



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

Intelligent overlay multicasten voor QoS-gevoelige multimediasdiensten

door

Jeroen FAMAHEY

Promotoren: Prof. Dr. Ir. B. DHOEDT en Prof. Dr. Ir. F. DE TURCK
Begeleider: B. DE VLEESCHAUWER

Scriptie ingediend tot het behalen van de academische graad van
licentiaat informatica

Academiejaar 2006–2007



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

Intelligent overlay multicasten voor QoS-gevoelige multimediasdiensten

door

Jeroen FAMAHEY

Promotoren: Prof. Dr. Ir. B. DHOEDT en Prof. Dr. Ir. F. DE TURCK
Begeleider: B. DE VLEESCHAUWER

Scriptie ingediend tot het behalen van de academische graad van
licentiaat informatica

Academiejaar 2006–2007

Woord vooraf

Een aantal mensen hebben een onmiskenbare bijdrage geleverd tot het voltooiën van mijn thesis. Bij deze zou ik hen dan ook willen bedanken.

Eerst en vooral zou ik mijn promotoren Prof. Dr. Ir. F. De Turck en Prof. Dr. Ir. B. Dhoedt willen bedanken voor de tips, de goeie raad en vooral voor alle kansen die ze me gegeven hebben.

Mijn dank gaat ook uit naar mijn begeleider Bart De Vleeschauwer voor de vele uren die hij voor me heeft vrijgemaakt en natuurlijk ook voor het grondig en kritisch nalezen van mijn thesisboek.

Ik zou ook mijn ouders bij deze willen bedanken, omdat ze me steeds hebben gestimuleerd en natuurlijk omdat ze mijn studies bekostigd hebben.

En als laatste, maar zeker niet als minste, zou ik Veronic willen bedanken. Ze stond, en staat, steeds voor me klaar.

Jeroen Famaey, juni 2007

Toelating tot bruikleen

“De auteur geeft de toelating deze scriptie voor consultatie beschikbaar te stellen en delen van de scriptie te kopiëren voor persoonlijk gebruik.

Elk ander gebruik valt onder de beperkingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze scriptie.”

Jeroen Famaey, juni 2007

Intelligent overlay multicasten voor QoS-gevoelige multimediasdiensten

door

Jeroen FAMAHEY

Scriptie ingediend tot het behalen van de academische graad van
licentiaat informatica

Academiejaar 2006–2007

Promotor 1: Prof. Dr. Ir. B. DHOEDT
Promotor 2: Prof. Dr. Ir. F. DE TURCK
Scriptiebegeleider: B. DE VLEESCHAUWER
Faculteit Ingenieurswetenschappen
Universiteit Gent

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

Samenvatting

Het internet, in zijn huidige vorm, biedt geen ondersteuning voor end-to-end multicasting en biedt geen garanties in verband met QoS. Door gebruik te maken van een overlay netwerk, een virtuele netwerklaag bovenop het bestaande internet, kunnen end-to-end multicastsessies worden ondersteund. Daarbovenop kunnen garanties worden geboden in verband met de maximale end-to-end delay, wat de QoS verbetert.

Er wordt een Overlay Multicast Architectuur voorgesteld die many-to-many multicast ondersteuning aanbiedt aan multimedietoepassingen, zoals: video on demand, video conferenties en massive multiplayer online games. Hierbij wordt zo kostenefficiënt mogelijk gerouteerd en rekening gehouden met een delay bound, wat de QoS verbetert.

Bij het configureren van het overlay netwerk maakt het platform gebruik van een aantal algoritmen, voor het kiezen van een Rendezvous Point (RP) en het berekenen van de multicast distributiebomen. Het RP is een centraal punt in het netwerk, waar alle dataverkeer van een overlay multicastsessie doorgaat. Het kan onder andere gebruikt worden voor pakketverwerking, zoals het bufferen van stromen bij videoconferenties. Het ontworpen Multicastboom Algoritme maakt gebruik van een algemene takkost om de multicast distributie bomen op een kost- en bandbreedte-efficiënte manier te kiezen. Met behulp van een delay bound kan een maximale end-to-end delay worden opgelegd.

Trefwoorden

overlay netwerk, many-to-many, multicast, QoS

Intelligent overlay multicasting for QoS-sensitive multimedia services

Jeroen Famaey

Supervisor(s): Bart De Vleeschauwer, Bart Dhoedt, Filip De Turck

Abstract—Many multimedia services such as IP-TV, video conferencing and online gaming need to send the same data to multiple clients. To avoid having to send a copy from the sender to every receiver, multicasting can be used. Many of these services also need guarantees concerning the Quality of Service (QoS).

In this paper we propose the use of an overlay network to resolve these issues. We implemented a software platform that automatically configures the overlay servers to support many-to-many multicastsessions. A number of algorithms were designed that intelligently calculate the multicast distribution trees to minimize the used bandwidth and end-to-end delay.

Keywords—overlay network, many-to-many, multicast, QoS

I. INTRODUCTION

THE internet, in its current form, offers no support for end-to-end multicasting. Although IP layer multicasting is supported by some domains, it is not widely implemented, and as such cannot be used for end-to-end communication. End-to-end many-to-many multicast services can be offered by using an overlay network [1]. By intelligently choosing the multicast distribution trees, the available bandwidth can be more efficiently used and guarantees can be offered regarding the maximum end-to-end delay, which significantly improves QoS.

We have developed a software platform, called the Overlay Management Platform (OMP). It configures the overlay servers in function of a set of senders and receivers to support many-to-many overlay multicasting. For every multicastsession, one of the overlay servers is selected as Rendezvous Point (RP). The reason for this is twofold. On one hand the complexity of the multicast distribution trees is reduced, lessening the configuration overhead. On the other hand it allows the architecture to support multimedia applications that need a server side component. The OMP supports this in the form of server plugins, running on the RP.

To allow the OMP to intelligently select an RP and calculate the multicast distribution trees, two algorithms were designed. The RP Algorithm selects a suitable RP for each multicastsession. The multicast tree algorithm calculates a multicast distribution tree for each sender of the session, that contains all receivers. The algorithm minimizes bandwidth usage and improves QoS (by taking into account a delay bound).

The components of the architecture and their function are discussed in section II. In section III a description of the two designed algorithms is given. Section IV gives some of the more important results obtained from the tests used to evaluate the algorithms. Conclusions are drawn in section V.

II. OVERLAY MULTICAST ARCHITECTURE

The architecture consists of the Overlay Management Platform (OMP), a web interface, the Data Management Platform

(DMP) and the overlay servers themselves. Fig. 1 gives an overview of the different components that constitute the architecture. The arrows show which components interact with each other and what technology is used for the interaction. For completeness the figure also shows how the components interact with users and administrators. In the rest of this section the function of each component is given and the technologies used to implement them will be briefly discussed.

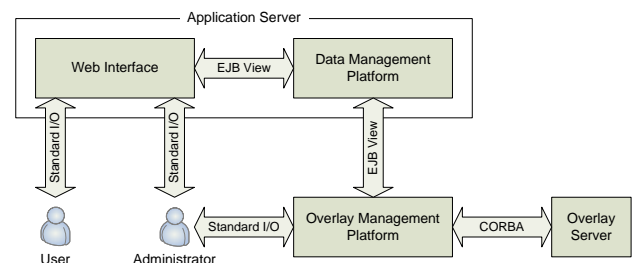


Fig. 1. The components of the architecture

A. Data Management Platform

The DMP is used to persistently store information about multicastsessions, the overlay topology and server plugins. The component consists of a front-end and a relational database, which is used to store the actual data. The front-end was implemented using Java EE technologies. Entity classes are used as an abstraction on top of the database itself. Each entity class represents one or more tables in the database. Each entity instance represents a row in a table.

The DMP contains a number of facades, which are objects that are called by the other components of the architecture to communicate with it. Each facade is a stateless session bean with a remote view (an interface defining methods that can be called by remote clients). There is one facade for the web interface and one for the OMP.

B. Web Interface

The web interface allows users and administrators to interact with the other components of the architecture. It allows normal users to create new multicastsessions, register as sender and/or receiver for an existing session and upload new server plugins. Administrators can perform a few additional tasks, such as viewing information about the overlay topology.

The web interface was also implemented using Java EE and consists of a number of JavaServer Pages (JSP) and Servlets which dynamically generate the content of the webpages.

C. Overlay Management Platform

The OMP is responsible for configuring the overlay servers for each multicast session. At regular intervals it polls the DMP to see if there are any multicast sessions that are about to start. For every multicast session an RP is selected and multicast distribution trees are calculated. Finally the trees are converted to routing rules, which are transmitted to the correct overlay servers. If the session needs to use a server side component, the OMP contacts the RP and asks it to start the correct plugin.

The OMP was implemented as a stand-alone Java application. It communicates with the overlay servers using CORBA [2] and with the DMP via the remote view of a facade session bean.

D. Overlay Servers

The overlay servers route datapackets throughout the overlay network from senders to receivers. The routing logic was implemented using the Click modular router technology [3]. On top of it is a Java front-end, responsible for communicating with the OMP, configuring the click router and starting/stopping plugins.

Through elaborate testing it was shown that the overlay servers themselves were able to process around 100 Mbps and on average only added 57 microseconds to the one way delay.

III. ALGORITHMS

As stated before, bandwidth usage can be minimized and QoS can be improved by intelligently selecting the RP multicast distribution trees. For this two algorithms were designed.

The path from the sender to each receiver, in the multicast distribution tree, must contain the RP. Therefore the multicast distribution trees are constructed out of a path (from the sender to the RP) and a tree (rooted at the RP, which contains all receivers) that will be called the RP-tree. Although the path is separate for each sender, multiple senders may share the same RP-tree. Fig. 2 shows an example of a multicast session. The paths to the RP (Overlay Server 4) are shown in green and red. The senders share the same RP-tree (shown in blue).

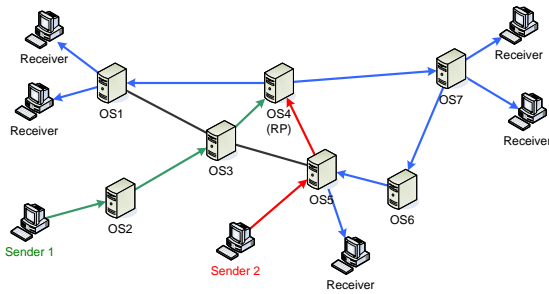


Fig. 2. An example multicast session with two senders and five receivers

Two weights are associated with the overlay edges. They represent the cost to reserve an edge (to minimize bandwidth usage) and the end-to-end delay (to improve QoS).

A. RP Algorithm

The total cost of all multicast distribution trees of a multicast session depends on the chosen RP. The algorithm estimates this cost for each possible RP and selects the one with the lowest estimated cost. In the first step a fast, but inaccurate, estimation

is used. The K best candidates are selected and passed to the second step. There a more accurate, but slower, estimation of the total cost is made, and the best candidate is selected.

A higher value of K will give a more accurate result, while a lower value will make the algorithm run faster.

B. Multicast Tree Algorithm

This algorithm calculates a multicast distribution tree for each sender. It minimizes the cost of each tree and keeps the delay from the sender to all receivers below a given delay bound (if possible). The algorithm consists of three steps:

- 1. Path Selection:** In this step all paths from each sender to the RP that do not cross the delay bound are searched. Afterwards several path selection criteria are used to remove paths that cannot lead to an optimal solution.
- 2. Clustering:** The senders are divided into a number of clusters. All the senders in a cluster will use the same RP-tree.
- 3. RP-Tree Selection:** In the final step the algorithm selects an RP-tree for each cluster. The paths found in the first step are taken into account, as different path delays lead to RP-trees with different costs. The best path for each sender is also chosen in this step. For calculating the RP-trees an existing algorithm, such as the one proposed in [4], can be used.

IV. EVALUATION

In this section the most important results of the tests that were performed on the algorithms will be briefly discussed.

After executing the path selection criteria only 1 or 2 paths were left for over 90% of the senders. This greatly reduces the number of RP-trees that have to be calculated in the last step.

If all the senders were put into a single cluster the total cost was less than 9% higher than when every cluster contained only one sender, but the algorithm executed up to 100 times faster. This shows that the clustering step can speed up the algorithm considerably, resulting in only a slightly higher cost.

As the discussed algorithms are heuristics, their solution is suboptimal. Overall the total cost of the solution generated by the algorithm was, for 3 or more clusters, always less than 6% higher than the optimal cost, when using an optimal algorithm to calculate the RP-trees.

V. CONCLUSIONS

We showed that by using overlay multicasting some of the limitations of the internet, and more specifically of IP multicast, can be overcome. We have developed a platform that supports end-to-end many-to-many multicast sessions, taking into account a delay bound and minimizing bandwidth usage.

REFERENCES

- [1] D. Clark, B. Lehr, S. Bauer, P. Faratin, R. Sami, and J. Wroclawski, "Overlay networks and the future of the internet," *Communications & strategies*, vol. 3, no. 63, pp. 1–21, 2006.
- [2] Object Management Group, "Common object request broker architecture: Core specification," <http://www.omg.org/docs/formal/04-03-12.pdf>, 2004.
- [3] E. Kohler, R. Morris, C. Benjie, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.
- [4] B. De Vleeschauwer, F. De Turck, B. Dhoedt, and P. Demeester, "Online management of QoS enabled overlay multicast services," in *Proceedings of IEEE GLOBECOM 2006*, San-Francisco, USA, 2006.

Inhoudsopgave

1	Inleiding	1
2	Situering	3
2.1	Overlay Netwerk	3
2.2	IP Multicasting	5
2.3	Overlay Multicasting	6
2.4	Technologieën	8
2.4.1	UDP	8
2.4.2	CORBA	9
2.4.3	Java EE	10
2.4.4	Click Modular Router	11
3	Overlay Multicast Architectuur	12
3.1	Algemeen	13
3.2	Data Management Platform	17
3.2.1	Entities	17
3.2.2	Entity Facades	20
3.2.3	Nearest Overlay Server Finder	22
3.2.4	Encryption Unit	22
3.2.5	Client Facade	23
3.2.6	Management Facade	23
3.3	Web Interface	24
3.4	Overlay Management Platform	26

3.4.1	Database Communication Module	28
3.4.2	Server Plugin Handler	28
3.4.3	Overlay Communication Module	29
3.4.4	Graphical User Interface	30
3.4.5	Port Generator	30
3.4.6	Multicast Algorithms	31
3.4.7	Multicast Session Retriever	31
3.4.8	Multicast Tree Calculator	31
3.5	Overlay Servers	33
3.5.1	Management Communication Module	33
3.5.2	Server Plugin Handler	35
3.5.3	Click Interface	35
3.5.4	Click Router	35
3.5.5	Performantietests	36
4	Algoritmen	41
4.1	Probleemstelling	41
4.2	Rendezvous Point Algoritme	42
4.2.1	Initiële Schatting	43
4.2.2	Rendezvous Point Selectie	44
4.3	Multicastboom Algoritme	46
4.3.1	Pad Selectie	46
4.3.2	Clustering	48
4.3.3	Boom Creatie	52
4.4	Optimaal Algoritme	55
5	Evaluatie van de Algoritmen	57
5.1	Rendezvous Point Algoritme	58
5.1.1	Initiële Schatting	58
5.1.2	RP Selectie met Initiële Schatting	60

5.1.3	Besluit	63
5.2	Clustering Algoritme	63
5.3	Multicastboom Algoritme	65
5.3.1	Pad Selectie	66
5.3.2	Clustering	69
5.3.3	Volledig Algoritme	69
5.3.4	Besluit	73
6	Verder Onderzoek	74
6.1	Online Compileren van Plugins	74
6.2	Beveiliging	75
6.3	Dynamische Multicastbomen	75
7	Conclusies	77
A	Virtual Meeting Place	80
A.1	Protocol	80
A.2	Client	82
A.3	Server	83
B	Inhoud CD-ROM	85

Lijst Met Afkortingen

BLOB	Binary Large Object
CMP	Container-Managed Persistence
CORBA	Common Object Request Broker Architecture
DCMST	Diameter Constrained Minimal Steiner Tree
DMP	Data Management Platform
EJB	Enterprise Java Bean
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ILP	Integer Linear Programming
IP	Internet Protocol
Java EE	Java Platform, Enterprise Edition
JSP	Java Server Page
MDT	Minimal Delay Tree
MICD	Minimal Intra Cluster Dissimilarity Algorithm
MinMaxD	MinMax End-to-End Delay
OCM	Overlay Communication Module
OMA	Overlay Multicast Architecture
OMP	Overlay Management Platform
PAM	Partition Around Medoids
QoS	Quality of Service
RMI	Remote Method Invocation
RP	Rendezvous Point
SC	Silhouette Coefficient
SUB	Subnetwork Heuristic Algorithm
TLS	Transport Layer Security
TTL	Time to Live
UDP	User Datagram Protocol
VMP	Virtual Meeting Place
WTC	Worst Case Total Multicast Cost
WTrC	Worst Case Multicast Tree Cost

Hoofdstuk 1

Inleiding

Multimediadiensten op het internet vervullen een steeds grotere rol in de moderne maatschappij. Enkele voorbeelden van belangrijke multimedia netwerktoepassingen zijn: video on demand, IPTV, videoconferenties en massive multiplayer online games. Bij veel van deze diensten worden vaak dezelfde gegevens tegelijkertijd naar verschillende gebruikers gestuurd. Als een identiek kopie van de data wordt verzonden naar elke ontvanger, wordt inefficiënt gebruik gemaakt van de beschikbare bandbreedte. Vermits het verantwoord gebruiken van bandbreedte steeds belangrijker wordt, kunnen de gegevens met behulp van multicasttechnieken op een efficiënte manier verstuurd worden van een verzender naar meerdere ontvangers.

Het internet bestaat uit een groot aantal subnetwerken [15]. Niet al deze netwerken bieden echter ondersteuning voor multicasting, waardoor niet end-to-end kan gemulticast worden over het internet. Daarbovenop worden ook geen garanties geboden in verband met Quality of Service (QoS), zoals de maximale end-to-end delay tussen de verzenders en ontvangers.

Deze problemen kunnen worden opgelost door gebruik te maken van een overlay netwerk. Dit is een virtuele netwerklaag bovenop het bestaande internet. Op deze overlay laag kunnen multicastdiensten worden aangeboden. Door de multicast distributiebomen op een intelligente manier te kiezen, kan het bandbreedte gebruik geminimaliseerd worden en kunnen garanties worden geboden in verband met de maximale end-to-end delay, wat de QoS ten goede komt.

Het overlay netwerk bestaat uit een aantal, over het internet verspreide, overlay servers. Ze worden in functie van de multicastsessies geconfigureerd door een Overlay Management Platform (OMP). Het OMP en de overlay servers vormen samen de Overlay Multicast Architectuur

(OMA). Daarbovenop bevat de OMA nog een aantal extra componenten die instaan voor de interactie met gebruikers en de opslag van gegevens.

De configuratie van de overlay servers in functie van de verzenders en ontvangers van een multicastsessie gebeurt met behulp van één of meerdere multicast distributiebomen. Bij veel multimedietoepassingen is er ook nood aan een centraal punt waar de gegevens kunnen verwerkt worden, voordat ze worden doorgestuurd naar de ontvangers. Daarom wordt voor elke sessie één van de overlay servers als Rendezvous Point (RP) aangeduid. De OMA biedt ondersteuning voor serverside plugins, die instaan voor het verwerken van datapakketten op het RP.

Het op een intelligente manier kiezen van de distributiebomen en RPs gebeurt met behulp van een aantal algoritmen. Deze ondersteunen many-to-many multicastsessies door voor elke verzender, van de sessie, een multicast distributieboom te berekenen. Bij veel multimedietoepassingen is de QoS en meer specifiek de maximale end-to-end delay zeer belangrijk. Bij het berekenen van de multicast distributiebomen wordt aan de algoritmen een delay bound meegegeven. Dit is een maximale delay grens, tussen elke verzender en ontvanger, die niet mag worden overschreden. De algoritmen zullen hiermee, in de mate van het mogelijke, rekening houden. Om de beschikbare bandbreedte efficiënt te benutten, wordt bovenop de delay nog een tweede, algemene takkost gebruikt. Deze wordt geminimaliseerd over elke boom.

In hoofdstuk 2 worden een aantal belangrijke begrippen in verband met overlay netwerken en multicasting gesitueerd en verklaard. Er wordt ook een kort overzicht gegeven van de belangrijkste technologieën die gebruikt werden bij de implementatie van de componenten van de OMA. In hoofdstuk 3 wordt een gedetailleerde beschrijving gegeven van de architectuur. Zowel de implementatie van de componenten, als hun functie en mogelijkheden worden besproken. Op het einde van het hoofdstuk wordt nog een kort overzicht gegeven van een aantal testresultaten in verband met de routeringsperformantie van de geïmplementeerde overlay servers. In hoofdstuk 4 worden een aantal algoritmen besproken voor het kiezen van een RP en het berekenen van de multicast distributiebomen. In hoofdstuk 5 wordt de performantie en kwaliteit van de oplossing van de ontworpen algoritmen getest. Tenslotte worden in de twee laatste hoofdstukken een aantal voorstellen gedaan voor verder onderzoek en kort de conclusies bijgelicht.

In bijlage A wordt een testapplicatie besproken die werd gebruikt om de werking van de OMA en bijhorende algoritmen te testen. In bijlage B wordt een kort overzicht gegeven van de inhoud van de CD-ROM, die bij de ingebonden versie van dit boek werd geleverd.

Hoofdstuk 2

Situering

In dit hoofdstuk worden een aantal belangrijke begrippen en concepten gesitueerd en verklaard. Vervolgens worden ook de belangrijkste technologieën, die gebruikt werden bij het implementeren van de architectuur, kort toegelicht.

2.1 Overlay Netwerk

Een overlay netwerk is een, over het internet verspreide, set van servers (overlay servers genoemd) dat [4]:

- Een infrastructuur voorziet voor één of meerdere applicaties.
- Applicatie-data op een andere manier doorstuurt en verwerkt dan de standaard internet-diensten.
- Op een coherente wijze door een derde partij kan worden geconfigureerd en georganiseerd.

Het is dus een virtueel netwerk, bovenop het bestaande internet, dat extra functionaliteit aanbiedt. Voorbeelden hiervan zijn: content distributie, beveiliging, multicast en QoS.

We beschouwen het overlay netwerk als een extra netwerklaag bovenop de IP laag van het internet en onder de applicatielaag. Elke overlay server wordt in het IP netwerk geplaatst en stemt dus overeen met een node van dit netwerk. Overlay servers zijn onderling verbonden via edges (ook links genoemd). Elke overlay edge stemt overeen met een pad op de IP laag, dat de

begin- en eindnode van de edge met elkaar verbindt in het IP netwerk. In veel gevallen is het overlay netwerk full mesh, wat wil zeggen dat elk paar overlay servers rechtstreeks met elkaar verbonden is.

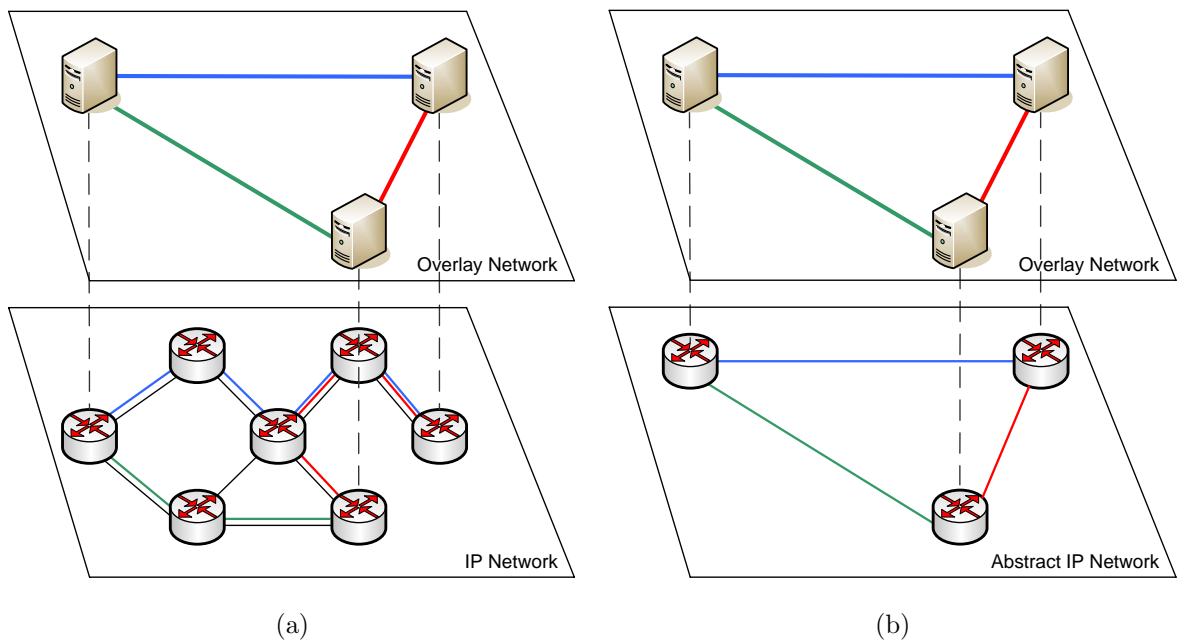
Omdat bij het multicasten rekening moet worden gehouden met een aantal factoren, worden aan elke IP edge twee takkosten verbonden. Er wordt aan elke tak een algemene kost toegekend, die we verder gewoon “kost” zullen noemen. Bij het berekenen van de multicast distributiebomen wordt deze kost zoveel mogelijk geminimaliseerd. Hierdoor kan bijvoorbeeld een lagere kost worden toegekend aan edges die prioritair gebruikt moeten worden of kunnen alle edges een gelijke kost krijgen, zodat het aantal edges per boom zoveel mogelijk beperkt wordt. Er wordt daarnaast nog een tweede takkost toegekend, namelijk de delay. Deze is evenredig met de tijd om een bericht te sturen van de begin- naar de eindnode van de edge. Beide takkosten zijn additief, zodat zowel de kost als de delay van een overlay edge worden gelijk gesteld aan de som van de takkosten van het onderliggende IP pad.

Het IP pad horende bij een overlay edge kan op een aantal manieren gekozen worden. Voorbeelden zijn:

- Het IP pad met de laagste totale delay.
- Het IP pad met de laagste totale kost.
- Het kortste hop pad. Dit is het pad dat het minst aantal IP edges (en dus ook nodes) bevat.

Het is mogelijk dat, afhankelijk van de situatie, de beschikbare informatie over het IP netwerk beperkt is. In dat geval kan er abstractie worden gemaakt van de IP laag. De informatie over IP nodes, die niet samenvallen met een overlay server, kan dan worden weggelaten. Elke overlay edge komt dan overeen met de IP edge (in plaats van een pad) die dezelfde servers met elkaar verbindt in het IP netwerk.

Fig. 2.1 toont een overlay netwerk met onderliggend IP netwerk. De overlay servers worden verbonden met hun overeenkomstige IP node via een verticale stippellijn. De paden op de IP laag worden in dezelfde kleur weergegeven als hun overeenkomstige overlay edge. Merk op dat eenzelfde IP edge tot meerdere paden kan behoren. In Fig. 2.1(b) wordt een abstractie van het IP netwerk getoond. Dit bevat enkel de IP nodes die overeenstemmen met een overlay server. De IP paden worden vervangen door één edge.



Figuur 2.1: Een overlay netwerk met onderliggend IP netwerk (a) Informatie over het volledige IP netwerk wordt getoond (b) Hetzelfde overlay netwerk met een abstracte versie van het onderliggend IP netwerk

2.2 IP Multicasting

Er zijn een aantal mogelijkheden voor het connecteren van verzenders en ontvangers:

- Unicast: Dit is een connectie tussen één verzender en één ontvanger.
- One-to-many Multicast: Hierbij wordt één verzender verbonden met één of meerdere ontvangers.
- Many-to-many Multicast: Hierbij worden één of meerdere verzenders verbonden met één of meerdere ontvangers.

Het is duidelijk dat unicast en one-to-many multicast speciale gevallen zijn van many-to-many multicast verbindingen.

De meeste communicatie over het internet gebeurt via unicastconnecties. Er wordt, in beperkte mate, ook ondersteuning geboden voor one-to-many en many-to-many multicastconnecties met behulp van IP layer multicast. Ontvangers die geïnteresseerd zijn in het ontvangen van een bepaalde datastream kunnen zich inschrijven in een multicast groep, met behulp van het

Internet Group Management Protocol (IGMP) [3]. Aan elke groep wordt een IP multicast adres toegekend, dat zich tussen 224.0.0.0 en 239.255.255.255 bevindt. Voor IP multicasting over het internet worden adressen van 224.0.1.0 tot 238.255.255.255 gebruikt.

