Ontwikkeling ant-based computer netwerk routing protocol

Afstudeerverslag

Technische Informatica

Een afstudeerproject van Maarten van Eeuwijk
'20066118'

Haagse Hogeschool
Proxy Services B.V.

Versie v0.9
29-05-2012

Referaat

Maarten van Eeuwijk, "Ontwikkeling ant-based computer netwerk routing protocol", Afstudeerverslag Technische Informatica, Haagse Hogeschool, 2012.

Samenvatting

Mieren hebben een zeer interessante manier ontwikkeld om een routenet bij te houden. Zij doen dit door geursporen achter te laten in hun omgeving terwijl ze zich bewegen. Een mierenkolonie ontwikkelt op deze manier een routenet in de omgeving van het nest. Dit gedrag kan gemodelleerd worden in een algoritme en losgelaten worden op tal van problemen. Waaronder het vinden van routes in een netwerk.

Een voorgaand onderzoek heeft een ontwerpbeschrijving opgeleverd waarmee een mogelijke implementatie en toepassing word geschetst. Dit afstudeerproject is er op gericht deze te verwezenlijken en aan een test te onderwerpen.

Descriptoren (tags)

- Afstudeerverslag
- Ant-based algoritmen
- Routing protocol

Voorwoord

Dit document is als afstudeerverslag, of scriptie zoals het ook wel bekend staat, het sluitstuk op mijn afstudeerproject en zo doende ook op mijn opleiding. Het is het laatste document wat ik oplever, aan het eind van de weg naar een HBO getuigschrift. Het is tevens ook een mijlpaal in een project wat voor mij heel leerzaam is geweest. Een project wat feitelijk eerder is begonnen dan het afstuderen en wellicht daarna nog lang door zal lopen.

In een lesblok wat als designatie T7 heeft gekregen heb ik samen met Vincent Olsthoorn een literatuuronderzoek gedaan naar de bruikbaarheid van ant-based algoritmen in IP routing. De vorm waarin we antwoord hebben gegeven op die vraag is een ontwerpbeschrijving. Dat is een uitgebreide manier om 'ja' als antwoord te geven. Maar een bewijs was niet geleverd: Er was op dat moment nog geen implementatie die buiten het laboratorium zou functioneren. Ook een publiek protocol ontbrak.

Ik vond het T7 onderzoek danig interessant dat ik er graag verder mee wilde. Ik heb om die reden toen mijn zinnen gezet op van de logische continuatie van T7 mijn afstudeerproject maken: Het ontwikkelen van een ant-based routing protocol en bijbehorende implementatie. Ik heb met mijn verhaal Proxy Services B.V. enthousiast gekregen voor het onderwerp, waarop ze mij een afstudeerplek geboden hebben. Met de hulp van Hans de Vreught is er het juiste schoolkader aan gegeven.

Als voortzetting van het onderzoek uit T7 is het een project wat eigenlijk al begonnen was voor aanvang van het afstuderen. Dit maakt het voor mij specialer dan gewoon maar een opdracht. Door de uren die ik er al in gestoken heb is het ook iets persoonlijks geworden.

En er is "één voor allen, allen voor één". Wat een manier is om open source te omschrijven. Ik gebruik al een tijd met veel plezier tal van stukken open source software. Als ik met mijn protocol en implementatie een open source project kan starten doe ik daarmee mijn eerste bijdrage aan het spectrum van software waar ik zoveel aan te danken heb.

Geen van dit alles had mogelijk geweest zonder de Haagse Hogeschool en de mensen verbonden hieraan. Personen direct betrokken bij mij afstuderen waren Hans de Vreught als examinator, Cobie van der Hoek als afstudeerbegeleidster en Madelon Nieuwland als studieloopbaanbegeleidster. Andere docenten die significante invloed hebben gehad op mijn ontwikkeling waren Rutger Spaans, Nico Huiberts, Ron van Neijhof, Tony Andrioli en Charles Doest. Samen met de mensen genoemd als direct betrokkenen hebben deze docenten een grote invloed gehad op hoe ik mij persoonlijk en als technisch informaticus heb ontwikkeld.

Een andere partij die het zeker verdient genoemd te worden is het Proxy Services B.V. van Erik de Rijk en Marco van der Kolk. Zij hebben mij in staat gesteld mijn opdracht uit te voeren door het bieden van middelen, hulp en de vrijheid het project te doen zoals ik graag wilde. Het vermelden waard is dat ik mijn werk mag uitbrengen onder de General Public License. Niet veel bedrijven staan dat toe en dit is dan ook een lovenswaardig gegeven. Ook hebben beide heren waardevolle hulp geboden bij het oplossen van programma-technische problemen, iets waar geen tekort aan was gedurende het project.

Dit verslag brengt mij (zo hoop ik althans) aan het einde van mijn opleiding technische informatica. Ik zou hierbij alle docenten waarvan ik les heb gehad en in het speciaal de hierboven genoemden willen bedanken, tezamen met de mensen bij Proxy Services B.V., die mij een uitstekende afstudeerplek hebben geboden.

Was getekend: Maarten van Eeuwijk

31-05-2012, Rotterdam.

Inhoudsopgave

1. Inleiding

Ant-based algoritmen zijn een interessant samenkomen van biologie en informatie technologie. Het zijn routines die gemodelleerd zijn naar mierengedrag.

Wat is er zo interessant aan mierengedrag om er algoritmen naar te modelleren?

Mieren hebben zeer interessante manieren ontwikkeld om bepaalde vraagstukken op te lossen. Een mierenkolonie is continu bezig met voedsel binnenhalen naar het nest. Terwijl ze dit doen komen ze voor het volgende vraagstuk te staan: Hoe kom ik, op een snelle manier, van voedselbron naar nest?

In tegenstelling tot onze manieren om op deze vraag een antwoord te vinden, voeren mieren hier geen rationele benadering op uit. Er is zelfs geen centrale manager mier die alles in goede banen leidt.

