nœuds *3*, *9* et *18* proposent un service *http* et les nœuds *6*, *7* et *15* proposent un service *vidéo*.

Nous introduisons ici un premier exemple de métrique associée aux services. Cette métrique représente une note fictive de qualité du service et est située dans l'entrée *TXT* des entrées DNS. L'exemple suivant montre les entrées DNS pour le service *http* nommé "web-15" annoncé par le nœud "myDevice15.local":

```
record[srv,in-unique,web-15._http._tcp.local.,120000/103165,myDevice15.local.:888]
record[txt,in-unique,web-15._http._tcp.local.,120000/103165, metric=10]
record[ptr,in,_http._tcp.local.,120000/100165,web-15._http._tcp.local.]
```

La figure I.15 montre la table DNS et la table de routage de SLSR du nœud clusterhead *2*[53]. La table DNS affiche tous les services (ici, 6 services) disponibles dans le réseau ainsi que les appareils, et leurs noms DNS. Si l'on compare avec un nœud non-clusterhead, sur la figure I.16, le nœud *1* lui n'a pas la liste complète des services du réseau. Son clusterhead, le CH 2, a filtré les services et ne lui a présenté que les services qu'il a considérés comme les meilleurs. Les métriques contenues dans les entrées TXT des différents services sont les suivantes:

- web-15 valeur de la métrique = 10.
- web-7 valeur de la métrique = 26.
- web-6 valeur de la métrique = 85.

Dans notre exemple, le service *http* "web-6" apparaît dans la liste. Il est le meilleur (avec la plus haute valeur) des services de type *http* parmi ceux proposés.

[53] La table de routage des clusterheads ne contient pas les nœuds membre du cluster. Ainsi, la table de routage du CH 2 ne contient que 18 entrées sur les 20 nœuds du réseau.

I.8 Conclusion

Dans ce chapitre nous avons montré comment chacune des contributions précédentes participe à l'architecture de découverte de services. Nous avons détaillé les raisons du choix de Zeroconf comme protocole de découverte de services pour les réseaux ad hoc et avons remplacé la structure multicast par notre protocole SLSF/SLSR.

Les métriques proposées et différents points d'implantations d'informations de contexte vont permettre une découverte de services contextuelle. De plus, on se dirige, avec le filtrage collaboratif, vers un contexte social de l'utilisateur. Nous avons proposé une architecture de découverte de services hybride, où le réseau ad hoc et les protocoles sont capables de tirer avantage de la présence d'une infrastructure, tout en restant indépendants de celle-ci.

Nous avons décrit les outils capables d'intégrer le facteur humain, qui vont nous permettre d'évaluer et de modéliser les scénarios relatifs à la découverte de services.Enfin, nous avons présenté plusieurs expériences avec des vrais appareils qui démontrent la faisabilité des protocoles proposés.

