Faculteit Toegepaste Wetenschappen
Vakgroep Informatietechnologie
Voorzitter: Prof. Dr. Ir. P. LAGASSE

Ontwikkeling van een resource discovery protocol voor infrastructured mobiele ad hoc netwerken

door

Stijn EECKHAUT & Wouter PUYPE

Promotoren: Prof. Dr. Ir. I. MOERMAN & Prof. Dr. Ir. B. DHOEDT
Scriptiebegeleiders: Ir. J. HOEBEKE & Ir. T. VAN LEEUWEN

Scriptie ingediend tot het behalen van de academische graad van
burgerlijk ingenieur in de computerwetenschappen

Academiejaar 2003–2004

Woord vooraf

Allereerst wensen wij onze promotoren, prof. dr. ir. I. Moerman en prof. dr. ir. B. Dhoedt, te bedanken voor het leveren van het interessante en brandend actuele thesisonderwerp. Verder willen wij onze begeleiders, ir. Jeroen Hoebeke en ir. Tom Van Leeuwen, bedanken. Ze waren steeds bereid om ons met raad en daad bij te staan.

Onze dank gaat ook uit naar onze ouders: zij gaven ons de kans in de best mogelijke omstandigheden te studeren.

Ook onze zussen, Ellen en Tine, en onze medestudenten Jan De Cock, Stijn Eyerman en Stijn Notebaert hebben bijgedragen tot de leuke momenten die we naast onze thesis dit jaar beleefd hebben.

Tot slot wens ik, Wouter, mijn vriendin Ellen te bedanken om er elke keer voor mij te zijn wanneer ik haar nodig had.

Stijn Eeckhaut & Wouter Puype, mei 2004

Toelating tot bruikleen

Stijn Eeckhaut & Wouter Puype, mei 2004

Ontwikkeling van een resource discovery protocol voor infrastructured mobiele ad hoc netwerken

door

Stijn EECKHAUT & Wouter PUYPE

Scriptie ingediend tot het behalen van de academische graad van
burgerlijk ingenieur in de computerwetenschappen

Academiejaar 2003–2004

Promotoren: Prof. Dr. Ir. I. MOERMAN & Prof. Dr. Ir. B. DHOEDT
Scriptiebegeleiders: Ir. J. HOEBEKE & Ir. T. VAN LEEUWEN
Faculteit Toegepaste Wetenschappen
Universiteit Gent

Vakgroep Informatietechnologie
Voorzitter: Prof. Dr. Ir. P. LAGASSE

Samenvatting

Een resource discovery protocol is een protocol dat toelaat dat een knoop in een netwerk een geschikte service, bv. een printer, vindt en kan gebruiken. Voor klassieke, bedrade netwerken bestaan hiervoor reeds een aantal gestandaardiseerde implementaties. Voor mobiele ad hoc netwerken daarentegen bevinden dergelijke protocollen zich nog in een experimentele fase. Mobiele ad hoc netwerken zijn autonome, mobiele, draadloze netwerken waar elke host eveneens een router is. De communicatie gebeurt over bandbreedtebeperkte, draadloze links en omdat het zendbereik van elke host beperkt is, wordt vaak gebruik gemaakt van zgn. multihop paden.
In deze thesis ontwikkelden we een resource discovery protocol voor infrastructured mobiele ad hoc netwerken. Dit zijn netwerken die naast een mobiel ad hoc netwerk ook bestaan uit een vast gedeelte dat eventueel met het Internet verbonden is. Er werd voor een gelaagde aanpak gekozen: het resource discovery protocol bevindt zich in de protocolstack enerzijds onder de applicatielaag en anderzijds boven de routing. We implementeerden zowel een reactief als een proactief protocol en deden dit in de netwerksimulator GloMoSim. De ontwikkeling werd gevolgd door een uitgebreid prestatieonderzoek.
In wat volgt beschrijven we eerst de context waarin we aan resource discovery willen gaan doen. Daarna wordt een studie gedaan van de reeds bestaande implementaties. Vervolgens wordt het ontwikkelde protocol voorgesteld en de prestatie ervan onderzocht. Tot slot trekken we een aantal besluiten uit het onderzoek.

Trefwoorden

mobiele ad hoc netwerken, resource discovery, protocolontwerp

Inhoudsopgave

Lijst van afkortingen

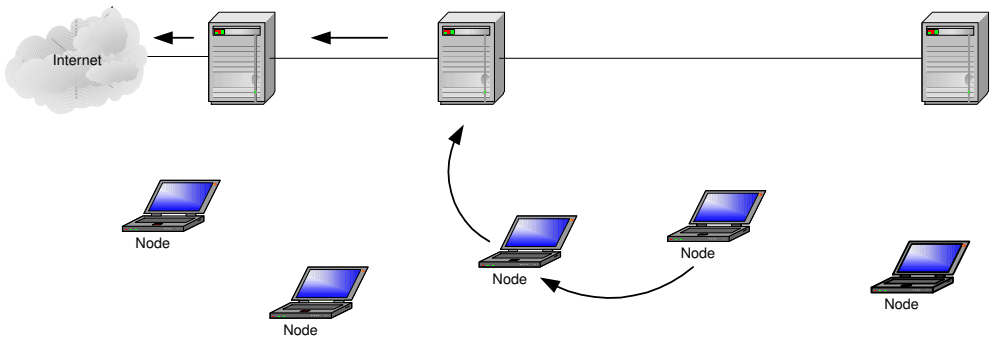
AODV	Ad hoc On-demand Distance Vector routing protocol
API	Application Programming Interface
DA	Directory Agent
DAML	DARPA Markup Language
DARPA	Defense Advanced Research Projects Agency
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
FTP	File Transfer Protocol
GloMoSim	Global Mobile information systems Simulation
GPS	Global Positioning System
GSM	Global System for Mobile communications
HTTP	Hyper Text Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
JVM	Java Virtuele Machine
MAC	Medium Access Control
MANET	Mobile Ad hoc NETwork
OS	Operating System
PBS	Pseudo Base Station
PDA	Personal Digital Assistant
P2P	Peer to Peer
QoS	Quality of Service
RD	Resource Discovery
RMI	Remote Method Invocation

SA	Service Agent
SDP	Service Discovery Protocol
SLM	Salutation Manager
SLP	Service Location Protocol
SSDP	Simple Service Discovery Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TTL	Time To Live
UA	User Agent
UDP	User Datagram Protocol
URL	Uniform Resource Locator
UPnP	Universal Plug and Play
WLAN	Wireless Local Area Network
WRP	Wireless Routing Protocol
XML	eXtensible Markup Language

Hoofdstuk 1

Inleiding

Mobiele ad hoc netwerken zijn autonome systemen van vrij bewegende mobiele knopen die beperkt zijn in resources en die met elkaar kunnen communiceren over draadloze verbindingen met een variabele bandbreedte en beperkte capaciteit. Dergelijke netwerken kunnen zeer snel ontplooid worden en lenen zich voor toepassingen zoals conferenties, reddingsoperaties, file-sharing... Recent gaat er echter ook aandacht uit naar de zogenaamde infrastructured ad hoc netwerken (zie figuur 1.1). Dergelijke netwerken bevatten naast mobiele knopen ook statische knopen. Deze knopen beschikken typisch over meer resources (zoals energievoorziening, rekencapaciteit) en kunnen onderling verbonden zijn via een vast netwerk. De mobiele knopen kunnen van deze infrastructuur gebruik maken voor de ondersteuning van routing en QoS, toegang tot het Internet of het gebruik van andere diensten zoals opslagruimte of rekencapaciteit. Een typisch voorbeeld van een dergelijk netwerk zou een conferentieruimte kunnen zijn die voorzien is van een draadloos backbonenetwerk waartoe de mobiele gebruikers toegang hebben.
Om dergelijke infrastructured ad hoc netwerken optimaal te kunnen gebruiken is er nood aan



Figuur 1.1: Een infrastructured mobiel ad hoc netwerk

een resource discovery protocol waarmee mobiele knopen de meest nabije knoop kunnen leren kennen die hen de gewenste diensten kan aanbieden. Dit protocol kan zowel reactief, proactief of hybried zijn. Proactief betekent dat iedere mobiele knoop op elk ogenblik de services in zijn omgeving kent. Dit kan gebeuren door de service providers periodiek berichten te laten broadcasten over hun services. Reactief daarentegen betekent dat een mobiele knoop pas op zoek gaat naar een geschikte service wanneer hij deze effectief nodig heeft. In deze thesis werd zowel een reactief als proactief resource discovery protocol ontwikkeld. Bijzondere aandacht werd besteed aan het beperken van de overhead (zoals aantal en bereik van protocolberichten) en de stabiliteit van het protocol in een dergelijke dynamische netwerktopologie. Beide versies van dit protocol werden geïmplementeerd in de GloMoSim-netwerksimulator. Beide technieken werden qua prestatie met elkaar vergeleken onder uiteenlopende netwerkomstandigheden van mobiliteit, aantal knopen, gemiddelde afstand in hops tot de services,...

Hoofdstuk 2

Literatuurstudie

2.1 Ad hoc netwerken, infrastructured netwerken en infrastructured ad hoc netwerken

Mobiele ad hoc netwerken De omgeving die gevormd wordt door mobiele gebruikers die enkel met elkaar communiceren over een draadloos medium is totaal verschillend van de traditionele infrastructuurgebaseerde omgeving bestaande uit PC's met bedrade verbindingen. Een mobiel ad hoc netwerk (MANET) is een autonoom systeem van mobiele routers (en geassociëerde hosts) die verbonden zijn via draadloze links. De nodes hebben een beperkt zendbereik en hebben dan ook dikwijls *multihop* paden nodig om een bron met een bestemming te verbinden. Om deze multihop paden te ontdekken en te onderhouden worden ad hoc routing protocollen (zie 2.2.1) gebruikt. De nodes mogen zich random verplaatsen en configureren zichzelf. De draadloze topologie van het netwerk kan dus snel en onvoorspelbaar veranderen. Hierdoor zullen bepaalde paden na verloop van tijd onbruikbaar worden en moeten nieuwe paden gezocht worden. Een MANET kan op zichzelf opereren of verbonden zijn met het Internet, en kan snel opgebouwd worden zonder dat daarvoor vaste netwerkinfrastructuur nodig is. Het energieverbruik van een node moet bovendien beperkt blijven omdat ze haar energie typisch uit een batterij haalt. De links tussen nodes hebben over het algemeen een lage capaciteit en verbindingen zijn dikwijls afhankelijk van de aanwezigheid van andere nodes. De meeste nodes zijn dus resource-arm.

Infrastructured draadloze netwerken Infrastructured mobiele draadloze netwerken zoals GSM-netwerken of Wireless LAN's (WLAN) bestaan uit een of meerdere vaste, bedrade gateways. Deze gateways worden ook wel *base stations* genoemd en vormen de brug tussen de

mobiele knopen enerzijds en het vaste netwerk anderzijds. Een mobiele gebruiker in het netwerk maakt *rechtstreeks* verbinding met een basisstation binnen zijn zendbereik. Wanneer een mobiele node buiten het bereik van een basisstation gaat en in het bereik van een ander komt, vindt een *handover* plaats.

Infrastructured vaste netwerken Deze categorie van netwerken (zoals het Internet begin 21^{e} eeuw) wordt gekenmerkt door haar statische en hiërarchische aard.

Infrastructured ad hoc netwerken In traditionele ad hoc netwerken worden alle nodes gelijk verondersteld. Het is echter ook mogelijk dat bepaalde nodes over een constante stroomvoorziening beschikken (netstroom), terwijl andere nodes toch nog verder werken op batterij. De resource-rijke nodes kunnen dan een groot deel van de routing en de forwarding op zich nemen. Deze nodes hebben een vaste positie. Door gebruik te maken van deze *pseudo basisstations* (PBS) [7] kunnen de mobiele nodes energie uitsparen. De mobiele nodes hebben verder de eigenschappen zoals beschreven in de paragraaf 'mobiele ad hoc netwerken' (zie boven).

2.2 Routing en resource discovery in mobiele ad hoc netwerken

2.2.1 Routing protocollen

Proactieve protocollen

Proactieve routingprotocollen kennen op elk ogenblik een route naar iedere andere bereikbare node. Dit gebeurt door de nodes periodiek berichten te laten broadcasten over hun aanwezigheid. Het nadeel van deze protocollen is dat ze veel overhead genereren. Het voordeel bestaat erin dat nodes die data willen verzenden niet meer moeten wachten op het zoeken van een route naar de bestemming en dus onmiddellijk met het verzenden van de data kunnen starten. In de simulaties van ons resource discovery protocol gebruiken we WRP [1] als proactief routing protocol.

Reactieve protocollen

In dit geval gaat een mobiele knoop pas op zoek naar de route naar een bepaalde knoop wanneer hij deze effectief nodig heeft. Wanneer er weinig trafiek is zal een reactief protocol minder overhead genereren dan een proactief protocol omdat dan weinig routes moeten gezocht worden. Het nadeel is echter dat een node die data wil verzenden moet wachten met de verzending totdat

een route gevonden werd door het routingprotocol. In de simulaties van ons resource discovery protocol gebruiken we AODV [2] als reactief routing protocol.

Hybride protocollen

Hybride protocollen combineren de principes van proactieve en reactieve protocollen. In de simulaties beperken we ons echter tot reactieve en proactieve routingprotocollen.

2.2.2 Overzicht van bestaande resource discovery protocollen

In statische of traag evoluerende netwerken (vb. IP-gebaseerde netwerken) wordt resource locatie geregeld door hiërarchische, gecentraliseerde services. Naamresolutie op het Internet bijvoorbeeld wordt door het Domain Name System (DNS) geregeld. DNS is echter hiërarchisch en zeer statisch. Er zijn een beperkt aantal root name servers en een groot aantal lokale name servers. Ieder IP-subnet heeft een vaste statische referentie naar de fysieke locatie van de lokale name server die gebruikt wordt om de naam-naar-IPadres vertaling te maken. Een veel gebruikte oplossing om nodemobiliteit in IP-netwerken te ondersteunen is het Dynamic Host Configuration Protocol (DHCP) dat een tijdelijk IP-adres toekent en IP-adressen geeft voor name servers, gateways en dergelijke. Dit vraagt echter nog steeds het gebruik van een server: de DHCP-server. De bestaande resource discovery protocollen zijn dus eerder ontworpen met bedrade netwerken met lage latency, betrouwbare links en genoeg bandbreedte in het achterhoofd. De meeste van deze protocollen maken gebruik van een centrale server waar service-aanbieders hun service kunnen registeren en waar service-gebruikers de door hun gewenste service kunnen opzoeken. Om resource discovery in mobiele ad hoc netwerken toe te laten op een schaalbare manier, wordt echter beter een peer-to-peer (P2P) aanpak gebruikt. P2P-applicaties en -protocollen crëeren een bepaald globaal gedrag op basis van een aantal lokale node-gebaseerde regels en beslissingen. In P2P-systemen is er geen gecentraliseerde controle en geen hiërarchische organisatie. De software die in iedere node draait is equivalent met de software in iedere andere node.
Er wordt voor gekozen het resource discovery protocol niet in te bedden in het gebruikte ad hoc routing protocol, maar dit te implementeren in een aparte laag boven de routinglaag en onder de applicatielaag. Hierna wordt een overzicht gegeven van de voornaamste resource discovery protocollen voor statische bedrade netwerken, samen met hun voornaamste eigenschappen en de mogelijke problemen bij gebruik in ad hoc netwerken.

Service Location Protocol (SLP)

SLP [12] is ontwikkeld door het IETF (Internet Engineering Task Force). Het is ontworpen voor TCP/IP-netwerken en is schaalbaar naar grote bedrijfsnetwerken. SLP kan zowel met als zonder gecentraliseerde controle werken. Algemeen bestaat de SLP-architectuur uit 3 componenten:

User Agents (UA) doen aan service discovery voor de clients.

Service Agents (SA) adverteren de locatie en karakteristieken van de services waarvoor ze optreden.

Directory Agents (DA) verzamelen de adressen en informatie die ze van SA's ontvangen in hun database en antwoorden op de service-aanvragen die de UA's maken.

De UA's en de SA's kunnen het adres van een DA op verschillende manieren te weten komen: ofwel statisch via DHCP, ofwel actief via een request aan het SLP multicast groep adres (239.255.255.253), ofwel passief door het opvangen van de advertisements die een DA uitzendt. Met betrekking tot de bruikbaarheid van dit protocol in mobiele ad hoc netwerken is het gelukkig zo dat het gebruik van een DA niet verplicht is. In dat geval zenden de UA's hun service request naar het SLP multicast adres.
Services worden beschreven door een *Service URL* en een *Service Template*. Het eerste bevat het IP-adres van de service, alsook het poortnummer en het pad. Het tweede specifiëert de attributen en hun default waarden.

Jini

Jini werd ontwikkeld door Sun en is een uitbreiding van de Java-programmeertaal. Alle hosts die Jini willen gebruiken moeten een Java Virtuele Machine (JVM) hebben. Jini heeft een Lookup Table op een Lookup server. In de Lookup Table kan naast de informatie over een service ook objectcode opgeslagen worden die door de host kan gedownload worden wanneer die de service wil gebruiken. Zo wordt in besturingssysteemonafhankelijkheid voorzien. Helaas is Jini niet bruikbaar zonder centrale server, waardoor dit niet echt bruikbaar is in mobiele ad hoc netwerken. Ook is de mobiliteit van de objectcode moeilijk te realiseren in een omgeving met lage bandbreedte waar de verplaatsing van nodes voortdurend communicatie veroorzaakt.

Salutation

Salutation [15] werd ontwikkeld door het Salutation Consortium. Iedere host heeft een Salutation Manager. Services kunnen zich registeren bij de Salutation Manager van hun host. Voor de beschrijving van services worden Service Records gebruikt. De Salutation Managers van verschillende hosts communiceren met elkaar door middel van het Salutation Manager Protocol. Op die manier raken de Salutation Managers op de hoogte van services die door andere hosts aangeboden worden. Voor de communicatie wordt gebruik gemaakt van remote procedure calls. Naar de hogere lagen van de architectuur biedt de Salutation Manager de SLM-API (SLM-Application Programming Interface) aan. Op die manier kunnen Salutation applicaties zo geschreven worden dat ze via het SLM-protocol met elkaar kunnen communiceren, onafhankelijk van de transportlaag. Onder de Salutation Manager bevindt zich de Transport Manager die de Salutation Manager transportonafhankelijk maakt. Salutation is dus onafhankelijk van de onderliggende infrastructuur en van de programmeertaal. Er zijn al een aantal commerciële implementaties van dit protocol. Salutation kan zonder centrale server gebruikt worden; de principes van dit protocol kunnen dus interessant zijn voor gebruik in mobiele ad hoc netwerken.

Universal Plug And Play (UPnP)

UPnP [14] werd ontwikkeld door een consortium dat geleid wordt door Microsoft. Het protocol breidt Microsoft's Plug and Play uit naar devices die verbonden zijn via een TCP/IP-netwerk. Er is geen centraal serviceregister voorzien. Het Simple Service Discovery Protocol (SSDP) wordt door de controlepunten in de nodes gebruikt om services aan te vragen en aan te bieden. SSDP gebruikt HTTP over UDP en is dus ontwikkeld voor gebruik in IP-netwerken. Voor SSDP hebben de nodes dus een IP-adres nodig. Dit kan verkregen worden via DHCP (wat moeilijk gaat in mobiele ad hoc netwerken wegens de DHCP-server) of via AUTO-IP wanneer er geen DHCP-server is. Deze laatste mogelijkheid zou in een ad hoc netwerk kunnen aangewend worden. Voor de beschrijving van de services wordt XML gebruikt.

Bluetooth Service Discovery Protocol (SDP)

SDP [13] werkt zonder een centrale server. SDP 1.0 voorziet niet in functionaliteit om toegang te krijgen tot services. SDP voorziet enkel in informatie *over* de services. Voor de werkelijke toegang kan een ander protocol zoals SLP of Salutation gebruikt worden. Bij het zoeken naar een service kan men zoeken op service type, op service attribuut, en men kan ook aan service

Tabel 2.1: Vergelijking van bestaande discovery protocollen

Eigenschap	**SLP**	**Jini**	**Salutation**	**UPnP**	**SDP**
Ontwikkelaar	IETF	Sun	Salutation Consortium	Microsoft	Bluetooth SIG
Licentie	open source	open source	open source	open for members	royaltee free
Versie	2	1.0	2.1	0.91	1.0
Netwerktransport	TCP/IP	onafh	onafh	TCP/IP	onafh
Programmeertaal	onafh	Java	onafh	onafh	onafh
OS & platform	afhankelijk	onafh	afhankelijk	afhankelijk	afhankelijk
Codemobiliteit	nee	ja (Java RMI)	nee	nee	nee
Zoeken op service-attributen	ja	ja	ja	nee	ja
Central cache repository	ja (optie)	optie via SLP	ja (optie)	nee	nee
Werking zonder directory	ja	Lookuptabel verplicht	ja	n.v.t.	n.v.t.
Leasing	ja	ja	nee	ja	ja
Beveiliging	IP afhankelijk	Java based	authenticatie	IP afhankelijk	authenticatie

browsing doen zonder op voorhand de karakteristieken van de service te kennen.

Overzicht

[3] vat de verschillen tussen de discovery protocollen samen zoals getoond in tabel 2.1.

2.2.3 Ontwerpsaspecten

De in 2.2.2 aangehaalde resource discovery protocollen (behalve Bluetooth SDP) zijn ontworpen met een weinig veranderlijke, hiërarchische netwerkstructuur in het achterhoofd. Bij het ontwerp van een discovery protocol voor ad hoc netwerken moet rekening gehouden worden met enkele zaken. In deze sectie worden een aantal van deze algemene ontwerpsaspecten nader besproken.

Netwerkvorming In traditionele bedrade netwerken gebeurt de netwerkvorming op een systematische manier. De nodes krijgen hun adres toegekend door een administrator of door een andere node in het netwerk. De nodes verplaatsen zich zelden of niet. In ad hoc netwerken kan men hier niet van uit gaan. Auto-configuratietechnieken zoals Auto-IP [16] kunnen gebruikt worden om de nodes van een adres te voorzien zonder dat hiervoor een centrale controle nodig is.

Opslag van service informatie Opslag van deze informatie in een centrale server behoort voor een ad hoc netwerk niet tot de mogelijkheden. Men kan er immers niet van uitgaan dat de node die het centraal register bevat, voor altijd in het netwerk blijft. Om geen informatie te verliezen t.g.v. het plotse verdwijnen van zo'n node zou de informatie al op verschillende plaatsen moeten gedupliceerd worden. Men zou ook een algoritme kunnen gebruiken dat bij het verdwijnen van een registernode een andere node uitkiest waar de nodes vanaf dan hun service kunnen registreren. Maar in dat geval moet het hele register opnieuw worden samengesteld, wat een kostelijke operatie zou zijn. De gemakkelijkste oplossing voor dit probleem is een volledig gedistribueerde benadering, waarbij iedere node lokaal bijhoudt welke services ze aanbiedt en welke services andere nodes aanbieden.

Beschrijving van service informatie Gebruik van XML lijkt hier interessant. Ook het gebruik van Service Templates zoals bij SLP (zie sectie 2.2.2) behoort tot de mogelijkheden. Een andere mogelijkheid is de DARPA Markup Language (DAML) [20] die in een semantische matching van services kan voorzien daar waar XML zuiver syntactische matching voorziet. In de simulaties kunnen we het echter eenvoudiger aanpakken door een service en de kwaliteit van deze service gewoon aan te duiden met gehele getallen (vb. FTP-server = 21, kwaliteit 10).

Verspreiden van service informatie Dit kan zowel door service advertisement (proactief) als door service discovery (reactief). Bij *service advertisements* pushen de service providers hun service informatie in het netwerk d.m.v. flooding. Dit kan op periodische basis gebeuren en/of ten gevolge van een gebeurtenis, zoals een nieuwe node die zich aanmeldt. De nodes die een service announcement ontvangen slaan de service informatie op en forwarden het bericht naar de andere nodes in hun zendbereik. In dit scenario moet broadcasting ondersteund worden. Bij *service discovery* doet de node die een service wil gebruiken een broadcast waarbij om de service gevraagd wordt. Een andere node kan hierop antwoorden wanneer zij de service aanbiedt of

een node kent die de service aanbiedt. Als de node geen antwoord weet kan ze de aanvraag forwarden.

Flooding en schaalbaarheid Om de vloed van berichten enigszins schaalbaar te houden kan men aan de berichten time-to-live (TTL) informatie toevoegen. Wanneer het bericht geforward wordt door een node zal deze de TTL-waarde decrementeren. Als dit de TTL op 0 brengt, stopt de forwarding. Wanneer een node een service request verzendt kan ze dit dus eerst doen met een kleine TTL, in de hoop een gewenste service-provider in de buurt te vinden. Als er geen antwoord komt op de aanvraag kan een nieuwe aanvraag met hogere TTL verstuurd worden zodat de aanvraag verder in het netwerk raakt (*expanding ring search*). Nadeel is wel dat de nodes in de onmiddellijke buurt van de aanvragende node mogelijks meerdere keren een bericht moeten doorsturen. Daartegenover staat dat nodes verder in het netwerk minder belast worden. In [5] wordt een andere manier voorgesteld om de flooding te beperken : iedere node die host is voor 1 of meerdere services, stuurt service advertisements in het netwerk over een gespecifieerde diameter (aantal hops). De nodes die binnen deze diameter liggen, cachen de service-informatie in hun serviceregister. Voor iedere node die een entry heeft in zo'n serviceregister wordt ook een lijst bijgehouden van services die 'in de nabijheid van die node' aangeboden worden door naburige hosts. Op die manier kan een node die een service aanvraag doet en geen overeenkomende service vindt in de lokale node of in een node die een entry heeft in haar serviceregister, kijken of in de nabijheid van een node uit het serviceregister de gewenste dienst aangeboden wordt. Dit laat een selectieve verzending van de service aanvraag toe naar de desbetreffende node, die op haar beurt de aanvraag in een bepaalde richting kan sturen.