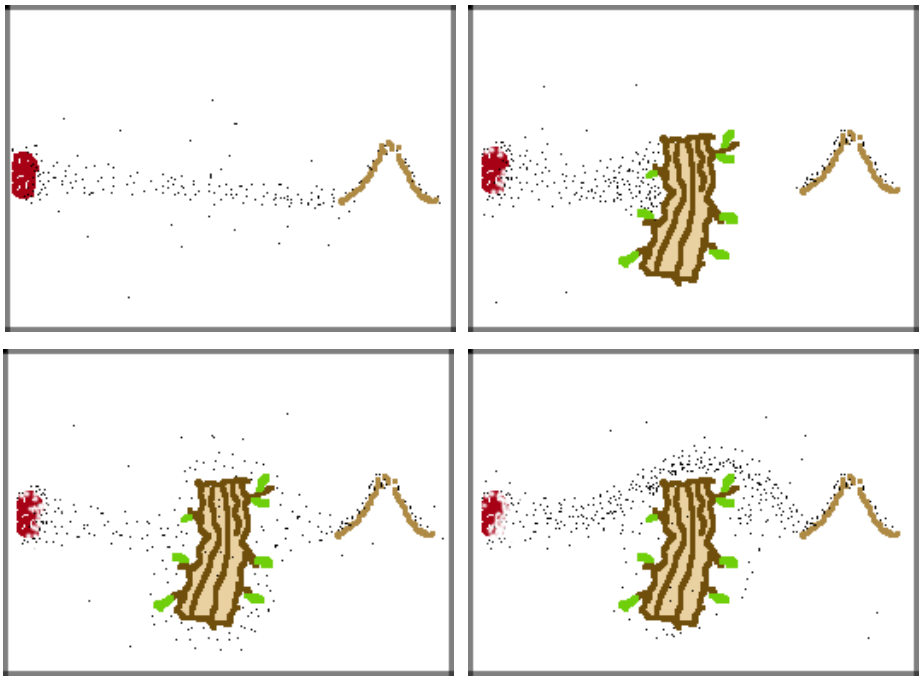
De kern in de aanpak van mieren is stigmergie[1]. Stigmergie is een vorm van indirecte communicatie. Wanneer mieren rondlopen in hun omgeving laten ze daar signalen in achter. Deze signalen kunnen door andere mieren in de directe omgeving opgepikt worden.

De signalen waar mieren gebruik van maken zijn geuren. Mieren laten terwijl ze zich rond bewegen in hun omgeving geuren achter. Ze gebruiken daarvoor honderden geuren die allemaal hun eigen betekenis hebben[2].

Naar gelang welk geurspoor mieren achterlaten heeft het een bepaalde aantrekkingskracht op andere mieren. Stel dat een mier op verkenningstocht is: Het geurspoor wat de mier in deze "modus" achterlaat is niet heel aantrekkelijk, er zullen daarom misschien maar een paar mieren in het kielzog volgen.

Wanneer een mier een voedselbron treft verandert dat. De mier zal beginnen voedsel terug te transporteren naar het nest (met dank aan het eerder gelegde geurspoor). Maar in deze "modus" legt de mier een heel andere geur neer: Een geur die juist heel aantrekkelijk is voor andere mieren. Dat wil zeggen dat andere mieren dit pad nauwelijks kunnen weerstaan en naar de voedselbron geleid worden. En ook die mieren zullen op hun weg terug feromoon (geurstof) op het pad achterlaten, waardoor die route nog aantrekkelijker wordt.

Stel dat er op de route een stuk obstructie terecht komt: Bijvoorbeeld een stuk boomstam waar moeilijk overheen te klimmen is. Deze boomstam ligt zo dat er een lange weg en een korte weg omheen is.



De mieren zullen na geconfronteerd te zijn met de versperring door middel van trial-and-error op zoek gaan naar een alternatieve weg en initieel zowel linksom als rechtsom gaan.

Omdat er per tijdseenheid meer mieren over de kortere weg lopen zal de kortere weg een sterkere

1 Guy Theraulaz, Eric Bonabeau: “A Brief History of Stigmergy”, MIT Artificial Life 5: 97–116 (1999)
2 Pierre-Paul Grassé: “La Reconstruction du nid et les Coordinations Inter-Individuelles chez Bellicositermes Natalensis et Cubitermes sp. La théorie de la Stigmergie: Essai d’interpretation du Comportement des Termites Constructeurs.” Insectes Sociaux 6: 41-81 (1959)

geur en dus hogere aantrekkingskracht krijgen. Zowel het langere als kortere pad zullen in gebruik blijven, maar het kortere (snellere, efficiëntere) pad zal dominant worden.

Twee concurrerende routes kunnen ook anders ontstaan. Mieren volgen een pad namelijk nooit precies. Al hun acties zijn eigenlijk wat heet pseudo-willekeurig. Ze zijn willekeurig in wat ze doen, maar bevooroordeeld door de signalen achtergelaten in de omgeving. Het kan dus prima zijn dat een mier toevallig een andere route dan de dominante loopt. Er zullen wellicht een paar mieren volgen, die eveneens feromoon achter laten.

Als deze route toevallig sneller is, zullen er per tijdseenheid meer mieren overheen lopen, waardoor de geur sterker wordt. Er kan dus gemakkelijk een tweede route ontstaan, die een deel van het verkeer overneemt. Wellicht zelfs dat het de dominante route wordt, mocht het een snellere route zijn. Het routenet wat mieren maken is dus nooit af. Het is altijd aan verandering onderhevig.

De willekeurigheid van mieren is niet het enige wat het routenet constant doet veranderen. Een andere factor daarin is dat de omgeving ook altijd aan verandering onderhevig is. Niet alleen de routes zelf kunnen verstoord worden, ook de redenen dat ze er zijn veranderen continu. Een voedselbron raakt namelijk een keer op. Wanneer dat het geval is zullen steeds meer mieren kiezen weer op zoek te gaan naar een andere bron, in plaats van voedsel terug te transporteren. Hierdoor zal de route steeds minder feromoon (geurstof) ontvangen.

Wanneer een route minder feromoon ontvangt zal deze minder aantrekkelijk worden. Achtergelaten feromoon verdampt namelijk langzaam. Routes blijven dus bestaan zolang er mieren overheen lopen. Vermindert of stopt dat, dan vervaagt de route langzaam, uiteindelijk tot deze verdwenen is.

Waar dit mechanisme als geheel in resulteert is dat een routenet zich continu tracht aan te passen aan de dan geldende situatie. Nieuwe routes worden gevormd, goede routes krijgen meer aandacht en irrelevant geworden routes verdwijnen.

Dit fenomeen staat bekend als emergentie[3]: Iets lijkt uit het niets te ontstaan. Het opzetten en bijhouden van een routenet is een complexe taak. Maar de complexiteit van het systeem als geheel is niet terug te zien in de individuen. Mieren zijn op zichzelf relatief dom. Willekeurig maar bevooroordeeld door geurtjes rondlopen doet niet superintelligent aan. Maar een kolonie van deze simpele beestjes doet als geheel iets heel complex. Zonder dat er controle van bovenaf is. Zonder dat de complexiteit in elk individu terug te vinden is.

Dit is een zeer interessant fenomeen. Tot nu toe bouwen wij namelijk alleen nog maar technologie naar een reductioneel model. Maar zoals op tal van plaatsen in de natuur te zien is hoeft dat helemaal niet. Een slim geheel kan prima uit simpele stukjes bestaan. Door simpele stukjes te "tunen" kan iets verkregen worden wat als geheel een complexe taak kan uitvoeren.

Wat kan dit dan betekenen voor de IT (Informatie Technologie)?