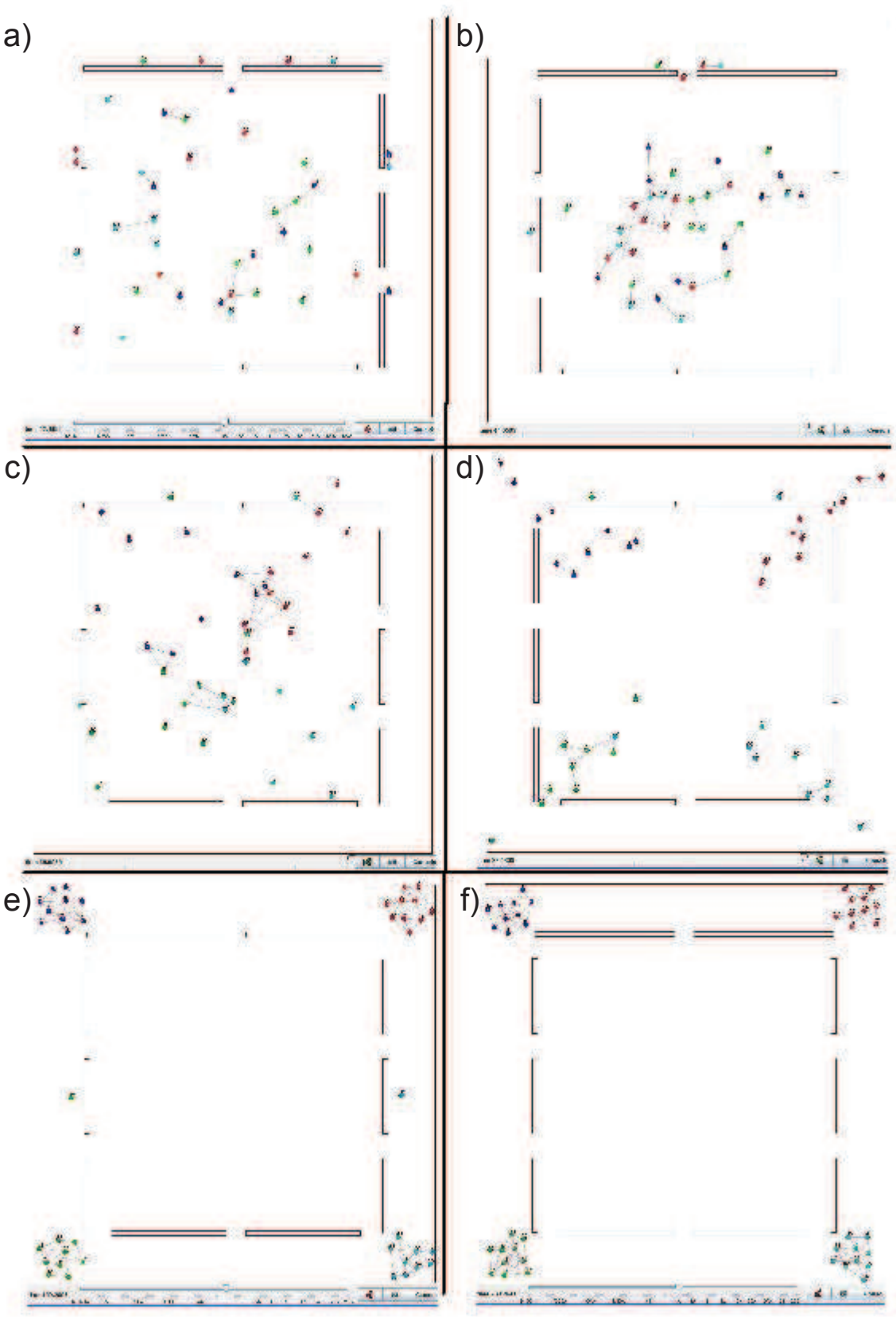


Figure I.8: Scénario de mouvement de groupes. Evolution (de a à f) de nœuds placés aléatoirement bougeant vers leur zone cible.

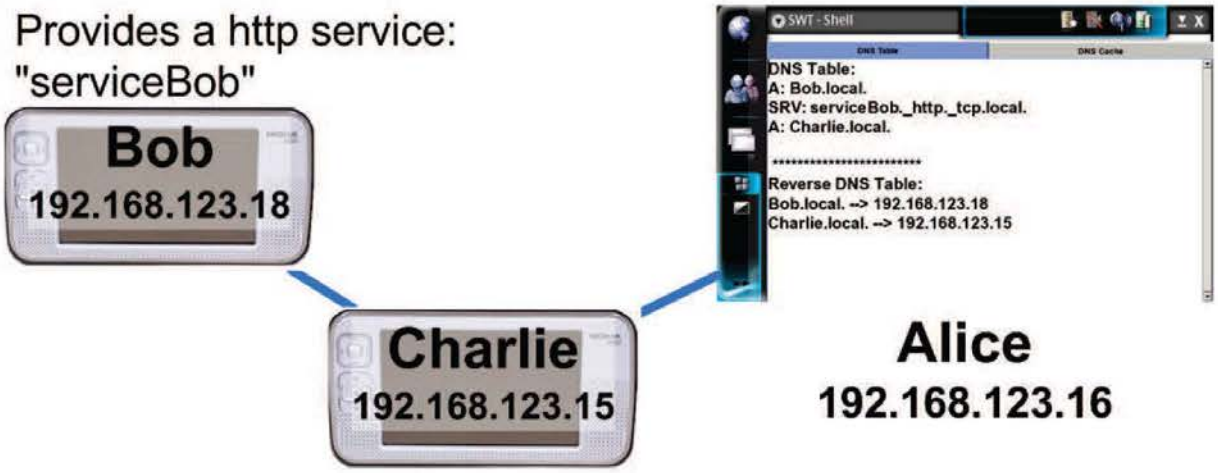


Figure I.9: Expérience avec Zeroconf au-dessus de SLSR.

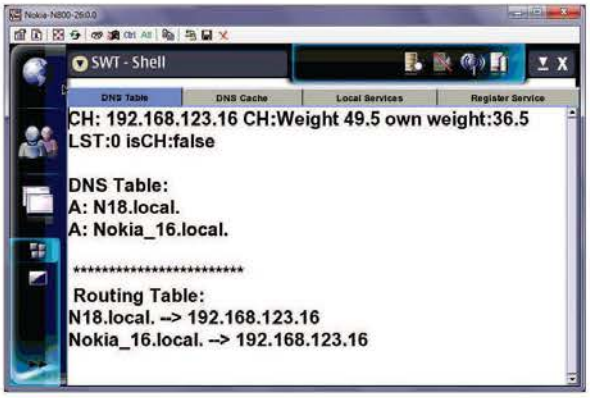


Figure I.10: Table DNS.

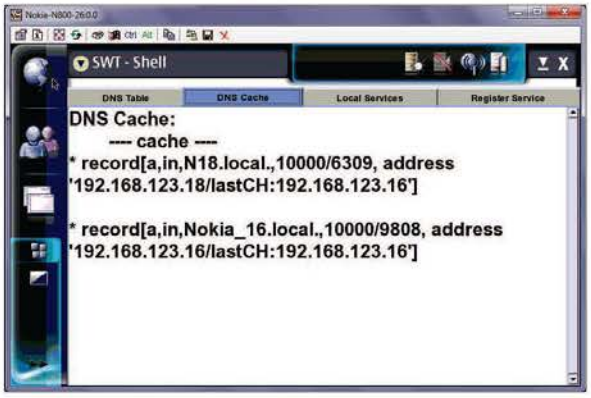


Figure I.11: Cache DNS.



Figure I.12: Services disponibles dans Zeroconf.



Figure I.13: Fenêtre de chat et d'interaction.

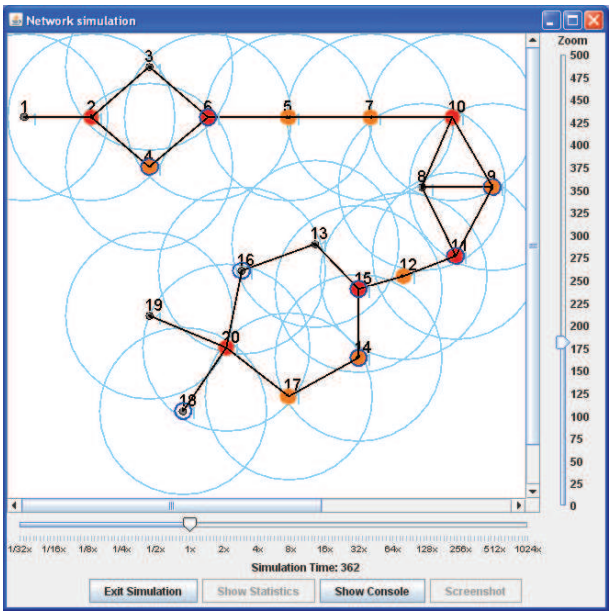


Figure I.14: Table DNS.