Service delivery Er kan voor gekozen worden om naast de service discovery ook aan service delivery te doen. In dat geval houdt men zich niet alleen bezig met het ontdekken en beheren van service informatie, maar ook met het transporteren van data uit de hogere lagen van de protocolstack. Ons protocol zal enkel de service discovery uitvoeren.

Protocol stack Een resource discovery protocol dat onafhankelijk van de routing werkt, wordt best geïmplementeerd tussen de netwerklaag (waarin de routing gebeurt) en de applicatie (zie ook figuur 3.1).

Energieverbruik Traditioneel worden in mobiele ad hoc netwerken alle nodes qua mogelijkheden en energievoorziening als zijnde equivalent beschouwd. Het is echter interessant een

onderverdeling te maken tussen resource-rijke en resource-arme nodes. Een notebook die werkt op netstroom bijvoorbeeld heeft meer resources dan een notebook die op zijn batterij werkt, en deze laatste heeft bvb. meer resources dan een PDA. De resource-rijke nodes zouden een groter deel van de taken met betrekking tot resource discovery op zich kunnen nemen ten voordele van de resource-arme nodes. Deze techniek is echter pas echt zinvol wanneer ook de berichten ten behoeve van de routing grotendeels door de resource-rijke nodes behandeld worden, dus wanneer het routingprotocol en het resource discovery protocol gecombineerd worden tot 1 protocol.

Hoofdstuk 3

Protocolvoorstel

3.1 Algemeen

Eerst zullen we de karakteristieken van de nodes en de onderzochte services beschrijven waarvoor de protocollen ontwikkeld werden. Vervolgens gaan we dieper in op de criteria die in acht kunnen genomen worden bij de service selectie. Daarna wordt de werking van het proactieve en het reactieve protocol beschreven.

3.2 Karakteristieken van de nodes

Mobiliteitsmodel De beweging van de nodes in een mobiel ad hoc netwerk kan op verschillende manieren gemodelleerd worden. In een deel van de simulaties gaan we uit van het *Random Waypoint Mobility Model* van GloMoSim. In dit model gaat men ervan uit dat de verschillende nodes onafhankelijk van elkaar bewegen. Verder wordt verondersteld dat een node na een beweging voor een bepaalde periode ter plekke blijft. Na deze periode kiest de node een random bestemming en begint ze naar deze bestemming te reizen met een snelheid die ligt tussen een ondergrens en een bovengrens. Bij aankomst op de bestemming blijft de node weer een bepaalde periode ter plekke en herhaalt de situatie zich.
In de andere simulaties houden we ofwel alle nodes vast, ofwel laten we ze bewegen volgens een zelf gedefinieerd patroon dat kan vastgelegd worden met het GloMoSim-bestand *mobility.in*.
Ook de initiële plaatsing van de nodes kan beïnvloed worden. GloMoSim biedt eerst en vooral een aantal standaard plaatsingsmodellen. In het *random* model worden de nodes random geplaatst over het hele terrein. In het *uniform* model wordt het terrein opgedeeld in een aantal cellen waarbij in elke cel random één node geplaatst wordt. In het *grid* model tenslotte worden

de nodes op vaste afstand van elkaar geplaatst in een rechthoekig rooster. De nodes kunnen ook geplaatst worden volgens een zelf gedefinieerd patroon dat kan vastgelegd worden met het GloMoSim-bestand *nodes.input*.

Transmissie en ontvangst We veronderstellen dat transmissie en ontvangst gebeuren via de IEEE802.11 wireless LAN standaard.

Energievoorziening Er zijn zowel infrastructured nodes als mobiele nodes. De infrastructured nodes worden gekenmerkt door een vaste, niet-kritische stroomvoorziening. De mobiele nodes worden verondersteld op batterij te werken.

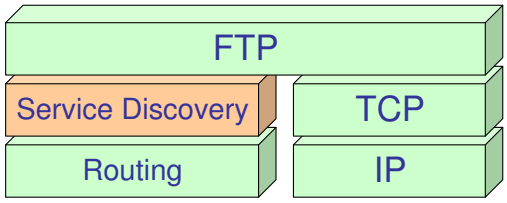
3.3 Karakteristieken van de onderzochte services

Voor de simulaties zijn we uitgegaan van een FTP-service en een HTTP-service (WWW-trafiek via een Internet gateway).

FTP Bij een FTP-sessie veronderstellen we dat een node een bestand heeft dat ze op een FTP-server (binnen het ad hoc netwerk) wenst te plaatsen. Gedurende de sessie is het onmogelijk van server te veranderen aangezien dit het bestand zou fragmenteren.

HTTP Bij een HTTP-sessie veronderstellen we dat een node documenten van een webserver wil halen. De node maakt hiertoe verbinding met de webserver via een (infrastructured) Internet gateway. Deze gateway bevindt zich in het ad hoc netwerk, maar is ook vast verbonden met het Internet, waarin ook de webserver zich bevindt. Er kunnen zich meerdere gateways in het ad hoc netwerk bevinden, en gedurende een sessie is het mogelijk om van gateway te veranderen telkens een document van de server afgehaald wordt. Een client kan dus via meerdere gateways verbinding maken met de webserver. Het resource discovery protocol houdt zich enkel bezig met het vinden van de meest geschikte gateway, niet met het verleggen van de verbinding tussen gateway en webserver wanneer van gateway veranderd wordt (handover). Deze handover wordt wel mogelijk verondersteld.

De service kan dus zowel een *single commodity* als een *multi commodity* service zijn. Bij een single commodity service is het mogelijk van service te veranderen wanneer men reeds begonnen is hem te gebruiken (HTTP voorbeeld). Bij een multi commodity service is dit niet mogelijk (FTP voorbeeld).



Figuur 3.1: Plaats van resource discovery en routing in de protocolstack

Iedere service heeft een *starttijd*, een *stoptijd*, en een *service attribuut*. Dit service attribuut is een 'kunstmatig' getal en geeft de kwaliteit van de service aan. Een applicatie die een service aanvraagt, zal bij deze aanvraag ook telkens een gewenst minimaal service attribuut opgeven. Enkel de services die minimaal dit gevraagde attribuut leveren, zullen kunnen gebruikt worden.

3.4 Uitgangspunten

Geen routing door resource discovery protocol Het resource discovery protocol wordt geïmplementeerd in een laag die zich bevindt tussen de applicatie en het routingprotocol. De routing gebeurt in de netwerklaag en is dus volledig gescheiden van de resource discovery (zie figuur 3.1). Het resource discovery protocol houdt zich enkel bezig met het ontdekken en beheren van services in het netwerk, en dus niet met routing. Het resource discovery protocol interageert wel met het routingprotocol om bijvoorbeeld de 'next hop' naar een node te kennen.

Alle nodes draaien hetzelfde protocol Zowel de infrastructured nodes als de mobiele nodes draaien hetzelfde resource discovery protocol. Het protocol maakt ook geen opdeling van nodes naargelang ze infrastructured of mobiel zijn. Bij de service selectie kunnen infrastructured nodes echter wel een ander gewicht krijgen wanneer de score van een service berekend wordt (zie sectie 3.6.6).

Proactieve en reactieve routing De invloed van het onderliggend routingprotocol op de resource discovery wordt onderzocht. Bij de simulaties wordt WRP gebruikt om de invloed te onderzoeken van een proactief routingprotocol; om de invloed van een reactief routingprotocol na te gaan wordt AODV gebruikt.

Niet locatiegebaseerd protocol Het protocol maakt voor zijn werking geen gebruik van de *absolute* posities van de nodes. Er wordt dus geen GPS-informatie gebruikt.

3.5 Service selectie

Single commodity en multi commodity services Service selectie moet steeds gebeuren wanneer een service voor het eerst aangevraagd wordt. Afhankelijk van het feit of tijdens het gebruik van de service nog van provider kan gewisseld worden, is het mogelijk dat er ook tijdens het gebruik van de service nog selecties nodig zijn. Een voorbeeld hiervan is het veranderen van Internet gateway tussen het opvragen van 2 HTTP-documenten (zie sectie 3.3).

Absolute metriek Wanneer een node een bestand van 2 MegaByte wil uploaden naar een FTP-server, kunnen alleen FTP-servers in aanmerking komen die ook werkelijk 2 MegaByte aan opslagruimte kunnen reserveren voor dit bestand. Dit is een voorbeeld van een absolute metriek. In ons protocol gebruiken we de abstracte parameter *service attribuut* als absolute metriek. Wanneer een node bijvoorbeeld een service met service attribuut 4 aanvraagt, dan zullen enkel de providers die deze service minimaal met service attribuut 4 aanbieden, kunnen geselecteerd worden.

Relatieve metriek Alle services die voldoen aan de absolute metriek kunnen in principe voor gebruik aangeboden worden aan de aanvragende applicatie. Het is echter wenselijk om van al deze services de beste te kunnen bepalen. Hiervoor kunnen een aantal criteria in aanmerking genomen worden :

lifetime Hoe langer de service nog aangeboden wordt, hoe interessanter het is om deze service te kiezen.

service attribuut Het service attribuut kan ook in de relatieve metriek gebruikt worden. Hoe groter het attribuut, hoe interessanter de service.

server komt dichter of verwijdert zich De geleverde kwaliteit van een server die aan het naderen is zal wellicht in de nabije toekomst nog verbeteren. De kans is groot dat de hopcount naar de service nog verder verkleint. Omgekeerd is het minder interessant om te connecteren met een service die zich op een verwijderende server bevindt.

infrastructured Infrastructured nodes zijn over het algemeen resource-rijk en hebben een lage mobiliteit en een constante stroomvoorziening. Het kiezen van infrastructured nodes kan de mobiele nodes (die hun energie uit een batterij halen) ontlasten, waardoor deze laatste langer kunnen functioneren.

hopcount Hoe minder tussenliggende nodes moeten gebruikt worden om de server te bereiken, hoe groter de kans dat het pad dat gevormd wordt door deze nodes een stabiel pad zal zijn, en hoe hoger de throughput zal zijn.

responstijd Bij reactieve resource discovery is dit de tijd tussen het aanvragen van een service (*service request*) en het krijgen van een antwoord (*service reply*). Bij proactieve discovery kan de responstijd niet gebruikt worden omdat geen service requests verstuurd worden.

serverbelasting Bij voorkeur wordt een server gebruikt die weinig belast is.

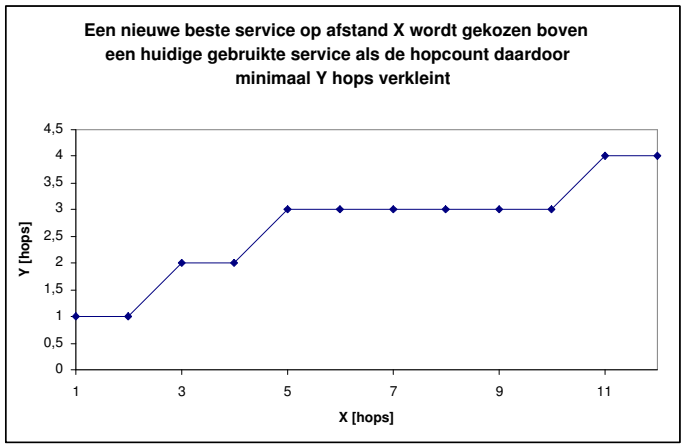
Deze criteria kunnen als volgt in een breuk samengenomen worden om de score van een service te bepalen. Hoe groter de score, hoe groter de *Quality Of Service*. Het belang van elke parameter wordt onderzocht in sectie 4.5.

$$score = \frac{(lifetime)\,(service\,attribuut)\,(server\,komt\,dichter)\,(infrastructured)}{(hopcount)\,(serverbelasting)\,(server\,verwijdert\,zich)\,(responstijd)}$$

Zo goed mogelijke services Veronderstel dat een applicatie *voortdurend* een *zo goed mogelijke* service verlangt. Dit kan voorvallen wanneer ze een single commodity service gebruikt. Dit geval kan gesimuleerd worden door de client de service te laten aanvragen met een gewenst minimaal service attribuut dat lager is dan alle service attributen van de aanwezige services in het ad hoc netwerk.

Vermijden van pingpongen tussen gelijkwaardige services Het is mogelijk dat een applicatie die een zo goed mogelijke service verlangt de keuze heeft uit een aantal providers die de service met ongeveer even goede score aanbieden. Wanneer de applicatie de service voor het eerst aanvraagt, zal de provider met de beste score geselecteerd worden. Wanneer er na verloop van tijd echter een provider is die de service met een hogere score begint aan te bieden (doordat hij zich tegen dan bijvoorbeeld dichter bij de client bevindt) zal de afweging moeten gemaakt worden wanneer men de overschakeling van de ene naar de andere provider moet maken. Deze overschakeling gebeurt best pas wanneer het scoreverschil tussen de momenteel gebruikte provider en de nieuwe betere provider aanzienlijk wordt. In het protocol wordt hiervoor een *treshold* van 20% gebruikt. Wat dit inhoudt voor een score van $\frac{1}{(1+hopcount)}$ wordt weergegeven in figuur 3.2. Zo wordt een nieuwe beste service op afstand 1 gekozen boven een huidige gebruikte

service op afstand 2 of hoger. Een nieuwe beste service op afstand 4 wordt echter niet gekozen boven een huidige gebruikte service op afstand 5. Dit gebeurt slechts wanneer de nieuwe beste service op afstand 3 zit.



Figuur 3.2: Treshold voor het kiezen van een betere service

3.6 Proactieve resource discovery

In dit geval wordt er gebruik gemaakt van *service advertisements*. Bij service advertisements pushen de service providers hun service informatie in het netwerk d.m.v. flooding. Dit kan op periodische basis gebeuren en/of ten gevolge van een gebeurtenis, zoals een nieuwe node die zich aanmeldt. De nodes die een service announcement ontvangen, slaan de service informatie op en forwarden het bericht naar de andere nodes in hun zendbereik nadat ze eventueel informatie hebben toegevoegd over de services die ze zelf aanbieden.

3.6.1 Protocolstructuren in een node

Service_found_table

Iedere node beschikt over een tabel waarin ze de informatie bijhoudt over de services die beschikbaar zijn in het netwerk. We zullen deze tabel verder de *service_found_table* noemen. De voornaamste velden waaruit deze *service_found_table* bestaat worden toegelicht in tabel 3.1.

Tabel 3.1: Voornaamste velden van een *service_found_table* entry

Veld	**Toelichting**
service	Geeft het type service aan (vb. FTP-server = 21)
provider	Het adres van de node die de service aanbiedt
service attribuut	Geeft de kwaliteit van de service aan (zie sectie 3.5)
arrival	Aankomsttijd van laatste authoritative (re)announcement
expiration	Eindtijd van de service
reannouncement interval	Geeft de tijd aan die passeert tussen 2 authoritative (re)announcements
hopcount	Afstand tot de service
approaching	Geeft aan of de service dichter komt
leaving	Geeft aan of de service zich verwijdert
infrastructured	Geeft aan of de serviceprovider infrastructured is
single commodity	Geeft aan of het om een single commodity service gaat

Service_request_table

Verder beschikt een node ook nog over een *service_request_table*. Wanneer bij een service aanvraag van een applicatie de node niet kan antwoorden met een service omdat er zich hierover geen informatie bevindt in de *service_found_table*, dan zal de aanvraag tijdelijk in de *service_request_table* geplaatst worden. Bij het ontvangen van een service announcement bericht zal telkens gekeken worden of de wachtende aanvragen in deze tabel kunnen beantwoord worden.

Sequence_number_list

De *sequence_number_list* houdt bij welke sequentienummers al gebruikt werden. Op die manier worden verschillende exemplaren van hetzelfde bericht herkend.

Opmerking In de tekst worden sequentienummers die worden toegekend door nodes aan pakketten, beschouwd als uniek voor het ganse netwerk. In de praktijk weet een node echter niet welke sequentienummers zijn toegekend door andere nodes en kan een sequentienummer dus slechts uniek zijn per node. Om toch uniciteit op netwerkniveau te kunnen waarborgen, zal men bijgevolg in een implementatie van het protocol gebruik moeten maken van een koppel van getallen: een sequentienummer en het adres van de node die het pakket heeft gecreëerd en het sequentienummer eraan heeft toegekend.

3.6.2 Inhoud van protocolpakketten

De service providers broadcasten *service announcements* om de informatie over hun services in het netwerk te verspreiden. De entries in zo'n pakket hebben velden zoals aangegeven in tabel 3.2.

3.6.3 Werking van een node

Aanbieden van een service

Nemen we het voorbeeld van figuur 3.3. Node 2 start een FTP-server op in de applicatielaag. Deze applicatie stuurt een bericht naar het resource discovery protocol met daarin o.a. de aangeboden service, het aangeboden service attribuut en de tijdsspanne dat de FTP-server (vermoedelijk) zal beschikbaar zijn.
Het resource discovery protocol in node 2 neemt vervolgens deze informatie op in haar *service_found_table*, en checkt of er in de node nog een applicatie aan het wachten was op deze

Tabel 3.2: Voornaamste velden van een *service_announcement* entry

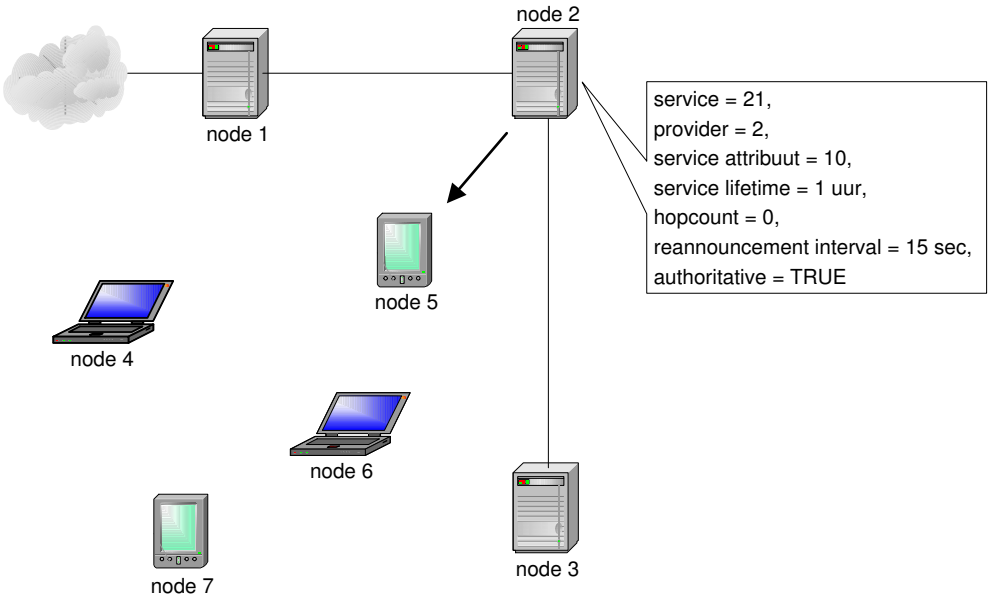
Veld	**Toelichting**
service	Geeft het type service aan (vb. FTP-server = 21)
provider	Het adres van de node die de service aanbiedt
service attribuut	Geeft de kwaliteit van de service aan (zie sectie 3.5)
service lifetime	Tijd dat de service nog beschikbaar is
reannouncement interval	Geeft de tijd aan die passeert tussen 2 authoritative (re)announcements
authoritative	Geeft aan of deze entry rechtstreeks van de provider of uit een cache komt
hopcount	Afstand tot de service
infrastructured	Geeft aan of de serviceprovider infrastructured is
single commodity	Geeft aan of het om een single commodity service gaat

service.

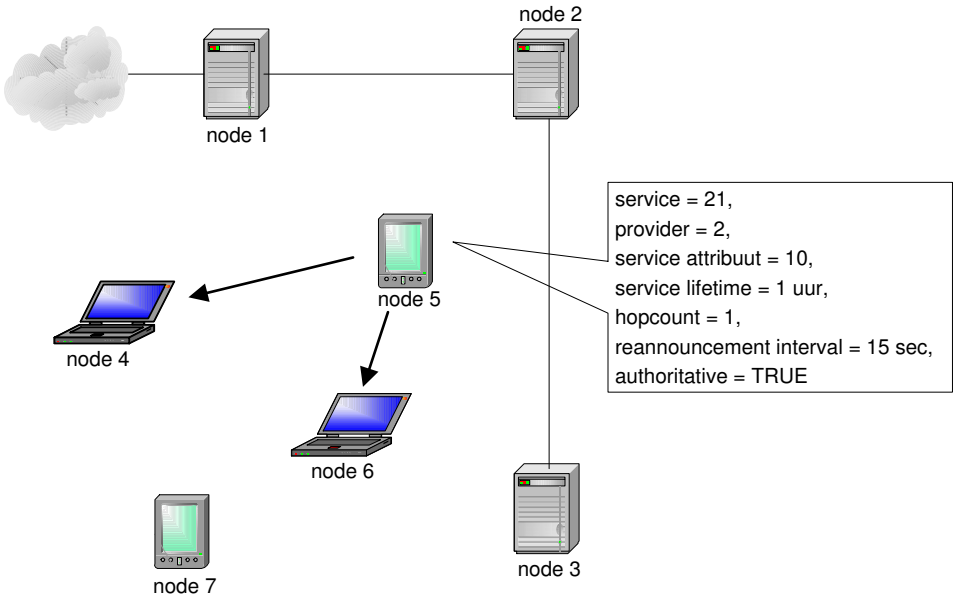
Daarna wordt de informatie over de FTP-server samen met de informatie over de andere, door de node gekende services in een service announcement bericht gestoken en gebroadcast naar de omgeving (d.i. node 5 in figuur 3.3). Er wordt een timer ingesteld die ervoor zorgt dat binnen de MAXNOANNOUNCEMENT tijd een reannouncement zal verstuurd worden. Announcements en reannouncements worden gegeven aan IP in de netwerklaag. Vanaf daar gebeurt de afhandeling van de verzending door IP in de netwerklaag en IEEE802.11 in de datalinklaag.

Doorsturen van service informatie

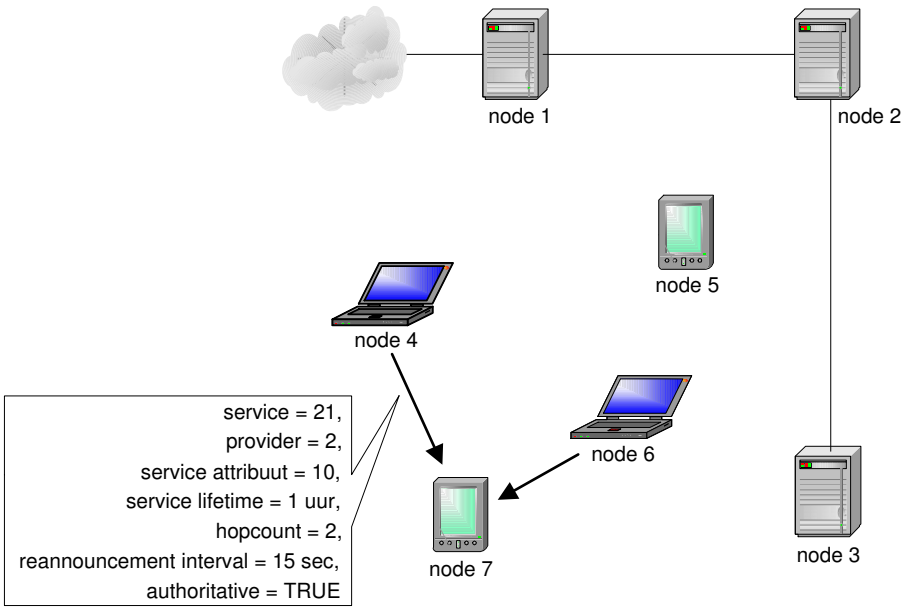
Het resource discovery protocol in node 5 ontvangt het service announcement van node 2. Eerst wordt het sequentienummer gecontroleerd om te zien of er al eerder een exemplaar van het service announcement verwerkt werd (zie figuur 3.6). Indien dit niet het geval is, zal de informatie uit het service announcement verwerkt worden. Hiertoe wordt voor iedere service die in het bericht staat het volgende gedaan : indien de service voor de aangegeven provider nog niet in de *service_found_table* staat, wordt de informatie over deze service in de tabel geplaatst, onafhankelijk van het feit of deze informatie uit een cache komt of direct van de provider. Indien de (service,provider)-combinatie wel al in de tabel staat, wordt deze entry enkel geüpdatet wanneer de informatie direct van de provider komt (zie paragraaf 'Correct gedrag bij caching'



Figuur 3.3: Node 2 biedt een FTP-service aan



Figuur 3.4: Forwarding door node 5



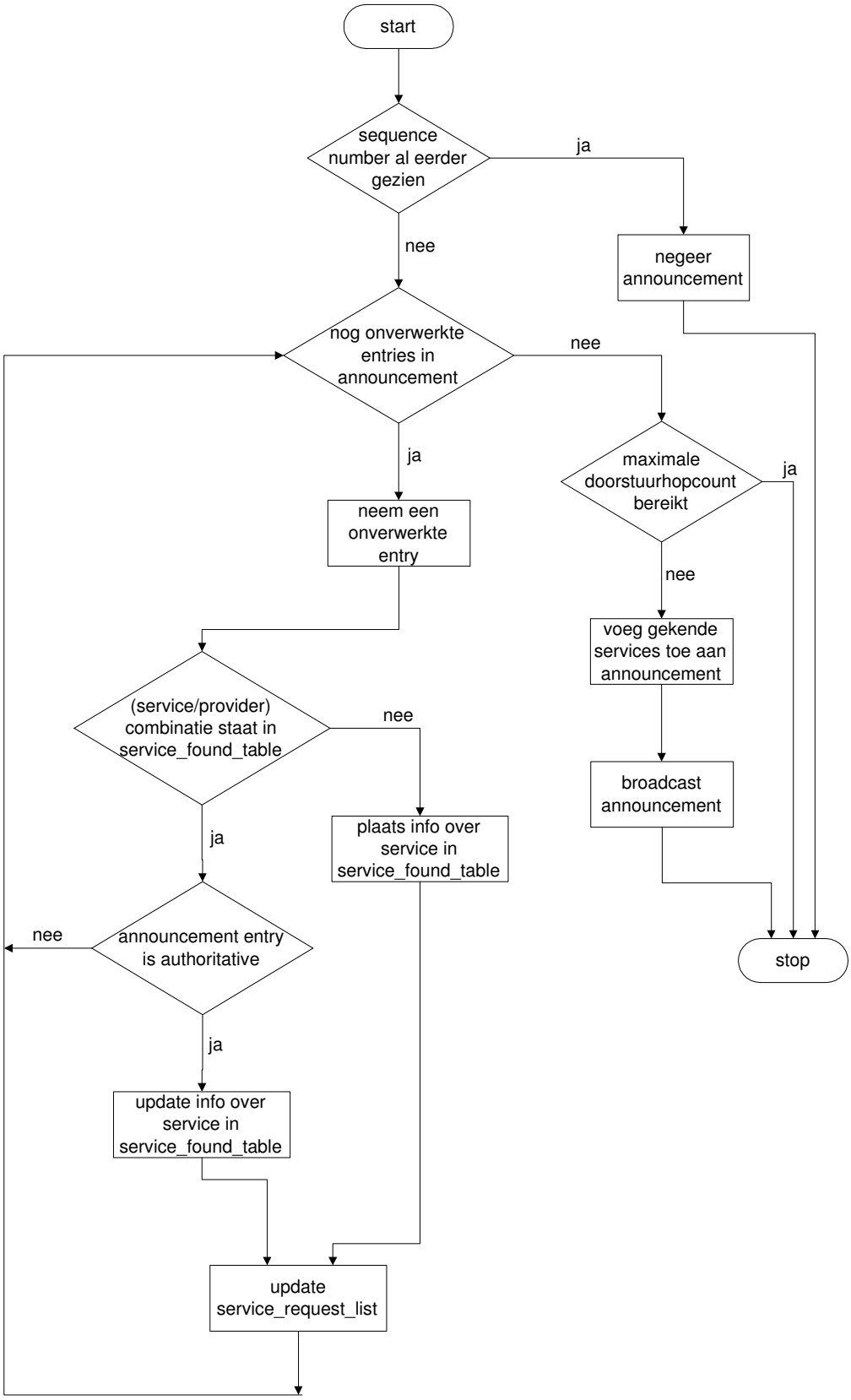
Figuur 3.5: Forwarding door nodes 4 en 6

in sectie 3.6.5).