Er wordt een onderscheid gemaakt tussen twee types van protocols [3]. De Intradomain Multicast Protocols worden binnen een multicastdomein gebruikt. Een voorbeeld hiervan is Protocol Independent Multicast (PIM). De Interdomain Multicast Protocols worden gebruikt om multicastverkeer te routeren tussen de verschillende multicastdomeinen. Een voorbeeld hiervan is het Multicast Source Discovery Protocol (MSDP), dat verzenders detecteert die zich in een ander domein bevinden.

IP multicasting heeft een aantal beperkingen waardoor het niet bruikbaar is om end-to-end multicastdiensten aan te bieden over het hele internet:

- Het aantal IP multicast adressen is beperkt. Daarbovenop worden telkens 32 verschillende IP multicast adressen afgebeeld op hetzelfde MAC adres [3], wat het aantal mogelijkheden nog kleiner maakt.
- Niet elke domein biedt ondersteuning voor intra- en interdomein multicast protocols. Hierdoor is end-to-end multicasting over het hele internet niet mogelijk.

Door deze beperkingen van IP multicast, stellen we voor om gebruik te maken van een overlay netwerk. Hierdoor wordt het, met behulp van overlay multicasting, wel mogelijk om end-to-end multicastdiensten op het internet te ondersteunen.

2.3 Overlay Multicasting

Als alternatief voor IP Multicasting kan, met behulp van een overlay netwerk, Overlay Multicasting worden aangeboden. In dit boek stellen we het gebruik van een software platform, het Overlay Management Platform (OMP) genaamd, voor. Het doel van dit platform is het aanbieden van many-to-many overlay multicastsessies. Via een gebruikersinterface kunnen gebruikers deze multicastsessies configureren. Het platform zal dan, met behulp van de opgegeven sessie informatie, het overlay netwerk automatisch configureren. In de volgende paragrafen worden nog een aantal begrippen in verband met overlay multicasting die verder in dit boek aan bod komen verklaard en gedefinieerd.

Elke multicastsessie bestaat uit een set van verzenders en ontvangers. Met elke verzendende en ontvangende client is een overlay server verbonden. Deze wordt de toegangsserver of dichtstbijzijnde overlay server genoemd. De communicatie tussen elke client en het overlay netwerk gebeurt via zijn toegangsserver. In de context van de algoritmen zijn de verzenders en ontvangers deze toegangsservers en niet de clients zelf. Een overlay server wordt beschouwd als een verzender als hij dienst doet als toegangsserver voor tenminste één verzendende client. Dit is analoog voor een ontvangende overlay server. In de context van de architectuur wordt, indien nodig, steeds vermeld of het om verzendende (ontvangende) clients of overlay servers gaat. Bij het berekenen van de multicast distributiebomen wordt geen rekening gehouden met de clients zelf. Hoe de routing tussen de clients en hun toegangsservers gebeurt, wordt dan ook in het midden gelaten. Er kan bijvoorbeeld een unicast verbinding worden opgezet tussen elke client en zijn toegangsserver. Als een toegangsserver meerdere clients bedient, kan, als er ondersteuning voor wordt geboden, gebruik worden gemaakt van IP multicasting.

Zoals eerder vermeld is er bij veel multimedietoepassingen nood aan een centraal punt waar de data verwerkt kan worden, voor ze wordt doorgestuurd naar de ontvangers. Bij video-toepassingen kan de data bijvoorbeeld worden gebuffered en bij online games kan worden nagegaan of de acties van de spelers voldoen aan de regels van het spel. Voor elke sessie wordt een Rendezvous Point (RP) gekozen, dat dienst doet als centraal punt.

Bij elke multicastsessie hoort een delay bound. Deze wordt gedefinieerd als de maximaal toegelaten delay tussen elke verzender en ontvanger van de sessie. Bij het berekenen van de multicast distributiebomen wordt ervoor gezorgd dat de delay bound (in de mate van het mogelijke) niet wordt overschreden. Om de end-to-end delay zo laag mogelijk te houden, wordt de delay bound meestal vrij streng ingesteld. Bijvoorbeeld als het maximum van de minimale delays tussen elke verzender en ontvanger (waarbij het pad tussen de verzenders en ontvangers het RP moet bevatten).

Bij elke verzender van de multicastsessie hoort een multicast distributieboom (ook gewoon multicastboom genoemd). Deze boom geeft aan op welke manier de data van de verzender naar elke ontvanger moet worden gerouteerd. Omdat het RP zich op het pad van de verzender naar elke ontvanger moet bevinden, wordt de multicastboom opgesplitst in twee delen. Hij bestaat uit een pad van de verzender naar het RP en een boom met als wortel het RP, die alle ontvangers bevat. Het boomgedeelte van de multicastboom wordt ook de RP-boom genoemd. Aangezien een overlay server zowel tot het pad als de RP-boom van een multicast distributieboom kan

behoren, is dit geen boom in de strikte zin van het woord. De RP-boom is dit wel.

Hoewel elke verzender een verschillend pad heeft naar het RP, kunnen meerdere verzenders wel dezelfde RP-boom delen. Als alle verzenders dezelfde RP-boom gebruiken wordt gesproken over gedeelde multicastbomen (shared trees). Als elke verzender over zijn eigen RP-boom beschikt, heet dit verzender multicastbomen (source trees). Tusseloplossingen, waar een aantal verzenders een RP-boom delen, zijn ook mogelijk.

De kost van een pad wordt gedefinieerd als de som van de kosten van alle takken van het pad. De delay van een pad en de kost van een boom worden analoog gedefinieerd. De delay van een boom is gelijk aan de maximale delay van de wortel naar één van de bladeren. De kost (delay) van een multicast distributie boom is gelijk aan de som van de kosten (delays) van zijn pad (van de verzender naar het RP) en zijn RP-boom. De totale kost van een multicastsessie is gelijk aan de som van de kosten van de multicastbomen van de verzenders. Als n verzenders van de sessie dezelfde RP-boom delen, wordt de kost ervan n keer bij de totale kost van de sessie geteld.

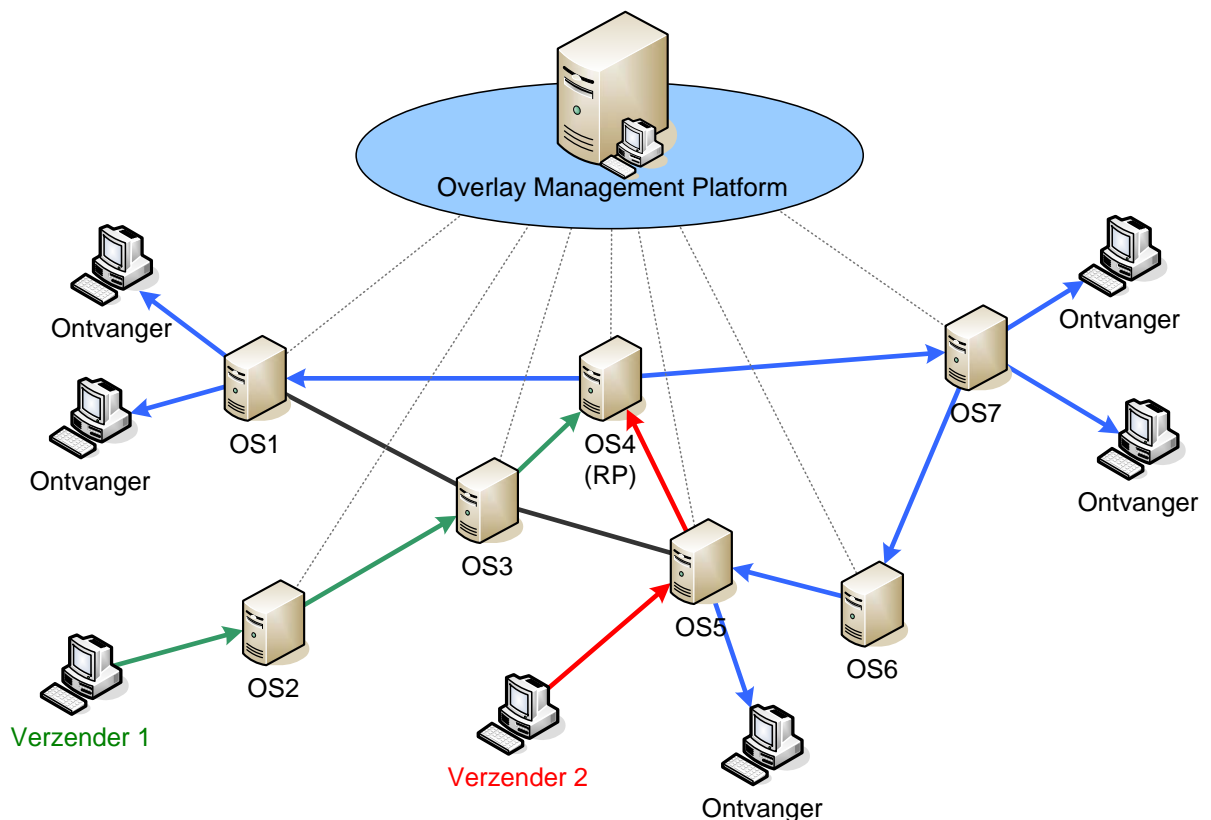
Fig. 2.2 toont een multicastsessie met twee verzenders en vijf ontvangers. De multicast distributieboom van Verzender 1 bestaat uit het pad naar het RP aangegeven met de groene pijlen en de RP-boom aangegeven met de blauwe pijlen. De tweede verzender zijn pad naar het RP is aangegeven met de rode pijlen en gebruikt dezelfde RP-boom. Overlay server 4 doet dienst als RP. Overlay servers 2 en 5 zijn verzendende overlay servers en 1, 5 en 7 zijn ontvangers. Het OMP (bovenaan de figuur) is verbonden met elke overlay server en staat in voor de configuratie van hun routingstabellen in functie van de multicast distributieboomen.

2.4 Technologieën

2.4.1 UDP

Het User Datagram Protocol (UDP) is een veelgebruikt transportprotocol op het internet[18]. Het is connectieloos en biedt als enige diensten het multiplexen en demultiplexen van pakketten tussen de netwerk- en applicatielaag en een rudimentaire foutcontrole (met behulp van een checksum) [15].

De overlay servers gebruiken UDP om pakketten te forwarden voor de multicastsessies. Ook voor de communicatie tussen de clients en hun toegangsservers wordt dit protocol gebruikt.



Figuur 2.2: Een multicastsessie met twee verzenders, die hun RP-boom delen (aangegeven door de blauwe pijlen). Hun paden naar het RP (Overlay Server 4) worden respectievelijk aangegeven door de groene en rode pijlen

2.4.2 CORBA

Common Object Request Broker Architecture (CORBA) is een Remote Method Invocation (RMI) standaard, voorgesteld door de Object Management Group (OMG) [11, 5, 7]. Het laat processen toe om methoden van objecten, die zich in een ander proces bevinden, op te roepen. De methoden die vanuit een ander proces oproepbaar zijn, worden gedefinieerd in een remote interface, die met behulp van de CORBA Interface Definition Language (IDL) wordt gespecificeerd. Elk remote object bezit een remote object referentie. Dit is een uniek identificatienummer, zowel in de tijd als in de ruimte. Bij het oproepen van de methoden van een remote object wordt steeds zijn remote object referentie gebruikt.

Het grote voordeel van CORBA ten opzichte van andere RMI standaarden, zoals Java RMI, is dat er geen beperkingen worden opgelegd in verband met de gebruikte programmeertaal. Het is dan ook mogelijk om twee processen, geschreven in een verschillende programmeertaal, met

elkaar te laten communiceren via CORBA.

CORBA wordt gebruikt voor de communicatie tussen het OMP en de overlay servers. Door de taalafhankelijkheid van CORBA is het mogelijk om in de toekomst bijvoorbeeld een C of C++ implementatie te gebruiken voor de overlay servers, zonder dat het OMP hierdoor aangepast moet worden.

2.4.3 Java EE

Java Platform, Enterprise Edition (Java EE) is een programmeerplatform, gebaseerd op de Java programmeertaal [2]. Het uitvoeren van Java EE applicaties gebeurt met behulp van een Application Server (bijvoorbeeld de Sun Java System Application Server). Java EE bestaat uit een aantal verschillende technologieën. Degenen die in de implementatie van de OMA gebruikt worden zijn:

- **Enterprise Java Beans (EJB):** Een modulaire component die gebruikt wordt voor het implementeren van business logica. EJBs bevinden zich in een bean container. Dit is een onderdeel van de application server. De methoden van een EJB die door andere componenten kunnen worden opgeroepen worden gedefinieerd in een view. Dit is een interface die een subset van de methoden van de EJB bevat. Een local view kan gebruikt worden door andere EJBs in dezelfde bean container. Via een remote view kunnen andere componenten (niet noodzakelijk EJBs) van buiten de bean container communiceren met de EJB.
- **Entities:** Dit zijn objecten die gebruikt worden voor het persistent bijhouden van gegevens. Ze dienen als abstractie bovenop een relationele databank. Elke Entity Klasse stelt typisch een tabel uit de databank voor. Een instantie van de klasse komt overeen met een rij van die tabel. EJBs kunnen, via entities, toegang krijgen tot een databank.
- **Java Servlets:** Een klasse die clients toegang verleent tot een server, gebruik makend van het request-response programmeermodel. Vaak worden ze gebruikt, bij webapplicaties, om HTTP requests te verwerken en een gepast antwoord te genereren.
- **JavaServer Pages (JSP):** Een JSP heeft in se dezelfde functionaliteit als een Java Servlet. Het grootste verschil is dat de JSP bestaat uit statische data (bijvoorbeeld HTML

code) met ingebedde programmacode (voor het dynamisch genereren van inhoud). Een servlet bestaat uit programmacode met ingebedde statische inhoud. Een JSP is dan ook meer geschikt voor toepassingen waar de nadruk op weergave ligt. In meer complexe applicaties worden JSPs en Java Servlets vaak samen gebruikt.

Bij de implementatie van het Data Management Platform (DMP), een component van de OMA die instaat voor het persistent bijhouden van informatie, werd gebruik gemaakt van EJBs en entities. Bij de implementatie van de Web Interface, een component die gebruikers toelaat om met de andere componenten van de OMA te communiceren, werden Java Servlets en JSPs gebruikt. De Web Interface en het OMP communiceren met het DMP via een aantal remote EJB views.

2.4.4 Click Modular Router

Click is een software architectuur die toelaat om flexibele en configureerbare routers op te bouwen, uit verschillende modules (ook elementen genoemd) [14]. Ieder element is in staat om inkomende pakketten te verwerken en terug door te sturen naar één of meerdere uitgangen. De router wordt gemodeleerd als een gerichte graaf. De Click elementen bevinden zich op de knopen. Pakketten worden doorgestuurd langsheen de takken (edges) van de graaf.

De routeringslogica van de overlay servers werd geïmplementeerd met behulp van een Click modular router.

Hoofdstuk 3

Overlay Multicast Architectuur

In dit hoofdstuk wordt een beschrijving gegeven van de ontworpen Overlay Multicast Architectuur (OMA) en de componenten waaruit deze bestaat. De functie van de architectuur kan kort omschreven worden als:

“Het aanbieden van een platform dat many-to-many multicastsessies ondersteunt met behulp van een overlay netwerk, waarbij zo kostenefficiënt mogelijk wordt gerouteerd, rekening houdend met de QoS in de vorm van de totale end-to-end delay”

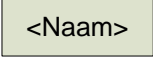

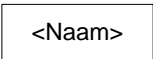
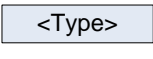
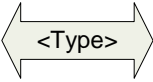



De architectuur bestaat uit een aantal componenten die elk een deel van de beschreven functionaliteit implementeren.

De interactie tussen de componenten, onderdelen van de componenten en de gebruikers wordt schematisch weergegeven met behulp van figuren. Tabel 3.1 toont een legende van de symbolen die in deze figuren gebruikt worden.

In sectie 3.1 wordt een algemeen overzicht gegeven van de componenten van de architectuur en hoe ze met elkaar communiceren. In de daaropvolgende secties wordt dieper ingegaan op elk van de componenten afzonderlijk. In sectie 3.5.5 worden tenslotte nog een aantal testresultaten besproken in verband met de routeringsperformantie van de geïmplementeerde overlay servers.

Om de architectuur te testen werd ook een testapplicatie geïmplementeerd. Informatie hierover is terug te vinden in bijlage A.

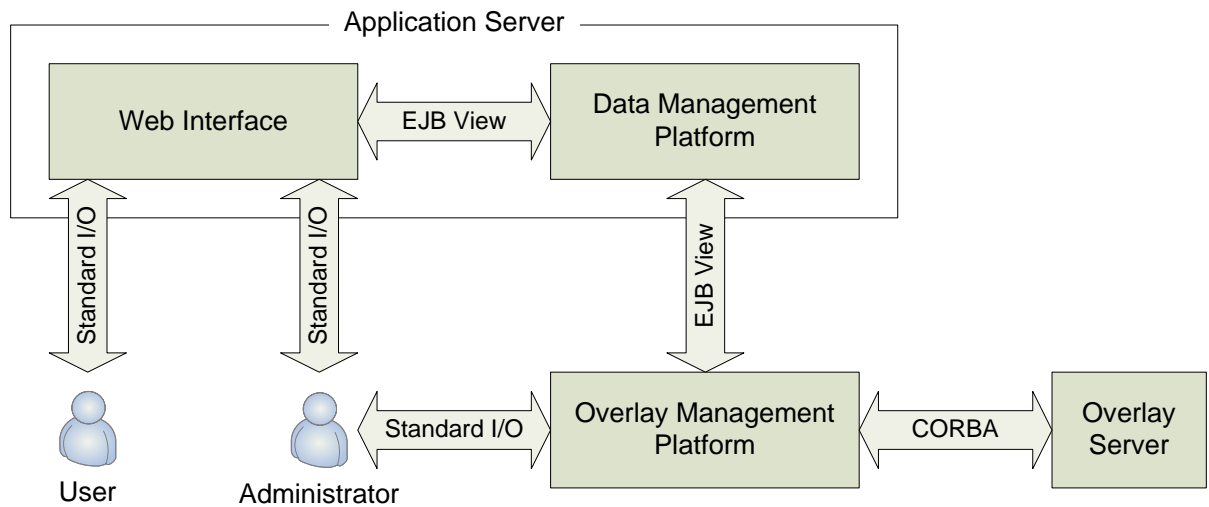
Tabel 3.1: Legende van de gebruikte symbolen in de schema's

Symbool	Betekenis
	Geeft een component van de architectuur weer. De naam van de component wordt binnenin getoond.
	De inhoud van een component. Binnenin worden de onderdelen van de component getoond.
	Een onderdeel van een component. De naam wordt binnenin getoond.
	Een EJB View. Het type (local of remote) wordt in het symbool weergegeven.
	Geeft aan dat twee componenten van de architectuur met elkaar communiceren. In de pijl wordt getoond op welke manier de communicatie verloopt.
	Geeft aan dat twee onderdelen van een component met elkaar interageren. Een interactie bestaat steeds uit een aanvraag en mogelijk een antwoord. De pijl vertrekt bij het aanvragende onderdeel.
	Een gebruiker van het systeem.
	Een relationele databank.

3.1 Algemeen

Fig. 3.1 geeft de verschillende componenten van de architectuur schematisch weer.

Het Data Management Platform (DMP) bestaat uit een Java EE front-end en een achterliggende relationele databank. De front-end houdt de details van de relationele databank verborgen voor de andere componenten van de architectuur. Hij bestaat uit een aantal Enterprise JavaBeans (EJB) en Entity Klassen. De andere componenten communiceren enkel met de EJBs en niet met de entities of de relationele databank zelf. Dit gebeurt via Remote EJB Views. Dit zijn interfaces die definities bevatten van de methoden die kunnen worden opgeroepen door clients



Figuur 3.1: Schematische voorstelling van de OMA

die zich niet in dezelfde bean container bevinden.

De interactie met gebruikers gebeurt voornamelijk via de Web Interface. Deze is geïmplementeerd met behulp van JavaServer Pages (JSP) en Servlets. De JSPs staan vooral in voor de grafische weergave en de Servlets voor de communicatie met het DMP.

Het DMP en de Web Interface worden uitgevoerd binnen dezelfde Application Server. De JSPs en Servlets bevinden zich in de web container, de EJBs in de bean container.

Het Overlay Management Platform (OMP) is een stand-alone Java applicatie. Deze voert een aantal belangrijke taken uit, zoals het berekenen van de multicastbomen en het configureren van de overlay servers. De communicatie met de overlay servers gebeurt met behulp van CORBA. Het OMP voorziet ook een Graphical User Interface (GUI) die de netwerkbeheerder toelaat om het netwerk te configureren.

De overlay servers zijn geïmplementeerd als Click routers. Daarbovenop bevindt zich een Java front-end dat instaat voor de communicatie met het OMP en de configuratie van de Click component.

Wanneer een gebruiker een nieuwe multicastsessie registreert via de webinterface, zorgen de verschillende componenten van het systeem er automatisch voor dat het overlay netwerk juist geconfigureerd wordt in functie van deze sessie. Hoe dit juist in zijn werk gaat, wordt kort in een aantal stappen getoond:

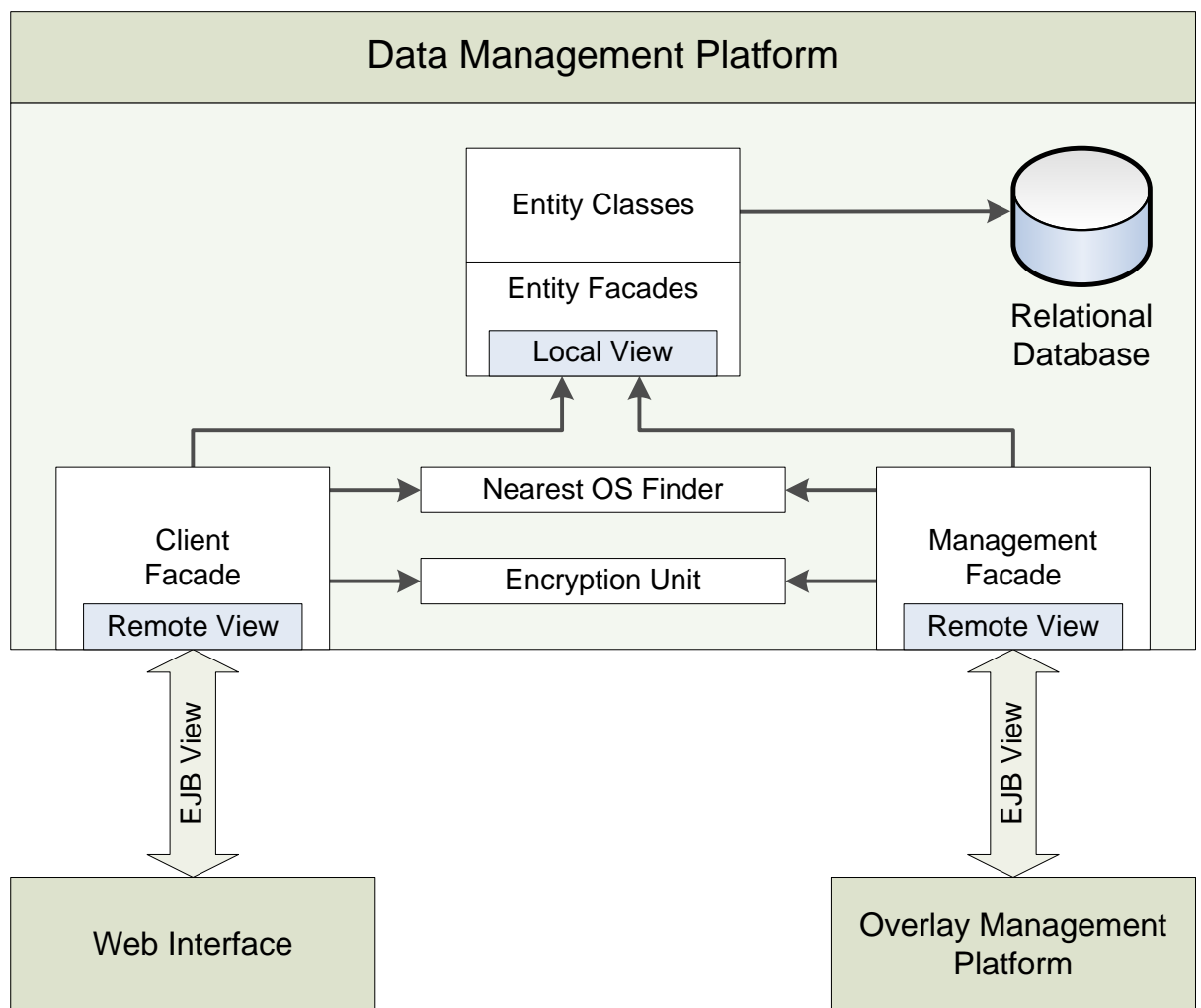
1. Een gebruiker van het systeem voegt een nieuwe multicastsessie toe via de web interface. De informatie hierover wordt doorgestuurd naar het DMP, die ze in de databank bijhoudt.
2. Een aantal gebruikers registreren zich met behulp van de web interface als verzender en/of ontvanger voor de sessie.
3. Het OMP vraagt informatie over de sessie op aan het DMP en berekent het RP en de multicastbomen.
4. Het OMP zet de multicastbomen om naar routeringsregels en stuurt deze met behulp van CORBA door naar de Java interface van de overlay servers.
5. De Java interfaces van de overlay servers configureren de onderliggende Click router in functie van de routeringsregels.
6. De overlay servers zullen nu data doorheen het overlay netwerk routeren van de verzenders naar de ontvangers van de sessie.

Bij netwerkkapplicaties is er vaak nood aan een server side component. Deze vervult, afhankelijk van de toepassing, een aantal verschillende taken. Bij online games kan er bijvoorbeeld worden gecontroleerd of de acties van de spelers voldoen aan de regels van het spel. Bij video conferenties kunnen de videostreamen gebuffered of bewerkt worden voordat ze naar de ontvangers worden gemulticast. In de OMA worden deze server componenten geïmplementeerd als server side plugins.

Gebruikers kunnen via de webinterface nieuwe plugins uploaden. Bij het aanmaken van een multicastsessie kan, indien gewenst, dan een plugin worden gekozen, die bij aanvang van de sessie op het RP wordt opgestart.

De plugins moeten voldoen aan een aantal eisen:

- Bij het opstarten van een plugin worden drie commandolijn argumenten meegegeven. Het eerste argument is een ontvangende poort. Het is de bedoeling dat de plugin luistert naar inkomende UDP datapakketten op deze poort. De twee overige argumenten zijn het IP adres en de poort waarnaar de plugin uitgaande pakketten moet verzenden. Het IP is een dummy adres, aangezien de click router de pakketten sowieso onderschept. De enige vereisten zijn dat het adres bestaat en verschillend is van het RP zijn eigen adres. Er kan bijvoorbeeld gebruik worden gemaakt van het IP adres van de gateway.



Figuur 3.2: Schematische voorstelling van het DMP

- Aangezien de Click router op een GNU/Linux systeem draait, moet de plugin ook onder dit besturingssysteem kunnen worden uitgevoerd. In hoofdstuk 6 wordt kort besproken hoe deze twee eisen kan vermeden worden.

Om de implementatie van de web interface en databank te vereenvoudigen mogen plugins bij het uploaden maar uit één bestand bestaan. Om toch plugins die uit meerdere bestanden bestaan te ondersteunen, moeten gebruikers alle bestanden van een plugin toevoegen aan een ZIP bestand, vooraleer ze deze uploaden. Dit moet ook gebeuren als de plugin maar uit één bestand bestaat.

Account :

String: username	String: password	boolean: isAdmin
------------------	------------------	------------------

Figuur 3.3: De attributen van de Account entity

3.2 Data Management Platform

Het DMP staat in voor het persistent bijhouden van informatie over het overlay netwerk, de multicastsessies en de server plugins. De component bestaat uit een relationele databank en een aantal entities. Entities zijn objecten die overeenkomen met één of meerdere tabellen in de databank. Een instantie van een entity komt overeen met een rij van een tabel. Doordat gebruik wordt gemaakt van Container-Managed Persistence (CMP), worden de implementatiedetails van de relationele databank automatisch door de bean container geregeld. Dit gebeurt automatisch en de relationele databank zelf wordt dus verder niet besproken.

Fig. 3.2 toont de verschillende onderdelen van de component. Er volgt een beschrijving van de onderdelen.

3.2.1 Entities

Alle informatie die in de databank wordt bijgehouden, wordt voorgesteld door één of meerdere entities. Er volgt een beschrijving van alle entities en de informatie die ze voorstellen.

3.2.1.1 Accounts

Accounts laten toe om gebruikers van het systeem te identificeren en na te gaan of het om een gewone gebruiker of een beheerder (administrator) gaat. Deze informatie wordt voorgesteld door de Account entity. Er wordt een gebruikersnaam, een geëncrypteerde versie van zijn wachtwoord en een boolean, die aangeeft of het om een admin gaat, bijgehouden. De verschillende attributen (en hun type) van de Account entity worden in Fig. 3.3 weergegeven.

