Grootschalige Genetwerkte Virtuele Omgevingen

Developing Virtual Environments for Large Audiences

Wim Deprez

Thesis voorgedragen tot het behalen van de graad van licentiaat in de informatica, afstudeervariant informatica-multimedia en de graad van doctorandus in de kennistechnologie.

Promotor: Prof. dr. Wim Lamotte
Begeleider: Peter Quax
Academiejaar: 2003-2004

Abstract

Genetwerkte virtuele omgevingen of NVE's liggen op het snijpunt van virtual reality en netwerking. De grote vraag blijft hoe men ze toegankelijk kan maken voor een groot aantal gebruikers. Hiervoor is onderzoek naar scalability nodig.

Na de definiëring van het onderzoeksdomein gebeurt er een literatuurstudie van voorbeelden uit de geschiedenis van de NVE's. Zo worden er een paar genetwerkte simulatoren van het Amerikaanse Departement van Defensie besproken en een aantal academische implementaties (zoals NPSNET, MASSIVE, DIVE, ...). Het opstellen van een taxonomie is noodzakelijk om de verschillende NVE's met elkaar te kunnen vergelijken en er de sterke en zwakke punten uit te kunnen halen.

De thesis verloopt verder met onderzoek naar multicasting en besluit met de bespreking van de reliable multicast transportprotocollen MTP, RAMP en NORM. Hiervoor werd een taxonomie voorgesteld om de verschillen in de drie protocollen te kunnen benadrukken.

Bij deze thesis hoort ook de implementatie van een multicast framework waar de besproken multicast transportprotocollen in verwerkt zijn. Dit framework kan gebruikt worden voor het ontwikkelen van NVE's door zijn algemene interface en is naar de toekomst toe uitbreidbaar met meerdere protocollen.

Voorwoord

Een werk als dit maak je niet alleen. Er is steun van vele mensen nodig om een thesis tot een goed einde te brengen.

Eerst en vooral ben ik mijn promotor Prof. dr. Wim Lamotte en mijn begeleider Peter Quax mijn dank verschuldigd voor het voorstellen van dit thesisonderwerp en het begeleiden ervan. Mijn vragen waren nooit te veel en een duidelijk antwoord werd altijd gegeven. Zonder hen was hier nooit structuur ingekomen.

Graag wil ik mijn familie bedanken, vooral mijn vader Erik en mijn broer Pieter, die tijdens deze periode een grote steun voor mij hebben betekend. Ook mijn vrienden, mijn studiegenoten en mijn kotgenoten waren altijd daar wanneer het nodig was. Hendrik Stinckens, Yves Willemaers, Geert Bosmans, Els Maurissen en Jan Moesen hebben altijd gezorgd voor het nodige duwtje in de rug.

Daarnaast hebben ook Lev Walkin en Brian Adamson met hun eeuwige geduld al mijn vragen blijven beantwoorden.

Zonder jullie zou dit werk nooit tot stand zijn gekomen.

Deze thesis zou ik willen opdragen aan mijn moeder Mieke De Laat (∘ 22 mei 1951, † 17 mei 2003), die veel te vroeg is heengegaan. Ik hoop dat ik je toch een beetje trots kan maken. *California dreaming. . . .*

Inhoudsopgave

Lijst van tabellen

Lijst van figuren

Samenvatting

Genetwerkte virtuele omgevingen of NVE's laten verschillende, onafhankelijke gebruikers tegelijkertijd via een netwerk met elkaar communiceren en interageren in een computergegenereerde omgeving. NVE's liggen op het snijpunt van virtual reality en netwerktechnologie en krijgen meer en meer aandacht: het onderzoek en aantal toepassingen neemt sterk in aantal toe. Door de toename van het aantal network-enabled devices stijgt ook het aantal potentiële gebruikers.

In de jaren '70 zag men vooral voor engineering het nut in van gedistribueerde, interactieve simulaties. Pas later bleken NVE's ook nuttig te zijn voor het aanleren en ontwikkelen van collectieve vaardigheden. De technologie die hiervoor nodig was, was echter nog niet beschikbaar. Dit kwam pas vanaf de jaren '80 met het opkomen van de microprocessortechnologie en WANs met hoge snelheid.

Het Amerikaanse Departement van Defensie was de grootste ontwikkelaar. De oorspronkelijke genetwerkte simulatoren werden ontwikkeld voor militaire training en simulaties van slagvelden. Er waren aangepaste software implementaties en dure hardware voor nodig. SIMNET was de eerste succesvolle implementatie.

Hieruit groeide de eerste standaard voor gedistribueerde, interactieve simulaties: DIS (IEEE 1278). Het maakt gebruik van het zenden van Protocol Data Units (PDUs) over het netwerk om de toestand van de objecten en de virtuele omgeving in het algemeen kenbaar te maken aan de deelnemers.

Nu personal computers geavanceerder zijn, snellere WAN connecties beschikbaar zijn en er meer bekwame software tools bestaan, is er een groter publiek voor allerhande gedistribueerde simulaties. De bedoeling is nu om simulaties over Internet voor bredere toepassingen mogelijk te maken dan enkel voor militaire doeleinden.

Het meeste onderzoek werd vroeger echter gedaan in opdracht van het Amerikaanse Departement van Defensie en daardoor was de opgedane kennis amper beschikbaar. Door dit tekort aan informatie moesten academici veel heruitvinden wat al ontwikkeld was.

Belangrijke mijlpalen zijn onder andere NPSNET en MASSIVE met hun Areas of Interest Management. Bij Spline wordt er niet zo zeer de nadruk gelegd op consistentie, er worden volledige toestanden doorgestuurd die de oude overschrijven en zo lossen inconsistenties zich vanzelf op.

Modernere pogingen zijn Urbi et Orbi en vGrid. Urbi et Orbi is volledig gedistribueerd. Het maakt gebruik van een eigen scriptingtaal (Goal) en een High Level Scene Description met conceptual graphs om de virtuele werelden te omschrijven. vGrid geeft een ander zicht op NVE's. Met dit project wilt men er een algemeen medium, een general-purpose tool voor communicatie van maken. NVE's worden in het algemeen namelijk op maat gemaakt voor één specifiek doel. Er bestaat geen algemene namespace voor het aanmaken of het opzoeken van NVE's. vGrid wil het opstarten van een NVE zo eenvoudig maken als het boeken van een vergaderruimte.

Na het Amerikaanse leger en de academici heeft ook de entertainmentindustrie zijn weg gevonden naar de genetwerkte virtuele omgevingen. De computerspelletjes van vroeger hebben zich ontwikkeld tot multiplayer games en zelfs het Amerikaanse Departement van Defensie moet erkennen dat hun technologie al een aantal jaren is voorbijgestreefd door die van de game-developers.

Na het beschouwen van deze voorbeelden in hoofdstuk 2 kan men in het algemeen stellen dat de ideale NVE zowel veel gebruikers moet kunnen ondersteunen als lage eisen stellen in verband met netwerking. Alles zou real-time moeten gebeuren op een betrouwbare manier. Maar deze criteria zijn tegelijkertijd niet haalbaar, het ene sluit het andere vaak uit. De oplossing is een product van veel afwegen en beslissingen nemen.

Genetwerkte virtuele omgevingen kunnen opgedeeld worden in een taxonomie aan de hand van keuzes die gemaakt zijn tijdens het ontwerp. Deze taxonomie wordt uitgelegd in hoofdstuk 3. Er wordt een onderscheid gemaakt in de vorm van netwerkcommunicatie en de mogelijke data- en updatemodellen. De belangrijke kenmerken van de netwerkcommunicatie zijn het bandbreedteverbruik, de latency, de distributiemethode en de mogelijke reliability. Datamodellen kunnen opgedeeld worden in gekopieerde en gedeelde modellen en behandelen de manier waarop de data toegankelijk wordt gemaakt voor de verschillende deelnemers. De updatemodellen zijn de methodes waarop de data bijgewerkt wordt.

De nieuwe generatie NVE's doelt op massapopulatie. Om een groot aantal gebruikers te kunnen ondersteunen, is er veel aandacht nodig voor scalability. Kan de NVE nog voldoende doeltreffend functioneren als de processing requirements en het aantal simultaan deelnemende gebruikers van het systeem toenemen? De beperkte systeembronnen vormen een bedreiging voor de scalability van NVE's. Om deze te ontlasten wordt er gebruik

gemaakt van compensatory techniques.

De meest voorkomende technieken zijn message compression, message aggregation, interest management, dead reckoning en het gebruik van levels of details.

Met message compression probeert men data met zo weinig mogelijk bits voor te stellen. Hierdoor kan men kleinere packets gebruiken om de nodige informatie te verspreiden.

Message aggregation vangt een aantal berichten op en verstuurt die in één samengevoegd packet in plaats van allemaal apart. Dit reduceert de vaste overhead die gepaard gaat met het versturen van een packet.

Bij het gebruik van interest management wordt de verstuurde data gefilterd -hetzij door het gebruik van verschillende multicastgroepen of door speciale servers en agents- zodat deelnemers enkel informatie ontvangen die voor hen relevant is. Dit vermindert het aantal berichten dat een deelnemer te verwerken krijgt.

Dead reckoning is een techniek waarbij de deelnemers de toestand van objecten binnen de virtuele wereld afleiden aan de hand van de informatie die men ontvangen heeft uit vorige packets. Nieuwe statusupdates worden door de eigenaar van het object pas verzonden als deze veronderstelt dat de andere gebruikers de toestand niet meer kunnen voorspellen.

Met het gebruik van levels of details ontvangen deelnemers niet alle details over een object of de virtuele wereld indien dit niet nodig blijkt. Bij objecten die op een grote afstand van de gebruiker liggen is het bijvoorbeeld niet nodig dat alle bijzonderheden gerenderd worden.

Zoals gezien bij de netwerkcommunicatie in hoofdstuk 3 laat multicasting groepen van willekeurige grootte toe te communiceren met maar één transmissie door de bron. Het is een efficiënte distributiemethode voor genetwerkte virtuele omgevingen aangezien men gecontroleerd data kan versturen naar een aantal (maar niet noodzakelijk alle) gebruikers. IP Multicasting is echter best-effort. Daarom worden er reliable multicast transportprotocollen ontwikkeld.

In hoofdstuk 4 wordt er een taxonomie voorgesteld om deze reliable multicast transportprotocollen met elkaar te kunnen vergelijken op basis van onder andere de betrouwbaarheid, de schaleerbaarheid en het one-to-many of many-to-many karakter van het protocol. Andere kenmerken zijn de mogelijkheden om packets te fragmenteren, het ordenen van de berichten of het controleren van het lidmaatschap van de groep.

Deze thesis omvat ook de implementatie van een multicastframework. Het framework is aanspreekbaar via een C++ API, zodat NVE's op een eenvoudige manier gebruik kunnen maken van verscheidene (reliable) multicast transportprotocollen. De gebruikte protocollen zijn MTP-2, RMF/RAMP en NORM. In het laatste hoofdstuk worden deze reliable multicast transportprotocollen besproken met behulp van de voorgestelde taxonomie.

Hoofdstuk 1

Inleiding

Networked Virtual Environments (NVE's) maken het mogelijk om verschillende, onafhankelijke gebruikers tegelijkertijd via een netwerk te laten communiceren en interageren in een virtuele, computergegenereerde omgeving. NVE's liggen op het snijpunt van virtual reality en netwerktechnologie en krijgen de laatste tijd meer en meer aandacht. Het onderzoek en aantal toepassingen neemt sterk in aantal toe. Door de toename van het aantal network-enabled devices stijgt ook het aantal potentiële gebruikers.

Dit is een trend die zal doorzetten en zich mogelijk zal integreren in de waaier van alledaagse informaticatoepassingen. NVE's vervagen de afstand tussen de verschillende gebruikers. Gebruikers A, B, ... mogen zich waar dan ook op de wereldbol bevinden, als er een netwerk en een NVE voor handen zijn, kunnen ze zich virtueel in dezelfde kamer bevinden en zo toch nog die broodnodige vergadering meemaken, informatie uitwisselen, les volgen, tegen elkaar opbieden voor een stuk antiek of meespelen in een levensecht ogend multiplayer game.

1.1 Definiëring van het onderzoeksdomein

Virtuele realiteit (Virtual Reality of VR) stelt een begrip van ruimte voor binnenin „cyberspace": binnen in een computer, console, ijskast, ... Virtuele realiteit beslaat virtuele omgevingen (Virtual Environments of VE's) waarmee men via een apparaat of apparaten kan interageren. NVE's zijn virtuele omgevingen die via een netwerk met elkaar verbonden zijn en zo meerdere gebruikers kunnen ondersteunen die niet enkel met de omgeving zelf, maar ook met elkaar kunnen communiceren en interageren.

1.1.1 Virtual Environments

Het is zeer moeilijk om één bindende definitie te vinden voor het begrip „virtuele omgeving", verscheidene definities zijn dan ook voorgesteld door verschillende onderzoekers. In het algemeen [65] mag men aannemen dat een VE een artificiële omgeving is, die meestal drie dimensionaal en interactief is. De techniek achter virtual environments is afhankelijk van de vorderingen op drie andere gebieden: sensor technologieën (input), computation (waaronder het renderen van graphics) en output mogelijkheden.

De input moet duidelijk maken wat de gebruiker wil, waar hij of zij naartoe wil gaan, welke handeling er uitgevoerd moet worden, ... Dit is al lang niet meer beperkt tot muis en toetsenbord. Nieuwe technieken zijn ontstaan zoals het gebruik van speciale handschoenen of pakken met sensoren, haptics (touch & force), spraak, 3D muizen, ...

De functionaliteit, modeling en representatie van een VE wordt verzorgd door de computation. Dit omvat ook het gedrag en de mogelijkheden van interactiviteit tussen gebruiker en omgeving (later ook tussen gebruikers onderling).

De output moet de ervaring voor de gebruiker zo waarheidsgetrouw mogelijk maken. Hier wordt niet enkel gesproken over visual displays (zoals een scherm, bril of head-mounted displays), maar ook geluid en haptics.

Deze technologieën zijn nog steeds aan ontwikkeling onderhevig en dit maakt de ervaring van de gebruiker iedere dag beter en de mogelijkheden met VE's iedere dag uitgebreider.

1.1.2 Networked Virtual Environments

In het begin waren VR ervaringen beperkt tot *single user* applicaties, maar sinds de zogenaamde *desktop VR*, waarbij enkel een stevige personal computer vereist is, is de deur geopend voor allerhande multi user applicaties, zodat deze gebruikers samen gedeelde virtuele werelden kunnen ervaren en erin interageren.

Opdat die verschillende gebruikers zich overal zouden kunnen bevinden, werd bij de VR technologie netwerking toegevoegd. Dankzij netwerking kunnen NVE's ook systeembronnen delen (distributed system resources), zoals het verdelen van intensieve berekeningen en operaties over verschillende computers, om een betere performantie na te streven.

In deze thesis worden technieken onderzocht die het mogelijk maken om het gebruik van genetwerkte virtuele omgevingen niet meer te beperken tot een lokaal netwerk (Local Area Network of LAN), zodat er een applicatie- en netwerkframework ontwikkeld kan worden dat toelaat om grote aantallen gebruikers gelijktijdig van een NVE applicatie gebruik te laten maken. Hierbij zijn vooral aspecten zoals scaling zeer belangrijk.

1.2 Terminologie

Om genetwerkte virtuele omgevingen te beschrijven is er een aangepaste terminologie nodig. *Persistentie* houdt in dat de virtuele wereld blijft voortbestaan, ook als er geen enkele gebruiker is ingelogd. De klok tikt bij wijze van spreke verder. *Immersiveness* gaat over het gevoel van echt aanwezig te zijn in de virtuele wereld en hoe sterk dat gevoel is.

Daarnaast wordt ook kort uitgelegd wat een *avatar*, de virtuele verpersoonlijking van de gebruiker, inhoudt en wat een aantal van diens taken zouden kunnen zijn.

1.2.1 Persistentie

Als een virtuele omgeving blijft voortbestaan, ook als er geen gebruikers meer zijn ingelogd of als de verbinding is wegvallen, dan wordt deze omgeving **persistent** genoemd. De omgeving blijft dus bestaan, onafhankelijk van het al dan niet aanwezig zijn van deelnemers. Dit maakt het mogelijk dat een virtuele wereld zich kan ontwikkelen, dat de gebruiker een gevoel van tijd kan ervaren en dat bepaalde processen blijven doorgaan. Een persistente virtuele wereld hoeft in principe maar één keer te worden opgestart.

Persistentie is zeer interessant voor onder andere strategie of roll playing games, zoals de MMORPGs (zie sectie 2.4.2) waarin de deelnemer een alter ego heeft waarmee hij of zij in de virtuele wereld kan rondtrekken, opdrachten uitvoeren, andere deelnemers ontmoeten, een beroep uitoefenen, ... Of de speler nu is ingelogd of niet, de virtuele wereld zal blijven doordraaien.

1.2.2 Immersiveness

De **immersiveness** van een virtuele omgeving is het realiteitsgevoel dat heerst bij de gebruikers van de omgeving. Een immersive VE is een virtuele omgeving die zeer realistisch aanvoelt, de gebruiker voelt zich ondergedompeld[1] in de virtuele realiteit.

Immersiveness wordt al grotendeels behaald door het gebruik van een first-person view, waarbij de gebruiker door de ogen van de avatar (zie sectie 1.2.3) kijkt. Een goed voorbeeld van immersive virtuele omgevingen zijn de 3D shoot'em ups (zie sectie 2.4.1) zoals *Doom* en *Quake* van id Software.

Als de gebruiker daarentegen een algemeen topdown zicht krijgt (zoals het bovenaanzicht van een stad, een plan voor een vliegtuigmotor, ...), voelt de gebruiker zich minder aanwezig in de virtuele ruimte, want hij krijgt een overzicht en heeft daardoor niet echt het

[1] het Engelse werkwoord *immerse* betekent *onderdompelen*

gevoel dat hij kan rondwandelen, ergens tegen aan kan botsen, ...

1.2.3 Avatars

Een **avatar** [73] is de verpersoonlijking van de gebruiker in de virtuele wereld. Deze avatar *is* in feite de gebruiker in de virtuele wereld. Waar de avatar van een gebruiker aanwezig is, is de gebruiker virtueel aanwezig. Avatars duiden onder andere de positie en de oriëntatie van de gebruiker aan en doen ook de handelingen binnen de virtuele omgeving. Via zijn avatar kan de gebruiker met objecten of andere avatars (van andere gebruikers) in de NVE interageren.

Door de avatar voelt de gebruiker zich aanwezig in de virtuele ruimte en andere deelnemers voelen dat zij met deze gebruiker kunnen interageren. Dit is het gevoel van *presence* en *copresence*. De gebruiker ziet andere avatars in de virtuele wereld en veronderstelt zo dat er andere deelnemers aanwezig zijn en dat hij hiermee kan interageren.

Wat de avatar „ziet" krijgt ook de deelnemer te zien. Normaal gesproken ziet een gebruiker zijn eigen avatar niet, zoals een persoon in de echte wereld zichzelf ook niet kan zien, behalve als deze in een spiegel kijkt. Als de gebruiker bij bepaalde handelingen een deel van het virtuele lichaam (bijvoorbeeld de handen bij het vastnemen van een object, de voeten als de gebruiker naar beneden kijkt, ...) kan zien, dan voelt de deelnemer zich aanwezig in de virtuele omgeving. Een realistische avatar helpt mee aan de immersiveness van een omgeving.

Avatars kunnen zeer belangrijk zijn in realistische (3D) NVE's. Ze nemen de taken over van het menselijk lichaam in de echte wereld. Bij geavanceerde avatars ziet niet alleen de deelnemer wat de avatar ziet, andere deelnemers kunnen aan de houding, de positie en de oriëntatie van een avatar zien wat de andere gebruiker op dat moment aan het bekijken is. Een avatar kan ook de emotie weergeven van een gebruiker zodat anderen merken in welke gemoedstoestand deze zich bevindt.

De avatar van een gebruiker kan aangeven of de gebruiker wel dan niet beschikbaar is voor interactie. Dit kan gebeuren door een klassiek „niet storen"-tekentje of door voldoende informatie te tonen over de huidige activiteit. Als de beschikbaarheid niet duidelijk gemaakt kan worden, kunnen andere gebruikers niet uitmaken of een deelnemer te druk bezig is of hen negeert wegens geen interesse.

Bovendien kan ook duidelijk gemaakt worden welke actie de gebruiker aan het uitvoeren is. Wanneer een avatar een object opraapt, het een bepaalde tijd met zich mee draagt en het dan uiteindelijk loslaat of neerzet, dan weten de andere deelnemers dat die gebruiker

net een object in de virtuele wereld verplaatst heeft. Deze informatie kan van groot belang zijn in NVE's waar samenwerking of communicatie tussen de gebruikers van groot belang is.

Als een gebruiker in een virtuele omgeving (uitgaande van het feit dat niet alle gebruikers hun avatars er hetzelfde uitzien) telkens dezelfde avatar gebruikt, dan kunnen andere deelnemers hem of haar daaraan herkennen van vorige sessies. Aan de avatar kunnen ook aanpassingen gebeuren om aan te duiden dat deze gebruiker een hogere sociale status heeft binnen de virtuele omgeving. Moderators kunnen bijvoorbeeld een herkenbaar uniform dragen. In Active Worlds[2] krijgen de niet-leden een toeristenpakje aan (zie figuur 1.1).

Figuur 1.1: Sociale status in Active Worlds: de toerist.

Doordat de avatar de gebruiker in de virtuele wereld voorstelt, wordt verondersteld dat beide termen in de volgende hoofdstukken door elkaar gebruikt kunnen worden zonder verwarring te scheppen.

1.3 Conclusie

In dit hoofdstuk werd het onderzoeksdomein van deze thesis verduidelijkt. De besproken terminologie zal in het volgende hoofdstuk gebruikt worden om overzichtelijk voorbeelden van een aantal bestaande genetwerkte virtuele omgevingen te geven.

Voor meer gedetailleerde informatie wordt er doorverwezen naar de werken [45], [59] en [65].

[2] http://www.activeworlds.com/

Hoofdstuk 2

Voorbeelden van Networked Virtual Environments

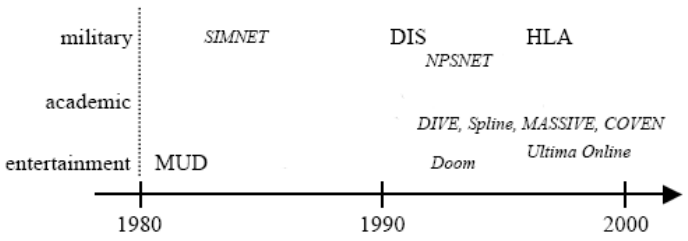
Na de definiëring van het onderzoeksdomein en de terminologie, kan in dit hoofdstuk een aantal van de belangrijkste genetwerkte virtuele omgevingen behandeld worden om een overzicht te verschaffen van wat de mogelijkheden en de vorderingen zijn. Hierdoor kan men ze onderling vergelijken en conclusies trekken van wat de vereisten zijn om een NVE te ontwikkelen voor een groot aantal gebruikers.

Op figuur 2.1 wordt een voorbeeld van de ontwikkeling geschetst, een tijdlijn van de ontwikkelingen op het gebied van Networked Virtual Environments, gebaseerd op de tijdlijn in „A Review on Networking and Multiplayer Computer Games" van J. Smed, T. Kaukoranta en H. Hakonen [61].

Dit werk kan echter nooit een volledige, gedetailleerde opsomming en beschrijving van al deze systemen geven (en dat is ook niet het doel van deze thesis), maar zal de algehele ontwikkeling en trends weergeven en zich richten op de netwerkvereisten, zoals de keuze van het onderliggende netwerkprotocol, de gebruikte bandbreedte en de software architectuur.

2.1 De oorsprong van NVE's: Genetwerkte simulatoren

Het prille begin van de NVE's situeert zich in de genetwerkte simulatoren. Toen (jaren '70) zag men vooral voor engineering het nut in van gedistribueerde, interactieve simulaties. Pas later werd ingezien dat NVE's ook nuttig zijn voor het aanleren en ontwikkelen van collectieve „vaardigheden", maar de technologie die hiervoor nodig was, was nog niet beschikbaar. Het ontwikkelen van grote netwerken voor simulatoren werd pas mogelijk



Figuur 2.1: De trend op het gebied van NVE's: van genetwerkte simulatoren tot computer games. vrij naar: [61]

vanaf de jaren '80, met het opkomen van de microprocessortechnologie en WANs met hoge snelheid.

Het Amerikaanse Departement van Defensie (The US Department of Defense of DoD) was (in het begin) de grootste ontwikkelaar van grote genetwerkte virtuele omgevingen. De oorspronkelijke multi-user VE's werden ontwikkeld als militaire programma's en hadden dure hardware en aangepaste software implementaties nodig. Simulator Networking (SIMNET) was de eerste succesvolle implementatie en werd gebruikt voor militaire training en simulaties van een slagveld.

2.1.1 SIMNET

Ontwikkeld door DARPA, US Army, Bolt Beranek and Newman Inc. (BBN), Perceptronics, Inc. en Delta Graphics tussen 1983 en 1990, is SIMNET gebouwd rond drie belangrijke elementen [47]: Object/event architectuur, autonome simulatieknopen en *dead reckoning* algoritmes.

Object/event architectuur modelleert de wereld als een verzameling objecten die met elkaar interageren in bepaalde gevallen (events). Events zijn de berichten die over het netwerk gestuurd worden om de worldstate, de objectstatus of een verandering in één van deze aan te geven. De objecten zijn de voertuigen en de wapens (die dus met elkaar kunnen interageren). Opvallend is wel dat gebouwen en de terreinen op zich in de simulatie niet als objecten worden gezien. Als men in de simulatie een brug of een huis wilt opblazen, moeten dezen als objecten worden opgenomen.

Autonome simulatieknopen (in tegenstelling tot een gecentraliseerde benadering) beheren autonoom de status van één of meerdere objecten in de gesimuleerde wereld

en spelen „events" geassocieerd met dat object (of die objecten) door naar andere simulatieknopen (simulation nodes). Een knoop die een bepaald object beheert, wordt de *home node* van dat object genoemd.

Knopen kunnen onafhankelijk van de anderen meedoen aan een simulatie of deze verlaten. Iedere knoop zendt *absolute truth* over zijn objecten en „events". Gebaseerd op deze informatie kunnen andere knopen dan beslissen of (en hoe) ze hierdoor beïnvloed worden. Om de communicatieoverhead zo klein mogelijk te houden, worden de status updates enkel doorgeseind als het gedrag van het object verandert (zie de *dead reckoning algoritmes*).

Het ontbreken van een centrale server wilt ook zeggen dat als één enkele node faalt in het systeem, dit niet de gehele simulatie zal belemmeren.