Daarna wordt in de *service_request_table* nagegaan of er nog applicaties in node 5 wachten op een service die geadverteerd werd in het ontvangen service announcement.

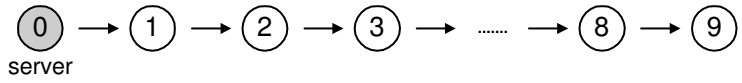
Indien de maximale doorstuurhopcount voor het bericht nog niet bereikt is, wordt het ontvangen service announcement vervolgens verder geforward via een broadcast (zie figuren 3.4 en 3.5). Hierbij worden ook de entries toegevoegd die wel al in de *service_found_table* stonden, maar nog niet in het service announcement.

Opmerking Men zou de forwarding van het service announcement een tijdje kunnen uitstellen om op die manier toe te laten dat de node in de tussentijd nog service announcements van andere nodes ontvangt die dan kunnen gebundeld worden tot 1 nieuw service announcement. Wanneer er zich meer dan 1 server in het netwerk bevindt kan zo het aantal berichten beperkt worden. Het nadeel is echter dat service informatie trager door het netwerk zal stromen. Stel bvb. dat server 0 in figuur 3.7 informatie wil verspreiden. Stel bovendien dat nodes 0 t/m 9 elk (onafhankelijk van elkaar) om de 15 seconden een service announcement verzenden waarin de service announcements verwerkt zijn die ze de laatste 15 seconden ontvangen hebben van andere nodes alsook de extra informatie uit hun eigen *service_found_tables*. In het slechtste geval zal de informatie van node 0 er dan 8 x 15 seconden over doen om in node 9 te geraken. Om dit sneller te laten gebeuren zou de announcementfrequentie dus moeten verhoogd worden maar



Figuur 3.6: Acties in een doorsturende node

dan verliest men het voordeel van de bundeling van berichten. Dit idee is dus beter bruikbaar in een statisch netwerk dan in een mobiel netwerk.



Figuur 3.7: Doorstroom van service informatie

Aanvraag van een service

Stel dat node 7 (op figuur 3.5) een FTP-client opstart in haar applicatielaag. Deze applicatie stuurt een bericht naar het resource discovery protocol, met daarin de vraag een FTP-server te vinden die voldoet aan het gewenste minimaal service attribuut. Indien er in de *service_found_table* een of meerdere services staan die hieraan voldoen dan wordt de beste hiervan geselecteerd en doorgegeven aan de FTP-client. Indien de aanvraag niet kan beantwoord worden met een service uit de tabel, dan kan de node niet op zoek gaan naar een geschikte service in het netwerk. Het enige wat ze kan doen is wachten tot er een service announcement passeert waarin de gewenste service opgenomen is. In dat geval wordt de aanvraag tijdelijk bijgehouden in de *service_request_table*, in de hoop dat binnen afzienbare tijd wel een geschikte service zal gekend zijn. Bij het ontvangen van een service announcement bericht zal telkens gekeken worden of de wachtende aanvragen in deze tabel kunnen beantwoord worden.

3.6.4 Detectie van servicemobiliteit

De detectie van servicemobiliteit gebeurt aan de hand van een aantal criteria zoals hopcountveranderingen, en nieuwe vermeldingen en expiraties van services in de *service_found_table* die iedere node bijhoudt. De detectie van mobiliteit gebeurt door iedere node apart door observatie van deze tabel. Bij de detectie wordt er gefocust op servicemobiliteit, niet op nodemobiliteit.

Hopcountveranderingen Iedere node observeert de hopcountverandering voor iedere service die opgenomen is in haar *service_found_table*. Hiertoe wordt om de MOBILITYCALCULATION_INTERVAL seconden de informatie over iedere service geëvalueerd. De hopcount van iedere service wordt vergeleken met de hopcount die deze service had bij de vorige controle. Indien dit verschil meer dan HOPCOUNTTRESHOLD bedraagt, wordt voor deze service beslist dat de serviceprovider in beweging is. In het veld *approaching* wordt bijgehouden of de

serviceprovider dichterbij komt, in het veld *leaving* wordt bijgehouden of de provider zich verwijdert. Wanneer er minstens MINMOBILITYCHANGES_HOPCOUNT services zijn waarvoor een aanzienlijke hopcountverandering wordt gedetecteerd, wordt door de node beslist dat het hele netwerk zich in een staat van hoge mobiliteit bevindt.

Verandering van de servicetabel Een node kan detecteren wanneer in haar servicetabel een nieuwe servernode verschijnt of wanneer een servernode haar laatste regel verliest. Een nieuwe service kan in de tabel verschijnen wanneer een reeds bestaande serviceprovider de node nadert, zodat de announcements de node bereiken. Een nieuwe service kan ook in de tabel verschijnen wanneer een service opgestart wordt in een provider die reeds binnen het bereik (rechtstreeks of multihop) van de node ligt. Omgekeerd verdwijnt een service uit de tabel wanneer de service ophoudt te bestaan of wanneer de serviceprovider buiten het berichtenbereik van de node gaat. Wanneer er minstens MINMOBILITYCHANGES_AVAILABILITY nieuwe servers in de servicetabel verschijnen of hun laatste serviceregel verliezen, wordt door de node beslist dat het netwerk zich in een staat van hoge mobiliteit bevindt. Deze evaluatie gebeurt eveneens om de MOBILITYCALCULATION_INTERVAL seconden.

Servicemobiliteit, geen nodemobiliteit Er werd voor gekozen om mobiliteit te detecteren door informatie over de services in het netwerk te observeren, en niet door nodes te observeren. Er kunnen immers ook nodes zijn die geen service aanbieden. Het routingprotocol kan deze nodes echter wel gebruiken om de trafiek te routeren, maar met betrekking tot de resource discovery zou de mobiliteit van deze nodes geen aanleiding mogen geven tot een andere beslissing over de mobiliteit wanneer de afstand tot een service niet aanzienlijk verandert. We hebben immers gekozen voor de gelaagde aanpak. Stel bijvoorbeeld dat een gebruiker zich aan een bepaalde kant van een weg bevindt. Wanneer deze gebruiker informatie in zijn tabel heeft staan over een service die zich in een node aan de andere kant van de weg bevindt, mogen passerende gebruikers geen invloed hebben op het resource discovery protocol wanneer deze gebruikers zelf geen service aanbieden.

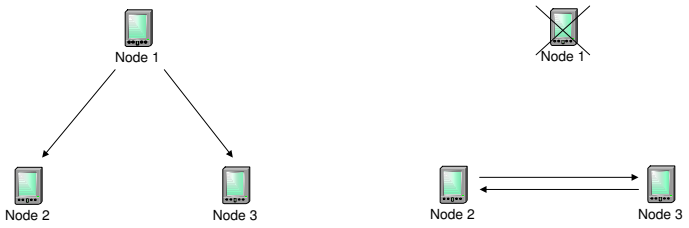
Simulatieresultaten Deze manier van mobiliteitsdetectie, waarbij een evaluatie gebeurt iedere MOBILITYCALCULATION_INTERVAL seconden, wordt in hoofdstuk 4 (sectie 4.5.2) aan simulaties onderworpen. Hieruit zal blijken dat deze manier van werken problemen oplevert o.a. omdat de snelheid van de nodes in een ad hoc netwerk onbekend is.

3.6.5 Aanpassingen voor ondersteuning van mobiliteit

Actueel houden van service informatie bij topologieveranderingen Bij ieder announcement wordt het reannouncementinterval met het bericht meegestuurd. Hierdoor weet een ontvangende node wanneer ze het volgende reannouncement kan verwachten. Het niet meer ontvangen van een aantal (PERMITTED_PACKET_LOSS) opeenvolgende reannouncements terwijl de stoptijd van de service nog niet bereikt werd, wijst er dan op dat de service te ver verwijderd is of niet meer bestaat.
Oorspronkelijk werd ook nog op de volgende manier geprobeerd om in te spelen op de mobiliteit: wanneer een providernode beslist dat het ad hoc netwerk zich in een staat van hoge mobiliteit bevindt, dan kan deze node het interval tussen 2 reannouncementberichten verkleinen, ervan uitgaande dat de informatie over mobiliteitsveranderingen dan sneller doordringt in het netwerk, ook al moeten hiervoor meer berichten verzonden worden. Deze werkwijze leverde echter niet het verhoopte resultaat op, enerzijds omwille van de complicaties bij de mobiliteitsdetectie (zie sectie 4.5.2), en anderzijds omwille van het feit dat iedere node apart beslist over hoge of lage mobiliteit.

Correct gedrag bij caching Gecachte informatie over een service mag geen voorrang krijgen op informatie die rechtstreeks van de servernode afkomstig is. Een node kan immers in haar *service_found_table* informatie hebben over services die op andere servers aangeboden worden (*caching*). Als deze node zelf een service aanbiedt, zal ze service announcements sturen, en in deze announcements zal ze ook de informatie over deze andere services steken. Deze informatie moet een lagere prioriteit hebben dan informatie die rechtstreeks afkomstig is van de aanbieder van de service. Dit wordt bewerkstelligd door het *authoritative* veld in de entries van het service announcement bericht. Dit veld geeft voor iedere service aan of de informatie *rechtstreeks* afkomstig is van de aanbieder. Maken we dit onderscheid niet, dan zijn ping-pong situaties mogelijk waarbij cachende nodes services in elkaars cache houden.

Nemen we als voorbeeld de situatie van figuur 3.8 (links). In dit voorbeeld biedt node 1 een FTP-service aan. Om deze service bekend te maken bij de andere nodes in het netwerk zal node 1 om de MAXNOANNOUNCEMENT tijd een service (re)announcement starten. Dit announcement bereikt node 2 en node 3, die elk de informatie uit dit announcement in hun *service_found_table* stoppen. Stel vervolgens dat node 1 verdwijnt (zie figuur 3.8 (rechts)). Hierdoor zullen in node 2 en 3 geen announcements meer toekomen die gestart werden door node 1. De



Figuur 3.8: Gebruik van authoritative announcements

informatie over de FTP-service in node 1 zit echter wel nog in de *service_found_table* van nodes 2 en 3, en deze nodes kunnen zelf reannouncements sturen (wanneer ze zelf services aanbieden of een service announcement van een andere provider forwarden). Op deze manier is het mogelijk dat node 2 en 3 voortdurend de informatie over node 1 aan elkaar geven terwijl node 1 al lang weg is. Door gebruik van het *authoritative* veld kan dit vermeden worden. Hierdoor weten node 2 en 3 bij ontvangst van informatie over de FTP-service in node 1 of deze informatie afkomstig is uit een cache of direct (rechtstreeks of multihop) van node 1. Als node 2 en node 3 geen reannouncements meer ontvangen die rechtstreeks van node 1 afkomstig zijn, zal de informatie over de FTP-service in node 1 verdwijnen uit de tabellen in node 2 en 3, zoals het hoort. Het cachen van informatie blijft echter nuttig omdat op die manier de informatie over de FTP-service meer frequent verspreid wordt en dus sneller in een nieuw opduikende node kan belanden.
Op basis van het *authoritative* veld zal een ontvangende node het volgende ondernemen wanneer ze een entry uit een service announcement verwerkt : Een *authoritative* entry uit een announcement pakket kan zowel informatie in de *service_found_table* plaatsen als updaten. Een *niet-authoritative* entry kan enkel informatie in de tabel plaatsen, niet updaten. Hierdoor zal de informatie over een service uit de tabel verdwijnen als van de provider van de service geen reannouncements meer ontvangen worden.

3.6.6 Service selectie

Hetzelfde bericht loopt via meerdere paden Een bericht (dit komt overeen met 1 sequentienummer) kan zich langs meerdere paden naar een bepaalde node begeven. Er kunnen dus verschillende exemplaren van eenzelfde bericht in een node aankomen. Om ervoor te zorgen dat het exemplaar met de kleinste hopcountindicatie gebruikt wordt, wordt bij iedere forwarding van het bericht een kleine vertraging toegevoegd. Hierdoor zal een bericht eerst alle bereikbare nodes op afstand 1 bereiken, vervolgens alle nodes op afstand 2, enz. Hierdoor wordt een realistische hopcountindicatie verzekerd.

Oorspronkelijk werd dit geval anders opgelost, nl. door telkens het eerste exemplaar van het bericht te gebruiken dat ontvangen werd. Het was dus steeds het exemplaar dat langs het snelste pad reisde dat gekozen werd, ervan uitgaande dat dit exemplaar de kleinste hopcountindicatie gaf. Dit was echter een verkeerde onderstelling: men moet ook rekening houden met o.a. de belasting van de doorsturende nodes waardoor berichten vertraagd kunnen worden.

Opmerking Aangezien resource discovery en routing gescheiden van elkaar gebeuren, moet er rekening mee gehouden worden dat de hopcount die aangegeven wordt door het resource discovery protocol tijdelijk licht kan verschillen van de hopcount die aangegeven wordt door het routing protocol. Dit is inherent aan de gelaagde aanpak wanneer niets aan het routingprotocol veranderd wordt. Hierdoor kan de verwachte throughput tijdelijk verschillen van de werkelijke throughput. Dit neemt niet weg dat de hopcount voor het resource discovery protocol de meest betrouwbare parameter blijkt te zijn om de service selectie op te baseren (zie sectie 4.5).

Tabel 3.3: Voornaamste velden van een *service_found_table* entry

Veld	**Toelichting**
service	Geeft het type service aan (vb. FTP-server = 21)
provider	Het adres van de node die de service aanbiedt
service attribuut	Geeft de kwaliteit van de service aan (zie sectie 3.5)
arrival	Aankomsttijd van laatste update
expiration	Eindtijd van de eigen service of van de informatie over de vreemde service (zie paragraaf 'lifetime' in sectie 3.7.8)
hopcount	Afstand tot de service tijdens huidig sequentienummer
previous hopcount	Afstand tot de service tijdens vorig sequentienummer
approaching	Geeft aan dat de service dichter komt
leaving	Geeft aan dat de service zich verwijdert
infrastructured	Geeft aan of de serviceprovider infrastructured is
single commodity	Geeft aan of het om een single commodity service gaat
sequence number	Sequentienummer van laatste update

3.7 Reactieve resource discovery

Bij reactieve resource discovery worden enkel berichten verstuurd wanneer een node de vraag van een applicatie naar een service niet kan beantwoorden aan de hand van de informatie in haar eigen tabel. In dit geval worden *request* pakketten gebroadcast in het netwerk tot er een *reply* pakket toekomt van een andere node. Een node kan een ontvangen *request* pakket al dan niet forwarden of (wanneer ze nuttige informatie kent) een *reply* pakket terugsturen.

3.7.1 Protocolstructuren in een node

Service_found_table

Iedere node beschikt over een tabel waarin zij de informatie bijhoudt over de services die zij kent in het netwerk. We zullen deze tabel verder de *service_found_table* noemen. De voornaamste velden waaruit deze *service_found_table* bestaat worden toegelicht in tabel 3.3.

Sequence_number_list

De *sequence_number_list* van een node houdt de sequentienummers bij van de reeds verwerkte *request* pakketten. Wanneer in een node een *request* pakket toekomt, wordt gecontroleerd of het sequentienummer van het pakket al toegevoegd is aan de *sequence_number_list*. Is dit het geval, dan wordt het pakket genegeerd. Is dit niet het geval, dan wordt het sequentienummer toegevoegd aan de lijst en wordt het pakket door de node verwerkt.

3.7.2 Inhoud van protocolpakketten

Het reactieve resource discovery protocol maakt gebruik van twee soorten pakketten: *request* pakketten en *reply* pakketten.

Request pakket

Wanneer een applicatie een bepaalde service aanvraagt, controleert de node eerst de eigen *service_found_table*. Wanneer deze geen geschikte service bevat, zal de node een *request* pakket broadcasten om alsnog de service in het netwerk te vinden. De voornaamste velden van een dergelijk pakket vindt u in tabel 3.4.

Reply pakket

Wanneer een node een *request* pakket ontvangt en informatie bezit in haar *service_found_table* over een of meerdere geschikte service(s), zal ze deze informatie bundelen in een *service_table*. Deze tabel wordt in een *reply* pakket gestopt en teruggezonden naar de aanvragende node. De voornaamste velden van de *service_table* vindt u in tabel 3.5.

3.7.3 Werking van het protocol

We leggen de werking van het protocol uit aan de hand van het voorbeeldnetwerk op figuur 3.9. Een FTP-clientapplicatie op node A wenst een connectie op te zetten met een FTP-server. Node B is zowel een FTP-server als een HTTP-server.

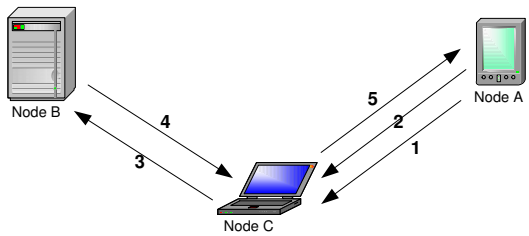
Node A vraagt de service aan De FTP-clientapplicatie geeft aan het resource discovery protocol van haar node de nodige informatie over de door haar gewenste service. Vooraleer de service in het netwerk te zoeken, onderzoekt het protocol de eigen *service_found_table* (zie sectie 3.7.1). Bevat deze tabel geen nuttige informatie, dan wordt de beslissing genomen de service in

Tabel 3.4: Voornaamste velden van een *request* pakket

Veld	**Toelichting**
service	Geeft het type van de gezochte service aan (vb. FTP-server = 21)
service attribuut	Geeft de minimale kwaliteit van de gezochte service aan (zie sectie 3.5)
requesting node	Het adres van de node die de service zoekt
maximale hopcount	Het maximaal aantal hops dat dit pakket ver mag reizen
hopcount	Het aantal hops dat dit pakket reeds gepasseerd is
sequence number	Sequentienummer van het request pakket

Tabel 3.5: Voornaamste velden van een *service_table* entry

Veld	**Toelichting**
service	Geeft het type service aan (vb. FTP-server = 21)
provider	Het adres van de node die de service aanbiedt
service attribuut	Geeft de kwaliteit van de service aan (zie sectie 3.5)
lifetime	Tijd dat de informatie in het pakket over de service geldig is
hopcount	Afstand tot de service
infrastructured	Geeft aan of de serviceprovider infrastructured is
single commodity	Geeft aan of het om een single commodity service gaat
sequence number	Sequentienummer van het pakket



Figuur 3.9: Werking van het protocol

het netwerk te zoeken.

Hiertoe wordt een *request* pakket (zie sectie 3.7.2) opgesteld met daarin de informatie over de gezochte service. Omdat het gaat om de eerste poging van de node om de service te vinden, wordt het veld *maximale hopcount* op '1' gezet. Het veld *hopcount* wordt altijd initieel op '0' gezet. Het pakket wordt gebroadcast (pakket 1 op fig. 3.9) en tegelijkertijd wordt een timer ingesteld. Deze zal ervoor zorgen dat na *'WAIT_TIME_FOR_DELIVERY'* milliseconden een balans wordt opgemaakt van de ontvangen services. Tenslotte wordt het sequentienummer, dat de node aan het pakket heeft toegekend, toegevoegd aan de *sequence_number_list* (zie sectie 3.7.1) van node A.

Node C ontvangt het eerste request pakket De eerste actie die node C onderneemt bij ontvangst van het *request* pakket is het vergelijken van het sequentienummer van het pakket met de *sequence_number_list*. In het voorbeeld heeft node C nog nooit een *request* pakket ontvangen met hetzelfde sequentienummer en zal beslist worden het pakket verder te verwerken.

Deze verwerking houdt in dat eerst de eigen *service_found_table* wordt onderzocht op nuttige informatie. Wordt dit niet gevonden, dan rest node C enkel nog het pakket te forwarden. Dit gebeurt echter enkel in het geval de waarde van het veld *maximale hopcount* strikt groter is dan de waarde van het veld *hopcount* + 1. Aangezien hier deze velden resp. de waarden 1 en 0 bevatten, wordt hier dus geen verdere actie ondernomen.

Node A maakt eerste balans op Na *'WAIT_TIME_FOR_DELIVERY'* milliseconden maakt node A een balans op van de ontvangen informatie door haar *service_found_table* te onderzoeken. In het voorbeeld zal er nog geen informatie bijgekomen zijn en zal de node haar zoektocht in het netwerk moeten verder zetten.

Hiertoe stelt node A een tweede *request* pakket op. De informatie over de gezochte service zal de-

zelfde zijn als in het eerste pakket, alsook de waarde '0' voor het *hopcount* veld. Wel zal het pakket een nieuw sequentienummer krijgen, dat eveneens aan de eigen *sequence_number_list* zal worden toegevoegd, en zal het veld *maximale hopcount* op '2' gezet worden. Ook hier wordt het pakket gebroadcast in het netwerk (pakket 2 op fig. 3.9) en wordt tegelijkertijd een timer ingesteld die ervoor zorgt dat na een vastgelegde tijd (*maximale hopcount* x *'WAIT_TIME_FOR_DELIVERY'* milliseconden) een nieuwe balans zal worden opgemaakt.

Node C ontvangt het tweede request pakket Daar het tweede *request* pakket een voor node C nog onbekend sequentienummer heeft, zal deze node dit pakket verwerken. De eigen *service_found_table* wordt opnieuw onderzocht, maar omdat er nog geen informatie is bijgekomen, levert dit niets op en wordt onderzocht of de node het pakket moet forwarden. De velden *maximale hopcount* en *hopcount* hebben nu resp. de waarden 2 en 0 en in tegenstelling tot het eerste pakket wordt dus beslist het tweede pakket wel verder te broadcasten (pakket 3 op fig. 3.9). Alvorens dit gebeurt, wordt het veld *hopcount* geïncrementeerd.

Node B ontvangt het request pakket Nadat het door node C geforward *request* pakket op zijn sequentienummer is gecontroleerd, wordt in de *service_found_table* van node B gezocht naar een geschikte service. De *service_found_table* van node B bevat o.a. informatie over de eigen FTP-server en wanneer deze server aan alle voorwaarden opgenomen in het *request* pakket voldoet, wordt beslist een *reply* pakket op te maken. Hierin verzamelt de node alle informatie over de FTP-server in een *service_table* (zie sectie 3.7.2). Deze *service_table* wordt bovendien uitgebreid met informatie over alle andere services die zijn opgenomen in de *service_found_table* van node B. De waarde van het veld *hopcount* wordt voor elk element van de *service_table* ingevuld met de waarde uit de *service_found_table*. In het geval van de FTP-server is dit '0'.
Wanneer nu dit pakket rechtstreeks naar node A verstuurd zou worden, zouden tussenliggende nodes niet in de mogelijkheid zijn de informatie uit de *service_table* op te nemen in de eigen *service_found_table*. Pakketten in tussenliggende nodes worden immers slechts verwerkt tot op het niveau van de netwerklaag, en het resource discovery protocol bevindt zich hierboven. Daarom zal het resource discovery protocol rechtstreeks het routing protocol aanspreken om de volgende hop voor bestemming 'node A' te kennen. Eenmaal deze informatie beschikbaar, wordt het *reply* pakket verstuurd naar deze *next hop* (pakket 4 op figuur 3.9).

Node C ontvangt het reply pakket Wanneer een node dienst doet als tussenliggende node voor een *reply* pakket, vindt een uitwisseling van informatie plaats tussen enerzijds het pakket en anderzijds de *service_found_table* van de node. Informatie uit de *service_found_table* van node C wordt verwerkt in de *service_table* van het pakket en node C slaat de informatie uit het pakket op in de eigen *service_found_table*. Voor de nieuwe entries van de *service_table* wordt het veld *hopcount* ingevuld met de waarde uit de *service_found_table*, de waarden van het veld *hopcount* van de oude entries worden geïncrementeerd.
Opnieuw wordt het routing protocol aangesproken om de volgende hop naar node A te kennen en wordt het pakket naar deze *next hop* verstuurd (pakket 5 op figuur 3.9).

Node A ontvangt het reply pakket Het *reply* pakket komt toe in node A, de node die een FTP-server heeft aangevraagd. Het veld *hopcount* wordt voor elk element van de *service_table* geïncrementeerd en de informatie uit de *service_table* wordt opgeslagen in de *service_found_table* van de node.