3.2.1.2 Server Plugins

De informatie over de serverside plugins wordt door twee entities voorgesteld. In de Server-Plugin entity wordt algemene informatie over de plugin bijgehouden. Dit is de naam, het

commando dat moet worden uitgevoerd om hem op te starten en een korte beschrijving. Er wordt ook een verwijzing naar een `ServerPluginData` entity bijgehouden. Bij het uploaden wordt een plugin omgezet naar een byte array. Zo wordt hij ook in de databank voorgesteld. De `ServerPluginData` entity stelt een segment van die byte array voor. In de huidige implementatie is elk segment 64000 bytes groot. De byte array wordt opgesplitst in delen van 64000 bytes (met het laatste segment mogelijk kleiner). Elk segment (behalve het laatste) verwijst ook naar het volgende segment van de plugin. Als de plugin wordt opgevraagd uit de databank worden eerst alle segmenten weer samengevoegd tot één array.

De datasegmenten zelf worden voorgesteld door Binary Large Objects (BLOB). De plugin wordt opgesplitst in verschillende segmenten om de beperkingen, die aan BLOBs worden opgelegd door veel databankimplementaties, te omzeilen:

- **Maximale Grootte:** De grootte van BLOBs wordt beperkt tot een maximum. Plugins die deze maximale grootte overschrijden zouden in dit geval niet gebruikt kunnen worden.
- **Vaste Grootte:** Deze beperking is een uitbreiding op de maximale grootte. Voor elke BLOB wordt in dit geval een veld van vaste grootte gereserveerd. Plugins die groter zijn kunnen dan, net zoals bij maximale grootte, niet worden gebruikt. Een bijkomend probleem is dat als een plugin kleiner is dan de gereserveerde ruimte er geheugen verloren gaat.

De door ons gebruikte databankimplementatie gebruikt BLOBs met een vaste grootte van 64000 bytes. Om de beperkingen die hierdoor veroorzaakt worden te omzeilen, wordt de plugin dus opgesplitst in segmenten van diezelfde grootte. Afhankelijk van de manier waarop BLOBs zijn geïmplementeerd kan de grootte van de segmenten worden aangepast.

De attributen van de entities die gebruikt worden om een server plugin voor te stellen worden weergegeven in Fig. 3.4.

3.2.1.3 Netwerk Topologie

Het netwerk wordt voorgesteld door een aantal entities. Door de `IPNode` en `IPLink` entities wordt informatie over de IP laag bijgehouden. Een IP node heeft als attributen een unieke naam en een IP adres. Bij een IP link hoort een unieke naam, een begin- en eindnode, een

ServerPlugin:			
String: name	String: cmd	String: description	ServerPluginData: data
ServerPluginData:			
byte[]: dataPiece	ServerPluginData: nextPiece		

Figuur 3.4: De attributen van de `ServerPlugin` en `ServerPluginData` entities

delay en een kost. De `OverlayNode` en `OverlayLink` entities stellen informatie over de overlay servers en de edges ertussen voor. Aangezien het overlay netwerk een virtuele laag bovenop het IP netwerk is, komt elke overlay server overeen met een onderliggende IP node en elke overlay edge met een pad van IP edges. Een pad op de IP laag wordt voorgesteld door de `IPPath` entity. Dit is een gelinkte lijst die uit een aantal `IPPathNode` entities bestaat. Voor elke padnode wordt een bijhorende IP node en een verwijzing naar de volgende padnode bijgehouden.

Zoals gezegd in sectie 2.1, is het mogelijk dat er niet genoeg informatie over het IP netwerk beschikbaar is om de hele IP topologie te reconstrueren. De `IPNode` en `IPPath` entities kunnen ook worden gebruikt om informatie over een abstractie van het IP netwerk, in plaats van de volledige topologie, bij te houden.

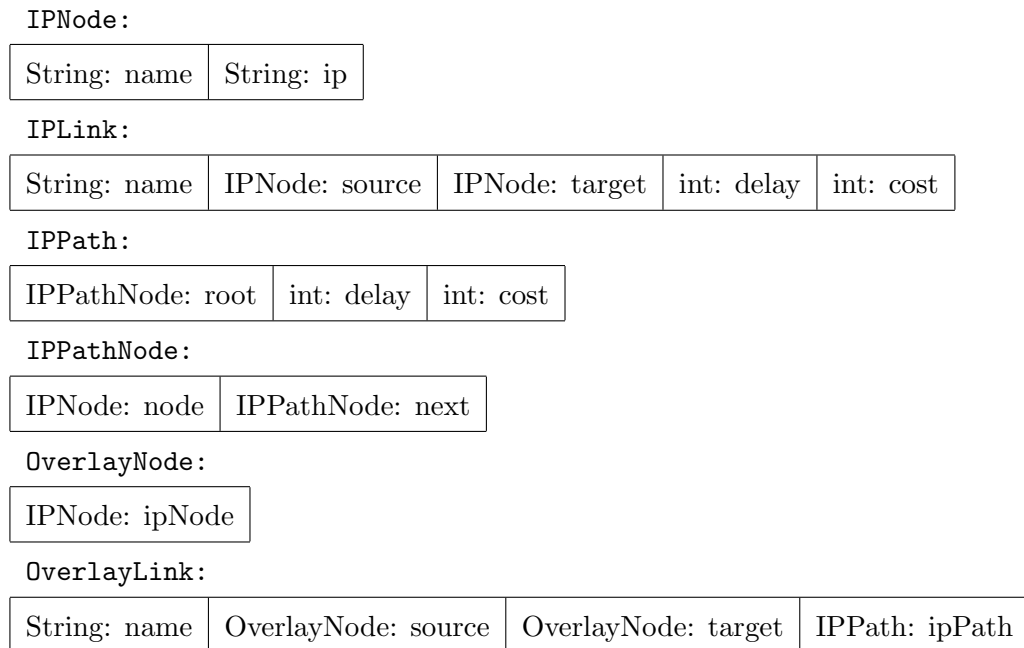
De attributen van de verschillende entities die het netwerk voorstellen worden getoond in Fig. 3.5.

3.2.1.4 Multicastsessies

Algemene informatie over multicastsessies wordt bijgehouden in de `MulticastSession` entity. De attributen die worden opgeslagen, zijn: de serverside plugin (in de vorm van een `ServerPlugin` entity), een beschrijving, een begin- en eindtijdstip, het berekende RP (in de vorm van een `OverlayNode` entity), een set van multicastbomen, een set van verzenders en een set van ontvangers.

De verzenders en ontvangers worden voorgesteld door de `MulticastSource` en `MulticastTarget` entities. Voor beiden wordt de account van de gebruiker, een IP adres, een poort en de dichtstbijzijnde overlay server bijgehouden.

De multicastbomen bestaan steeds uit een pad van een verzendende overlay server naar het RP en een boom van het RP naar alle ontvangende overlay servers (de RP-boom). De paden



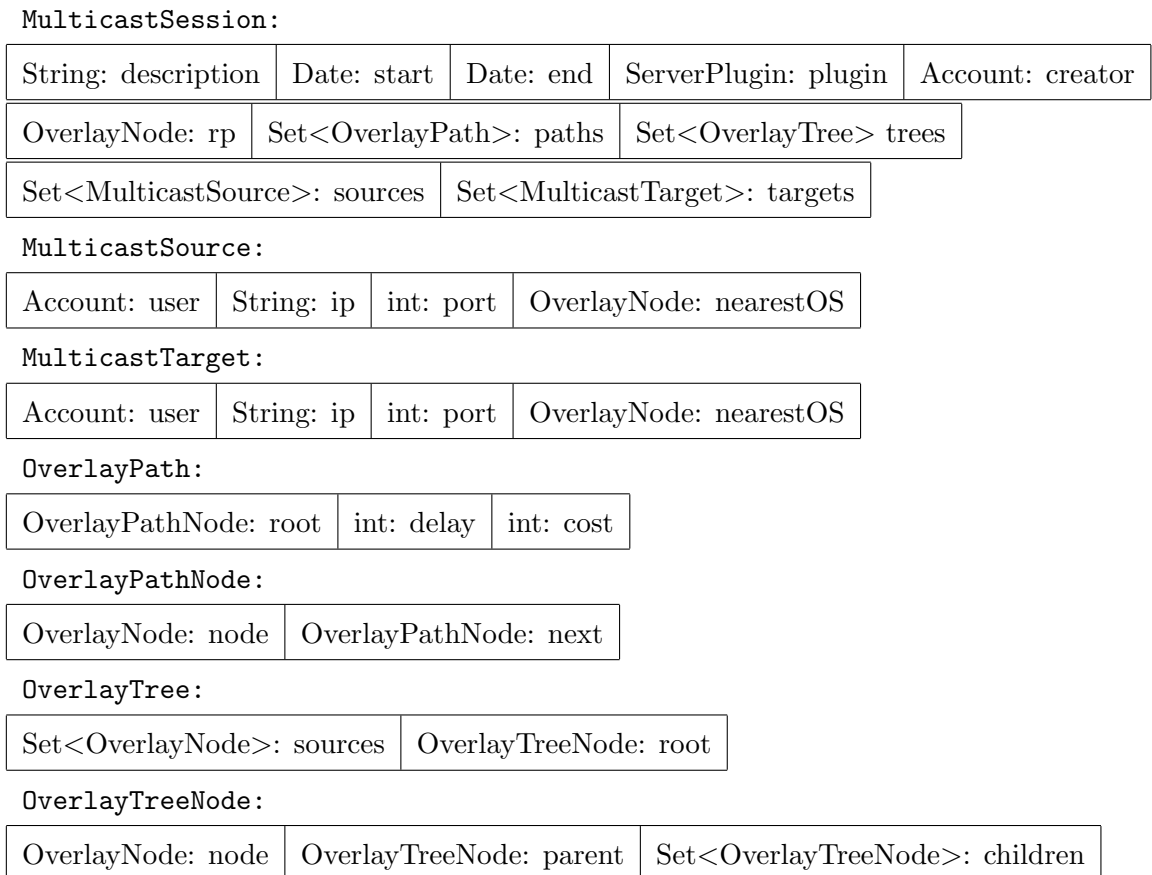
Figuur 3.5: De attributen van de IPNode, IPLink, IPPath, IPPathNode, OverlayNode en OverlayLink entities

worden voorgesteld door de `OverlayPath` entity. Deze bestaat, net zoals de `IPPath` entity, uit een gelinkte lijst van padnodes (in dit geval `OverlayPathNode` entities). De RP-bomen worden voorgesteld door de `OverlayTree` entity. Deze bestaat uit een `OverlayTreeNode` entity die de wortel van de boom voorstelt en een set van verzendende overlay servers (aangezien meerdere verzenders dezelfde RP-boom kunnen delen). Voor een boomnode wordt een bijhorende overlay server en, in tegenstelling tot bij padnodes, niet één, maar een set van opvolgers (kinderen) bijgehouden.

De attributen van alle entities die gebruikt worden om informatie over een multicastsessie voor te stellen, worden getoond in Fig. 3.6.

3.2.2 Entity Facades

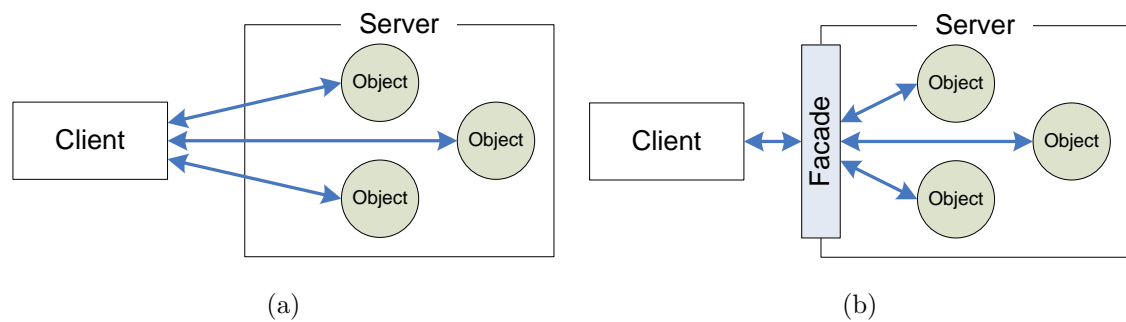
Facades laten toe om de functionaliteit van een aantal objecten te bundelen en zo de interactie ermee te vereenvoudigen. De werking van het façade ontwerppatroon [7] wordt geïllustreerd in Fig. 3.7. De aanwezigheid van een facade zorgt ervoor dat de details van de achterliggende objecten verborgen blijven voor de clients en dat ze voor operaties waar meerdere serverside objecten bij betrokken zijn, toch maar één interactie met de server moeten uitvoeren.



Figuur 3.6: De attributen van de entities die informatie in verband met multicastsessies voorstellen

In dit geval zijn de objecten entities, de facade een stateless session bean en de client een andere session bean in dezelfde bean container. Door facades toe te voegen moeten de andere beans niet rechtstreeks de entities aanspreken. Door het groot aantal entities, wordt door de facades het gebruik ervan sterk vereenvoudigd.

Er is een facade voorzien voor de accounts, de plugins, het netwerk en de multicastsessies. Elke facade voorziet methoden om entities aan te maken, te verwijderen en te zoeken (op basis van verschillende criteria). Elke facade heeft een local view, dit is een interface die definities bevat van methoden die kunnen worden opgeroepen door EJBs in dezelfde bean container. Deze facades kunnen dus niet rechtstreeks worden aangesproken door andere componenten van de architectuur (zoals de Web Interface of het OMP).



Figuur 3.7: Het façade ontwerppatroon (de conventies uit Tabel 3.1 gelden niet voor deze figuur) (gebaseerd op [7]) (a) Client-server interactie zonder facade (b) Met facade

3.2.3 Nearest Overlay Server Finder

Elke client is verbonden met een overlay server die hem toegang verleent tot het overlay netwerk. Verzenders sturen datapakketten door naar deze server, die ze dan verder in het overlay netwerk zal routeren. Elke ontvanger ontvangt datapakketten van deze overlay server. De Nearest Overlay Server Finder component selecteert voor elke client een geschikte toegangsserver. Het kiezen van deze server is een complex onderwerp en valt buiten de doelstellingen van deze thesis. De huidige implementatie geeft dan ook bij elke aanvraag gewoon een willekeurige overlay server terug.

In een implementatie geschikt voor praktisch gebruik kan bijvoorbeeld de overlay server gekozen worden die zich fysiek het dichtst bij de client bevindt of degene waarbij de delay tot de client het laagst is.

De component bestaat uit een interface (die een `getNearestOverlayServer()` methode definieert) en een implementatie. Hierdoor kan de implementatie makkelijk vervangen worden door een andere versie, zonder dat de andere EJBs die deze component gebruiken moeten worden aangepast.

3.2.4 Encryption Unit

Om de privacy van de gebruikers te verbeteren worden wachtwoorden geëncrypteerd vooraleer ze in de databank worden opgeslagen. Als een gebruiker dan zijn wachtwoord ingeeft, wordt de geëncrypteerde versie ervan vergeleken met het geëncrypteerde wachtwoord in de databank. De component heeft een `encrypt()` methode die een String encrypteert.

Net zoals bij de Nearest Overlay Server Finder is er een interface- en een implementatiegedeelte. Hierdoor kan het type van de encryptie makkelijk gewijzigd worden, zonder dat hierdoor de implementatie van de andere onderdelen van het DMP moet worden aangepast. De huidige implementatie berekent een MD5 hash [20].

3.2.5 Client Facade

Bij de communicatie tussen het DMP en de andere componenten van de architectuur (Web Interface en OMP) wordt net zoals bij de entities (zie sectie 3.2.2) gebruik gemaakt van het façade ontwerppatroon. In dit geval zijn de clients de andere componenten van de architectuur en niet EJBs in dezelfde bean container. Daarom wordt er hier een remote in plaats van een local view voorzien.

De client facade staat in voor de communicatie met de Web Interface. De facade voorziet methoden voor:

- Het toevoegen van nieuwe accounts, wijzingen van wachtwoorden en gewone accounts promoveren tot beheerders.
- Het opvragen van informatie over de netwerktopologie.
- Het toevoegen van nieuwe server plugins en het bekijken van informatie over reeds bestaande plugins.
- Het toevoegen van nieuwe multicastsessies en het opvragen van informatie over bestaande sessies. Er is ook een methode voorzien die gebruikers toelaat om zich voor sessies te registreren als verzender en/of ontvanger.

Bij het oproepen van een methode moet steeds een gebruikersnaam en wachtwoord worden meegegeven. Aan de hand hiervan wordt gecontroleerd of de gebruiker gemachtigd is om de methode op te roepen. Sommige methoden mogen namelijk enkel door beheerders worden uitgevoerd.

3.2.6 Management Facade

De management facade is gelijkaardig aan de client facade, maar wordt gebruikt door het OMP om met het DMP te communiceren. De facade voorziet methoden voor:

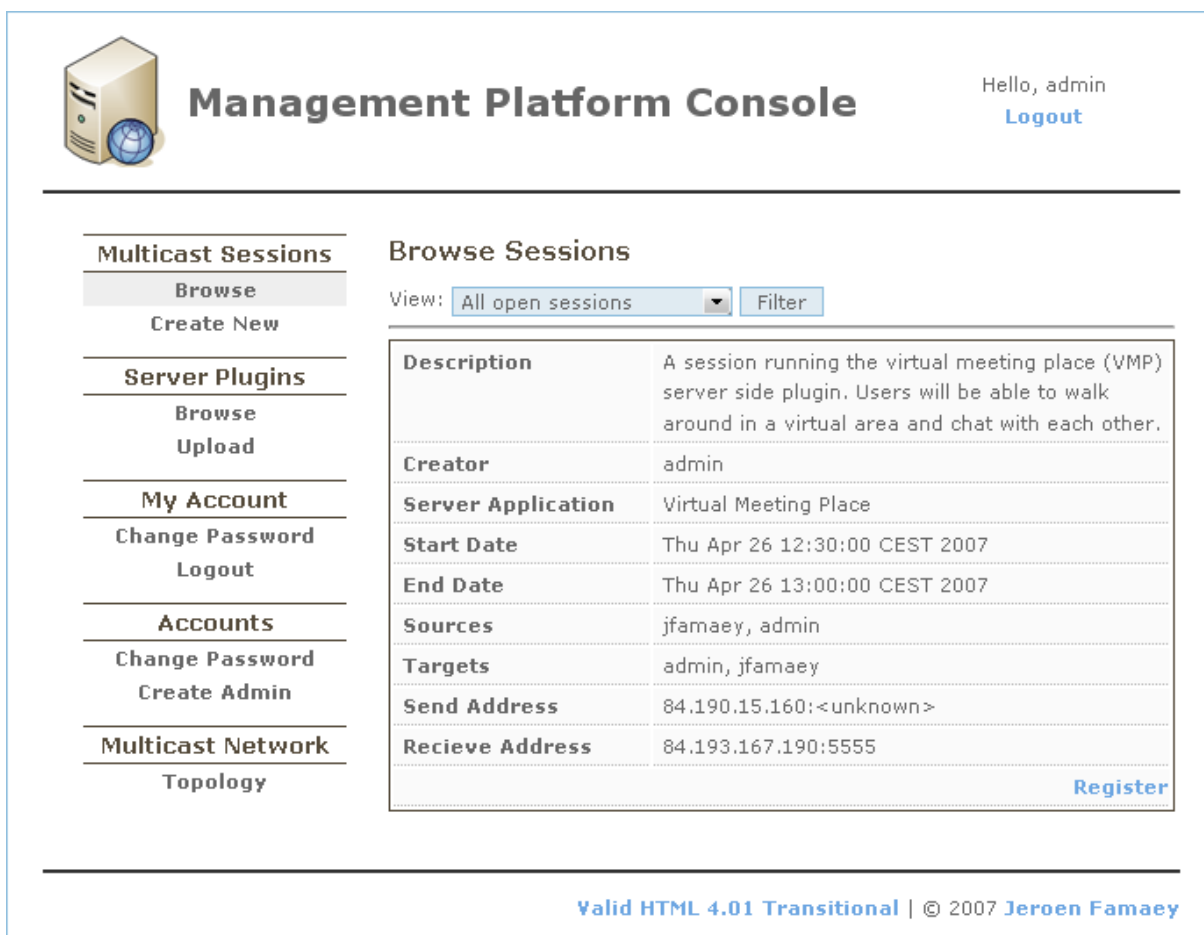
- Het opvragen en instellen van de netwerktopologie. Het is mogelijk om overlay servers en links ertussen toe te voegen en te verwijderen.
- Het opvragen van informatie over multicastsessies. Er is ook een methode voorzien om het RP en de multicastbomen toe te voegen aan de databank nadat ze zijn berekend.
- Het opvragen van een plugin, in de vorm van een byte array.

De methoden van deze facade mogen enkel door beheerders worden uitgevoerd. Bij het opstarten van het OMP moet een beheerder zijn gebruikersnaam en wachtwoord opgeven, dit wordt dan steeds meegestuurd bij het oproepen van een methode van de Management Facade.

3.3 Web Interface

De Web Interface laat gebruikers toe om met behulp van een browser een aantal taken uit te voeren. Er wordt een onderscheid gemaakt tussen gewone gebruikers en beheerders. Gewone gebruikers en beheerders kunnen de volgende taken uitvoeren:

- Hun eigen wachtwoord wijzigen.
- Nieuwe multicastsessies aanmaken.
- Informatie over multicastsessies opvragen. Er kan een lijst van sessies worden opgevraagd op basis van een aantal criteria:
 - Alle sessies die nog niet begonnen zijn.
 - Alle sessies die de ingelogde gebruiker aangemaakt heeft en die nog niet geëindigd zijn.
 - Alle sessies waarvoor de ingelogde gebruiker is geregistreerd als verzender en/of ontvanger en die nog niet geëindigd zijn.
- Zich registreren voor een multicastsessie als verzender en/of ontvanger.
- Informatie over de reeds geuploadede server plugins bekijken.
- Nieuwe server plugins uploaden. Alle bestanden van de plugin moeten samengevoegd worden in één ZIP bestand. Verder moet de gebruiker ook de naam, een beschrijving en



Management Platform Console Hello, admin [Logout](#)

Multicast Sessions

[Browse](#)
[Create New](#)

Server Plugins

[Browse](#)
[Upload](#)

My Account

[Change Password](#)
[Logout](#)

Accounts

[Change Password](#)
[Create Admin](#)

Multicast Network

[Topology](#)

Browse Sessions

View:

Description	A session running the virtual meeting place (VMP) server side plugin. Users will be able to walk around in a virtual area and chat with each other.
Creator	admin
Server Application	Virtual Meeting Place
Start Date	Thu Apr 26 12:30:00 CEST 2007
End Date	Thu Apr 26 13:00:00 CEST 2007
Sources	jfamaey, admin
Targets	admin, jfamaey
Send Address	84.190.15.160:<unknown>
Recieve Address	84.193.167.190:5555

[Register](#)

Valid HTML 4.01 Transitional | © 2007 Jeroen Famaey

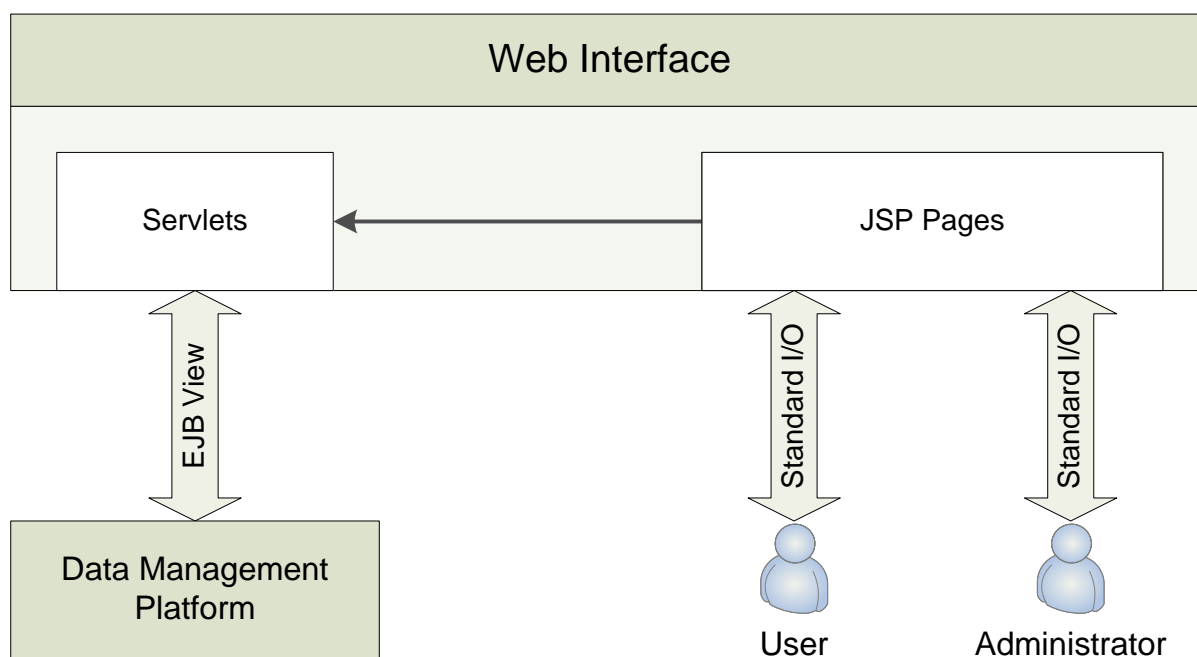
Figuur 3.8: Een screenshot van een webpagina gegenereerd door de web interface

het commando dat moet worden uitgevoerd om de plugin op te starten meegeven. Bij een Java plugin die zich in een jar file “foo.jar” bevindt is het commando dat moet worden uitgevoerd bijvoorbeeld “java -jar foo.jar”.

Beheerders kunnen daarbovenop nog een aantal extra taken uitvoeren:

- Het wachtwoord van andere gebruikers wijzigen.
- Een gebruiker beheerdersrechten geven.
- Informatie over de huidige netwerktopologie bekijken. Er kan zowel informatie worden opgevraagd over de IP laag als over het overlay netwerk.

Fig. 3.8 toont een screenshot van de webpagina gegenereerd door de web interface. De ingelogde gebruiker is een beheerder met de naam “admin”, die alle nog niet begonnen multicastsessies



Figuur 3.9: Schematische voorstelling van de web interface

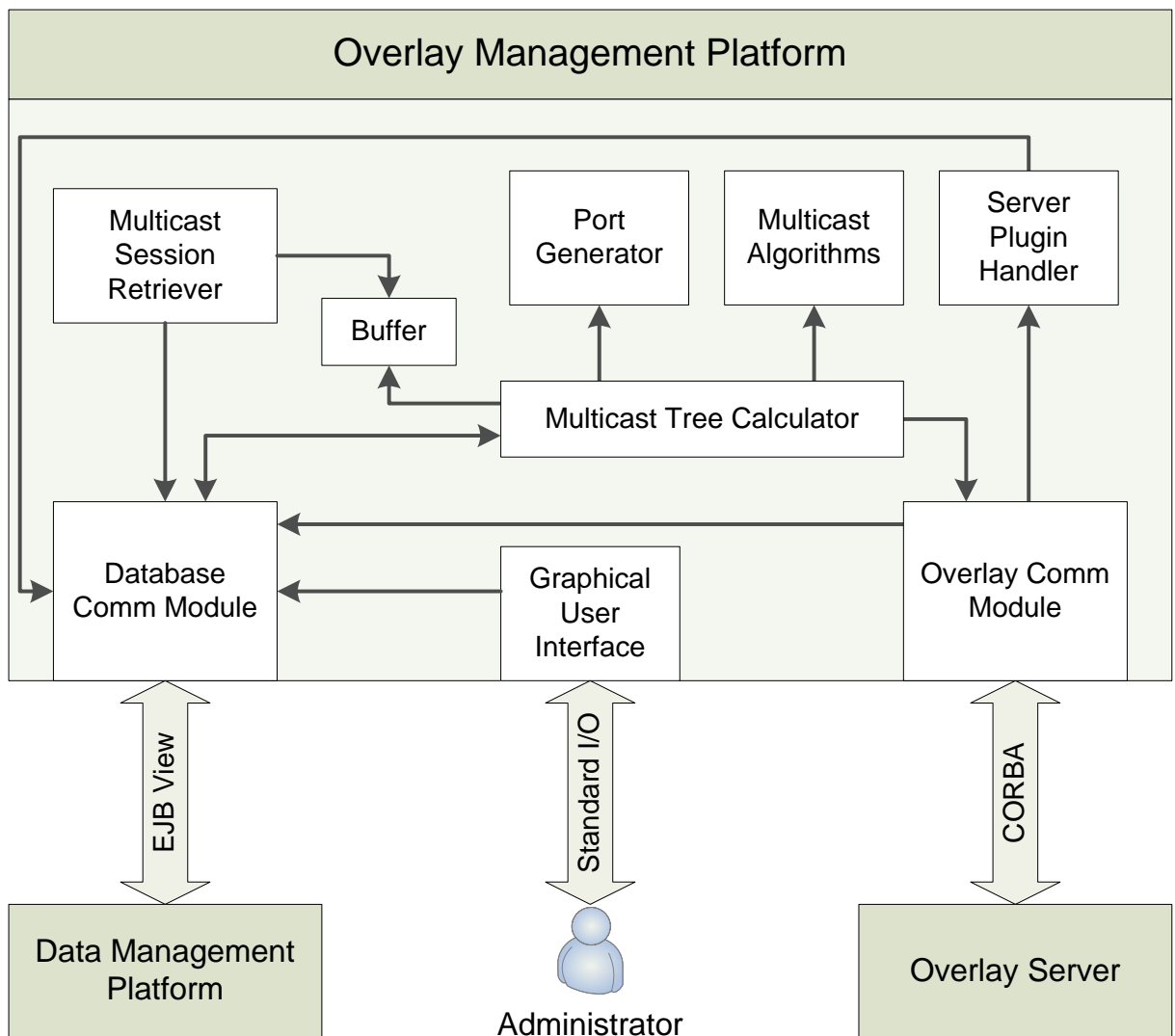
aan het bekijken is (één in dit geval). De poort naar waar de gebruiker boodschappen moet sturen was tijdens het nemen van de screenshot onbekend (unknown). Dit is omdat deze pas wordt bepaald wanneer de sessie gaat beginnen. Op dat moment kunnen gebruikers via de webinterface hun doelpoort te weten komen.