De impact van een verschuiving naar dit paradigma is potentieel heel groot. Het is veel makkelijker om een klein onderdeeltje, programma of circuit te ontwerpen dan een grote. Bij een klein stuk kan de ontwerper zich op een kleiner gebied concentreren, waardoor de kans kleiner wordt dat fouten onopgemerkt blijven.

Hoe vertaalt dit zich naar TI (Technische Informatica)?

TI is een vakgebied/opleiding wat voor een groot deel bestaat uit netwerktechnologie. Een netwerk is een systeem waarover gegevens getransporteerd worden. Voor dat mogelijk is moet er op een manier een route bekend zijn. In de begindagen van computernetwerken was het definiëren van routes handwerk, maar al snel werden netwerken daarvoor te groot. Menselijke fouten slopen in de

3 http://en.wikipedia.org/wiki/Emergence

tabellen met regels en het werk om ze bij te houden liep de spuigaten uit. De eerste automatische mechanismen voor het bijhouden van routing tabellen werden toen uitgevonden.

Deze mechanismen zijn in de loop der tijd geëvolueerd. Nu kennen we ze onder de namen EIGRP, IS-IS, OSPF, BGP en vele andere[4]. Wat deze protocollen allemaal gemeen hebben is dat het paradigma waar ze naar gemodelleerd zijn reductionisme is. Het reductionele model heeft echter een aantal nadelen.

De programma's op de routers hebben de complexiteit van het gehele systeem in zich. Elk van deze routers heeft een beeld van het gehele systeem. Bij elke verandering hieraan, groot of klein, moet elke router z'n gehele datastructuur opnieuw doorrekenen om z'n routing tabel te actualiseren. Dit is uiteraard geen probleem wanneer de topologie klein blijft; het aantal veranderingen per tijdseenheid blijft dan relatief laag. Wordt het netwerk groter dan neemt het aantal veranderingen ook relatief toe. Hierdoor wordt de last op de hardware hoger.

De toename in de last is alleen niet lineair. Het aantal takken gaat namelijk niet lineair mee met de hoeveelheid knopen in het netwerk, gezien netwerken veelal worden aangelegd met twee of meer takken per knoop omwille van redundantie. Hierdoor neemt het aantal mogelijke paden over het netwerk vaak, afhankelijk van de topologie, exponentieel toe.

Nu is meer hardware inzetten tot nu toe afdoende geweest. Echter komt er ooit een keer een moment waarop technieken als route summarization en nog snellere/grotere chips niet meer afdoende zijn. Er is een plafond. Hoe ver we er nog vanaf zitten is moeilijk te voorspellen, maar het plafond wordt een keer geraakt[5].

Van mieren hebben we echter gezien dat het ook anders kan: Om een intelligent systeem te bouwen hoeft die intelligentie niet perse terug te vinden te zijn in de onderdelen waar het systeem uit bestaat.

Hoe kunnen we emergente effecten benutten in een computernetwerk?

Het gedrag wat mierenkolonies vertonen is prima te modelleren in algoritmen[6]. Wanneer mieren zich bewegen zijn ze continu bezig met het vinden van paden, selecteren van paden en afwegen van paden. Deze functies zijn eveneens nodig in een computer netwerk.

Rond ant-based algoritmen zijn dus mogelijk routing protocollen te ontwikkelen. Naar de vruchtbaarheid voor zo'n poging is al eens een onderzoek gedaan[7]. Dit onderzoek heeft laten zien dat er prima mogelijkheden zijn om een protocol te ontwikkelen gebaseerd op ant-based algoritmen, maar dat geen enkele tot nu toe de laboratorium omgeving uit is gekomen.

Wat is de inhoud van dit afstudeerproject?

In het genoemde onderzoek is een aanzet gegeven tot een op ant-based algoritmen gebaseerd protocol. In dit project wordt deze aanzet opgepakt en verder ontwikkeld. Het project is er op gericht een protocol en implementatie te ontwikkelen wat de gedragingen van mieren imiteert en gebruikt voor het vinden en afwegen van routes door het netwerk. Deze implementatie kan dan aan een test worden onderworpen.

4 Cisco Netacad: CCNA, semester 2, v3.1, http://cisco.netacad.net

5 Bezalel (Ben) Gavish: “Hard Limits on the Growth of the Internet and Computing Capacity”, International Journal of Information Science and Management, special issue (2010)

6 Alberto Colorni, Marco Dorigo, Vittorio Maniezzo: “An investigation of some properties of an Ant algorithm”, Proceedings of the Parallel Problem Solving from Nature conference (PPSN '92), Brussels, Belgium, Elsevier Publishing, 509–520 (1992)

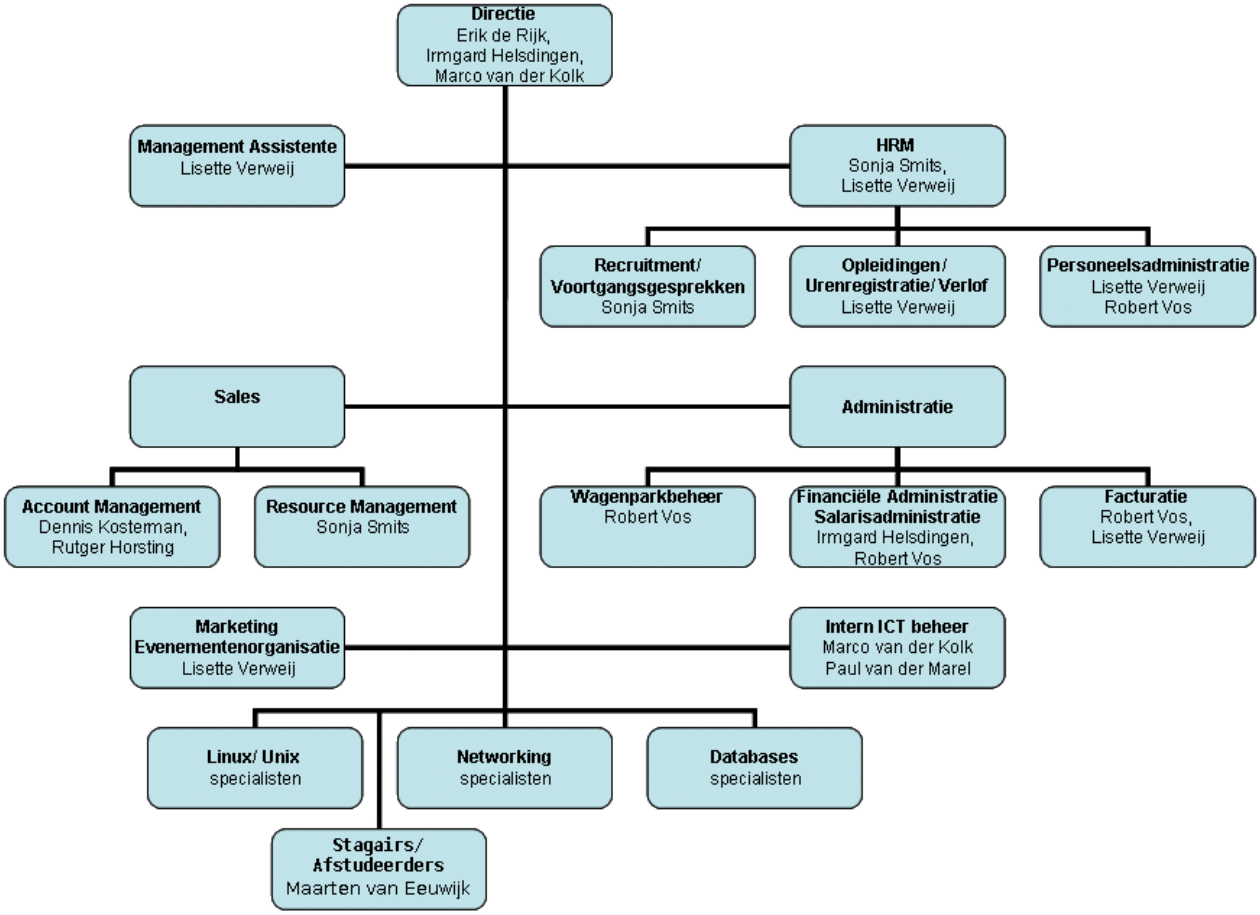
7 Maarten van Eeuwijk, Vincent Olsthoorn: Ant-based algoritmen in IP routing (2011), *Bijlage A*