Dead reckoning algoritmes: hiermee kan de huidige toestand van een object afgeleid worden via de informatie die al eerder ontvangen is. Het gedrag van het object wordt dus voorspeld aan de hand van zijn vorige toestanden. Zo wordt het netwerkverkeer verminderd, er worden enkel packets in verband met de object state op het netwerk geplaatst als de home node van dat object beslist dat de andere nodes niet meer de juiste toestand van het object kunnen voorspellen. (zie ook sectie 3.3.4)

SIMNET definieert zijn netwerkvereisten als volgt: de totale latency mag maximum 250 ms bedragen, de jitter (het verschil in latency) moet laag zijn en er mogen maximum 1 tot 2 % van de datagrammen verloren gaan, anders komt de simulatie te onnatuurlijk over. Hoewel SIMNET geen specifieke hardware of lower level protocol vereist, wordt er toch aangeraden om met de speciale SIMNET network library [40] te werken, om zo voordeel te halen uit de multicast mogelijkheden van Ethernet.

De grootste uitvoering met SIMNET was in maart 1991, waar 850 actieve simulatoren, verspreid over vijf plaatsen, aan meededen. De nodige bandbreedte was net onder de T-1 snelheid (1,544 Mbps).

In 1993 werd de uit SIMNET ontworpen Distributed Interactive Simulation (DIS) standaard als ANSI/IEEE standaard aanvaard (IEEE 1278 [19]) en deze werd herzien in 1995 [33].

2.1.2 DIS standaard

Uit de SIMNET technologie werd een standaard ontworpen, de Distributed Interactive Simulation (DIS) standaard. Deze werd als ANSI/IEEE standaard (IEEE 1278 [19]) aanvaard in 1993 en herzien in 1995 [33]. Hierdoor is het dan niet verwonderlijk dat DIS en SIMNET een zelfde designfilosofie hanteren (zie de drie belangrijke elementen bij SIMNET: Object/event architectuur, autonome simulatieknopen en dead reckoning algoritmes).

Tabel 2.1: SIMNET in de taxonomie (zie hiervoor hoofdstuk 3).

aantal deelnemers	850
bandbreedte	T-1 (1,544 Mbps)
distributiemethode	broadcast (multicast met de SIMNET network library)
datamodel	replicated homogeneous world database
access control	niet van toepassing (replicated)
updatemodel	active copying

Compensatory techniques

compressie	niet bekend
aggregatie	nee
interest management	nee
dead reckoning	ja
level of details	nee

DIS definieert een infrastructuur [20] voor het linken van verschillende soorten simulaties op meerdere locaties om realistische, complexe virtuele werelden te creëren voor de simulatie van hoog interactieve activiteiten. De DIS-infrastructuur maakt het mogelijk dat verschillende systemen, technologieën, producten en platformen die ontwikkeld werden voor verschillende doeleinden (alhoewel hoofdzakelijk militaire[1]) met elkaar kunnen communiceren vanaf verschillende geografische plaatsen.

De DIS standaard (IEEE 1278) bestaat uit twee delen:

De applicatie protocols

Dit deel definieert hoe de data packetten, gebruikt voor communicatie tussen de simulatoren onderling, er moeten uitzien. Deze packetten worden de Protocol Data Units (PDUs) genoemd. Het gebruik van de PDUs heeft zijn weerslag op keuzes in verband met het ontwerp van simulatoren, zo moet bijvoorbeeld de positie van een entiteit regelmatig (om een bepaald interval, de heartbeat) opnieuw doorgestuurd worden.

Er worden 27 PDUs gedefinieerd volgens de DIS standaard, gerangschikt in de volgende 6 categorieën: Entitity information/interaction, Warfare, Radio Communications, Logistics,

[1] DIS werd hoofdzakelijk ontwikkeld voor militaire doeleinden, maar E. A. Fitzsimmons en J. D. Fletcher hebben in hun paper „Beyond DoD: non-defense training and education applications of DIS" [22] aangetoond dat de standaard ook op andere gebieden van nut kan zijn. DIS is bijvoorbeeld praktisch gebleken voor Air Traffic Control Trainers en Intelligent Transport Systems.

Distributed Emission Regeneration en Simulation Management.

Slechts vier van de Protocol Data Units worden door de nodes gebruikt om te interageren met de omgeving en dat zijn de PDUs uit de categorieën Entity information en Warfare. Een demonstratie op de *Interservice/Industry Training and Education Conference* toonde aan dat de Entity State PDUs 96% van het totale netwerkverkeer beslaan.

In de meeste simulaties die later gebruikt werden en compatibel zijn met de DIS standaard (zoals bijvoorbeeld NPSNET, zie sectie 2.2.1) worden dan ook niet alle 27 PDUs gebruikt. De andere PDUs worden gewoon genegeerd of veroorzaken een korte error die aangeeft dat er een niet ondersteunde PDU ontvangen werd.

De communicatie vereisten, services en profiles

Vroeger (ten tijde van SIMNET en de eerste implementaties van DIS) zaten alle simulatieknopen op één enkele LAN en konden de PDUs gewoon gebroadcast worden. Dat is echter niet meer zo optimaal als DIS simulaties over een WAN moeten gebeuren: een realistische simulatie kan duizenden entiteiten bevatten, bestuurd door mensen of de computers zelf.

De netwerkvereisten voor DIS [58, 57] kunnen als volgt worden samengevat [45]: De levering van data moet real-time gebeuren. Hiermee wordt „real-time" in termen van menselijke gewaarwording bedoeld: de latency mag maximum een paar honderd milliseconden bedragen en de jitter moet laag blijven. Het reserveren van bandbreedte en andere resources, en vooral een vorm van een gedeelde „on demand" capaciteit moeten gebruikt worden om meer economische oplossingen aan te bieden. Niet real-time data zou zoveel mogelijk moeten overgebracht worden tijdens de voorbereiding van een sessie. Het is mogelijk dat het netwerk gecomprimeerde video en andere real-time tools zal moeten ondersteunen. Ook zou multicasting mogelijk moeten zijn met duizenden multicastgroepen en hoge join/leave snelheden. Voor multicasting wordt het many-to-many paradigma gebruikt waarbij bijna alle deelnemers in iedere groep zowel zenders als ontvangers zijn. Een andere vereiste kan beveiligde networking zijn, voor het geval dat het om geheime simulaties gaat.

De netwerkinfrastructuur van de DIS-standaard is heterogeen, ieder type computer of machine dat in het netwerk geplugd kan worden, DIS PDUs (zie het deel *De applicatie protocols* van 2.1.2) kan lezen en schrijven en deze PDUs kan beheren, kan volledig meedraaien in een DIS omgeving. De virtuele omgeving kan zowel virtual players (die door personen aan consoles worden bestuurd), constructive players (deze worden volledig softwarematig bestuurd, ook *bots* genaamd) als live players (echte wapensystemen die ingeplugd worden in het DIS netwerk) bevatten.

Ongeveer honderd verschillende systemen werken met de DIS standaard, DIS mag dus best succesvol genoemd worden. Toch zijn er een aantal beperkingen. De PDU packets zijn

groter dan die van SIMNET. De Entity State PDU bevat veel data die niet veranderd, ze bevatten meer bits dan nodig is en recent onderzoek tracht de lengte terug te brengen tot 20%. Dit zou een hele verbetering betekenen, aangezien deze PDUs het overgrote deel (zo een 96%) van de netwerkcommunicatie zijn. Een mogelijkheid zou bijvoorbeeld zijn om telkens enkel de veranderingen door te sturen en op regelmatige basis (na X aantal packets of na X aantal milliseconden) de volledige Entity State door te sturen, om de consistentie te behouden.

DIS is goed bruikbaar voor real time simulaties tot 300 gebruikers, maar zal niet goed schaleerbaar zijn voor grote simulaties met duizenden deelnemers.

Tabel 2.2: DIS in de taxonomie.

aantal deelnemers	300
bandbreedte	geen concrete cijfers
distributiemethode	oorspronkelijk enkel broadcast, modernere versies ook multicast
datamodel	replicated homogeneous world database
access control	niet van toepassing (replicated)
updatemodel	active copying

Compensatory techniques

compressie	niet bekend
aggregatie	nee
interest management	nee
dead reckoning	ja
level of details	nee

2.1.3 High Level Architecture

In de eerste plaats werd DIS ontwikkeld voor militaire training (real-time simulaties voor tactische oefeningen). Met de volgende generatie, DIS++, wil men ook niet-militaire simulaties kunnen ondersteunen. DIS++ omvat de concepten en technologieën van het Common Technical Framework, opgebouwd uit Data Standards, een Conceptual Model of the Mission Space en een High Level Architecture (HLA). De HLA werd in 2000 erkend als IEEE standaard (IEEE 1516) maakt het gemakkelijker om een simulatie en zijn componenten te hergebruiken door een algemene structuur van interfaces tussen simulaties te ontwerpen, zonder specifieke eisen te stellen over de implementatie van iedere simulatie.

De vier basisconcepten zijn [45]:

Run-Time Infrastructure (RTI) is een implementatie van een gedistribueerd operating system als basis voor toekomstige HLA applicaties die de communicatie verzorgen tussen alle simulatiemodellen. Belangrijkste diensten zijn: Federation Management, Declaration Management, Object Management, Ownership Management en Time Management.

Interface Specification is een formele, functionele beschrijving van de interface tussen de HLA applicatie en de RTI.

Rules is een verzameling van tien technische regels waar een HLA deelnemer zich aan moet houden.

Object Model Templates zijn gestandaardiseerde formaten om de functionaliteit van simulatiemodellen en de interactie onderling te bepalen voordat de eigenlijke simulatie plaatsvindt.

Om het aantal berekeningen en het netwerkverkeer onder controle te houden, wordt er gebruik gemaakt van *publicatie* of *subscriptie*. Publicatie houdt in dat iedere simulatie registreert welke objecten en attributen het zal weergeven. Subscriptie wilt zeggen dat iedere simulatie registreert welke attributen en interacties het zal nodig hebben om zijn taak uit te voeren. Beiden zijn dynamisch en kunnen veranderd worden tijdens een sessie.

Vergeleken met DIS ondersteunt HLA een breder gamma aan simulaties. HLA is een architectuur, het is niet enkel een protocol. DIS gebruikte oorspronkelijk enkel broadcast, terwijl HLA selectief data doorstuurt tijdens simulaties, waardoor het netwerkverkeer vermindert. HLA is dus meer scalable dan DIS. Ten laatste scheidt HLA de simulatie applicatie van het onderliggende communicatie protocol, dit in tegenstelling tot DIS, waar de PDUs veel van de applicatie bepalen. Met HLA is het besef gekomen dat er een breder publiek voor genetwerkte virtuele omgevingen is[2], het heeft dan ook geen vaste militaire PDUs zoals „Fire" en consorten.

2.2 Genetwerkte multi-user virtual environments

De oorspronkelijke multi-user VE's werden vooral ontwikkeld voor militaire doeleinden en hadden dure hardware en aangepaste software implementaties nodig. Nu de personal computers geavanceerder zijn, de snellere WAN connecties ook mogelijk zijn voor minder kapitaalkrachtigen en er meer bekwame software tools bestaan, is er een groter publiek gecreëerd voor allerhande gedistribueerde simulaties. Het ontwerpen van een High Level Architecture was al een eerste stap in de goede richting. De bedoeling is nu om simulaties over Internet voor bredere toepassingen mogelijk te maken dan enkel voor militaire

[2]zie ook *Beyond DoD: Non-defense training and education applications of DIS* van E. A. Fitzsimmons en J. D. Fletcher [22]

doeleinden.

Het grote probleem was echter dat het verrichte onderzoek, voor het overgrote deel gefinancierd door het Amerikaanse Department of Defense, en de opgedane kennis amper beschikbaar was. Door dit tekort aan informatie moesten de academici veel heruitvinden wat al ontwikkeld was.

In deze sectie worden een aantal multi-user VE's bekeken die ontwikkeld zijn in de academische en de onderzoekswereld.

2.2.1 NPSNET

De Naval Postgraduate School Networked Vehicle Simulator (NPSNET)[3] werd ontwikkeld in 1986 door de NPSNET Research Group van de Naval Postgraduate School in Monterey, California en is de eerste VE die opereerde over het Multicast Backbone (MBone - zie sectie 4.1) gebruik makend van het IEEE 1278 DIS protocol. NPSNET kan meer dan 1000 gelijktijdige gebruikers (die onder andere mensen, zeeschepen, voertuigen of vliegtuigen simuleren) ondersteunen.

Geschiedenis van NPSNET [42, 62, 14]

Het oorspronkelijke NPSNET-1 werd gedemonstreerd op ACM's Siggraph conference van 1991[4]. Het werkt op de workstations op een LAN, maar stuurt packets over het LAN tegen framerate: telkens als het scherm hertekend moet worden wordt er een packet verstuurd, zodoende wordt het LAN overspoeld door packets. De opvolgers NPSNET-2 en NPSNET-3 onderzochten betere en snellere manieren om de graphics te genereren en de terreindatabanken uit te breiden. Vanaf NPSNET-IV wordt het DIS protocol gevolgd en gebruik gemaakt van IP Multicast [42]. Deze versie past ook dead reckoning algoritmes toe en bevat spatial sound.

De laatste versie NPSNET-V [14] is een Java-based componentenarchitectuur voor NVE-applicaties en heeft als doel om een praktisch werkplatform voor het onderzoek naar en gebruik van NVE's te worden.

In NPSNET-V stelt een entiteit een object (zoals een persoon, een boom, een voertuig, een vliegtuig, ...) in de virtuele wereld voor. Deze entiteit bevat statusinformatie van het object (zoals locatie, oriëntatie, snelheid, kleur, ...) en een lijst van protocols dat dit object kent. Deze protocols zijn een mapping tussen network messages en een verzameling

[3] http://www.npsnet.org/%7Enpsnet/
[4] http://www.siggraph.org/%7Efujii/etech/1991%5F17.html

van acties en gedragingen van de entiteit. Als een protocol een network message ontvangt, bepaalt het de inhoud hiervan en verandert dan, op een gepaste manier, de status van de entiteit. De protocols drijven de entiteiten aan. Om het gedrag van een entiteit aan te passen of om een nieuwe actie toe te voegen, moet men dus het protocol veranderen of aanvullen.

NPSNET-V is uitbreidbaar: als een client een entity type voor de eerste keer tegenkomt, moet het diens beschrijving downloaden. Zo kunnen clients dynamisch nieuwe entiteiten en omgevingen leren begrijpen. Op dezelfde manier zijn ook de protocols dynamisch uit te breiden. Deze versie maakt gebruik van de Extensible Markup Language (XML) om de status van bepaalde modules te kunnen opslaan, bewerken, ... op een leesbaar en version-save formaat[5]. Op ieder moment tijdens run-time kan elk deel of alles van een applicatie via XML aangepast worden door een ontwerper en toegepast of opgeslagen worden als een prototype. Zo zijn de virtuele werelden en de entiteiten (maar ook andere componenten, zoals de netwerkprotocollen, het area of interest management, ...) op een praktische manier uit te breiden.

Het players and ghosts paradigma, scalability en de Area of Interest Manager [42, 45]

De filosofie achter NPSNET is gebaseerd op het *players and ghosts* paradigma [42]: een object dat bestuurd wordt, wordt op zijn eigen host een player genoemd en zijn representaties op andere hosts noemt men ghosts. Via een dead reckoning algoritme (zie sectie 3.3.4 voor uitleg over dit soort algoritmes) worden de posities van de ghosts berekend. De hosts sturen regelmatig updates van hun players naar de andere deelnemers.

In de DIS standaard heeft de Research Group een aantal problemen in verband met scalability ontdekt [45]:

zo is de vereiste bandbreedte voor grootschalige VE's enorm en één host kan niet alle nodige berekeningen (dead reckoning, collision detection, modelleren, ...) uitvoeren als er zich meer dan duizend entiteiten in de omgeving bevinden. DIS eist ook dat alle identiteiten regelmatig Entity PDUs verzenden, het maakt niet uit of hun status is veranderd of niet. Dit is overbodig als het over statische objecten (zoals bruggen) of moeilijk verplaatsbare entiteiten gaat. De status van dezen wordt tijdens een sessie meestal slechts een aantal keer veranderd, een brug kan je maar één keer opblazen en daarna zal er niet meer veel verandering in komen.

Daarnaast verwacht DIS dat iedere simulator „eerlijk" de juiste status doorstuurt (absolute truth). Alle modellen en werelddatabanken moeten gekopieerd worden naar iedere simulator, want er is geen mechanisme dat deze op aanvraag distribueert. De NPSNET

[5] http://www.npsnet.org/%7Enpsnet/v/index.html

Research Group beweert ook dat de datarepresentaties niet compatibel zijn met verschillende soorten hardware die als simulatoren gebruikt worden.

In NPSNET hoopt men een aantal scalability-problemen op te lossen door de software architectuur:

Door het gebruik van multicast in plaats van broadcast zijn sessies niet meer beperkt tot LANs, maar kunnen ze ook over WANs of over het Internet gebruikt worden.

Een Area of Interest Manager deelt de VE op in een verzameling kleinere omgevingen en vermindert zo de druk op de hosts om alles individueel te berekenen. De theorie hierachter is dat hosts enkel informatie moeten krijgen van de gebeurtenissen in hun eigen deelomgeving (en eventueel buuromgevingen). Dit vermindert ook de netwerkcommunicatie. Het opdelen mag gebeuren op basis van de ruimte, van functionele eigenschappen, ... De deelomgevingen worden gemapt op multicastgroepen.

NPSNET-V maakt gebruik van het Lightweight Directory Access Protocol (LDAP) om componenten te lokaliseren en te downloaden [14]. LDAP is een protocol om online directory services[6] aan te spreken. Het is een relatief simpel protocol voor het updaten en doorzoeken van directories, gebruik makend van TCP/IP [7].

De hiërarchie van de directories laat scalability toe door middel van partitioned responsibility. Delen van de namespace -en dus van de virtuele wereld(en)- kunnen opgedeeld worden en toegewezen worden naar verschillende autoriteiten, zoals het Domain Name System (DNS). Hierdoor wordt het beheer van de laadbare componenten gemakkelijker uitvoerbaar.

Door het kopiëren van de directories naar andere server machines, vermindert LDAP ook het gevaar op een bottleneck zoals bij de traditionele server-based systemen. De informatie wordt over verscheidene servers verspreid en zo wordt de belasting op de verschillende servers verminderd zodat het gehele systeem minder kwetsbaar wordt.

2.2.2 CAVE

De Cave Automatic Virtual Environment (CAVE) [17] ziet er uit als een kleine kamer (10x10x9 feet of ongeveer 3x3x2.7 meter). De gebruiker is omringd door grote schermen. De beelden op die schermen worden door shuttered glasses[8] bekeken. De beelden veranderen naarmate de hoofdbewegingen van de gebruiker (head-tracked). Als verscheidene deelnemers zich tegelijkertijd in de CAVE bevinden, dan is de head-tracking slechts op één iemand van toepassing, de beelden hangen af van de positie van die persoon zijn

[6] een gedistribueerde, cross-platform, gestandaardiseerde database (zie ook vGrid in sectie 2.3.2)

[7] http://en2.wikipedia.org/wiki/LDAP, http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?LDAP

[8] een speciale bril die zeer snel afwisselend het linker- of het rechteroog afdekt om zo stereoscopisch zicht te verkrijgen

Tabel 2.3: NPSNET-V in de taxonomie.

aantal deelnemers	meer dan 1000
bandbreedte	geen concrete cijfers
distributiemethode	multicast
datamodel	shared distributed client-server databases
access control	gecentraliseerd
updatemodel	copying on demand

Compensatory techniques

compressie	niet bekend
aggregatie	nee
interest management	ja
dead reckoning	ja
level of details	nee

hoofd. In de CAVE gebruikt men meestal een soort draadloze 3D muis (een *wand*) die via een antenne informatie over zijn positie en oriëntatie naar de computer stuurt. Dit om te interageren met de VE.

CAVE is geen NVE, maar een immersive projection technology of IPT. Toch werd deze technologie kort toegelicht omdat zij gebruikt wordt om volgende (en soortgelijke) projecten zichtbaar of "beleefbaar" te maken voor gebruikers:

Collaborative Architectural Layout Via Immersive Navigation

Een nuttige eigenschap van een virtuele omgeving is dat ze is opgetrokken uit software in plaats van uit bouwmateriaal. Dit maakt het dan ook interessant om dure ontwikkelingen in de echte wereld eerst eens virtueel te bekijken. Dat ziet er stukken beter uit dan een 2D blueprint op papier en motiveert de beslissingen van de opdrachtgever(s).

Collaborative Architectural Layout Via Immersive Navigation (CALVIN) maakt gebruik van een CAVE. Het is een prototype interface voor bouwkundig ontwerp in VR en collaboratieve visualisatie hetgeen verschillende perspectieven beklemtoont. CALVIN is gebaseerd op het idee van een persistente VE, waar een gecentraliseerde databank zowel de huidige als vorige versies van de omgeving bijhoudt.

Verschillende CALVINs zijn via een ATM netwerk verbonden met de databank. Voorafgaand aan de sessie moet de databank (met alle scenes, objecten en avatars) geladen

worden. Iedere gebruiker beheert enkel het deel van de databank die informatie bevat die de gebruiker aangaat (lokale scene, avatar van de gebruiker, positie in de VE, ...). De consistentie van de gecentraliseerde databank wordt op ieder moment behouden.

Virtual Prototyping System

Het Virtual Prototyping System (VPS), een collaboratief VR prototyping systeem gebaseerd op CAVE, werd ontwikkeld door een team van het National Center for Supercomputing Applications in samenwerking met het German National Research Center for Information Technology. Het systeem bestaat uit Computer Aided Design (CAD) objecten en een package dat dynamica simuleert.

Na de opstellingstijd, haalt de simulatie een snelheid van 15 frames per seconde. Communicatie tussen de locaties in de VE wordt via IP multicast over een ATM netwerk geregeld. In VPS hebben de gebruikers geen avatars (in tegenstelling tot de meeste VE's), hoewel iedere gebruiker zijn eigen gezichtspunt bewaart. Via MBone tools kunnen de gebruikers elkaar audio en/of video doorsturen.

VPS is niet scalable: alle deelnemers behoren tot dezelfde multicastgroep en kunnen wel verbinden en afsluiten wanneer ze willen, maar iedere deelnemer houdt een lijst bij van alle actieve deelnemers. Het aantal gebruikers is beperkt omdat iedere extra deelnemer een toename van ongeveer 1Mbit/s betekent. Het systeem hangt sterk af van dedicated ATM links. Ieder zendt de nodige informatie voor de synchronisatie van de VE. Inconsistentie door latency wordt echter niet behandeld. Dead reckoning wordt niet gebruikt.

2.2.3 MASSIVE

Het Model, Architecture and System for Spatial Interaction in Virtual Environments (MASSIVE) [30, 28, 29] is ontwikkeld aan de Computer Science Department van de University of Nottingham en is een multi-user distributed VR collaborative environment. MASSIVE wordt vooral gebruikt voor teleconferencing.

MASSIVE-1

De eerste versie, MASSIVE-1, was een experimenteel systeem: het doel was om een grootschalige omgeving gebaseerd op het *spatial model of interaction* te ontwikkelen. Het spatial model of interaction is gebaseerd op de concepten aura en awareness [30] (zie ook sectie 3.3.3).

Tabel 2.4: CAVE VPS in de taxonomie.

aantal deelnemers	2 CAVEs
bandbreedte	1Mbit/s per gebruiker
distributiemethode	multicast
datamodel	shared distributed client-server databases
access control	gecentraliseerd
updatemodel	active copying

Compensatory techniques

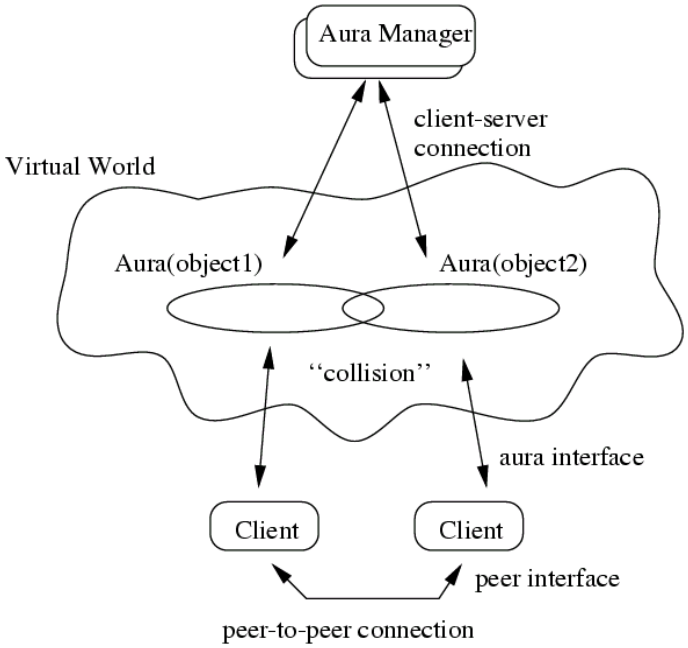
compressie	niet bekend
aggregatie	nee
interest management	nee
dead reckoning	nee
level of details	nee

Ieder object in de virtuele wereld heeft een aura voor ieder medium (beeld, geluid, tekst, ...) waarin dat object kan interageren. Dit bepaalt in welke mate interactie met andere objecten mogelijk is: interactie tussen twee objecten via een bepaald medium is enkel mogelijk indien de betreffende aura's elkaar raken of deels overlappen (collide), dit is een zogenaamde aura collision (zie figuur 2.2). Zulke aura collisions moeten gedetecteerd worden door het aura manager proces en de desbetreffende objecten moeten geïnformeerd worden, zodat ze peer-to-peer connecties kunnen opzetten.

Het bewustzijn (awareness) van het ene object over een ander object bepaalt de betekenis van het andere object in een gegeven medium. Het bewustzijn moet wederkerig zijn voordat er interactie kan gebeuren, dit brengt met zich mee dat er verschillende niveaus van bewustzijn overeen gekomen worden tussen objecten. Hiervoor worden de termen focus (waar de waarnemer zijn aandacht op rust) en nimbus (de plaatsen vanwaar een object waarneembaar is) gebruikt.

Alle communicatie is gebaseerd op point-to-point connecties tussen twee interfaces. Zo kunnen er verschillende soorten transacties over dezelfde verbinding gebeuren. Het nadeel is echter dat zulke point-to-point connecties enkel gepast zijn voor één enkel request-reply paar. Dit maakt ook de scalability voor kritiek vatbaar, want multicasting en broadcasting worden niet ondersteund.

Bij grote virtuele werelden zal de aura manager voor een bottleneck zorgen. De ervaring bij het testen leert dat MASSIVE tot ongeveer tien deelnemers kan ondersteunen via het internet en hiermee halen de ontwikkelaars niet de doelstelling om grootschalig te zijn.



Figuur 2.2: Aura collision in MASSIVE. bron: [30]

MASSIVE-2

MASSIVE-2 gebruikt het spatiale model van zijn voorganger, maar breidt het uit met third party objects, onafhankelijke objecten die het bewustzijn van andere objecten beïnvloeden. Dit beïnvloeden kan op twee manieren gebeuren:

Adaptation vergroten of verminderen van de awareness.