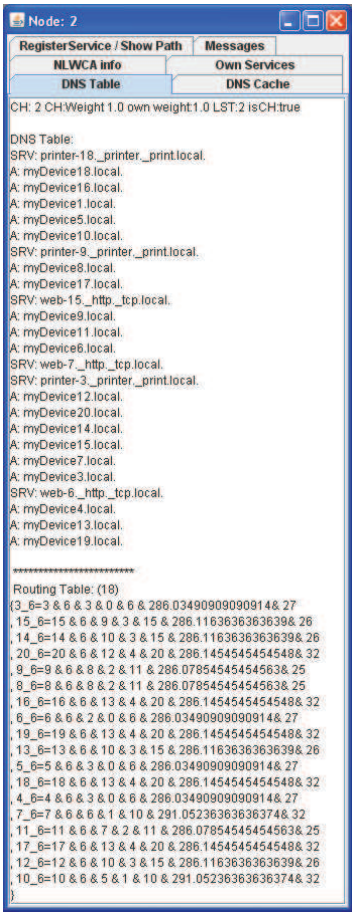


Figure I.15: Services disponibles dans Zero-conf.

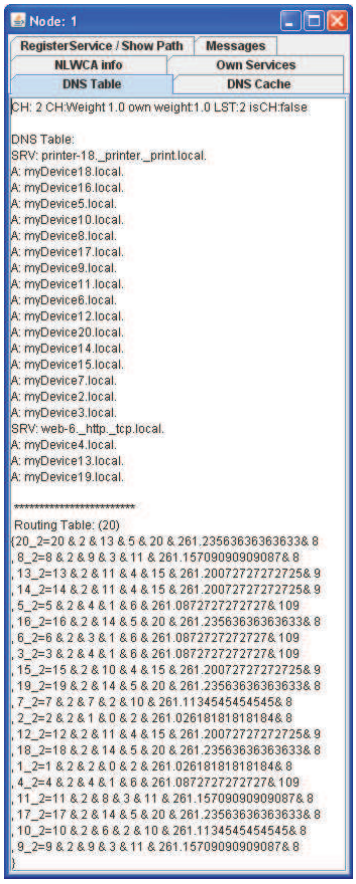


Figure I.16: Cache DNS.

Appendix J

Conclusion

Dans la première partie de cette thèse nous avons introduit les différents domaines liés à la découverte de services dans les réseaux ad hoc. Tout en considérant la découverte de services comme objet principal à améliorer, notre approche a aussi pris en compte les autres éléments, impliqués dans le travail, conduisant à cet objectif d'améliorer la découverte de services. Le but de cette thèse était de :

> Fournir une découverte de services spécifiquement adaptée aux réseaux ad hoc de telle sorte qu'elle puisse, en s'adaptant au mieux au contexte environnant, prendre en compte une grande mobilité, de nombreux services disponibles ainsi que des équipements hétérogènes.

Ainsi nous avons fait des propositions dans plusieurs domaines qui contribuent à divers niveaux à la découverte de services. La conception de ces propositions avait pour but d'obtenir un comportement symbiotique des diverses couches de protocoles et applications. Un protocole ou une application doit être capable de tirer profit des efforts déjà fournis par les couches sous-jacentes sans pour autant dégrader ses performances.

Les contributions proposées sont les suivantes :

- Dissémination:
  - Nous proposons une structure de dissémination, Stable Linked Structure Flodding, qui est basé sur des clusters à un saut. Cette structure permet d'améliorer les performances et de réduire le coût de la dissémination en sélectionnant de manière judicieuse les nœuds intermédiaires.
- Routage:
  - En prenant avantage de la structure de SLSF, nous proposons un protocole de routage qui est capable d'incorporer ses annonces de routage dans le trafic présent dans le réseau. Le protocole proposé, SLSR, peut adapter la diffusion dans le réseau de certaines informations de routage selon le contexte.
- Filtrage Collaboratif :
  - Nous présentons un algorithme de filtrage collaboratif, basé sur la même structure de cluster que SLSF, que nous proposons d'utiliser pour filtrer les services d'un réseau pour n'en garder que les meilleurs.
- Outils de simulations:

  - Pour assister la recherche, nous proposons un framework qui permet l'interaction de modèles et la multi-simulation. Ainsi notre approche permet de prendre en compte simultanément le comportement du réseau et des utilisateurs pour réaliser des scénarios adaptés aux réseaux ad hoc.

- Découverte de services : Chacune des propositions précédentes contribue à la découverte de services d'une manière différente:

  - Dissémination: La découverte de services diffuse ses annonces dans le réseau. Optimiser la dissémination des messages améliore aussi les performances de la découverte de services
  - Routage: Le routage va permettre, une fois un service découvert, de le contacter individuellement.
  - Filtrage collaboratif : Dans un réseau dense, comportant de nombreux services, le filtrage collaboratif permet de ne proposer, aux utilisateurs, que les meilleurs services.
  - Outils de simulation : Les outils développés contribuent de manière générale à la recherche dans les réseaux ad hoc et plus indirectement à la recherche sur la découverte de services.
  - Zeroconf: Nous avons utilisé Zeroconf, un protocole de découverte de services standardisé et très populaire. Nous proposons d'adapter Zeroconf aux réseaux ad hoc en remplacant le multicast avec notre structure de dissémination SLSF.