2. Organisatiebeschrijving

Proxy Services B.V.: De naam van het bedrijf komt van het Latijnse woord voor gevolmachtigde. Het symboliseert Proxy's rol als dienstverlener in de ICT, gezien een dienstverlener taken uit handen neemt of doet uit naam van. Proxy is namelijk een detacheerder. Proxy heeft mensen gestationeerd bij ING, Defensie, Havenbedrijf Rotterdam, Dutch Aerospace, Fokker-Stork, Justitie, Tele2, Rabobank, ANWB, PostNL, AT&T, Shell, Rijkswaterstaat en vele andere bedrijven en instellingen. Als detacheerder richt Proxy zich hoofdzakelijk op ervaren mensen in dienst hebben. Het is een klein bedrijf, totaal 42* werknemers, waarvan 8 administratief / management.

Proxy is opgestart in 1997 door Marco van der Kolk, Erik de Rijk en de toenmalige man van Irmgard Helsdingen. Het bedrijf groeide snel naar 40 werknemers. In 2002 echter volgde een nieuwe start wegens splitsen van het bedrijf. Het deel dat Proxy werd bestaat nog.

Vandaag de dag ziet de organisatie van Proxy er als volgend uit:



Ik bevind mij geheel onderaan het organogram, tussen de Linux/Unix en networking specialisten. De werknemers die op een dag aanwezig waren kon ik ten alle tijden aanspreken. Proxy kent geen gesloten deuren of toegangspasjes voor afdelingen. Iedereen is bereikbaar. De enige afgesloten deur is die van de server room, omwille van dataveiligheid. En de voordeur, uiteraard.

* Geen grap; geen referentie naar 'The Hitchhiker's Guide to the Galaxy'.

3. Vormgeven van het project

Het project startte met het opstellen van een plan van aanpak. Deze is opgesteld naar de opdrachtomschrijving die al in het afstudeerplan[8] was vastgelegd.

3.1. Opdrachtomschrijving

De opdracht die ik ga uitvoeren in mijn project is een voortzetting van een onderzoek wat ik eerder heb uitgevoerd. In een lesblok gedurende mijn studie (binnen de Haagse Hogeschool aangeduid als T7) heb ik onderzoek gedaan naar wat bekend staat als ant-based algoritmen[9].

Aanleiding

Er is momenteel al enig onderzoek gedaan naar de toepasbaarheid van ant-based algoritmen voor het oplossen van complexe problemen. Met de term complex wordt hier gedoeld op de wiskundige definitie van complexiteit; een probleem waarbij de hoeveelheid werk tot de oplossing snel (bijvoorbeeld exponentieel) toeneemt bij het groter maken van het probleem[10].

De traditionele manier van problemen oplossen, alle mogelijkheden doorrekenen, geeft de garantie tot de meest optimale oplossing. Deze aanpak is echter duur. Ant-based algoritmen geven niet perse de meest optimale oplossing, maar een oplossing die waarschijnlijk goed genoeg is: Ant-based algoritmen zijn heuristieken. Hier staat tegenover dat ze met veel minder middelen (tijd, stroom etc.) tot deze mogelijk sub-optimale oplossing komen.

Eén van de gebieden waar men met complexe problemen te maken heeft is netwerken. Een gebied waar ant-based algoritmen mogelijk een toepassing gaan vinden.

Probleemstelling

Er is nog geen (publiek of gestandaardiseerd) netwerk routing protocol wat ontworpen is rond ant-based algoritmen. Hierdoor is de veelbelovendheid van deze algoritmen op netwerkgebied nog niet op de proef gesteld.

Doelstelling

Het doel van de opdracht is het verkrijgen van inzicht in de mogelijkheden van ant-based netwerk routing, door middel van een proof-of-concept* waaruit blijkt of ant-based algoritmen werken op netwerk gebied.

(*De proof is een programma, het concept is het protocol)

Resultaat

Het resultaat van de opdracht is een ant-based routing protocol getest met de prototype implementatie.

3.2. Van afstudeerplan naar plan van aanpak

Het afstudeerplan heeft voor aanvang van het project de opdrachtomschrijving, de uit te voeren werkzaamheden en de producten die opgeleverd moeten worden al vastgelegd.

De uit te voeren werkzaamheden en op te leveren producten zijn beide naar een fasering opgesteld. Elke fase bestaat uit één of meerdere uit te voeren werkzaamheden en elke fase levert een product op.

3.2.1 Methodiek

De door mij opgestelde opdracht betreft experimenteel werk. Er is van te voren niet precies bekend hoe het verloop van het project zal zijn. Deze opdracht vraagt daarom veel flexibiliteit.

Het gebruiken van een vaste of gestandaardiseerde methode kan daarom het succes van deze opdracht in de weg staan. Om deze reden heb ik gekozen geen gestandaardiseerde methode als RUP, Prince2, ITIL of de waterval methode te gebruiken.