Secondary Sourcing een algemeen zicht op een *groep* objecten.

Nu kunnen objecten gegroepeerd worden per gebied, kamer, ... Het structureren gebeurt dynamisch en hier kan dan multicast [28, 29] zijn intrede doen. De structuur waarin de objecten gegroepeerd zijn, beïnvloeden het wederzijdse bewustzijn en het uitwisselen van informatie. De groepen kunnen op één of meer multicastgroepen gemapt worden (voor verschillende media: ook in MASSIVE-1 heeft ieder medium zijn eigen aura, focus en nimbus). Dit model is nu dus bruikbaar voor IP-multicast, hetgeen MASSIVE-2 meer scalable maakt dan de eerste versie.

Tabel 2.5: MASSIVE-1 in de taxonomie.

aantal deelnemers	10
bandbreedte	geen concrete cijfers
distributiemethode	unicast
datamodel	replicated homogeneous world database
access control	geen
updatemodel	copying on demand

Compensatory techniques

compressie	niet bekend
aggregatie	nee
interest management	ja
dead reckoning	nee
level of details	nee

2.2.4 mWorld

mWorld [45] is een 3D multi-user CVE systeem, ontwikkeld voor real-time 3D bewerkingen en animatie: gebruikers mogen door de omgeving bewegen, maar ook de belichting veranderen, objecten vastpakken, veranderen, toevoegen en verwijderen. Het is gebaseerd op het Joint Editing Service Platform (JESP), gesteund door de Europese Unie. mWorld bestaat uit OpenInventor objecten van Silicion Graphics Inc. (SGI) en is compatibel met VRML 1.0.

De session control en data communication service naar applicaties worden verzorgd door JESP:

De session control dient voor het ondersteunen van de Quality-of-service (QoS) en de consistentie wordt gecontroleerd door een token passing mechanisme. Dit mechanisme bepaalt wie er veranderingen mag aanbrengen aan de scène, namelijk enkel de gebruiker die het token bezit. Hierna worden die veranderingen dan doorgestuurd naar de rest. Een andere gebruiker moet het token aanvragen bij de huidige eigenaar, en kan het ook enkel van hem krijgen.
Hierlangs worden ook nieuwe deelnemers aan een al bestaande sessie toegevoegd, gedistribueerde applicaties opgeroepen, exceptions en failures afgehandeld en de taken verdeeld.

Bij de mWorld architectuur beheert iedere gebruiker een lokale kopie van de scène. Om de consistentie te bewaren, wordt er gebruik gemaakt van event-driven algoritmes, zonder dat er een globaal tijd systeem nodig is. De architectuur werkt bij de meeste best-effort netwerk protocols.

Tabel 2.6: MASSIVE-2 in de taxonomie.

aantal deelnemers	60
bandbreedte	geen concrete cijfers
distributiemethode	multicast
datamodel	replicated homogeneous world database
access control	geen
updatemodel	copying on demand

Compensatory techniques

compressie	niet bekend
aggregatie	nee
interest management	ja
dead reckoning	nee
level of details	nee

mWorld bestaat met JESP uit een gelaagde architectuur, hetgeen aanpassingen (en dus verbeteringen) vergemakkelijkt. De netwerkbelasting is echter hoog: een meting bij een JESP sessie waar twee deelnemers door een VE navigeerden gaf gemiddelde bit rates van 50,12 Mbit/s op een ATM netwerk. Dit maakt mWorld enkel bruikbaar voor een klein aantal gebruikers.

2.2.5 Spline

Aan het Mitsubishi Electric Research Laboratory is het Scalable Platform for Large Interactive Networked Environments (Spline) [71, 4] ontwikkeld. Spline gebruikt een object georiënteerde databank dat van ieder object zijn locatie en eigenschappen bijhoudt. Veranderingen aan een object mogen enkel door de „eigenaar" gebeuren, en het bezit mag doorgegeven worden van één proces naar een andere. Voor het weergeven van de VE, levert de databank een momentopname aan de applicatie. Spline werd gebruikt om het Diamond Park te ontwikkelen [71].

De latency wordt laag gehouden doordat met ieder proces er een kopie van het wereldmodel gemaakt wordt (zie figuur 2.3). Bij veranderingen in het wereldmodel worden naar de anderen de volledige nieuwe toestand van het wereldmodel-object gestuurd. Zoals bij SIMNET en DIS bevat een bericht de nieuwe toestand van het veranderde object *in zijn geheel*, dit in tegenstelling tot enkel *de verandering zelf* (zogenaamde incremental updates). De berichten worden via een betrouwbaar multicast schema rondgestuurd, in tegenstelling tot het onbetrouwbare IP-multicast dat gebruikt wordt door onder andere NPSNET (zie

Tabel 2.7: mWorld in de taxonomie.

aantal deelnemers	een klein aantal, geen concrete cijfers
bandbreedte	50,12 Mbit/s bij twee deelnemers
distributiemethode	multicast simulation via JESP, eigenlijk unicast [26]
datamodel	shared distributed databases
access control	gedecentraliseerd met token passing
updatemodel	active copying

Compensatory techniques

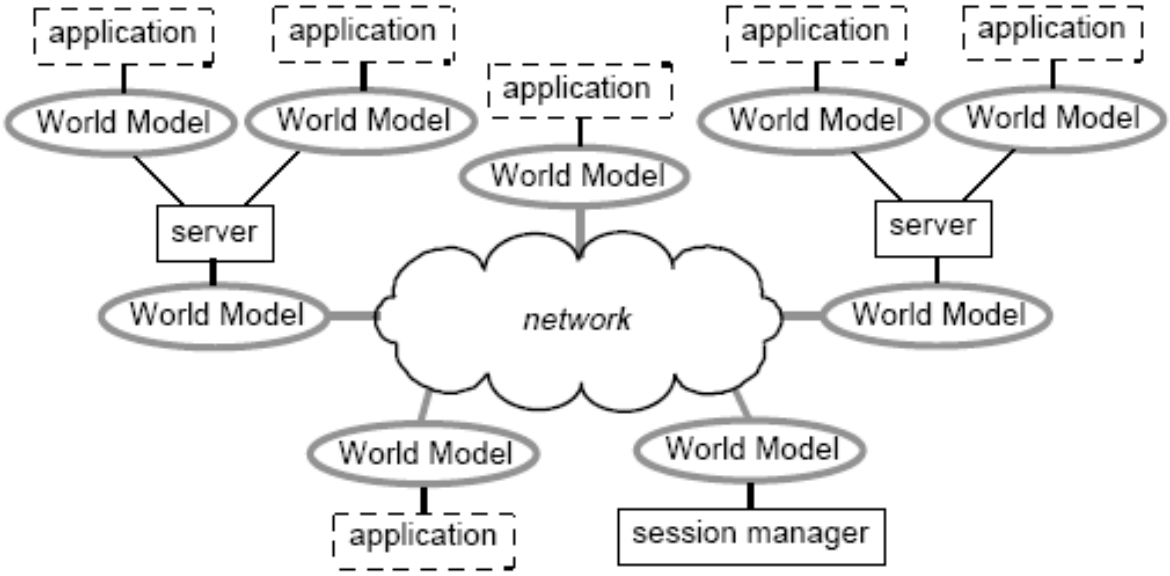
compressie	niet bekend
aggregatie	nee
interest management	nee
dead reckoning	nee
level of details	nee

2.2.1). Andere opvallende kenmerken zijn de mogelijkheid tot het toevoegen van nieuwe objecten at run time, de audio en video rendering en de mogelijkheid om het bezit van objecten door te geven.

Er zijn drie soorten data die via berichten moeten rondgestuurd worden in Spline: kleine objecten die eenvoudig en snel kunnen veranderen, grotere objecten die amper veranderen (graphics, geluid, gedrag) en continue datastromen. De veranderingen in de kleine objecten en de datastromen worden via UDP gestuurd. De grotere objecten hebben een URL en hier kunnen de standaard WWW protocollen gebruikt worden.

Spline probeert een groot aantal deelnemers te ondersteunen door ten eerste niet te eisen dat de kopieën van de wereldmodellen volledig hetzelfde moeten zijn. Verschillen worden toegestaan, omdat er geen incremental updates worden gebruikt, zullen die met verloop van tijd toch worden opgelost. Ten tweede wordt het wereldmodel opgedeeld in kleinere stukken of *locals*, zo kan iedere local geassocieerd worden met een multicastgroep. Deze visie is gebaseerd op het idee dat iedere gebruiker op een bepaald moment enkel lokale zaken kan waarnemen: „what a single user can observe at a given moment is local in nature." [45]

Spline gebruikt een combinatie van point-to-point client/server communicatie en peer-to-peer multicast communicatie tussen servers onderling. Servers staan in voor het beheer van de locals en voor het cachen. Spline heeft niet één centraal proces, maar gebruikt verschillende gespecialiseerde, gecentraliseerde processen zoals de servers (beheer van locals en cachen) en session managers.



Figuur 2.3: Het Spline model. bron: [71]

2.2.6 DIVE

Het Distributed Interactive Virtual Environment of *DIVE*[9] [5, 25, 64], een software platform voor multi-user VE's, is ontwikkeld aan het *Swedish Institute of Computer Science* of *SICS* te Stockholm. Sinds 1991 is DIVE verscheidene veranderingen ondergaan en nu is men al toe aan versie 3.3x[10]. DIVE is een volledig gedistribueerde VE en bestaat uit processen die tegelijkertijd draaien op verschillende nodes over het netwerk. Deze processen beheren elk een kopie van (een deel van) de databank die de virtuele wereld voorstelt. Het concept van DIVE kan dus gezien worden als een geheugen dat gedeeld wordt over het netwerk.

Een deelnemer in DIVE, een actor genaamd, wordt voorgesteld door een avatar en is ofwel een representatie van een (menselijke) gebruiker, ofwel een geautomatiseerd applicatie proces. Objecten in de virtuele wereld kunnen zowel statisch (de vloer en de muren) als verplaatsbaar zijn (blokken, stoelen, tools, ...). Gebruikers kunnen interageren met objecten door ze te creëren, te verwijderen of aan te passen. Gebruikers communiceren onderling door tekst en audio. DIVE kan gebruikt worden met een (al dan niet immersive) interface, met een VRML interface op de gewone desktop pc of met CAVE (zie sectie 2.2.6,

[9] http://www.sics.se/dive/
[10] http://www.sics.se/dive/distr/v3.3x/

Tabel 2.8: Spline in de taxonomie.

aantal deelnemers	geen concrete cijfers
bandbreedte	geen concrete cijfers
distributiemethode	multicast (Scalable Reliable Multicast of SRM) tussen de servers en unicast voor client/server communicatie
datamodel	shared distributed databases met peer-to-peer updates
access control	gedecentraliseerd, hosts "bezitten" een object
updatemodel	migration / partial copying

Compensatory techniques

compressie	niet bekend
aggregatie	nee
interest management	ja (opdeling in locals)
dead reckoning	nee
level of details	nee

Spelunking). Voor het dynamisch gedrag van objecten te bepalen, worden Tcl scripts gebruikt. De scripts worden geactiveerd door events in het systeem, zoals interactie met de gebruiker, timers, collisions, ...

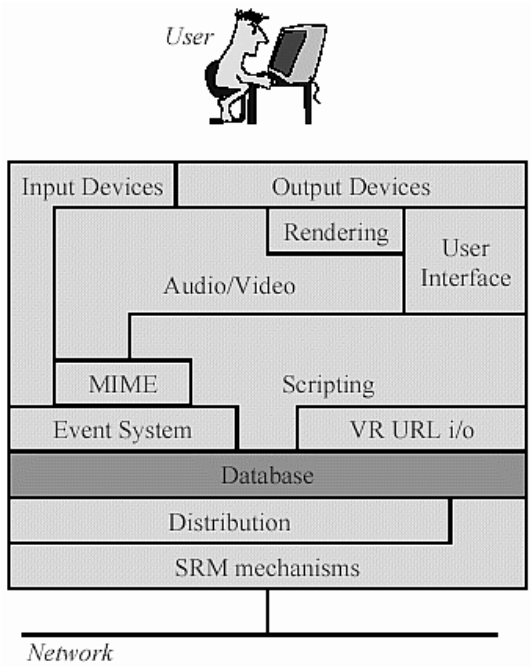
Toepassingen van DIVE zijn onder andere militaire simulaties, ruimtelijke interactiemodellen, virtuele agenten, real-world robot control en multi-modal interaction.

Op conceptueel en programmeerniveau is DIVE (zie figuur 2.4) gebaseerd op een hiërarchische database van objecten, entities genaamd. Deze databank wordt dus deels gekopieerd op alle nodes die deelnemen aan een sessie.

Om de verschillende kopieën van (delen van de) databank consistent te houden, is er netwerkverkeer nodig. De updates worden via multicasting rondgestuurd en bevatten een volledige beschrijving van de events die de updates hebben veroorzaakt, zodat andere hosts die een kopie van de betrokken entities bijhouden de events zonder ambiguïteit kunnen uitvoeren.

DIVE moet een systeem hebben om het multicastverkeer over het netwerk gecontroleerd te laten verlopen. Hiervoor maakt het gebruik van een variatie op het scalable reliable multicast of SRM [23, 25].

Zoals eerder vermeld gebeurt het verspreiden van informatie in DIVE in het algemeen via multicasting, maar niet elk netwerk ondersteunt multicast. Om de multicast-eilandjes met



Figuur 2.4: De verschillende componenten van het DIVE systeem. bron: [64]

elkaar te verbinden, werd DiveBone[11] ontwikkeld. Netwerken die geen multicast ondersteunen hebben dan een proxyserver nodig die naar multicastgroepen luistert en de nodige packetjes doorstuurt via unicast.

Voorbeelden van projecten rond DIVE zijn:

London Travel, een COVEN project

Het COVEN project[12] [52] Londen Travel [63] is ontwikkeld aan de University College London en aan het SICS te Stockholm, om virtuele bezoeken aan de stad Londen te brengen. De gebruiker kan door een virtueel model (met een grootte van 160 km^2) van Londen routes plannen, toeristische informatie aanvragen en andere reizigers of gidsen ontmoeten.

[11] alhoewel de doelstelling hetzelfde is, is DiveBone onafhankelijk van MBone. De DiveBone tools kunnen wel gebruikt worden voor het verbinden van subeilandjes van MBone die slecht of niet geconnecteerd zijn, http://www.sics.se/dive/divebone/

[12] http://www.cs.ucl.ac.uk/research/vr/Coven

Spelunking

Spelunking [64] is ook een project aan de University College London en aan het Swedish Institute of Computer Science of SICS te Stockholm dat gebruik maakt van het DIVE systeem. De klemtoon bij Spelunking ligt op het uitbreiden van DIVE zodat het het gebruik van immersive projection technologies of IPTs (zoals CAVE) ondersteunt. Dit zou beschikbaar moeten zijn in versie 3.4x van DIVE.

Tabel 2.9: DIVE in de taxonomie.

aantal deelnemers	16 - 32
bandbreedte	geen concrete cijfers
distributiemethode	multicast
datamodel	shared distributed databases met peer-to-peer updates
access control	gedecentraliseerd
updatemodel	migration / partial copying

Compensatory techniques

compressie	ja (gzip)
aggregatie	ja
interest management	nee
dead reckoning	nee
level of details	nee

2.3 Moderne Large Scale Networked Virtual Environments

De nieuwe generatie NVE's doelt op massapopulatie. Om een groot aantal gebruikers te kunnen ondersteunen, is er veel vooruitgang nodig op het vlak van scalability (zie het volgende hoofdstuk). Hiervoor zijn er een aantal nieuwe technieken onderzocht en gebruikt. Een aantal van deze NVE's worden in deze sectie besproken.

2.3.1 Urbi et Orbi

Urbi et Orbi [70] is een project dat ontwikkeld werd aan het Franse EPITA Research and Development Laboratory, of het LRDE. Het is een distributed environment framework om virtuele werelden mee op te bouwen. Aan het LRDE wordt er echter niet meer aan dit

project gewerkt[13].

Net zoals bijvoorbeeld MASSIVE (zie sectie 2.2.3) en DIVE (zie sectie 2.2.6), is Urbi et Orbi volledig gedistribueerd, er wordt geen gebruik gemaakt van een client-server hiërarchie. De informatie over de (huidige toestand van de) wereld wordt verspreid over alle deelnemers.

Het experimenteren met Urbi et Orbi gebeurde op een Fast Ethernet LAN (100 Mbps) met personal computers die elk een 3D videokaart hadden. Gemiddeld werden er zo een 25 frames per seconde gehaald met hoge kwaliteitsbeelden en een uitstekende interactiviteit. Urbi et Orbi maakt gebruik van de functionele programmeertaal Objective Caml (zie Goal in sectie 2.3.1) en maakt veel gebruik van scripts.

High Level Scene Description

Een belangrijk kenmerk van Urbi et Orbi, is zijn high level scene description. Dit houdt in dat er gebruik wordt gemaakt van high-level beschrijvingen van wereldcomponenten. Zo bestaan bossen uit bomen, waarvan er verschillende soorten bestaan; objecten zoals huizen hebben een binnenkant en een buitenkant, ... Virtuele werelden worden abstract beschreven.

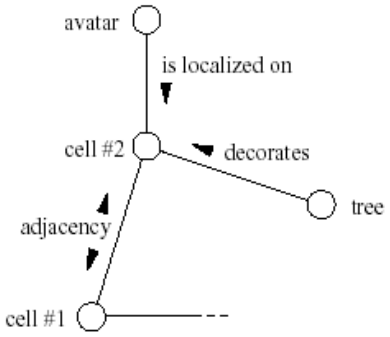
Hierdoor kan elke terminal de wereld voorstellen, onafhankelijk van het type (tekstueel, met 3D rendering, ...) . Telkens kan men met de omgeving en de andere gebruikers interageren. De ontwerpers van Urbi et Orbi hebben zelf gebruik gemaakt van een tekstuele terminal als shell voor het programma en een 3D engine met OpenGL.

De abstractie heeft ook een voordeel voor het renderen van de graphics. Er kunnen verschillende perspectieven worden voorzien. Voor een bos kan dat lopen van een groene vlek tot een volledige, gedetailleerde rendering van een aantal verschillende soorten bomen. Als er aan de horizon een bos te zien is, is het niet nodig om alle bomen te renderen, maar volstaat de groene vlek. De gangbare oplossing is het gebruik van level of details (LOD, zie ook sectie 3.3.5) gegenereerd door mesh simplification: de objecten worden afhankelijk van de afstand tot de waarnemende gebruiker meer of minder gedetailleerd gerenderd.

De methode gebruikt in Urbi et Orbi heeft als nadeel dat er verschillende perspectieven voorzien moeten worden, maar vereist geen mesh berekeningen zoals bij de LOD-aanpak. Ook is het niet zeker of de gebruiker wel over een terminal met 3D rendering beschikt. De terminal hoeft enkel te weten dat het hier over het type „bos" gaat.

[13] http://www.lrde.epita.fr/cgi-bin/twiki/view/Projects/UrbiEtOrbi

Tussen de objecten in de wereld moeten er relaties gelegd worden. Hiervoor werden conceptual graphs[14] gebruikt met spatiale relaties zoals „staat op" (is on), „grenst aan" (is adjacent to), ... en niet-spatiale relaties zoals „bestaat uit" (is composed of), „activeert" (activates), ...



Figuur 2.5: Het gebruik van conceptual graphs bij Urbi et Orbi. bron: [70]

Figuur 2.5 geeft een voorbeeld van het gebruik van dit soort graphs. Hier worden twee cellen (bepaalde zones in de virtuele wereld) voorgesteld. Dit voorbeeld gebruikt drie relaties. De link tussen cel 1 en cel 2 duidt aan dat men vanuit de ene cel de andere kan bereiken, *localized on* duidt op de aanwezigheid van een avatar in cel 2 en de relatie *decorates* wilt zeggen dat er een boom in cel 2 is.

Alhoewel de relaties *decorates* en *localized on* gelijkaardig zijn, ligt het verschil in het feit dat *decorates* te kennen geeft dat het hier slechts om bijkomende informatie gaat. Bij een eerste benadering van de cel kan de boom achterwege gelaten worden als er op dat moment bijvoorbeeld dringender updates doorgevoerd moeten worden.

Goal

Om werelden te beschrijven, wordt er in Urbi et Orbi gebruik gemaakt van een ad hoc gedefinieerde scriptingtaal GOAL, gebaseerd op Objective Caml of O'Caml[15], een dialect van Meta Language (ML). De ontwerpers kozen voor een uitbreiding van O'Caml omdat dit een functionele, strongly typed taal is die scripts kan interpreteren en eenvoudig kan interageren met andere programmeertalen (de 3D rendering in Urbi et Orbi is bijvoorbeeld geschreven in C).

[14] http://en2.wikipedia.org/wiki/Conceptual%5Fgraph, http://www.cs.uah.edu/%7Edelugach/CG/, http://www.jfsowa.com/cg/cgexamp.htm
[15] http://www.ocaml.org/, http://caml.inria.fr/

Het functionele programmeermodel (functional paradigm) is geschikt voor de transformaties die er moeten gebeuren op de gedistribueerde data. Hierdoor kunnen de delen waar imperatief programmeren noodzakelijk is geïsoleerd worden. Strong typing[16] is een vereiste voor het omschrijven van de werelden, diens objecten en hun attributen, wegens het gebruik van de high level scene description en om ambiguïteit te voorkomen.

Scriptingtalen hebben over de jaren heen al lang bewezen dat ze zeer handig zijn voor systeemuitbreidingen en programmer interfaces (zoals shell scripts, TCL, ...). Scripting zorgt onder andere voor eenvoudige prototyping, dynamische uitbreidingen en aanpassingen. Zo hoeft de software voor een simpele aanpassing niet opnieuw gecompileerd en geïnstalleerd worden. Dit komt zeker van pas bij gedistribueerde systemen. O'Caml voorziet zowel script interpretation, byte-code compilation[17] en native compilation.

Bovendien is Ensemble (zie hieronder) ook geprogrammeerd in O'Caml.

Ensemble

Voor het netwerkingsgedeelte wordt Ensemble [32] gebruikt. Het is een modulaire, herconfigureerbare toolkit[18], ontwikkeld aan de Cornell University in samenwerking met de Hebrew University of Jerusalem, geprogrammeerd in Objective Caml en ondersteunt momenteel applicaties ontwikkeld in ML, C, C++ en Java.

De toolkit levert high-level protocols aan het programma. Deze protocols bestaan uit een aantal verschillende lagen die elk zijn eigenschap (zoals het behouden van de volgorde van de packetjes, beveiliging, synchronisatie, ...) heeft. De lagen kunnen individueel aangepast worden, dit maakt Ensemble een zeer flexibel platform.

Het eerste prototype van Urbi et Orbi was in Java geïmplementeerd en de virtuele werelden werden beschreven in VRML. De combinatie van Java, VRML en een Object Request Broker (ORB) voor het distribueren van de data, betekende echter een grote overhead. Er kwamen ook problemen bij het simultaan verwerken van de communicatie, de display en de gebruikersinteractie. Volgens de ontwikkelaars kan VRML ook geen extra object-attributen zonder geometrische semantiek of complexe relaties tussen objecten aan.

De ontwerpers zijn geen voorstanders voor het gebruik van deze standaarden en prefereren hun eigen ad hoc scriptingtaal. Dit wilt echter niet zeggen dat het gebruik van standaarden een slechte zaak is, maar wel dat er nog verbetering nodig is.

[16] het strikt naleven van de type regels, zonder uitzonderingen
[17] zoals bijvoorbeeld bij de programmeertaal Java en zijn Java Virtual Machine
[18] http://www.cs.cornell.edu/Info/Projects/Ensemble/index.htm

Tabel 2.10: Urbi et Orbi in de taxonomie.

aantal deelnemers	geen concrete cijfers
bandbreedte	zonder problemen getest op een Fast Ethernet LAN (100 Mbps)
distributiemethode	IP multicast voor onbelangrijke data, reliable multicast voor kritieke data
datamodel	shared distributed databases met peer-to-peer updates
access control	gedecentraliseerd
updatemodel	copying on demand

Compensatory techniques

compressie	niet bekend
aggregatie	niet bekend
interest management	ja (verschillende cellen)
dead reckoning	nee
level of details	ja (geen mesh simplification maar verschillende modellen)

2.3.2 vGrid

Bij het ontwerpen van een NVE komen er allerlei problemen aan te pas, zo moet er onder andere rekening gehouden worden met het feit dat alle interactie real-time moet gebeuren, de latency op het netwerk minimaal gehouden wordt, de applicatie kan functioneren op een grote diversiteit aan systemen (zowel wat hardware als besturingssysteem betreft), ... (zie de aandachtspunten in sectie 3). Dit zijn allemaal moeilijke problemen waarvoor er nog steeds geen algemene oplossing is gevonden, dus zoekt men specifieke oplossingen voor concrete applicaties, omgevingen, data, ...

Daarom zijn huidige VE's vaak geconstrueerd voor één specifiek doel, hebben ze maar een klein bereik en zijn ze domeinspecifiek. NVE's zijn ook duur en moeilijk om te construeren, bovendien is er nog steeds geen standaard mechanisme voor het opzoeken en het verbinden met een persistente virtuele omgeving, noch een standaard namespace voor het aankondigen en het zoeken van VE's[19].

Het creëren van een virtuele ruimte zou zo eenvoudig moeten zijn als het aanvragen van een vergaderzaal. Ideaal zou zijn dat de gebruiker slechts een aantal criteria moet ingeven (zoals de soorten data die gebruikt zullen worden, de mensen waarmee men wil samenwerken, ...) zodat de VE die aan deze criteria voldoet gezocht of automatisch geconstrueerd kan worden.

[19]Een uitzondering is de VRML-standaard die gebruik maakt van de URL namespace.

Het project vGrid [69], ontwikkeld aan de Australian National University in Canberra, heeft als doel om van virtuele omgevingen een algemeen medium, een general-purpose tool voor communicatie en collaboratie te maken[20] in plaats van ze enkel te gebruiken voor gespecialiseerde (visuele) opdrachten. Het biedt een basis voor het integreren van een brede waaier aan verschillende datatypes en clientapplicaties en tracht de diversiteit te beheren en de uitvoering van VE's te ondersteunen. vGrid is een framework dat gebruik maakt van grid computing[21].

Een computational grid[24] is een hard- en software infrastructuur die betrouwbare en consistente toegang biedt tot high-end computing vermogens. Het is een meta-computing omgeving die de nodige diensten en meta-data voorziet om veeleisende problemen via een samenwerking van (super)computers op te lossen.