Appendix K

Perspectives

Contexte et métriques pour la découverte de services

Nous proposons une architecture de découverte de services qui est capable d'incorporer des métriques et le contexte à différents niveaux. Dans un travail futur nous allons proposer plus de métriques et analyser plus en détail l'impact et la pertinence des métriques à chaque niveau. Notre but, est de caractériser au mieux la situation afin d'adapter au mieux les protocoles à ce contexte.

Filtrage collaboratif

De plus, nous prévoyons d'étudier les avantages de l'utilisation du filtrage collaboratif pour la découverte de services.

Evaluation de SLSR

Prochainement nous allons évaluer en détail SLSR, le protocole que nous proposons. Nous prévoyons d'utiliser le modèle de mobilité qui inclut une boucle fermée entre l'utilisateur et le réseau pour évaluer les performances et les limites de SLSR. Dans une analyse plus poussée, nous nous intéresserons spécifiquement au gain du comportement symbiotique.

Vers de nouveaux modèles de mobilité standard

Dans le court terme, nous prévoyons de montrer à l'opposé, l'effet perturbateur des comportements non-conformes et d'étendre nos expérimentations à des protocoles et des scénarios plus avancés.

Expérimentations

Afin d'évaluer toutes nos simulations, nous prévoyons d'expérimenter nos protocoles SLSF et SLSR avec Zeroconf dans des scénarios à plus grande échelle. Il sera particulièrement intéressant de comparer ces résultats avec ceux obtenus lors des simulations. Pour aller encore plus loin, nous pourrions alors peaufiner le comportement des utilisateurs en utilisant les résultats d'expérience pour obtenir des résultats de simulation plus réalistes.

Bibliography

[Abde 07] F. Abdesslem, L. Iannone, M. Dias De Amorim, K. Obraczka, I. Solis, and S. Fdida. "A Prototyping Environment for Wireless Multihop Networks". In: *Proceedings of the 3rd Asian conference on Internet Engineering: Sustainable Internet*, pp. 33–47, Springer-Verlag, Berlin, Heidelberg, 2007.

[Adji 02] C. Adjih, P. Jacquet, and L. Viennot. "Computing connected dominated sets with multipoint relays". Research Report RR-4597, INRIA, 2002.

[Andr 08a] A. Andronache and S. Rothkugel. "HyTrace Backbone-Assisted Path Discovery in Hybrid Networks". *Communication Theory, Reliability, and Quality of Service, 2008. CTRQ '08. International Conference on Communication Theory, Reliability, and Quality of Service*, pp. 34–40, 29 2008-July 5 2008.

[Andr 08b] A. Andronache and S. Rothkugel. "NLWCA Node and Link Weighted Clustering Algorithm for Backbone-Assisted Mobile Ad Hoc Networks". *Networking, 2008. ICN 2008. Seventh International Conference on Networking*, pp. 460–467, April 2008.

[Bado 05] R. Badonnel and O. Festor. "Management of mobile ad-hoc networks: evaluating the network behavior". In: *IM 2005. 9th IFIP/IEEE International Symposium on Integrated Network Management*, pp. 17–30, IEEE, 2005.

[Bai 04] F. Bai and A. Helmy. "A Survey Of Mobility Models". In: *Wireless Adhoc Networks*, 2004.

[Basu 01] P. Basu, N. Khan, and T. Little. "A mobility based metric for clustering in mobile ad hoc networks". *icdcsw*, p. 0413, 2001.

[Bell 58] R. Bellman. "On a Routing Problem". *Quarterly of Applied Mathematics*, Vol. 16, No. 1, pp. 87–90, 1958.

[Belo 05] R. Belotti, C. Decurtins, M. C. Norrie, B. Signer, and L. Vukelja. "Experimental platform for mobile information systems". In: *MobiCom '05: Proceedings of the 11th annual international conference on Mobile computing and networking*, pp. 258–269, ACM, New York, NY, USA, 2005.

[Blav 02] L. Blažević, S. Giordano, and J.-Y. Le Boudec. "Self Organized Terminode Routing". *Cluster Computing*, Vol. 5, No. 2, 2002.

[Blum 04] J. Blum, M. Ding, A. Thaeler, and X. Cheng. "Connected Dominating Set in Sensor Networks and MANETs". In: *Handbook of Combinatorial Optimization*, pp. 329–369, D.-Z. Du and P. Pardalos, Kluwer Academic Publisher, 2004.

[Bona 01] P. Bonacich and P. Lloyd. "Eigenvector-like measures of centrality for asymmetric relations". *Social Networks*, Vol. 23, No. 3, pp. 191 – 201, 2001.

[Bray 08] "Extensible Markup Language (XML). `http://www.w3.org/TR/xml/`". 2008.

[Broc 98] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. "A performance comparison of multi-hop wireless ad hoc network routing protocols". In: *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pp. 85–97, ACM, New York, NY, USA, 1998.

[Camp 02] T. Camp, J. Boleng, and V. Davies. "A Survey of Mobility Models for Ad Hoc Network Research". *Wireless Communications and Mobile Computing (WCMC): Special issue on Mobile Ad hoc networking: Research, trends, and applications*, Vol. 2, pp. 483–502, 2002.

[Ches 05] S. Cheshire and D. Steinberg. *Zero Configuration Networking: The Definitive Guide*. O'Reilly Media, Inc., 2005.

[Ches 06] S. Cheshire and M. Krochmal. "DNS-Based Service Discovery". `http://tools.ietf.org/html/draft-cheshire-dnsext-dns-sd-04`, 2006.