Node A maakt tweede balans op Na 2 maal *'WAIT_TIME_FOR_DELIVERY'* milliseconden maakt node A voor de tweede keer een balans op van de ontvangen informatie door haar *service_found_table* te onderzoeken. Nu wordt in de tabel informatie gevonden over de FTP-server die wordt aangeboden door node B. Wanneer deze FTP-server aan de minimale eisen van de vragende applicatie voldoet, wordt de informatie doorgespeeld aan de FTP-client. De applicatie kan vanaf dan zelfstandig een FTP-sessie opstarten met node B.

Opmerking Het door node C geforward *request* pakket zal uiteraard ook ontvangen worden door node A. Het sequentienummer van het pakket zal echter al opgenomen zijn in de *sequence_number_list* van node A en het pakket zal bijgevolg niet verwerkt worden. Om de figuur niet te overladen, werd dit niet afgebeeld op figuur 3.9.

3.7.4 Expanding ring search

Bij het ontwerp van een reactief resource discovery protocol is het belangrijk het aantal berichten zoveel mogelijk te beperken. Om dit doel te bereiken, maakt het protocol gebruik van *expanding ring search*.
Wanneer een node beslist een service aan te vragen in het netwerk, zal ze bij de eerste poging het veld *maximale hopcount* van het *request* pakket (zie tabel 3.4) op '1' zetten. Dit zorgt ervoor

dat een node die dit pakket ontvangt het pakket nooit zal forwarden.
Heeft de aanvragende node na *'WAIT_TIME_FOR_DELIVERY'* milliseconden geen geschikte service gevonden, dan wordt hetzelfde *request* pakket opnieuw verstuurd, maar wordt de waarde van het veld *maximale hopcount* geïncrementeerd. Nu zal een naburige node het pakket wel forwarden maar zullen haar buren dat niet meer doen...
Op deze manier zal zolang geen geschikte service gevonden is, de zoekruimte uitgebreid worden tot het veld *maximale hopcount* een maximumwaarde bereikt heeft. Heeft het zoeken dan nog geen resultaat opgeleverd, dan veronderstelt men dat het netwerk geen geschikte service kent.
Het is belangrijk op te merken dat de tijd die de node aan het netwerk geeft om informatie te verschaffen evenredig moet zijn met de zoekruimte. Daarom wordt de wachttijd van *'WAIT_TIME_FOR_DELIVERY'* milliseconden vermenigvuldigd met de waarde van het veld *maximale hopcount* van het verstuurde *request* pakket.

3.7.5 Uitbreiding van het protocol voor single commodity services

Sectie 3.7.3 beschrijft de werking van het protocol wanneer een applicatie een multi commodity service (vb. een FTP-server) aanvraagt. Eigen aan dit soort service is dat eenmaal een service afgeleverd is aan de applicatielaag, het werk van het resource discovery protocol voltooid is. Men zal tijdens de sessie immers niet meer van service veranderen. Om ook single commodity services (vb. HTTP-server via gateway, zie sectie 3.3) te kunnen aanbieden, zal het protocol echter moeten uitgebreid worden. Om ervoor te zorgen dat steeds de beste service gebruikt wordt tijdens een sessie, zal het resource discovery protocol op regelmatige tijdstippen het netwerk moeten onderzoeken.
Wanneer bijvoorbeeld een HTTP-client het resource discovery protocol aanspreekt om een gateway te vinden, zal het protocol allereerst zorgen voor een eerste oplossing. Ofwel kan dit dankzij informatie die reeds beschikbaar is in de eigen *service_found_table*, ofwel moet het daarvoor een zoektocht in het netwerk starten. Dit is volledig analoog aan een zoektocht voor een multi commodity service. Nieuw is echter dat op het moment van aanvraag van de applicatie een timer gestart wordt die vanaf dan om de *'UPDATE_TIME_FOR_BULK_SERVICE'* seconden een nieuwe zoektocht in het netwerk zal doen beginnen. Om het aantal berichten te beperken, wordt ook hier *expanding ring search* keer op keer toegepast.
Elke keer de applicatie nu een nieuwe pagina wenst op te vragen in het Internet, wordt eerst het adres van de beste gateway in het ad hoc netwerk opgevraagd. Het resource discovery pro-

tocol zal op basis van de dan beschikbare informatie telkens de beste service aan de toepassing afgeven. Dankzij de regelmatige updates heeft het protocol een goed beeld van de services in de omgeving van de node en kan het een gefundeerde beslissing nemen.

3.7.6 Sequentienummers en detectie van service mobiliteit

Sequentienummers worden in de eerste plaats gebruikt opdat een node een *request* pakket niet tweemaal zou verwerken (zie sectie 3.7.1). Maar het reactieve resource discovery protocol maakt eveneens gebruik van sequentienummers voor de detectie van service mobiliteit.
Wanneer een node een *service_table* opstelt voor een *reply* pakket, vult ze het veld *sequence number* in met het sequentienummer van het *request* pakket waar de node op reageert. Wanneer een *reply* pakket een tussenliggende node aandoet en er nieuwe services aan zijn *service_table* worden toegevoegd, dan wordt het sequentienummer van de oude entries eveneens aan de nieuwe entries van de tabel toegekend. Op deze manier wordt een zekere tijdsstempel aan de informatie over services gegeven.
Komt een *reply* pakket toe in de aanvragende node, dan wordt de informatie van de *service_table* verwerkt in de *service_found_table* van de node. Het is hier dat sequentienummers zullen worden aangewend om mobiliteit van services te detecteren.
Wanneer een nieuwe entry wordt toegevoegd aan de *service_found_table* (m.a.w. de node wist nog niet dat service x door node y werd aangeboden), wordt het veld *previous hopcount* op '-1' gezet, *approaching* en *leaving* op '0' en wordt het sequentienummer en de hopcount uit de *service_table* gekopieerd in resp. de velden *sequence number* en *hopcount* (zie tabel 3.3).
Weet de node wel al dat service x door node y wordt aangeboden, dan vergelijkt ze het sequentienummer van de *service_found_table* met dat van de nieuwe informatie uit de *service_table*.
Wanneer het sequentienummer uit het *reply* pakket groter is dan dat van de *service_found_table*, beschouwt men dit als een update van informatie: het veld *hopcount* wordt dan toegekend aan het veld *previous hopcount* en de velden *hopcount* en *sequence number* krijgen de waarden aangegeven in de *service_table*. Vervolgens vergelijkt met de velden *previous hopcount* en *hopcount*. Bevatten de velden dezelfde waarde, dan besluit men dat de node niet is bewogen t.o.v. de node die de service verleent en geeft men de velden *approaching* en *leaving* de waarde '0'. Is *hopcount* groter dan *previous hopcount*, dan wordt besloten dat de node die de service verleent zich verwijdert en wordt het veld *leaving* op '1' gezet, *approaching* op '0'. Is *hopcount* kleiner dan *previous hopcount*, dan krijgen de velden *approaching* en *leaving* resp. de waarden '1' en

'0', de service komt dichter.

Zijn de sequentienummers van de entry in de *service_found_table* en van de corresponderende entry in de *service_table* gelijk, dan vergelijkt men de waarden van het veld *hopcount* uit beide tabellen. Is de hopcount uit de *service_table* kleiner dan die uit de *service_found_table* van de node, dan beschouwt men de oude hopcount als niet correct: een node antwoordde wel eerder op het *request* pakket, maar gaf hierbij niet de lengte van het optimale pad. Bijgevolg wordt in de *service_found_table* de waarde van het veld *hopcount* geactualiseerd. Vervolgens worden de velden *approaching* en *leaving* opnieuw berekend door de nieuwe hopcount te vergelijken met de (onveranderde) waarde van het veld *previous hopcount*.

3.7.7 Wisselwerking met het routing protocol

Wanneer een node een *reply* pakket heeft opgesteld, stuurt ze dit niet rechtstreeks naar de aanvragende node. Wil men immers dat de nodes op weg naar die aanvragende node informatie kunnen uitwisselen met het pakket, dan moet het pakket expliciet verstuurd worden naar die tussenliggende nodes. Hiervoor zal de antwoordende node het routing protocol aanspreken om de eerste hop naar de bestemming te kennen. Vervolgens wordt het pakket naar deze *next hop* verstuurd. Hetzelfde algoritme wordt gebruikt in een tussenliggende node om het pakket verder te versturen richting aanvragende node.

Wordt er gebruik gemaakt van een proactief routing protocol, dan hoeft het resource discovery protocol enkel de tabelstructuren van het routing protocol aan te spreken om de *next hop* naar een bestemming te kennen. Door het proactief karakter van het routing protocol kan immers ervan uitgegaan worden dat die informatie steeds gekend is.

Maakt men gebruik van een reactief routing protocol zoals AODV, dan is het vinden van de *next hop* minder vanzelfsprekend. De node die het pakket een hop verder wenst te versturen, zal eerst het routing protocol aanspreken om te weten of een *next hop* naar de bestemming gekend is. Is dit het geval, dan kan uiteraard het pakket meteen verstuurd worden. Maar aangezien we hier werken met een reactief routingprotocol, is het helemaal niet verzekerd dat deze informatie beschikbaar is! Kent de node geen route naar de aanvragende node, dan zal het resource discovery protocol het routing procotol zo aansturen dat de zoektocht naar een route wordt gestart. Vervolgens worden de tabellen van het routing protocol op regelmatige basis gecontroleerd door het resource discovery protocol. Eenmaal voldoende informatie is vergaard, kan het pakket probleemloos naar de *next hop* verstuurd worden.

Het is belangrijk op te merken dat de zoektocht naar een node van een reactief routing protocol veel tijd in beslag neemt en de aanvragende node dus typisch veel langer zal moeten wachten om *reply* pakketten te ontvangen dan wanneer men gebruik maakt van een proactief routing protocol. Hiermee zal sterk rekening gehouden moeten worden wanneer men de waarde van de parameter *'WAIT_TIME_FOR_DELIVERY'* bepaalt. Dit is de tijd dat een node het netwerk de kans geeft om haar informatie te verschaffen over geschikte services alvorens een balans op te maken en de beste service aan de vragende applicatie te geven. Voor de bepaling van de optimale waarde van deze parameter, wordt verwezen naar sectie 4.2.

Opmerking Men zou zich m.b.t. het reactieve routing protocol de vraag kunnen stellen of een nieuwe zoektocht in het netwerk wel nodig is elke keer een *next hop* onbekend is. Het is immers mogelijk dat de node reeds aan het zoeken is naar een route voor de aangevraagde bestemming, maar dat dit onbekend is voor het resource discovery protocol. Hier stuit men echter op de beperkingen van een gelaagde aanpak.
Elke keer het resource discovery protocol een zoektocht laat starten in het netwerk, brengt dit enerzijds een reeks broadcasts met zich mee en anderzijds een of meerdere unicast berichten als antwoord.
Als alternatief zou het protocol bij een ongekende *next hop* een dummy pakket kunnen opmaken en dit versturen naar de finale bestemming van het te versturen *reply* pakket. Werken we bijvoorbeeld met AODV, dan zou dit dummy pakket, wanneer geen route bekend is, gebufferd worden en wordt daarna een zoektocht naar de bestemming gestart. Het voordeel bij deze aanpak is nu dat wanneer het resource discovery protocol voor een tweede keer een *reply* pakket naar dezelfde bestemming wenst te versturen en er nog geen route bekend is, geen nieuwe zoektocht in het netwerk zal gestart worden. Er zal enkel een tweede dummy pakket aangemaakt worden. AODV merkt op dat er geen route gekend is en stopt het tweede dummy pakket eveneens in de buffer. Het protocol merkt op dat de node reeds een route in het netwerk aan het zoeken is en er wordt geen nieuwe zoektocht gestart. Op deze manier zal er voor een reeks te versturen *reply* pakketten slechts één reeks broadcasts nodig zijn en enkele unicasts als antwoord. Bij de geïmplementeerde aanpak echter moet men dit aantal vermenigvuldigen met het aantal opeenvolgende te versturen *reply* pakketten. Op het eerste zicht lijkt de alternatieve aanpak dus veel voordeliger. Men mag echter het versturen van de dummy pakketten niet vergeten. Het gaat hier immers om evenveel unicast berichten als er *reply* pakketten verstuurd moeten worden. We-

tende dat unicast berichten veel meer tijd en energie vragen in het IEEE802.11 MAC-protocol dan broadcast berichten, zal de alternatieve aanpak dus wellicht niet veel voordeel opleveren. De twee mogelijkheden zijn samengevat in tabel 3.6.

Tabel 3.6: Aantal berichten door een onbekende *next hop* in een reactief routing protocol

Communicatiemethode	**Aantal berichten tijdens een periode dat de next hop naar de bestemming onbekend is (x = aantal opeenvolgende te versturen reply pakketten)**
Route-zoektocht per reply pakket	x broadcasts + x unicasts
Gebruik van dummy-pakketten	1 broadcast + 1 unicast + x unicasts van dummies

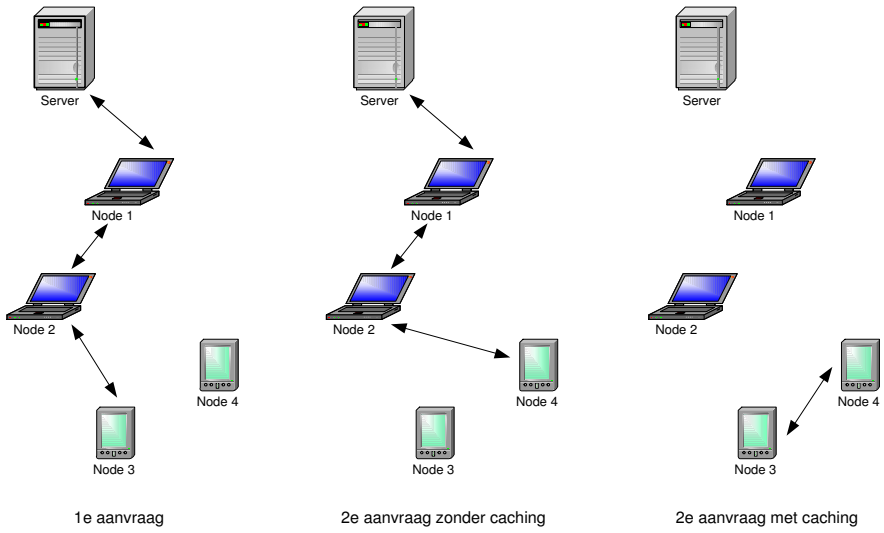
Nog een andere mogelijkheid zou zijn om in het resource discovery protocol een lijst bij te houden van bestemmingen waar het routing protocol op dat moment een route naar aan het zoeken is voor het resource discovery protocol. Er wordt dan enkel opnieuw een route gezocht wanneer de bestemming nog niet in deze lijst is opgenomen. Op deze manier zal echter ook een zoektocht gestart worden wanneer het routing protocol voor een transportlaagprotocol naar dezelfde bestemming al aan het zoeken is! In dit geval levert dit dus slechts weinig voordeel op. We kunnen inzien dat deze inefficiëntheden te maken hebben met het feit dat resource discovery en routing aparte protocollen zijn in verschillende lagen van de protocolstack. Op die manier kan bijvoorbeeld resource discovery niet rechtstreeks opzoekingen doen in de buffers van het routing protocol. Wanneer we beide protocollen zouden combineren tot één geïntegreerd protocol, zijn efficiëntere methodes denkbaar.

3.7.8 Caching en aanpassingen voor ondersteuning van mobiliteit

Caching Wanneer een node antwoordt op een vraag van een andere node, zal ze behalve informatie over de aangevraagde service ook informatie over alle andere door haar gekende services opslaan in het *reply* pakket. Komt een *reply* pakket in een tussenliggende node terecht op weg naar de bestemming, dan slaat deze node alle informatie op die het pakket bevat en breidt ze het pakket uit met de door haar gekende informatie. In de bestemming tenslotte, wordt de ontvangen informatie eveneens een tijdlang opgeslagen.

Op deze manier wordt service informatie in het netwerk verspreid waardoor het aantal benodigde berichten beperkt wordt en applicaties sneller een geschikte service zullen ontvangen van het

resource discovery protocol. Dit wordt geïllustreerd aan de hand van figuur 3.10.



Figuur 3.10: Voordeel van caching

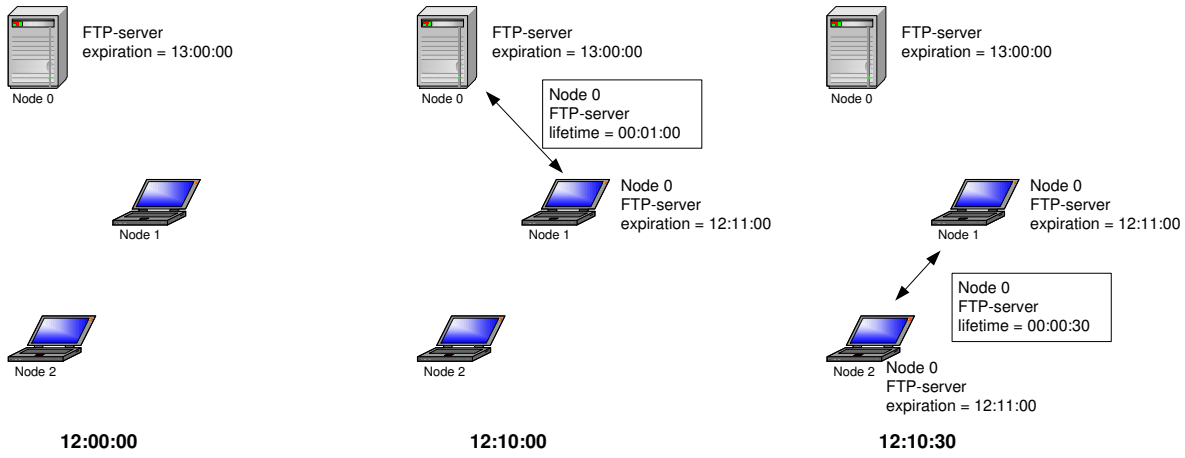
Initieel weet alleen de server zelf dat hij een FTP-serverapplicatie aan het lopen heeft en zijn de *service_found_tables* van de andere nodes leeg. Eerst vraagt node 3 een FTP-server aan. Na de eerste poging zal node 3 nog tweemaal zijn zoekruimte moeten vergroten alvorens de server antwoordt en de node een connectie kan opzetten met de server. Daarna vraagt node 4 een FTP-server aan. Implementeren we geen caching, dan zal opnieuw de zoekruimte tweemaal moeten worden uitgebreid alvorens nuttige informatie gevonden wordt. Maakt caching wel deel uit van het protocol, dan moet men de zoekruimte niet uitbreiden en vindt men meteen informatie over de geschikte service. Zo worden heel wat berichten uitgespaard en kan het resource discovery protocol in node 4 veel sneller een service afgeven aan de applicatie.

Lifetime Wegens het mobiel karakter van het netwerk is het uitermate belangrijk dat informatie over services beperkt geldig is in de tijd. Elke update van informatie geeft immers slechts een momentopname van het netwerk weer. In enkele minuten tijd kan het netwerk in die mate veranderen dat het zinloos is om een service te selecteren op basis van oude informatie. Om dit te vermijden, maken we gebruik van de velden *lifetime* in de *service_table* van het *reply* pakket en *expiration* in de *service_found_table*.

In het proactief resource discovery protocol geeft het veld *expiration* het moment aan waarop de server stopt met zijn service aan te bieden. In het reactief protocol is de betekenis van het veld

subtieler. Wanneer het gaat om een service die de node zelf verleent, bevat het veld *expiration* dezelfde informatie, maar wanneer de service door een andere node dan haarzelf wordt verleend (en dus ontvangen werd via een *reply* pakket), geeft *expiration* het moment aan waarop de informatie over de service in de *service_found_table* ongeldig wordt.

Wanneer een *reply* pakket wordt opgemaakt, wordt het veld *lifetime* ingevuld met het minimum van de volgende twee tijden: *expiration* min de actuele tijd of *LIFETIME_OF_OLD_INFO*, een maximumtijd die meestal op 1 minuut wordt ingesteld. Komt deze informatie toe in een node, dan wordt deze *lifetime* opgeteld bij de huidige kloktijd en wordt het resultaat ingevuld in het veld *expiration* van de *service_found_table*. Op deze manier krijgt dit veld zijn correcte waarde en blijft informatie in het netwerk maximum 1 minuut geldig.



Figuur 3.11: Beperkt geldig houden van informatie

We illustreren het concept aan de hand van het voorbeeld op figuur 3.11.

Wanneer node 1 om 12.10 u een FTP-server aanvraagt, zal het veld *lifetime* in het *reply* pakket ingevuld worden met 1 minuut, het minimum van 50 minuten (= *expiration* - actuele tijd) en 1 minuut (= *LIFETIME_OF_OLD_INFO*). Bij aankomst van het pakket in node 1 wordt de actuele tijd opgeteld bij de waarde van het veld *lifetime* om de *expiration* te bepalen: 12.11 u.

Wanneer node 2 30 seconden na node 1 een FTP-server aanvraagt, zal node 1 meteen antwoorden. Nu wordt 30 seconden in het veld *expiration* ingevuld, het minimum van 30 seconden (= *expiration* - actuele tijd) en 1 minuut (= *LIFETIME_OF_OLD_INFO*). In node 2 komt hierdoor opnieuw 12.11 u in het veld *expiration* te staan.

Uit het voorbeeld blijkt dat informatie nooit langer dan 1 minuut in het netwerk geldig zal blijven. Men moet echter opmerken dat omwille van de eenvoud in het voorbeeld geen rekening

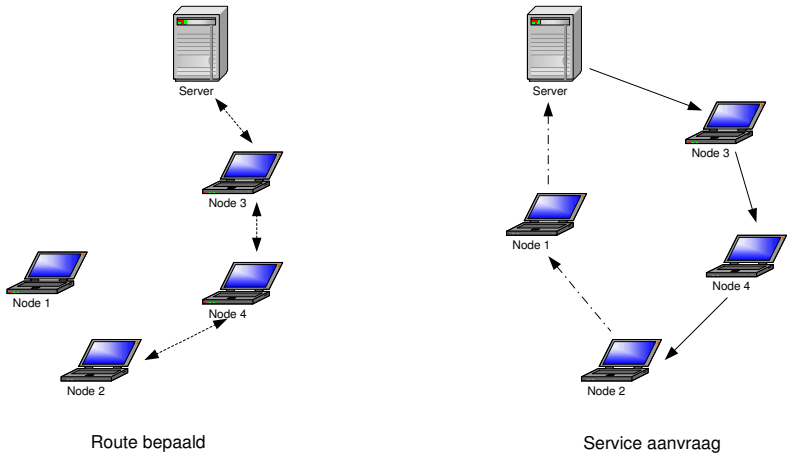
gehouden wordt met de propagatietijd van de pakketten. In de praktijk zal daarom informatie iets langer dan 1 minuut geldig blijven.

3.7.9 Service selectie

Controle van de expiration Bij de selectie van de beste service zal het resource discovery protocol enkel deze services beschouwen waarvan de *expiration* nog niet verlopen is. De betekenis van dit veld in acht genomen, betekent dit dat een eigen service enkel kan geselecteerd worden wanneer deze nog wordt aangeboden en een vreemde service enkel geselecteerd kan worden wanneer de informatie in de *service_found_table* omtrent de service nog geldig is.

Hopcount Een van de essentiële elementen van het reactieve resource discovery protocol is het gebruik van het *hopcount* veld. In een *request* pakket wordt een *hopcount* bijgehouden om de zoekruimte te beperken. In een *reply* pakket gebeurt dit ten behoeve van de service selectie: het geeft een maat voor de afstand van de service naar de node die een service wenst te selecteren. Beide *hopcounts* worden echter volledig onafhankelijk van elkaar bepaald.
In een *request* pakket wordt initieel het veld *hopcount* op '0' gezet. Elke keer een pakket ontvangen wordt, wordt het veld geïncrementeerd en het pakket eventueel doorgestuurd. Wanneer een node een *request* pakket kan beantwoorden, wordt de waarde van dit veld echter niet herbruikt: in het *reply* pakket krijgt *hopcount* initieel de waarde uit de *service_found_table*. Elke tussenliggende node zal de *hopcount* incrementeren en het pakket doorsturen tot in de aanvragende node alwaar de *hopcount* een laatste keer verhoogd wordt.
Men kan zich nu terecht de vraag stellen waarom men bij creatie van het *reply* pakket niet meteen de waarde uit het *request* pakket optelt bij die uit de *service_found_table*. Dan zou geen enkele andere node nog het veld moeten incrementeren en kan het zonder aanpassingen opgeslagen worden in de *service_found_table* van de aanvragende node. Het *reply* pakket zal echter niet altijd dezelfde route gebruiken als het *request* pakket en het is nu net de route van het *reply* pakket die gebruikt zal worden voor het eerste deel van de service delivery en dus van belang is bij de service selectie.

Figuur 3.12 illustreert het feit dat de weg van het *request* pakket niet altijd dezelfde zal zijn als die van het *reply* pakket. Stel dat er al een route bepaald werd tussen node 2 en de server. Kort nadien vraagt node 2 een FTP-server aan. Doordat het netwerk intussen licht gewijzigd is, wordt de server al bereikt wanneer node 1 het *request* pakket forwardt. Om het *reply* pakket



Figuur 3.12: Soms volgt het reply pakket een andere route dan het request pakket

te versturen, zal de server echter nog de route gebruiken die het routing protocol kort daarvoor had gevonden. In de *service_found_table* van node 2 wordt genoteerd dat de FTP-server zich 3 hops ver van de node bevindt. Aangezien de route van het *reply* pakket aanvankelijk gebruikt zal worden voor de delivery, is dit de correcte waarde voor het *hopcount* veld. Via de optimale weg op dit moment is de server slechts 2 hops ver, maar aangezien niet de optimale weg gebruikt wordt, is dit irrelevant.