Een bekende grid is bijvoorbeeld seti@home[22] van de Berkely University in California dat gebruikt wordt in het SETI-project voor onderzoek naar het bestaan van buitenaards intelligent leven. Gebruikers kunnen op de website een programma downloaden. Als de computer van de gebruiker weinig resources gebruikt[23], dan staat die zijn overige potentiële rekenkracht af aan het project om data van een radiotelescoop te analyseren. Op het moment van schrijven zijn er in totaal, sinds 1999, al bijna vijf miljoen computers gebruikt en is de rekenkracht zo een 68 teraflops per seconde.

vGrid deelt de netwerkomgeving op in „sites". Een site is een verzameling gebruikers die een gemeenschappelijk (administratief) domein delen. Binnenin zo een site wordt aangenomen dat men via een (lokale) hoge snelheidsverbinding kan communiceren. Alle sites zijn impliciet met elkaar verbonden door het Internet. Het Internet is echter een netwerk dat slechts een beperkte quality of service biedt, daarnaast kunnen sites dan nog eens verbonden zijn via gespecialiseerde netwerken die een betere service garanderen.

vGrid bestaat uit een directory service[24] die een grote hoeveelheid aan meta-data over de systeembronnen (zoals de machines, de datasets en de netwerken) en de informatie over de virtuele werelden beheert. Het wordt dus gebruikt als opslagplaats voor meta-data die de computing environment beschrijft. Daarnaast voorziet de directory service ook een namespace waarin de servers de systeembronnen die ze aanbieden kunnen bekendmaken. Zodoende kunnen de clients erin op zoek gaan naar de systeembronnen die ze nodig hebben.

[20] Dit in sterk contrast met Urbi et Orbi, die een ad hoc scriptingtaal gebruiken, terwijl vGrid net een algemene standaardoplossing probeert te ontwikkelen.

[21] http://en.wikipedia.org/wiki/Computational%5Fgrid

[22] http://setiathome.berkeley.edu/

[23] dit wordt CPU scavenging genoemd

[24] zoals het Lightweight Directory Access Protocol (LDAP), zie ook NPSNET-V in sectie 2.2.1

Niet alle resources zijn echter duidelijk te identificeren, bijvoorbeeld de dynamisch gegenereerde of transient[25] bronnen. Deze bronnen kunnen daarom niet opgenomen worden in de public name space maar worden aangesproken via **Brokers**. Brokers worden ook gebruikt als er resource management, reservatie en allocatie policies nodig zijn. Brokers voorzien toegang tot resources via een handle-like mechanisme hetgeen een PROMISE (Belofte) genoemd wordt. Het verkrijgen van een Promise is hetzelfde als een resource reservation, het evalueren van een Promise hetzelfde als de toewijzing van een resource.

Als een verkregen Promise na een bepaald tijdsinterval nog altijd niet geëvalueerd is, dan vervalt de Promise en wordt de systeembron niet toegekend. Wordt de Promise wel geëvalueerd, dan is deze een handle, een „handvat" voor de Broker naar de resource. Deze blijft toegewezen totdat die expliciet gedesalloceerd wordt.

Er worden twee soorten Brokers gebruikt, data Instance Brokers en protocol Adaptor Brokers:

Instance Brokers worden gebruikt om toegang te verschaffen tot individuele instances van een data type. Deze Brokers worden gebruikt om het beheer van en de toegang tot resources aan te spreken. Alle data sets worden beheerd door een Instance Broker, zowel statische als dynamisch gegenereerde content. Het is via deze Broker dat clients onderhandelen over de toegang tot een bepaalde data set. Er zijn twee soorten Instance Brokers: de grote data opslagplaatsen die de data instances statisch bijhouden en de simulation process managers die toegang verschaffen tot dynamisch gegenereerde data.

Elke Instance Broker definieert ook een verzameling protocols om te gebruiken voor het exporteren en het delen van de data instances. Het aantal protocols kan beperkt zijn of het is mogelijk dat de client applicatie andere protocols gebruikt om zich te verbinden met de virtuele wereld. Dan worden er ADAPTORS gebruikt om een brug te maken tussen twee of meer protocollen -dit kan in één of in beide richtingen gebeuren- om het sharen van de instances toch mogelijk te maken. Omdat Adaptors uniek zijn voor een bepaalde data instance binnen een virtuele wereld, wordt de toegang geregeld door **Adaptor Brokers**. Als Adaptors eenmaal gealloceerd zijn, worden ze geregistreerd in de directory als deel van een bepaalde wereld.

Client applicaties zouden rechtstreeks in de directory service opzoek kunnen gaan naar resources en zo hun eigen werelden creëren of betreden. Het is echter niet altijd praktisch om clients bewust te maken van hoe de vGrid nu precies werkt. Daarom maakt men gebruik van **Agents**, die in naam van de client dan gebruik maken van de directory service. Ze voorzien een eenvoudige interface naar de vGrid infrastructuur en verbergen zo onnodig detail en complexiteit, hetgeen ten goede komt voor de Quality of Service en de distribu-

[25] een software object dat maar kort bestaat en niet wordt opgeslagen voor hergebruik

tion en network transparency. Zo staat de Connection Agent in voor het verbinden van de client met virtuele werelden en de World Manager Agent voor het creëren of wissen van werelden. De vGrid infrastructuur blijft verborgen voor de gebruiker, zo heeft deze laatste voldoende aan de naam van de virtuele wereld om er een actie op uit te voeren.
Het verschil tussen Brokers en Agents, is dat Brokers elementen van de vGrid zijn en aangeduid staan in de directory. Agents daarentegen zijn niet publiekelijk bekend en zijn een product van een bepaalde site of applicatie.

2.4 De multiplayer games en de MMORPGs

Na het Amerikaans leger en het academisch onderzoek naar NVE's, is het tijdperk van de games aangebroken. Tegenwoordig kan een spel niet meer overleven zonder de multiplayeroptie.

Terwijl het Amerikaanse Department of Defense (het DoD) de basis heeft gelegd voor de genetwerkte virtuele omgevingen met gespecialiseerde image generators, dure hardware en aangepaste software implementaties, heeft er zich een parallelle ontwikkeling voortgedaan op het gebied van de computerspelletjes, de games. Zelfs het DoD erkent dat de games nu het grootste gebruik van genetwerkte omgevingen betekenen en dat de technologie van de game-developers die van de Amerikaanse Defensie al een aantal jaren is voorbijgestreefd [13].

Door de commerciële belangen van ieder bedrijf dat games ontwikkelt, is het echter zeer moeilijk om bruikbare informatie te vinden over welke technologie er gebruikt wordt. De grote online games maken meestal gebruik van een reeks servers (al dan niet hiërarchisch gestructureerd) verspreid over de wereld, om het grote aantal gebruikers te ondersteunen.

2.4.1 Multiplayer shoot'em ups

Op 10 december 1993 bracht id Software de first person shooter *Doom* op de markt, een spel waarbij de speler losgelaten wordt in een 3D wereld en als doel heeft alles neer te schieten dat hij of zij onderweg tegenkomt. Het eerste level van Doom werd door id Software vrijgegeven op het internet.

In single player modus kan de speler zich te goed doen aan de monsters die rondlopen in het spel, maar Doom had ook een multiplayer functie, zodat spelers op elkaar jacht konden maken. Met 15 miljoen downloads en 250.000 geregistreerde gebruikers, is duidelijk dat Doom een hit was en dit was dan ook het begin van een grote bloei in interesse in 3D

shootem ups, spelletjes als shareware en, vooral, multiplayer games.

Als Doom op een LAN gespeeld wordt, kunnen tot maximaal vier spelers tegen elkaar spelen. Het spel gebruikt geen compressietechnieken of dead reckoning, maar stuurt packets over het LAN tegen framerate. Telkens als het scherm hertekend moet worden, wordt er een packet verstuurd, zodoende wordt het LAN overspoeld.

Na Doom kwam het onvermijdelijke Doom II. De volgende stap in het multiplayer gebeuren werd ook genomen door id Software, in 1996 kwam *Quake* en in 1997 *Quake II* op de markt. Naast een enorme vooruitgang in 3D graphics, dankzij een nieuwe graphics engine, en de mogelijkheid om echt in elke richting te kijken, was het nu ook mogelijk om een multiplayer game op te starten tot 32 deelnemers.

2.4.2 De MMORPGs

De *MMORPGs* of „Massive Multiplayer Online RolePlaying Games" zijn de nieuwe hype op internet. Het eerste succesvolle grote internetgame was *Ultima Online*[26], een groot client-server spel van Electronic Arts uit 1997. Duizenden spelers kunnen inloggen op verscheidene gameservers (shards) om tegelijk deel te nemen aan een fantasy wereld genaamd *Britannia*. Het aantal leden loopt in de honderdduizenden.

In 1999 kwam Sony met Everquest[27] van Verant Interactive op de markt en dit is momenteel één van de populairste online games. Deze MMORPG bestaat uit een reeks servers die elk een onafhankelijke, persistente wereld hosten en iedere server kan tot 3000 gelijktijdige gebruikers ondersteunen. Voor 2004 verwacht Sony een ledenaantal van 450.000.

Het grootste aantal gebruikers ligt echter bij Lineage[28] dat in 1998 door NCsoft uitgebracht werd. Er zijn meer dan 4 miljoen gebruikers, voornamelijk Zuid-Koreanen.

[26] http://www.uo.com/
[27] http://www.everquest.com/, http://everquest.station.sony.com/
[28] http://www.lineage.com/

2.5 Conclusie

In dit hoofdstuk werden een aantal van de meest bekende virtual environments toegelicht en de ontwikkeling weergegeven van de grote ontwerpen aan het US Department of Defense (SIMNET en DIS) tot de academische architecturen (zoals NPSNET en DIVE) en de populaire, commerciële multiplayer games.

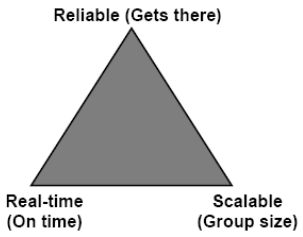
Er werd een overzicht gegeven van de belangrijkste ontwikkelingen om de mogelijkheden te beschrijven. Hiermee kan rekening gehouden worden voor de komende ontwerpen. Voor meer gedetailleerde informatie wordt er doorverwezen naar de werken:

- voor genetwerkte simulatoren: [47, 40], [19, 20, 33]
- voor de academische NVE's: [42, 14], [30, 29, 28], [71, 4], [5, 25], [70], [69]
- voor computer games: [60, 61]

Hoofdstuk 3

Taxonomie, Scalability en Compensatory Techniques

De ideale NVE zou zowel veel gebruikers moeten kunnen ondersteunen als lage eisen hebben in verband met netwerking. Alles zou real-time moeten gebeuren op een betrouwbare manier. Momenteel zijn deze criteria echter tegelijkertijd nog niet haalbaar (zie figuur 3.1), in de huidige realiteit sluit het ene vaak het andere uit. De oplossing is een product van veel afwegen en beslissingen nemen.



Figuur 3.1: Drie criteria voor de perfecte NVE: Scalability en real-time reliable delivery. bron: [62]

3.1 Taxonomie voor genetwerkte virtuele omgevingen

In hun werk „A Taxonomy for Networked Virtual Environments" [41] omschrijven Michael R. Macedonia en Michael J. Zyda een opdeling voor de soorten genetwerkte virtuele omgevingen. Dit geeft een beter overzicht van de mogelijkheden.

3.1.1 Netwerkcommunicatie

De netwerkcommunicatie is een belangrijk deel van NVE's. De belangrijke kenmerken zijn het bandbreedteverbruik, de latency, de distributiemethode en de reliability.

Bandbreedteverbruik

De beschikbare bandbreedte bepaalt de grootte en de mogelijkheden (*richness*) van een virtuele omgeving. Als het aantal deelnemers verhoogt, is er ook meer bandbreedte nodig. Het gebruik van video, audio, 3D graphics modellen, ... in real-time vereist veel bandbreedte. Door de verscheidenheid aan data zijn nieuwe protocollen en technieken nodig om de data gepast te behandelen over een netwerkverbinding.

Bandbreedte wordt gemeten in bits per seconde (bps).

Een tekort aan bandbreedte kan men op verschillende manieren oplossen, eerst en vooral door het investeren in een lijn met veel bandbreedte. Men kan ook een extra lijn aanleggen en de lijnen dan parallel gebruiken (zoals bijvoorbeeld ISDN bonding).

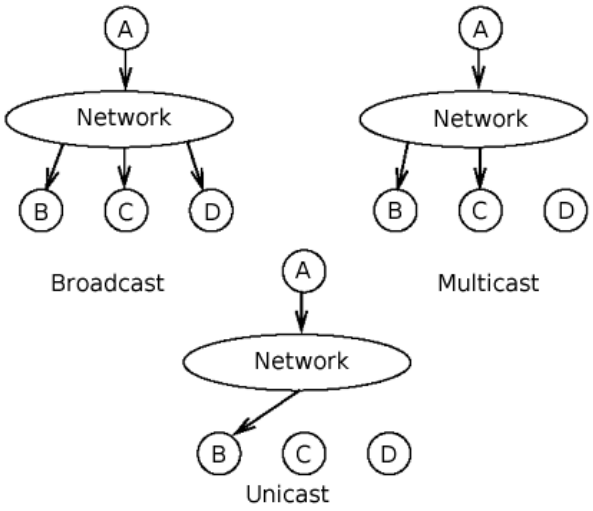
Men kan er echter niet van uitgaan dat iedere gebruiker bereid is zich een nieuwe verbinding aan te schaffen, zeker niet als de NVE over het huidige Internet zou moeten werken. Er zijn technieken nodig die het bandbreedteverbruik beperken. Bij het ontwerpen van de NVE kan er **code** gebruikt worden die zuinig omspringt met de beschikbare bandbreedte door bijvoorbeeld een kleinere packetsize te gebruiken als dat mogelijk is.

Een tweede mogelijkheid is het **beperken** van de data die wordt doorgestuurd. Voor textures of andere tekeningen en foto's kan men een lagere resolutie, geïndexeerde kleurenschema's of zelfs zwart-wit beelden gebruiken als dat nodig is. Om de virtuele omgeving zo realistisch mogelijk te maken voor de (menselijke) gebruikers, moeten de 3D graphics gegenereerd worden tegen een snelheid van 30 tot 60 beelden per seconde. In plaats van de 60 beelden per seconde proberen te halen, kan men hier dus wat terugschroeven als het bandbreedteverbruik te gretig wordt. Het blijft altijd afwegen tussen het realiteitsgevoel en het verbruik van de resources.

Andere technieken, zoals **compressie** en **dead reckoning** worden uitgelegd in sectie 3.3.

Distributiemethodes

Bepaalde distributiemethodes (zie figuur 3.2) zijn beter voor grootschalige NVE's dan andere.



Figuur 3.2: Voorbeelden van de distributiemethodes. bron: [41]

Broadcast (hardware-based of via het Internet Protocol (IP)) verzendt de data naar alle hosts binnen het netwerk. *Unicast* wordt gebruikt voor point-to-point communicatie tussen twee hosts. *Multicast* laat groepen van willekeurige grootte toe te communiceren over een netwerk met maar één transmissie door de bron.

Broadcast en unicast (zie figuur 3.2) zijn vrij inefficiënt voor het gebruik bij grote aantallen deelnemers.

Bij broadcasting wordt het netwerk overspoeld met packets en is het moeilijk om routing loops te vermijden. IP broadcast verplicht ook altijd elke host om het rondgestuurde packet te bekijken. Dit betekent een verlies aan performance als blijkt dat het packet niet voor de betreffende host bedoeld is.

Voor unicast moet er een verbinding of pad van de ene node naar alle andere nodes in het netwerk opgesteld worden. Dit geeft een totaal van $N \times (N - 1)$ virtuele connecties in een groep van N nodes.

Bij multicasting bereikt men daarentegen een selecte groep deelnemers die zich hebben ingeschreven om deze informatie te ontvangen. Men kan gecontroleerd data versturen naar een aantal (maar niet noodzakelijk alle) gebruikers en is een efficiëntere distributiemethode voor NVE's.

Het nadeel van multicast is dat (momenteel) slechts weinig Internet Service Providers IP multicast ondersteunen. Hiervoor werd MBone (zie sectie 4.1) ontwikkeld.

Daarnaast zijn IP multicastgroepen moeilijk te beveiligen en te beschermen tegen afluis-

teren (eavesdropping). De tijd die nodig is om zich aan te sluiten bij een multicastgroep (de join latency [27]) is ook een nadeel want tijdens een sessie wordt er in het algemeen regelmatig gewisseld en de join latency is nog te groot.

Latency

Met latency wordt de tijd aangeduid die nodig is om een packet van de bron naar de geadresseerde(n) te krijgen, en moet zo klein mogelijk zijn. Latency bepaalt de interactiviteit van de NVE en hoe dynamisch deze is.

Om de virtuele omgeving zo realistisch mogelijk te maken voor de (menselijke) gebruikers, moeten de 3D graphics gegenereerd worden tegen een snelheid van 30 tot 60 beelden per seconde. Om dit te bereiken moeten de netwerkpackets met minimale latency geleverd kunnen worden. De (onvermijdelijke) latency moet zo constant mogelijk blijven, de jitter[1] mag met andere woorden niet te groot worden.

Voor alle genetwerkte toepassingen is latency een zeer belangrijk aspect. Zoals al gezegd kan een tekort aan bandbreedte opgelost worden, maar latency kan nooit volledig weggewerkt worden[2].

bijvoorbeeld: de lichtsnelheid (in het luchtledige) bedraagt ongeveer $300 \times 10^6 \ \frac{m}{s}$. Tegen deze topsnelheid zal een afstand van 1 km altijd een vertraging opleveren van minstens $\frac{1.000m}{300 \times 10^6 m/s} = 3 \ ms$ per packet. Aangezien er wordt aangenomen dat men de lichtsnelheid nooit zal overtreffen, is deze 3 ms/km een vertraging waar altijd rekening mee zal moeten gehouden worden.

Bij veel jitter (zeer variabele latency) is het systeem onbetrouwbaar. Als de gebruiker een commando geeft en de reactie blijft uit, dan kan bij een systeem met veel jitter niet bepaald worden of het commando nu nog altijd niet aangekomen is, of of het commando gewoon niet uitgevoerd werd. Moet de gebruiker dan nog even wachten, of moet hij/zij het commando nog eens uitvoeren? Dit belemmert het realiteitsgevoel en is zelfs ronduit vervelend.

Reliability

De reliability (de betrouwbaarheid) van de communicatie beslaat in welke mate men kan aannemen dat verstuurde data correct aankomt. Het bepaalt hoe vaak data herverzonden

[1] het verschil in latency, hoeveel de latency varieert
[2] http://www.stuartcheshire.org/rants/Latency.html

Tabel 3.1: De taxonomie voor NVE datamodellen. bron: [41].

NVE datamodellen
1. replicated homogeneous world database
2. shared, centralized databases shared, distributed databases met peer-to-peer updates shared, distributed, client-server databases

moet worden. Om te garanderen dat de verstuurde data aankomt, moet de onderliggende networking echter acknowledgement en error recovery schema's hebben. Dit kan vertraging opleveren en dat gaat ten koste van het realtime-gevoel van de gebruiker. Er moet afgewogen worden tussen de snelheid van het netwerk en de betrouwbaarheid ervan.

Een oplossing is het kiezen voor verschillende gradaties van betrouwbaarheid. Zo kan er een protocol gebruikt worden dat garandeert dat bepaalde (belangrijke) packetten (zoals errorberichten, het opzetten van een sessie en andere niet-frequente informatie) aankomen. Data die niet zo kritisch is en voortdurend wordt verzonden (zoals de player state) mag met minder garantie of zelfs best effort doorgestuurd worden. Dit is afhankelijk van het type NVE en er is geen kant-en-klare, algemene oplossing voor handen.

3.1.2 Datamodellen

De keuze van het datamodel heeft veel invloed: waar moet de data, die de staat van de virtuele wereld en zijn objecten bepaalt, bijgehouden worden?

Datamodellen zijn ofwel replicated (gekopieerd) ofwel shared (gedeeld).

Replicated homogeneous world database

Bij het gebruik van gekopieerde, homogene werelddatabanken, wordt aan het begin van de sessie de werelddatabank geïnitialiseerd met informatie over het terrein, de geometrie van de modellen, de textures en het gedrag van alles dat zich in de virtuele wereld bevindt. Deze databank wordt dan in zijn geheel gekopieerd naar alle gebruikers zodat de sessie kan beginnen. Tijdens de sessie werkt iedere gebruiker zijn kopie van de databank systematisch

bij met de informatie die ze van elkaar over het netwerk ontvangen.

De berichten die tijdens de sessie doorgestuurd worden zijn relatief klein, de grootste brokken data werden al gedurende de initialisatiefase naar alle gebruikers doorgestuurd. Maar dit systeem is weinig flexibel. Ten tweede geldt dat hoe groter of complexer de virtuele wereld is of hoe meer zich in die wereld bevindt, hoe groter de database van *elke* deelnemer moet zijn.

Daarnaast is het ook zo goed als onmogelijk om de consistentie van de wereld te bewaren: de updates die rondgestuurd worden kunnen verloren gaan. Dit zou kunnen opgelost worden door het gebruik van een betrouwbaar netwerkprotocol, maar dan kan het verschil in latency storen waardoor de gebruikers verschillende beelden te zien krijgen.

Bij **shared** (gedeelde) **datamodellen** wordt de toestand van de virtuele wereld bijgehouden in één of meer databanken. Daarbij is consistentie beter te behouden, maar daar staat tegenover dat er grotere berichten nodig zijn om updates door te voeren en dat er een mechanisme moet zijn om de toegang tot de gedeelde data te controleren (zie access control).

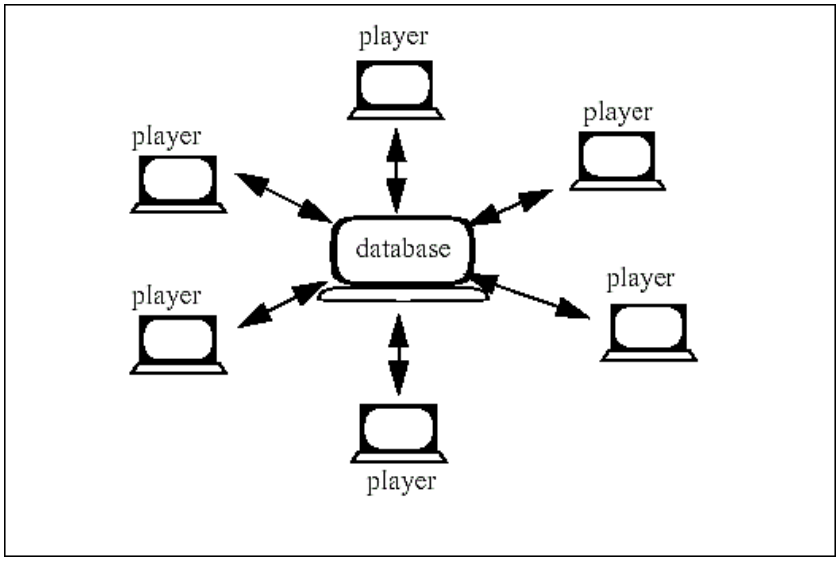
Bij shared (gedeelde) datamodellen wordt er een onderscheid gemaakt tussen het gebruik van een gecentraliseerde databank en een gedistribueerde databank.

Shared centralized databases

Het shared model met gecentraliseerde databanken (zie figuur 3.3) is analoog aan de client-server architectuur (zie het linkerdeel van figuur 3.4). De gebruikers worden clients genoemd en deze zijn allemaal verbonden met een centrale computer (de server). Via deze server gebeurt de communicatie: de clients kunnen informatie enkel naar de server sturen. Verstuurde informatie naar de server wordt een command genoemd (*cmd* in figuur 3.4).

De server zal deze commands dan behandelen om de nieuwe „toestand" in de wereld (de worldstate) te berekenen en indien nodig de veranderingen (of de volledige toestand) doorsturen naar de juiste bestemmeling(en), zodat de clients de wereld opnieuw kunnen renderen. De clients kunnen niet rechtstreeks met elkaar communiceren, maar krijgen enkel instructies of informatie van de server.

Het datamodel met centrale databank is relatief eenvoudig in ontwerp en altijd consistent. De server kan administratieve taken op zich nemen en een betere beveiliging en controle verzekeren. Dit model is echter niet geschikt voor grootschalige VE's.



Figuur 3.3: Het gedeelde model met gecentraliseerde databank. bron: [41]

De server vormt een bottleneck[3] en hierdoor is het aantal gebruikers van een systeem met dit model beperkt. Om een groot aantal clients te kunnen ondersteunen zou de server over een groot geheugen, een grote rekenkracht (voor alle operaties) en goede connecties moeten beschikken. Een deeloplossing is om verschillende servers te gebruiken, die elk een aantal clients onder zich nemen en zelf onder een *master server* staan. Servers kunnen ook andere servers onder zich hebben staan, zo kan er een hele hiërarchie van servers opgebouwd worden, met de master server als top van de piramide.

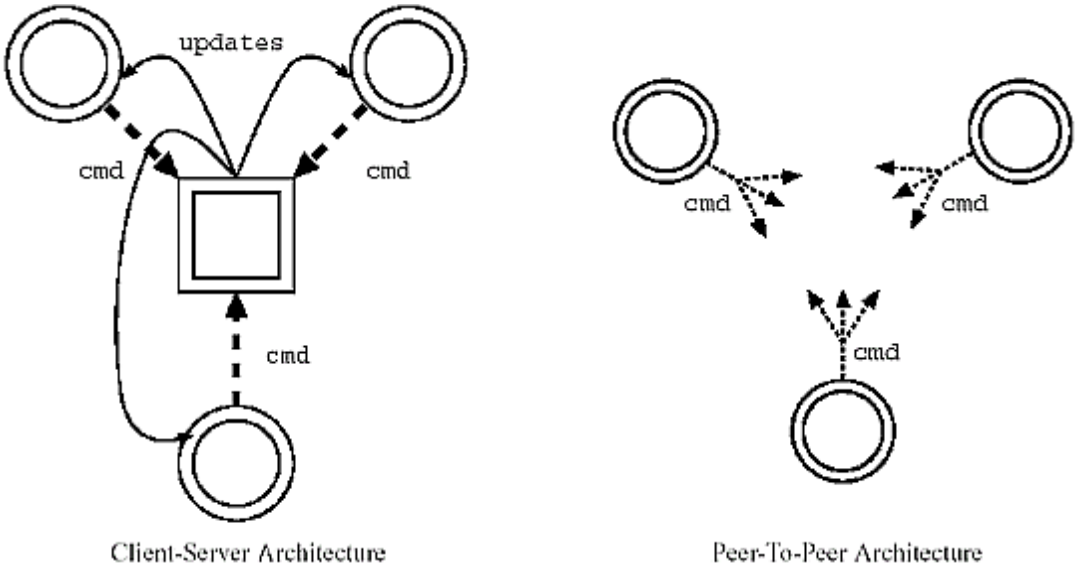
De message latency is groter dan bij het gedistribueerde model, clients moeten hun commands naar een (mogelijk ver verwijderde) server sturen en wachten tot de server de nieuwe worldstate heeft berekend en de updates verzonden heeft vooraleer de nieuwe toestand gerenderd kan worden. Worldstate consistency is echter geen probleem aangezien er slechts één kopie wordt bijgehouden (namelijk op de server) en de server heeft altijd het laatste woord.