[Ches 11] S. Cheshire and M. Krochmal. "Multicast DNS". draft-cheshire-dnsext-multicastdns-13, `http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt`, 2011.

[Ciar 01] L. Ciarletta, V. Iordanov, and A. Dima. "Using Intelligent Agents to assess Pervasive Computing Technologies". In: *IAWTIC 2001*, p. 10 p, Las Vegas, USA, 2001.

[Ciar 02] L. Ciarletta. *Contribution à l'évaluation des technologies de l'informatique ambiante*. PhD thesis, Université Henri Poincaré - Nancy I, 2002.

[Ciar 07] L. Ciarletta, T. Leclerc, G. Pujolle, P. Borras, and S. Gashti. "Etude des protocoles de découverte de service dans le cadre des réseaux adhoc". Contrat, 2007.

[Ciar 09] L. Ciarletta, T. Leclerc, and L. Reynaud. "Architecture pour la découverte de services avancée". 2009.

[Clau 03] "Optimized Link State Routing Protocol (OLSR), rfc3626. `http://www.ietf.org/rfc/rfc3626.txt`". 2003.

[Cost 05] R. Cöster and M. Svensson. "Incremental collaborative filtering for mobile devices". In: *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pp. 1102–1106, ACM, New York, NY, USA, 2005.

[Davi 00] P. Davidsson. "Multi Agent Based Simulation: Beyond Social Simulation". In: *3. Workshop on Multi Agent Based Simulation (MABS) 2000, LNAI*, pp. 97–107, Springer, 2000.

[De C 04] D. S. J. De Couto and R. T. Morris. "High-throughput routing for multi-hop wireless networks". Tech. Rep., Ph.D. thesis, MIT, 2004.

[De S 06] R. De Spindler, M. C. Norrie, M. Grossniklaus, and B. Signer. "Spatio-Temporal Proximity as a Basis for Collaborative Filtering in Mobile Environments". 2006.

[Dey 01] A. K. Dey. "Understanding and Using Context". *Personal Ubiquitous Comput.*, Vol. 5, No. 1, pp. 4–7, 2001.

[Duda 02] A. Duda. “Ambient Networking”. Réseaux Haut-débit et Multimédia (RHDM’2002), 2002.

[Dyna 97] “Dynamic Host Configuration Protocol (DHCP) `http://www.ietf.org/rfc/rfc2131.txt`”. 1997.

[EIGRP 92] “Enhanced Interior Gateway Routing Protocol (EIGRP)”. 1992.

[Elec 00] I. of Electrical, E. Engineers, and I.-S. S. Board. *IEEE standard for modeling and simulation (M&S) High Level Architecture (HLA): framework and rules. IEEE standard*, Institute of Electrical and Electronics Engineers, 2000.

[Foro 05] F. Foroozan and K. Tepe. “A high performance cluster-based broadcasting algorithm for wireless ad hoc networks based on a novel gateway selection approach”. In: *PE-WASUN ’05*, pp. 65–70, ACM, 2005.

[Free 79] L. Freeman. “Centrality in social networks: Conceptual clarification”. *Social Networks*, Vol. 1, No. 3, pp. 215–239, 1979.

[Frie 91] N. E. Friedkin. “Theoretical Foundations for Centrality Measures”. *The American Journal of Sociology*, Vol. 96, No. 6, pp. 1478–1504, may 1991.

[Gaud 07] N. Gaud. *Systèmes multi-agents holoniques : de l’analyse à l’implantation. Méta-modèle, méthodologie, et simulation multi-niveaux.* PhD thesis, Université de Franche Compté et Universite de Belfort-Montbéliard, 7 Dec. 2007.

[Gerh 02] M. Gerharz, C. de Waal, M. Frank, and P. Martini. “Link Stability in Mobile Wireless Ad Hoc Networks”. *LCN, Annual IEEE Conference on Local Computer Networks*, Vol. 0, p. 0030, 2002.

[Gola 99] “Simple Service Discovery Protocol (SSDP), internet draft. `http://tools.ietf.org/id/draft-cai-ssdp-v1-03.txt`”. 1999.

[Gorg 07] D. Gorgen, H. Frey, and C. Hiedels. “JANE-The Java Ad Hoc Network Development Environment”. *Simulation Symposium, 2007. ANSS ’07. 40th Annual*, pp. 163–176, March 2007.

[graphviz] Graphviz - Graph visualization software. `http://www.graphviz.org/`.

[Grat 08] P. Gratz, A. Andronache, and S. Rothkugel. “Ad Hoc Collaborative Filtering for Mobile Networks”. In: *SUTC ’08: Proceedings of the 2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (sutc 2008)*, pp. 355–360, IEEE Computer Society, Washington, DC, USA, 2008.

[Grat 09] P. Gratz and T. Leclerc. “Delay-tolerant collaborative filtering”. In: *MobiWAC ’09: Proceedings of the 7th ACM international symposium on Mobility management and wireless access*, pp. 109–113, ACM, New York, NY, USA, 2009.

[Haas 02] Z. J. Haas, M. R. Pearlman, and P. Samar. “The Zone Routing Protocol (ZRP) for Ad Hoc Networks”. Tech. Rep., Networking Laboratory,Helsinki University of Technology, July 2002.

[Hami 09] E. Ben Hamida, G. Chelius, and J.-M. Gorce. “Impact of the Physical Layer Modeling on the Accuracy and Scalability of Wireless Network Simulation”. *Simulation*, Vol. 85, pp. 574–588, 09 2009.

[Hann 05] R. A. Hanneman and M. Riddle. "Introduction to social network methods: 10. Centrality and power". `http://www.faculty.ucr.edu/~hanneman/nettext/C10_Centrality.html`, 2005.