---

8 Project documentatie: Afstudeerplan, *Bijlage B*

9 Maarten van Eeuwijk, Vincent Olsthoorn: Ant-based algoritmen in IP routing (2011), *Bijlage A*

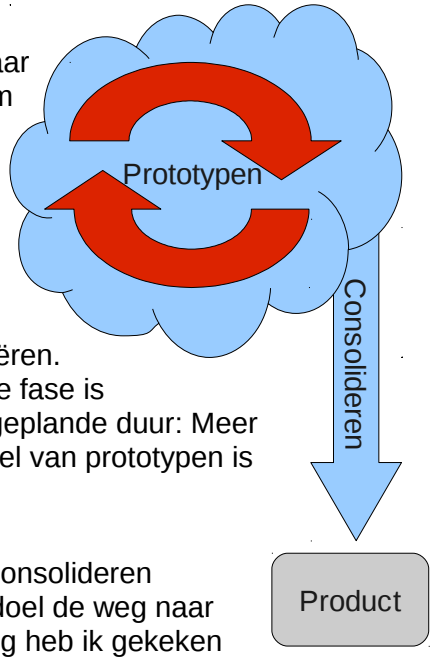
10 https://nl.wikipedia.org/wiki/Computationele_complexiteitstheorie

Dit betekent zeker niet dat de aanpak van dit project niet methodisch is. Er is wel degelijk een structuur, maar in plaats van middels een vooropgezet raamwerk heb ik die zelf vormgegeven. Dit liet mij vrij om fasen en taken in te delen geheel naar eigen inzicht. Dit eigen inzicht is echter wel voor een groot deel geïnspireerd op de principes van software engineering[11].

Structuur

Het inzicht waar de indeling naar gemaakt is gaat uit van flexibiliteit. Het project betreft een ontdekkingsreis. Dat impliceert impliciet dat het project veel onbekende factoren heeft en dat er een hoop verrassingen op handen zijn.

Om zo'n project te kunnen accommoderen moet er een structuur bedacht worden waarin het experimenteren uitgevoerd kan worden. Maar het einddoel is uiteraard dat er een resultaat moet zijn. Het project is om die reden uit feitelijk twee delen komen te bestaan: Experimenteren, gevolgd door alles wat geleerd is consolideren in een product. Dit leek voor aanvang de beste hybride om vanuit een experimentele aanpak naar een product te werken.



Analyseren & prototypen

Voor het experimenteren gedeelte is gekozen een enkele fase te definiëren. Deze fase heeft een vrij karakter, met als doel dingen te proberen. Deze fase is 'analyseren & prototypen' gedoopt. Het betreft de fase met de langste geplande duur: Meer dan de helft van de projecttijd wordt door deze fase ingenomen. Het doel van prototypen is uitvinden wat de mogelijkheden zijn.

Na het eindigen van deze fase begint ook gelijk het consolideren. Het consolideren gedeelte is wel opgesplitst in meerdere fasen. Deze fasen hebben als doel de weg naar een product te begeleiden. Voor inspiratie op het gebied van die indeling heb ik gekeken naar RUP en SDM. De waterval van SDM is daar als voorbeeld genomen. Wat naamgeving betreft heb ik naar RUP gekeken. Het consolidatieproces is op die manier ingedeeld in de fasen 'requirements bepalen', 'software architectuur ontwikkelen' en 'implementeren & testen'.

Requirements bepalen

De requirements zijn de eisen waaraan het programma moet voldoen. Vertaald naar dit project houdt dat in het opstellen van het protocol wat het uiteindelijke product moet implementeren. Dit protocol is in de prototyping fase al bedacht maar nog niet formeel beschreven. Het in documentatie vastleggen van wat er uitgedokterd is tijdens het prototypen is een prima voorbeeld van het 'consolideren' waar eerder over gesproken is.

Software architectuur ontwikkelen

Het doel van de architectuur ontwikkelen is aan de hand van wat er geleerd is bij het prototypen de koers uitzetten naar de uiteindelijke implementatie. Dit kan betekenen dat er terug gekomen wordt op (ontwerp)keuzes die eerder gemaakt zijn tijdens het prototypen. Het resultaat van deze fase is een ontwerp voor de "proof" van het "concept". Ofwel een ontwerp voor een programma wat het protocol implementeert.

Implementeren & testen

Het doel van de implementeren & testen fase is het programma schrijven en testen. Afhankelijk van de koers die is uitgezet kan een bepaalde hoeveelheid werk uit de prototyping fase worden overgenomen. Deze implementatie kan vervolgens aan een test worden onderworpen. Wat inhoudt dat het aan de tand gevoeld wordt op het vermogen om routes te vinden. De feitelijke "proof" van het concept wordt hier mogelijk geleverd.

11 Ian Sommerville: “Software Engineering”, Addison Wesley/Pearson Education (1982-2010)

Gelijkenis

Deze aanpak als geheel lijkt sterk op een iteratief ontwikkelproces gevolgd door een afgeslankte enkele run SDM. Dit maakt dat het project zowel iteratieve als lineaire (waterval-achtige) elementen heeft. Op de juiste wijze uitgevoerd zou deze opgestelde aanpak het project moeten dienen tijdens de uitvoer.

Het plan van aanpak is zo doende mijn methode behorende bij de uitvoer van dit project.

3.2.2 Uitzetten van het ontwikkeltraject

Bij de start van het project zijn de opdrachtomschrijving, de uit te voeren werkzaamheden en de op te leveren producten vanuit het afstudeerplan overgenomen naar het plan van aanpak. In het plan van aanpak zijn ze verder uitgewerkt tot een ontwikkeltraject.

Om het te belopen ontwikkeltraject af te maken was nog een drietal andere zaken nodig: Een tijdsindeling, voorwaarden en risico's.

Tijdsindeling

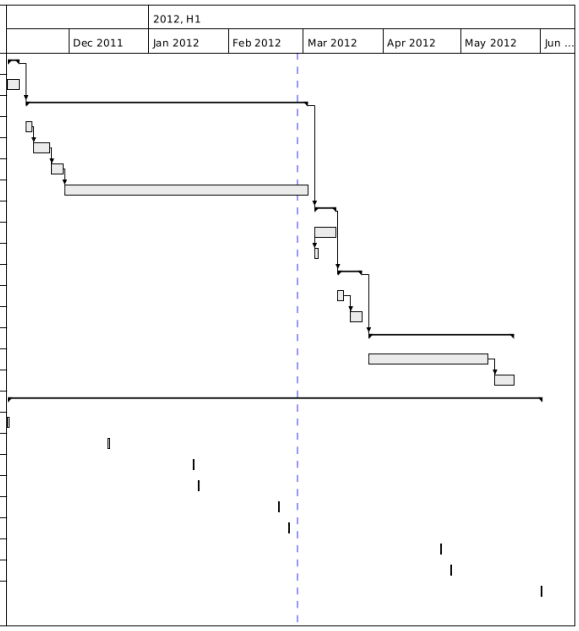
De belangrijkste hiervan was de tijdsindeling, ofwel een planning. De op te leveren producten en uit te voeren werkzaamheden waren al ingedeeld naar een fasering, wat er restte was hier een reële tijdsindeling aan geven. Hier heb ik geen goede objectieve methode bij kunnen vinden of bedenken. De enige optie is daarom om mijn beste inzicht er op los te laten.