Wait_time_for_delivery Eenmaal een *request* pakket gebroadcast werd in het netwerk, wacht de node *maximale hopcount* maal *'WAIT_TIME_FOR_DELIVERY'* milliseconden om een balans op te maken van de ontvangen services. Voor de prestatie van het reactief resource discovery protocol is het van cruciaal belang een goede waarde voor *'WAIT_TIME_FOR_DELIVERY'* te kiezen.

Wanneer de parameter te klein wordt gekozen, zal te snel een balans worden opgemaakt. In dat geval kan het gebeuren dat een *reply* pakket reeds onderweg is, maar nog niet is aangekomen in de node. Vindt de node geen andere geschikte service in haar tabellen, dan zal dit voor gevolg hebben dat de zoektocht wordt verdergezet met een vergrote zoekruimte, wat heel veel nutteloze overhead veroorzaakt. Is de zoekruimte reeds maximaal, dan concludeert de node zelfs verkeerdelijk dat er geen geschikte service in het netwerk aanwezig is! Zijn er wel alternatieven in de tabellen te vinden, dan wordt de service die nog onderweg is gediscrimineerd en wordt misschien niet de beste service gekozen die in de zoekruimte te vinden is.

Het is echter duidelijk dat *'WAIT_TIME_FOR_DELIVERY'* niet oneindig groot gekozen mag worden. Een grote waarde betekent een langere wachttijd voor de service vragende applicatie.

In sectie 4.2 trachten we de optimale waarde voor deze parameter te bepalen aan de hand van experimenten.

Hoofdstuk 4

Simulatieresultaten

4.1 Algemeen

De simulaties werden uitgevoerd in de GloMoSim-netwerksimulator [17]. De bevindingen uit de simulaties kunnen opgedeeld worden in een aantal topics :

- De schaalbaarheid naar het aantal nodes, het aantal requests, het aantal servers, en de maximale berichthopcount.
- De snelheid van de resource discovery (tijd die verstrijkt alvorens met de delivery kan gestart worden).
- De criteria die gebruikt kunnen worden voor service selectie.
- De detectie van mobiliteit.
- De invloed van de routing op de resource discovery.

Deze topics zullen in dit hoofdstuk verder uitgewerkt worden.

GloMoSim De simulaties werden uitgevoerd in de GloMoSim-netwerksimulator [17][18]. GloMoSim is een library-gebaseerde, sequentiële en parallelle simulator voor hoofdzakelijk draadloze netwerken. Deze simulator is ontworpen als een verzameling library-modules die elk een specifiek protocol van de protocolstack simuleren. De protocolstack voor draadloze netwerken is onderverdeeld in lagen, elk met hun eigen API. Protocollen in de éne laag kunnen met protocollen in een andere laag communiceren via deze API's. Deze modulaire implementatie laat toe om meerdere protocollen in eenzelfde laag op een consistente manier met elkaar te vergelijken, en

laat ook toe om gemakkelijk een nieuw protocol toe te voegen. Voor de ontwikkeling van de library werd gebruik gemaakt van Parsec [19], een C-gebaseerde simulatietaal voor sequentiële en parallelle uitvoering van *discrete-event* simulatiemodellen. Ons resource discovery protocol werd in deze taal geïmplementeerd. Voor meer informatie over GloMoSim en Parsec verwijzen we naar [17] en [19].

Opmerking Wanneer geen expliciete nodenummering opgegeven wordt, gebeurt de nummering van de nodes in GloMoSim steeds van links onder in het netwerk naar rechts boven in het netwerk. In een netwerkje van 3 x 3 nodes bijvoorbeeld zal de nummering gebeuren zoals getoond in figuur 4.1.

6	7	8
3	4	5
0	1	2

Tabel 4.1: Nodenummering in GloMoSim

4.2 Reactieve resource discovery: bepaling wait_time_for_delivery

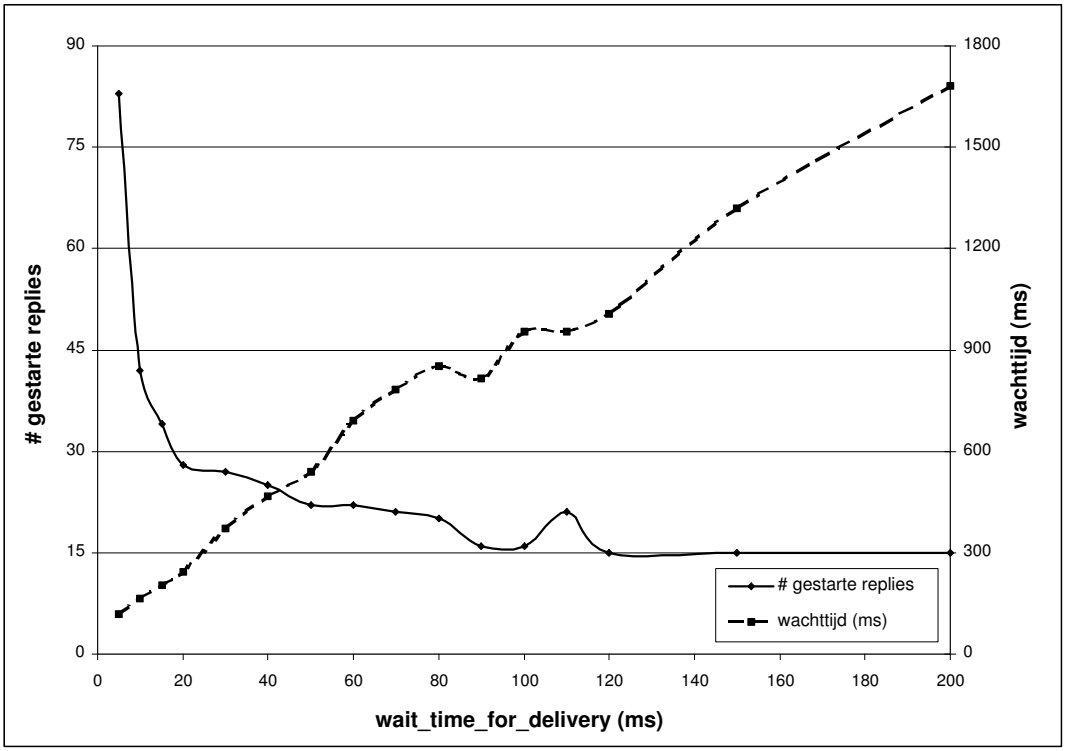
In sectie 3.7.9 werd beschreven wat de gevolgen kunnen zijn van een slechte keuze voor de parameter *WAIT_TIME_FOR_DELIVERY* van het reactieve resource discovery protocol. Hoe kleiner de parameter gekozen wordt, hoe sneller het protocol, maar hoe groter de kans wordt dat replies gemist worden.
Om een optimale waarde te bepalen voor de *WAIT_TIME_FOR_DELIVERY* parameter werden een aantal simulaties gedaan. Hierbij werden de volgende parameters vastgehouden:

parameter	**waarde**
nodeplaatsing	vast, uniform, 36 nodes (6 x 6)
grootte netwerk	1200 m x 1200 m
simulatietijd	20 min
aantal service requests	15
aantal servers	1

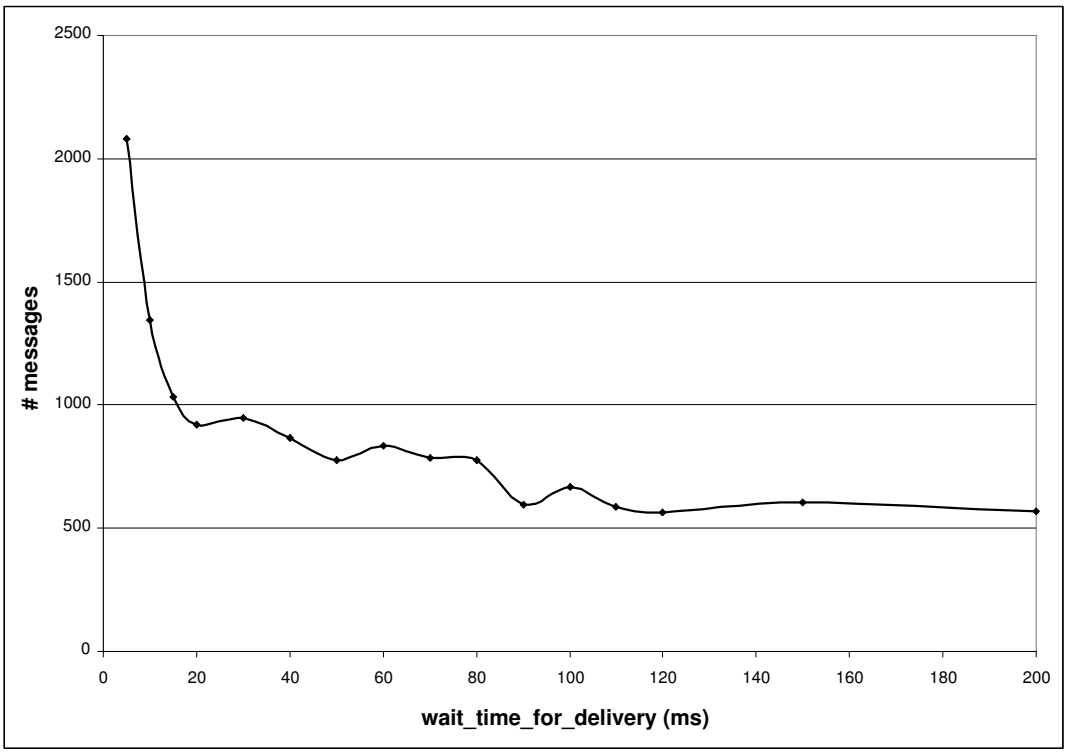
We lieten de waarde van *WAIT_TIME_FOR_DELIVERY* variëren tussen twee grenzen die afhankelijk waren van het gebruikte routing protocol. Werd een reactief routing protocol gebruikt zoals AODV, dan werd 5 ms als ondergrens en 200 ms als bovengrens genomen. Wanneer een proactief routing protocol zoals WRP werd gebruikt, werden 3 en 30 ms als grenzen gekozen. De figuren 4.1 en 4.3 tonen, resp. voor AODV en WRP, enerzijds het aantal *reply* pakketten dat door de nodes wordt gecreëerd om te antwoorden op een vraag vanuit het netwerk en anderzijds de gemiddelde wachttijd voor een aanvragende node alvorens een geschikte service gevonden te hebben en de zoektocht te stoppen. Het aantal verstuurde resource discovery berichten wordt afgebeeld op figuren 4.2 en 4.4.
Tijdens elke simulatie werden 15 services aangevraagd. Door ervoor te zorgen dat er voldoende tijd verstreek tussen 2 service requests, kon telkens enkel de server in het netwerk de aanvraag beantwoorden. Telkens een node een zoektocht voor een nieuwe aanvraag startte, was alle gecachte informatie in het netwerk immers niet meer geldig. Dit betekent dat voor een goeie waarde van de *WAIT_TIME_FOR_DELIVERY* parameter het aantal gestarte *reply* pakketten eveneens 15 zou moeten bedragen. Zijn het er meer, dan wijst dit erop dat *WAIT_TIME_FOR_DELIVERY* te klein is. Er wordt dan immers zo vroeg een balans opgemaakt dat het eerste *reply* pakket nog niet is aangekomen en een nieuwe, onnodige en meer uitgebreide, zoektocht wordt gestart die

aanleiding geeft tot extra gecreëerde *reply* pakketten. Deze kunnen zowel verstuurd worden door de server als door een node die wel al de tijd had gekregen het eerste *reply* pakket te cachen. Dit zorgt uiteraard voor extra berichten en de gemiddelde wachttijd voor de aanvragende node verhoogt eveneens onnodig.

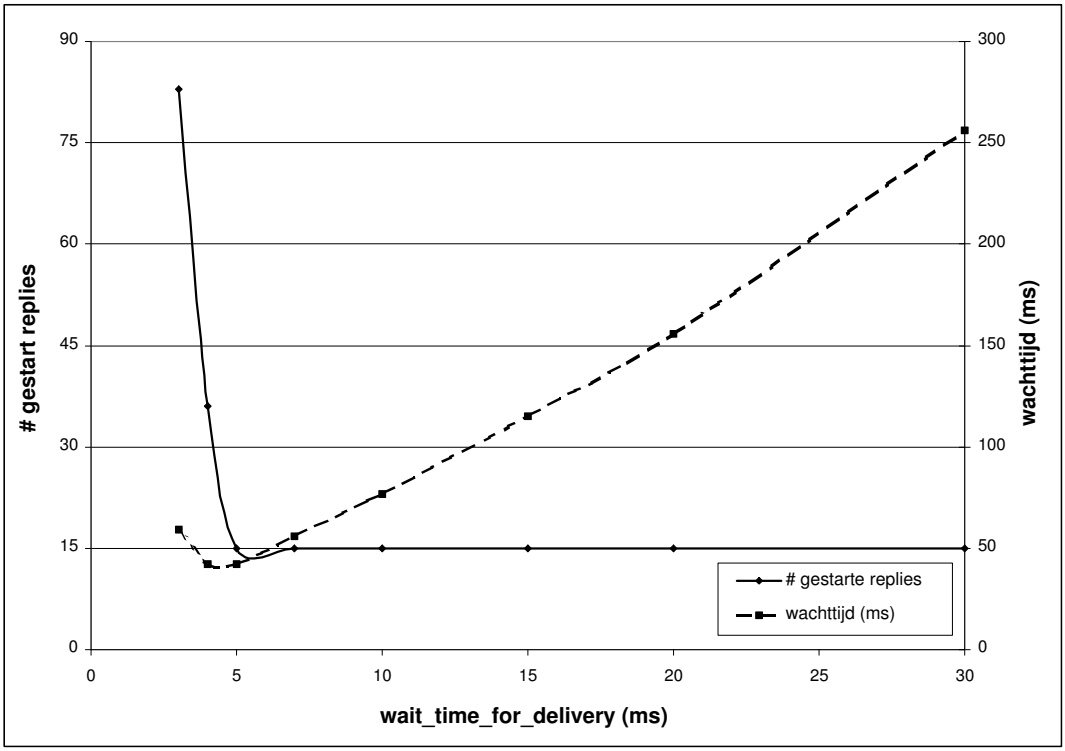
Er werd besloten *120 ms te nemen als WAIT_TIME_FOR_DELIVERY parameter wanneer reactieve routing gebruikt wordt.* We zien immers dat slechts vanaf deze waarde er niet meer dan 15 replies gestart worden. Mochten we de parameter groter kiezen, dan zou het aantal gestarte replies hetzelfde zijn, maar de gemiddelde wachttijd veel groter dan nodig. Analyseren we de grafieken voor proactieve routing, dan zien we dat er reeds vanaf 5 ms niet meer dan 15 replies gestart worden. Daarom kozen we ervoor *5 milliseconden in te vullen als waarde voor de WAIT_TIME_FOR_DELIVERY parameter wanneer WRP als routing protocol gebruikt wordt.* Dit betekent dat het resource discovery protocol 24 keer sneller een service in het netwerk kan vinden wanneer proactieve i.p.v. reactieve routing gebruikt wordt. Dit is uiteraard te wijten aan het feit dat, vooraleer een reply verstuurd kan worden, er bij reactieve routing eerst nog een route naar de aanvragende node gezocht moet worden. *Wanneer snelheid belangrijk is, moeten we daarom altijd met proactieve routing werken.*



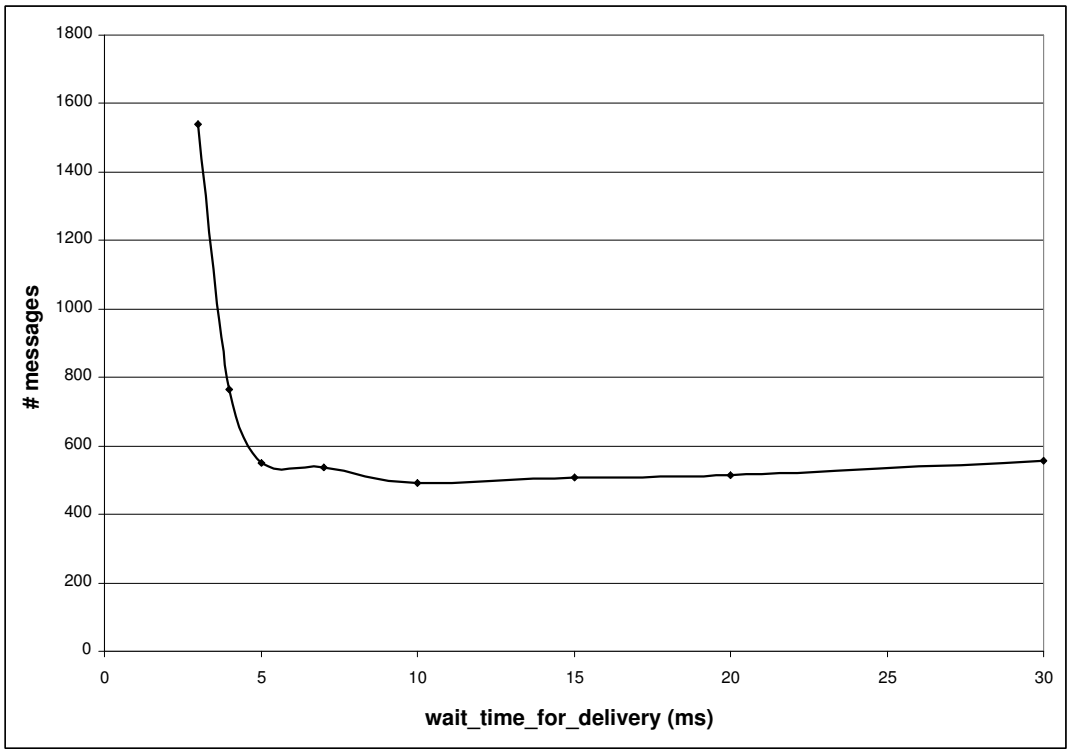
Figuur 4.1: Aantal gestarte replies en gemiddelde wachttijd i.f.v. wait_time_for_delivery voor AODV



Figuur 4.2: Aantal berichten i.f.v. wait_time_for_delivery voor AODV



Figuur 4.3: Aantal gestarte replies en gemiddelde wachttijd i.f.v. wait_time_for_delivery voor WRP



Figuur 4.4: Aantal berichten i.f.v. wait_time_for_delivery voor WRP

4.3 Schaalbaarheid

Er werd onderzocht hoe de overhead van de protocollen evolueert bij een toename van het aantal nodes in het netwerk, bij een toename van het aantal nodes die een aanvraag voor een service maken en bij een toename van het aantal nodes die een service aanbieden. Ook werd gekeken wat de invloed is van de maximale berichthopcount (het maximaal aantal hops dat een bericht mag reizen door het netwerk).
De schaalbaarheid werd onderzocht door het aantal berichten te evalueren dat nodig is om de resource discovery uit te voeren. Anderzijds werd ook de energie geëvalueerd die verbruikt wordt door de nodes.

Energie in GloMoSim De door een node verbruikte energie wordt in GloMoSim in de radiolaag bepaald. Dit is de onderste laag van de protocolstack. Iedere transmissie over de radio interface verhoogt de verbruikte energie proportioneel met de grootte van het verzonden pakket. Naast de resource discovery heeft o.a. ook de routing (= zoeken van een pad) en de delivery (bvb. FTP-trafiek) invloed op de verbruikte energie. Een node verbruikt ook energie proportioneel met de tijd dat haar ontvanger in werking is (= signalen kan ontvangen). De energiebepaling gebeurt in het bestand */radio/radio_accnoise.pc*.

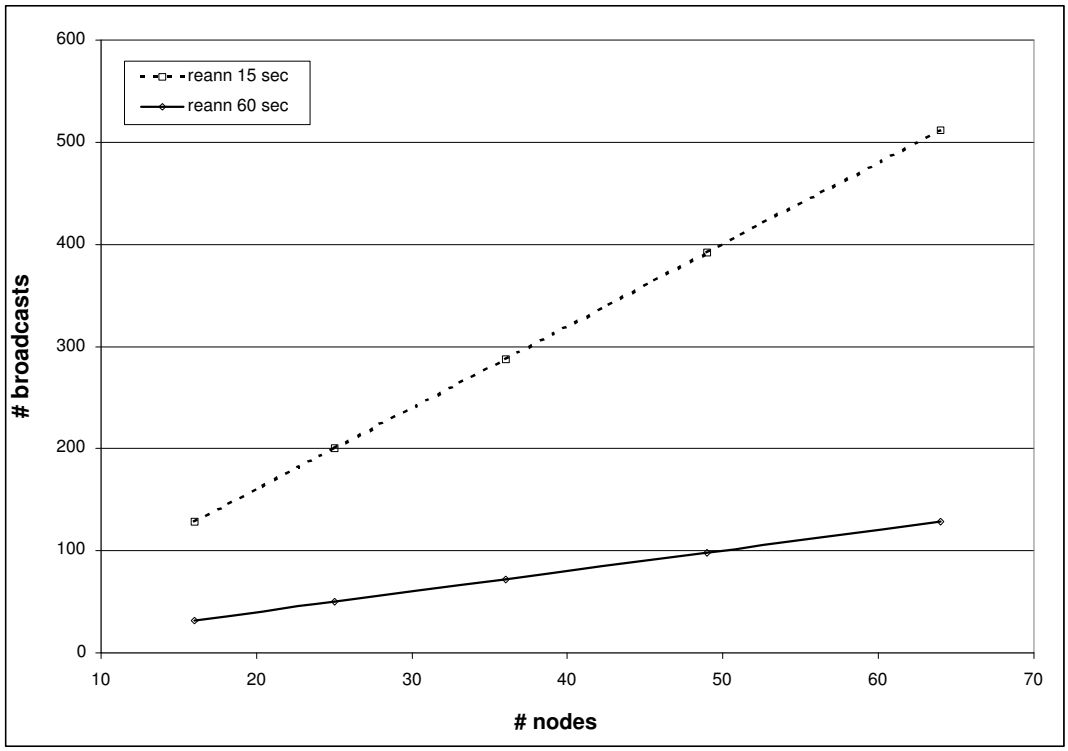
4.3.1 Schaalbaarheid naar het aantal nodes

Proactieve resource discovery

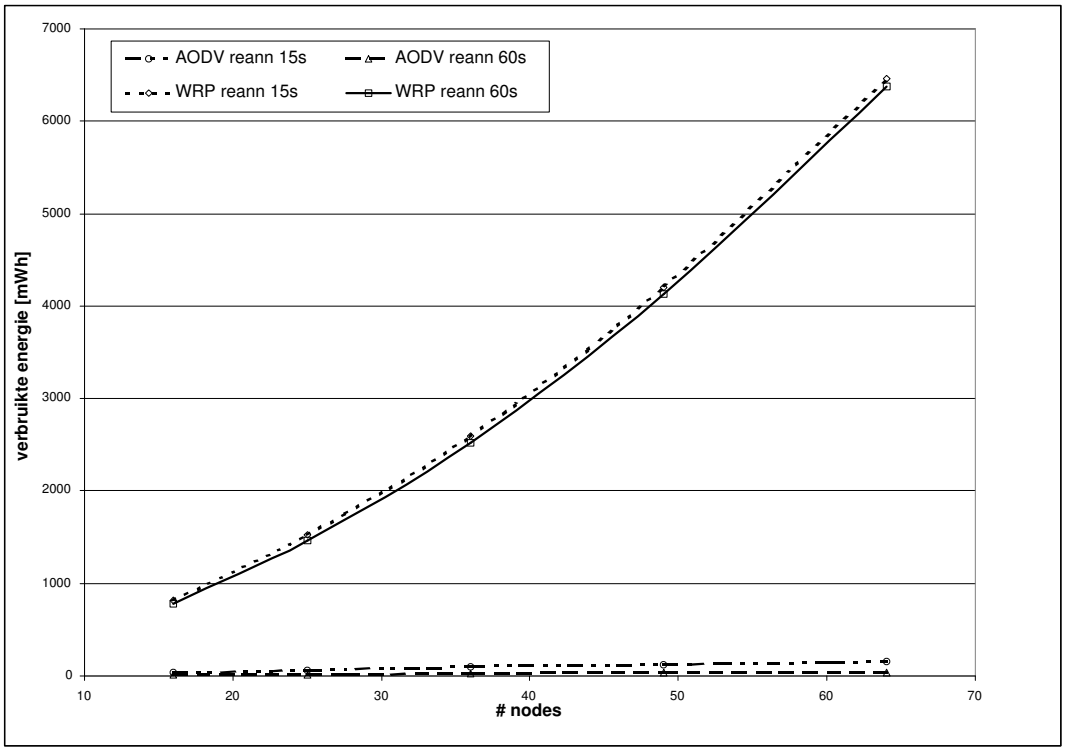
Bij deze simulatie werden de volgende parameters vast gehouden :

parameter	**waarde**
nodeplaatsing	vast, in grid
simulatietijd	2 min
reannouncement interval	15 en 60 s
aantal service requests	geen

De grootte van het netwerk werd gevarieerd van 800 meter x 800 meter (4 x 4 nodes) tot 1600 meter x 1600 meter (8 x 8 nodes). Er werd telkens 1 server gekozen in het midden van het netwerk (zie tabel 4.2).



Figuur 4.5: Aantal broadcasts i.f.v. het aantal nodes



Figuur 4.6: Verbruikte energie i.f.v. het aantal nodes

Tabel 4.2: De gebruikte servers in de simulatie

aantal nodes	**servernode**
16 (4 x 4)	5
25 (5 x 5)	12
36 (6 x 6)	14
49 (7 x 7)	25
64 (8 x 8)	27

Figuur 4.5 geeft weer hoe het aantal broadcasts verandert wanneer het netwerk groter wordt. *Het aantal broadcasts stijgt lineair met het aantal nodes.* Wanneer het netwerk zo groot wordt dat er berichten zijn die wegens het bereiken van hun maximale berichthopcount niet in alle nodes terechtkomen, zal de grafiek afvlakken. Bovendien stijgt het aantal broadcasts omgekeerd evenredig met de grootte van het reannouncement interval.