[Hela 03] S. Helal, N. Desai, V. Verma, and C. Lee. "Konark - a service discovery and delivery protocol for ad-hoc networks". *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, Vol. 3, pp. 2107–2113 vol.3, March 2003.

[Helb 05] D. Helbing, L. Buzna, A. Johansson, and T. Werner. "Self-Organized Pedestrian Crowd Dynamics: Experiments, Simulations, and Design Solutions". *Transportation Science*, Vol. 39, No. 1, pp. 1–24, 2005.

[Herl 99] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. "An algorithmic framework for performing collaborative filtering". In: *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 230–237, ACM, New York, NY, USA, 1999.

[Hogi 06] L. Hogie, P. Bouvry, and F. Guinand. "An Overview of MANETs Simulation". *Electron. Notes Theor. Comput. Sci.*, Vol. 150, pp. 81–101, March 2006.

[Holb 06] H. Holbrook, B. Cain, and B. Haberman. "Internet Group Management Protocol Version 3 (IGMPv3)". 2006.

[Hong 99] X. Hong, M. Gerla, G. Pei, and C.-C. Chiang. "A group mobility model for ad hoc wireless networks". In: *Proceedings of the 2nd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pp. 53–60, ACM, New York, NY, USA, 1999.

[Hugu 06] L. Hugues, N. Bredeche, and T. I. Futurs. "Simbad: an Autonomous Robot Simulation Package for Education and Research". In: *in Proceedings of The Ninth International Conference on the Simulation of Adaptive Behavior (SAB'06). Roma, Italy - Springer's Lecture Notes in Computer Sciences / Artificial Intelligence series (LNCS/LNAI) n*, pp. 831–842, 2006.

[J SIM] "J-Sim (formerly known as JavaSim)". `http://sites.google.com/site/jsimofficial/`.

[Jacq 01] P. Jacquet, A. Laouiti, P. Minet, and L. Viennot. "Performance Analysis of OLSR Multipoint Relay Flooding in Two Ad Hoc Wireless Network Models". *RSRCP, Special issue on Mobility and Internet*, 2001.

[jmDNS] "JmDNS. `http://jmdns.sourceforge.net/`".

[Jodr 06] J. Jodra, M. Vara, J. Cabero, and J. Bagazgoitia. "Service discovery mechanism over OLSR for mobile ad-hoc networks". *AINA 2006. 20th International Conference on Advanced Information Networking and Applications*, Vol. 2, pp. 6 pp.–, April 2006.

[Jr 56] L. F. Jr. "Network Flow Theory". Paper P-923, The RAND Corperation, Santa Moncia, California, August 1956.

[jung] Jung - Java Universal Network/Graph Framework. `http://jung.sourceforge.net/`.

[Junh 09] L. Junhai, Y. Danxia, X. Liu, and F. Mingyu. "A survey of multicast routing protocols for mobile Ad-Hoc networks". *Communications Surveys Tutorials, IEEE*, Vol. 11, No. 1, pp. 78 –91, 2009.

[Kuma 01] I. Kumaran and S. I. Kumaran. *Jini Technology: An Overview*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[Kurk 05] S. Kurkowski, T. Camp, and M. Colagrosso. "Manet simulation studies: The incredibles". *ACM SIGMOBILE Mobile Computing and Communications Review*, Vol. 9, pp. 50–61, 2005.

[Lecl 07] T. Leclerc. "Métriques contextuelles et outils d'évaluation pour la découverte de service dans les réseaux hybrides". Research Report, MADYNES-INRIA LORRAINE, 2007.

[Lecl 08] T. Leclerc, L. Ciarletta, A. Andronache, and S. Rothkugel. "OLSR and WCPD as Basis for Service Discovery in MANETs". *Mobile Ubiquitous Computing, Systems, Services and Technologies, 2008. UBICOMM '08.*, pp. 184–190, 29 2008-Oct. 4 2008.

[Lecl 09] T. Leclerc, A. Andronache, L. Ciarletta, and S. Rothkugel. "Stabilizing cluster structures in mobile networks for OLSR and WCPD as Basis for Service Discovery". *International Journal on Advances in Internet Technologies*, Vol. 2, pp. 206–214, 2009.

[Lecl 10a] T. Leclerc, L. Ciarletta, L. Reynaud, and A. Schaff. "Prototype d'architecture de découverte de services avancée". Research Report, Feb. 2010.

[Lecl 10b] T. Leclerc, L. Ciarletta, and A. Schaff. "SLSF: Stable Linked Structure Flooding For Mobile Ad Hoc Networks". In: *IEEE ISWPC*, Palazzo Ducale, Modena, Italy, 2010.

[Lecl 10c] T. Leclerc, L. Ciarletta, and A. Schaff. "A Stable Linked Structure Flooding for Mobile Ad Hoc Networks with Fault Recovery". In: *WWIC*, pp. 204–215, 2010.

[Lecl 10d] T. Leclerc, J. Siebert, V. Chevrier, L. Ciarletta, and O. Festor. "Multi-modeling and co-simulation-based mobile ubiquitous protocols and services development and assessment". In: *7th International ICST Conference on Mobile and Ubiquitous Systems - Mobiquitous 2010*, Sydney Australia, 12 2010.

[Lend 05] V. Lenders, M. May, and B. Plattner. "Service discovery in mobile ad hoc networks: a field theoretic approach". *World of Wireless Mobile and Multimedia Networks, 2005. WoWMoM 2005. Sixth IEEE International Symposium on a*, pp. 120–130, June 2005.