Shared distributed databases met peer-to-peer updates

Dit systeem (zie het rechterdeel van figuur 3.4) wordt gebruikt om een soort shared memory architectuur te bekomen. Er wordt geen server gebruikt maar alle gebruikers zijn onderling met elkaar verbonden. Hierdoor valt het hiërarchische verschil tussen de computers

[3]een plaats die veel meer verkeer te verwerken krijgt dan de rest van het systeem en daardoor bij hoge pieken kan falen

(zoals bij het client-server model, waar het belangrijk is om een onderscheid te maken) weg, daarom worden de computers meestal hosts of terminals genoemd. Vaak gebruikt men echter toch de term clients, hetgeen verwarring kan scheppen.



Figuur 3.4: De gecentraliseerde en de gedistribueerde architectuur. bron: [16]

Shared distributed databases met peer-to-peer updates is dynamischer dan het replicated model. Het is ook niet noodzakelijk dat de volledige databank naar iedere gebruiker wordt gekopieerd, elke gebruiker kan zijn eigen deel van de databank beheren. Hierbij wordt het uitbreiden eenvoudiger: in plaats van eerst aan te melden bij servers, vormt deze aanpak een web van gebruikers. Hosts zenden rechtstreeks naar elkaar informatie. Doordat de tussenstap naar een server wegvalt vermindert de message latency.

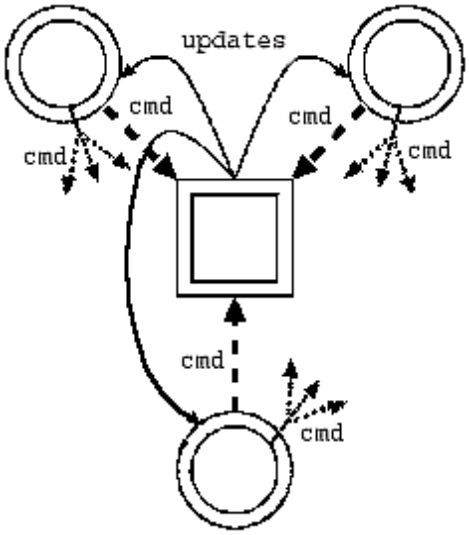
Als er echter een groot aantal gebruikers is, verhoogt de netwerkbelasting sterk. Deze is nodig voor het consistent houden van de gedistribueerde databank. Het verkrijgen van een globaal, up-to-date overzicht van de situatie wordt bemoeilijkt en er is maar een lage graad van beveiliging mogelijk.

Iedere gebruiker is verantwoordelijk voor het bijhouden van zijn eigen kopie van de world-state. Deze kopie wordt aangepast aan de hand van de messages (commands) die de gebruiker van andere gebruikers ontvangt en de consistentie van de worldstate behouden is daarom ook een groot probleem. Commands kunnen altijd vertraging oplopen, verloren gaan of in een verkeerde volgorde aankomen. Hier is extra controle voor nodig.

Shared distributed client-server databases

Het gedeelde model met gedistribueerde, client-server databanken is een variant op het client-server model waarbij de database verdeeld is over de clients en communicatie geregeld wordt door één of meerdere centrale servers. Als een gebruiker beweegt door de virtuele omgeving wordt zijn of haar databank bijgewerkt via de server omdat deze weet welke client welk deel van de wereld beheert.

Dit datamodel brengt de voordelen van de gedistribueerde databanken en de gecentraliseerde databanken samen. Wel moet er opgelet worden dat de servers geen bottleneck gaan vormen.



Figuur 3.5: Het hybride model: een combinatie van de gecentraliseerde en de peer-to-peer architectuur.

Een andere mogelijkheid is de (master)server voornamelijk te gebruiken om nieuwe clients aan te melden en voor andere administratieve taken, en via peer-to-peer connecties de clients elkaar up-to-date te laten houden (zie figuur 3.5). Hier heeft men minder kans op een strakke bottleneck bij de server. Het nadeel is echter dat het moeilijker is om de consistentie van de wereld te behouden en peer-to-peer connecties zijn gevoelig voor eavesdropping en andere inbreuken op de veiligheid.

Access control bij gedeelde datamodellen

Een extra moeilijkheid bij het implementeren van een shared datamodel in een NVE is dat er een mechanisme moet zijn om de toegang tot de gedeelde data te controleren opdat de databank consistent zou blijven. Hiervoor zijn volgende methodes mogelijk [45]:

Als gebruikers van het systeem niet proberen om tegelijkertijd te handelen met dezelfde objecten is het misschien te overwegen om helemaal **geen toegangsbeheer** toe te passen. Zo kunnen de gebruikers de VE onafhankelijk veranderen. Deze methode vereist weinig communicatie, maar kan geen consistentie garanderen.

Voor **gecentraliseerd toegangsbeheer** maakt men gebruik van een centrale server die de toegang tot de objecten controleert. Hier zijn er twee mogelijkheden: active locking met acknowledgement en active locking zonder acknowledgement.

Om toegang tot een object te krijgen, moet de host bij active locking met acknowledgement een lock aanvragen bij de server. De server stuurt daarna een bericht naar de host terug of hij het lock aanvaardt of afkeurt. De host moet dus eerst bevestiging krijgen voordat hij iets mag veranderen.

Bij active locking zonder acknowledgement worden er berichten doorgegeven om een lock op een object te plaatsen voordat dat object veranderd wordt. Naderhand worden er ook weer berichten doorgestuurd om het lock weer vrij te geven. Er is geen bevestiging door het centrale orgaan, dus locks bepalen hier geen exclusieve toegang: omdat ze niet aanvaard of afgekeurd worden, kunnen er verscheidene locks bestaan op één object. Daarom worden de „locks" in dit geval eerder gebruikt om de anderen te waarschuwen dat een bepaald object veranderd zal worden.

Gedecentraliseerd toegangsbeheer werkt zonder een centrale server. In de plaats hiervan moet de NVE gebruik maken van andere mechanismen zoals turn-taking, locking, master entities, ...

Conclusie

In „A Taxonomy for Networked Virtual Environments" door Michael R. Macedonia en Michael J. Zyda [41] wordt de studie van de datamodellen als volgt besloten:

> Replicated databanken zijn efficiënter op het gebied van communicatie dan de gecentraliseerde of de gedistribueerde gedeelde databankschema's, maar ze schieten in het algemeen te kort wat betreft het consistent houden van de wereld (een probleem met onbetrouwbare transport mechanismen zoals UDP). Ze missen ook de mogelijkheid om de VE van nieuwe objecten of werkingen te

voorzien. Grote VE's zouden een gecombineerd model kunnen gebruiken: de client initialisatie met kleine, replicated data sets en een gedistribueerd client-server model. Dit laat meer dataconsistentie en datapersistentie toe als er een mechanisme of heuristiek wordt gebruikt om de transfer latency te verminderen.

3.1.3 Updatemodellen

De gestelde netwerkingseisen worden ook beïnvloed door de hoeveelheid informatie die gekopieerd moet worden om de informatie van de verschillende gebruikers bij te werken [45]. Er zijn hier drie methodes voor, namelijk: active copying, copying on demand en migration combined with partial copying.

Active copying houdt in dat een kopie van ieder object wordt verspreid onder de deelnemers. Dit veroorzaakt veel netwerkverkeer, maar als de overdracht betrouwbaar is, verzekert dit wel dat alle kopieën consistent zijn.

Als de informatie enkel verdeeld wordt onder de gebruikers die hier interesse in hebben, spreken we van **copying on demand**. Er is minder netwerkverkeer, maar nu is de applicatie zelf verantwoordelijk voor het consistent houden van de data.

Het verschil tussen migratie en kopiëren, is dat bij migratie niet de kopieën van objecten verspreid worden over de hosts, maar de echte objecten. Bij **migration combined with partial copying** worden de objecten verdeeld (migration) over de deelnemers volgens criteria zoals hoe zwaar de host belast is, hoe snel hij gegevens kan verwerken, hoe vaak informatie over dat object wordt opgevraagd, ... Door het gedeeltelijk kopiëren (partial copying) wordt het netwerk minder belast dan bij een systeem dat enkel migratie gebruikt.

3.2 Scalability

Bij het ontwerpen van een genetwerkte virtuele omgeving voor grote aantallen gebruikers, moet er veel aandacht zijn voor scalability. Scalability of „schaleerbaarheid" is het vermogen om zich aan te passen aan de grootte van een probleem. Scalable oplossingen voor een probleem werken ook goed als de grootte van dit probleem toeneemt.

Voor een NVE gaat het hier over hoe goed deze bestand is tegen wijzigingen in de systeembronnen [60], meer bepaald hoe zij zich aanpast bij een toename van gebruikers en als de gebruikers zich verder van elkaar bevinden. Kan het systeem nog voldoende doeltreffend functioneren als de processing requirements en het aantal simultaan deelnemende gebruikers van het systeem toenemen [44]? De worldstate (of delen daarvan) moet bekend

zijn bij de nodige deelnemers en ook de berekeningen voor entiteiten die geen gebruikers zijn moeten verdeeld worden.

Een pure client-server NVE is niet goed schaleerbaar, er is een grens aan het aantal clients die een server kan ondersteunen en hoeveel verbindingen hij kan aangaan. Gedistribueerde systemen en NVE's die gebruik maken van een hybride oplossing komen wel in aanmerking.

Scalability hangt af van verschillende factoren [44]:

de capaciteit van de gebruiker: De hardware van de gebruiker (processor, rendering, netwerkconnectie, ...) moet toepasselijk zijn om de data te kunnen verwerken. Door goede code en design van de NVE moeten deze eisen beperkt kunnen blijven.

de capaciteit van de servers: Indien de NVE niet volledig gedistribueerd is, maakt deze gebruik van servers. Hun verwerkingssnelheid en de throughput moet voldoende hoog of groot zijn voor het verwachte aantal gebruikers van het systeem. Er moet ook een oplossing zijn voor als de serverload te groot wordt en deze druk herverdeeld moet worden over andere servers.

de capaciteit van het netwerk: Het netwerk moet het dataverkeer aankunnen. De bandbreedtebenodigdheden kunnen verschillen per node: een server moet bijvoorbeeld meer bandbreedte ter beschikking hebben dan een gewone deelnemer. Er zijn verschillende technieken om hiermee om te springen, zie sectie 3.3.

3.3 Compensatory Techniques

De scalability van een genetwerkte virtuele omgeving komt in het gedrang doordat er een beperking is op de systeembronnen. Resources zijn beperkt. Om deze wat te ontlasten bestaan er een aantal technieken [59, 60].

Het zijn echter geen onfeilbare oplossingen, de compensatory techniques kunnen gebruikt worden bovenop de NVE-architectuur om het netwerkverkeer tussen de verschillende nodes draaglijker te maken. De meeste van deze technieken wisselen bandbreedteverbruik in voor rekenkracht. Aangezien de ontwikkeling van processoren snel gaat, de Wet van Moore[4] is nog altijd van kracht, zijn deze technieken een zeer handig hulpmiddel.

De resource consumption van een genetwerkte applicatie is afhankelijk van de hoeveelheid data die verzonden en ontvangen moet worden door iedere deelnemende computer en hoe

[4]The observation, made in 1965 by Intel co-founder Gordon Moore while preparing a speech, each new memory integrated circuit contained roughly twice as much capacity as its predecessor, and each chip was released within 18-24 months of the previous chip. (http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?Moore's+Law)

snel het moet geleverd worden aan het netwerk. Dit is het Networked Virtual Environment Information Principle van Sandeep Singhal en Michael Zyda [59]. De sleutel tot het verbeteren van de scalability en de networking kan samengevat worden als de volgende Information Principle Equation:

$$Resources = M \times H \times B \times T \times P \tag{3.1}$$

waarbij M het aantal berichten is dat uitgewisseld wordt, H het gemiddeld aantal bestemmelingen per bericht, B de bandbreedte die nodig is voor een bericht, T de tijd die een host nodig heeft om een ontvangen bericht te verwerken en P het aantal processor cycles die er nodig zijn voor het ontvangen en verwerken van een bericht.

3.3.1 Compressie

Het doel van compressie is het terugdringen van het aantal bits die nodig zijn om bepaalde informatie voor te stellen. Bij het comprimeren van berichten ruilt men dus rekenkracht in om minder bandbreedte te verbruiken. Compressie kan zowel lossless als lossy (zoals het doorsturen van webcambeelden in JPEG-formaat bijvoorbeeld) gebeuren.

Omdat de netwerkberichten een *stroom* van gegevens zijn, kan er nog een ander onderscheid voor compressie gemaakt worden: er zijn de internal en de external technieken:

Internal wil zeggen dat de packets op zichzelf, enkel gebaseerd op hun inhoud, gecomprimeerd worden. Bij een external techniek worden de packets gemanipuleerd op basis van voorgaande berichten. Er kunnen incremental updates gebruikt worden in plaats van het doorsturen van de volledige nieuwe status, of zelfs pointers naar data die al doorgestuurd geweest is als gelijkaardige data aan bod komt.

Voor external technieken moet er evenwel een zekere reliability zijn, de voorgaande berichten moeten ook aangekomen zijn bij de andere bestemmelingen zodat dezen ook het huidige packet kunnen decomprimeren.

3.3.2 Packets samenvoegen (aggregation)

Door informatie van verscheidene berichten in één bericht, één packet te plaatsen, vermindert message aggregation het aantal berichten dat over het netwerk verstuurd wordt en de overhead die daarmee gepaard gaat [59]. Ieder netwerkbericht dat verstuurd moet worden draagt namelijk niet alleen informatie voor de NVE, maar ook een packetheader: voor een UDP-packet bedraagt de header 28 bytes[5]. Bij het samenvoegen van verschillende

[5] 8 bytes voor UDP en 20 bytes voor het onderliggende IP, zie http://www.networksorcery.com/enp/protocol/udp.htm

packets wordt er dus minder bandbreedte verbruikt doordat deze vaste overhead bij de extra berichten wegvalt.

Het nadeel aan aggregatie is dat het een vertraging veroorzaakt (al dan niet zeer klein) voor het verzenden van nieuwe packets door het wachten tot er „genoeg" berichten zijn om samen te voegen. Hoe langer men echter wacht op het verzenden van het nieuwe netwerkbericht, hoe meer afzonderlijke berichten er samengevoegd worden, hoe groter de winst aan bandbreedte. Wat is dan „genoeg"? Om dit te bepalen, zijn er drie mogelijkheden: met time-outs, met een quorum of een hybridisch algoritme.

Voor de eerste mogelijkheid verzamelt de aggregator individuele packets tot er een timer afgaat, hierna wordt het resulterende bericht verzonden. Een quorum-based beleid houdt in dat er met verzenden gewacht wordt tot er een vast aantal individuele berichten verzameld is.

De time-out methode heeft als voordeel dat er een bovengrens is aan de vertraging die veroorzaakt wordt. Het is echter ook mogelijk dat tijdens die wachttijd er zich slechts één bericht aandoet, dan is die wachttijd zinloos geweest en betekent dan enkel verlies.

Bij het gebruik van de quorum methode wordt er door het vaste aantal samengevoegde berichten een bepaalde vermindering van het bandbreedteverbruik gegarandeerd. Wat niet bekend is, is de tijd die er zal verstrijken tot dat aantal is bereikt. Als er niet voldoende berichten voorhanden zijn, is het dus mogelijk dat de aggregator oneindig lang blijft wachten.

Een betere manier is de beide methodes te combineren om zo de nadelen te minimaliseren. Er worden zowel een quorum als een time-out vastgelegd en als één van de twee voldaan is, worden ze gereset en het bekomen packet verstuurd. De time-out zal nuttig blijken bij een trage aanvoer van berichten en voorkomt het lange wachten op een voldoende aantal. Het quorum heeft de bovenhand bij een snelle aanvoer en voorkomt zo onnodige (extra) vertragingen.

3.3.3 Interest Management

Het gebruik van interest management [28, 30, 48, 61] houdt in dat de nodes duidelijk maken in welk soort data ze geïnteresseerd zijn. Deze interesse wordt bepaald door verschillende attributen en eigenschappen, zoals de locatie van de gebruiker, en wordt uitgedrukt in *interest expressions*.

Agents in de NVE (interest managers) ontvangen de interest expressions van de gebruikers en gebruiken deze om de netwerkberichten te filteren en op te delen in verzamelingen die voldoen aan de eisen van de gebruiker. Zo krijgen de nodes enkel data toegestuurd die relevant voor hen is. Dit reduceert de H, het gemiddeld aantal bestemmelingen per bericht in formule 3.1. Een goede tactiek is om per verzameling van data een bepaald multicastadres te gebruiken en de gebruikers die deze data relevant vinden zich laten inschrijven op dit adres.

Een interest expression noemt men een *aura* of een *area (domain) of interest*. Het is een subruimte waar interactie gebeurt. Als twee gebruikers hun aura's elkaar snijden (zie bijvoorbeeld figuur 2.2) zijn ze zich van elkaar en elkaars acties bewust. Interest management met areas of interest is symmetrisch: als aura's elkaar snijden, ontvangen beide partijen berichten van elkaar.

Men kan nog een stap verder zetten en aura opdelen in een focus en een nimbus. De focus is het deel van het aura dat de gebruiker op dat moment waarneemt en de nimbus is het gebied van waar een object waargenomen kan worden. Een gebruiker zijn focus moet de nimbus van een object of een andere gebruiker snijden vooraleer de eerste zich van de andere bewust kan zijn. Focussen en nimbussen bezorgen een meer verfijnde message filtering, het bewustzijn is niet meer noodzakelijk symmetrisch.

Aura's (of de focussen en de nimbussen) kunnen aangepast worden door *adapters* (zoals bij MASSIVE-2). In de virtuele ruimte kunnen objecten zoals bijvoorbeeld verrekijkers of camouflagenetten gebruikt worden.

3.3.4 Dead Reckoning

Dead reckoning is een techniek waarbij de hosts de toestand van objecten binnen de virtuele wereld afleiden aan de hand van vorige status updates. Nieuwe status updates worden pas doorgestuurd als er een bepaalde drempelwaarde overschreden wordt waaruit blijkt dat de hosts de toestand van dat bepaald object niet meer kunnen voorspellen. De positie van een object wordt bijvoorbeeld berekend op basis van het startpunt van de beweging, zijn oriëntatie en de snelheid ervan. Er worden zo enkel packets in verband met de object state op het netwerk geplaatst als de home node (de node die het object beheert) dat nodig vindt.

Aangezien status en positie updates een groot deel van de netwerkberichten behelst, is het gebruik van dead reckoning een grote verlichting van het bandbreedteverbruik. Nadelig is het feit dat dit inconsistentie met zich meedraagt: niet alle hosts delen dezelfde toestand in verband met het object, daardoor is het belangrijk om een goede drempelwaarde te kiezen.

Als er een nieuwe status update aankomt bij een node, kan blijken dat de voorspelde positie niet correct is. Er zijn technieken ontwikkeld om te voorkomen dat bewegende objecten op een schokkerige manier voorgesteld gaan worden zodat de gebruiker hier niets storend van opmerkt. De voorspellingsalgoritmes moeten gecombineerd worden met convergentiealgoritmes.

Voor meer informatie wordt er doorverwezen naar de werken [59] , [42], [61] en [55].

3.3.5 Level of Details

Objecten op een grote afstand zijn in de echte, fysieke wereld amper waarneembaar. Hun details ontgaan ons. Een bos aan de horizon ziet er uit als een groene vlek. In een NVE is het ook niet nodig om gedetailleerde informatie te ontvangen van objecten die te ver afgelegen zijn om duidelijk te kunnen waarnemen.

Deze eigenschap kan benut worden door middel van level of details (LOD). Verschillende levels of details kunnen gedefinieerd worden voor eenzelfde object. Afhankelijk van de afstand wordt dan een meer (voor dichtbij) of minder gedetailleerd model gebruikt. Dit kan tijdens het renderen gebeuren door middel van mesh simplification of door op voorhand een aantal modellen te definiëren (zoals bij Urbi et Orbi, zie sectie 2.3.1 - High Level Scene Description). De mesh simplification methode heeft een sterkere rekenkracht nodig, terwijl de oplossing met de verschillende modellen meer netwerkberichten zal gebruiken om de gebruikte modellen te kunnen initialiseren.

Voor NVE's kan dit begrip nog uitgebreid worden tot de frequentie van het sturen van status-update packets (vooral zeer belangrijk bij het gebruik van dead reckoning). Ver verwijderde objecten hun snelheid en oriëntatie zijn moeilijk te bepalen. Daardoor mogen synchronisatieberichten voor deze objecten een lagere prioriteit hebben en minder frequent doorgestuurd worden naar de gebruiker. Bij het gebruik van dead reckoning kan men ook gebruik maken van minder precieze algoritmes, dit is ook ten voordele van het processorverbruik.

3.3.6 Samenvatting

De hierboven beschreven technieken hebben allen een invloed op de formule 3.1 en beïnvloeden zo de nood aan resources en de relatie tussen de networking en het verwerken van informatie binnen de genetwerkte virtuele omgeving. Tabel 3.2 geeft een overzicht.

Tabel 3.2: De compensatory techniques en hun invloed. bron: [62]

Techniek	Invloed op formule 3.1
Message Compression	minder B, meer P
Message Aggregation	minder M en B, meer P en T
Interest Management	minder H, meer M, P en T
Dead Reckoning	minder M, meer P
Level of Details	minder H en B, meer M en P

Legende:

M	aantal berichten dat wordt uitgewisseld
H	gemiddeld aantal bestemmelingen per bericht
B	nodige bandbreedte per bericht
T	tijd voor het verwerken van een ontvangen bericht
P	aantal processor cycles voor het ontvangen en verwerken van een bericht

3.4 Conclusie

In dit derde hoofdstuk werd de taxonomie voor genetwerkte virtuele omgevingen (opgesteld door Michael R. Macedonia en Michael J. Zyda) omschreven. Deze is al gebruikt geweest in het vorige hoofdstuk om een idee te geven van de noden en de scalability van de besproken voorbeelden.

Scalability kan samengevat worden als de vraag: „Kan de NVE nog goed functioneren als de processing requirements en het aantal gebruikers van het systeem toenemen?". Dit is een zeer belangrijk begrip waar nog geen algemene oplossing voor bestaat. Indien men een goed schaleerbaar systeem wilt ontwerpen, moet men rekening houden dat het op het huidige Internet nog niet mogelijk is om daarnaast ook een real-time data-levering met hoge betrouwbaarheid te garanderen (zie figuur 3.1).

De Information Principle Equation (zie formule 3.1) laat ons zien dat het verbruik van resources afhangt van vijf variabelen: het aantal berichten dat over het netwerk wordt gestuurd (M), het gemiddeld aantal bestemmelingen per bericht (H), de nodige bandbreedte per bericht (B), de tijd nodig voor het verwerken van een ontvangen bericht (T) en het aantal processor cycles nodig voor het ontvangen en het verwerken van een bericht (P).

De formule geeft aan dat bij het verlagen van één of meer van de vijf variabelen ook de systeemvereisten verlagen. Deze verlaging moet evenwel gecompenseerd worden door ofwel een andere variabele te laten toenemen ofwel toe te staan dat de kwaliteit van de virtuele ervaring afneemt.

Via de besproken compensatory techniques kan men het verbruik van een bepaalde resource inruilen tegen een hogere eis van een andere. Meestal ruilt men bandbreedteverbruik in voor meer rekenkracht. De keuze van welke variabelen verlaagd worden en welke variabelen als compensatie verhoogd worden, hangt af van de eisen van de applicatie en de bottlenecks. Zo kan men de scalability van het systeem beïnvloeden.

Voor meer gedetailleerde informatie wordt er doorverwezen naar de werken: [41], [59], [61], [60], [44], [42] en [35].

Vanaf dit punt zal het verdere verloop van deze thesis zich spitsen op mogelijke distributiemethodes voor genetwerkte virtuele omgevingen, meer bepaald op multicasting en reliable multicasting. In het volgende hoofdstuk wordt het inleidende werk gedaan voor de implementatie van een framework waarmee vergelijkende tests uitgevoerd worden.

Hoofdstuk 4

Multicasting

Multicasting laat groepen van willekeurige grootte toe te communiceren over een netwerk met maar één transmissie door de bron, men kan gecontroleerd data versturen naar een aantal (maar niet noodzakelijk alle) gebruikers.

Gebruikers moeten zich inschrijven op een multicastgroep om de packets die hiernaar verzonden worden te kunnen ontvangen, om data te verzenden naar een multicastgroep is inschrijving echter niet vereist. Als men niet meer wenst gebruik te maken van een bepaalde multicastgroep, kan men zich hiervoor uitschrijven. Gebruikers kunnen zich tegelijkertijd voor verscheidene multicastgroepen inschrijven.

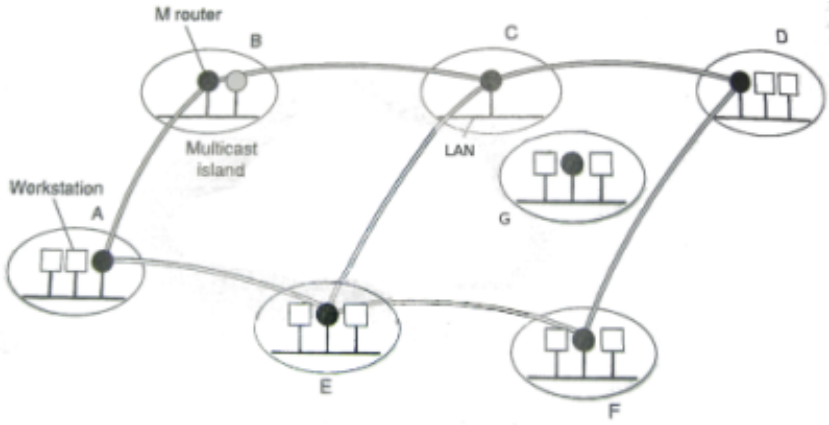
Zoals al gezegd is multicasting een efficiënte distributiemethode voor genetwerkte virtuele omgevingen vergeleken met de andere mogelijkheden. Er moeten geen afzonderlijke verbindingen opgesteld worden van elke node naar de rest (in tegenstelling tot unicast) en aangezien packets enkel gestuurd moeten worden naar de leden van een groep, moeten de berichten niet het hele netwerk doorkruisen (in tegenstelling tot broadcast).

Een worst-casescenario is dat iedere gebruiker slechts geïnteresseerd is in een klein deel van de informatie op elke multicastgroep en zich alzo moet inschrijven voor iedere groep om de nodige data te ontvangen. In dit geval ontaardt de NVE in een broadcasting systeem. Maar als de opdeling in multicastgroepen op een zinvolle, verantwoorde manier gebeurt zou dit scenario niet mogen voorkomen.