Figuur 4.6 toont de invloed van het aantal nodes op de verbruikte energie in het geval dat WRP en AODV gebruikt worden als onderliggend routingprotocol. Aangezien er in de simulatie geen enkele service aangevraagd wordt, is er ook geen delivery. Dit betekent dat *in het geval van AODV de stijging van de energie volledig toe te schrijven is aan de berichten ten behoeve van de proactieve resource discovery* (de berichten raken verder in het netwerk). In het geval van WRP is de energiestijging zowel toe te schrijven aan het resource discovery protocol als aan de proactieve routing. Deze laatste component heeft de overhand aangezien de energie verbruikt voor de resource discovery dezelfde is als wanneer AODV gebruikt wordt. WRP veroorzaakt dus een aanzienlijke energiestijging. Wanneer het aantal nodes stijgt, moet WRP immers meer routes onderhouden. *Een proactief routingprotocol is dus niet schaalbaar naar grotere aantallen nodes. In het geval van WRP is het aandeel van de proactieve resource discovery in de verbruikte energie dus klein in vergelijking met het aandeel van de proactieve routing.*

Reactieve resource discovery

Aangezien in het gebruikte scenario geen services worden aangevraagd, zal het reactief resource discovery protocol geen berichten versturen en zal het dus ook geen bijdrage leveren tot de verbruikte energie. Gebruiken we bovendien AODV als routing protocol, dan zal er helemaal geen energie verbruikt worden omdat er evenmin delivery is. Met WRP als routing protocol zal er wel energie verbruikt worden voor de routing en zal deze stijgen wanneer het netwerk groter

wordt.

Hieruit kunnen we besluiten dat *het reactief resource discovery protocol schaalbaar is naar het aantal nodes*. Het enige effect dat een groter netwerk op dit protocol kan hebben is dat het mogelijk wordt dat een service verder in het netwerk wordt gezocht. Dit zou natuurlijk resulteren in meer berichten.

4.3.2 Schaalbaarheid naar het aantal requests

Bij deze simulatie werden de volgende parameters vast gehouden :

parameter	**waarde**
nodeplaatsing	vast, in grid, 36 nodes (6 x 6)
grootte netwerk	1200 m x 1200 m
simulatietijd	2 min
reannouncement interval	60 s
aantal servers	1 (node 14)

Het aantal requests werd gevarieerd van 0 tot 7 :

aantal requests	**vragende clientnode(s)**
0	-
1	0
2	0, 5
3	0, 5, 24
4	0, 5, 24, 34
5	0, 5, 24, 34, 29
6	0, 5, 24, 34, 29, 3
7	0, 5, 24, 34, 29, 3, 17

Bij een stijgend aantal service requests blijft het aantal service announcements dat door de proactieve resource discovery verzonden wordt constant (zie figuur 4.8). *Proactieve discovery is dus schaalbaar naar het aantal service requests.* Figuur 4.8 toont ook het aantal berichten voor reactieve discovery met en zonder caching. Zonder caching blijft het aantal berichten stijgen bij een stijgend aantal requests. Wanneer wel caching gebruikt wordt, is het echter niet nodig

dat iedere vragende client het antwoord rechtstreeks van de server krijgt. Nodes die de service aanvraag ontvangen en er een antwoord op kennen, mogen hier eveneens op antwoorden, wat het aantal berichten minder snel doet stijgen na enkele aanvragen. De caching kan gebruikt worden wanneer de aanvragen elkaar voldoende snel opvolgen. Zo kan van de gecachte informatie gebruik gemaakt worden wanneer ze nog niet geëxpireerd is. *Reactieve discovery is dus schaalbaar naar het aantal requests wanneer caching gebruikt wordt.*

Figuur 4.7 toont dat de verbruikte energie bij een stijgend aantal requests weinig afhankelijk is van de resource discovery. Het geval *#requests = 0* komt overeen met het geval *#nodes = 36* in figuur 4.6. De routing zorgt voor 2 banden in de grafiek: *het is steeds zo dat proactieve routing meer energie verbruikt dan reactieve routing.* Tabel 4.3 geeft de oorzaken van energie*stijging* weer voor de verschillende gevallen van resource discovery en routing.

Tabel 4.3: Oorzaken van energiestijging bij stijgend aantal service requests

Resource discovery / routing	**Energiestijging veroorzaakt door...**
proactief op proactief	service delivery (FTP-trafiek)
proactief op reactief	routing & service delivery
reactief op proactief	resource discovery & service delivery
reactief op reactief	resource discovery, routing &service delivery

De stijging van verbruikte energie is dus voornamelijk te wijten aan het stijgend aantal deliveries. De helling van de curves is niet constant omwille van de verschillende afstanden tussen client en server waardoor voor de éne delivery meer energie vereist is dan voor de andere.

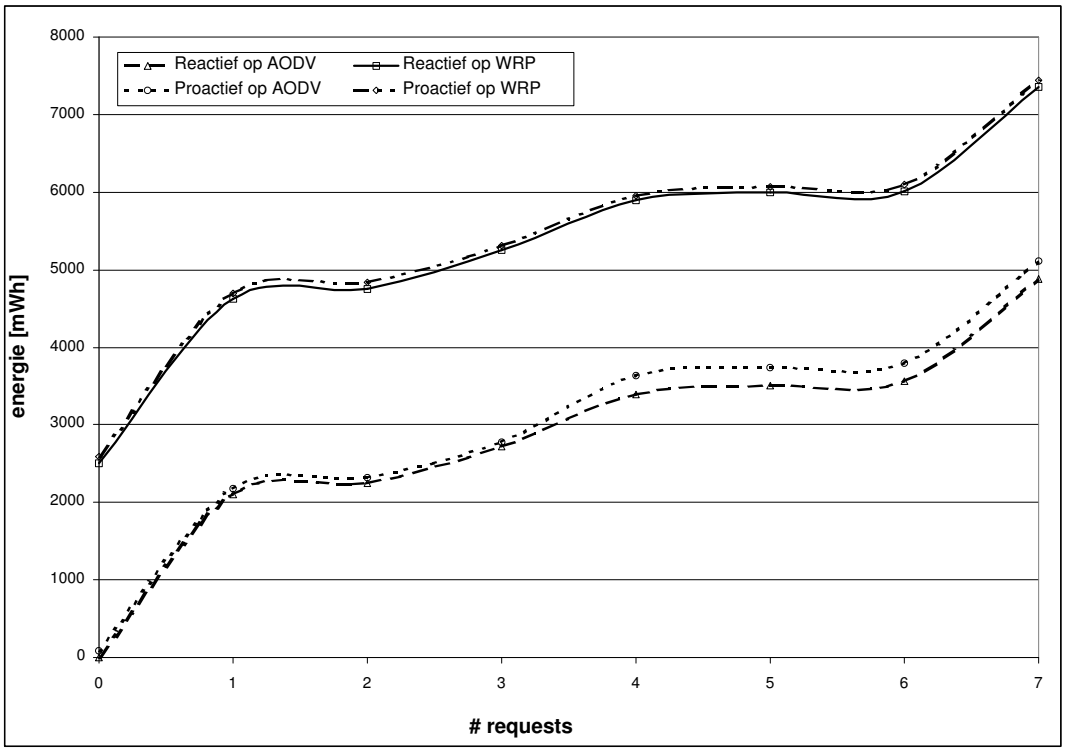
4.3.3 Schaalbaarheid naar het aantal servers

Proactieve resource discovery

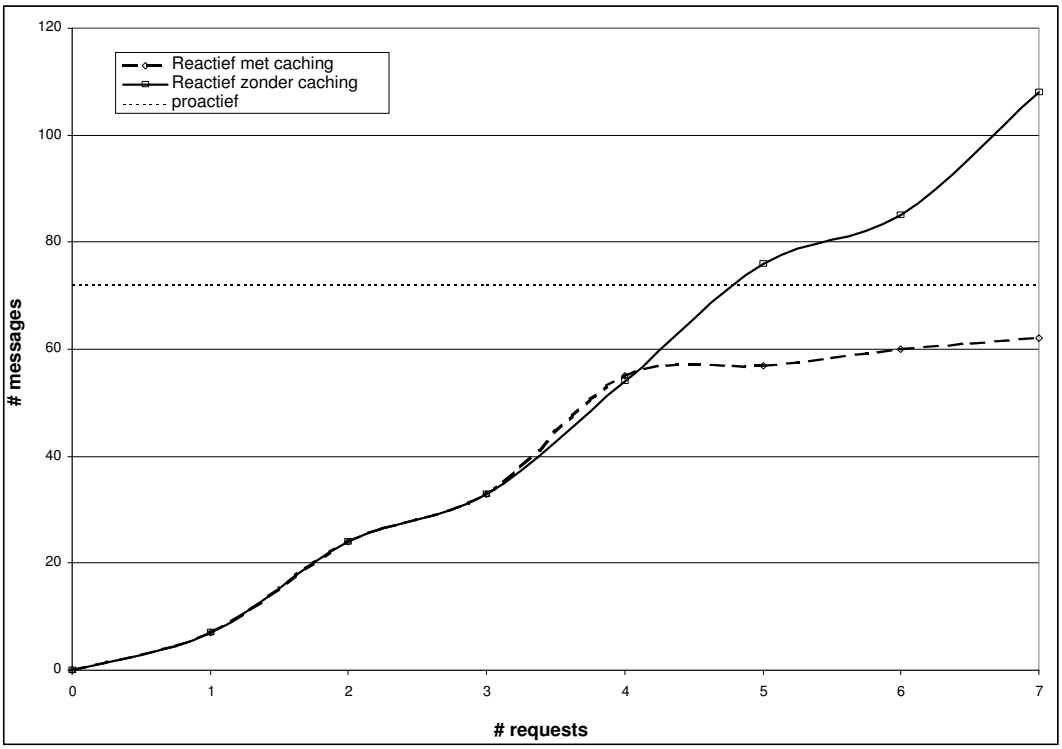
De parameters die bij deze simulatie vast werden gehouden, worden getoond in tabel 4.4. Het aantal servers werd gevarieerd van 1 tot 7 (zie tabel 4.5).

Het aantal broadcasts stijgt lineair met het aantal servers (zie figuur 4.9). Deze grafiek zal niet afvlakken wanneer het aantal servers aanzienlijk groter wordt.

Figuur 4.10 toont de evolutie van de verbruikte energie bij een stijgend aantal servers. Het geval *#servers = 1* komt overeen met het geval *#nodes = 36* in figuur 4.6. We merken een aanzienlijke stijging van de energie bij een stijgend aantal servers, zowel bij gebruik van AODV als WRP. De stijging is volledig te wijten aan de proactieve resource discovery. AODV wordt



Figuur 4.7: Verbruikte energie i.f.v. het aantal service requests



Figuur 4.8: Aantal berichten i.f.v. het aantal service requests

Tabel 4.4: Vaste parameters bij de simulatie

parameter	**waarde**
nodeplaatsing	vast, in grid, 36 nodes (6 x 6)
grootte netwerk	1200 m x 1200 m
simulatietijd	2 min
reannouncement interval	15 s
aantal service requests	geen

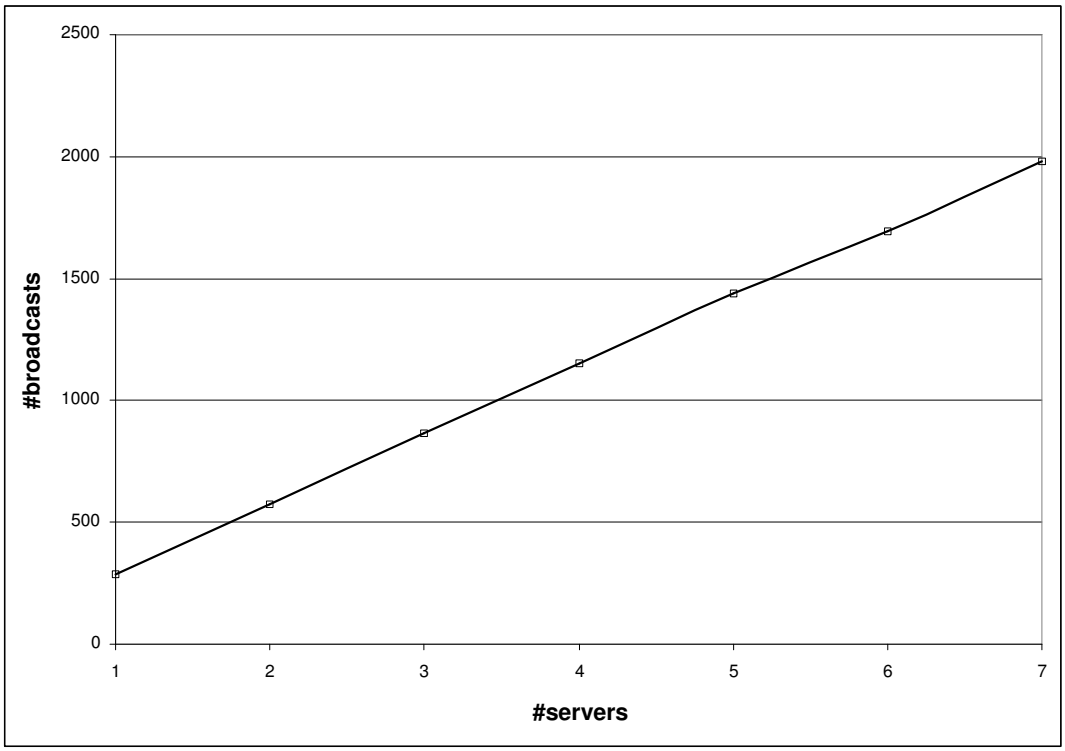
Tabel 4.5: Gebruikte servers in de simulatie

aantal servers	**servernode(s)**
1	14
2	14, 1
3	14, 1, 5
4	14, 1, 5, 24
5	14, 1, 5, 24, 34
6	14, 1, 5, 24, 34, 17
7	14, 1, 5, 24, 34, 17, 29

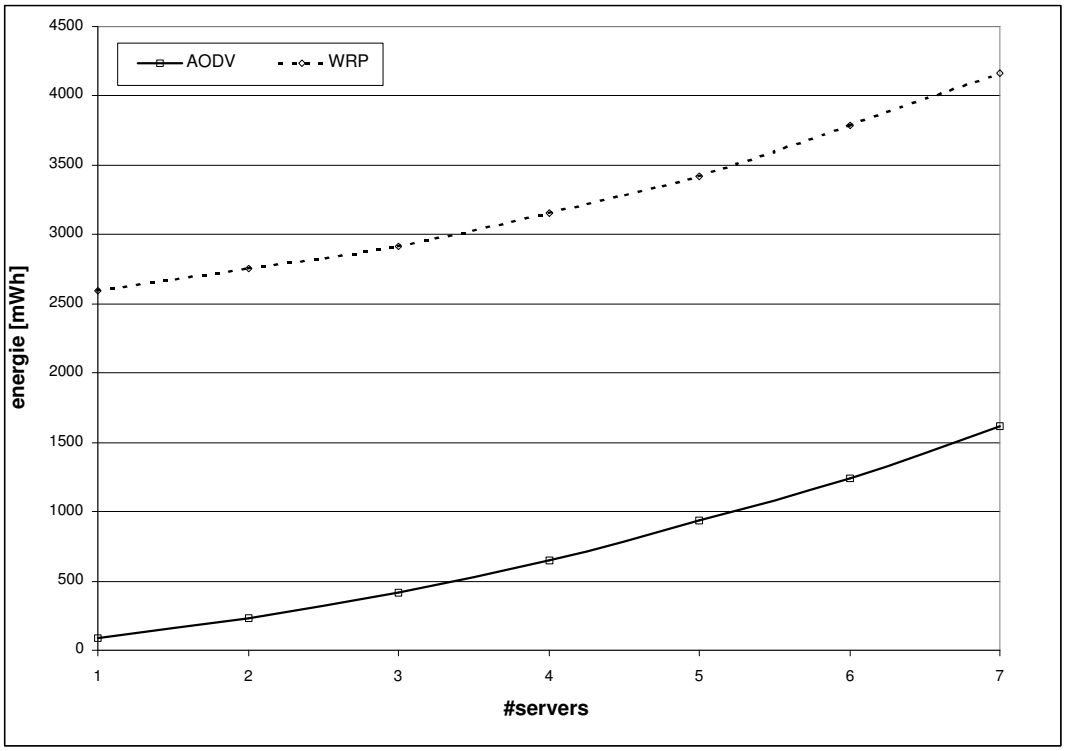
immers niet gebruikt, want er is geen delivery; anderzijds is het effect van WRP op de energie vast (het verschil tussen de twee banden), want het aantal nodes in het netwerk is vast en dus ook de energie die nodig is om proactief de routes tussen al deze nodes te bepalen. *Proactieve resource discovery is dus niet goed schaalbaar naar het aantal servers.*

Reactieve resource discovery

Het aantal services die in het netwerk worden aangeboden zullen op zich uiteraard geen invloed hebben op het aantal berichten nodig voor reactieve resource discovery. *Het reactief resource discovery protocol is dus schaalbaar naar het aantal servers.* Men moet echter wel opmerken dat wanneer er meer services in het netwerk worden aangeboden, de kans groot is dat er sneller een geschikte service gevonden wordt. Dit zou het aantal berichten kunnen doen dalen.



Figuur 4.9: Aantal broadcasts i.f.v. het aantal servers



Figuur 4.10: Verbruikte energie i.f.v. het aantal servers

4.3.4 Schaalbaarheid naar de maximale berichthopcount

Proactieve resource discovery

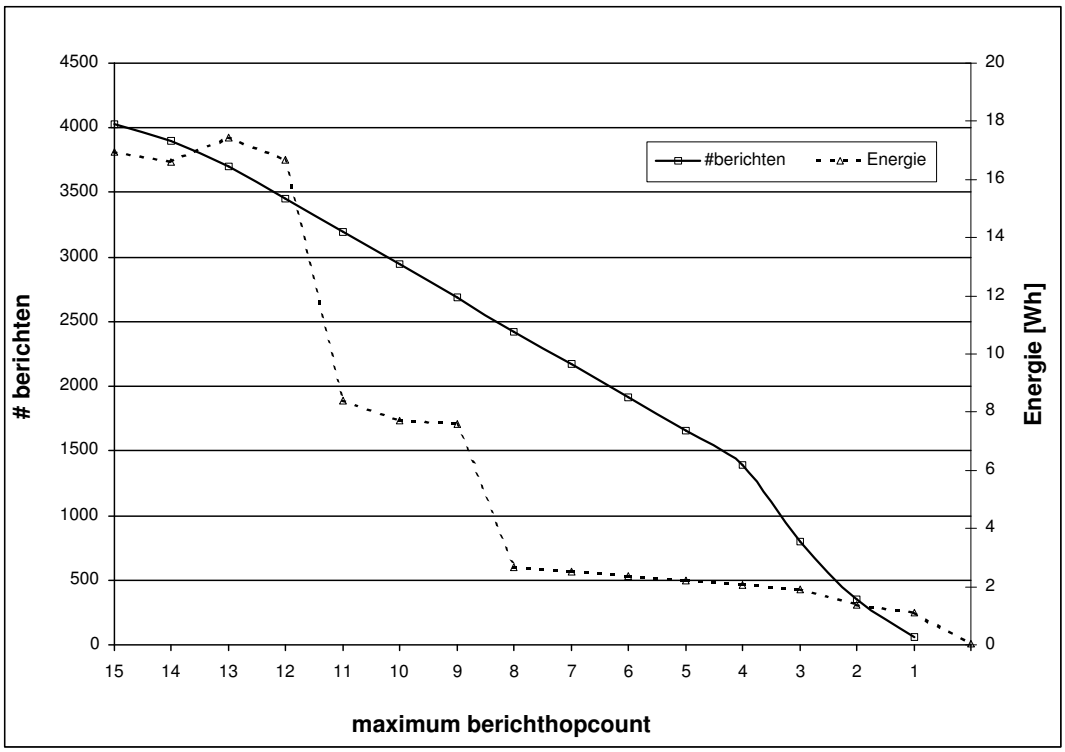
Wanneer de service announcements verder in het netwerk mogen doordringen, zal de overhead van het protocol vergroten en zullen meer nodes in de mogelijkheid gesteld worden om de aangeboden service te gebruiken.
Bij deze simulatie werden de volgende parameters vast gehouden :

parameter	**waarde**
nodeplaatsing	vast, in grid, 64 nodes (16 x 4)
grootte	3200 m x 800 m
simulatietijd	4 min
reannouncement interval	15 s
aantal FTP servers	1 (node 0)
aantal FTP clients	4 (nodes 12, 18, 48, 63)
routing	AODV

De protocolparameter MAX_MESSAGE_HOPCOUNT werd gevarieerd van 15 tot 1. Figuur 4.11 toont de invloed van het toegestane bereik van de service announcements van servernode 0 op het aantal berichten gegenereerd door het protocol en op de verbruikte energie. Zoals verwacht merken we een geleidelijke daling van het aantal berichten bij een verlaging van het toegestane berichtbereik. Ook de verbruikte energie daalt bij een dalend toegestaan berichtbereik doordat de aangeboden service dan door steeds minder nodes kan gebruikt worden. De daling gebeurt hoofdzakelijk in trapjes : er is dan steeds een FTP-client waarvoor de FTP-server onbereikbaar geworden is. Hierdoor stopt de delivery en de routing tussen deze client en de server, en is er hiervoor geen energieverbruik meer (effect van delivery overweegt). Het verzenden van een kleiner aantal protocolberichten zorgt ook voor een lichte daling, maar dit effect overweegt niet.

Reactieve resource discovery

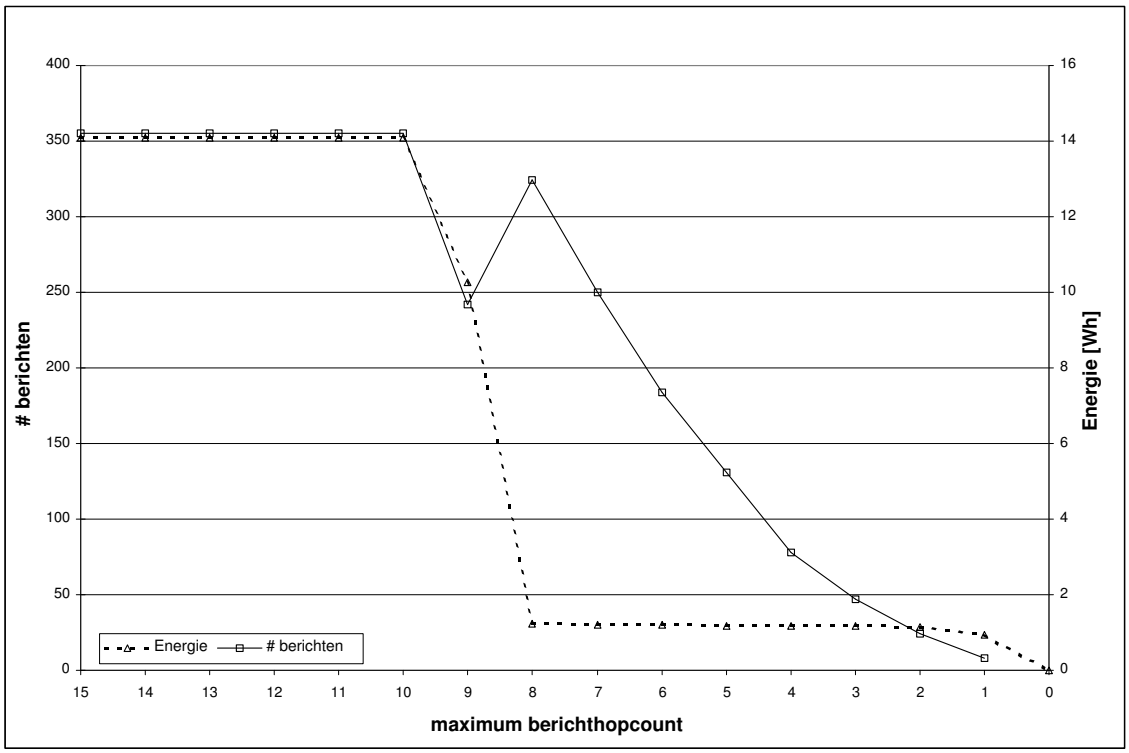
In het reactieve resource discovery protocol bepaalt de parameter *MAX_HOPCOUNT_FOR_RING* wat de maximale waarde is die aan het veld *maximale hopcount* van een *request* pakket (zie tabel 3.4) mag worden toegekend. Deze parameter beperkt bijgevolg de ruimte waarin het protocol een



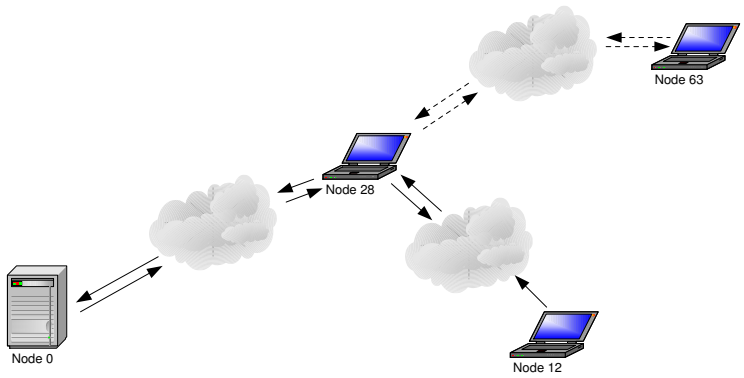
Figuur 4.11: Proactieve discovery: aantal berichten en verbruikte energie i.f.v. de maximale berichthopcount

service voor een applicatie mag zoeken. Een grotere waarde voor *MAX_HOPCOUNT_FOR_RING* zal de kans vergroten dat een service in het netwerk wordt gevonden, maar kan het aantal protocolberichten sterk laten toenemen.
Er werden simulaties gedaan en hiervoor werd hetzelfde scenario gebruikt als bij het proactief resource discovery protocol. Figuur 4.12 toont het aantal protocolberichten en de verbruikte energie in functie van *MAX_HOPCOUNT_FOR_RING*. Wanneer we de parameter gelijk stellen aan 1, kan reeds 1 aanvraag in het netwerk beantwoord worden. Bij de waardes 2, 9 en 10 voor de parameter kan er telkens een extra sessie gestart worden. Zolang niet alle aanvragen in het netwerk beantwoord kunnen worden, neemt het aantal berichten van het resource discovery protocol toe (behalve wanneer *MAX_HOPCOUNT_FOR_RING* gelijk is aan 9, zie verder): voor de nog niet beantwoorde aanvragen wordt de zoekruimte elke keer uitgebreid en dit levert extra berichten op. Eenmaal alle aanvragen beantwoord kunnen worden, neemt het aantal berichten niet meer toe: elke node heeft een geschikte service gevonden in een straal van 10 hops en de zoekruimte wordt niet nodeloos uitgebreid, ook al laat de waarde van *MAX_HOPCOUNT_FOR_RING* dit wel toe. Wanneer we de curve van de verbruikte energie bestuderen, zien we dat deze vooral afhankelijk is van het aantal gestarte sessies. Hieruit volgt dat routing en vooral delivery het leeuwendeel uitmaken van de verbruikte energie en het aandeel van de resource discovery eerder klein is.