[Li 05] L. Li and L. Lamont. "A Lightweight Service Discovery Mechanism for Mobile Ad Hoc Pervasive Environment Using Cross-Layer Design". In: *PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, pp. 55–59, IEEE Computer Society, Washington, DC, USA, 2005.

[Lin 04] G. Lin, N. G, and R. Rajaraman. "Mobility models for ad hoc network simulation". In: *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, p. 4 vol. (xxxv+2866), 2004.

[Liu 05] C. Liu and J. Kaiser. "A Survey of Mobile Ad Hoc network Routing Protocols". Tech. Rep., University of Ulm, Oct. 2005.

[Lou 04] W. Lou and J. Wu. "Double-covered broadcast (DCB): a simple reliable broadcast algorithm in MANETs". In: *INFOCOM 2004*, 2004.

[Madh] "Madhoc - The Metropolitan Adhoc network simulator". `http://www-lih.univ-lehavre.fr/~hogie/madhoc/`.

[Maed 05] K. Maeda, K. Sato, K. Konishi, A. Yamasaki, A. Uchiyama, H. Yamaguchi, K. Yasumoto, and T. Higashino. "Getting urban pedestrian flow from simple observation: realistic mobility generation in wireless network simulation". In: *Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pp. 151–158, ACM, New York, NY, USA, 2005.

[Mobi] "Mobile Ad-Hoc Networks(MANET) Working Group,Internet Engineering Task Force. `http://www.ietf.org/html.charters/manet-charter.html`".

[Mock 87] "Domain Names - Concepts and facilities, rfc1034. `http://www.ietf.org/rfc/rfc1034.txt`". 1987.

[Muso 06] M. Musolesi and C. Mascolo. "A community based mobility model for ad hoc network research". In: *Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*, pp. 31–38, ACM, New York, NY, USA, 2006.

[Naic 06] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai. "A Survey of Peer-to-Peer Network Simulators". *Proceedings of The Seventh Annual Postgraduate Symposium, Liverpool, UK*, 2006.

[Nguy 11] A. D. Nguyen, P. Sénac, V. Ramiro, and M. Diaz. "STEPS - an approach for human mobility modeling". In: *Proceedings of the 10th international IFIP TC 6 conference on Networking - Volume Part I*, pp. 254–265, Springer-Verlag, Berlin, Heidelberg, 2011.

[Ni 99] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. "The broadcast storm problem in a mobile ad hoc network". In: *MobiCom '99*, ACM Press, NY, USA, 1999.

[ns 2] "ns-2 - The network simulator". `http://www.isi.edu/nsnam/ns/`.

[ns 3] "ns-3 - The network simulator". `http://www.nsnam.org/`.

[Oliv 00] B. Olivier. "Jini: a platform for building adaptive integrated learning environments". Report from the Centre for Learning Technology (CeLT), University of Wales Bangor, United Kingdom,, 2000.

[OMNE] "OMNET++". `http://www.omnetpp.org/`.

[Opne] "Opnet". `http://www.opnet.com/`.

[Pari 09] G. Parissidis, M. Karaliopoulos, R. Baumann, T. Spyropoulos, and B. Plattner. "Routing Metrics for Wireless Mesh Networks". In: S. Misra, S. C. Misra, and I. Woungang, Eds., *Guide to Wireless Mesh Networks*, pp. 199–230, Springer London, 2009.

[Peng 01] W. Peng and X. Lu. "AHBP: An efficient broadcast protocol for mobile Ad hoc networks". *Journal of Computer Science and Technology*, Vol. 16, No. 2, 2001.

[Perk 03] C. Perkins, E. Royer, and S. Das. "RFC 3561 Ad hoc On-Demand Distance Vector (AODV) Routing". 2003.

[Perk 94] C. E. Perkins and P. Bhagwat. "Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers". In: *Proceedings of the conference on Communications architectures, protocols and applications*, pp. 234–244, ACM, New York, NY, USA, 1994.

[Qual] "Qualnet". `http://www.scalable-networks.com/products/qualnet/`.

[Rats 02] O. Ratsimor, D. Chakraborty, A. Joshi, and T. Finin. "Allia: alliance-based service discovery for ad-hoc environments". In: *WMC '02: Proceedings of the 2nd international workshop on Mobile commerce*, pp. 1–9, ACM, New York, NY, USA, 2002.

[Reyn 10] L. Reynaud, F. Jan, L. Ciarletta, T. Leclerc, and G. Shahab. "Intégration des services à la plateforme pour démonstrateurs". 2010. Livrable L5.2, Sous-Projet 5 (SP 5) : Expérimentations.

[Reyn 87] C. W. Reynolds. "Flocks, Herds, and Schools: A Distributed Behavioral Model". *Computer Graphics*, Vol. 21, pp. 25–34, 1987.

[Saha 02] L. Sahasrabuddhe and B. Mukherjee. "Multicast routing algorithms and protocols: A tutorial". *Network, IEEE*, Vol. 14, No. 1, pp. 90–102, 2002.

[Sail 05] F. Sailhan and V. Issarny. "Scalable Service Discovery for MANET". *PerCom 2005. Third IEEE International Conference on Pervasive Computing and Communications*, pp. 235–244, March 2005.

[Sant 05] P. Santi. "Topology control in wireless ad hoc and sensor networks". *ACM Comput. Surv.*, Vol. 37, No. 2, 2005.

[Sarw 01] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. "Item-based collaborative filtering recommendation algorithms". In: *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pp. 285–295, ACM, New York, NY, USA, 2001.

[Schi 08] R. Schifanella, A. Panisson, C. Gena, and G. Ruffo. "MobHinter: epidemic collaborative filtering and self-organization in mobile ad-hoc networks". In: *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pp. 27–34, ACM, New York, NY, USA, 2008.