Fig. 3.9 geeft een schematisch voorstelling van de onderdelen van de web interface. De JSPs zorgen voor de grafische weergave van de webpagina's. De Servlets staan in voor het verwerken van HTTP requests en de communicatie met het DMP. De Servlets zorgen ook voor het juist weergeven van de informatie die opgehaald wordt uit de databank.

3.4 Overlay Management Platform

Het OMP is een stand-alone Java applicatie, die een belangrijk deel van de functionaliteit van de architectuur implementeert. Het voert de volgende taken uit:

- Het RP en de multicastbomen berekenen van multicastsessies die gaan beginnen.
- De routingstabellen van de overlay servers configureren in functie van de berekende bomen.



Figuur 3.10: Schematische voorstelling van het Overlay Management Platform

- De netwerktopologie aanpassen als nieuwe overlay servers online komen of wanneer ze offline gaan.
- Server plugins ophalen uit de databank en doorsturen naar de overlay servers.

Deze functionaliteit wordt gerealiseerd door een aantal onderdelen. Deze worden schematisch weergegeven in Fig. 3.10. In de volgende secties gaan we dieper in op de taak van de specifieke onderdelen en hun relatie tot elkaar.

3.4.1 Database Communication Module

De Database Communication Module (DCM) staat, zoals de naam doet vermoeden, in voor de communicatie met het DMP. Dit gebeurt via een remote view van een facade, geïmplementeerd als een stateless session bean.

Het wijzigen van de netwerktopologie mag enkel gebeuren via de DCM. Er zijn dan ook methoden voorzien om het IP netwerk in te stellen en om overlay servers en edges aan het netwerk toe te voegen of eruit te verwijderen. De DCM zorgt ervoor dat alle wijzigingen aan de netwerktopologie worden doorgegeven aan het DMP. Er wordt ook een lokale versie van de netwerktopologie bijgehouden. De volledige topologie uit de databank lezen is namelijk een zeer dure operatie. Op deze manier moet ze enkel volledig aan het DMP worden opgevraagd bij het opstarten van het OMP. Aangezien wijzigingen aan het netwerk enkel via deze component gebeuren, kan ervoor gezorgd worden dat beide exemplaren van de topologie steeds consistent zijn.

De DCM is ook in staat om informatie over multicastsessies op te halen uit de databank. Het RP en de multicastbomen kunnen nadat ze berekend zijn ook worden teruggestuurd.

Als er een server plugin nodig is voor een bepaalde multicastsessie kan deze via de DCM aan de databank worden opgevraagd. Deze wordt teruggegeven in de vorm van een byte array.

Het is soms nodig dat andere componenten op de hoogste worden gebracht van wijzigingen aan de netwerktopologie. Hiervoor wordt gebruik gemaakt van het Observer ontwerppatroon [10]. Andere componenten kunnen zich registreren als luisteraars. Telkens de topologie verandert, worden alle luisteraars hierover ingelicht.

3.4.2 Server Plugin Handler

Om te vermijden dat plugins, die in meerdere sessies worden gebruikt, steeds opnieuw uit de databank moeten worden opgehaald, worden de opgevraagde plugins naar de schijf geschreven. Als één van de overlay servers een plugin nodig heeft, wordt eerst gecontroleerd of deze zich al in het plugin cache bevindt. Als dit het geval is wordt hij van de schijf gelezen, in plaats van uit de databank. De Server Plugin Handler is verantwoordelijk voor het beheren van dit cache. De andere onderdelen van het OMP zullen dan ook niet rechtstreeks plugins opvragen aan de DCM, maar dit via de Server Plugin Handler doen (die dan indien nodig het verzoek verder stuurt naar de DCM).

3.4.3 Overlay Communication Module

De Overlay Communication Module (OCM) regelt de communicatie tussen het OMP en de overlay servers. Dit gebeurt met behulp van CORBA. De overlay servers zoeken via de CORBA naming service een remote object referentie naar de OCM. Ze kunnen dan de methoden gedefinieerd in de remote interface oproepen. Aangezien de communicatie gebeurt in twee richtingen wordt gebruik gemaakt van het Callback ontwerp patroon [5]. Via de `registerOverlayServer()` methode van de OCM geven de overlay servers een remote referentie naar een zogenaamd callback object door aan de OCM. Via dit object kan de OCM dan methoden van de overlay servers oproepen.

Als de overlay servers zich registreren, wordt dit doorgegeven aan de DCM en worden ze automatisch aan de netwerktopologie toegevoegd. Als ze offline gaan kunnen ze zichzelf uitschrijven via de `unregisterOverlayServer()` methode. Met behulp van een ping mechanisme detecteert de OCM ook overlay servers die door een crash of een ander probleem onbeschikbaar worden. Overlay servers die offline gaan of onbeschikbaar zijn, worden uit de netwerktopologie verwijderd.

Het ping mechanisme werd geïmplementeerd als een methode die gewoon de boolean waarde `true` teruggeeft. Het OMP roept deze methode, die aan de overlay server zijde is geïmplementeerd, op via het callback object. Als de overlay server niet meer beschikbaar is, genereert de onderliggende CORBA component een uitzondering. De OCM vangt deze uitzondering op en weet dan dat de server offline is.

De OCM bevat ook een `addMulticastRule()` methode die door andere componenten van het OMP kan worden opgeroepen. Deze methode voegt routeringsregels toe aan de overlay servers, om de multicastbomen op overlay niveau in te stellen. Het toevoegen van deze regels aan de overlay servers gebeurt met behulp van het callback object.

Via het callback object kan de OCM ook aan een overlay server laten weten dat hij voor een multicastsessie een bepaalde plugin moet opstarten. Als de overlay server nog niet over een lokale versie van de plugin beschikt, vraagt hij deze aan de OCM. Die zal op zijn beurt de plugin verkrijgen van de Server Plugin Handler.

3.4.4 Graphical User Interface

De Graphical User Interface (GUI) laat beheerders toe om een aantal taken uit te voeren. De belangrijkste taak is het instellen van de IP topologie. Via de GUI kunnen IP nodes en edges worden toegevoegd aan de databank. Verder kan ook informatie over de volledige netwerktopologie worden opgevraagd (zowel over het IP als het overlay netwerk).

De GUI heeft ook een console die gebruikt wordt om foutboodschappen en andere informatie te tonen. Als er dan iets misloopt kan een beheerder het probleem identificeren aan de hand van de gegenereerde foutboodschappen.

3.4.5 Port Generator

Aangezien het mogelijk is dat de overlay servers boodschappen van verschillende multicastsessies tegelijkertijd moeten routeren, is het nodig dat ze deze van elkaar kunnen onderscheiden. Dit gebeurt aan de hand van het poortnummer waarop de boodschappen toekomen. Ze zullen dan ook diezelfde poort gebruiken om de boodschappen door te sturen naar andere overlay servers.

De Port Generator staat in voor het toekennen van een unieke poort aan elke multicastsessie. Deze component houdt ook het eindtijdstip van elke sessie bij, zodat de poort kan hergebruikt worden eens de sessie voorbij is. Bij het aanmaken wordt een begin- (S) en eindpoort (E) opgegeven. Alle gegenereerde poorten zullen zich in het interval $[S, E]$ bevinden. De enige beperkingen voor de goede werking van de generator is dat er op elke moment maximum $E - S + 1$ sessies aan de gang zijn en dat de overlay servers de poorten in dit interval niet voor andere doeleinden gebruiken.

De Click routers op de overlay servers zijn geconfigureerd om alle pakketten die toekomen op poorten in het interval $[10000, 20000]$ te onderscheppen. Pakketten die toekomen op een andere poort worden doorgestuurd naar het besturingssysteem. De gegenereerde poorten zullen zich dan ook in dit interval bevinden.

Voor elke multicastsessie, die gebruik maakt van een serverside plugin, wordt nog een tweede poort gegenereerd. De pakketten die toekomen op het RP van de sessie moeten namelijk eerst naar de plugin worden gestuurd voor ze worden doorgerouteerd. Deze tweede poort bevindt zich in het interval $[20001, 30000]$. De pakketten die naar het RP worden gestuurd, hebben deze tweede poort als doelpoort. De Click router laat deze pakketten door en de plugin zal op deze

poort luisteren. Deze kan de pakketten dan verwerken en vervolgens doorsturen naar de Click router, die ze dan verder door het overlay netwerk routeert.

3.4.6 Multicast Algorithms

Deze component bevat een aantal algoritmen. Het eerste algoritme zoekt een geschikt RP voor elke multicastsessie. Het tweede berekent voor elke verzendende overlay server van de sessie een multicast distributieboom (bestaande uit een pad naar het RP en een RP-boom). In hoofdstuk 4 worden een aantal mogelijke algoritmen voor deze problemen besproken. De huidige versie van de component bevat een implementatie van de daar besproken algoritmen.

3.4.7 Multicast Session Retriever

De Multicast Session Retriever staat in voor het ophalen van de multICASTsessies uit de databank. Hiervoor wordt een thread opgestart die elke minuut via de DCM alle sessies opvraagt aan het DMP die over maximum T minuten beginnen. T is een parameter die kan worden ingesteld afhankelijk van de tijdsduur nodig om de multicastbomen te berekenen. Het DMP zorgt er zelf voor dat sessies die reeds afgelopen zijn, of waarvan de bomen reeds berekend zijn, worden weggefilterd. De gevonden sessies worden toegevoegd aan een buffer. Sessies die zich reeds in de buffer bevinden worden genegeerd.

3.4.8 Multicast Tree Calculator

De Multicast Tree Calculator component haalt multICASTsessies uit de buffer en berekent het RP en de multicastbomen. Hiervoor wordt een threadpool gecreëerd, waarvan het maximaal aantal threads, die worden opgestart, kan worden ingesteld. Zolang de buffer niet leeg is, zullen de threads hieruit sessies afhalen. Met behulp van de multicast algoritmen worden het RP en de multicastbomen berekend. Deze informatie wordt doorgegeven aan de DCM, die ze zal doorsturen naar de databank.

Als de netwerktopologie verandert tijdens het uitvoeren van de algoritmen, moet dit gedetecteerd worden en moeten de algoritmen opnieuw worden uitgevoerd voor de huidige sessies. De Multicast Tree Calculator zal zich daarom registreren als luisteraar bij de DCM (zie sectie 3.4.1). Als de netwerktopologie wijzigt zal hij dan worden ingelicht.

De berekende multicastbomen worden omgezet in een aantal routeringsregels. Deze worden dan doorgegeven aan de OCM, die ervoor zorgt dat elke routeringsregel doorgestuurd wordt naar de juiste overlay server. Elke regel bestaat uit een bron, een set van doelen en een time to live (ttl). Elke bron en elk doel bestaat uit een IP adres en poortnummer. De doelen geven de adressen aan naar waar de pakketten moeten doorgestuurd worden die toekomen op de interface en poort aangegeven door de bron. De ttl geeft het aantal seconden aan tot het einde van de bijhorende multicastsessie en dus het aantal seconden dat de regel moet blijven bestaan.

Het algoritme voor het omzetten van de multicastbomen naar routeringsregels kan als volgt worden beschreven:

1. Via de Port Generator worden twee poorten voor de sessie genereerd. De eerste, $port_s$, bevindt zich in het interval [10000, 20000] en wordt gebruikt om pakketten naar de overlay servers te sturen (zowel van een verzendende client als van een andere overlay server). De tweede, $port_{RP}$, bevindt zich in het interval [20001, 30000] en wordt gebruikt om pakketten naar de server plugin te sturen. Het IP van een overlay server O wordt weergegeven als IP_O .
2. Kies een verzendende overlay server V . Bij V hoort een pad P (naar het RP) en een RP-boom B (van het RP naar alle ontvangers). Als V gelijk is aan het RP ga dan naar stap 5.
3. Voeg voor elke overlay server S , van het pad P (behalve voor de twee laatsten, het RP en zijn voorganger), een routeringsregel toe aan S . De bron bestaat uit IP_S en $port_s$. De doelset bevat enkel de opvolger van S op het pad P , met als poort $port_s$.
4. Voeg voor de voorganger S van het RP op het pad P (de voorlaatste node van het pad) een routeringsregel toe aan S . De bron bestaat uit IP_S en $port_s$, de doelset bevat één entry met IP_{RP} en $port_{RP}$.
5. Stel S gelijk aan het RP.
6. Voeg aan S een routeringsregel toe. Als S gelijk is aan het RP is het bron IP gelijk aan het dummy IP adres dat wordt gebruikt door de server plugin (bijvoorbeeld het gateway adres) (zie sectie 3.1), anders is het gelijk aan IP_S . De bronpoort is gelijk aan $port_s$. De set van doelen bestaat enerzijds uit de kinderen van S in de boom B (dit zijn overlay

servers) en anderzijds uit de ontvangende clients die S als dichtstbijzijnde overlay server hebben. De doelpoort van de overlay servers (kinderen van S in B) is gelijk aan $port_s$, de doelpoort van de ontvangende clients is gelijk aan de poort die ze opgaven toen ze zich registreerden als ontvangers van de sessie.

7. Herhaal stap 6 recursief door elk van de kinderen van S in B gelijk te stellen aan S.
8. Herhaal stap 2 voor elke verzendende overlay server.

Als de multicastsessie geen gebruik maakt van een serverside plugin, moeten er een aantal kleine aanpassingen worden gemaakt aan het bovenstaande algoritme. In plaats van een uniek poortnummer te genereren voor $port_{RP}$, wordt deze gelijk gesteld aan $port_s$. Daarbovenop moet in stap 6 het dummy IP adres worden vervangen door het IP adres van het RP.

Tenslotte geeft de Multicast Tree Calculator ook door aan de OCM welke server plugin er op het RP van de sessie moet worden opgestart. Hierbij wordt ook $port_{RP}$ meegegeven. Deze stap wordt natuurlijk weggelaten als de sessie geen plugin gebruikt.

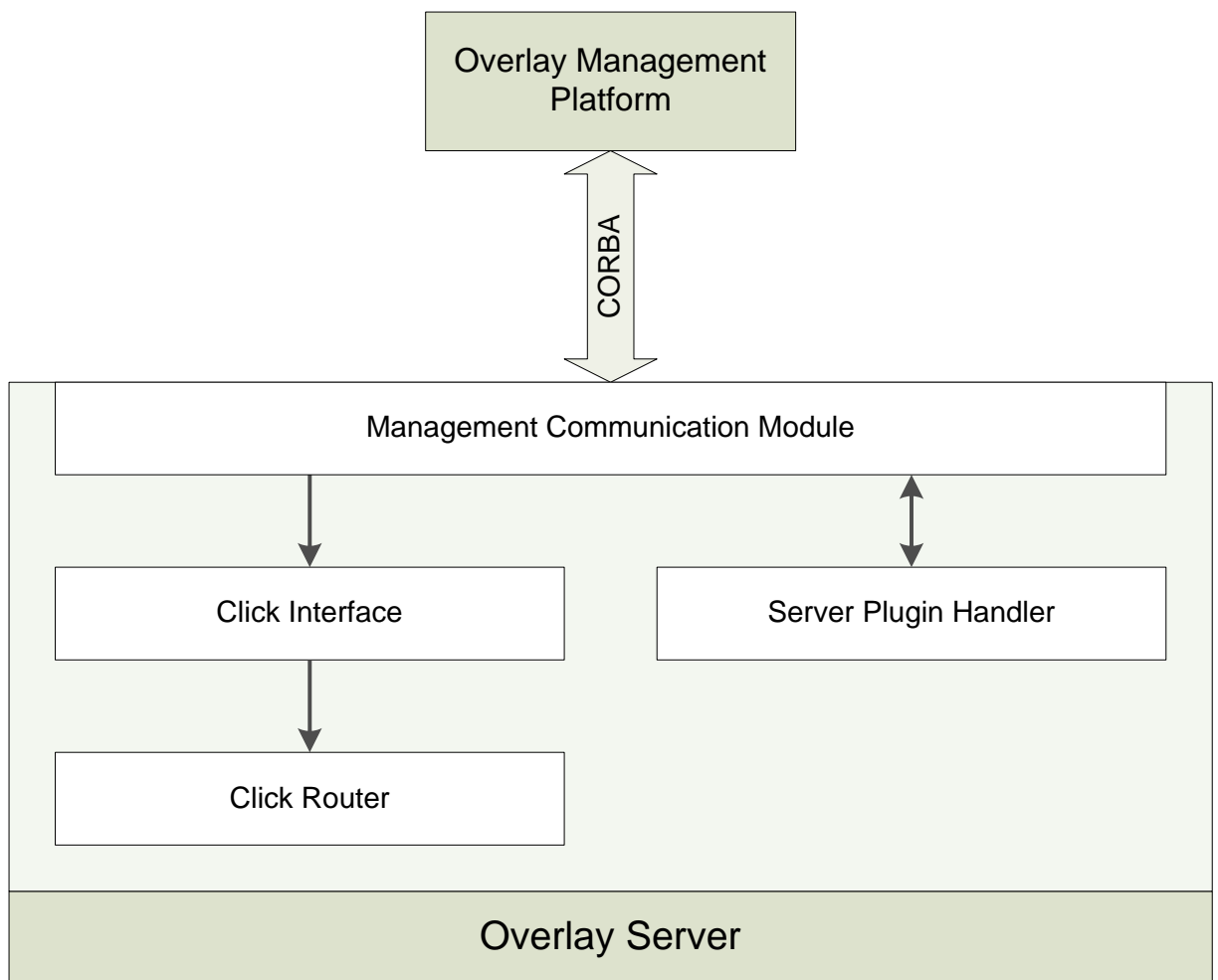
3.5 Overlay Servers

Elke overlay server bestaat uit een Click router met daarbovenop een Java front-end. De front-end staat in voor de communicatie met het OMP en het instellen en verwijderen van routingsregels in de router. De communicatie met het OMP gebeurt met behulp van CORBA en is geïmplementeerd in de Management Communication Module (MCM). De Click Interface is een Java component die kan gebruikt worden om de Click router aan te spreken. De Server Plugin Handler staat in voor het downloaden, opstarten en afsluiten van plugins.

Fig 3.11 geeft een overzicht van de onderdelen van de overlay server en hun relatie tot elkaar. In tegenstelling tot de MCM, Click Interface en Server Plugin Handler bevindt de Click router zich niet op de applicatielaag, maar onder het besturingssysteem. Om hier de nadruk op te leggen, werd het schema omgekeerd getekend.

3.5.1 Management Communication Module

De MCM verzorgt de communicatie met het OMP en het toevoegen en verwijderen van routingsregels in de Click router. De communicatie met het OMP gebeurt met behulp van CORBA,



Figuur 3.11: Schematische voorstelling van een Overlay Server

door gebruik te maken van een callback object. Als de overlay server opstart registreert hij zich bij het OMP en geeft daarbij een callback object door. Via dit callback object kan het OMP dan multicastregels toevoegen aan de MCM die ze op zijn beurt doorgeeft aan de Click interface. Het OMP kan via het callback object ook vragen om een bepaalde server plugin op te starten. Dit verzoek wordt dan doorgegeven aan de Server Plugin Handler.

De Click router is niet in staat om zelf regels te verwijderen op basis van een ttl. Daarom zal de MCM zelf regels verwijderen waarvan de ttl verstreken is. Routeringsregels die worden toegevoegd worden daarom ook door de MCM zelf in een lijst bewaard. Bij het opstarten van de overlay server wordt een thread gestart die telkens opnieuw de lijst overloopt. Als er een regel wordt gevonden waarvan de ttl verstreken is, wordt deze verwijderd uit de lijst en wordt ook aan de Click interface doorgegeven om hem uit de router te verwijderen.

3.5.2 Server Plugin Handler

Net zoals het OMP houdt elke overlay server een cache van server plugins bij, die op schijf worden bewaard. Als de Server Plugin Handler een verzoek krijgt om een bepaalde plugin op te starten, zal deze eerst controleren of er een lokaal exemplaar van de plugin aanwezig is. Als dit niet het geval is wordt via de MCM de plugin opgevraagd aan het OMP. Aangezien de plugins worden geupload in de vorm van een ZIP bestand, worden ze eerst uitgepakt en daarna naar de schijf geschreven.

Bij het opstarten van een plugin geeft de Server Plugin Handler, als command line argumenten, de poort waarnaar de plugin moet luisteren en het IP en de poort waarnaar hij moet verzenden mee. Als de sessie, waartoe de plugin behoort, afgelopen is, wordt hij automatisch afgesloten. Als de overlay server offline gaat, sluit de Server Plugin Handler automatisch alle plugins af.

3.5.3 Click Interface

De Click interface is een Java klasse die dienst doet als abstractie bovenop de Click router. De andere onderdelen van de Java front-end communiceren met de interface in plaats van rechtstreeks met de router. De interface voorziet methoden om de Click router op te starten en af te sluiten. Er is ook een methode voorzien om routeringsregels aan de router toe te voegen en eruit te verwijderen.

3.5.4 Click Router

De Click router staat in voor het correct routeren van de datapakketten die doorheen het overlay netwerk verstuurd worden. De routeringsregels in de router bestaan uit een bron IP, bron poort, doel IP en doel poort. Als een pakket toekomt op één van de poorten die voorbehouden zijn voor het overlay multicastverkeer, worden alle routeringsregels met de bron gelijk aan het doel van het toegekomen datapakket geselecteerd. Als er meer dan één regel wordt gevonden, wordt het pakket eerst gedupliceerd. Vervolgens wordt het naar alle doelen van de gevonden regels doorgestuurd.

3.5.5 Performantietests

Om de performantie van de Click router te meten, werden een aantal tests uitgevoerd. Hierbij werd gebruik gemaakt van een Smartbits machine, die grote hoeveelheden UDP pakketten kan genereren en versturen. De Smartbits machine is verbonden met een overlay server. Ze beschikken elk over twee Gigabit netwerkinterfaces. Eén voor het verzenden van datapakketten en één voor het ontvangen ervan. De verzendende interface van de Smartbits machine werd verbonden met de ontvangende interface van de overlay server en omgekeerd. Als overlay server werd gebruik gemaakt van een AMD Athlon XP 2100+ machine (1733.517 Mhz) met 512 MB RAM en een 2.6.16.13 Linux kernel.

Aan de Click component van de overlay server werd bij elke test tenminste één routeringsregel toegevoegd. Deze zorgt ervoor dat pakketten die toekomen op de ontvangende netwerkinterface van de overlay server verder werden doorgestuurd naar de ontvangende interface van de Smartbits machine. Verder wordt naar deze regel verwezen als de “smartbits regel”.

Het doel van de tests is de invloed van een aantal parameters op de gemiddelde delay en het pakketverlies te meten. De delay geeft de tijd tussen het versturen van een datapakket en het terug ontvangen van een antwoord erop en wordt steeds uitgedrukt in microseconden (μs). Het pakketverlies geeft het percentage van de verstuurd pakketten aan dat niet terugkeert. Zowel de delay als het pakketverlies kan worden gemeten met de Smartbits machine. De parameters waarvan de invloed wordt bekeken zijn:

- **Load:** Dit is de hoeveelheid data die door de Smartbits machine naar de overlay server wordt gestuurd en wordt uitgedrukt in Megabit per seconde (Mbps). Merk op dat de load de absolute hoeveelheid verstuurd data voorstelt en niet het aantal verstuurd pakketten. Als de pakketten groter zijn, zullen er bij een constante load dus minder worden verstuurd.
- **Aantal duplicaties:** Als de overlay server meerdere routeringsregels bevat die als bron het doel hebben van het ontvangen pakket, wordt het pakket eerst een aantal keer gedupliceerd en dan verstuurd naar het doel van elke gevonden regel.
- **Pakketgrootte:** De grootte van de verstuurd datapakketten wordt aangegeven in bytes.
- **Aantal regels:** Bovenop de routeringsregels die zorgen voor pakketduplicatie, kan de overlay server nog regels bevatten met een andere bron. Doordat de overlay server bij het

ontvangen van een pakket steeds de regellijst moet doorzoeken naar de geschikte regels, zou het kunnen dat de delay negatief wordt beïnvloed door de lengte van de regellijst.

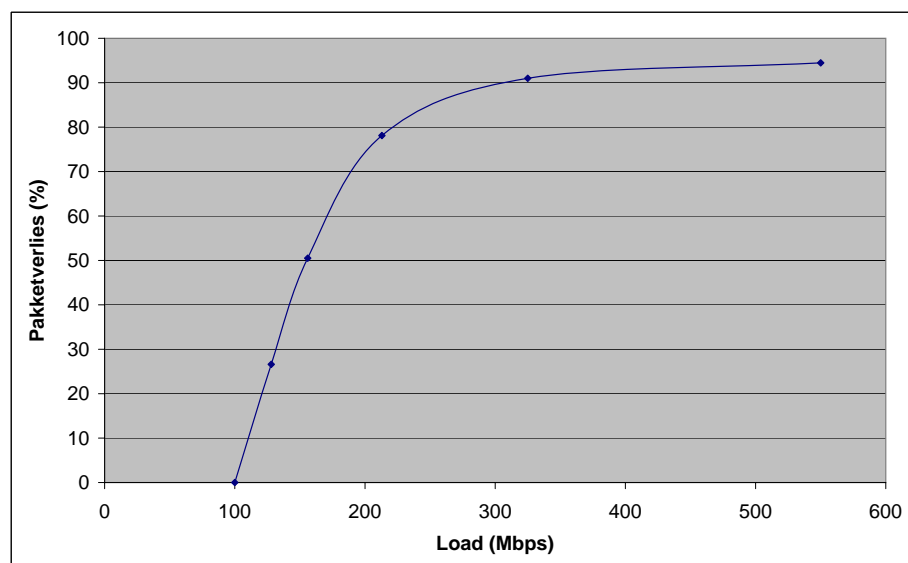
- **Positie in de regellijst:** Aangezien de regellijst moet worden doorzocht, kan de positie van de juiste regel in deze lijst ook invloed hebben op de delay. Dit wordt gesimuleerd door de smartbits regel vooraan of achteraan de lijst te plaatsen.

In de eerste test wordt de invloed van de load bekeken. Vervolgens wordt er gekeken naar het aantal duplicaties en de positie in de regellijst. In de derde test wordt nagegaan of de pakketgrootte invloed heeft op de delay en het pakketverlies. In de laatste test worden de grootte van de regellijst en opnieuw de positie erin beschouwd.

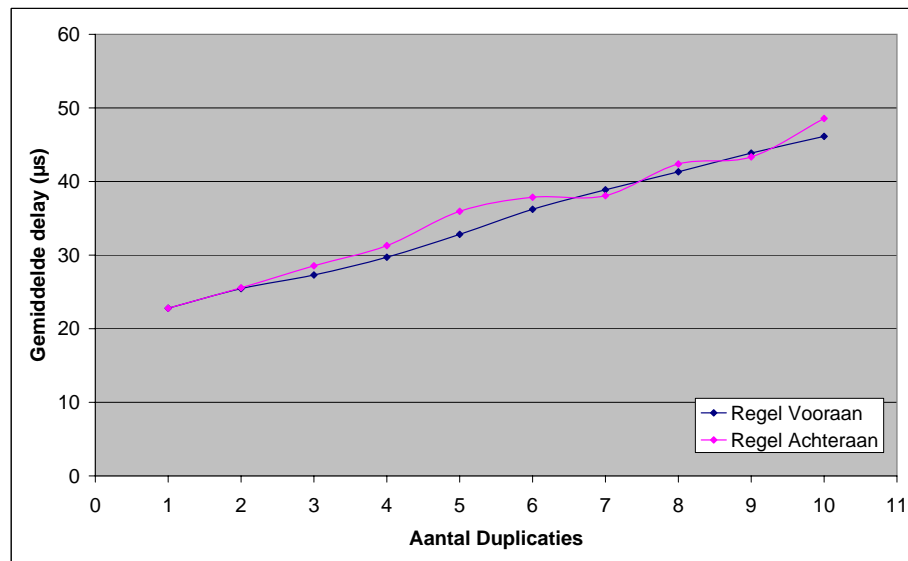
Om te voorkomen dat bij het testen van één parameter de andere parameters de resultaten zouden beïnvloeden, werden voor de niet-variabele parameters steeds dezelfde waarden gebruikt. De load werd ingesteld op 10 Mbps, pakketten waren steeds 128 bytes groot en de regellijst bevatte enkel de smartbits regel.