Middels een planningtool heb ik alle activiteiten verwerkt tot een hiërarchische indeling. Deze indeling werkt aan de hand van benodigde tijd per taak en sets van regels over welke taak wanneer kan beginnen. Door deze zaken in te vullen berekent de tool de indeling van taken vanaf de ingevulde startdatum. Het resultaat van deze berekening is een tabel, in de vorm van de dag waarop een taak start, eindigt en de duur van deze taak.

Naast deze tabel kan de tool ook nog een grafische weergave genereren in de vorm van een Gantt-diagram. Hierin is de tijdsindeling te zien in de vorm van balken die de tijdsspanne weergeven over de dagen en weken die het project duurt. Zo is in een enkele oogopslag duidelijk hoe de verhoudingen liggen tussen de geplande werkzaamheden.

In de volgende illustratie een impressie van de tabel en het Gantt-diagram:

Name	Work
Plan maken	**5d**
Een plan van aanpak opstellen	5d
Analyseren & prototypen	**80d**
Het onderzoeksrapport uit het voorlopende onderzoek (HHS lesblok T7) doornemen en verbeteren	3d
Analyseren van reeds bestaande netwerk routing frameworks op bruikbaarheid als basis / host voor ontwikkeling	5d
Onderzoeken wat de mogelijkheden zijn bij Proxy voor het simuleren van een netwerk om in te testen en ontwikkelen	3d
Door middel van prototypen mogelijkheden uitvinden voor een ant-based routing protocol	69d
Requirements bepalen	**9d**
Een ant-based routing protocol opstellen	7d
Bepalen wat het type, grootte en soort testnetwerk wordt waarin de proof-of-concept (van het ant-based routing protocol) getest gaat worden	2d
Software architectuur ontwikkelen	**8d**
Een gefundeerde keuze maken uit de eerder geanalyseerde routing frameworks	3d
De architectuur van de proof-of-concept ontwerpen	5d
Implementeren & testen	**41d**
Het programma schrijven	35d
Het programma testen op het vermogen om routes te vinden	6d
Belangrijke data	**9d**
Start	1d
Bezoek begeleidend examinator	1d
Voorlopig verslag inleveren	1d
Voorlopig verslag bespreken	1d
Inleveren documenten tussentijds assesment	1d
Tussentijds assesment	1d
Inleveren documenten tussentijds assesment (verlenging)	1d
Tussentijds assesment (verlenging)	1d
Afstudeerverslag + beoordeling bedrijfsmentor inleveren	1d



Zie het plan van aanpak[12] voor de volledige planning, inclusief Gantt-diagram op leesbaar formaat.

12 Project documentatie: Plan van aanpak, H3, *Bijlage D*

Toevoegingen

Het plan van aanpak is in een aantal aspecten ook wat uitgebreider dan het afstudeerplan. Zo heeft het plan van aanpak een uitgebreidere aanleiding gekregen bij de opdrachtomschrijving. Dit is gedaan om het document wat verder aan te kleden ten behoeve van de begrijpbaarheid. Hierdoor is het plan van aanpak beter voor te leggen aan collega's en leidinggevenden bij het gastbedrijf. De paragrafen 'Probleemstelling', 'Doelstelling', en 'Resultaat' zijn uiteraard ongewijzigd gebleven.

Tevens is er bij de werkzaamheden in het plan van aanpak een extra taak toegevoegd. Bij de fase 'Analyseren & prototypen' is de taak "Het onderzoeksrapport uit het voorlopende onderzoek (HHS lesblok T7) doornemen en verbeteren" toegevoegd. Het doel van deze taak is het T7 onderzoeksrapport van voor tot achter doornemen en verbeteren. Dit dient twee zaken: Al eerste een esthetisch doel. Omdat het T7 onderzoeksrapport wordt opgenomen in de bijlagen is het onderdeel van de presentatie van het onderzoek. Een goede presentatie doet een onderzoek goed. Als tweede een educatief doel. Door het onderzoeksrapport van voor tot achter opnieuw door te nemen was ik weer terug in de materie.

Risico's

De laatste toevoeging ten opzichte van het afstudeerplan was een risico analyse. De aard van een project heeft invloed op de risicobeheersing. De experimentele aard van dit project brengt aanzienlijke onzekerheid over welke weg precies afgelopen wordt en welke problemen onderweg tegengekomen worden. Daar zijn de risico's voor een groot deel naar gevormd. Het kunnen inspelen op onverwachte wendingen is dezelfde reden dat de methodiek is geworden zoals die is. Vier van de zeven risico's gaan over onverwachte wendingen tijdens de ontwikkeling. De overige drie risico's gaan over ongeluk.

Onverwachte wendingen:

- De licentie voor de simulatie omgeving blijkt ontoereikend.
- De beschikbare hardware blijkt ontoereikend.
- Een benodigd boek of ander stuk informatie is niet beschikbaar binnen het bedrijf.
- Onverwacht tijd tekort, uitloop.

Ongeluk:

- Uitval van de uitvoerder (een epilepsie patiënt).
- Falende hardware; dataverlies.
- Een kritieke faciliteit (koeling, stroom, bandbreedte) in de serverruimte van Proxy faalt of schiet tekort.

Voor elk van deze risico's is een benadering voor kansvermindering en een respons voor impactvermindering opgesteld. Deze zijn per risico te lezen in het plan van aanpak[13].

13 Project documentatie: Plan van aanpak, H7, *Bijlage D*

4. Analyseren en prototypen

Dit hoofdstuk beschrijft de activiteiten van de fase 'Analyseren & prototypen'. Dit was de eerste fase van de ontwikkeling van een ant-based routing protocol.

Activiteit 1:

De eerste stap van het plan is het analyseren en verbeteren van het T7 onderzoeksrapport[14]. Door een aantal dagen met het onderzoek bezig te zijn was ik weer helemaal terug in de materie. Tevens draagt het bij aan het uiterlijk van het project, gezien het onderzoeksrapport een belangrijke bijlage betreft.

Activiteit 2:

De tweede in het plan van aanpak gedefinieerde activiteit is reeds bestaande netwerk routing frameworks analyseren. Onderdeel van de aanpak is een bestaand stuk software gebruiken als basis om op te bouwen. Dit zou de hoeveelheid te implementeren zaken moeten doen afnemen, waardoor er meer tijd overblijft voor het werken aan project-unieke zaken als het protocol zelf.

Het analyseren van reeds bestaande netwerk routing frameworks start met het vinden van voor dit project relevante kandidaten. Op verscheidene methoden is er gezocht naar reeds bestaande projecten waarna hier een lijst van is samengesteld. Hierover is te lezen in §1 van dit hoofdstuk.