4.1 Multicast Backbone

Het nadeel van multicasting is dat (momenteel) slechts weinig Internet Service Providers IP multicast ondersteunen. Hiervoor werd MBone[1] ontwikkeld. Het maakt multicasting mogelijk over het datagramgeoriënteerde Internet en is operationeel sinds 1992.

MBone is een laag bovenop het Internet. Het bestaat uit eilandjes van netwerken die wel multicast-enabled zijn en die verbonden worden door „tunnels" (zie figuur 4.1). Ieder eiland bestaat uit een LAN of een aantal intergeconnecteerde LANs met multicast routers. Deze multicast routers zijn via de tunnels over het internet met elkaar verbonden, zodat MBone packets tussen de eilanden gestuurd kunnen worden. Als alle routers in de toekomst zelf multicastverkeer kunnen behandelen, zal MBone niet meer nodig zijn.



Figuur 4.1: MBone bestaat uit multicast eilanden verbonden door tunnels. bron: [66]

4.2 Reliable Multicast

IP Multicast vertrouwd op best-effort technieken. Daarom zijn er door de jaren heen reliable multicast protocollen ontwikkeld. Om deze protocollen met elkaar te vergelijken, is er een taxonomie nodig.

Er zijn een aantal papers die taxonomieën voorstellen: Brian Neil Levine en Jose Joaquin Garcia-Luna Aceves stellen voor [36, 37] om de protocollen op te splitsen in vier grote klassen: de sender-initiated protocollen, de receiver-initiated protocollen en de Tree- en Ring-based protocollen.

[1] http://www.lbl.gov/web/MBONE.html; http://graphics.stanford.edu/papers/mbone/

In het werk "Multicast Transport Protocols: A Survey and Taxonomy" [53] van Katie Obraczka wordt de opdeling gebaseerd op tien kenmerken:

Data propagation: Hoe wordt de data (niet de controleinformatie) gedistribueerd onder de leden?

Reliability mechanism: Op welke manier herstelt het systeem zich bij het verliezen van data (data loss)? Is het protocol sender-initiated of receiver-initiated?

Repair request: Hoe vraagt men het herversturen van verloren data aan?

Retransmission: Hoe verloopt dat herversturen?

Feedback control: Hoe wordt de hoeveelheid controleinformatie (die door de ontvangers wordt gegenereerd) beperkt?

Flow and congestion control: Gebruikt het protocol mechanismen om een netwerkopstopping tegen te gaan?

Locus of control: Is het een gecentraliseerd of een gedistribueerd protocol?

Ordering: Wordt er gegarandeerd dat de leden de berichten in de juiste volgorde ontvangen of de correcte volgorde kunnen reconstrueren?

Group management: Beheert men het lidmaatschap tot de multicastgroep?

Target application: Is het protocol ontwikkeld voor een specifieke applicatie of is het een algemene oplossing?

De Multicast Implementation STudy (MIST) van TASC [68] en Ricardo Galli [26] hanteren beiden dezelfde taxonomie: hoe reliable, heterogeen en scalable is het protocol; welke technieken worden er gebruikt voor flow control, ordening en fragmentatie/herstellen van de berichten; en hoe wordt er met late-joins en early-departures omgegaan?

Bovenstaande voorstellen worden in het volgende deel gebruikt om een taxonomie op te stellen. Deze zal gebruikt worden om een aantal bekende reliable multicast transportprotocollen met elkaar te vergelijken.

4.3 Taxonomie voor reliable multicast transportprotocollen

De theoretische veronderstelling uit de werken van Brian Neil Levine en Jose Joaquin Garcia-Luna Aceves [36, 37] dat elk protocol twee windows gebruikt: een congestion window en een memory allocation window wordt hier overgenomen. Zo kan men de mechanismen die nodig zijn voor het aanpassen van de transfersnelheid en die voor de geheugenallocatie scheiden. De grootte van het congestion window wordt bepaald door feedback

over de transmissiesnelheid en de errordetectie, gekregen van ontvangers. De grootte van het memory allocation window wordt bepaald door feedback van de ontvangers waarmee bepaald wordt of een verstuurd bericht uit het geheugen van de zender (en eventueel ook andere nodes, zie local recovery in sectie 4.3.2 en locus of control in sectie 4.3.3) verwijderd kan worden.

In de praktijk kan het voorkomen dat er slechts één window gebruikt wordt voor zowel de congestiecontrole als de geheugenallocatie maar dit brengt de gebruikte taxonomie niet in gevaar.

4.3.1 Het aantal zenders binnen de multicastgroep

De soorten reliable multicast transportprotocollen kunnen in twee grote groepen gesplitst worden. Dit zijn degenen met *one-to-many* en die met *many-to-many* levering van de packets.

Bij one-to-many is er slechts één zender in de multicastgroep en bij many-to-many kunnen alle ontvangers ook als zenders fungeren.

4.3.2 Reliability

Het ontdekken van data loss

Hoe betrouwbaar is het systeem? Het protocol moet merken wanneer er een foute transmissie gebeurd is. Een packet kan verloren gaan of aankomen maar beschadigd of niet het verwachte packet blijken te zijn. Bij wie ligt de verantwoordelijkheid om dit te ontdekken?

Sender-initiated protocollen Traditionele reliable unicast transportprotocollen (zoals TCP) laten de ontvanger een Acknowledgebericht (een ACK) sturen als de data juist ontvangen is geweest. Hierdoor is het de zender zijn verantwoordelijkheid om het verlies van data op te sporen en zo een transportprotocol heet sender-initiated.

Het belangrijke kenmerk van sender-initiated protocollen is dat de zender ACKs moet ontvangen hebben van alle nodes die de data horen te ontvangen en dan pas mag hij het geheugen, dat voor die data was gealloceerd, terug vrijgeven [36].

Er kan echter een ACK-implosie veroorzaakt worden telkens een zender een bericht naar de groep stuurt en de ontvangers de aankomst van het bericht willen bevestigen. Hierna zou de zender dan ook nog moeten controleren of wel iedere bestemmeling een ACK gestuurd heeft, dus de zender moet perfect op de hoogte zijn van wie er

zich allemaal in de multicastgroep bevindt.

Receiver-initiated protocollen Voor genetwerkte virtuele omgevingen zijn sender-initiated protocollen geen goede oplossing. De zender moet weten wie er allemaal in de multicastgroep zit en moet al die hun ACKs behandelen. Het gevaar voor een ACK-implosie is trouwens groot.

Beter is het om de verantwoordelijkheid aan de ontvangstzijde te leggen. Onderzoek [56] toont aan dat voor reliable multicasting de receiver-initiated aanpak beter presteert dan de sender-initiated methode.

Hier vragen de ontvangers een retransmissie aan door een Negative Acknowledgement (een NACK of NAK) te sturen. Dit gebeurt als er bijvoorbeeld een fout is opgetreden, de serienummers (zie sectie 4.3.6) blijk geven dat er een packet gemist werd, er een time-out is verstreken of er een packet met corrupte data is aangekomen.

Het versturen van NACKs maakt een protocol echter niet receiver-initiated. Sender-initiated protocollen kunnen gebruik maken van NACKs om extra controleinformatie door te voeren maar deze berichten hebben geen invloed op het memory allocation window.

De zender bij receiver-initiated protocollen is niet zeker wanneer hij geheugen mag vrijgeven. Om correct te moeten werken zouden de zenders van een receiver-initiated protocol een oneindig[2] geheugen moeten hebben, zie Theorem 2: „A receiver-initiated reliable protocol is not live" in [37].

Tree-based protocollen De ontvangers worden in een acknowledgement tree (zie figuur 4.2 (a)) onderverdeeld. De boom bestaat uit drie soorten knopen: een source node, leaf nodes en hop nodes. De source node is de zender van een nieuw packet, de leaf nodes zijn de uiteinden van de boom en deze hebben dus geen kinderen waarvoor ze verantwoordelijk zijn. De hop nodes zijn alle knopen die daar tussenin zitten en zijn telkens de ouders van andere nodes. Een ouder heet hier de "groepsleider" van zijn subtree.

Als een node in een groep een correct packet ontvangt, stuurt deze meteen een acknowledgement naar zijn groepsleider. Deze acknowledgements noemt men Hierarchical Acknowledgements of HACKs. Leaf nodes kunnen gelijkgesteld worden aan de ontvangers in een sender-initiated protocol. Het enige verschil is dat ze een HACK sturen naar hun groepsleider in plaats van rechtstreeks naar de zender.

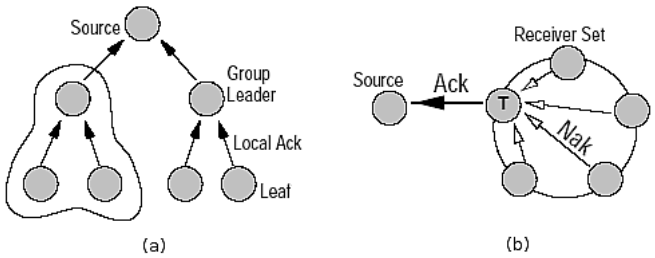
[2] genoeg geheugen om ieder packet ooit verzonden door de zender bij te houden

Indien de node geen leaf node is, dan stelt het ook een timer in. Als de node geen HACK toegestuurd krijgt van een kind voor de timer afloopt, dan is er een fout opgetreden en wordt het packet opnieuw doorgestuurd door de node. De source doet hetzelfde als hij een bericht naar zijn kinderen stuurt en niet op tijd alle HACKs krijgt. Groepsleiders sturen HACKs naar hun eigen ouder, enzovoort.

De groepsleiders beslissen wanneer het geheugen in het message allocation window vrijgegeven wordt, namelijk wanneer al de kinderen van de groepsleider een HACK gestuurd hebben.

Door deze structurele opbouw is er minder kans op een ACK-implosie en de zender moet niet meer alle ontvangers kennen. In [36] wordt aangetoond dat de knopen niet meer belast worden als het aantal leden van de multicastgroep stijgt.

Een acknowledgement tree met alleen maar bladeren en een source node stemt overeen met een sender-initiated protocol.



Figuur 4.2: Diagrammen van een tree-based (a) en een ring-based (b) protocol. bron: [37]

Ring-based protocollen Er is slechts één multicastlid (de token site, T in figuur 4.2 (b)) verantwoordelijk voor het zenden van een ACK naar de zender. Als de zender niet op tijd een ACK toegestuurd krijgt door dit lid, dan verstrijkt de time-out en stuurt de zender het bericht opnieuw. De ontvangers sturen een NACK naar T als er zich een fout voordoet.

Het token wordt pas doorgegeven aan de volgende node in de ring als deze alle packets correct heeft ontvangen. De zender wacht met zijn geheugen vrij te maken totdat het token doorgegeven is.

Forward Error Correction (FEC) Een andere techniek is Forward Error Correction (FEC) [51, 38] waarbij men uitgaat van het feit dat data verloren zal gaan en daarom stuurt men extra data mee. Dankzij deze extra data kunnen ontvangers alsnog de verloren data reconstrueren als er niet te veel verloren is gegaan.

Hier gaat een overhead mee gepaard. De zender moet de redundante data genereren en verzenden en de ontvanger moet bij data loss deze redundante data ontcijferen. De ontwikkelaars moeten beslissen in welke mate deze overhead een goede ruil is voor het voorkomen van het sturen van aanvragen en de retransmissies zelf.

Verzoek tot retransmissie

Op welke manier vraagt men het herversturen van data aan? Als er zich een probleem voordoet, dan moet dit duidelijk gemaakt worden aan een node (meestal, maar niet noodzakelijk, de originele zender) die de data kan herversturen. Deze aanvraag kan point-to-point verlopen of naar de hele groep gestuurd worden.

Als bij receiver-initiated protocollen een NACK naar de hele groep gestuurd wordt in plaats van enkel naar de verzender van het packet, dan kan men technieken gebruiken om te voorkomen dat andere nodes ook NACKS gaan zenden (zie *feedback control* in sectie 4.3.3). Dit heeft enkel zin als de retransmissies ook via multicasting gebeuren.

Sommige transportprotocollen gaan nog een stap verder en proberen via *local recovery* de verloren data te herstellen. Hierbij is het niet noodzakelijk dat men de verloren data aanvraagt bij de zender zelf, maar kunnen hosts die dichtbij[3] de aanvrager gelegen zijn de informatie doorspelen (zie ook de tree-based protocollen).

Sender-initiated en tree-based protocollen werken met timers. Als voor de time-out niet alle ACKs of HACKs ontvangen zijn geweest, dan moet de zender of de groepsleider het bericht opnieuw sturen. Hier gebeurd dus geen letterlijke aanvraag of request, maar baseert men zich op het niet (of te laat) ontvangen van een bevestiging. Communicatie tussen de zender en de token site bij ring-based protocollen, verloopt analoog.

De retransmissie

Dezelfde vraag kan gesteld worden voor de retransmissies zelf. Gebeurt dit via unicasting of wordt het meteen naar de hele groep gestuurd? Sommige protocollen gebruiken de local recovery methode waarvan hierboven sprake was.

Het multicasten van de retransmissie heeft als voordeel dat er verscheidene hosts tegelijkertijd beholpen kunnen worden. De data kan dan ook hosts bereiken wiens verzoek tot retransmissie zelf verloren was gegaan. Het nadeel ervan is dat diegenen die de data wel

[3] "dichtbij" of "ver weg" is hier geen ruimtelijk begrip, maar is gebaseerd op de round-trip tijd (RTT) tot de andere groepsleden of de verzender van de data.

correct hebben ontvangen deze packets eruit moeten kunnen filteren.

Bij tree-based protocollen is selective repeat de beste manier. Zo moet het packet niet opnieuw gebroadcast worden, maar kan het via de structuur worden doorgegeven van de groepsleider naar het kind.

Als de token site bij ring-based protocollen een NACK ontvangt voor een packet dat hij wel correct heeft ontvangen, dan wordt er selective repeat gebruikt om het packet door te sturen naar de andere node.

4.3.3 Scalability

Voor genetwerkte virtuele omgevingen die doelen op een groot aantal gebruikers, is het belangrijk dat het transportprotocol dat ze gebruiken scalable is.

Zoals al gezegd zijn tree-based protocollen interessant gezien het feit dat de inspanning die de hop nodes moeten leveren onafhankelijk is van het aantal ontvangers in de multicastgroep.

Er zijn een aantal punten waarop gelet moet worden:

Feedback control

Bij bijvoorbeeld receiver-initiated reliable multicasting die NACKs terugsturen als er zich een fout voordoet (zie hierboven) dreigt het gevaar op een feedbackimplosie. Nodes die in een networkopstopping zitten kunnen daarboven deze opstopping nog erger maken door steeds NACKs om de ontbrekende data aan te vragen te blijven multicasten[4].

Daarom moet er een vorm van feedback controle zijn, een mechanisme dat voorkomt dat er te veel controleinformatie gestuurd wordt door de multicastgroep.

Feedback controle bestaat in twee vormen, structure-based en timer-based:

Bij de structure-based techniek[5] stelt het protocol een site (een lid van de multicastgroep of een speciale server, zie sectie *locus of control* hieronder) aan voor het verwerken en filteren van feedback informatie. Zie bijvoorbeeld de ring-based transportprotocollen.

[4] ook bekend als het *crying baby*-probleem

[5] Document [36] geeft een andere definitie voor de structure-based techniek en baseert zich daarvoor op protocollen waar de leden van de multicastgroep in een structuur worden geplaatst om zo de feedback te kunnen filteren, zoals de tree-based protocollen. In dit werk wordt echter de bovenstaande definitie gehanteerd.

Timer-based oplossingen voor dit probleem onderdrukken de kans dat alle receivers tegelijkertijd NACKs gaan sturen.

Zo is er de mogelijkheid om de ontvanger die problemen heeft een willekeurige tijd (van 0 ms tot de tijd die een packet nodig heeft om de zender te bereiken) te laten wachten vooraleer een verzoek tot retransmissie te versturen. De bedoeling is om nodes die dichtbij de zender gelegen zijn hun feedback eerder te laten sturen en de NACKs van verder gelegen groepsleden te onderdrukken zodat er slechts één NACK verstuurd wordt (en dan nog degene waar het snelst op gereageerd kan worden). Zie bijvoorbeeld het NACK-Oriented Reliable Multicast Protocol of NORM in sectie 5.3 en [23].

Locus of control

Verscheidene multicast transportprotocollen gebruiken een centrale site om functies zoals message ordering, group membership, feedback control en retransmissies te controleren. Dit centraal punt kan een lid van de multicastgroep of een speciale server zijn. Het gebruik van zo een site garandeert een gecontroleerde uitvoer van dit soort functies, maar kan de schaleerbaarheid in de weg staan.

Deze aanpak noemt men *gecentraliseerd*. Als er geen centraal punt bestaat om controles uit te voeren, dan heet dat *gedistribueerd*.

Flow and congestion control

Mechanismen voor congestion control worden gebruikt om te voorkomen dat de bron de ontvangers overspoelt met data en zo het netwerk opstopt. Het bepalen van de snelheid waartegen packets over het netwerk verstuurd moeten worden is niet zo evident, er moet een methode gevonden worden die voor alle ontvangers acceptabel is. Dit heeft een grote impact op de algemene performantie van de NVE en dus op de ervaring van de gebruikers.

Vele multicast transportprotocollen implementeren geen expliciete vorm van controle. Ze vertrouwen erop dat de nood aan retransmissie bij opstopping de datastroom zal vertragen.

De klassieke manieren voor flow en congestion control zijn rate, window[6] en token based mechanismen. Hier mag men respectievelijk geen packets sturen als dat sneller blijkt te zijn dan afgesproken, als het maximum aantal (onbevestigde) packets is bereikt of als er geen token toegekend wordt.

[6] zoals bij TCP

Layered multicast De "gelaagde multicast"-techniek wordt gebruikt voor toepassingen waarbij de data kan opgesplitst worden in verschillende lagen. De opsplitsing moet zo gebeuren dat de laagste laag het minimum aan data bevat en iedere laag daar boven een soort increment is. Ze bevatten telkens een beetje extra data dat, als het bij de onderliggende laag wordt samengevoegd, resulteert in data van betere kwaliteit.

Bij bijvoorbeeld videostreaming kan het encoden zo gebeuren dat het een lossy compressie is waarvan de onderste laag een zeer laag kwaliteitsbeeld (dat weinig bandbreedte verbruikt) oplevert en hoe meer bovenliggende lagen men kan ontvangen, hoe minder lossy de encoding is, hoe hoger de kwaliteit van het ontvangen beeld zal zijn.

Een laag komt overeen met een multicastgroep. Hoe meer bandbreedte de ontvanger ter beschikking heeft, hoe meer multicastgroepen waar hij zich voor kan inschrijven. Zolang er geen problemen voorkomen, kan de ontvanger zich ook voor nog een "hogere" multicastgroep inschrijven. Hierdoor ontvangt hij meer lagen en des te beter de kwaliteit van de data. Als er congestie optreedt, dit wordt opgemerkt door de ontvanger als er packets verloren blijken te gaan, kan de ontvanger zich uitschrijven voor de hoogste laag (of lagen) waarvoor hij is ingeschreven, tot het probleem voorbij is.

Dit is echter geen algemene oplossing, maar is enkel toepasselijk voor data zoals videostreaming, maar kan wel voor een deel van de NVE (waar dat soort data zich voordoet) gebruikt worden. Een idee is om dit misschien te gebruiken voor data dat opgesplitst kan worden in level of details (zie sectie 3.3).

Vertegenwoordigers van een subtree Een andere methode gebruikt vertegenwoordigers: het selecteert dynamisch een klein aantal van de leden van de multicastgroep. Deze worden de "vertegenwoordigers" genoemd en moeten tijdig informatie geven over opgestopte subtrees. Deze vertegenwoordigers geven onmiddellijk feedback, hetgeen NACKs van andere ontvangers onderdrukt (en zo neemt ook de kans op een feedbackimplosie af, zie hierboven).

Dankzij deze feedback hoeft de zender zich enkel te concentreren op die bepaalde subtree(s) om de congestie op te lossen, hij kan zijn transmissiesnelheid op de verkregen informatie afstellen. Als er zich nieuwe opstoppingen voordoen binnen de boom, dan worden er nieuwe vertegenwoordigers geselecteerd.

Voor meer informatie over flow en congestion controle voor multicast transportprotocollen wordt er doorverwezen naar de papers op de research-site van Junsoo Lee: http://www-scf.usc.edu/~junsoole/research/multicast_congestion/.

4.3.4 Group management

Hoe beheert het protocol het lidmaatschap van de multicastgroep? Sommige protocollen beheren deze kennis helemaal niet, zenders versturen gewoon de data naar de multicastgroep en iedereen kan toetreden tot een multicastgroep. De verantwoordelijkheid wordt volledig bij de applicatie gelegd. Dit is de best schaleerbare manier en heet *implicit group management*.

Explicit group management vereist wel controle over het lidmaatschap van een multicastgroep. In dit geval is het lidmaatschap ofwel statisch (niet interessant voor een NVE die gebruik maakt van bijvoorbeeld Interest Management) gedurende de sessie ofwel moet het toetreden tot en het verlaten van een multicastgroep duidelijk gemaakt worden aan de groep (of een centraal punt, zie sectie 4.3.3).

Een minder strenge variant is dat enkel het toetreden tot de multicastgroep aangevraagd moet worden.

Protocollen die ontworpen worden voor zeer grote multicastgroepen hebben vaak geen uitgebouwd group management systeem. Dit brengt namelijk veel overhead met zich mee. Voor receiver-initiated multicasting is dit in principe ook niet nodig. Tree-based en ring-based protocollen eisen echter dat er een structuur wordt opgebouwd van de ontvangers en bij sender-initiated protocollen moet de zender de ontvangers in de multicastgroep kennen.

Late-joins

Bij niet-statisch group management zal de grote van de groep voortdurend aan verandering onderhevig zijn. Gebruikers zullen niet volledige sessies uitzitten. Als een gebruiker zich inschrijft voor een multicastgroep als de sessie al bezig is, dan noemt men dit een *late-join*, hetzelfde bestaat voor het uitschrijven alvorens de sessie afgesloten is: een *early-departure*. De mogelijkheid tot late-joins en early-departure hebben hun invloed op de flow en congestion control en de reliability mechanismen.

4.3.5 Fragmenteren en terug samenvoegen van packets

Een aantal multicast transportprotocollen (zoals het Reliable Adaptive Multicast Protocol of RAMP, zie sectie 5.2) staan toe dat berichten nog eens gefragmenteerd worden in verschillende kleinere packets. Bij andere protocollen ligt deze deze verantwoordelijkheid bij de applicatie.

4.3.6 Ordenen

Garandeert het protocol een ordening van de netwerkberichten zodat de ontvanger de volgorde kan reconstrueren en welke technieken gebruikt het hiervoor?

De volgende mogelijkheden zijn gehaald uit [26]:

Eén bron: als berichten m_1 en m_2 afkomstig zijn van dezelfde zender en verstuurd zijn naar dezelfde multicastgroep, dan krijgen de ontvangers ze in dezelfde volgorde.

Meerdere bronnen: als berichten m_1 en m_2 verstuurd zijn naar dezelfde multicastgroep, dan krijgen de ontvangers ze in dezelfde volgorde ook al zijn ze verstuurd door verschillende bronnen.
Hier wordt *causal ordering* toegepast: als de zender van bericht m_2 het bericht m_1 had ontvangen voor hij zijn bericht stuurde, dan moeten de ontvangers ook eerst m_1 ontvangen en dan pas m_2.

Meerdere groepen: als berichten m_1 en m_2 geleverd worden aan twee processen, dan zijn ze geleverd in de juiste volgorde, ook al zijn ze verstuurd door verschillende bronnen naar twee verschillende (maar overlappende) multicastgroepen.

Er kan een centraal punt (zie locus of control in sectie 4.3.3) gebruikt worden dat globale serienummers of timestamps aan de berichten geeft. Deze serienummers kunnen ook gebruikt worden als er een retransmissie aangevraagd moet worden. Andere protocollen gebruiken serienummers die door de zender toegewezen worden of helemaal geen.

4.3.7 Heterogeniteit

Moderne genetwerkte virtuele omgevingen vereisen geen speciale hardware meer zoals in de tijd van de genetwerkte simulatoren (zie 2.1). Sinds de desktop VR is er enkel een stevige personal computer nodig. Ontwerpers van transportprotocollen moeten rekening houden met de grote verscheidenheid aan netwerkkaarten, routers, verbindingen en hun snelheid, ... om aan deze vraag te voldoen.

Enkele protocollen laten bijvoorbeeld toe verschillende levels of service te gebruiken, afhankelijk van de gebruikers. Sommige gebruikers hebben dan toch een unreliable datastroom omdat het nu eenmaal niet anders kan.

Data propagation

Bij de meeste multicast transportprotocollen wordt IP multicast gebruikt. Er zijn een aantal protocollen die voor een paar gebruikers overschakelen naar unicast[7] als IP multicast niet ondersteund wordt door het netwerk dat ze gebruiken.

4.4 Conclusie

Multicast is de meest efficiënte distributiemethode voor genetwerkte virtuele omgevingen. Met één transmissie bereikt men een selecte groep gebruikers. Momenteel is er nog veel onderzoek naar betrouwbare multicastprotocollen die bruikbaar zijn voor genetwerkte virtuele omgevingen en andere multi-user toepassingen over het Internet.

Voor een goede scalability raadt [53] aan een receiver-initiated reliability (met NACKs) met een mechanisme dat feedbackimplosie tegengaat te gebruiken. In [37] wordt er echter aangetoond dat, om de kans op een deadlock volledig uit te sluiten, alle packetten die de zenders versturen en verstuurd hebben in een geheugen moeten worden bijgehouden gedurende de hele sessie. Omdat op voorhand niet geweten kan zijn hoe veel packets er verstuurd zullen worden, zou er dus oneindig veel geheugen vereist kunnen zijn.

Een vorm van positive acknowledgement is daarom toch gewenst. De performance van sender-initiated multicasting is echter veel te afhankelijk van het aantal ontvangers binnen de multicastgroep. Tree-based transportprotocollen zijn de enige reliable multicast protocollen waarbij de throughput garanderen onafhankelijk is van het aantal gebruikers.

Deze resultaten zijn afkomstig uit het werk van Brian Neil Levine en Jose Joaquin Garcia-Luna-Aceves [37] en gebaseerd op hetgeen een node te verwerken krijgt in plaats van het bandbreedteverbruik. In de praktijk worden receiver-initiated echter wel gebruikt aangezien men toch een schatting kan doen naar de vereiste grootte van het memory allocation window.

Voor meer gedetailleerde informatie wordt er doorverwezen naar de werken [36, 37, 53, 68, 26, 51, 38] en de online informatie op http://rmt.motlabs.com/ en http://www.ietf.org/html.charters/rmt-charter.html.