3.5.5.1 Test 1: Load

Het doel van deze test is nagaan vanaf welke load er pakketverlies begint op te treden. Hiervoor werd het aantal verloren gegane pakketten (%) gemeten bij verschillende loads (Mbps). Fig.



Figuur 3.12: Het pakketverlies (%) in functie van de totale load (Mbps)



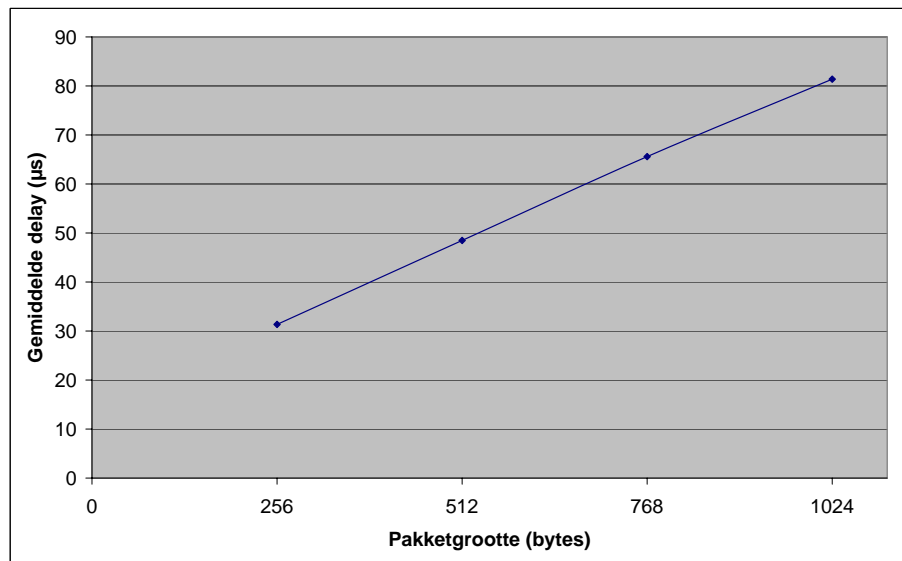
Figuur 3.13: De gemiddelde delay (μs) in functie van het aantal pakketduplicaties

3.12 toont het pakketverlies in functie van de load. In de figuur is te zien dat tot ongeveer 100 Mbps er geen pakketten verloren gaan. Het pakketverlies stijgt dan snel tot ongeveer 220 Mbps, waar reeds meer dan 80% van de verzonden pakketten verloren gaat.

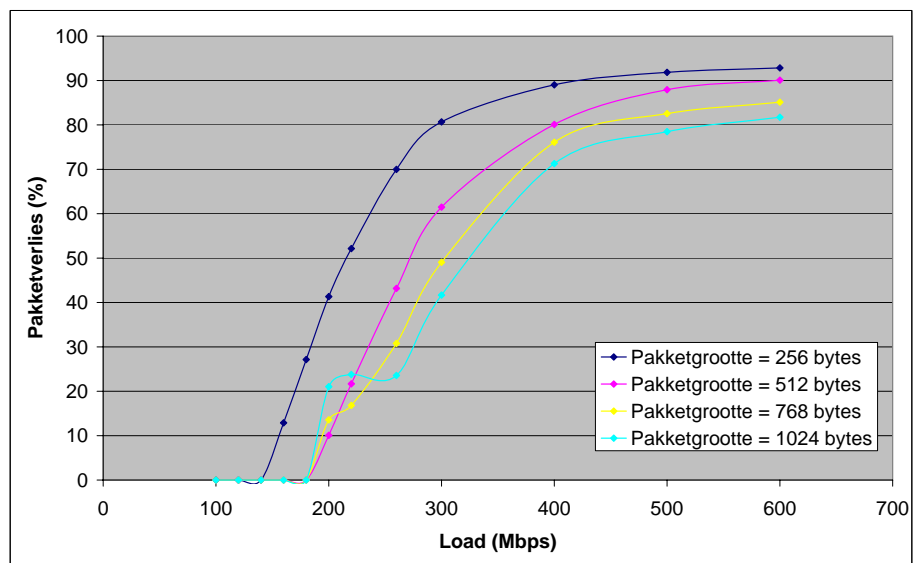
3.5.5.2 Test 2: Duplicatie

Bij de tweede test wordt gekeken welke invloed het aantal pakketduplicaties uitoefent op de delay. De duplicaties werden gesimuleerd door extra routeringsregels aan de overlay server toe te voegen. Deze kregen dezelfde bron als de smartbits regel, maar een niet bestaand doeladres. Hierdoor werden de pakketten wel gedupliceerd door de overlay server, maar werd uiteindelijk maar één van de kopieën teruggestuurd naar de smartbits machine. Om de invloed van de positie van een regel in de regellijst na te gaan werd de test twee keer uitgevoerd. De eerste keer werd de smartbits regel vooraan de lijst geplaatst, de tweede keer achteraan.

Fig. 3.13 toont de testresultaten. De gemiddelde delay (μs) wordt uitgezet tegenover het aantal keer dat het pakket werd gedupliceerd. Zoals verwacht stijgt de delay bij een hoger aantal duplicaties. Aangezien beide curves ongeveer gelijk lopen, heeft de positie van de regel in de regellijst waarschijnlijk weinig of geen invloed op de delay. Verder blijkt uit de resultaten dat zelfs bij 10 pakketduplicaties de gemiddelde delay nog steeds zeer laag is, namelijk in de grootorde van tientallen microseconden.



Figuur 3.14: De gemiddelde delay (μs) in functie van de pakketgrootte (bytes)



Figuur 3.15: Pakketverlies (%) in functie van de pakketgrootte (bytes)

3.5.5.3 Test 3: Pakketgrootte

In de derde test wordt de invloed van de pakketgrootte op de delay en het pakketverlies bestudeerd. Fig. 3.14 toont de gemiddelde delay (μs) in functie van de pakketgrootte (bytes). In de figuur zien we dat de delay stijgt bij grotere pakketten, waaruit kan worden afgeleid dat de overlay servers meer tijd nodig hebben om grotere pakketten te verwerken.

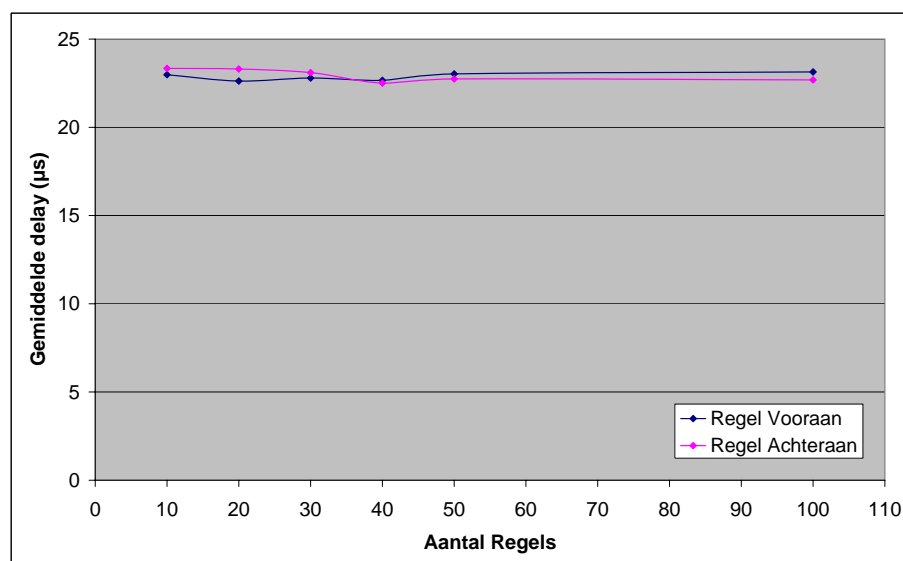
In Fig. 3.15 wordt het pakketverlies (%) uitgezet in functie van de load (Mbps) en de pakket-

grootte (bytes). Het doel hiervan is om na te gaan of de overlay servers meer of minder load aankunnen bij grotere pakketten. De figuur toont duidelijk dat bij grotere pakketten er pas verlies begint op te treden bij hogere loads. De verklaring hiervoor is dat bij grotere pakketten en bij gelijke load, minder pakketten per seconde moeten worden verwerkt.

3.5.5.4 Test 4: Aantal Regels

Het doel van de laatste test is om na te gaan of het aantal routeringsregels in de regellijst van een overlay server invloed heeft op de tijd die nodig is om een pakket te verwerken. Het is namelijk zo dat de server steeds op zoek moet gaan naar alle regels waarvan de bron overeenkomt met het doel van het ontvangen pakket. Bij de test werden extra regels toegevoegd aan de overlay server. De bron werd willekeurig gekozen, maar was steeds verschillend van de bron van de smartbits regel, zodat er zeker geen pakketduplicatie zou optreden. Net zoals bij de tweede test (zie sectie 3.5.5.2) werd de proef zowel uitgevoerd met de smartbits regel vooraan als achteraan de lijst.

De testresultaten worden getoond in Fig. 3.16, waar de gemiddelde delay (μs) wordt uitgezet in functie van het aantal regels. De gemiddelde delay blijft ongeveer constant, waaruit kan worden afgeleid dat het aantal routeringsregels weinig of geen invloed heeft op de verwerkingssnelheid van de overlay servers. De curves lopen ook ongeveer gelijk wat het resultaat uit test 2 dat de positie in de lijst niet van belang is, bevestigt.



Figuur 3.16: De gemiddelde delay (μs) in functie van het aantal regels

Hoofdstuk 4

Algoritmen

Vooraleer het OMP de overlay servers kan configureren, in functie van een multicastsessie, moeten er twee problemen worden opgelost. Eerst en vooral moet er een geschikt RP gekozen worden. Er moet ook, voor elke verzendende overlay server, een multicast distributie boom berekend worden die elke ontvangende overlay server bevat. Deze bomen bestaan uit een pad van de verzender naar het RP en een boom, met als wortel het RP, die elke ontvanger bevat (de RP-boom). Deze paden en bomen kunnen dan worden omgezet in routeringsregels, die gebruikt worden om de overlay servers te configureren.

In dit hoofdstuk worden een aantal algoritmen besproken voor het oplossen van deze problemen. Eerst wordt het probleem formeel geformuleerd. Vervolgens wordt een algoritme besproken voor het kiezen van een geschikt RP. Tenslotte wordt nog een algoritme beschreven voor het berekenen van de multicastbomen voor een many-to-many multicastsessie. In hoofdstuk 5 wordt de performantie en kwaliteit van de algoritmen geëvalueerd, met behulp van een aantal tests.

4.1 Probleemstelling

Het overlay netwerk kan gemodeleerd worden als een gerichte graaf met als knopen de overlay servers. Met elke tak (edge) is een kost en delay geassocieerd. Voor elke multicastsessie doet één van de overlay servers dienst als RP.

Voor elke verzender wordt gezocht naar de laagste-kost multicast distributie boom. Deze bestaat uit een pad van de verzender naar het RP en een RP-boom, met als wortel het RP, die alle

ontvangers bevat. De delay van de multicastboom (de som van de delay van het pad en die van de RP-boom) moet kleiner zijn dan de vooropgestelde delay bound. De RP-boom komt overeen met een Diameter Constrained Minimal Steiner Tree (DCMST) [27].

Formeel wordt dit probleem als volgt beschreven: Gegeven een gerichte graaf $G(V, E)$ (met V de set van overlay servers en E die van de edges), een niet-lege set van verzenders $S = \{s_1, \dots, s_n\}$ en ontvangers $R = \{r_1, \dots, r_m\}$, een RP en een delay bound D . Er geldt $S \subseteq V$, $R \subseteq V$ en $RP \in V$. Er zijn ook een kostfunctie $c(x, y)$ en delayfunctie $d(x, y)$ gegeven. Hierbij zijn $c(v_i, v_j)$ en $d(v_i, v_j)$ respectievelijk de kost en delay van de tak van v_i naar v_j . De functie $d_{min}(v_i, v_j)$ geeft de minimale delay van de knoop v_i naar v_j . De verkorte notaties $c(x)$ en $d(x)$ geven respectievelijk de kost en delay van een pad x . Voor een boom worden dezelfde notaties gehanteerd. De kost en delay van paden en bomen wordt berekend zoals gedefinieerd in hoofdstuk 2.

Voor elke verzender $s_i \in S$ gaan we op zoek naar een pad P_i en een RP-boom T_i die voldoen aan de volgende eigenschappen:

1. s_i is de eerste node van P_i en het RP de laatste.
2. Het RP is de wortel van T_i en $\forall r_j \in R: r_j \in T_i$.
3. $d(P_i) + d(T_i) \leq D$
4. $c(T_i) + c(P_i)$ is kleiner dan of gelijk aan de kost van alle pad-boom koppels die voldoen aan (1), (2) en (3).

Aangezien het DCMST probleem NP-compleet is [27], is het hier beschreven probleem ook NP-compleet. De ontworpen algoritmen zijn dan ook heuristische die in veel gevallen een niet-optimale oplossing teruggeven. Hun uitvoeringstijd is wel polynomiaal, wat bij een algoritme dat de optimale oplossing zou zoeken niet mogelijk is. In sectie 4.4 wordt een algoritme besproken, met exponentiële uitvoeringstijd, dat wel het optimale RP en de laagste kost multicastbomen zoekt.

4.2 Rendezvous Point Algoritme

Bij veel toepassingen is er nood aan een centrale server waar de data die verstuurd wordt tussen de verschillende gebruikers kan worden verwerkt. Bij het opzetten van de multicastsessie zal

er uit de set van overlay servers zo een centraal punt worden gekozen. Het multicast boom algoritme zal ervoor zorgen dat het pad van elke verzender naar elke ontvanger deze server bevat. We noemen dit punt het Rendezvous Point (RP).

Zoals gezegd in hoofdstuk 2 (zie sectie 2.3) wordt de totale kost van een multicastsessie gedefiniëerd als de kost om een boodschap te multicasten van elke verzender naar elke ontvanger. Deze kost is dus gelijk aan:

$$c = \sum_{s \in S} c(T_s)$$

Hierbij is S de set van verzenders en T_s de multicast distributie boom horende bij de verzender s .

Aangezien de multicastbomen van een sessie afhangen van het gekozen RP, zal de totale kost van de sessie hier ook van afhangen. De bedoeling van het RP algoritme is om een RP te kiezen waarvoor die totale kost de optimale kost zo dicht mogelijk benadert.

Het algoritme bestaat uit twee stappen. In de eerste stap wordt aan de hand van een bepaald criterium een subset van alle overlay servers gekozen. In de tweede stap wordt een schatting gemaakt van de totale multicastkost voor de overlay servers uit de eerder geselecteerde subset. De server waarvoor de schatting het laagst is, wordt als RP gekozen.

Er wordt dus in beide stappen een schatting gemaakt van de totale multicastkost van een aantal overlay servers. Het voordeel van in twee stappen te werken is dat er eerst een snelle maar minder nauwkeurige schatting kan worden uitgevoerd. Hierdoor moet de trage maar nauwkeurigere schatting niet voor elke server worden herhaald.

4.2.1 Initiële Schatting

In de eerste stap worden alle overlay servers gerangschikt volgens stijgende geschatte kost. De K servers met de laagste geschatte kost worden geselecteerd en doorgestuurd naar de tweede stap. We stellen twee technieken voor om een snelle initiële schatting uit te voeren.

4.2.1.1 MinMax End-to-End Delay

Deze techniek is een uitbreiding van de MinMaxD techniek [21, 12]. De geschatte kost van een kandidaat RP wordt gelijk gesteld aan het maximum van de minimale delay paden van elke

verzender naar elke ontvanger die de kandidaat als tussenliggende node bevat. De geschatte kost $\tilde{c}(o)$ van de overlay server o is dus:

$$\tilde{c}(o) = \max_{i=1..n} d_{min}(s_i, o) + \max_{j=1..m} d_{min}(o, r_j)$$

Met $S = \{s_1, \dots, s_n\}$ en $R = \{r_1, \dots, r_m\}$ respectievelijk de set van verzenders en ontvangers.

Als we aannemen dat de minimale delay tussen elk paar overlay servers reeds gekend is, is de tijdscomplexiteit om de kost van één overlay server te schatten gelijk aan $O(s + r)$, met s het aantal verzenders en r het aantal ontvangers. De totale tijdscomplexiteit is dan $O(o \cdot (s + r))$. Aangezien $s, r \leq o$, met o het aantal overlay servers, is de slechtste geval complexiteit van de schatter gelijk aan $O(o^2)$.

4.2.1.2 Minimal Delay Tree

Bij deze schatter wordt voor elk kandidaat RP een boom berekend. Deze bevat het minimale delay pad van de kandidaat naar elke ontvanger van de sessie. Dit is met andere woorden de kortste delay-pad boom. De schatting van de kost is de totale kost van die boom.

De minimale delay paden worden gezocht met behulp van het algoritme van Dijkstra [9]. De slechtste geval tijdscomplexiteit van dijkstra is kwadratisch in het aantal nodes, wat in dit geval $O(o^2)$ geeft. Het is mogelijk om in één keer een pad naar alle nodes te zoeken, zonder dat de slechtste geval complexiteit van het algoritme van dijkstra toeneemt. De complexiteit om een kortste pad naar elke ontvanger te zoeken is dus ook gelijk aan $O(o^2)$. Aangezien er een schatting moet gemaakt worden voor elke overlay server, is de slechtste geval complexiteit van de schatter gelijk aan $O(o^3)$.

4.2.2 Rendezvous Point Selectie

In deze stap wordt, net zoals in de eerste stap, een schatting gemaakt van de totale multicastkost. Hier wordt de kost enkel geschat voor de K beste kandidaten uit de initiële selectie stap van het algoritme. We stellen twee technieken voor om de multicastkost te schatten. In beide gevallen wordt het probleem herleid tot een DCMST probleem.

4.2.2.1 Worst Case Multicast Tree Cost

De Worst Case Multicast Tree Cost (WTrC) schatter, schat de kost van de overlay server o als de kost van de RP-boom met als bron o en delay bound $\max_{r \in R}(d_{min}(o, r))$. De multicastboom kan berekend worden met een DCMST heuristiek zoals het SUB algoritme [6].

Als het SUB algoritme gebruikt wordt is de complexiteit voor het berekenen van de RP-boom gelijk aan $O(o^2 \cdot r^2)$. Met o het aantal overlay servers en r het aantal ontvangers. Het aantal bomen dat moet berekend worden is gelijk aan het aantal kandidaten K dat in de eerste stap van het algoritme wordt geselecteerd. De totale complexiteit is dan $O(K \cdot o^2 \cdot r^2)$. Aangezien $K, r \leq o$ is dit in het slechtste geval $O(o^5)$.

4.2.2.2 Worst Case Total Multicast Cost

De Worst Case Total Multicast Cost (WTC) schatter is een uitbreiding van de WTrC schattingstechniek. Hier wordt bovenop de kost om te multicasten van het RP naar de ontvangers ook de kost om de boodschap te versturen van de verzenders naar het RP in acht genomen. Stel de boom zoals berekend in sectie 4.2.2.1 gelijk aan T_o . Het pad met minimale delay van overlay server v_i naar v_j is gelijk aan $p_{min}(v_i, v_j)$. De totale multicastkost voor de overlay server o wordt dan geschat op:

$$\tilde{c}(o) = |S| \cdot c(T_o) + \sum_{s \in S} c(p_{min}(s, o))$$

Hierbij stelt S de set van verzenders en $|S|$ het totaal aantal verzenders voor.

Als we weer gebruik maken van het SUB algoritme is de complexiteit voor het berekenen van de multicastboom gelijk aan $O(o^2 \cdot r^2)$. Met behulp van het algoritme van dijkstra kunnen de paden voor één kandidaat in $O(o^2)$ tijd berekend worden. Aangezien zowel de boom als de paden moeten berekend worden voor elk van de K kandidaten geselecteerd in de eerste stap, is de totale complexiteit gelijk aan $O(K \cdot (o^2 \cdot r^2 + o^2))$, wat overeenkomt met $O(K \cdot o^2 \cdot r^2)$. De slechtste geval complexiteit is dus gelijk aan die van de “Worst Case Multicast Tree Cost” schatter.

4.3 Multicastboom Algoritme

Zoals eerder vermeld (zie sectie 4.1) komt het zoeken van een RP-boom overeen met het DCMST probleem. In eerdere publicaties [1, 6, 27] zijn reeds oplossingen voorgesteld voor het zoeken van een DCMST. Er kan dus een bestaand algoritme worden gebruikt om de RP-boom te zoeken.

Op die manier kan voor elke verzender in twee stappen een multicast distributie boom worden opgesteld. Eerst wordt de lijst met paden gezocht van de verzender naar het RP die voldoen aan de delay bound D . Vervolgens wordt voor elk van deze paden een RP-boom gezocht (rekening houdend met de totale delay van het pad). Voor elke verzender wordt dan het pad-boom koppel gekozen die de totale kost minimaliseren.

Om de geheugen- en tijdscomplexiteit te beperken, wordt een extra stap toegevoegd aan het algoritme. Voor de multicastbomen worden berekend, worden de verzenders opgedeeld in een aantal groepen (clusters). Er wordt dan één RP-boom per cluster geselecteerd, in plaats van één per verzender. Hierdoor moeten niet alleen minder bomen worden opgeslagen (geheugencomplexiteit) maar moeten er ook minder worden berekend (tijdscomplexiteit). Door middel van een clusteralgoritme worden de verzenders die zich op een korte delay van elkaar bevinden (en dus waarschijnlijk ook gelijkaardige RP-bomen gaan hebben) in dezelfde cluster ondergebracht.

4.3.1 Pad Selectie

Voor elke verzender s_i wordt de geordende lijst $P_i = \{p_{i1}, \dots, p_{ij}\}$ van paden naar het RP gezocht. Deze paden zijn gesorteerd volgens stijgende delay, zodat geldt:

$$\forall p_{ik}, p_{il} \in P_i : k \leq l \Leftrightarrow d(p_{ik}) \leq d(p_{il})$$

Bij elk pad p_{ik} van de verzender s_i naar het RP hoort een multicastboom T_k . De delay bound van deze boom is gelijk aan $D_k = D - d(p_{ik})$. Met D de gegeven maximale delay bound tussen een verzender en een ontvanger. Zo een boom bestaat enkel als:

$$D_k \geq \max_{r \in R} (d_{\min}(RP, r))$$

Hierbij stelt R de set van ontvangers voor. Hieruit kan worden afgeleid dat enkel de paden p moeten beschouwd worden, waarvoor:

$$d(p) \leq D - \max_{r \in R} (d_{\min}(RP, r)) \tag{4.1}$$

Eigenschap 1. *Stel T_1 en T_2 twee laagste-kost multicastbomen voor dezelfde verzender en set van ontvangers. T_1 hoort bij het pad p_1 en heeft als delay bound $D_1 = D - d(p_1)$. T_2 hoort bij het pad p_2 en heeft als delay bound $D_2 = D - d(p_2)$. Er geldt:*

$$d(p_1) < d(p_2) \Rightarrow c(T_1) \leq c(T_2)$$

Hierbij stelt $c(T)$ de totale kost van de boom T voor en $d(p)$ de delay van het pad p .

Bewijs We geven een bewijs uit het ongerijmde. Stel dat geldt:

$$d(p_1) < d(p_2) \Rightarrow c(T_1) > c(T_2)$$

Uit $d(p_1) < d(p_2)$ kunnen we afleiden dat $D_2 < D_1$. De paddelay van de verzender naar elke ontvanger in T_2 is kleiner dan of gelijk aan D_2 . Aangezien $D_2 < D_1$ is de maximale paddelay in T_2 ook kleiner dan D_1 . Aangezien $c(T_2) < c(T_1)$ en T_2 voldoet aan de delay bound D_1 , is niet T_1 maar T_2 de laagste-kost multicastboom voor het pad p_1 . Dit is tegenstrijdig met het gegeven. \square

Uit de definitie van D_n volgt eenvoudig dat de delay bound van de boom omgekeerd evenredig is met de delay van het bijhorende pad. Met behulp van deze eigenschap en Eigenschap 1 kunnen een aantal paden worden geschrapt waarvoor de totale kost van het pad en de bijhorende boom toch niet optimaal kan zijn. Deze padselectie criteria zijn:

- Als een pad een hogere of gelijke kost heeft dan een pad met lagere delay kan dit worden geschrapt. De kost van de RP-boom horende bij het pad zal toch hoger zijn dan of gelijk zijn aan de kost van de RP-boom horende bij het pad met lagere delay (zie Eigenschap 1).
- Als verschillende paden eenzelfde delay hebben moet enkel dat met laagste kost worden bewaard. De boom horende bij de paden zal namelijk dezelfde zijn en dus enkel het laagste-kost pad zal tot de optimale oplossing kunnen leiden.

Eens deze criteria zijn toegepast, zal de delay van de paden van een bepaalde verzender omgekeerd evenredig zijn met de kost ervan.

In veel gevallen gaan alle (of toch een groot aantal) overlay servers rechtstreeks met elkaar verbonden zijn op overlay niveau. Hierdoor zal het netwerk een groot aantal takken bevatten.

Tabel 4.1: De delay (d) en kost (c) van de paden van een verzender (a) Voor padselectie (b) Na het uitvoeren van de padselectie criteria

(a)				(b)			
	pad	d	c		pad	d	c
	p_{11}	1	7		p_{11}	1	7
	p_{12}	3	9		p_{14}	5	4
	p_{13}	5	5		p_{15}	7	3
	p_{14}	5	4		p_{16}	8	2
	p_{15}	7	3				
	p_{16}	8	2				

De tijd om alle paden, die voldoen aan een bepaalde delay bound, te berekenen zal dan hoog oplopen. Daarom zullen we slechts een beperkt aantal paden zoeken. Met behulp van het algoritme van Yen [25] kunnen de K laagste-delay paden worden gevonden.

Het algoritme van Yen zal de K gevonden paden (of minder als er geen K zijn die voldoen aan de delay bound) teruggeven in volgorde van stijgende delay. In dit geval is het mogelijk om in $O(n)$ tijd (met n het aantal paden) de twee beschreven padselectie criteria toe te passen. Yen's algoritme heeft een complexiteit van $O(K \cdot n^3)$. In combinatie met de padselectie criteria geeft dit $O(K \cdot n^3 + n)$. Dit moeten worden herhaald voor elke verzender, wat de totale tijdscomplexiteit van deze stap $O(K \cdot s \cdot n^3)$ maakt, met s het aantal verzenders.

Voorbeeld Beschouw als voorbeeld de paden in Tabel 4.1(a). Volgens het eerste padselectie criterium kunnen we p_{12} schrappen (aangezien de kost van p_{11} lager is). Gebruik makend van het tweede criterium kan ook p_{13} worden geschrapt (aangezien p_{14} dezelfde delay heeft, maar een lagere kost). Tabel 4.1(b) geeft de situatie na het toepassen van de criteria.

□

4.3.2 Clustering

Om de verzenders te clusteren stellen we het “Minimal Intra Cluster Dissimilarity” (MICD) clusteralgoritme voor. Het algoritme is gebaseerd op k -means [16] en Partition Around Medoids (PAM) [13]. Net zoals bij PAM wordt gebruik gemaakt van een dissimilariteitsmatrix [22, 23], zodat het algoritme onafhankelijk is van de gekozen dissimilariteitsmaat.

De dissimilariteit is een maat voor ongelijkheid tussen twee elementen. Een dissimilariteitsmaat moet voldoen aan de volgende eigenschappen [22]:

1. $d(x, y) \geq 0$
2. $d(x, y) = 0 \Leftrightarrow x = y$
3. $d(x, y) = d(y, x)$

Een afstandsmaat voldoet daarbovenop ook aan de driehoeksongelijkheid:

4. $d(x, z) \leq d(x, y) + d(y, z)$

Elke afstandsmaat is dus een dissimilariteitsmaat, maar niet omgekeerd. In de dissimilariteitsmatrix komt de waarde op rij i en kolom j overeen met de dissimilariteit tussen de elementen e_i en e_j .