Nadat de relevante frameworks en daemons gevonden zijn moesten die bekeken en geanalyseerd worden, opdat er later een keus uit gemaakt kon worden. Hierover is te lezen in §2 van dit hoofdstuk.

De tweede activiteit wordt afgesloten met een afweging tussen de gevonden frameworks en een keuze hieruit. Het gekozen stuk software zal dan als basis dienen in ten minste de analyse & prototyping fase. Hierover is te lezen in §3 van dit hoofdstuk. De routing frameworks analyse is in het geheel te lezen in het analyse & prototyping rapport[15].

Activiteit 3:

De derde activiteit die voltooid moet worden om te kunnen experimenteren en prototypen is een simulatie omgeving selecteren. Hiervoor zijn verschillende methoden die afgewogen moeten worden tegen elkaar. Net als de routing daemons en frameworks moeten de opties eerst op een rij gezet worden, alvorens een oordeel mogelijk is. Hierover is te lezen in §4 van dit hoofdstuk. De volledige netwerksimulatie mogelijkheden uiteenzetting is te lezen in het analyse & prototyping rapport[16].

Activiteit 4:

Vanaf §5 is te lezen over het activiteit mogelijk gemaakt door de vorige drie activiteiten: Prototypen. In deze paragraaf wordt een beeld geschetst over het verloop van het prototypen en wat daar bij ontwikkeld is. Dit beeld wordt opgebouwd met behulp van snippets code, diagrammen en tekstuele beschrijvingen. Het volledige verloop van het prototypen is te lezen in het analyse & prototyping rapport[17].

14 Maarten van Eeuwijk, Vincent Olsthoorn: Ant-based algoritmen in IP routing (2011), *Bijlage A*
15 Project documentatie: Analyse & prototyping rapport, H3 en H4, *Bijlage E*
16 Project documentatie: Analyse & prototyping rapport, H5, *Bijlage E*
17 Project documentatie: Analyse & prototyping rapport, H6, H7 en H8, *Bijlage E*

4.1. Verzamelen beschikbare routing software

Er zijn verscheidene routing frameworks en daemons die als basis / host kunnen instaan. Deze stukken software moesten verzameld worden in een lijst en daarna stuk voor stuk beoordeeld worden met als doel tot de meest geschikte te komen. Voor het opstellen van deze lijst moesten als eerst alle relevante projecten gevonden worden. Gezien er in elk geval de mogelijkheid moest zijn een module te schrijven en de werking van het routing platform als geheel aan te passen was de relevantie beperkt tot open source projecten.

Het zoeken van open source routing frameworks en daemons is op drie manieren gedaan:

1. Als eerste is Wikipedia bezocht. Wikipedia bleek een lijst met open source routing software te hebben: http://en.wikipedia.org/wiki/List_of_open_source_routing_platforms
2. De lijst van Wikipedia geeft een goed beeld. Echter zelfs zo'n grote gebruikersgroep als die van Wikipedia kon een project gemist hebben. Om die reden is ook Google geraadpleegd. Hier is echter niets nieuws uit gekomen.
3. Na Wikipedia en Google zijn ook nog de Debian repositories doorzocht op routing software. Debian is een grote, veelgebruikte Linux distributie, wat suggereert dat alle projecten die er toe doen daarin opgenomen zouden moeten zijn. Dit gaf een enkel nieuw resultaat: babeld.

De lijst van gevonden routing frameworks en daemons is als volgend:

- Babel
- B.A.T.M.A.N.
- BIRD
- OpenBGPD & OpenOSPFD
- GNU Zebra
- Quagga
- XORP
- Click router

Nadat er een lijst was opgesteld van bruikbare routing platforms moest hieruit een keuze gemaakt worden. Gedurende in elk geval het prototypen zal dan alle aandacht op werken met dit project gefocust blijven om uit tenminste die optie het maximale te halen. Elk project is bekeken op voordelen en nadelen waarna een afweging is gemaakt.

4.2. Analyse beschikbare routing software

Het analyseren van de beschikbare routing software projecten is in de volledigheid te lezen in het analyse & prototyping rapport[18]. Nu volgend een samenvatting van de analyses:

Babel

- Web: http://www.pps.jussieu.fr/~jch/software/babel/ & http://en.wikipedia.org/wiki/Babel_(protocol)
- Gevonden via: Debian repositories

Babel is een distance vector routing protocol geïnspireerd op onder andere EIGRP. Het programma is niet generiek van opzet.

B.A.T.M.A.N.

- Web: http://www.open-mesh.org/ & http://en.wikipedia.org/wiki/B.A.T.M.A.N.
- Gevonden via: Wikipedia

Batman creëert virtual interfaces en doet daarna frame forwarding. Het doet dus helemaal niets met IP routing; dat is iets wat eventueel over de virtual interfaces kan.

BIRD

- Web: http://bird.network.cz/ & http://en.wikipedia.org/wiki/Bird_Internet_routing_daemon
- Gevonden via: Wikipedia

BIRD ondersteunt meerdere routing protocollen: BGP, OSPF en RIP. Het geheel is geschreven in C en wordt na compilatie gelinkt naar 1 executable, met uitzondering van de CLI. Technische structuur doet goed aan. De documentatie lijkt goed en volledig.

OpenBGPD & OpenOSPFD

- Web: http://openbgpd.org/ & http://en.wikipedia.org/wiki/OpenBGPD
- Gevonden via: Wikipedia

OpenBGPD en OpenOSPFD zijn twee daemons uit het OpenBSD project. Ze zijn praktisch OpenBSD-only. Ze zijn niet generiek van opzet.

GNU Zebra

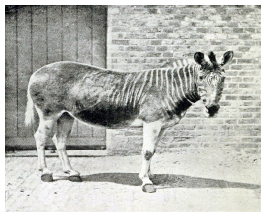
- Web: http://zebra.org/ & http://en.wikipedia.org/wiki/GNU_Zebra
- Gevonden via: Wikipedia

Zebra is een collectie deamons, geschreven in C. Deze communiceren middels een binary protocol via Unix sockets of TCP/IP verbindingen; een manier om functionaliteit te scheiden. Het hoofdproces bewerkt de kernel routing table. Zebra is echter sinds 2005 niet meer in ontwikkeling.

18 Project documentatie: Analyse & prototyping rapport, H3.1 t/m H3.8, *Bijlage E*

Quagga

- Web: http://quagga.net/ & http://en.wikipedia.org/wiki/Quagga_(software)
- Gevonden via: Wikipedia



Quagga is een fork van de laatste GNU Zebra release. Quagga heeft dezelfde opzet als GNU Zebra. Documentatie voor gebruikers is op orde. Documentatie voor ontwikkelaars is schaars. Ofwel de code en comments daarin zijn de documentatie. Als vangnet is er de mailinglist en IRC kanaal.