[7] het gaat hier over gewone data, geen controle informatie

Hoofdstuk 5

Implementatie van het multicastframework

Voor deze thesis is er een multicast framework geïmplementeerd. Via een algemene C++ API (zie sectie 5.4) kunnen genetwerkte virtuele omgevingen en andere applicaties op een eenvoudige manier gebruik maken van verschillende (reliable) multicast transportprotocollen. Momenteel worden[1] MTP-2, NACK en RMF/RAMP ondersteund. Deze protocollen worden hieronder besproken.

5.1 Multicast Transport Protocol (MTP en MTP-2)

5.1.1 MTP

Group Management

Het Multicast Transport Protocol (MTP) werd omschreven in RFC 1301 [6] in 1992. Het is een zeer betrouwbaar, consistent multicast protocol en garandeert dat de ontvangers de volgorde van de berichten kennen. Hier komt echter veel overhead bij kijken en de kans op een bottleneck is zeer groot (zie verder).

Er zijn drie soorten leden bij MTP: de master, producers en consumers. Men noemt dit de "membership classes".

De master is het centraal orgaan dat het verkeer binnen de sessie controleert. Bijna alle controleinformatie wordt bekeken door de master en dit geeft een grote kans op een bottleneck. Er is slechts één master per sessie. Consumers zijn nodes die enkel data ontvangen en niet wensen een producer te zijn. De producers zijn de hosts die data kunnen verzenden.

[1] naast het gewone IP Multicast

De master heeft ook de rechten van een producer.

Als een host zich lid wil maken van de sessie, moet dat aangevraagd worden aan de master met een "join[request]". De host stuurt dit packet naar de multicastgroep en als er een master aanwezig is, zal deze de join toestaan of afwijzen (via unicast). In de join[request] staan allerhande gegevens waarmee de host zijn wensen uitdrukt.

Zo wordt duidelijk gemaakt van welke membership class de host is en of hij een reliable verbinding met de groep wilt of enkel best-effort. Waarden voor de minimumthroughput (in kilobytes per seconde), de maximale grootte van de data per packet, de duur van een heartbeat, hoe lang verzonden packets in het geheugen van een producer bijgehouden moeten worden[2] en de grootte van het congestion window[3] die acceptabel zijn voor de host worden doorgestuurd. Ook of de host van meerdere producers tegelijk kan ontvangen, het one-to-many of het many-to-many karakter van de sessie, wordt hier beslist.

Wanneer na een heartbeat er nog geen antwoord van de master is ontvangen geweest, dan kan de join[request] herhaald worden. Als de parameters hun waarde (vooral heartbeat, window en retention zijn belangrijk) aanvaardbaar zijn, stuurt de master een join[confirm] met daarin de huidige waarden die in de sessie gebruikt worden. Anders wordt de host geweigerd.

Bij het begin van een sessie moet er beslist worden welke host de rol van master heeft. Dit kan het MTP protocol niet oplossen, de applicatie moet dat doen. Een host die een nieuwe sessie wilt starten en daarvan master wilt zijn, moet naar het gekozen multicast-adres join[request]'s sturen. Indien er geen antwoord op komt, wordt er aangenomen dat het adres niet in gebruik is door een andere MTP-sessie en mag de host zich daar als master vestigen en andere hosts toevoegen aan de sessie.

Om de sessie te verlaten stuurt men via unicast een quit[request] naar de master. Als deze antwoordt met een quit[confirm]-packet, dan is de host uitgeschreven. Indien er na een heartbeat geen bevestiging heeft gekregen, dan stuurt men opnieuw een quit[request].

Leden die een overtreding begaan kunnen uit de groep gestoten worden door ze een quit[request] te sturen. Indien er door de master berichten ontvangen worden van hosts die geen lid (meer) zijn van de sessie, dan hoort deze altijd[4] te antwoorden met een quit[request].

Een master die uit de sessie wilt stappen moet dit duidelijk maken aan de hele groep. De master moet hiervoor alle transmit tokens (zie hieronder) in zijn bezit te hebben en kan dan een quit[request] multicasten naar het hele web. Dit packet moet bevestigd worden

[2] de "retention" (ρ), gemeten in heartbeats. Dit betekent dat de ontvangers ρ heartbeats de tijd krijgen om een packet te NACK'en.

[3] in RFC 1301 [6] gewoon "window" genoemd

[4] behalve als het om een join[request] gaat

door alle leden of er komen retransmissies[5]. Of hiermee de hele sessie is afgesloten of dat de applicatie in staat is een nieuwe master aan te duiden, wordt in de specificatie [6] niet vermeld.

Reliability

In MTP is een *message* de user data uit een reeks packets. Het verdelen van de data over packets is de verantwoordelijkheid van de applicatie, niet van het MTP-protocol. MTP heeft zelf geen ondersteuning voor fragmentatie. Een producer moet in het bezit zijn van een transmit token voordat het data (in de vorm van messages) mag doorsturen naar de groep. Deze worden via unicasting aan de master gevraagd.

Als de master het token doorgeeft aan de producer (token[confirm]), dan bevat het token ook het volgnummer van de message. Dit volgnummer mag enkel aangepast worden door de master en garandeert zo een globale volgorde van de berichten. De volgnummers van de afzonderlijke packets binnen de message worden toegewezen door de producer. Ieder packet dat verstuurd wordt door de producer bevat het gekregen message-nummer en een ingesteld packet-nummer. Met het laatste packet van de message (data[eom]) geeft de producer het token weer vrij aan de master.

Wanneer een lid één of meer packets gemist heeft, stuurt hij via unicast een NACK (met daarin de lijst van de gewenste packets hun nummers in stijgende volgorde) naar de producer, die dan via multicast de data opnieuw verstuurt naar de hele groep. Leden kunnen dus dezelfde packets meerdere malen ontvangen. Als de producer een packet niet meer in het geheugen heeft (ρ heartbeats zijn verstreken), wordt de aanvrager hier van op de hoogte gebracht (nak[deny]), zodat de applicatie van de aanvrager indien nodig kan reageren.

Om ervoor te zorgen dat een nieuw lid enkel volledige messages zal ontvangen, mag de master een join[request] niet goedkeuren vooraleer hij in het bezit is van alle transmit tokens. Anders kan door de nieuwe host een bericht ontvangen worden met een packet-nummer boven nul en zal deze alle packets met een kleiner packet-nummer NACK'en.

De transmit tokens beperken het aantal messages van producers dat tegelijkertijd bezig mag zijn (één message per token en het aantal tokens is beperkt) en is zo een soort congestiecontrole voor de hele groep. Daarnaast heeft in MTP iedere producer ook nog een congestion window dat het maximum aantal data-packets[6] bepaalt dat deze per heartbeat mag sturen.

[5] zoveel als de retention (ρ) bedraagt

[6] zowel nieuwe data-packets als retransmissies. De retransmissies hebben voorrang op nieuwe data. Hoe meer retransmissies er moeten gebeuren binnen een heartbeat, hoe minder nieuwe data-packets er tijdens die periode nog verstuurd kunnen worden.

Ontvangers die een packet gemist hebben, moeten rechtstreeks naar de producer een NACK sturen. Maar wat als het om een message gaat met maar één packet? Om dit probleem te voorkomen is er een minimum aantal (namelijk ρ) packets dat een message moet bevatten. Wordt dit aantal niet gehaald, dan wordt de message aangevuld met lege packets (empty[dally]) tot het aantal bereikt is. Dit is een nogal onhandige maatregel en zorgt voor zinloos bandbreedteverbruik.

Scalability

De parameters (heartbeat, window en retention) die de leden bij het begin van de sessie hebben doorgestuurd bepalen het minimum aan quality of service (QoS), het worst-casescenario, dat geaccepteerd wordt. Clients die zich hier niet aan kunnen houden kunnen verzocht worden uit de sessie te stappen met een quit[request].

Tijdens de sessie kunnen de QoS parameters aangepast worden. Dit is vooral wenselijk als de master in bezit is van alle tokens en er dus geen data verzonden wordt. Door de heartbeat te verlagen wordt er zo minder zinloos netwerkverkeer gecreëerd. Elke heartbeat worden er namelijk lege packets gestuurd om de verbinding te behouden.

Als er tokens uitgedeeld zijn, zullen producers de heartbeat willen verkleinen en het congestion window vergroten, zodat er minder kans is dat ze onnodig moeten wachten om data te mogen doorsturen.

Wegens het receiver-initiated karakter van MTP is het echter de verantwoordelijkheid van de consumers om problemen te melden. Bij congestieproblemen zullen zij voorstellen om de heartbeat te vergroten en het window te verkleinen. De retention kan dan ook verhoogd worden om meer retransmissies mogelijk te maken.

De parameters mogen echter nooit beneden het worst-casescenario gaan.

Het Multicast Transport Protocol is een goed reliable multicast transportprotocol. Een pluspunt is de garantie dat de volgorde van de messages bepaald kan worden. Er komt echter veel overhead bij kijken [11, 12].

De regel dat een message minstens ρ packets moet bevatten, kan voor veel zinloos bandbreedteverbruik zorgen. Voor het gebruik bij NVE's kan dit een groot verlies betekenen, aangezien de meerderheid van de netwerkcommunicatie bestaat uit statusupdates. Statusupdates zijn niet noodzakelijk lange messages en kunnen gerust uit slechts één packet bestaan. Als ρ de voorgestelde minimumwaarde 3 heeft [6], dan zijn er dus 2 empty[dally] packets nodig. Op het gebied van statusupdates is dat 200% verspilling van de beschikbare bandbreedte.

Het group management en de transmit tokens vereisen veel van de host die als master fungeert. Daarboven moet elk packet met sessieinformatie door de master verwerkt worden en dit geeft een potentiële bottleneck. De transmit tokens beperken het aantal producers

dat tegelijkertijd kan zenden. Dit geeft echter als probleem dat als de producers lange messages sturen deze de wachttijd op tokens voor andere zenders hoog kan oplopen.

Er is wel sprake van het aanpassen van de QoS parameters, maar in RFC 1301 wordt geen concrete methode gegeven om deze mogelijkheid toe te passen. Met MTP-2 komt daar verandering in.

Tabel 5.1: MTP

zenders	one-to-many of many-to-many (zie join[request])
verantwoordelijkheid verzoek retransmissie	receiver-initiated NACKs via unicast multicast
feedback control locus of control flow and congestion control	niet nadrukkelijk aanwezig gecentraliseerd (master) token- en window-based
group management late-joins	explicit de master moet alle transmit tokens in zijn bezit hebben
fragmentatie	verantwoordelijkheid van de applicatie
ordenen	volgorde mogelijk met meerdere bronnen
data propagation	multicast

5.1.2 MTP-2

Aan de Technische Universität van Berlijn heeft men het Multicast Transport Protocol herzien en er een verbeterde versie, MTP-2 [11], van gemaakt.

Het toetreden tot een sessie is vereenvoudigd. De join[confirm] bevat het huidige volgnummer waardoor nieuwe hosts berichten die ouder zijn dan het moment waarop ze lid werden van de groep kunnen negeren. Na de join[request] moeten ze dus niet meer wachten tot alle tokens in het bezit zijn van de master. De join latency is hierdoor verminderd.

Bij MTP was er niet duidelijk wat er moest gebeuren als de master de sessie verlaat of wegvalt. In MTP-2 merken de andere gebruikers wanneer de master onbereikbaar is geworden door het uitblijven van packets of nieuwe waarden voor de QoS-parameters (zie verder) en kan er een nieuwe master worden gekozen.

MTP bepaalt dat een message minstens ρ packets moet bevatten. Zoals hierboven besproken is dit zeer negatief. In MTP-2 is dit niet meer noodzakelijk. Het verbeterde protocol laat consumers toe om NACKs naar de master of andere producers te sturen als de eigenlijke zender van het packet onbekend is. De producer zal dit packet dan naar de zender doorsturen of naar de master als ook hij niet weet wie de zender is.

Het herversturen van data wordt niet meer noodzakelijk gemulticast. Dit gebeurt nu op een selectieve manier zodat enkel de delen van de groep die met congestie kampen de retransmissie ontvangen.

Bepaalde data is dringender dan andere. Daarvoor heeft MTP-2 twee oplossingen.

In de token[request] kan een melding gemaakt worden van de hogere prioriteit. De master zal het token dan eerder doorgeven aan deze producer.

Daarnaast gebruikt MTP-2 ook *streams*. De ene stream heeft een hogere prioriteit dan de andere en ze zijn volledig onafhankelijk van elkaar. Elke stream heeft zijn eigen messages. MTP-2 data-packets hebben naast een packet volgnummer en een message volgnummer ook een attribuut dat aanduidt tot welke stream het packet behoort. Zo komt de message ordening niet in het gedrang.

Token[request]'s waren een groot deel van het netwerkverkeer bij MTP. Het komt ook regelmatig voor dat een producer een nieuw token nodig heeft als hij al een message aan het verzenden is. Daarom staat MTP-2 om efficiëntie redenen toe een token-request flag te gebruiken bij data packets. Op deze manier moet er dan geen apart token[request] gestuurd worden.

Bij MTP was er ook het probleem van de lange messages. Door het beperkte aantal tokens was er het gevaar dat producers lang moesten wachten op het vrijkomen van een token als de messages die verstuurd werden lang waren. In MTP-2 kan de master messages afbreken en later laten voortzetten.

In RFC 1301 is er wel sprake van het aanpassen van de QoS parameters[7], maar er wordt geen concrete methode gegeven om deze mogelijkheid toe te passen. In MTP-2 kan enkel de master nieuwe waarden invullen. Om deze bekend te maken bij de andere gebruikers wordt het nieuwe packet empty[info] [11] gebruikt dat maximum ρ keer verstuurd wordt (één per heartbeat).

De bedoeling is dat producers die op dat moment een token bezitten de nieuwe waarden overnemen en in hun packets vermelden. Zodra de master een packet opmerkt met de nieuwe parameters, stopt hij met het verzenden van empty[info]. Zijn er geen producers die een token bezitten, dan wordt aangenomen dat ρ verzendingen van het packet genoeg zijn om alle leden te informeren.

[7] de heartbeat, retention ρ en window

Tabel 5.2: MTP-2

zenders	one-to-many of many-to-many
verantwoordelijkheid verzoek retransmissie	receiver-initiated NACKs via unicast selective
feedback control locus of control flow and congestion control	niet nadrukkelijk aanwezig gecentraliseerd (master) token- en window-based
group management late-joins	explicit zijn mogelijk zonder dat de master alle transmit tokens in zijn bezit moet hebben
fragmentatie	verantwoordelijkheid van de applicatie
ordenen	volgorde mogelijk met meerdere bronnen
data propagation	multicast

5.1.3 Implementatie in het multicast framework

Op `ftp://ftp.cs.tu-berlin.de/pub/local/kbs/mtp/mtp-2/` staat de code van een implementatie van MTP-2, de laatste versie dateert echter van 1996. Deze versie is beschikbaar voor Solaris en Sun OS, daarom werd het voor het gebruik in het framework eerst geport naar Win32.

MTP-2 bestaat uit een library en een daemon, die door de library wordt opgestart. De daemon kan via de library worden aangesproken en verzorgt het netwerkverkeer.

5.2 Reliable Adaptive Multicast Protocol (RAMP)

Het Reliable Adaptive Multicast Protocol (RAMP) werd ontwikkeld aan TASC en door Stephen Zabele en anderen besproken in RFC 1458 [12] en [34]. Het zorgt voor betrouwbaar netwerkverkeer van geordende packets en probeert de feedback te beperken.

In het multicastframework dat bij deze thesis hoort, wordt RAMP gebruikt in combinatie met het Reliable Multicast Framework of RMF (zie sectie 5.2.4).

5.2.1 Group Management

Als een host zich wil aansluiten bij een sessie, maakt deze (zoals bij MTP) duidelijk of hij een zender of een ontvanger is. Dit gebeurt door het sturen van respectievelijk een *connection request* (CR) of een *accept response* (AR) naar een multicast adres. Als er al ontvangers zijn en een zender wilt zich aansluiten, dan moeten aanwezige ontvangers een AR op de CR antwoorden. De zender mag beginnen multicasten zodra hij minstens één AR ontvangt. Op dat moment spreekt men van een verbinding tussen de zender en de ontvanger(s).

Een verbinding die opgestart wordt door een CR wordt "active open" genoemd, een verbinding die opgestart wordt door een AR wordt "passive open" genoemd.

Wanneer een ontvanger een CR of een data-packet ontvangt van een zender waarmee hij nog niet in verbinding staat (voor de controleinformatie, zie verder), dan wordt er een AR naar deze zender gestuurd en gaat de ontvanger daar een nieuwe verbinding mee aan. Een ontvanger kan tegelijkertijd in verbinding staan met meerdere zenders op hetzelfde multicastadres, maar elke verbinding wordt apart beheerd.

Bij het late-joinen van een zender is er geen probleem, want de connectie is in wezen nog niet opgestart (één RAMP-verbinding per zender). Als een ontvanger later bij de groep komt, is het eerste bericht dat hij van de zender ontvangt een data-packet. Om problemen te vermijden wordt het volgnummer van dit packet door de ontvanger aangenomen als het eerste nummer van zijn sessie en mag hij geen data aanvragen met een nummer dat lager is. De ontvanger stuurt dan een AR en maakt zich zo kenbaar aan de zender.

De zender staat namelijk in voor het group management en voegt het nieuwe lid toe aan zijn lijst van ontvangers. Om de zender duidelijk te maken dat de ontvangers nog altijd aanwezig zijn, moeten dezen regelmatig een idle-packet naar de zender sturen (via unicast). Als de time-out periode hiervoor bij de zender is afgelopen zonder dat er een bericht van een bepaald lid is ontvangen, dan wordt de connectie daarmee opgeheven.

5.2.2 Reliability

Zenders kunnen in hun messages een flag zetten dat de berichten unreliable doorgestuurd worden. Hiermee maakt de zender duidelijk dat er geen feedback verwacht wordt. Ook de ontvangers hebben de mogelijkheid om (onafhankelijk van de zender) unreliable te werken, dan controleren ze niet meer of een packet verloren is gegaan of niet en produceren dus geen feedback.

Door de ordening van de packets kunnen verloren packets met behulp van de volgnummers opgespoord worden. Als een ontvanger merkt dat er een volgnummer mist bij de ontvangen packets, wordt er een NACK (met daarin het nummer van de gemiste data) gestuurd naar de verzender van dat packet. Dit gebeurt via unicasting.

Hoe verlopen de retransmissies? Volgens de RFC is multicasten van een packet dat slechts dooreen klein aantal hosts werd aangevraagd namelijk onnodige belasting van het netwerk. Enkel unicast gebruiken is echter ook geen goede oplossing, want wat als heel de groep een packet opnieuw aanvraagt?

Daarom behandelt de zender een binnengekomen NACK niet meteen, maar wacht naargelang de waarde van de "retransmission hold timer" om zo een aantal NACKs tegelijkertijd af te kunnen handelen. Als er dan verscheidene NACKs van ontvangers aankomen voor hetzelfde datapacket, zal de zender de data opnieuw sturen naar de groep via multicast. Is het slechts een klein aantal ontvangers dat problemen ondervond, dan zullen deze de data via unicast toegestuurd krijgen. De threshold om te beslissen wanneer te multicasten en wanneer niet, wordt op voorhand vastgelegd.

Als er veel NACKs binnenkomen, wordt de retransmission hold timer vermeerderd. Zo wordt de kans groter dat men meer NACKs voor hetzelfde packet in één keer kan behandelen en verkleint men de kans op dubbele retransmissies.

Wanneer de data die gevraagd wordt zich niet meer in het geheugen van de zender bevindt, stuurt de zender (zoals bij MTP) een NACK naar de aanvragers (zij het via multicast of unicast).

5.2.3 Scalability

Via het aantal aanvragen voor het herverzenden van data, kan men een schatting maken naar de data loss op het netwerk en zo de "rate reduction factor" (RRF) berekenen. De RRF bepaalt de tijd (de "delay") die de zender tussen packets moet laten. Wanneer er NACKs binnenkomen[8] kan de RRF verhoogd worden. Zo lang er geen NACKs aankomen mag men sneller verzenden.

De RRF wordt als volgt berekend:

$$RRF = \frac{LostSeg \times Weight}{SentSeg} \tag{5.1}$$

met

LostSeg = het aantal packets dat verloren gegaan is

Weight = een waarde tussen 0 en 1 die gebruikt kan worden om de vertraging per verloren packet zwaarder te laten doorwegen of net niet.

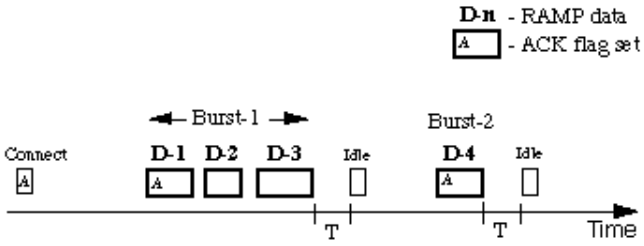
SentSeg = het aantal verstuurde packets

[8] of een bericht van de router om de snelheid te verminderen

De zender kan tijdens de sessie in twee modi functioneren: burst of idle. Tijdens de sessie kan er veranderd worden van modus. De modus wordt duidelijk gemaakt aan de ontvangers door een mode flag.

In **burst mode** stuurt de zender aan het begin van een "burst" een Acknowledge Flag mee. Een ACK flag wordt gebruikt om zender en ontvangers te synchroniseren. Wanneer deze flag gezet is, moeten de ontvangers ook een ACK (met daarin het volgnummer van het packet) naar de zender sturen om de synchronisatie geldig te maken. Een burst is een reeks berichten waarvan de tijd tussen twee opeenvolgende berichten kleiner is dan een vastgelegde tijdspanne (T in figuur 5.1). Is de tijd tussen twee berichten groter dan T, dan is de burst gedaan en deze wordt door de zender beëindigd met een idle-packet. Het idle-packet heeft hetzelfde volgnummer als het voorafgaande data-packet. In burst mode worden er nu geen packets meer gestuurd door de zender tot er zich weer verzendbare data voordoet. Als dit gebeurt, start de zender weer een burst (met ACK flag).

Als de zender na het beginnen van een burst geen ACK ontvangt van een bepaald lid, dan weet de zender dat er data (of zelfs de verbinding met de ontvanger) verloren is gegaan. De zender zal de data opnieuw versturen[9]. Als na een aantal retransmissies de ontvanger nog altijd niets teruggestuurd heeft, dan wordt de verbinding met die host verbroken. Burst mode is enkel aangewezen bij kleine groepen ontvangers. Het gebruikt minder netwerkverkeer maar vergt meer controleinformatie (de ACKs). De overhead aan feedback moet opwegen tegen de winst aan bandbreedte voor de data.



Figuur 5.1: Burst mode. bron: [34]

In **idle mode** stuurt de zender een reeks idle-packets in plaats van slechts één als er geen (nieuwe) data te versturen is. Iedere tijdsperiode T moet er nieuwe data of een idle-bericht gestuurd worden door de zender. Als een ontvanger geen packet ontvangt voor een tijd langer dan T, dan is er data loss en wordt er een NACK gestuurd naar de zender. Belangrijk is hier op te merken dat retransmissies van data packets niet gezien worden als "nieuwe data". Retransmissies resetten de timers bij de ontvangers niet[10] en daarom moet

[9] dit maakt is RAMP sender-initiated (zie sectie 4.3.2) aan het begin van elke burst in burst mode.

[10] retransmissies kunnen namelijk langs multicast of unicast verstuurd worden dus niet iedereen ontvangt

er evenzeer een idle-packet gestuurd worden.

Idle mode is aangeraden voor grote groepen. De zender stuurt meer packets (de idle-berichten) naar de groep, maar heeft veel minder controleinformatie te verwerken aangezien er geen ACKs nodig zijn.

Wanneer een message te groot is om in één data packet te passen zal de zender deze opsplitsen in verschillende segmenten die wel in een data packet passen. Aan de ontvangstzijde zullen de hosts de segmenten terug samenvoegen tot de oorspronkelijke message.

Tabel 5.3: RAMP

zenders	one-to-many many-to-many wordt mogelijk gemaakt door het gebruik van verschillende connecties (één per zender)
verantwoordelijkheid verzoek retransmissie	receiver-initiated en sender-initiated aan het begin van elke burst in burst mode NACKs via unicast via unicast of multicast
feedback control locus of control flow and congestion control	niet nadrukkelijk aanwezig gedistribueerd rate-based aanpasbaar met rate reduction factor (zie formule 5.1)
group management late-joins	explicit group management door de zenders zijn mogelijk
fragmentatie	wordt ondersteund
ordenen	één bron (verbindingen worden apart beheerd)
data propagation	multicast

De manier waarop RAMP zijn retransmissies verstuurt is interessant. De retransmission hold timer betekent wel overhead, maar geeft een mogelijkheid om te beslissen tussen multicast of selective repeat. Multicasting van retransmissies is onnodig als er slechts één ontvanger is die het packet gemist heeft en unicasting geeft problemen als een groot deel van de groep een retransmissie nodig heeft.

noodzakelijkerwijs het packet.

5.2.4 Reliable Multicast Framework (RMF)

Het Reliable Multicast Framework (RMF) [18] is geen specifiek protocol, maar biedt een structuur die gebruikt kan worden om een reeks (reliable) multicast transportprotocollen te definiëren. Het doel is dat men uiteindelijk als ontwerper zelf de nodige protocolvereisten kan instellen voor de doelapplicatie.

RMF gebruikt een one-to-many multicasting structuur, de zender kiest het multicast-algoritme dat gebruikt zal worden en bepaalt zo het gedrag van de ontvangers. Zoals bij RAMP kan er echter een many-to-many sessie bekomen worden door iedere zender als een aparte sessie te behandelen. RMF kan dus in principe geen betere message ordening geven dan die gebaseerd op één bron.

Self-identifying packets

De data packets die de zender verstuurd zijn self-identifying packets. Dit wil zeggen dat het bericht zelf informatie bevat om aan de ontvanger duidelijk te maken wat zijn gewenste gedrag is. Zo krijgt men *per-packet reliability*[18].

Door het lezen van het bericht weet de universal receiver (zie verder) welk soort acknowledgement gewenst is en of dit gemulticast of geünicast moet worden.

Data-level en session-level informatie

RMF maakt een verschil tussen data-level informatie en session-level informatie.

Het data-level is alles wat handelt over de inhoud van de messages, de echte data die door de applicaties doorgegeven wordt.

Naast de self-identifying packets is er ook session-level informatie die het algemene karakter van de sessie meebepaalt. Dit is meestal optionele informatie zoals naar welk adres acknowledgements gestuurd moeten worden, de specificatie van time-outs, ...

Meer algemeen kan men stellen dat de self-identifying packets op zichzelf staan en enkel informatie geven over hetgeen er moet gebeuren bij het ontvangen van dat bepaald bericht. State-level informatie bepaalt mee het algemene karakter van de sessie. Aan het begin van een sessie gebeurt de configuratie van de ontvangers (de Universal Receivers, zie hieronder) en hier kan al algemene session-level informatie doorgeven worden.