Bovenop de dissimilariteitsmatrix moet ook het aantal clusters K worden opgegeven. Het MICD algoritme volgt in grote lijnen dezelfde stappen als k -means:

1. Voeg elk te clusteren element (in deze context de verzenders) volgens een bepaald criterium toe aan één van de K clusters (bijvoorbeeld at random).
2. Kies een willekeurig element en voeg het toe aan de cluster waarbij de gemiddelde dissimilariteit van het element tot de elementen van die cluster zo laag mogelijk is. De gemiddelde dissimilariteit van a tot de cluster C wordt, als $a \notin C$, gegeven door:

$$\frac{\sum_{c \in C} \text{diss}(a, c)}{|C|}$$

Als $a \in C$ wordt deze gegeven door:

$$\frac{\sum_{c \in C} \text{diss}(a, c)}{|C| - 1}$$

Hierbij stelt $|C|$ het aantal elementen in de cluster C voor. Als een element zich alleen in een cluster bevindt, is de gemiddelde dissimilariteit tot zijn eigen cluster gelijk aan 0.

3. Herhaal stap 2 tot er geen enkel element meer van cluster verandert of totdat een bepaalde iteratiegrens bereikt is.

Voor elke cluster kan de som van de dissimilariteiten van elk element tot de elementen van de cluster worden opgeslagen. De tijdscomplexiteit om deze tabel up to date te houden bij het toevoegen van een element tot of het verwijderen van een element uit een cluster is $O(n)$ (met n het aantal te clusteren elementen). Met behulp van deze tabellen kan voor een gegeven element de cluster die de gemiddelde dissimilariteit minimaliseert worden gevonden in $O(K)$ tijd (met K het aantal clusters). Aangezien stap 2 van het algoritme per iteratie voor elk element de beste cluster moet zoeken wordt de totale complexiteit $O(t \cdot n \cdot (n + K))$, met t het totaal aantal iteraties. Aangezien $K \leq n$ wordt dit $O(t \cdot n^2)$. In de praktijk is $t \ll n$ en benadert de tijdscomplexiteit $O(n^2)$. Hoewel dit slechter is dan de complexiteit van k -means ($O(t \cdot n \cdot K)$), is het algoritme toch nog zeer schaalbaar.

Door deze aanpassing om de tijdscomplexiteit te verbeteren wordt er bovenop de elementen van de cluster voor elke cluster ook een rij met n getallen bijgehouden. De totale geheugencomplexiteit wordt dan $O(K \cdot n)$. Bij k -means moet enkel de cluster van elk element worden bijgehouden en het gemiddelde van die cluster, wat een geheugencomplexiteit geeft van $O(n + k)$, wat $O(n)$ geeft (aangezien $K \leq n$).

In de clustering stap van het multicastboom algoritme wordt de dissimilariteit tussen de verzenders s_i en s_j gedefinieerd als de minimale delay tussen s_i en s_j . Aangezien de minimale delay van s_i naar s_j verschillend kan zijn dan die van s_j naar s_i , wordt nog het maximum van de twee genomen. Anders is namelijk niet voldaan aan eigenschap 3 van een geldige dissimilariteitsmaat. Dit kan worden uitgedrukt als:

$$diss(s_i, s_j) = \max(d_{min}(s_i, s_j), d_{min}(s_j, s_i)) \quad (4.2)$$

De voorgestelde dissimilariteitsmaat (vergelijking 4.2) voldoet niet aan de driehoeksongelijkheid en is dus geen geldige afstandsmaat. Het is eenvoudig aan te tonen dat wel is voldaan aan de drie eigenschappen van een geldige dissimilariteitsmaat.

Voorbeeld We illustreren de werking van het MICD algoritme aan de hand van een voorbeeld.

Beschouw de dissimilariteitsmatrix:

$$D = \begin{pmatrix} 0 & 1 & 9 & 7 \\ 1 & 0 & 7 & 6 \\ 9 & 7 & 0 & 2 \\ 7 & 6 & 2 & 0 \end{pmatrix}$$

De matrix bevat de dissimilariteit tussen de elementen x_1 , x_2 , x_3 en x_4 . Merk op dat de waarden op de hoofddiagonaal gelijk zijn aan 0 (eigenschap 2 van de afstandsmetriek) en dat de matrix symmetrisch is (eigenschap 3 van de dissimilariteitsmaat).

Stel nu het aantal clusters $K = 2$. Volgens stap 1 (de opbouw-stap) van het MICD algoritme voegen we elk element toe aan een willekeurige cluster. Bijvoorbeeld $C_1 = \{x_1, x_3\}$ en $C_2 = \{x_2, x_4\}$. Vervolgens gaan we voor elk element de cluster zoeken tot welke de gemiddelde dissimilariteit zo laag mogelijk is (stap 2 van het MICD algoritme). De gemiddelde dissimilariteit van x_1 tot de twee clusters is gelijk aan:

$$\begin{aligned} \text{diss}(x_1, C_1) &= \text{diss}(x_1, x_3) &&= 9 \\ \text{diss}(x_1, C_2) &= \frac{1}{2}(\text{diss}(x_1, x_2) + \text{diss}(x_1, x_4)) &&= \frac{8}{2} = 4 \end{aligned}$$

Het element x_1 wordt toegevoegd aan C_2 . De clusters zijn nu $C_1 = \{x_3\}$ en $C_2 = \{x_1, x_2, x_4\}$. x_2 heeft de volgende gemiddelde dissimilariteiten:

$$\begin{aligned} \text{diss}(x_2, C_1) &= \text{diss}(x_2, x_3) &&= 7 \\ \text{diss}(x_2, C_2) &= \frac{1}{2}(\text{diss}(x_2, x_1) + \text{diss}(x_2, x_4)) &&= 3.5 \end{aligned}$$

x_2 blijft in zijn huidige cluster. x_3 bevindt zich alleen in een cluster. De gemiddelde dissimilariteit tot zijn eigen cluster is dan 0 en het zal dus sowieso niet van cluster veranderen. Voor x_4 krijgen we:

$$\begin{aligned} \text{diss}(x_4, C_1) &= \text{diss}(x_4, x_3) &&= 2 \\ \text{diss}(x_4, C_2) &= \frac{1}{2}(\text{diss}(x_4, x_1) + \text{diss}(x_4, x_2)) &&= \frac{13}{2} = 6.5 \end{aligned}$$

Element x_4 wordt toegevoegd aan C_1 en de eerste iteratie eindigt met clustering $C_1 = \{x_3, x_4\}$

en $C_2 = \{x_1, x_2\}$. De tweede iteratie geeft:

$$diss(x_1, C_1) = \frac{1}{2}(diss(x_1, x_3) + diss(x_1, x_4)) = 8$$

$$diss(x_1, C_2) = diss(x_1, x_2) = 1$$

$$diss(x_2, C_1) = \frac{1}{2}(diss(x_2, x_3) + diss(x_2, x_4)) = 6.5$$

$$diss(x_2, C_2) = diss(x_2, x_1) = 1$$

$$diss(x_3, C_1) = diss(x_3, x_4) = 2$$

$$diss(x_3, C_2) = \frac{1}{2}(diss(x_3, x_1) + diss(x_3, x_2)) = 8$$

$$diss(x_4, C_1) = diss(x_4, x_3) = 2$$

$$diss(x_4, C_2) = \frac{1}{2}(diss(x_4, x_1) + diss(x_4, x_2)) = 6.5$$

In de tweede iteratie verandert geen enkel element nog van cluster. Het algoritme stopt dus na twee iteraties met de clusters $C_1 = \{x_3, x_4\}$ en $C_2 = \{x_1, x_2\}$. \square

4.3.3 Boom Creatie

In de laatste stap wordt per cluster gezocht naar een geschikte RP-boom en een pad van elke verzender, van de cluster, naar het RP. De paden en boom worden zo gekozen zodat de kost om een boodschap te multicasten van elke verzender in de cluster naar elke ontvanger zo laag mogelijk is. Deze kost is gelijk aan:

$$c = k \cdot c(T_i) + \sum_{j=1}^k c(p_j)$$

Hierbij is k het aantal verzenders in de cluster, T_i de gekozen multicastboom en p_j het gekozen pad voor verzender s_j .

We beschouwen de cluster C_i , met de set van verzenders $S_i = \{s_1, \dots, s_k\}$. Elke verzender s_j heeft een lijst van paden $P_j = \{p_{j1}, \dots, p_{jl}\}$ bekomen in de padselectie stap van het algoritme. We stellen dat de paden gerangschikt zijn volgens stijgende delay (en zoals vermeld in sectie 4.3.1 dus ook op dalende kost). Voor elke verzender s_j van C_i wordt één pad uit P_j gekozen. De multicastboom horende bij deze set van paden is enkel afhankelijk van het pad met hoogste delay. Daarom wordt een boom geassocieerd met een paddelay in plaats van met een set van

paden. T_a stelt dan ook de boom voor horende bij de paddelay a . Elk pad uit de set die hoort bij T_a heeft een delay kleiner of gelijk aan a .

De multicastboom en bijhorende paden worden als volgt bepaald. Stel s_m de verzender waarvoor $d(p_{m1}) = \max_{j=1..k}(d(p_{j1}))$, met andere woorden het laagste-delay pad van s_m heeft een hogere of even hoge delay dan het laagste-delay pad van de andere verzenders in de cluster. Alle paden met lagere delay mogen buiten beschouwing worden gelaten. Als we namelijk een multicastboom zoeken voor een paddelay lager dan $\max_{j=1..k}(d(p_{j1}))$, zal s_m geen pad naar het RP hebben, zodat de delay bound D niet wordt overschreden.

Vervolgens maken we een lijst met alle mogelijke paddelays in de cluster (in stijgende volgorde). Zo bekomen we de lijst $\{a_1, \dots, a_n\}$ (merk op dat $a_1 = d(p_{m1})$). Met behulp van een DCMST algoritme kan dan voor elke delay a de multicastboom T_a gevonden worden. De boom heeft als wortel het RP, als ontvangers de set R van ontvangers en als delay bound $D_a = D - a$. Voor een boom T_a selecteren we van elke verzender het pad met minimale kost en delay $\leq a$.

Het is mogelijk om een ondergrens te bepalen van de totale kost, voor een gegeven paddelay aan de hand van reeds berekende bomen voor lagere paddelays. Stel dat de kost van de RP-boom voor de paddelay a reeds gekend is. Uit Eigenschap 1 en het feit dat $a < b$ (en dus $D_a > D_b$) kunnen we afleiden dat $c(T_a) \leq c(T_b)$. Een ondergrens voor de totale multicastkost horende bij de paddelay b is dan:

$$\tilde{c}_b = k \cdot c(T_a) + \sum_{j=1}^k c(p_j)$$

Hierbij is p_j het laagste kostpad van verzender s_j naar het RP, waarvoor $d(p_j) \leq b$. Als deze kost hoger is dan de huidige optimale kost, heeft het geen zin om T_b te berekenen.

De totale tijdscomplexiteit van deze stap hangt grotendeels af van het gekozen algoritme voor het berekenen van de RP-bomen. Stel dat de complexiteit van dit algoritme $O(X)$ is. In het slechtste geval moet voor elk pad van elke cluster een multicastboom worden gezocht. De totale complexiteit wordt dan $O(k \cdot p \cdot X)$, met k het aantal clusters en p het maximaal aantal paden per cluster. Het SUB algoritme heeft complexiteit $O(o^2 \cdot r^2)$ [6] met o het aantal overlay servers en r het aantal ontvangers. De complexiteit is dan $O(k \cdot p \cdot o^2 \cdot r^2)$. In de praktijk gaat het totaal aantal te berekenen bomen, dankzij de pad-selectie criteria en de ondergrens techniek, veel kleiner zijn dan $k \cdot p$

Tabel 4.2: De delay (d) en kost (c) van de paden van de 4 verzenders van een cluster

s_1			s_2			s_3			s_4		
pad	d	c	pad	d	c	pad	d	c	pad	d	c
p_{11}	1	7	p_{21}	4	3	p_{31}	2	1	p_{41}	2	12
p_{12}	5	4	p_{22}	6	2				p_{42}	4	11
p_{13}	7	3							p_{43}	5	8
p_{14}	8	2							p_{44}	6	7

Opmerking. Aangezien het RP en de set R onveranderd blijven tijdens de loop van het algoritme zal eenzelfde paddelay steeds aanleiding geven tot dezelfde RP-boom. Daarom kan het handig zijn om reeds berekende bomen in het geheugen te houden, aangezien een paddelay in verschillende clusters kan terugkomen. Op die manier moet eenzelfde boom nooit meer dan één keer worden berekend. \square

Opmerking. Hoewel Eigenschap 1 in theorie altijd geldt, zal het gebruikte algoritme voor het berekenen van de RP-bomen in veel gevallen een heuristisch zijn. Het is mogelijk dat de oplossingen van deze heuristiek niet aan de eigenschap voldoen. Het algoritme kan aangepast worden zodat de eigenschap blijft gelden onafhankelijk van het gebruikte algoritme. Stel dat de boom T_a voor paddelay a is berekend. T_a voldoet aan de delay bounds horende bij paddelays lager dan a . We vervangen alle bomen horende bij deze paddelays die een hogere kost hebben dan T_a door T_a .

Het is mogelijk dat T_a voldoet aan de delay bound van een paddelay hoger dan a . Als de kost van de bomen horende bij deze paddelays hoger is dan die van T_a moeten ze er ook door vervangen worden. \square

Voorbeeld We illustreren de boomcreatie stap van het algoritme met behulp van de verzenders in Tabel 4.2. s_m is in dit voorbeeld s_2 met $d(p_{21}) = 4$. De lijst van verschillende paddelays wordt dan $\{4, 5, 6, 7, 8\}$. Stel dat de minimale-kost bomen voor deze paddelays de volgende kosten hebben:

$$c(T_4) = 20, \quad c(T_5) = 20, \quad c(T_6) = 40, \quad c(T_7) = 44, \quad c(T_8) = 46$$

Eerst berekenen we de totale kost voor de paddelay 4:

$$delay = 4 : kost = c(p_{11}) + c(p_{21}) + c(p_{31}) + c(p_{42}) + 4 c(T_4) = 102 \text{ (exact)}$$

Het huidige optimum wordt op 102 ingesteld en de ondergrens voor paddelay 5 wordt berekend:

$$\text{delay} = 5 : \text{kost} = c(p_{12}) + c(p_{21}) + c(p_{31}) + c(p_{43}) + 4 c(T_4) = 96 \text{ (ondergrens)}$$

Aangezien de kost van T_5 gelijk is aan die van T_4 is de berekende ondergrens gelijk aan de echte totale kost en wordt het nieuwe optimum 96. De ondergrens voor paddelay 6 is:

$$\text{delay} = 6 : \text{kost} = c(p_{12}) + c(p_{22}) + c(p_{31}) + c(p_{44}) + 4 c(T_5) = 94 \text{ (ondergrens)}$$

Dit is lager dan het huidige optimum dus moet ook T_6 worden berekend:

$$\text{delay} = 6 : \text{kost} = c(p_{12}) + c(p_{22}) + c(p_{31}) + c(p_{44}) + 4 c(T_6) = 174 \text{ (exact)}$$

Deze kost is hoger dan het optimum, dus we gaan verder met het berekenen van de ondergrens voor de paddelay 7 (we gebruiken de kost van T_6):

$$\text{delay} = 7 : \text{kost} = c(p_{13}) + c(p_{22}) + c(p_{31}) + c(p_{44}) + 4 c(T_6) = 173 \text{ (ondergrens)}$$

Deze ondergrens is hoger dan het huidig optimum en T_7 wordt dus niet berekend. De ondergrens voor de paddelay 8 is:

$$\text{delay} = 8 : \text{kost} = c(p_{14}) + c(p_{22}) + c(p_{31}) + c(p_{44}) + 4 c(T_6) = 172 \text{ (ondergrens)}$$

Deze is ook hoger dan het huidige optimum en T_8 wordt ook niet berekend. De laagste totale kost wordt dus bereikt bij T_5 met bijhorende paden $\{p_{12}, p_{21}, p_{31}, p_{43}\}$. \square

4.4 Optimaal Algoritme

Een algoritme dat de globaal laagste kost multicastboom berekent voor elke verzender (met het optimale RP) kan worden afgeleid uit de besproken heuristiek. De stappen van de heuristiek moeten als volgt worden aangepast:

- **Rendezvous Point:**

Het beschreven RP algoritme geeft niet gegarandeerd het best mogelijke RP terug. Het exact algoritme moet dan ook herhaald worden voor elke overlay servers als RP.

- **Clustering:**

De clustering stap moet worden weggelaten. Als er gebruik wordt gemaakt van clustering

wordt aan elke verzender de multicastboom van de slechtste verzender in die cluster toegerekend, wat tot een suboptimale oplossing leidt. Deze verandering kan makkelijk worden doorgevoerd in een bestaande implementatie van de heuristiek door het aantal clusters in te stellen op het totaal aantal verzenders (elke verzender zal zich dan als enige element in een cluster bevinden).

- **Pad Selectie:**

Het algoritme gebruikt om de K kortste-delay paden te zoeken moet nu alle paden teruggeven die voldoen aan de delay bound gegeven in vergelijking 4.1. De pad-selectie criteria schrappen enkel paden die tot niet-optimale oplossingen leiden en kunnen dus nog steeds worden toegepast.

- **Boom Creatie:**

De heuristiek gebruikt om het RP-boom probleem op te lossen moet vervangen worden door een exact algoritme. Het ILP algoritme [6] kan hiervoor worden gebruikt.

Het beschreven exacte algoritme is zeer slecht schaalbaar en kan dus enkel gebruikt worden voor netwerken met een klein aantal overlay servers. Het is dan ook in de praktijk niet bruikbaar en enkel nuttig bij het uitvoeren van tests, om de oplossing van een heuristiek te vergelijken met de globaal optimale oplossing.

Hoofdstuk 5

Evaluatie van de Algoritmen

Aan de hand van een aantal willekeurig gegenereerde testnetwerken hebben we de performantie en schaalbaarheid van de algoritmen, besproken in hoofdstuk 4, getest. De testnetwerken zijn gegenereerd met behulp van de BRITE topology generator [17]. Hierbij wordt gebruik gemaakt van het algoritme van waxman [24] dat willekeurige grafen genereert. De kans dat twee nodes u en v met elkaar verbonden zijn, wordt bij dit algoritme gegeven door:

$$P(u, v) = \alpha e^{\frac{-d(u,v)}{\beta L}}$$

Hierbij stelt $d(u, v)$ de euclidische afstand tussen u en v voor en L de maximale afstand tussen twee nodes. De α en β parameters zijn, in de testopstelling, ingesteld op 0.15 en 0.2. Voor elke test werden netwerken van verschillende groottes gebruikt. Hierbij variëren het aantal IP nodes, overlay servers, verzenders en ontvangers. Voor elke node worden er twee bidirectionele edges gegenereerd, zodat er tweemaal zoveel edges zijn als nodes. De delay van een edge is recht evenredig met de afstand tussen de begin- en eindnode ervan. De kost van elke edge wordt willekeurig gekozen, met uniforme distributie, tussen 0 en een maximale waarde die afhangt van de grootte van het netwerk. Elke test is steeds herhaald met 30 verschillende testnetwerken van elke gebruikte grootte.

Bovenop het gegenereerde IP netwerk wordt steeds een full mesh virtueel overlay netwerk geconstrueerd. De servers van het overlay netwerk worden willekeurig gekozen uit de set van IP nodes. De edge tussen twee overlay servers komt overeen met het kortste-hop pad, op de IP laag, dat de twee servers met elkaar verbindt. De delay en kost van een overlay edge zijn gelijk aan de totale delay en kost van dit onderliggend pad.

In de eerste sectie van dit hoofdstuk wordt het rendezvous point algoritme beschouwd. In de eerste test worden de twee besproken technieken voor de eerste stap van het RP algoritme (initiële schatting) met elkaar vergeleken. Vervolgens worden de schattingstechnieken voor de tweede stap (RP selectie) met elkaar vergeleken.

In sectie 5.2 wordt het MICD algoritme vergeleken met een aantal bekende clusteralgoritmen. Er wordt zowel gekeken naar de uitvoeringstijd als de kwaliteit van de bekomen clusters.

Tenslotte wordt in sectie 5.3 dieper ingegaan op het multicastboom algoritme. In de eerste test wordt nagegaan hoeveel niet-optimale paden de padselectie stap schrapt. Vervolgens wordt in de tweede test de invloed van het aantal clusters op de uitvoeringstijd en totale kost van de multicastbomen bekeken. In de laatste test wordt de oplossing van het multicastboom algoritme vergeleken met de globaal optimale oplossing. Hierbij worden ook de invloed van het RP algoritme en het aantal clusters op de totale kost beschouwd.

5.1 Rendezvous Point Algoritme

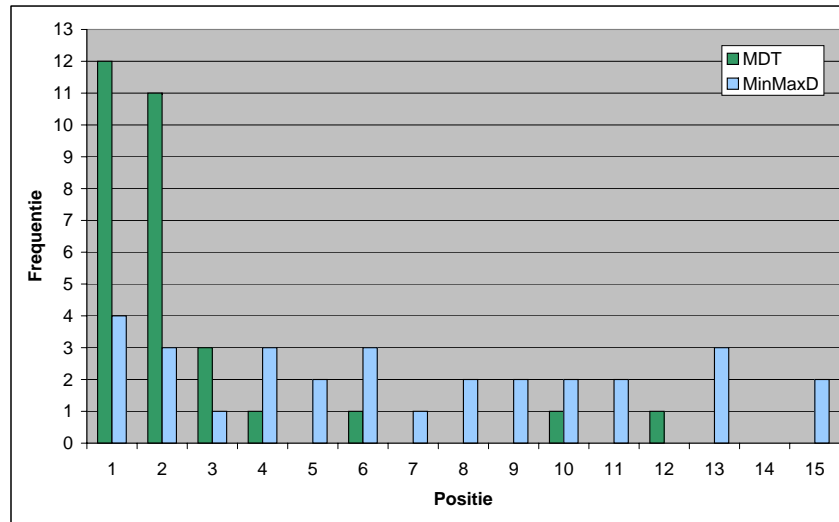
Het algoritme bestaat uit twee stappen waar telkens een schatting wordt gemaakt van de totale multicastkost voor een aantal mogelijk RP's. In een eerste test wordt de nauwkeurigheid van een aantal schatters voor de eerste stap van het algoritme nagegaan. In de tweede test worden een aantal schatters voor de tweede stap getest.

De tests werden uitgevoerd met testnetwerken van twee groottes. Het kleine type heeft 30 IP nodes, 15 overlay servers, 5 verzenders en 5 ontvangers. Netwerken van het grote type hebben 60 IP nodes, 30 overlay servers, 10 verzenders en 10 ontvangers. Merk op dat de verzenders en ontvangers niet noodzakelijk gelijk zijn.

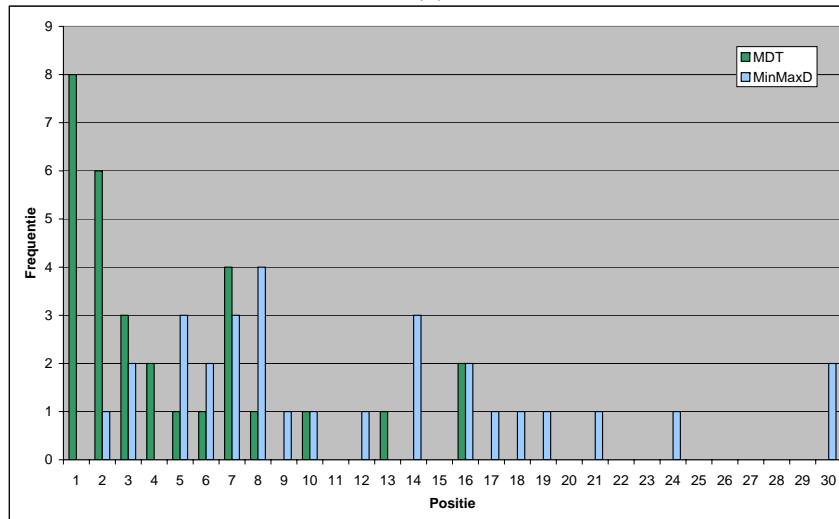
5.1.1 Initiële Schatting

Het doel van de eerste stap van het algoritme is K kandidaat RP's te selecteren uit de set van overlay servers. Op die manier wordt het totaal aantal te controleren kandidaten in de tweede stap, die een aanzienlijk grotere rekentijd per kandidaat vraagt, sterk gereduceerd.

Om de kwaliteit van de schattingstechniek te evalueren worden de kandidaten geranscht van laagste naar hoogste geschatte kost. Vervolgens wordt de positie van de optimale kandidaat



(a)



(b)

Figuur 5.1: Testresultaten voor de initiële schatting. Het aantal keer dat het optimale RP op een bepaalde positie gerangschikt wordt (a) Kleine netwerken (b) Grote netwerken

beschouwd. Hoe lager deze positie, hoe lager de waarde van K kan worden ingesteld zonder dat de optimale kandidaat wordt geschrapt. De optimale kandidaat wordt bepaald door het exacte algoritme (beschreven in sectie 4.4) voor elk mogelijk RP uit te voeren. Het RP waarvoor de totale kost van alle multicastbomen zo laag mogelijk is, is de optimale kandidaat.

Fig. 5.1 geeft de testresultaten weer. De horizontale as toont de mogelijke posities van het optimale RP in de gesorteerde lijst. Positie 1 geeft aan dat de schattingstechniek de optimale kandidaat ook als optimum geschat heeft. De verticale as toont het aantal keer dat de optimale

kandidaat op een bepaalde positie voorkomt. Het is de bedoeling dat deze aantallen zo hoog mogelijk zijn bij lage posities. Dan is er in veel gevallen een goede schatting gemaakt van het optimale RP.

Fig. 5.1(a) vertoont voor MinMax End-to-End Delay (MinMaxD) geen echte piek. Het is zelfs zo dat de aantallen evenredig verdeeld zijn over alle mogelijke posities. Bij grote netwerken is het resultaat beter met een piek rond positie 8. Voor de Minimum Delay Boom (MDT) schatter vertoont zowel het histogram voor kleine als grote netwerken een piek rond positie 1. Zoals eerder gezegd is dit een gewenste eigenschap van een schatter. De MDT techniek is dan ook duidelijk de beste keuze. Voor kleine netwerken is de gemiddelde positie 6 bij MinMaxD en 2 bij MDT. Voor grote netwerken is dit respectievelijk 11 en 4.

5.1.2 RP Selectie met Initiële Schatting

In de tweede test wordt de combinatie van de twee stappen geëvalueerd. Aangezien in de tweede stap enkel de kandidaat met de laagst geschatte totale multicastkost geselecteerd wordt, is het minder belangrijk dat de kost van de optimale kandidaat zo laag mogelijk geschat wordt. Het is belangrijker dat de totale multicastkost van de geselecteerde kandidaat die van de optimale kandidaat zo dicht mogelijk benadert. Als maat voor de evaluatie van de oplossing wordt dan ook de percentuele verhouding tussen de totale multicastkost van het geselecteerde RP en die van het optimale RP gebruikt. Deze verhouding wordt als volgt berekend:

$$\frac{c(\textit{geselecteerde RP})}{c(\textit{optimale RP})} \cdot 100$$

Hierbij stelt $c(x)$ de totale multicastkost voor, als de overlay server x als RP wordt gebruikt.

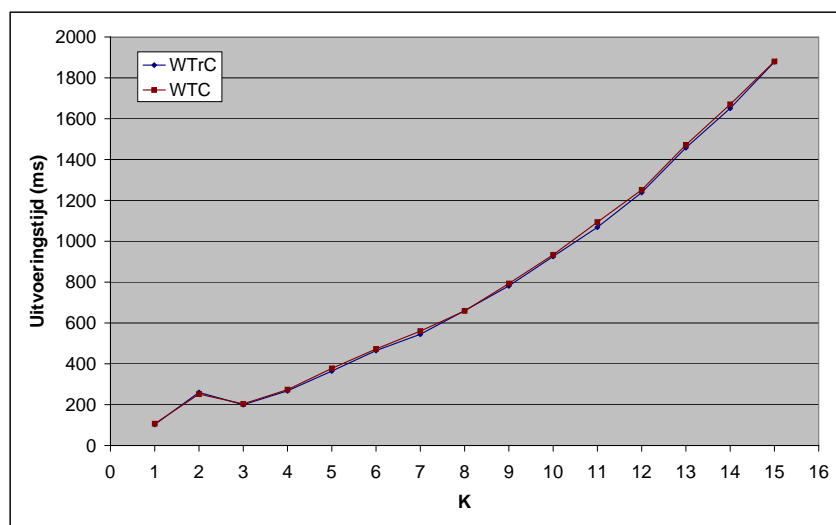
Als initiële schatter wordt MDT gebruikt, die presteert beter dan MinMaxD (zie sectie 5.1.1). Om het effect van de parameter K , van de initiële schatter, op de oplossing en uitvoeringstijd te achterhalen laten we deze variëren tussen 1 en $|O|$, met $|O|$ het aantal overlay servers.