Opmerking Op figuur 4.12 is te zien dat zolang niet alle aanvragen in het netwerk beantwoord zijn, het aantal berichten stijgt wanneer de parameter *MAX_HOPCOUNT_FOR_RING* groter wordt. Maar voor de waarde 9 van deze parameter lijkt deze trend doorbroken. Hoe komt dat? Figuur 4.13 laat zien wat er gebeurt. Eerst vraagt node 12 een FTP-server aan. Wanneer *MAX_HOPCOUNT_FOR_RING* kleiner of gelijk is aan 8, vindt deze node geen service in haar zoekruimte. Stellen we de parameter in op de waarde 9, dan is er wel een *request* pakket dat node 0, de server, bereikt. Deze node stuurt daarop een *reply* pakket terug naar node 12. Vooraleer dit pakket in zijn bestemming aankomt echter, maakt node 12 reeds de balans op van de ontvangen services. Aangezien ze nog geen informatie ontvangen heeft, denkt de node dat in de zoekruimte geen service is gevonden. De zoekruimte was reeds maximaal en de node besluit dat er in het netwerk geen geschikte service te vinden is. Even later zoekt node 63 een FTP-server. Is *MAX_HOPCOUNT_FOR_RING* kleiner dan 9, dan wordt er evenmin een service gevonden. Maar wanneer we *MAX_HOPCOUNT_FOR_RING* gelijk stellen aan 9, vindt de node plots heel snel een geschikte service. Node 28 ligt immers op het pad van het *reply* pakket dat



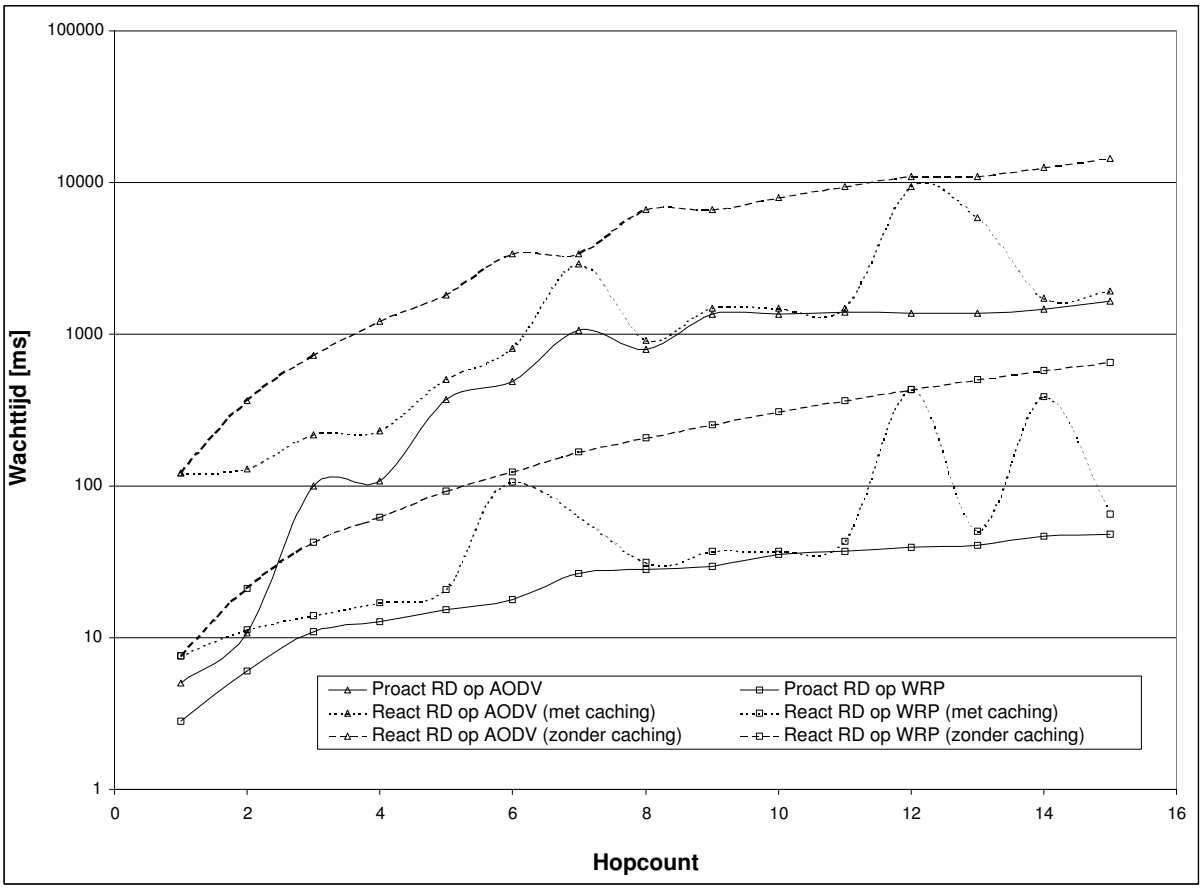
Figuur 4.12: Reactieve discovery: aantal berichten en verbruikte energie i.f.v. de maximale berichthopcount



Figuur 4.13: Minder berichten voor *MAX_HOPCOUNT_FOR_RING* gelijk aan 9

de server *kortelings* daarvoor naar node 12 gestuurd heeft en heeft bijgevolg informatie over de server opgeslagen. Deze node ligt behoorlijk dicht in de buurt van node 63 en antwoordt op een *request* pakket van haar. Bijgevolg kan node 63 snel een extra sessie starten en waren er niet veel berichten nodig om de geschikte service te vinden: de zoekruimte werd geenszins maximaal uitgebreid.

Wanneer *MAX_HOPCOUNT_FOR_RING* gelijk aan 10 wordt gesteld, zal node 12 een extra zoektocht toelaten (met *maximale hopcount* gelijk aan 10). Na deze zoektocht wordt opnieuw een balans opgemaakt en het *reply* pakket van node 0 zal nu wel de tijd gekregen hebben om door te dringen tot in node 12. Hierdoor kan deze node eveneens een sessie starten met de server. Door de extra zoektocht zullen er veel pakketten in het netwerk gebroadcast worden en dit zorgt ervoor dat het aantal pakketten op figuur 4.12 weer stijgt.



Figuur 4.14: Wachttijd tot delivery kan beginnen

4.4 Wachttijd voor een vragende node

Stel dat een FTP-client bij het resource discovery protocol een aanvraag doet om een FTP-server met geschikte eigenschappen af te leveren. Het zal een zekere tijd in beslag nemen vooraleer deze aanvraag zal kunnen beantwoord worden. De wachttijd die de vragende applicatie ondervindt, kan gedefinieerd worden als de tijd die passeert tussen het ogenblik dat de applicatie de aanvraag doet bij de resource discovery laag en het ogenblik dat de applicatie kan starten met de delivery (bvb. FTP-trafiek starten). Deze tijd is afhankelijk van de gebruikte routing- en resource discovery protocollen. Figuur 4.14 geeft voor de verschillende gevallen van routing en resource discovery deze wachttijd weer in functie van de afstand tussen client en server.

Vooreerst merken we voor alle gevallen de stijgende trend van de wachttijd wanneer de afstand tussen client en server vergroot. Merk op dat gewerkt wordt met een logaritmische as voor de wachttijd. De vergroting van de wachttijd bij groter wordende afstand is dus aanzienlijk. De propagatietijd van een bericht over meerdere hops kan dus oplopen.

Laten we vervolgens het verschil tussen proactieve en reactieve routing (WRP vs. AODV) bestuderen. De onderste 3 curves in de grafiek hebben betrekking op proactieve routing, de bovenste 3 curves tonen de invloed van reactieve routing. Er kan dus gesteld worden dat reactieve routing een significant grotere wachttijd zal opleveren omdat het pad tussen client en server nog moet gezocht worden alvorens de delivery kan starten. Wanneer de snelheid van resource discovery dus belangrijk is, wordt best een proactief routing protocol gebruikt. Indien men tevreden is met een langere wachttijd, en het belangrijker vindt dat het energieverbruik van de nodes beperkt blijft, wordt beter een reactief routingprotocol gekozen (zie sectie 4.3).
Beschouwen we vervolgens de invloed van de resource discovery bij proactieve routing (onderste 3 curves in figuur 4.14). De besluiten voor de resource discovery bij reactieve routing (bovenste 3 curves) zullen gelijkaardig zijn. Qua resource discovery wordt het onderscheid gemaakt tussen 3 gevallen :

- Proactieve resource discovery
- Reactieve resource discovery zonder caching
- Reactieve resource discovery met caching

Van deze 3 mogelijkheden levert proactieve resource discovery de laagste wachttijd op. Reactieve discovery zonder caching levert de langste wachttijd van de 3 op. De wachttijd veroorzaakt door reactieve resource discovery met caching schommelt tussen de wachttijden van de vorige 2. Wanneer in dit laatste geval de aanvraag van de vragende client moet beantwoord worden door de server zelf, dan zal de wachttijd even lang zijn als bij reactieve discovery zonder caching. Wanneer de aanvraag van de vragende client kan beantwoord worden door een node die op het pad van een recent gestarte client naar de server ligt, dan zal de wachttijd kleiner zijn. Wanneer de vragende node zelf al het antwoord kent op haar vraag, zal de wachttijd gelijk zijn aan de wachttijd voor proactieve resource discovery. *We kunnen dus besluiten dat de reactieve aanpak een grotere wachttijd zal opleveren dan de proactieve aanpak, en dit zowel wat betreft routing als resource discovery.*

4.5 Criteria voor service selectie

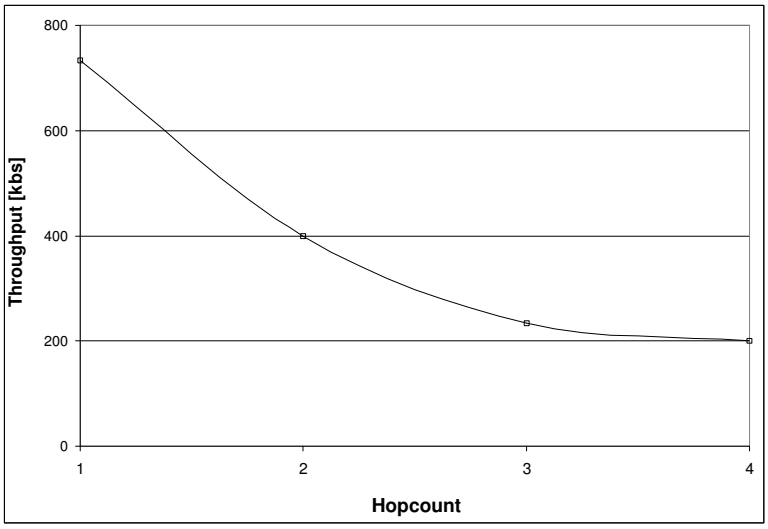
4.5.1 Belang van hopcountgerelateerde parameters

In sectie 3.5 werd reeds ingegaan op de criteria die kunnen in acht genomen worden om de beste service te kiezen uit een aantal mogelijke services. Algemeen konden deze criteria als volgt in een breuk samengenomen worden om de score van een service te bepalen. Hoe groter de score, hoe groter de *Quality Of Service*.

$$score = \frac{(lifetime)\,(service\,attribuut)\,(server\,komt\,dichter)\,(infrastructured)}{(hopcount)\,(serverbelasting)\,(server\,verwijdert\,zich)\,(responstijd)}$$

Niet ieder criterium is echter even belangrijk of even nauwkeurig te bepalen. In wat volgt zullen we de bepaling en het belang van ieder criterium verder uitwerken.

***Throughput* is sterk afhankelijk van de hopcount** De hopcount is gemakkelijk en accuraat te bepalen zowel voor reactieve als proactieve discovery. Figuur 4.15 toont de throughput van een FTP-sessie in een statisch netwerk in functie van de hopcount. Uit deze simulatie is gebleken dat de throughput sterk afhankelijk is van de afstand tussen client en server. Het is dus belangrijk om de hopcountgerelateerde parameters een groot belang te geven in de breuk voor service selectie.



Figuur 4.15: Throughput i.f.v. afstand tussen client en server

***Lifetime* is weinig relevant wegens mobiel karakter** Een netwerk waarin de nodes zich niet verplaatsen en altijd beschikbaar zijn, heeft geen last van link breaks of services die onbe-

schikbaar worden. In een mobiel ad hoc netwerk echter kan men hier niet van uitgaan. Wanneer een service op server x in principe nog 1 uur beschikbaar is voor client y dan is het vrij onwaarschijnlijk dat dit ook werkelijk het geval zal zijn wegens de topologieveranderingen die kunnen optreden. Hierdoor wordt de lifetime van een service minder relevant om de service selectie op te baseren. Deze parameter wordt dan ook beter niet opgenomen in de breuk voor service selectie.

***Responstijd* is onbruikbaar wegens caching** Bij proactieve resource discovery is de responstijd niet bruikbaar omdat de service informatie niet bekomen wordt door het verzenden van *requests* en het ontvangen van *replies*.
Bij reactieve resource discovery is de responstijd enkel bruikbaar wanneer geen caching gebruikt wordt. Enkel in dat geval is de responstijd proportioneel met de belasting van het pad tussen client en server en van de server zelf. In het geval van caching kan het antwoord komen van een tussenliggende node. De responstijd is dan niet representatief voor de kwaliteit van de service. Aangezien caching een gunstige invloed heeft op het aantal berichten en op de wachttijd voor een applicatie (zie secties 4.3.2 en 4.4) is caching wenselijk en kunnen we de responstijd dus niet gebruiken voor de service selectie.

***Serverbelasting* implementeren als lokale beslissing door server** Een beslissing over de serverbelasting kan best genomen worden door de server zelf. De server beslist dan afhankelijk van zijn huidige belasting of hij een bepaalde service verder blijft beschikbaar stellen voor nieuwe connecties.

Service attribuut* en *infrastructured Deze parameters kunnen als absolute metriek beschouwd worden, wat de breuk verder vereenvoudigt.

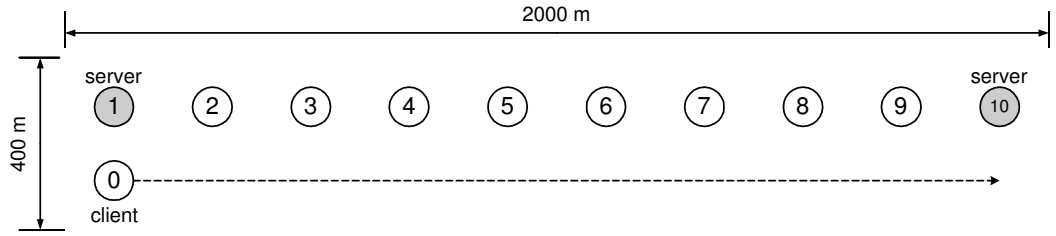
De vorige bemerkingen in acht nemend komen we tot een breuk voor service selectie waarin enkel nog hopcountgerelateerde parameters staan :

$$score = \frac{(server\,komt\,dichter)}{(hopcount)\,(server\,verwijdert\,zich)}$$

In wat volgt zullen we de overblijvende parameters nader bekijken.

4.5.2 Detectie van mobiliteit

Proactieve resource discovery De detectie van mobiliteit gebeurt door het protocol op een relatieve manier. Om de MOBILITY_CALCULATION_INTERVAL seconden evalueert een node



Figuur 4.16: Gesimuleerd scenario

de services die in haar *service_found_table* vermeld staan. Voor iedere service wordt de waarde van de huidige hopcount vergeleken met de waarde van de hopcount bij de vorige evaluatie. Afhankelijk van de grootte van de verandering wordt beslist of de node die de service aanbiedt relatief gezien dichterbij gekomen is (*approaching*) of zich verwijderd heeft (*leaving*). Deze beslissing zal dan geldig blijven tot de volgende evaluatie MOBILITY_CALCULATION_INTERVAL seconden later. De plattegrond van het simulatiescenario is afgebeeld in figuur 4.16; tabel 4.6 toont de parameters die voor deze simulatie gekozen werden. Node 0 is de enige bewegende node. Ze beweegt van server 1 naar server 10. Node 1 verwijdert zich relatief gezien van node 0, node 10 komt relatief gezien dichterbij. Nodes 2 tot 9 staan vast en dienen als tussenhops. In deze simulatie wordt nagegaan hoezeer het aan/uit springen van *approaching* en *leaving* afhankelijk is van de waarde van de parameter MOBILITY_CALCULATION_INTERVAL (voor een bepaalde snelheid van node 0). Figuren 4.17, 4.18 en 4.19 geven de waarde van de *approaching* en *leaving* informatie in node 0 over server 1 en 10 i.f.v. de plaats van node 0 bij groter wordend MOBILITY_CALCULATION_INTERVAL.

Tabel 4.6: Parameters voor de simulatie

parameter	**waarde**
positie nodes 1 tot 10	vast, zie figuur 4.16
positie node 0	mobiel, snelheid 4 m/s
simulatietijd	10 min
reannouncement interval	15 s
aantal FTP-servers	2 (nodes 1 en 10)
aantal FTP-clients	1 (node 0)
HOPCOUNTTRESHOLD	1

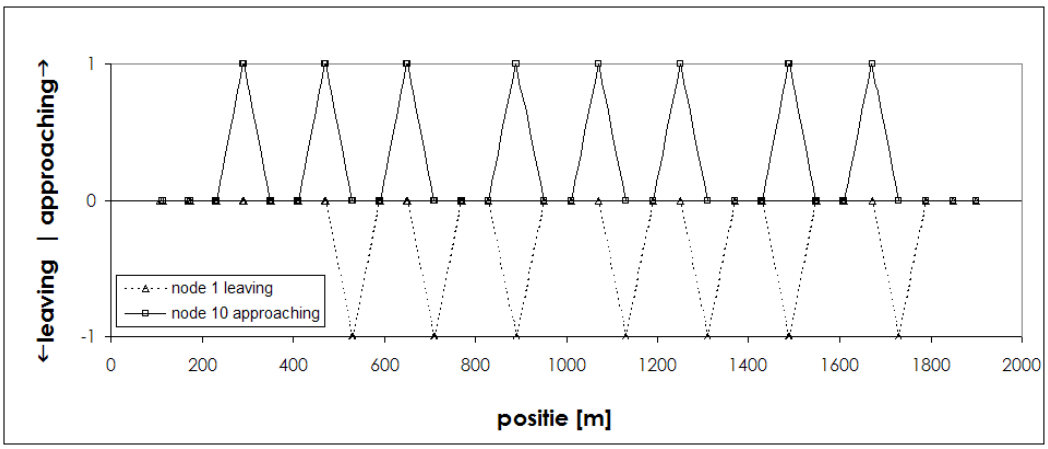
Wanneer het evaluatie-interval te klein gekozen wordt (figuur 4.17), dan blijkt voor de gekozen

snelheid van node 0 de hopcountverandering voor server 1 en 10 na dit interval niet altijd voldoende te zijn om de *approaching* en *leaving* indicatie voortdurend aan te houden. Deze indicatie kan stabieler gemaakt worden door het evaluatie-interval langer te maken (figuren 4.18 en 4.19). Hierdoor zal echter de respons van de indicatie trager worden, waardoor het weer mogelijk is dat de indicatie niet overeen komt met de werkelijkheid. Zowel een lang als een kort MOBILITY_CALCULATION_INTERVAL levert dus zijn eigen problemen op. Aan de basis hiervan ligt het feit dat *de snelheid van de nodes niet gekend is.* Mocht deze snelheid wel gekend zijn dan zou in principe een optimale waarde kunnen bepaald worden voor het evaluatie-interval. Een grotere nodesnelheid bvb. zal de hopcount sneller doen veranderen waardoor het evaluatie-interval kleiner zou kunnen gemaakt worden. Aangezien de nodesnelheid niet gekend is kan men dus niet zeker zijn van de stabiliteit van *approaching* en *leaving.* Wanneer we deze parameters in de breuk voor service selectie zouden opnemen, dan zou de breuk grote schommelingen kunnen vertonen afhankelijk van het ogenblik waarop ze geëvalueerd wordt. Dit is uiteraard niet gewenst. Daarom laten we *approaching* en *leaving* beter weg uit de breuk. We komen dan tot een service selectie waarin enkel de hopcount weerhouden wordt als beslissingscriterium :

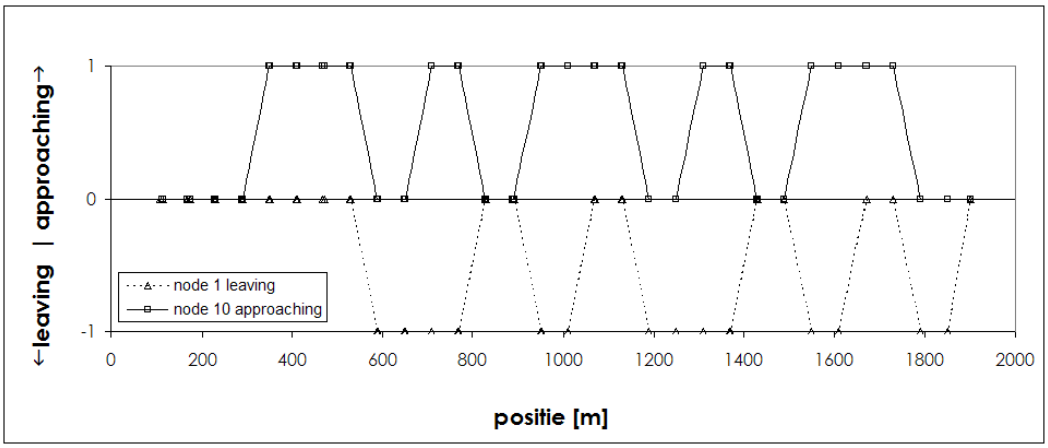
$$score = \frac{1}{(hopcount)}$$

Reactieve resource discovery Telkens wanneer een node informatie over een service ontvangt, onderzoekt het reactief resource discovery protocol of ze die service al kende. Is dit zo, dan vergelijkt ze de hopcount uit het toegekomen pakket met de hopcount uit de *service_found_table* om de velden *approaching* en *leaving* eventueel aan te passen. Voor een meer uitgebreide bespreking van dit algoritme wordt verwezen naar sectie 3.7.6.

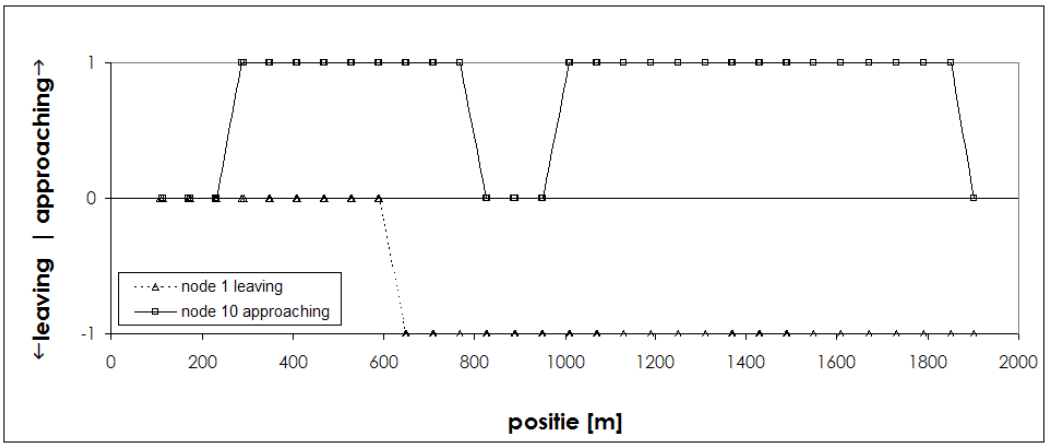
Om de nauwkeurigheid van de bepaling van *hopcount*, *approaching* en *leaving* na te gaan, werden een aantal simulaties gedaan met hetzelfde scenario als voor het proactieve resource discovery protocol. Hier werd echter een HTTP-server gezocht i.p.v. een FTP-server. We lieten enerzijds de snelheid van node 0 variëren en anderzijds de frequentie waarmee een nieuwe gateway in het netwerk gezocht wordt (het gaat immers om een *single commodity service*, zie sectie 3.7.5). Figuren 4.20 t.e.m. 4.22 tonen de resultaten voor de simulaties waar AODV als routing protocol werd gebruikt. Figuren 4.23 t.e.m. 4.25 tonen de resultaten voor de simulaties waar WRP als routing protocol werd gebruikt. De abscis stelt de horizontale plaats voor van node 0. De ordinaat toont de hopcount van de services, en de parameters *approaching* en *leaving.* Wanneer een service dichter komt, springt *approaching* op *waar*, wanneer een service zich verder verwijdert,



Figuur 4.17: Approaching/leaving bij MOBILITY_CALCULATION_INTERVAL = 15 s



Figuur 4.18: Approaching/leaving bij MOBILITY_CALCULATION_INTERVAL = 30 s



Figuur 4.19: Approaching/leaving bij MOBILITY_CALCULATION_INTERVAL = 45 s

springt *leaving* op *waar*.