[Sieb 09] J. Siebert, J. Rehm, V. Chevrier, L. Ciarletta, and D. Mery. "AA4MM coordination model: event-B specification, RR-7081". Tech. Rep., INRIA, 2009.

[Sieb 10] J. Siebert, L. Ciarletta, and V. Chevrier. "Agents and Artefacts for Multiple Models coordination. Objective and decentralized coordination of simulators.". In: *SAC 2010 25th Symposium on Applied Computing*, ACM, Lausanne Suisse, 2010.

[simbad] "Simbad simulator. `http://simbad.sourceforge.net/`".

[Simp 00] "Simple Object Access Protocol (SOAP). `http://www.w3.org/TR/soap/`". 2000.

[Sriv 05] V. Srivastava and M. Motani. "Cross-layer design: a survey and the road ahead". *Communications Magazine, IEEE*, Vol. 43, No. 12, pp. 112–119, 2005.

[Tekn 00] K. Teknomo, Y. Takeyama, and H. Inamura. "Review on microscopic pedestrian simulation model". In: *Proceedings Japan Society of Civil Engineering Conference*, 2000.

[Toh 97] C.-K. Toh. "Associativity-Based Routing for Ad Hoc Mobile Networks". *Wireless Personal Communications*, Vol. 4, pp. 103–139, 1997. 10.1023/A:1008812928561.

[Unif 94] "Uniform Resource Locators (URL). `http://www.ietf.org/rfc/rfc1738.txt`". 1994.

[UPnP] Universal Plug And Play Forum. `http://www.upnp.org/`.

[UPnP doc] Universal Plug And Play Documentation and Specifications. `http://upnp.org/sdcps-and-certification/standards/`.

[Veiz 97] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. "Service Location protocol". 1997.

[Weis 95] M. Weiser. "Human-computer interaction". Chap. The computer for the 21st century, pp. 933–940, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[Wu 02] J. Wu, F. Dai, M. Gao, and I. Stojmenovic. "On Calculating Power-Aware Connected Dominating Sets for Efficient Routing in Ad Hoc Wireless Networks". *IEEE/-KICS Journal of Communications and Networks*, Vol. 4, pp. 59–70, 2002.

[Yoon 03] J. Yoon, M. Liu, and B. Noble. "Random waypoint considered harmful". In: *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, pp. 1312 – 1321 vol.2, 2003.

[Zeng 98] X. Zeng, R. Bagrodia, and M. Gerla. "GloMoSim: A Library for Parallel Simulation of Large-Scale Wireless Networks". In: *Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS'98)*, pp. 154–161, IEEE Computer Society, May 1998.

[Zero] Zeroconf. "Zerconf official homepage. `http://files.zeroconf.org/rfc3927.txt`".

Résumé

Lors de la dernière décennie, le nombre d'appareils possédant des capacités sans fil a très fortement augmenté, attirant ainsi le grand public vers les réseaux mobiles sans fil. Nous considérons le cas des réseaux mobiles ad hoc aussi connu sous le nom de MANET (Mobile Ad hoc NETworks). La caractéristique principale des MANETs est la grande dynamicité des nœuds (induite pas le mouvement des utilisateurs), la propriété volatile des transmissions sans fil, le comportement des utilisateurs, les services et leurs utilisations.

Cette thèse propose une solution complète pour la découverte de service dans les réseaux ad hoc, de la couche réseau sous-jacente à la découverte de service à proprement dite. La première contribution est le protocole Stable Linked Structure Flooding (SLSF) qui établi une structure basée sur des clusters stable et permet d'obtenir une dissémination efficace qui passe à l'échelle. La seconde contribution est SLSR (Stable Linked Structure Routing) qui utilise la structure de dissémination de SLSF et permet de faire du routage à travers le réseau. En utilisant ces protocoles comme base, nous proposons d'améliorer la découverte de service en prenant en compte le contexte. De plus, nous avons contribué à la simulation réseau en couplant des modèles et des simulateurs de domaines différents qui une fois couplés permettent d'élaborer et la simuler des scénarios riches et variés adaptés aux MANETs. Cette thèse à été réalisé au sein du projet ANR SARAH qui avait pour but le déploiement de service multimédia dans une architecture ad hoc hybride.

Mots-clés: Réseaux ad hoc, découverte de services, dissémination, clustering, routage, context aware, simulation, couplage de modèles

Abstract

In the last decade, the number of wireless capable devices increased drastically along with their popularity. Devices also became more powerful and affordable, attracting more users to mobile networks. In this thesis we consider service discovery in Mobile Ad hoc NETworks, also called MANETs, that are a collection of devices that communicate with each other spontaneously whenever they are in wireless transmission range without any preexisting infrastructure. The main characteristic of MANETs is the high dynamic of nodes (induced by the users moving around), the volatile wireless transmissions, the user behavior, the services and their usage.

This thesis proposes a complete solution for service discovery in ad hoc networks, from the underlying network up to the service discovery itself. A first contribution is the Stable Linked Structure Flooding (SLSF) protocol that creates stable based cluster structure and thereby provides scalable and efficient message dissemination. The second contribution is the Stable Linked Structure Routing (SLSR) protocol that uses the SLSF dissemination structure to enable routing capabilities. Using those protocols as basis, we propose to improve service discovery by additionally considering context awareness and adaptation. Moreover, we also contributed on improving simulations by coupling simulators and models that, together, can model and simulate the variety and richness of ad hoc related usage scenarios and their human characteristic.

This thesis was accomplished within the ANR SARAH project that aimed at deploying advanced multimedia services in a hybrid ad hoc network architecture.

Keywords: Ad hoc networks, service discovery, dissemination, clustering, routing, context aware, simulation, model coupling