Tabel 5.1 toont de verhouding van de totale multicastkost van het geselecteerde RP tegenover het optimale RP. Zowel het gemiddelde als het maximum over de 30 testnetwerken is gegeven. Een percentage van 100% betekent dat de totale multicastkost voor het geselecteerde RP gelijk is aan die voor het optimale RP (en het geselecteerde RP dus het optimale is). Hoe hoger het percentage, des te hoger de totale multicastkost voor het geselecteerde RP en dus hoe slechter de oplossing.

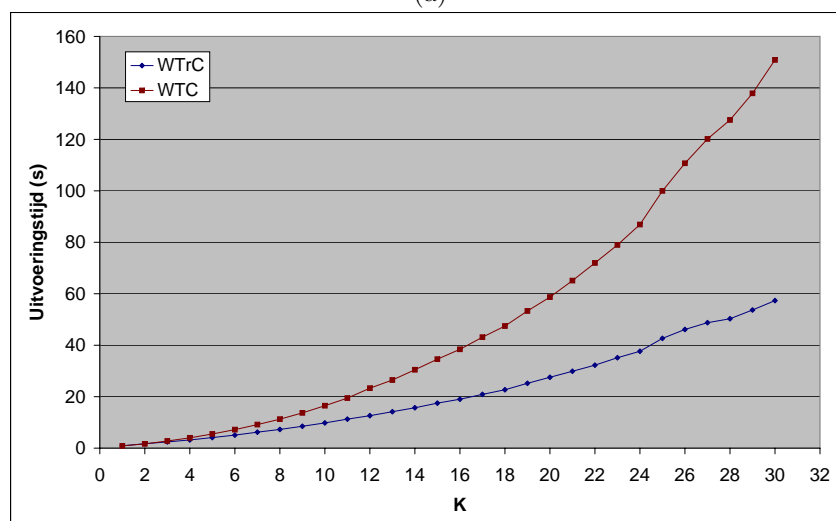
Tabel 5.1: De gemiddelde en maximale totale multicastkost van het gekozen RP ten opzichte van het optimum (%), met een variërend aantal geselecteerde kandidaten K bij de initiële schattingsstap (a) Kleine netwerken (b) Grote netwerken

(a)													
K	WTrC (%)		WTC (%)		Rand (%)		K	WTrC (%)		WTC (%)		Rand (%)	
	Avg	Max	Avg	Max	Avg	Max		Avg	Max	Avg	Max	Avg	Max
1	107	126	107	126	107	126	9	104	126	103	126	121	154
2	105	126	104	126	110	153	10	104	126	103	126	120	162
3	104	126	103	126	110	133	11	104	126	103	126	124	187
4	104	126	103	126	109	134	12	104	120	102	113	125	187
5	104	126	103	126	114	173	13	104	120	102	113	127	196
6	104	126	103	126	113	173	14	104	120	102	113	126	222
7	104	126	103	126	116	151	15	104	120	102	113	132	218
8	104	126	103	126	118	151							
(b)													
K	WTrC (%)		WTC (%)		Rand (%)		K	WTrC (%)		WTC (%)		Rand (%)	
	Avg	Max	Avg	Max	Avg	Max		Avg	Max	Avg	Max	Avg	Max
1	105	117	105	117	105	117	16	105	124	103	115	116	159
2	104	113	104	113	105	119	17	105	124	103	115	115	155
3	104	112	104	113	107	129	18	105	124	103	115	114	147
4	104	116	104	116	109	135	19	105	124	103	115	112	147
5	105	124	104	116	111	138	20	105	124	103	115	114	137
6	106	124	104	116	109	131	21	105	124	103	115	116	184
7	105	124	104	115	108	132	22	105	124	103	115	114	155
8	105	124	103	115	108	138	23	105	124	103	115	115	134
9	105	124	103	115	111	138	24	105	124	103	115	119	156
10	105	124	103	115	112	147	25	105	124	103	115	116	184
11	105	124	103	115	113	137	26	105	124	103	115	120	166
12	105	124	103	115	116	160	27	105	124	102	115	120	156
13	105	124	103	115	115	147	28	105	124	102	115	119	152
14	105	124	103	115	113	140	29	105	124	102	115	121	158
15	105	124	103	115	114	141	30	105	124	102	115	122	161

De eerste twee hoofdkolommen geven het percentage voor de twee RP-selectie technieken Worst Case Multicast Tree Cost (WTrC) en Worst Case Total Multicast Cost (WTC) (zie sectie 4.2.2). De derde kolom toont het resultaat voor een schatter die telkens een willekeurige kandidaat als



(a)



(b)

Figuur 5.2: De uitvoeringstijd van het RP algoritme in functie van het aantal geselecteerde servers K bij initiële selectie (a) Kleine netwerken (b) Grote netwerken

RP uitkiest. De gemiddelde kost neemt maar weinig af bij een stijgend aantal geselecteerde kandidaten K bij initiële selectie. Bij de kleine netwerken is de maximale kost wel lager bij grotere K waarden. Bij netwerken van het grote type is dit bij WTrC niet het geval en is hij zelfs iets hoger bij grotere K . WTC geeft in alle gevallen voor zowel het gemiddelde als het maximum een beter oplossing dan WTrC.

Fig. 5.2 toont de uitvoeringstijd van het RP algoritme voor de twee voorgestelde RP-selectie technieken. Aangezien bij WTC dezelfde boom wordt gezocht als bij WTrC en daarbovenop ook een aantal kortste paden worden berekend, wordt verwacht dat de uitvoeringstijd van WTC

hoger is. Bij de testnetwerken van het grote type is dit duidelijk het geval. Bij de kleine netwerken is het verschil in uitvoeringstijd minimaal. Hieruit kunnen we besluiten dat bij kleine netwerken de tijd om de kortste paden te berekenen verwaarloosbaar is tegenover de tijd om de boom te berekenen. Als het netwerk groter wordt, heeft de pad-berekeningstijd een duidelijke invloed op de uitvoeringstijd (zie Fig. 5.2(b)).

5.1.3 Besluit

Als initiële schattingstechniek raden we het gebruik van MDT aan. Dit geeft veel betere resultaten dan MinMaxD en bezit toch nog een lage tijdscomplexiteit, zodat de techniek ook kan gebruikt worden bij grotere netwerken. De kostwinst bij gebruik van de RP-selectie technieken WTrC en WTC bij grotere K waarden is klein, terwijl de uitvoeringstijd wel sterk toeneemt. Zo geeft een K waarde bij kleine en grote netwerken van respectievelijk 4 en 8, gebruik makend van WTC, een oplossing waarvan de kost maar 3% hoger ligt dan de optimale kost. Voor de maximale K waarde ligt de kost nog 2% hoger dan het optimum. In het laatste geval was de uitvoeringstijd bij kleine en grote netwerken wel respectievelijk meer dan 5 en 10 keer zo groot dan bij K gelijk aan respectievelijk 4 en 8.

Als RP-selectie techniek raden we WTC aan. Aangezien deze zelfs voor kleine K waarden een betere oplossing geeft dan WTrC voor grote K waarden. De uitvoeringstijd van WTC bij kleine K is ook beter dan die van WTrC bij grote K .

5.2 Clustering Algoritme

In dit onderdeel vergelijken we het, in sectie 4.3.2 voorgestelde, MICD algoritme met een aantal bekende clusteralgoritmen. In eerste instantie beschouwen we twee varianten van PAM. We bekijken zowel een standaard PAM implementatie [13] als PAMSIL [23]. Deze laatste geeft in veel gevallen een betere oplossing dan de standaard implementatie.

Vervolgens wordt ook k -medoids beschouwd, een variant op k -means [16]. De stappen van het algoritme zijn gelijk aan die van k -means. Maar k -medoids maakt gebruik van een medoide, in plaats van een gemiddelde, als middelpunt van een cluster. De medoide is het element van de cluster dat zich het dichtst bij het gemiddelde, over alle elementen van de cluster, bevindt. Verder

wordt ook de afstand tussen twee elementen bepaald aan de hand van een dissimilariteitsmatrix, zoals bij PAM.

Als maat voor schaalbaarheid wordt de uitvoeringstijd in functie van het aantal te clusteren elementen gebruikt. De kwaliteit van de oplossingen wordt bepaald door de silhouette coëfficiënt (SC) [23]. De silhouette waarde van een element i wordt gedefinieerd als:

$$s(i) = \frac{\bar{d}_{min}(i) - \bar{d}(i)}{\max\{\bar{d}_{min}(i), \bar{d}(i)\}}$$

Hierbij stelt $\bar{d}(i)$ de gemiddelde dissimilariteit van i tot de andere elementen van dezelfde cluster voor. Verder geldt:

$$\bar{d}_{min}(i) = \min_{C \neq A} \bar{d}(i, C)$$

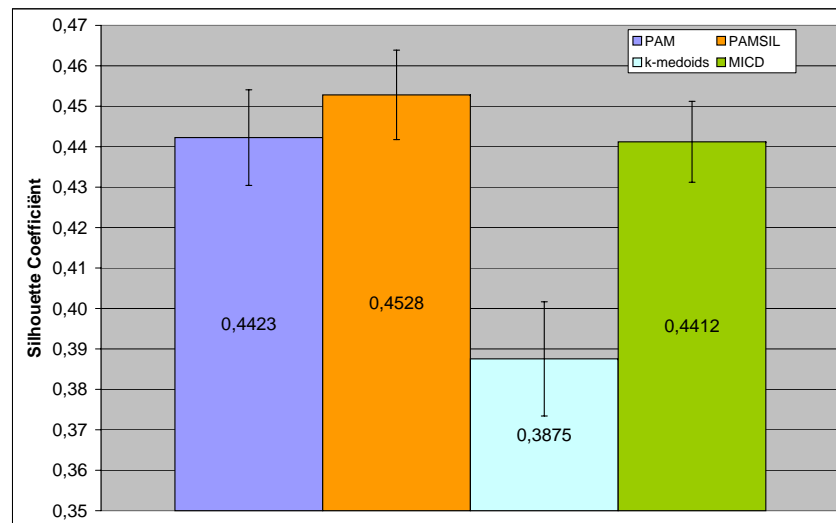
C stelt een willekeurige cluster voor en A de cluster waartoe i behoort. $\bar{d}(i, C)$ is de gemiddelde dissimilariteit tussen i en alle elementen van de cluster C . De SC van een clustering is het gemiddelde van de silhouette waarde van elk te clusteren element. De SC bevindt zich steeds in het interval $[-1, 1]$. Een hoge waarde betekent dat de elementen goed geclusterd zijn, een lage waarde betekent dat de clustering slecht is.

Elk van de algoritmen heeft als input een dissimilariteitsmatrix en het totaal aantal clusters K . Bij het meten van de uitvoeringstijd werd gebruik gemaakt van testnetwerken met 10, 15, 20, 25, 30, 35, 40, 45 en 50 verzenders. Elk van de algoritmen werd steeds uitgevoerd voor alle K in het interval $[1..|S|]$, met $|S|$ het aantal verzenders. De totale uitvoeringstijd is gelijk aan de som van de uitvoeringstijden voor elke mogelijke K .

Voor elk algoritme is een optimale SC berekend, uitgemiddeld over alle 270 testnetwerken (9 types, 30 netwerken per type). De optimale SC is het maximum van de SC's voor $K = 1..|S|$.

Fig. 5.3 toont voor ieder getest algoritme de optimale SC. Zoals verwacht geeft PAMSIL de beste oplossing en ligt de SC van k -medoids beduidend lager dan die van de twee PAM implementaties. MICD presteert ongeveer even goed als standaard PAM.

Fig. 5.4 toont per algoritme de gemiddelde uitvoeringstijd in functie van het aantal verzenders. In Fig. 5.4(a) is duidelijk te zien dat de uitvoeringstijd van k -medoids en MICD te verwaarlozen



Figuur 5.3: De optimale SC voor de verschillende algoritmen. Gemiddelden \pm de standaardfout

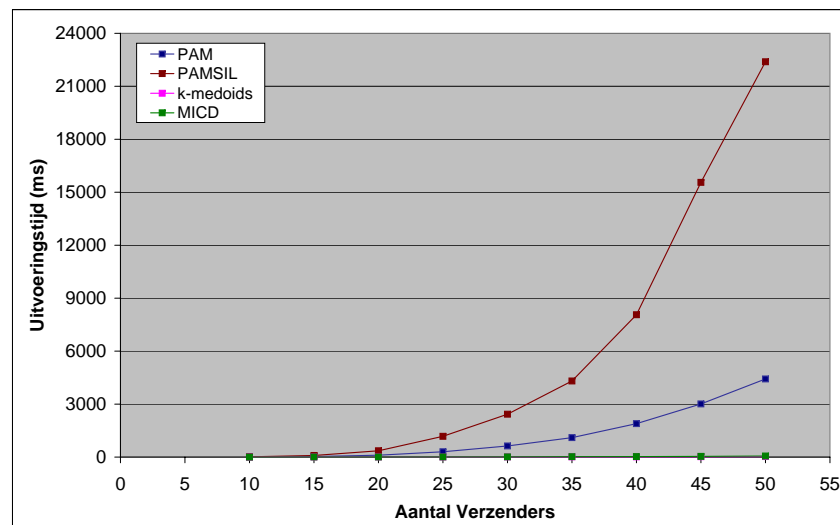
is in vergelijking met die van de PAM implementaties. De figuur toont ook dat PAMSIL een hogere uitvoeringstijd heeft dan standaard PAM.

In Fig. 5.4(b) worden enkel k -medoids en MICD met elkaar vergeleken. De uitvoeringstijd van MICD stijgt sneller bij een toenemend aantal verzenders.

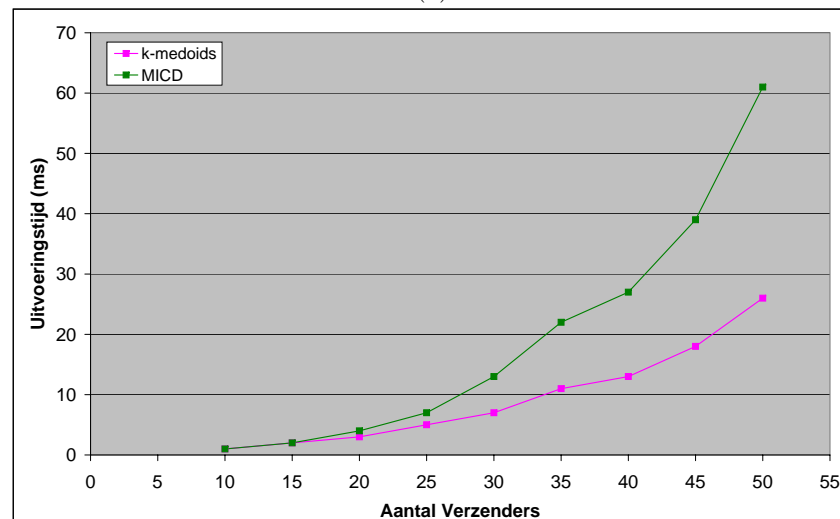
Zowel PAM als k -medoids hebben duidelijk hun beperkingen. Zo zijn de PAM implementaties weinig schaalbaar, door hun sterk toenemende uitvoeringstijd bij een groter aantal verzenders. De kwaliteit van de oplossing bij k -medoids is dan weer niet altijd naar behoren. MICD biedt hier een goede tussenweg. Het heeft een hogere uitvoeringstijd dan k -medoids, maar is nog altijd bruikbaar bij een groter aantal verzenders. Daarbovenop heeft de oplossing van MICD over het algemeen een goede SC, met een gemiddelde dat ongeveer even hoog ligt als bij standaard PAM.

5.3 Multicastboom Algoritme

In deze sectie worden de padselectie en clustering stap van het multicastboom algoritme geëvalueerd. Aan de hand van de tests worden de kwaliteit van de oplossing, de schaalbaarheid en de invloed van de parameters erop, geëvalueerd. Tenslotte wordt ook het hele algoritme onderworpen aan een aantal tests. De boomcreatie stap van het algoritme op zich wordt niet getest. Deze stap steunt volledig op een algoritme om DCMST problemen mee op te lossen. In sectie 4.3.3 werd voorgesteld om hiervoor een bestaand algoritme te gebruiken. Het heeft dan



(a)



(b)

Figuur 5.4: De gemiddelde uitvoeringstijd in functie van het aantal verzenders (a) Vergelijking van alle algoritmen (b) Vergelijking van k -medoids en MICD

ook niet veel zin om dat opnieuw te evalueren.

5.3.1 Pad Selectie

Voor deze test werd net zoals bij het RP algoritme gebruik gemaakt van testnetwerken van twee verschillende groottes. De netwerken van het kleine type hebben 50 IP nodes, 15 overlay servers en 5 verzenders. De netwerken van het grote type hebben 100 IP nodes, 30 overlay servers en 10 verzenders. Het aantal ontvangers is niet van belang, aangezien bij het zoeken van de paden

Tabel 5.2: Het aantal paden per verzender dat overblijft na het uitvoeren van de pad selectie criteria
(a) Kleine netwerken (b) Grote netwerken

(a)	Aantal Paden	Aantal Verzendders	Percentage (%)	Cummulatief Percentage (%)
	1	116	86,5	86,5
	2	16	12,0	98,5
	3	2	1,5	100,0
(b)	Aantal Paden	Aantal Verzendders	Percentage (%)	Cummulatief Percentage (%)
	1	218	74,1	74,1
	2	57	19,3	93,4
	3	13	4,4	97,8
	4	5	1,7	99,5
	5	1	0,5	100,0

hiermee geen rekening wordt gehouden.

Het belangrijkste onderdeel van deze stap van het algoritme zijn de padselectie criteria. Deze criteria hebben twee verschillende doelstellingen.

Ten eerste worden ze gebruikt om het aantal mogelijke paden van elke verzender naar het RP te beperken. Bij grote netwerken kunnen er namelijk zeer veel paden gevonden worden, zelfs als er een maximale delay grens is. Aangezien het aantal RP-bomen die berekend moeten worden in de boomcreatie stap afhangt van het aantal gevonden paden, kan hierdoor de totale rekentijd sterk worden verminderd.

Ten tweede is er ook nog het probleem van het zoeken van alle paden tussen twee nodes in een graaf. In sectie 4.3.1 werd het algoritme van Yen voorgesteld om dit probleem op te lossen. Hierbij moet een parameter K worden opgegeven. Deze staat voor het totaal aantal te zoeken paden. Als K kan worden ingesteld op een lage waarde, zonder veel mogelijk optimale paden te verliezen, kan de tijdscomplexiteit van deze stap worden beperkt zonder een slechtere oplossing te krijgen. Als dus blijkt dat de padselectie criteria alle paden boven een bepaalde index schrappen, kan K op deze index worden ingesteld. Deze criteria schrappen namelijk enkel paden die niet tot de optimale oplossing kunnen leiden.

Tabel 5.2 toont het aantal paden dat overbleef na het uitvoeren van de padselectie criteria en voor hoeveel verzenders dit het geval was. Resultaten waar de verzender het RP is werden niet

Tabel 5.3: De indices van de geselecteerde paden en het aantal paden dat per index werd geselecteerd
 (a) Kleine netwerken (b) Grote netwerken

(a)		Index Aantal Paden		Index Aantal Paden	
	1	134	6	1	
	2	9	7	2	
	3	5	9	2	
	5	1			

(b)		Index Aantal Paden		Index Aantal Paden		Index Aantal Paden	
	1	291	9	1	23	1	
	2	31	11	4	31	1	
	3	18	12	1	34	1	
	4	10	13	1	65	1	
	5	11	14	2	119	1	
	6	8	15	1	123	1	
	7	3	19	1	166	1	
	8	3	21	2	184	1	

meegeteld omdat het enige geldige pad dat van de verzender naar zichzelf is. Daarom zijn er in totaal voor kleine en grote netwerken respectievelijk 134 en 294 verzenders in plaats van 150 en 300. Bij de kleine netwerken werden er gemiddeld per verzender 3 paden gevonden. Na padselectie bleven er nog 1,15 over. Bij de grote werden er gemiddeld 19 gevonden, waarvan er maar 1,35 overbleven. De resultaten worden dus duidelijk beter bij grotere netwerken. In bijna elk geval bleven er bij netwerken van het kleine type maar 1 of 2 paden over, bij grote is dit 1, 2 of 3. In een paar gevallen werden er bij grote netwerken 4 of 5 paden geselecteerd. Maar zelfs dit maximum is nog veel lager dan het gemiddelde aantal voor pad selectie.

Tabel 5.3 toont de indices van de paden die overbleven na padselectie en het aantal keer dat een pad met elke index geselecteerd werd. De index van een pad is het volgnummer ervan in de lijst die door het algoritme van Yen wordt teruggegeven. De paden in deze lijst zijn gesorteerd volgens stijgende delay. Bij de testnetwerken van het kleine type werd maar in 3,9% van de gevallen een pad met index groter dan 3 geselecteerd. Als de grens van de parameter K van het algoritme van Yen dan op 3 zou worden ingesteld gaat maar een klein percentage van de mogelijk optimale paden verloren. Bij netwerken van het grote type had maar 5,3% van de

geselecteerde paden een index groter dan 8.

5.3.2 Clustering

In sectie 5.2 is de schaalbaarheid en kwaliteit van de oplossing van het MICD algoritme reeds vergeleken met die van een aantal veelgebruikte cluster algoritmen. Nu wordt de invloed van het MICD algoritme op het multicastboom algoritme besproken. Er werden weer 2 types van netwerken gebruikt. Het kleine type heeft 100 IP nodes, 20 overlay servers, 7 verzenders en 7 ontvangers. Het grote type 150 IP nodes, 30 overlay servers, 10 verzenders en 10 ontvangers.

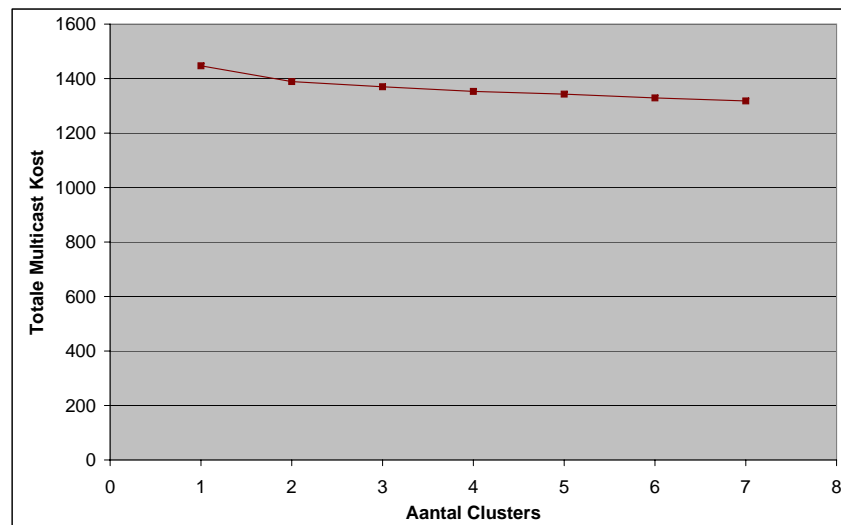
Bij het clusteren wordt als parameter het aantal clusters K opgegeven. Deze kan waarden aannemen tussen 1 en het aantal verzenders $|S|$. Alle verzenders in dezelfde cluster zullen een RP-boom delen. Bij $K = 1$ bevinden alle verzenders zich in één cluster en wordt er dus gebruik gemaakt van één enkele RP-boom. Bij $K = |S|$ heeft elke verzender zijn eigen boom. Het doel van deze tests is nu de invloed van de parameter K op de uitvoeringstijd van het multicastboom algoritme en de totale multicast kost van de oplossing te bestuderen.

Aangezien er bij een kleinere K waarde minder bomen moeten berekend worden, wordt verwacht dat de uitvoeringstijd zal stijgen bij een toenemend aantal clusters. Anderzijds delen meer verzenders eenzelfde boom en krijgen ze er dus mogelijk één toegewezen met een sub-optimale kost. De kost zou dus normaal gezien moeten dalen bij toenemende K .

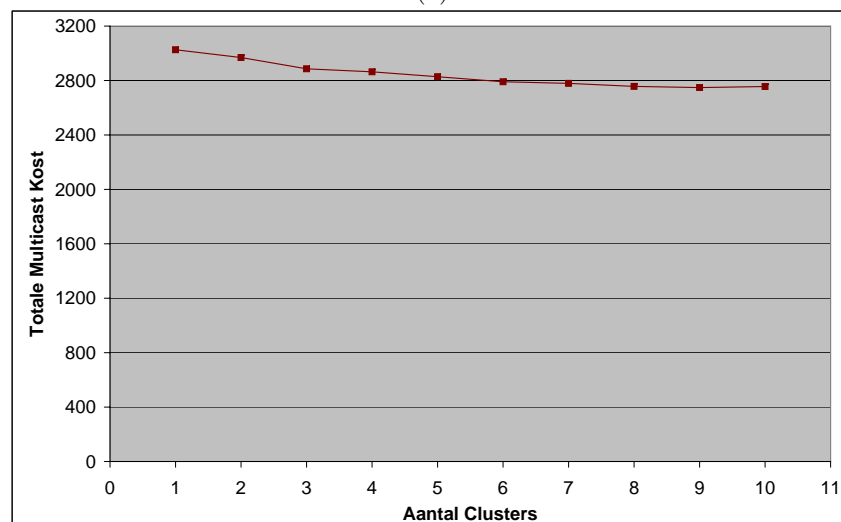
Fig. 5.5 toont de totale kost van de oplossing in functie van K . Zoals verwacht daalt de kost als K toeneemt. In Fig. 5.6 zien we dat de uitvoeringstijd toeneemt als K stijgt. Dit is ook wat werd verwacht. Het interessante aan dit resultaat is dat de kost weinig daalt, terwijl de uitvoeringstijd sterk toeneemt. Bij een laag aantal clusters geeft de clustering stap een grote tijds winst met maar een beperkte kosttoename als gevolg.

5.3.3 Volledig Algoritme

In de laatste test wordt de oplossing van het multicastboom algoritme vergeleken met de globaal optimale oplossing. Het globaal optimum wordt berekend met behulp van het algoritme beschreven in sectie 4.4. Er werden weer twee types van testnetwerken gebruikt. De kleine netwerken hebben 30 IP nodes, 15 overlay servers, 5 verzenders en 5 ontvangers. De grote netwerken hebben 60 IP nodes, 30 overlay servers, 10 verzenders en 10 ontvangers.



(a)

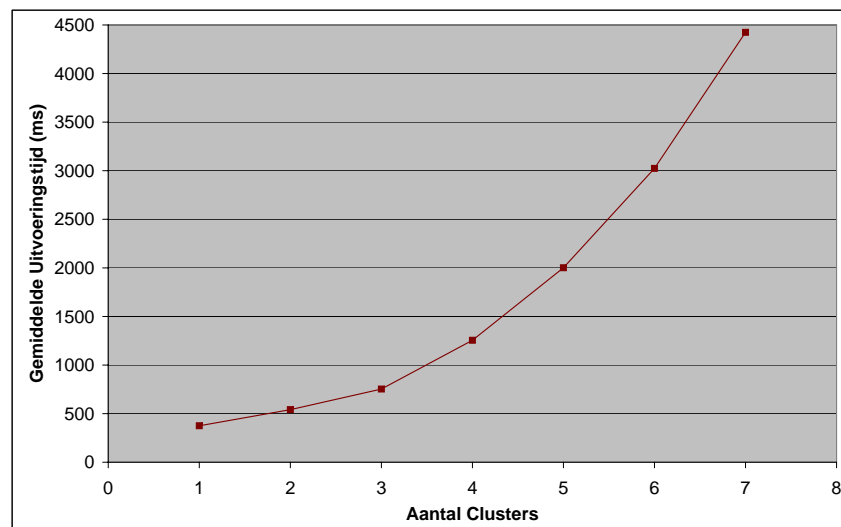


(b)

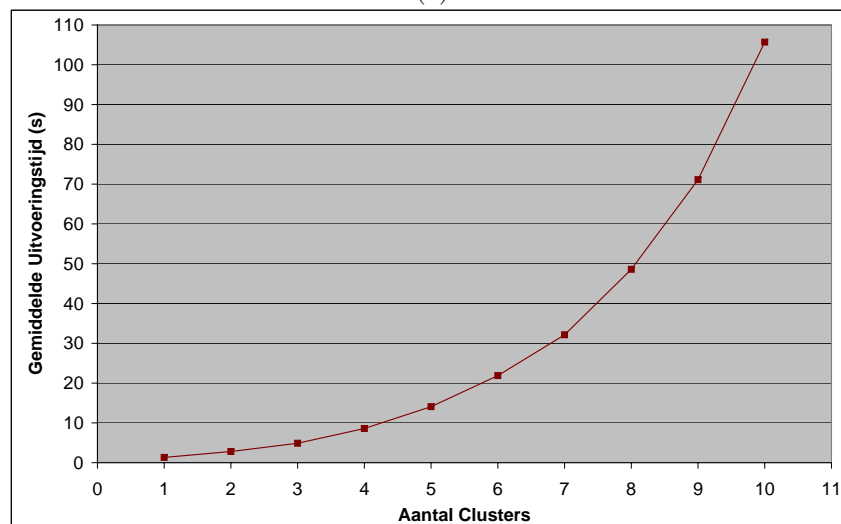
Figuur 5.5: De totale multicast kost van de oplossing in functie van het aantal clusters K (a) Kleine netwerken (b) Grote netwerken

In de boomcreatie stap van het multicastboom algoritme werd net zoals bij het exact algoritme (zie sectie 4.4) gebruik gemaakt van het ILP algoritme. Hierdoor wordt enkel de kostverhoging geïntroduceerd door de padselectie en clustering stappen in rekening gebracht. Aangezien het ILP algoritme weinig schaalbaar is, zal in realiteit steeds een heuristisch gebruikt worden voor het berekenen van de RP-bomen. De kost van de oplossing zal dan hoger zijn dan bij de hier besproken testresultaten.