Figuur 4.20 toont hoe de resultaten zijn wanneer AODV als routing protocol wordt gebruikt, 1 m/s als snelheid van node 0 wordt gekozen en het update-interval 15 s bedraagt. Informatie wordt een minuut lang gecacht. We zien dat eerst node 1 in het netwerk gebruikt wordt, later node 10. De hopcount van deze services veranderen heel regelmatig van waarde waaruit besloten kan worden dat de bepaling van deze parameter bijzonder accuraat is. Door de constructie van het resource discovery protocol geven *approaching* en *leaving* enkel bij een verandering van hopcount aan dat er mobiliteit is. In dit geval geeft dit een verkeerd beeld weer. Men zou zich kunnen afvragen waarom we dan, net zoals bij het proactieve protocol, niet werken met een *MOBILITY_CALCULATION_INTERVAL*. Welnu, bij proactieve resource discovery kan men ervan uitgaan dat de informatie over de services *regelmatig* geüpdatet wordt, maar bij reactieve discovery is dit zeker *niet altijd het geval*, zeker niet wanneer het gaat om een *multi commodity service*. Bijgevolg is een regelmatige aanpassing van *approaching* en *leaving* nutteloos: het kan gebeuren dat de informatie over de service nooit wordt geüpdatet tijdens het interval. Rechts op de grafiek kan men zien dat de hopcount even een sprong maakt. Dit is een gevolg van het feit dat heel af en toe pakketten te laat toekomen: het pakket dat de node vertelde dat de hopcount naar node 10 2 bedroeg was wel op tijd, het pakket dat vertelde dat de hopcount naar node 10 1 bedroeg was te laat om in de balans te worden betrokken.

In figuur 4.21 bleven de waarden van de protocolparameters dezelfde, maar werd de snelheid van node 0 verhoogd tot 4 m/s. Door de hogere snelheid zal de node meer afstand afgelegd hebben op het moment dat ze opnieuw services gaat zoeken in het netwerk. Tijdens de tocht van de node wordt de informatie daarom minder vaak geüpdatet. Op de grafiek zien we dat de parameters van het protocol slecht aangepast zijn aan de hoge snelheid: gecachte informatie wordt te lang vastgehouden waardoor de hopcounts niet meer altijd even accuraat zijn.

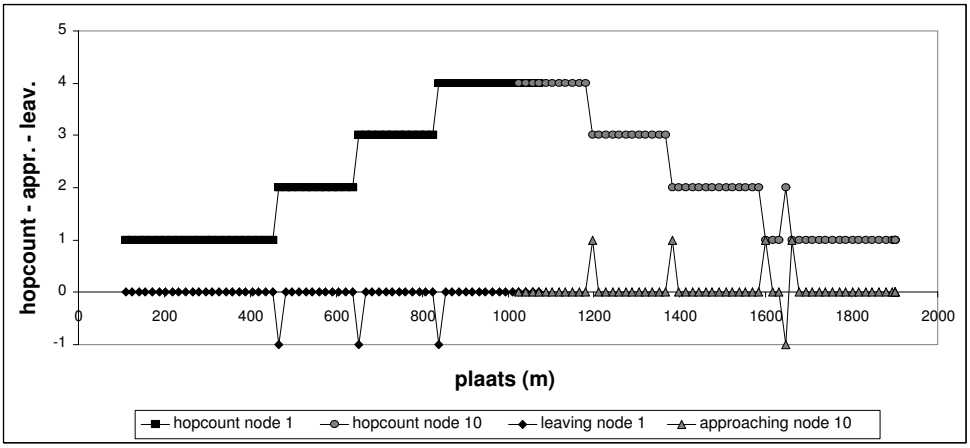
Voor figuur 4.22 zijn de protocolparameters wel aangepast aan een nodesnelheid van 4 m/s: er werd naar een update gezocht om de 3,75 s en informatie bleef slechts 15 s geldig. Door deze aanpassing is de grafiek nagenoeg een kopie van figuur 4.20: als de snelheid vermenigvuldigd wordt met 4, moet ook de updatefrequentie met 4 vermenigvuldigd worden om een goed resultaat te bekomen. Het enig verschil met figuur 4.20 is dat node 10 te lang als 3 hops ver beschouwd wordt. Dit komt omdat de parameters van AODV niet aangepast zijn aan de hoge nodesnelheid: een oude route wordt te lang gebruikt. Een route is immers geldig in AODV tot er link failures optreden of tot wanneer er een zekere periode voorbijgaat waarin er geen trafiek over de route

passeert. De hopcount van node 1 is wel accuraat: er wordt vaker van route veranderd omdat de oude route niet meer werkt, node 0 verwijdert zich immers van node 1. Wanneer ook de AODV-parameters aangepast zouden worden, kan ook dit verholpen worden.
Voor figuren 4.23, 4.24 en 4.25 werden exact dezelfde parameters en nodesnelheid gebruikt als voor resp. figuren 4.20, 4.21 en 4.22. Het enig verschil is dat nu WRP gebruikt wordt i.p.v. AODV als routing protocol. Op figuren 4.23 en 4.25 zien we dat, als de parameters aan de snelheid zijn aangepast, de hopcount heel accuraat bepaald kan worden. Doordat de update-frequentie van WRP heel hoog is ingesteld, voldoet het routing protocol zelfs wanneer de node aan 4 m/s beweegt. Op figuur 4.24 zijn de parameters van het resource discovery protocol niet aangepast aan de hoge nodesnelheid: dit zorgt voor een minder nauwkeurige hopcountbepaling. Bovendien wou het toeval dat er iets misliep: wanneer node 0 zich ongeveer 600 m ver bevond, ontving node 1 enkele *request* pakketten niet (de ether was sterk bevuild met HTTP-trafiek tussen node 0 en node 1). Daardoor breidde node 0 haar zoektocht uit en vond ze node 10 als HTTP-server. Vanaf dan werd deze nieuwe server gebruikt. Na 1 minuut waren alle caches in de nodes leeg en moest opnieuw gezocht worden naar een service tot in de server zelf. Op dat moment werd node 1 opnieuw gevonden en gebruikt. Dit bleef zo tot het uiteindelijk voordeliger werd om opnieuw node 10 als server te gebruiken.
Uit de simulaties kan besloten worden dat de hopcount naar een service accuraat en gemakkelijk bepaald kan worden wanneer de parameters van het resource discovery protocol zijn aangepast aan de snelheid van de nodes. Wanneer dit niet het geval is, is de meting minder nauwkeurig, maar vaak nog voldoende om geen slechte keuze te maken op basis van de hopcount bij de selectie van de beste service. De bepaling van de parameters *approaching* en *leaving* is echter nog moeilijker dan bij het proactieve resource discovery protocol. Behalve de absolute snelheid van de nodes, kennen we hier nl. ook de frequentie niet waarmee updates verwacht mogen worden. Daarom wordt besloten om ook voor het reactieve resource discovery protocol enkel de hopcount te behouden in de breuk voor service selectie:

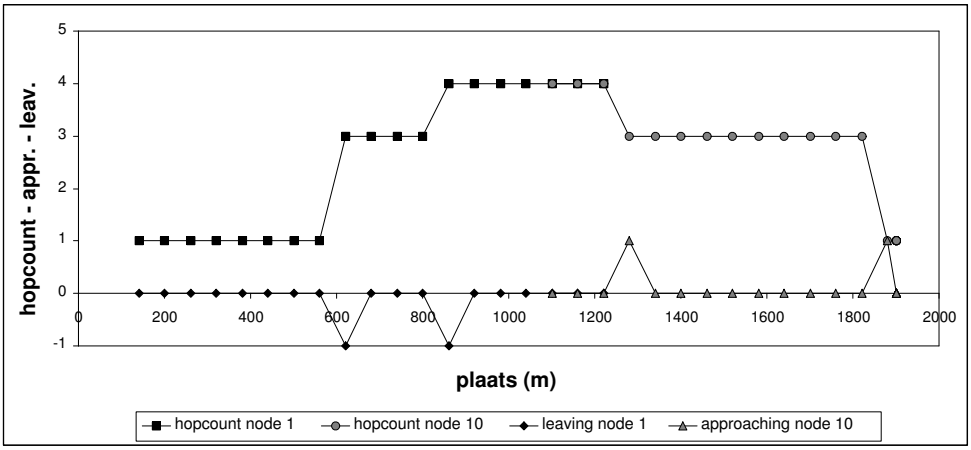
$$score = \frac{1}{(hopcount)}$$

4.5.3 Vermijden dat onnodig van resource veranderd wordt

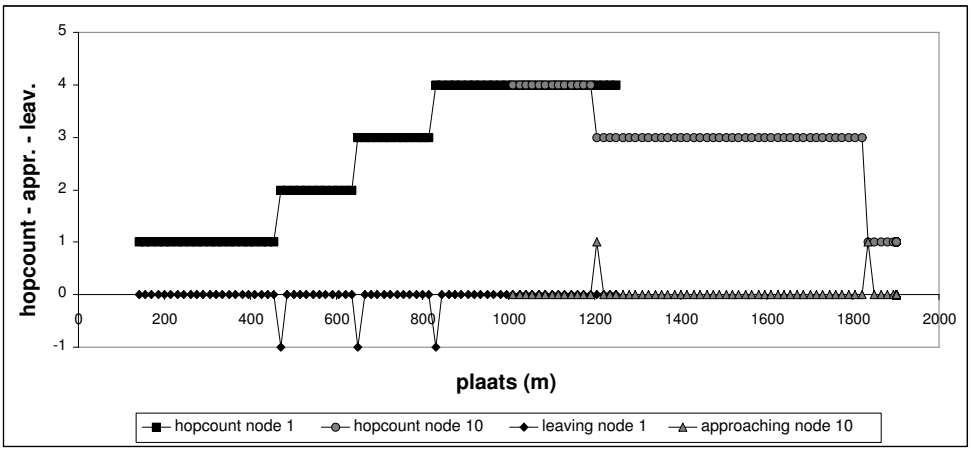
Breuk voor service selectie zo stabiel mogelijk maken Wanneer *approaching* en *leaving* berekend worden zoals hierboven uiteengezet en opgenomen worden in de scoreberekening van de services, dan zullen deze parameters aanleiding geven tot onaanvaardbare schommelingen van de



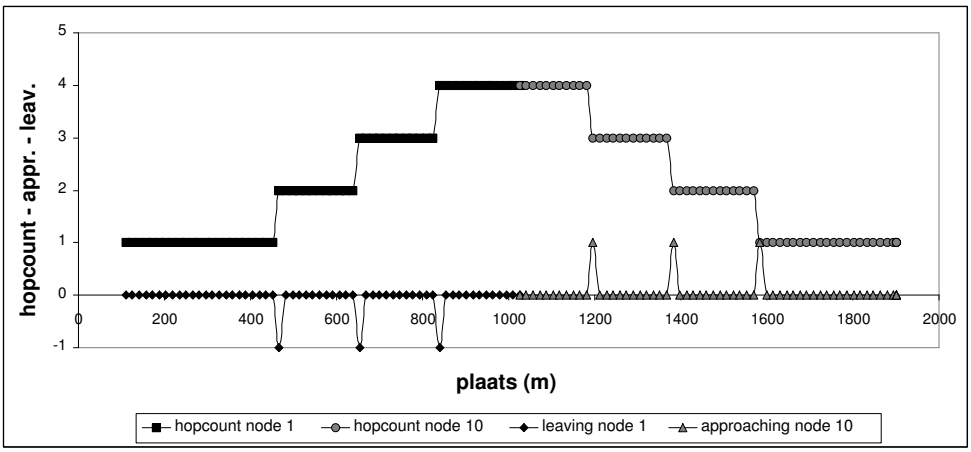
Figuur 4.20: Hopcount en approaching/leaving (AODV, nodesnelheid = 1 m/s, interval = 15 s)



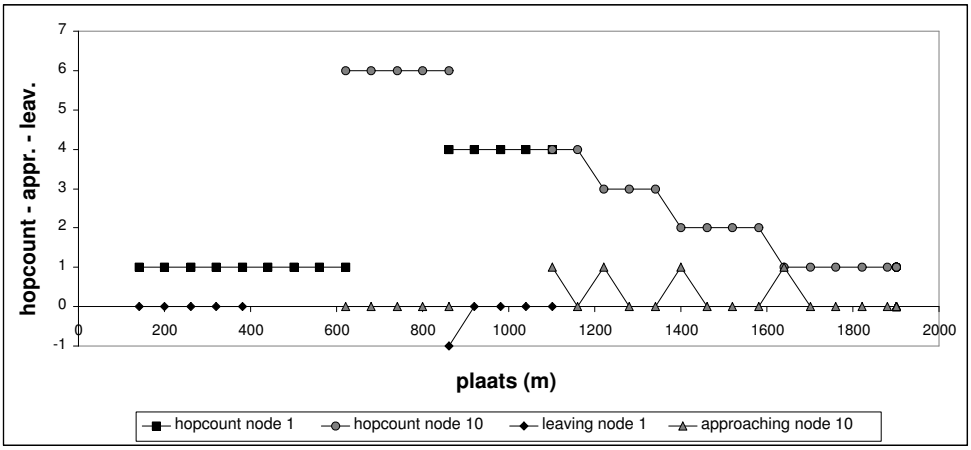
Figuur 4.21: Hopcount en approaching/leaving (AODV, nodesnelheid = 4 m/s, interval = 15 s)



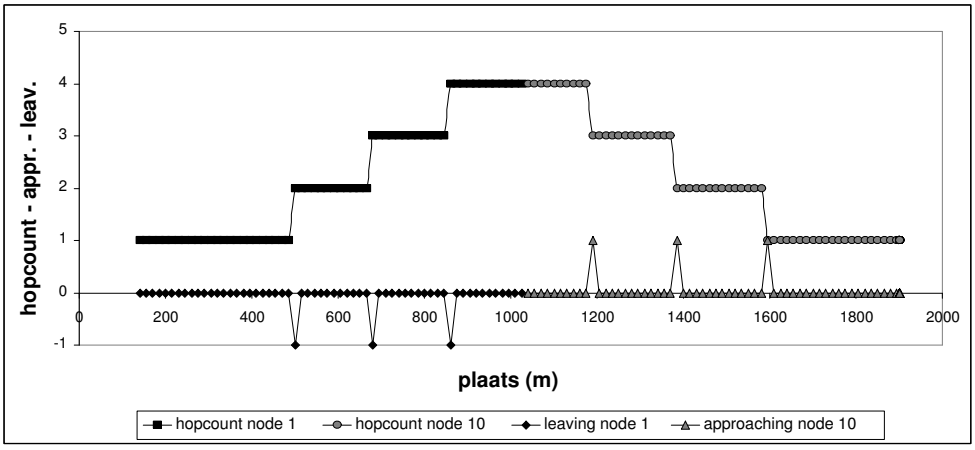
Figuur 4.22: Hopcount en approaching/leaving (AODV, nodesnelheid = 4 m/s, interval = 3,75 s)



Figuur 4.23: Hopcount en approaching/leaving (WRP, nodesnelheid = 1 m/s, interval = 15 s)



Figuur 4.24: Hopcount en approaching/leaving (WRP, nodesnelheid = 4 m/s, interval = 15 s)



Figuur 4.25: Hopcount en approaching/leaving (WRP, nodesnelheid = 4 m/s, interval = 3,75 s)

Tabel 4.7: Aantal serviceveranderingen voor verschillende gewichten van approaching/leaving

scoreberekening	**reactief**	**proactief**
$\frac{1}{(1+hopcount)}$	11	15
$\frac{1}{(1+hopcount-approaching+leaving)}$	39	38
$\frac{1}{(1+hopcount-2*approaching+2*leaving)}$	68	70

servicescores. Hierdoor is het mogelijk dat een suboptimale service gekozen wordt. In het geval van een single commodity service (bvb. kiezen van een gateway voor verbinding met HTTP-server) is het hierdoor mogelijk dat er meer gatewayveranderingen zullen optreden dan nodig. Figuur 4.7 toont het aantal serviceveranderingen voor verschillende gewichten van *approaching* en *leaving* voor eenzelfde referentiescenario (zie appendix A). Wanneer deze parameters een groter gewicht krijgen, veroorzaken ze meer serviceveranderingen. *Men gebruikt dus best enkel de hopcount om een stabiele scoreberekening te bekomen.*

Huidige gebruikte gateway proberen hergebruiken In het geval van HTTP-trafiek, waarbij men tussen het opvragen van 2 webpagina's van gateway kan veranderen, doet men er goed aan om de huidige gebruikte gateway te hergebruiken als er ondertussen geen andere gateway opgedoken is die *aanzienlijk* beter is (bvb. 20%; zie sectie 3.5). Hierdoor kan het aantal onnodige serviceveranderingen verder beperkt worden.

4.6 Routing

Uit de gedane simulaties is gebleken dat de prestatie van de resource discovery zeer sterk afhankelijk is van welk routing protocol men gebruikt. Hoe de keuze van het routing protocol de resource discovery beïnvloedt, wordt hier nog eens op een rijtje gezet.

Figuur 4.14 toont de wachttijd voor een applicatie: de tijd die verloopt tussen de aanvraag voor een service en het moment waarop de applicatie een geschikte service begint te gebruiken. Het blijkt dat wanneer voor proactieve routing gekozen wordt, de wachttijd veel korter is dan wanneer voor reactieve routing wordt geopteerd. Het proactief routing protocol zal nl. ten allen tijde een route kennen naar de gevonden service waardoor, vooraleer de delivery kan starten, geen route meer naar de service moet gezocht worden. Maken we gebruik van een reactief routing protocol, dan zal wel nog eerst een route gezocht moeten worden.

Kiezen we, in combinatie met reactieve routing, voor een proactief resource discovery protocol, dan moet de service aanvragende node een route aanvragen naar de te gebruiken server. Wordt gebruik gemaakt van reactieve resource discovery op reactieve routing, dan zal de informatie verschaffende node eerst een route naar de service aanvragende node moet zoeken om het *reply* pakket terug te kunnen sturen. Is de informatie verschaffende node eveneens de service verlenende node, dan kan het omgekeerde pad meteen gebruikt worden voor de delivery. Is dit echter niet zo, dan moet de service aanvragende node ook nog eens een pad zoeken naar de server. Hieruit volgt dat *wanneer we een snelle resource discovery wensen, we altijd gebruik moeten maken van proactieve routing.*

Figuren 4.6 en 4.7 tonen de verbruikte energie resp. i.f.v. het aantal nodes en i.f.v. het aantal service requests in het netwerk. Uit deze grafieken blijkt dat proactieve routing veel meer energie verbruikt dan reactieve routing en een proactief routing protocol extreem slecht schaalbaar is naar het aantal nodes. Kortom: *wanneer het belangrijk is dat het energieverbruik beperkt wordt, moet gekozen worden voor reactieve routing.*

Zoals reeds vroeger opgemerkt, loopt het routing protocol soms achter op het resource discovery protocol wat de hopcountindicatie betreft. Dit is het gevolg van de gelaagde aanpak: routing en resource discovery gebeuren gescheiden.

Hoofdstuk 5

Besluiten

Reactieve versus proactieve routing Er is gebleken dat de resource discovery aanzienlijk sneller gebeurt wanneer een onderliggend proactief routing protocol wordt gebruikt. Proactieve routing is echter slecht schaalbaar wanneer het aantal nodes in het netwerk toeneemt. Wanneer het dus belangrijk is dat de resource discovery zo snel mogelijk gebeurt, dan opteert men best voor proactieve routing. Indien de snelheid minder kritisch is of indien men het energieverbruik van de nodes wil beperkt houden, dan kiest men beter voor reactieve routing.

Reactieve versus proactieve resource discovery Caching heeft een gunstige invloed op de snelheid waarmee informatie over services zich in het netwerk kan verspreiden, en bij reactieve discovery ook op het aantal berichten. Dankzij caching kan reactieve resource discovery in bepaalde gevallen even snel worden als proactieve resource discovery (bij gebruik van hetzelfde routingprotocol), alhoewel dit niet kan gegarandeerd worden omdat dit afhankelijk is van de gecachte informatie op het ogenblik van de service aanvraag.
Het aandeel van de delivery (FTP- of HTTP-trafiek) en de routing in de door de nodes verbruikte energie blijkt dikwijls veel groter te zijn dan het aandeel van de resource discovery.
Reactieve resource discovery is schaalbaar naar aantal nodes en aantal servers. Wanneer caching kan gebruikt worden (o.a. wanneer requests elkaar snel opvolgen), is dit ook schaalbaar naar aantal requests. Proactieve resource discovery is schaalbaar naar aantal requests, schaalbaar naar aantal nodes, en niet schaalbaar naar aantal servers.

Gescheiden resource discovery en routing Er werd in deze thesis uitgegaan van een gelaagde aanpak waarbij de resource discovery en de routing gescheiden van elkaar gebeuren

in verschillende lagen van de protocolstack. Het zou interessant zijn om verder te onderzoeken of (en in welke gevallen) integratie van het resource discovery protocol in het routing protocol wenselijk is. In het bijzonder lijkt een geïntegreerd protocol waar proactieve discovery gebeurt op proactieve routing of reactieve discovery op reactieve routing ons zinvol.

Service selectie Het aantal hops tussen een client en een server heeft een grote invloed op de haalbare throughput tussen deze client en server. De hopcount naar een service is gemakkelijk te bepalen en de waarde ervan is zelden onderhevig aan grote schommelingen over korte periodes. Dit maakt dat de hopcount de beste parameter is om zich op te baseren voor de service selectie. De mobiliteitsdetectie door observatie van de hopcount en het detecteren van hopcountveranderingen blijkt in de praktijk moeilijk te realiseren en werkt onnodig veranderen van resource in de hand. Dit wordt dan ook beter niet gebruikt bij de service selectie.
Men zou kunnen proberen de mobiliteitsdetectie op een andere manier aan te pakken: bijvoorbeeld door GPS-informatie te gebruiken. Hierdoor kan een node haar *absolute* positie en haar eigen beweegrichting bepalen. Op basis daarvan zou ze kunnen voorspellen waar ze zich in de nabije toekomst waarschijnlijk zal bevinden. Wanneer ze deze informatie aan andere nodes meedeelt zouden deze hier rekening mee kunnen houden bij de service selectie.

Bijlage A

Referentiescenario

Er werd een referentiescenario gedefinieerd om het proactief en reactief resource discovery protocol met elkaar te kunnen vergelijken onder exact dezelfde netwerkomstandigheden : dezelfde startpositie en beweging van de nodes, dezelfde gegenereerde FTP- en HTTP-trafiek,... De details zijn te vinden in de GloMoSim-bestanden *nodes.input*, *mobility.in*, *config.in* en *app.conf* op de CD-ROM. (zie /cdrom/simulatie/referentiescenario).

Bijlage B

CD-ROM

Inhoud van de CD-ROM:

- */code*
  - */code/proactief:* GloMoSim-directory met de code en nodige aanpassingen voor proactieve resource discovery. Het resource discovery protocol staat in de bestanden *serviceProactive.h* en *serviceProactive.pc* in de subdirectory *./Glomosim/transport.*
  - */code/reactief:* GloMoSim-directory met de code en nodige aanpassingen voor reactieve resource discovery. Het resource discovery protocol staat in de bestanden *serviceReactive.h* en *serviceReactive.pc* in de subdirectory *./Glomosim/transport.*
- */boek:* Elektronische versie van de scriptie [pdf]
- */presentatie:* Eindpresentatie van de thesis [ppt]
- */simulatie:* GloMoSim-bestanden m.b.t. het referentiescenario (zie appendix A)

De volgende GloMoSim-bestanden werden aangepast/toegevoegd om het resource discovery protocol te integreren in GloMoSim:

/application/ftp_server.pc
/application/ftp_server.h
/application/ftp_client.pc
/application/ftp_client.h
/application/http_client.h

/application/http_client.pc

/application/http_server.h

/application/http_server.pc

/application/application.pc

/main/glomo.h

/main/glomo.pc

/include/network.h

/include/nwcommon.h

/include/api.h

/network/nwip.pc

/include/structmsg.h

/include/transport.h

/transport/transport.pc

/transport/udp.pc

/transport/serviceProactive.h

/transport/serviceProactive.pc

/transport/serviceReactive.h

/transport/serviceReactive.pc

Bibliografie

[1] S. Murthy, J.J. Garcia-Luna. "An efficient routing protocol for wireless networks", Baltzer Journals, 1996

[2] C.E. Perkins. "Ad hoc networking", 2001, Addison-Wesley, ISBN 0-201-30976-9

[3] C. Bettstetter, C. Renner. "A comparison of service discovery protocols and implementation of the Service Location Protocol", SEP-2000

[4] D. Doval, D. O'Mahony. "Nom: Resource location and discovery for ad hoc mobile networks", SEP-2002

[5] D. Chakraborty, A.U. Joshi, Y. Yesta, T. Finin. "GSD: A novel group-based service discovery protocol for MANETS", 2002

[6] E.M. Belding-Royer, C.E. Perkins. "Internet connectivity for ad hoc mobile networks", 2002

[7] A. Lindgren, O. Schelen. "Infrastructured ad hoc networks", 2002

[8] M.A. Ergin. "A cross layer protocol for service access in mobile ad hoc networks", 2003

[9] I.D. Chakeres, E.M. Belding-Royer. "Utilizing resource-rich nodes in ad hoc networks", 2003

[10] E.M. Royer, C.K. Toh. "A review of current routing protocols for ad hoc mobile wireless networks", 1999

[11] S. Helal, N. Desai, V. Verma, C. Lee. "Konark - A service discovery and delivery protocol for ad hoc networks", 2003

[12] E. Guttman, C. Perkins. RFC2608 : Service Location Protocol, version 2 (protocol overview) [www.ietf.org], JUN-1999

[13] Bluetooth specification, part E: Service Discovery Protocol [www.bluetooth.org], DEC-1999

[14] "Understanding UPnP", whitepaper [www.upnp.org], JUN-2000

[15] Salutation architecture specification V2.1 [www.salutation.org], 1998

[16] Y. Sun, E.M. Belding-Royer. "Dynamic Address Configuration in Mobile Ad Hoc Networks", 2002

[17] GloMoSim homepage [http://pcl.cs.ucla.edu/projects/glomosim], MEI-2004

[18] X. Zeng, R. Bagrodia, M. Gerla. "GloMoSim: A library for parallel simulation of large-scale wireless networks", 1998

[19] Parsec homepage [http://pcl.cs.ucla.edu/projects/parsec], MEI-2004

[20] R. Ouellet, U. Ogbuji. "Introduction to DAML"[http://www.xml.com], 2002