Universal Receiver

Door het gebruik van self-identifying packets wordt een implementatie van een Universal Receiver (UR) mogelijk, deze werkt onafhankelijk van een specifiek protocol. De UR moet

enkel een aantal attributen (die gezet zijn door de zender) die gespecificeerd zijn binnen het packet en eventueel sessioninformatie bekijken om te weten hoe te communiceren met de zender of welke reacties er van hem verwacht worden.

In principe kan men zo bestaande multicast transportprotocollen vertalen naar RMF en dat is wel heel interessant: enkel de zender moet op de hoogte zijn van de specificaties, door de UR kan iedere applicatie zonder problemen een ontvanger zijn. Het verschil in de protocollen zit dan in de attributen van de data packets en de session-level informatie.

Dit geeft ook als groot voordeel dat men bij het overschakelen naar een ander transport-protocol (of één met andere specificaties) men enkel de zender(s) moet aanpassen.

5.2.5 Implementatie in het multicast framework

Voor het multicastframework, dat deel uitmaakt van deze thesis, wordt er gebruik gemaakt van de RMF/RAMP implementatie van TASC [67].

Het one-to-many karakter, dat weliswaar omzeild kan worden door voor iedere zender een aparte verbinding op te starten, is echter geen goed uitgangspunt. Ook werd vastgesteld dat RMF/RAMP veel geheugen gebruikt en veel processortijd eist. Daardoor is RMF/RAMP momenteel nog niet praktisch voor genetwerkte virtuele omgevingen.

5.3 NACK-Oriented Reliable Multicast Protocol (NORM)

Het NACK-Oriented Reliable Multicast Protocol (NORM) [1, 2, 3, 50] is ontwikkeld aan het Naval Research Laboratory door Brian Adamson en Kenneth W. Flynn.

5.3.1 Group Management

Zodra er aan het begin van een sessie een zender aanwezig is, begint deze packets rond te sturen om de round-trip tijd en dergelijke informatie te kunnen vaststellen. Dit wordt gebruikt voor de congestiecontrole (zie sectie 5.3.3). Afhankelijk van de toepassing zal de zender nu wachten tot er zich ontvangers aandienen of meteen data beginnen te zenden als deze zich voordoet.

Het is geen probleem als een ontvanger zich aansluit bij de groep terwijl er al een sessie bezig is. Zoals bij RAMP mag die ontvanger geen NACK sturen voor packets met een volgnummer (NormTransportId) dat kleiner of gelijk is aan die van het eerste data packet

(dat geen retransmissie is) dat hij binnenkrijgt.

Zenders bepalen zelf het NormTransportId van hun packets. Toch is ieder packet te identificeren. Dit kan door het te samen gebruiken van de NormNodeId (uniek id voor een lid van de sessie, hier een zender) en de NormTransporId. Als bijvoorbeeld zender A een packet met NormTransportId m_x gestuurd heeft en zender B een m_y, dan kunnen de ontvangers niet weten of Am_x eerder verstuurd was dan Bm_y. Message ordening is bij NORM enkel mogelijk voor packets die van dezelfde zender komen (één bron).

5.3.2 Reliability

NORM kan drie soorten data aan: static data, files en oneindige streams van data (de drie soorten NormObjects). Als er data verloren gaat, probeert men dit te herstellen met de parity informatie die de ontvangers kregen door het gebruik van Forward Error Correction of FEC (zie sectie 4.3.2). Wanneer dit niet haalbaar blijkt te zijn, dan kunnen NACKs gestuurd worden voor het afhandelen van de data loss.

Voor het gebruik van FEC wordt er met elk data block een parity block geassocieerd. Hoe groter dit block, hoe beter het berekenen van de verloren data lukt, maar ook hoe meer processortijd er nodig is voor het aanmaken (aan de zenderzijde) en het verwerken (aan de ontvangstzijde) ervan. Er moet afgewogen worden tussen latency en overhead.

Er zijn drie mogelijkheden: geen parity blocks gebruiken en gedeeltelijke of volledige pariteit. Als er *geen parity blocks* gebruikt worden, vertrouwt men bij data loss volledig op NACKS. Hierdoor moeten de hosts bij data loss langer wachten vooraleer een bericht doorgegeven kan worden aan de applicatie (latency). *Volledige pariteit* geeft de meeste processoroverhead maar voorkomt wel vele aanvragen voor het herversturen van data en de retransmissies zelf.

De middenweg is *gedeeltelijke pariteit*. Hier worden kleinere parity blocks gebruikt zodat er niet te veel geëist wordt van de processor. Kan men de verloren data niet herstellen, dan worden er met NACKs extra parity blocks aangevraagd. NACKs kunnen naar de hele groep gemulticast worden of via unicast uitsluitend naar de zender gestuurd worden.

In de huidige implementatie van NORM worden standaard geen parity blocks met de data meegestuurd. Als er zich data loss voordoet, kunnen de ontvangers indien nodig parity blocks aanvragen met NACKs.

De data die de zender krijgt van zijn applicatie wordt indien nodig opgesplitst in kleinere segmenten vooraleer ze verzonden worden. Dit opsplitsen moet nauwkeurig gebeuren, want de Forward Error Correction hangt af van de data die zich in een packet bevindt.

Om de zender niet te overspoelen met NACKs, gebruikt NORM een timer-based feedback control mechanisme (zie 4.3.3). De ontvanger wacht een willekeurige tijd[11] vooraleer een NACK (of een herhaling daarvan) te sturen naar de zender.

Net zoals bij RAMP beantwoordt de zender repair requests niet meteen. Er wordt een tijd gewacht in de hoop een aantal NACKs tegelijkertijd te kunnen oplossen. In plaats van retransmissies, worden er meestal parity blocks gestuurd. Als er geen repair requests meer in de buffer van de zender zitten, dan veronderstelt deze terug nieuwe data te mogen verzenden. Moet er geen data meer verzonden worden, dan worden er idle-packets (NORM_CMD(FLUSH) in [1]) gestuurd, één per de tijd die nodig is voor een dubbele round-trip, om de verbinding te behouden.

5.3.3 Scalability

In [1] wordt er beweerd dat NORM in een internetomgeving schaleerbaar is tot tienduizenden ontvangers. De ontvangers binnen de multicastgroep moeten wel de status van de actieve zenders bijhouden. Dit kan het aantal gelijktijdige zenders binnen een sessie beperken.

De congestiecontrole bij NORM verloopt rate-based. De rate is ofwel een vaste waarde, ofwel wordt deze aangepast aan de hand van feedback (NORM_CMD(CC)) over opstoppingen. NORM_CMD(CC) packets worden door de zender gestuurd aan het begin van de sessie, wanneer de zender geen data te verzenden heeft of wanneer de zender geen feedback meer krijgt van een bepaalde ontvanger die de Current Limited Receiver (CLR) wordt genoemd.

5.3.4 Implementatie in het multicast framework

De code van NORM kan afgehaald worden van http://pf.itd.nrl.navy.mil/projects.php?name=norm en gebruikt de Protean Protocol Prototyping library http://pf.itd.nrl.navy.mil/projects.php?name=protolib.

5.4 Interface van het multicast framework

De class CwMAPI verzorgt de algemene interface van het framework. Bij het starten van een sessie moet meegegeven worden welk transportprotocol men wenst te gebruiken.

De standaardfuncties zijn Subscribe, Unsubscribe, UnsubscribeAll, SendTo, Peek, ReceiveFrom, GetMulticastType en List:

[11] in [1] "random backoff timeout" genoemd.

Tabel 5.4: NORM

zenders	many-to-many
verantwoordelijkheid verzoek retransmissie	receiver-initiated en forward error correction NACKs via multicast of unicast via multicast of selective
feedback control locus of control flow and congestion control	timer-based gedistribueerd rate-based
group management late-joins	implicit wordt ondersteund
fragmentatie	wordt ondersteund
ordenen	één bron (verbindingen worden apart beheerd)
data propagation	multicast (unicast mogelijk)

Subscribe(multicastgroep, poort) schrijft de host in voor een multicastgroep. Verschillende inschrijvingen zijn mogelijk tijdens een sessie.

Unsubscribe(multicastgroep, poort) schrijft de host uit een multicastgroep.

UnsubscribeAll() schrijft de host uit voor alle multicastgroepen die deel uitmaken van de sessie.

SendTo(multicastgroep, poort,data) zendt data naar een multicastgroep. De host moet niet noodzakelijk ingeschreven zijn op de multicastgroep om er data naar te versturen.

Peek(multicastgroep, poort) controleert of er data klaar staat om te ontvangen en geeft de lengte van de data terug. Indien er geen data klaarstaat, wordt lengte 0 teruggegeven.

ReceiveFrom(multicastgroep, poort) Na Peek() kan men data ontvangen van de multicastgroep.

GetMulticastType() geeft terug welk transportprotocol er momenteel gebruikt wordt.

List() geeft een lijst terug van alle multicastgroepen (met poortnummer) waar de host voor is ingeschreven.

Conclusie

Het **Multicast Transport Protocol** is een zeer reliable multicast transportprotocol, bewaart de volgorde van messages van meerdere bronnen en heeft een many-to-many karakter. In het framework wordt MTP-2 geïmplementeerd, een Win32-port van het framework ontwikkeld door Hans-Christian Gehrcke, Torsten Kerschat en Nils Seifert aan de Technische Universität van Berlijn.

Bij MTP is er veel overhead [11, 12]. MTP-2 is een goede verbetering van zijn voorganger. Het group management is verbeterd[12], de regel dat een message minstens ρ packets moet bevatten is weggevallen en lange messages kunnen onderbroken worden.

Door de gecentraliseerde aanpak is er echter veel kans op een bottle-neck bij de master en dit blijft het grote minpunt van dit protocol.

Het **Reliable Adaptive Multicast Protocol** wordt niet als dusdanig geïmplementeerd in het framework dat deel uitmaakt van deze thesis, maar wordt gebruikt door het Reliable Multicast Framework. RAMP is het enige bestaande protocol dat momenteel ondersteund wordt door RMF. RAMP belooft (in idle mode) zeer interessant te zijn en het gebruik van een "retransmission hold timer" maakt het mogelijk op een verantwoorde manier te beslissen over een multicast of een selective repeat. De burst mode lijkt echter onhandig door het sender-initiated begin, maar kan misschien getolereerd worden bij heel kleine groepen.

Het idee achter RMF is zeer interessant. Het gebruik van self-identifying packets en de scheiding tussen session-level en data-level informatie legt de verantwoordelijkheid van het goed functioneren van het protocol bij de zender en maakt het mogelijk een "Universal Receiver" te creëren. Enkel de zender moet op de hoogte zijn van de specificaties, iedere applicatie kan zonder problemen een ontvanger zijn. Het verschil in de protocollen zit dan in de attributen van de data packets en de session-level informatie. Bij het overschakelen naar een ander transportprotocol (of één met andere specificaties) hoeft men enkel de zender aan te passen.

Maar het project is blijkbaar stopgezet. Sinds 2001 zijn er geen publicaties of drafts meer te vinden en tascnets.com wordt niet meer gebruikt voor de ontwikkeling van RMF. Daarenboven vereist de huidige implementatie van RMF/RAMP veel processor- en geheugenver-

[12] de master moet niet meer alle transmit tokens in zijn bezit hebben voor een extra host mag deelnemen

bruik. Het Reliable Multicast Framework is wel een goed idee, maar kennelijk moeilijk om in de praktijk te realiseren. Tijdens de eerste testen kwamen niet alle packets aan en de grafische voorstelling van de NVE kon zich niet renderen omdat RMF alle resources van de host bezette.

Het **NACK-Oriented Reliable Multicast Protocol** maakt gebruik van Forward Error Correction en probeert met gedeeltelijke pariteit het aantal NACKs te beperken. Een andere techniek om de feedback te drukken, is de timer-based methode waarbij iedere ontvanger een willekeurige tijd wacht vooraleer een NACK te sturen. Daarnaast maakt NORM, net zoals RAMP, ook gebruik van een retransmission hold timer.

Alhoewel oorspronkelijk ontwikkeld voor het uitwisselen van bestanden, is het ook mogelijk om statische data en streams te verzenden met NORM. Daarom lijkt NORM toch een goede mogelijkheid voor het gebruik bij genetwerkte virtuele omgevingen.

Deze drie reliable multicast transportprotocollen worden gebruikt in het multicastframework dat deel uitmaakt van deze thesis. Op het einde van het vorige hoofdstuk werd hiervan de interface toegelicht. Via deze algemene interface kan men door het verzetten van een variabele een ander transportprotocol gebruiken. Zo kan getest worden welk protocol het meest geschikt en scalable is vvor de applicatie, zonder veel code van de NVE te moeten veranderen.

Het zou interessant zijn om in de toekomst opnieuw een poging te wagen voor een algemeen framework zoals RMF, dat geen specifiek transportprotocol ondersteunt, maar een universal receiver gebruikt die zich schikt naar de wensen van de zender. Dit zou het ontwikkelen van genetwerkte virtuele omgevingen vergemakkelijken. Hiervoor moet er echter nog veel onderzoek gedaan worden naar reliable multicasting en de schaleerbaarheid daarvan.

Bibliografie

[1] B. Adamson, C. Bormann, M. Handley and J. Macker. *NACK-Oriented Reliable Multicast Protocol (NORM)*. Internet Draft RMT Working Group, May 2004

[2] B. Adamson, C. Bormann, M. Handley and J. Macker. *NACK-Oriented Reliable Multicast (NORM) Building Blocks*. Internet Draft RMT Working Group, May 2004.

[3] B. Adamson and J. P. Macker. *Quantitative Prediction of NACK-Oriented Reliable Multicast (NORM) Feedback*. Internet Draft, Information Technology Division Naval Research Laboratory

[4] D. B. Anderson, J. W. Barrus, J. H. Howard, C. Rich, C. Shen and R. C. Waters. *Building Multi-User Interactive Multimedia Environments at MERL*. IEEE Multimedia, Vol. 2, No. 4, Winter 1995, pp. 77-82

[5] M. Andersson, C. Carsson, O. Hagsand and O. Støahl. *DIVE - The Distributed Interactive Virtual Environment Technical Reference Manual*. Kista, Swedish Institute of Computer Science, March 1994

[6] S. Armstrong, A. Freier and K. Marzullo. *Multicast Transport Protocol* RFC 1301, February 1992.

[7] R. A. Bangun and H. W. P. Beadle. *A Network Architecture For Multiuser Networked Games on Demand*. Proceedings of the 1997 International Conference on Information, Communications and Signal Processing, pages 1815-1819, Singapore, September 1997.

[8] D. Bauer, S. Rooney and P. Scotton. *Network Infrastructure for Massively Distributed Games*. Proceedings of the 1st workshop on Network and system support for games, NetGames 2002, pp. 36 - 43

[9] A. R. Bharambe, S. Rao and S. Seshan *Mercury: A Scalable Publish-Subscribe System for Internet Games*. Proceedings of the 1st workshop on Network and system support for games, NetGames 2002, pp. 3 - 9

[10] C. Bormann, J. Ott and N. Seifert. *MTP/SO: Self-Organizing Multicast*. Internet Draft http://www-rn.informatik.uni-bremen.de/research/som.htm.

[11] C. Bormann, J. Ott, H.-C. Gehrcke, T. Kerschat and N. Seifert. *MTP-2: Towards Achieving the S.E.R.O. Properties for Multicast Transport.* Presented at the ICCCN '94, San Francisco, September 1994.

[12] B. Braudes and S. Zabele. *Requirements for Multicast Protocols.* RFC 1458, May 1993.

[13] M. Capps, P. McDowell and M. Zyda. *A Future for Entertainment-Defense Research Collaboration.* IEEE Computer Graphics and Applications, vol. 21, no. 1, 2001.

[14] M. Capps, D. McGregor, D. Brutzman and M. Zyda. *NPSNET-V: A New Beginning for Dynamically Extensible Virtual Environments.* IEEE Computer Graphics and Applications, 2000.

[15] K. O'Connell, T. Dinneen, S. Collins, B. Tangney, N. Harris, and V. Cahill. *Techniques for Handling Scale and Distribution in Virtual Worlds.* Proceedings of the 7th workshop on ACM SIGOPS European workshop: Systems support for worldwide applications.

[16] E. Cronin, B. Filstrup, and A. Kurc. *A Distributed Multiplayer Game Server System.* UM EECS589 Course Project Report, May 4, 2001

[17] C. Cruz-Neira, D. Sandin, and T. DeFanti. *Surround-screen projection-based virtual reality: The design and implementation of the CAVE.* Computer Graphics (Proceedings of SIGGRAPH '93), pages 135-142, Aug. 1993.

[18] B. DeCleene, S. Bhattacharaya, T. Friedman, M.. Keaton, J. Kurose, D. Rubenstein and D. Towsley. *Reliable Multicast Framework (RMF): A White Paper.* Technical report, 1998.

[19] DIS *Standard for information technology, Protocols for distributed interactive simulation.* ANSI/IEEE std 1278-1993, March 1993.

[20] DIS Steering Committe. *The DIS Vision: A map to the future of distributed simulation.* Ver3 1, IST-SP-94-01, May 1994.

[21] S. Fiedler, M. Wallner and M. Weber *A Communication Architecture For Massive Multiplayer Games.* Proceedings of the 1st workshop on Network and system support for games, NetGames 2002, pp. 14 - 22

[22] E. A. Fitzsimmons, and J. D. Fletcher. *Beyond DoD: Non-defense training and education applications of DIS.* Proceedings of the IEEE, 83(8):1179-1187, Aug. 1995.

[23] S. Floyd, V. Jacobson, C. Liu, S. McCanne and L. Zhang. *A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing.* IEEE/ACM Transactions on Networking, 5(6), 784-803, 1997

[24] I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure.* Morgan Kaufmann Publishers, 1999.

[25] E. Frécon and M. Stenius. *DIVE: A Scaleable network architecture for Distributed Virtual Environments*. Distributed Systems Engineering Journal (special issue on Distributed Virtual Environments), Vol. 5, No. 3, Sept. 1998, pp. 91-100.

[26] R. Galli. *Group Communication Support and Database Consistency Methods for a Multisite-Collaborative 3D Editor*. Research Report, May, 1998.

[27] A. Garg, Sneha K. Kasera, R. Kumar and Don Towsley. *Measurement of Join Latency on the Mbone*, UMass Technical Report, CMPSCI TR99-47, August 1999.

[28] C. Greenhalgh. *Dynamic, embodied, multicast groups in MASSIVE-2*. Technical Report NOTTCS-TR-96-8, Department of Computer Science, University of Nottingham. December 8, 1996.

[29] C. Greenhalgh. *Spatial Scope and Multicast in Large Virtual Environments*. Technical Report NOTTCS-TR-96-7, Department of Computer Science, University of Nottingham. September 25, 1996.

[30] C. Greenhalgh and S. Benford. *Massive: A distributed virtual reality system incorporating spatial trading*. Proceedings of the 15th International Conference, volume 1242 of Lecture Notes in Computer Science, pages 113-128, Milan, Italy, May 21-23 1997. Springer Verlag.

[31] C. Griwodz *State Replication For Multiplayer Games*. Proceedings of the 1st workshop on Network and system support for games, NetGames 2002, p. 29 - 35

[32] M. Hayden. *The Ensemble system*. Technical Report TR98-1662, Cornell University, January 1998.

[33] Institute of Electrical and Electronics Engineers. *Standard for Distributed Interactive Simulation, 1995*. Revision of IEEE Std 1278-1993.

[34] A. Koifman and S. Zabele. *RAMP: A Reliable Adaptive Multicast Protocol*. INFOCOM (3), 1996.

[35] R. Lea, P. G. Raverdy, Y. Honda, and K. Matsuda. *Scaling a shared virtual environment*. Sony Computer Science Lab. Tokyo, Japan

[36] B. N. Levine and J. J. Garcia-Luna-Aceves. *A Comparison of Known Classes of Reliable Multicast Protocols*. Proc. Int'I. Conf. Network Protocols '96, October 1996.

[37] B. N. Levine and J. J. Garcia-Luna-Aceves. *A Comparison of Reliable Multicast Protocols*. Springer-Verlag, Multimedia Systems, Volume 6 , Issue 5 , September 1998.

[38] D. Li and D. R. Cheriton. *Evaluating the Utility of FEC with Reliable Multicast*. Proceedings of 7th IEEE International Conference on Network Protocols, ICNP October 31 - November 03 1999.

[39] M. Lim and D. Lee. *Improving Scalability Using Sub-Regions in Distributed Virtual Environments*. International Conference on Artificial Reality and Telexistence, Tokyo Japan, December 1999, pp. 179 - 184.

[40] J. Locke. *An introduction to the Internet networking environment and SIMNET/DIS*. unpublished Master's thesis, Naval Postgraduate School, 1995.

[41] M. R. Macedonia and M. J. Zyda. *A taxonomy for networked virtual environments*. IEEE Multimedia, 4(1):48-56, Jan.-Mar. 1997.

[42] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz. *NPSNET: A Network Software Architecture For Large Scale Virtual Environments*. Presence: Teleoperators and Virtual Environments, MIT Press, Boston, Vol. 3, Issue 4 - Fall 1994.

[43] J. P. Macker, J. E. Klinker and M. S. Corson. *Reliable Multicast Data Delivery for Military Networking*. Submitted to IEEE MILCOM 96' Conference.

[44] D. Marshall, D. Delaney, S. McLoone and T. Ward. *Challenges in modern Distributed Interactive Application design*. Technical Report: NUIM-CS-TR2004-02, January 2004.

[45] M. Matijašević. *A Review of Networked Multi-User Virtual Environments*. TR 97-8-1.

[46] M. Mauve and V. Hilt. *An Application Developer's Perspective on Reliable Multicast for Distributed Interactive Media*. ACM SIGCOMM Computer Communication Review, Volume 30 , Issue 3, July 2000, pp. 28 - 38.

[47] D. C. Miller and J. A. Thorpe. *SIMNET: The advent of simulator networking*. Proceedings of the IEEE, 83(8):1114-1123, Aug. 1995.

[48] K. L. Morse, L. Bic and M. Dillencourt. *Interest Management in Large-Scale Virtual Environments*. Presence Vol. 9, Issue 1 - February 2000 pp. 52-68.

[49] K. L. Morse and M. Zyda. *Multicast Grouping for Data Distribution Management*. Simulation Practice and Theory, Volume 9, Numbers 3-5, 15 April 2002.

[50] Naval Research Laboratory Protocol Engineering Advanced Networking Research Group. *NACK-Oriented Reliable Multicast*. http://norm.pf.itd.nrl.navy.mil/.

[51] J. Nonnenmacher and E. W. Biersack. *Reliable Multicast: Where to use FEC*. Proceedings of the TC6 WG6.1/6.4 Fifth International Workshop on Protocols for High-Speed Networks V, p.134-148, October 28-30, 1996.

[52] V. Normand et al. *The COVEN Project: Exploring Applicative, Technical and Usage Dimensions of Collaborative Virtual Environment*. Presence: Teleoperators and Virtual Environments, Vol. 8, Issue 2, MIT Press, April 1999, pp. 218 - 236.

[53] K. Obraczka. *Multicast Transport Protocols: A Survey and Taxonomy.* IEEE Communications Magazine, January 1998 pp. 94-102

[54] M. Oliveira, J. Mortensen, J. Jordan, A. Steed, and M. Slater. *The Pitfalls in System Design for Distributed Virtual Environments: A Case Study.* International Workshop on Immersive Telepresence (ITP 2002), France, December, 2002.

[55] L. Pantel and L. C. Wolf. *On The Suitability Of Dead Reckoning Schemes For Games.* Proceedings of the 1st workshop on Network and system support for games, NetGames 2002, p. 79 - 84.

[56] S. Pingali, D. Towsley and J. F. Kurose. *A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols.* Proceedings 1994 ACM SIGMETRICS Conf., May 1994.

[57] J. M. Pullen, M. Myjak, and C. Bouwens. *Limitations of Internet protocol suite for distributed simulation in the large multicast environment.* RFC 2502, February 1999.

[58] J. M. Pullen and D. C. Wood. *Networking technology and DIS.* Proceedings of the IEEE, 83(8):1156-1167, Aug. 1995.

[59] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation.* ACM Press Books, SIGGRAPH Series, Addison Wesley, 1999.

[60] J. Smed, T. Kaukoranta and H. Hakonen. *Aspects of Networking in Multiplayer Computer Games.* Proceedings of International Conference on Application and Development of Computer Games in the 21st Century, 2001.

[61] J. Smed, T. Kaukoranta and H. Hakonen. *A Review on Networking and Multiplayer Computer Games.* Turku Centre for Computer Science, April 2002.

[62] J. Snyers d'Attenhoven. *Etude des systèmes de jeux en ligne - Gestion des mondes distribués.* Master Thesis, Université Libre de Bruxelles, 2001-2002.

[63] A. Steed and E. Frécon. *Building and Supporting a Large-Scale Collaborative Virtual Environment* Department of Computer Science, University College Londen, United Kingdom; Swedish Institute of Computer Science.

[64] A. Steed, J. Mortensen, and E. Frécon. *Spelunking: Experiences using the DIVE System on CAVE-like Platforms* Proceedings of the Joint Fifth Immersive Projection Technology Workshop & Seventh Eurographics Workshop on Virtual Environments, Stuttgart, May 2001.

[65] R. Stuart. *The Design of Virtual Environments.* McGraw-Hill, 1996. The Eurographics Association 2003. 114.

[66] A. S. Tanenbaum. *Computer Networks.* 3rd Edition, Prentice Hall, 1996.

[67] TASC. *Reliable Multicast Framework and Reliable Adaptive Multicast Protocol (RMF/RAMP)*. Software Release Version 2.0 (March 2001) http://www.tascnets.com/newtascnets/Software/RMF/

[68] TASC. *Reliable Multicast Protocols*. A comparison by Multicast Implementation STudy (MIST). http://www.tascnets.com/mist/doc/mcpCompare.html

[69] S. Taylor and B. Corrie. *vGrid: An Infrastructure for Distributed VE*. SimTecT 2000, January 2000.

[70] D. Verna, Y. Fabre, and G. Pitel. *Urbi et Orbi: Unusual Design and Implementation Choices for Distributed Virtual Environments*. EPITA/LRDE, Proceedings of the 6th International Conference on Virtual Systems and MultiMedia, August 4, 2000.

[71] R. Waters, D. Anderson, J. Barrus, D. Brogan, M. Casey, S. McKeown, T. Nitta, I. Sterns and W. Yerazunis. *Diamond Park and SPLINE: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability*. tech. report 96-02, MERL, Cambridge, Mass., Jan. 1996 and PRESENCE, 6 (4), 461-481, 1997, MIT.

[72] B. Whetten and J. Conlan. *A Rate Based Congestion Control Scheme for Reliable Multicast*. Technical White Paper, GlobalCast Communications, November 24, 1998.

[73] M. Wijnants. *Avatar Animation in Networked Virtual Environments*. Master Thesis, June 2, 2003.