Om het kostverlies veroorzaakt door het RP algoritme op de uiteindelijke oplossing na te gaan,



(a)



(b)

Figuur 5.6: De gemiddelde uitvoeringstijd van het multicastboom algoritme in functie van het aantal clusters K (a) Kleine netwerken (b) Grote netwerken

werd de test zowel uitgevoerd met RP selectie, als zonder. Zonder RP selectie werd het optimale RP (datgene dat de beste oplossing geeft bij het exact algoritme) ook bij het multicastboom algoritme gebruikt. Met RP selectie werd het RP algoritme gebruikt om er één te selecteren. Bij het RP algoritme werd de parameter K van de initiële schatting gelijk gesteld aan het aantal overlay servers $|O|$, wat erop neer komt dat de initiële schattingsstap werd weggelaten.

Bij de padselectie stap werd het aantal te selecteren paden door het algoritme van Yen eveneens ingesteld op $|O|$ (zie sectie 5.3.1).

Tabel 5.4: De totale kost van de oplossing van de multicastboom heuristisch tegenover de kost van de optimale oplossing (in %), in functie van het aantal clusters (a) Kleine netwerken (b) Grote netwerken

(a)	Clusters	Optimaal RP (%)		RP Selectie (%)	
		Avg	Max	Avg	Max
	1	106,21	127,66	106,37	121,28
	2	102,39	126,99	103,38	118,14
	3	101,05	109,41	102,69	118,14
	4	100,38	104,62	102,34	113,38
	5	100,15	104,62	102,08	113,38

(b)	Clusters	Optimaal RP (%)		RP Selectie (%)	
		Avg	Max	Avg	Max
	1	111,28	129,48	111,28	124,64
	2	105,89	114,74	108,02	124,64
	3	103,36	108,68	105,77	124,64
	4	102,61	109,99	104,91	120,54
	5	102,03	107,12	104,59	118,74
	6	101,46	107,12	103,83	118,63
	7	100,89	105,15	103,45	117,62
	8	100,64	103,87	103,25	116,95
	9	100,46	103,87	102,92	116,55
	10	100,06	101,61	102,57	115,49

De resultaten in Tabel 5.4 bevestigen de resultaten van de tests van de afzonderlijke stappen van het algoritme. Net zoals de resultaten in sectie 5.3.2 aantoonen zal bij een stijgend aantal clusters de totale kost dalen. Opmerkelijk is dat het kostverschil tussen 1 en 2 clusters groot is, maar vanaf dan de kost maar weinig afneemt.

De eerste kolom van de laatste rij toont de gemiddelde kost voor het maximaal aantal clusters zonder RP selectie. Het enige verlies in dit resultaat is dus datgene veroorzaakt door de padselectie stap. Aangezien dit resultaat voor zowel kleine als grote netwerken dicht aanleunt bij 100% veroorzaakt deze stap nauwelijks verlies.

De gemiddelde kost bij RP selectie ligt maximaal maar 3 procent hoger dan de kost voor het optimale RP. Bij een groter aantal clusters is de maximale kost daarentegen wel veel hoger met RP selectie. Voor grote netwerken is dit voor 10 clusters zelfs bijna 14%.

5.3.4 Besluit

Uit de verschillende tests blijkt dat het verlies veroorzaakt door de padselectie stap verwaarloosbaar klein is als het aantal te zoeken paden (de parameter K uit het algoritme van Yen) voldoende groot gekozen wordt. In de test beschreven in sectie 5.3.3 was het gemiddelde verlies veroorzaakt door deze stap voor $K = |O|$ niet groter dan 0,15%.

Verder is het ook zo dat de kost weinig daalt bij een stijgend aantal clusters, terwijl de totale uitvoeringstijd veel toeneemt. In de meeste toepassingen is het dan ook aan te raden het aantal clusters te beperken.

Hoofdstuk 6

Verder Onderzoek

Hoewel de geïmplementeerde architectuur voldoet aan de basisvereisten, is het mogelijk om de functionaliteit uit te breiden. Het heeft niet veel zin om hier een exhaustieve opsomming te maken. We bespreken dan ook enkel een aantal van de, volgens ons, belangrijkere mogelijke uitbreidingen.

6.1 Online Compileren van Plugins

Op dit moment zijn er een aantal beperkingen verbonden aan het gebruik van server side plugins. Ze moeten namelijk door de gebruiker worden gecompileerd, voordat ze worden geupload. Als een systeemafhankelijke taal (zoals Java) wordt gebruikt, vormt dit geen probleem. In andere gevallen, zoals bijvoorbeeld wanneer de plugin is geschreven in C of C++, moet hij worden gecompileerd op een computer die gebruik maakt van hetzelfde besturingssysteem en dezelfde architectuur als de overlay servers. Als verschillende overlay servers gebruik maken van een verschillend besturingssysteem of draaien op een computer met een andere architectuur, geeft dit nog grotere problemen. In dat geval is het zelfs onmogelijk om plugins te gebruiken die geschreven zijn in een systeemafhankelijke taal.

Een mogelijk oplossing is om gebruikers de broncode van de plugins te laten uploaden, in plaats van een gecompileerde versie. Elke overlay server kan dan, met een geschikte compiler, de broncode omzetten in een uitvoerbaar programma voor zijn besturingssysteem en architectuur.

6.2 Beveiliging

Bij de implementatie van de OMA werd er weinig aandacht besteed aan beveiliging (security). In de huidige implementatie zijn de volgende beveiligingsmechanismen aanwezig:

- **Encryptie van wachtwoorden:** In de databank worden enkel de geëncrypteerde versies van de gebruikerswachtwoorden bijgehouden. In het geval dat de databankgegevens uitlekken, kan hieruit geen geldige login-informatie worden afgeleid.
- **Toegangsrechten:** Bepaalde acties kunnen enkel door gebruikers met beheerdersrechten worden uitgevoerd. Als een gebruiker via de Web Interface of het OMP een actie probeert uit te voeren, worden steeds zijn logingegevens meegestuurd naar het DMP. Daar wordt dan gecontroleerd of de gebruiker gemachtigd is om de actie uit te voeren.

De architectuur bevat nog een aantal mogelijke onveiligheden. Extra beveiligingsmechanismen die kunnen worden toegevoegd zijn:

- De communicatie tussen de verschillende componenten van de architectuur en gebruikers wordt niet geëncrypteerd. Zeker het versturen van wachtwoorden als platte tekst is een groot veiligheidsprobleem. Bij de communicatie tussen de gebruikers en de Web Interface kan bijvoorbeeld gebruik worden gemaakt van Secure HTTP (S-HTTP) [19] of het Transport Layer Security (TLS) Protocol [8], om hun logingegevens te beschermen.
- Aangezien elke gebruiker nieuwe server side plugins kan uploaden kan niet worden voorkomen dat deze, al dan niet opzettelijk, schadelijke code bevatten. Om de overlay servers te beschermen wanneer ze plugins opstarten kan er bijvoorbeeld voor worden gezorgd dat de plugins geen toegang krijgen tot het onderliggende besturingssysteem (zoals het bestandssysteem, of het uitvoeren van systeemcommando's). Ook kan hun toegang tot systeembronnen, zoals het geheugen en de CPU, worden beperkt om te voorkomen dat ze deze monopoliseren.

6.3 Dynamische Multicastbomen

De Rendezvous Point en Multicastboom algoritmen zijn op dit moment volledig statisch. Als de netwerktopologie verandert, moeten de algoritmen opnieuw worden uitgevoerd om aangepaste

multicastbomen te verkrijgen. Als er een nieuwe overlay server aan het netwerk wordt toegevoegd, moeten de multicastbomen niet worden aangepast aangezien ze nog steeds geldig zijn in het nieuwe netwerk. Als er een overlay server verdwijnt moeten alle multicastbomen die deze server bevatten wel worden herberekend.

In de huidige implementatie wordt bij het verdwijnen van een overlay server automatisch voor elke multicastsessie een nieuw RP gekozen en worden alle multicastbomen herberekend. In theorie is dit enkel nodig bij sessies die de verdwenen server als RP hebben. Met een gepast algoritme kan in de multicastbomen van de andere sessies de verdwenen server worden vervangen door één of meerdere andere overlay servers. Voor deze sessies moeten dan maar een paar overlay servers worden geherconfigureerd, waardoor mogelijk veel onnodige reken- en herconfiguratietijd wordt uitgespaard.

Hoofdstuk 7

Conclusies

Het internet heeft, als medium voor multimedietoepassingen, een aantal limitaties waaronder een beperkte ondersteuning voor multicasting en het gebrek aan QoS garanties. Met behulp van een overlay netwerk kunnen many-to-many multicastdiensten worden aangeboden, waardoor de beschikbare bandbreedte veel efficiënter kan gebruikt worden. Doordat het voor de bestaande netwerkinfrastructuur steeds moeilijker is om te voldoen aan de stijgende vraag naar bandbreedte, zal dit steeds belangrijker worden. Het overlay netwerk laat ook toe om, door op een intelligente manier de multicast distributiebomen te kiezen, de gebruikte bandbreedte te minimaliseren en garanties te bieden in verband met de QoS (bijvoorbeeld de maximale end-to-end delay).

In dit boek werd een Overlay Multicast Architectuur voorgesteld en ontworpen. De componenten van de architectuur zijn in staat om in functie van de noden van de gebruikers het overlay netwerk te configureren voor many-to-many multicastsessies. Hierbij wordt zowel rekening gehouden met het efficiënt gebruik van de beschikbare bandbreedte als met de maximale end-to-end delay. Gebruikers kunnen informatie over multicastsessies ingeven via een gebruiksvriendelijke Web Interface. Het Data Management Platform zorgt ervoor dat deze informatie persistent wordt bijgehouden in een databank. Het Overlay Management Platform gebruikt ze vervolgens om de Overlay Servers te configureren met behulp van routeringsregels. De Overlay Servers zelf gebruiken deze regels om de data te routeren doorheen het netwerk.

Door gebruik te maken van server side plugins zijn de overlay servers in staat om de gemulticaste datapakketten te verwerken voordat ze worden doorgestuurd naar de ontvangers. Het platform biedt dus zowel ondersteuning voor multimedietoepassingen die enkel nood hebben aan

een eenvoudig routeringsplatform dat QoS garanties biedt als voor toepassingen waarbij meer ingewikkelde pakketverwerking nodig is.

Om de routeringsperformantie van de geïmplementeerde overlay servers te meten werden er, met behulp van een Smartbits machine, een aantal tests op uitgevoerd. Hieruit bleek dat de gemiddelde delay recht evenredig is met de pakketgrootte en het aantal keer dat een pakket moet worden gedupliceerd. Het totaal aantal routeringsregels in de regellijst van een overlay server had dan weer geen invloed op de delay. Over het algemeen bleken de overlay servers in staat om bij een pakketgrootte van 128 byte ongeveer 100 Mbps te verwerken (± 100000 pakketten per seconden). Hierbij veroorzaakten ze een gemiddelde vertraging van 57 microseconden.

Voor het berekenen van de multicast distributiebomen werden een aantal algoritmen ontworpen. Er werd geopteerd voor een Rendezvous Point (RP) gebaseerde aanpak. Elke multicastsessie heeft dan een centraal punt in het netwerk, dat onder andere kan gebruikt worden voor datapakket bewerking. Het voorgestelde Rendezvous Point Algoritme is een heuristiek die een RP kiest waarvoor de totale kost van de multicast distributiebomen zo laag mogelijk is. Hierbij wordt een suboptimale oplossing gezocht om de uitvoeringstijd van het algoritme, voor dit NP-compleet probleem, te beperken.

Het Multicastboom Algoritme probeert op een efficiënte manier gebruik te maken van de beschikbare bandbreedte door een algemene takkost te minimaliseren over de multicast distributiebomen. Om de QoS te verbeteren wordt een delay bound gebruikt om de maximale end-to-end delay binnen bepaalde grenzen te houden.

De ontworpen algoritmen beschikken over een aantal vrij in te stellen parameters. Met behulp van deze parameters kan een afweging worden gemaakt tussen een lagere kost oplossing enerzijds en een kortere uitvoeringstijd anderzijds. Door, bij het multicastboom algoritme, gebruik te maken van padselectie criteria en een clustering stap kan met een minimale koststijging een grote tijds winst worden geboekt. Met behulp van deze technieken kan ook voor grote netwerken, binnen afzienbare tijd, een goede oplossing worden bekomen.

Bij de uitgevoerde tests bleven na de padselectie stap van het multicastboom algoritme bij meer dan 90% van de verzenders slechts 1 of 2 paden over. Hierdoor moet het algoritme voor bijna alle verzenders slechts 1 of 2 RP-bomen berekenen. Bij het maximaal aantal clusters was de totale kost van de multicastbomen minder dan 9% lager dan bij 1 cluster, terwijl de uitvoeringstijd tot meer dan 100 keer groter was. Door het aantal clusters laag te houden kan dus met een

bepaalde koststijging een grote tijdsinstorting geboekt worden. De totale kost van de multicast distributiebomen was, voor 3 of meer clusters, niet meer dan 6% hoger dan de totale kost van de optimale bomen.

Voor de clustering stap van het Multicastboom Algoritme werd het MICD clusteralgoritme ontworpen. Bij de uitgevoerde tests was de uitvoeringstijd vergelijkbaar met die van het schaalbare k -means algoritme. De Silhouette Coëfficiënt, die de kwaliteit van de bekomen clustering aangeeft, was vergelijkbaar met die van PAM, een weinig schaalbaar algoritme dat een oplossing van hogere kwaliteit geeft.

In dit boek werd aangetoond dat met behulp van Overlay Multicasting een aantal van de beperkingen van IP Multicasting (zoals geen end-to-end ondersteuning op het internet en gebrek QoS garanties) kunnen worden overwonnen. Het OMP configureert het overlay netwerk automatisch in functie van de noden van de gebruikers. Door gebruik te maken van plugins, worden zowel multimedietoepassingen die enkel nood hebben aan een routeringsplatform als toepassingen die gebruik maken van een serverside component ondersteund. Doordat de ontworpen algoritmen het RP en de multicast distributiebomen op een intelligente manier kiezen, worden het bandbreedtegebruik en de maximale end-to-end delay verbeterd.

Bijlage A

Virtual Meeting Place

De Virtual Meeting Place (VMP) client-server applicatie werd ontworpen om de Overlay Multicast Architectuur (zie hoofdstuk 3), in zijn geheel, te testen. De applicatie laat gebruikers toe om zich in een virtuele ruimte te verplaatsen en tekstboodschappen naar elkaar te sturen. We beginnen de bespreking met een korte beschrijving van het gebruikte communicatieprotocol. Vervolgens worden de client applicatie en de serverside plugin besproken.

A.1 Protocol

Het VMP protocol bestaat uit variabele lengte boodschappen. De eerste byte geeft het type van de boodschap aan. Elke boodschap bevat ook één of meerdere parameters. Het aantal parameters en het type ervan, zijn bij een bepaald type boodschap steeds gelijk. De verschillende parametertypes zijn:

- **Byte:** Een geheel getal bestaande uit 8 bits.
- **Integer:** Een geheel getal bestaande uit 4 bytes. De eerste byte is de meest significante, de laatste de minst significante (Big Endian).
- **String:** Een variabel aantal karakters, gecodeerd in UTF-8 formaat [26].
- **Color:** Een kleur wordt voorgesteld door 4 bytes. De eerste drie stellen de R(ed), G reen) en B(blue) componenten voor, de laatste is de alpha waarde.

De verschillende types van boodschappen en hun parameters zijn:

- **Connect (CON):**

byte: 1	string: <naam>	color: <avatar_kleur>
---------	----------------	-----------------------

Deze boodschap wordt verstuurd door een gebruiker die wil inloggen met de opgegeven naam. De <avatar_kleur> parameter geeft de kleur aan van het karakter dat hij zal besturen (de avatar).

- **Accept (ACC):**

byte: 2	string: <naam>	color: <avatar_kleur>
---------	----------------	-----------------------

Als de server een CON boodschap ontvangt en er nog geen client met de opgegeven naam online is, dan stuurt hij een ACC boodschap als antwoord naar alle clients. De twee parameters zijn dezelfde als die van de ontvangen CON boodschap.

- **Refuse (REF):**

byte: 3	string: <naam>
---------	----------------

Als de server een CON boodschap ontvangt en er is reeds een client online met de opgegeven naam, dan antwoordt hij met een REF boodschap.

- **Disconnect (DIS):**

byte: 4	string: <naam>
---------	----------------

Als een client offline gaat stuurt hij een DIS boodschap met zijn eigen naam als parameter. Als een server geen boodschappen ontvangt van een bepaalde client binnen een afgesproken interval, neemt hij aan dat de client offline is en stuurt hij zelf een DIS boodschap, met de client zijn naam.

- **TextMessage (TXT):**

byte: 5	string: <naam>	string: <boodschap>
---------	----------------	---------------------

Deze boodschap stelt een tekstboodschap van een client voor. De eerste parameter geeft de naam van de client aan, de tweede de tekstboodschap zelf.

- **Move (MOV):**

byte: 6	string: <naam>	int: <x>	int: <y>
---------	----------------	----------	----------

Als een client zijn avatar van positie laat veranderen wordt dit aan de andere clients meegedeeld met behulp van een MOV boodschap. De <x> en <y> parameters geven het aantal pixels dat de avatar werd verplaatst in horizontale en verticale richting.

- **Set (SET):**

byte: 7	string: <naam>	color: <avatar.kleur>	int: <links>	int: <top>
---------	----------------	-----------------------	--------------	------------

Op een vast interval stuurt elke client een SET boodschap. Deze geeft de kleur en exacte locatie van zijn avatar aan. Het sturen van deze boodschap heeft een aantal redenen:

- Als de server na een bepaalde tijd geen boodschap van de client heeft ontvangen neemt hij aan dat deze offline is gegaan. Door automatisch SET boodschappen te sturen, zal de server niet denken dat de client offline is als hij geen tekstboodschappen stuurt en zijn avatar niet beweegt.
- Het onderliggende protocol staat niet vast en is mogelijk onbetrouwbaar (bijvoorbeeld UDP). In dat geval kunnen MOV boodschappen verloren gaan. Met behulp van de SET boodschappen kunnen clients toch met elkaar synchroniseren.
- Als een nieuwe client online komt, kent hij de kleur en positie van de avatars van de andere clients niet. Doordat elke client SET boodschappen verstuurt, kan hij die snel te weten komen.

Aangezien de communicatie met en tussen de overlay servers via het UDP protocol verloopt, worden de VMP boodschappen in UDP pakketten geplaatst en verstuurd. Het VMP protocol is onafhankelijk van het onderliggende transportlaag-protocol en er zou, afhankelijk van de toepassing, ook bijvoorbeeld TCP kunnen worden gebruikt.

A.2 Client

De client werd geïmplementeerd als een eenvoudige stand-alone Java applicatie. Bij het opstarten moet de gebruiker een doel IP adres en poort opgeven. Om de applicatie te gebruiken in combinatie met de OMA zijn deze gelijk aan het IP van zijn dichtstbijzijnde overlay server en de poort van de sessie. Hij moet ook de poort opgeven waarop de applicatie UDP pakketten zal ontvangen. Dit is dezelfde poort als degene die de gebruiker opgeeft als hij zich registreert als ontvanger voor de multicastsessie.

De client stuurt vervolgens een CON boodschap. De server zal dan antwoorden met een ACC of een REF. Als de client een ACC ontvangt is alles in orde en logt hij in. Als hij een REF ontvangt geeft de client de gebruiker de mogelijkheid om een andere naam te kiezen. Als hij geen

antwoord krijgt, neemt de client aan dat de server offline is, of dat de gebruiker een verkeerd adres heeft opgegeven en krijgt hij de kans om een nieuw bron- en doeladres in te vullen.

De GUI toont de virtuele ruimte waarin de avatars van de clients kunnen rondbewegen. Onderaan worden de tekstboodschappen getoond die gebruikers naar elkaar sturen. Als de gebruiker zijn avatar verplaatst (met behulp van de pijltjes toetsen), stuurt de client een MOV boodschap naar de server. Als hij een tekstboodschap ingeeft wordt een TXT boodschap gestuurd. Daarbovenop stuurt de client op een vast interval (in de huidige implementatie elke 3 seconden) een SET boodschap naar de server.

Als de client wordt afgesloten stuurt hij automatisch een DIS boodschap naar de server. Als deze DIS boodschap verloren gaat, of de client slaagt er niet in om ze te versturen (bijvoorbeeld bij een crash), dan zal de server dit detecteren als hij een aantal seconden geen SET (of andere) boodschap meer van de client heeft ontvangen.

A.3 Server

De server werd net als de client geïmplementeerd als een Java applicatie. Aangezien de server door de OMA wordt opgestart als plugin, voldoet hij aan de eisen voor serverside plugins (zie sectie 3.1). De server houdt een lijst bij van gebruikers en informatie over hun avatars (hun kleur en locatie). Daarbovenop vervult hij nog een aantal taken:

- Als een client een CON boodschap verstuurt, onderschepst de server deze. Als de opgegeven naam nog niet in gebruik is, wordt de informatie over de client toegevoegd aan de lijst en stuurt de server een ACC bericht. De client van wie het CON bericht komt, weet dan dat hij toegelaten is, de andere clients weten door de ACC dat er een nieuwe gebruiker online is. Als de naam al in gebruik is, antwoordt de server met een REF bericht.
- Voor elke client houdt de server bij, wanneer hij er laatst een boodschap van heeft ontvangen. Als dit langer dan een bepaald aantal seconden geleden is (in de huidige implementatie 9), dan neemt hij aan dat de client niet meer online is. Hij zal hem dan verwijderen uit zijn lijst en een DIS bericht naar de andere clients sturen.

Verder zal de server elk bericht dat hij ontvangt verder doorsturen naar de clients (behalve het CON bericht, waarbij hij het antwoord verderstuurt). Bij het ontvangen van een MOV of SET

bericht zal hij daarbovenop de locatie van de gepaste avatar updaten. Bij het ontvangen van een DIS bericht verwijdert hij de client in kwestie uit zijn lijst.

De server kan mogelijk worden uitgebreid om nog een aantal andere taken te vervullen. Zo zou hij bijvoorbeeld SET berichten kunnen onderscheppen en op een vast interval een gecombineerd SET bericht (met de locatie van elke avatar) kunnen doorsturen naar elke client.

Bijlage B

Inhoud CD-ROM

Bij de ingebonden versie van het thesisboek werd een CD-ROM gevoegd. De inhoud daarvan wordt hier kort beschreven:

- `/thesisboek.pdf`:
Een digitale versie van dit thesisboek.
- `/extended_abstract.pdf`:
Een digitale versie van de extended abstract afzonderlijk.
- `/tex/thesisboek/`:
De \LaTeX sources van het thesisboek en alle bestanden die nodig zijn om het boek te compileren (zoals afbeeldingen en bibtex bestanden).
- `/tex/extended_abstract/`:
De \LaTeX sources van de extended abstract en de nodige extra bestanden.
- `/src/`:
De Java sourcecode van de componenten van de architectuur (`/src/architectuur/`), de algoritmen (`/src/algoritmen/`) en de Virtual Meeting Place applicatie (`/src/vmp/`).
- `/netbeans/`:
De netbeans projecten van de architectuur (`/netbeans/architectuur/`) en de Virtual Meeting Place applicatie (`/netbeans/vmp/`). De projecten van de verschillende componenten van de architectuur bevinden zich binnenin het hoofdproject.

- `/deployment.pdf`:

Instructies voor het deployen en uitvoeren van de verschillende componenten van de architectuur.

- `/javadoc/`:

Documentatie bij de implementatie van de algoritmen.

References

- [1] S. Aggarwal, M. Limaye, A. Netravali, and K. Sabnani. Constrained diameter steiner trees for multicast conferences in overlay networks. *Quality of Service in Heterogeneous Wired/Wireless Networks*, 1(1):262–271, 2004.
- [2] J. Ball, D. Carson, I. Evans, S. Fordin, K. Haase, and E. Jendorck. The Java EE 5 tutorial. <http://java.sun.com/javase/5/docs/tutorial/doc/>, 2006.
- [3] Cisco Systems. *IP Multicast Technology Overview*. http://www.cisco.com/univercd/cc/td/doc/cisintwk/intsolns/mcst_sol/mcst_ovr.htm, 2002.
- [4] D. Clark, B. Lehr, S. Bauer, P. Faratin, R. Sami, and J. Wroclawski. Overlay networks and the future of the internet. *Communications & strategies*, 3(63):1–21, 2006.
- [5] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design, Fourth Edition*. Addison-Wesley, Boston, 2005.
- [6] B. De Vleeschauwer, F. De Turck, B. Dhoedt, and P. Demeester. Online management of QoS enabled overlay multicast services. In *Proceedings of IEEE GLOBECOM 2006, the Global Telecommunications Conference [CD-ROM]*, San-Francisco, USA, 2006.
- [7] B. Dhoedt and F. De Turck. *Cursusnota's bij het opleidingsonderdeel ontwerp van gedistribueerde software*. Faculteit Ingenieurswetenschappen, Universiteit Gent, 2006.
- [8] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol: Version 1.1. *IETF Internet RFC-4346*, <http://www.ietf.org/rfc/rfc4346.txt>, 2006.
- [9] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

-
- [10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.
- [11] Object Management Group. Common object request broker architecture: Core specification. <http://www.omg.org/docs/formal/04-03-12.pdf>, 2004.
- [12] A. Karaman and H. Hassanein. Core-selection algorithms in multicast routing: comparative and complexity analysis. *Computer Communications*, 8(29):998–1014, 2006.
- [13] L. Kaufman and P. J. Rousseeuw. *Finding groups in data. An introduction to cluster analysis*. New York, Wiley, 1990.
- [14] E. Kohler, R. Morris, C. Benjie, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.
- [15] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet, Third Edition*. Addison Wesley, 2005.
- [16] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1(1):281–297, 1967.
- [17] A. Medina, A. Lakhina, I. Matta, and J. Byers. Brite: Universal topology generation from a user’s perspective. <http://www.cs.bu.edu/brite/docs.html>, 2001.
- [18] J. Postel. User datagram protocol. *IETF Internet RFC-768*, <http://www.ietf.org/rfc/rfc768.txt>, 1980.
- [19] E. Rescorla and A. Schiffman. The secure hypertext transfer protocol. *IETF Internet RFC-2660*, <http://www.ietf.org/rfc/rfc2660.txt>, 1999.
- [20] R. Rivest. The MD5 message-digest algorithm. *IETF Internet RFC-1321*, <http://www.ietf.org/rfc/rfc1321.txt>, 1992.
- [21] D. G. Thaler and C. V. Ravishanker. Distributed center-location algorithms. *IEEE Journal on Selected Areas in Communication*, 15(3):291–303, 1997.
- [22] O. Thas. *Cursusnota’s bij het opleidingsonderdeel multivariate dataverwerking*. Faculteit Bio-ingenieurswetenschappen, Universiteit Gent, 2006.

-
- [23] M. Van Der Laan, K. S. Pollard, and J. Bryan. A new partitioning around medoids algorithm. *Journal of Statistical Computation and Simulation*, 73(8):575–584, 2003.
- [24] B. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communication*, 9(6):1617–1622, 1988.
- [25] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.
- [26] F. Yergeau. UTF-8, a transformation format of ISO 10646. *IETF Internet RFC-3629*, <http://www.ietf.org/rfc/rfc3629.txt>, 2003.
- [27] X. Zhang and G. Zhang. A multicast routing algorithm for overlay network built on leased lines. *Applications and the Internet*, 1(1):118–124, 2005.