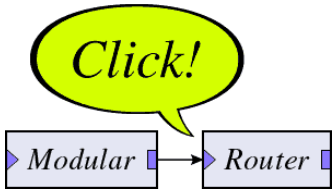
XORP

- Web: http://xorp.org/ & http://en.wikipedia.org/wiki/XORP
- Gevonden via: Wikipedia

XORP bestaat net als Zebra en Quagga uit losse binaries, albeit in C++. Deze communiceren via TCP/IP of Unix sockets. XORP's doel is om de kloof tussen netwerkonderzoek en netwerkbeheer te slechten. Opdeling van functionaliteit en veel documentatie (voor zowel gebruikers als ontwikkelaars) onderschrijven dit. De hoeveelheid comments in de source code valt tegen.

Click router

- Web: http://read.cs.ucla.edu/click/
- Gevonden via: XORP website



De Click router is eigenlijk geen routing daemon, maar een modulaire toolkit voor packet handling. Onderdelen of functionaliteiten benodigd bij het verwerken van IP packets kunnen als "blokjes" aan elkaar geschakeld worden. De Click router is dan ook vooral een alternatief voor de traditionele Unix kernel forwarding. Indien een eigen, geheel ant-based IP-stack maken in beeld komt kan die voor een groot deel met de Click router elementen gebouwd worden.

4.3. Selectie beschikbare routing software

Nadat deze analyse gemaakt is was de volgende stap een keuze te maken uit de genoemde projecten. Om deze keuze op een navolgbare manier te maken heb ik een aantal criteria opgesteld. De volledige motivatie is te lezen in het analyse & prototyping rapport[19]. Volgend de belangrijkste details.

Criteria

– De belangrijkste van de aandachtspunten was documentatie. Documentatie is zeer belangrijk bij het professioneel werken met software. Als het om een beetje aanklooien gaat maakt het niet heel veel uit als mysteries wat moeilijker zijn op te lossen. Echter wanneer er een tijdsbeperking voor het behalen van een doel in het spel is, is barrières zo snel en zo makkelijk mogelijk kunnen slechten zeer gewenst.

De hoeveelheid en vooral kwaliteit van de geboden documentatie is daarom een grote factor geweest in mijn keuze. Dit is iets waar in de open source wereld extra op gelet moet worden: Open source projecten worden vaak gedraaid door mensen die het voor de lol doen. Software schrijven en uitbreiden is nu eenmaal veel boeiender dan documentatie erbij schrijven. De kwaliteit van de software kan daarom heel hoog zijn, terwijl de bijbehorende documentatie enorm slecht is.

19 Project documentatie: Analyse & prototyping rapport, H4, *Bijlage E*

– Hoe bewerkbaar een stuk software is, is erg van belang. Waar ik naar zocht is iets wat zich makkelijk laat uitbreiden, aanpassen en aanvullen. Hoe het project is vormgegeven is hier allesbepalend in.

De regel is dat een programma zich makkelijker laat uitbreiden als functionaliteiten opgedeeld zijn. Talen, technieken en methodieken die encapsulation & information hiding aanmoedigen zijn om die reden uitgevonden[20]. Hoe meer binnen het programma de functionaliteiten gescheiden zijn, hoe beter. Abstractie is doorgaans nadelig voor de performance, maar voordelig voor de werkbaarheid van een stuk software.

– Als laatste is de taal waarin de software is geschreven een punt van aandacht. Elke taal heeft zo z'n mogelijkheden, beperkingen en eigenschappen. De gebruikte taal dicteert voor een aanzienlijk deel hoe een project opgezet is. Een voor de hand liggende is hoeveel libraries beschikbaar zijn of dat het programma in een objectgeoriënteerde style geprogrammeerd is; een paradigma wat encapsulation & information hiding aanmoedigt.

Selectie

Met de belangrijkste aandachtspunten op een rij kon er een keuze gemaakt worden. Deze keuze heb ik gemaakt in de vorm van een afvalrace.

De helft van de gegadigden kon in een enkele stap al afgestreept worden op basis van technische redenen. Babel, B.A.T.M.A.N, OpenBGPD & OpenOSPFD en de Click router zijn niet geschikt. De eerste vier omdat ze teveel zijn toegesneden op een enkel protocol of zelfs platform. De laatste omdat deze voor een ander (maar wel verwant) doel dan als routing platform geschreven is.

Er was ook nog een ander project gemakkelijk weg te strepen van de lijst: GNU Zebra is wegens het al lang uitblijven van updates (laatste is uit 2005) outdated. Quagga, als feitelijke opvolger, heeft daarom de voorkeur, omdat die in de tussentijd wel is mee gegaan met de ontwikkelingen in de wereld.

Met slechts twee selectierondes was de lijst al geslonken van 8 naar 3 gegadigden: BIRD, Quagga en XORP. Quagga echter schiet tekort in het documentatie gedeelte. Voor de gebruiker is er duidelijk werk in gestoken maar voor ontwikkelaars is het huidige aanbod zeer beperkt. Net als bijna elk ander actief open source project heeft Quagga een mailinglist en IRC kanaal, waar prima vragen gesteld kunnen worden. Echter hangt het krijgen van een antwoord van anderen af, die maar net even in de gelegenheid moeten zijn. Dit is een dealbreaker voor Quagga.

Dit liet BIRD en XORP over. Beide zijn technisch in orde en hebben op het eerste gezicht afdoende documentatie. Zo richt bij beide de documentatie zich niet alleen op gebruikers maar ook op ontwikkelaars. De BIRD source is ook goed gedocumenteerd door middel van comments.

Zowel XORP als BIRD richten zich op segregatie van functionaliteit, waarin XORP zelfs zover gaat dat het uit een aantal losse binaries bestaat. De XORP source tree is echter wel veel groter dan die van BIRD: 23 megabytes tegen slechts 2,7 megabytes.

En er is nog een verschil in taal waarin de programma's geschreven zijn. XORP is C++, BIRD is C. Dat betekent dat XORP objectgeoriënteerd is, in tegenstelling tot BIRD. Dit maakt verschil in aanpak. De reden dat object georiënteerde talen als C++ en Java zijn uitgevonden is om functionaliteit en informatie in te pakken en af te schermen; het zojuist genoemde encapsulation & information hiding. Dit maakt in theorie het programma beter te begrijpen. Als eerst omdat elk stukje afzonderlijk is te bestuderen en als tweede omdat het geheel bekeken kan worden zonder dat elk stukje precies doorgrond hoeft te zijn. Ook maakt het in theorie makkelijker functionaliteit toe te voegen omdat modules elkaar alleen via gedefinieerde input en output beïnvloeden.

20 D.L. Parnas: “On the Criteria To Be Used in Decomposing Systems into Modules”, Communications of the ACM, Volume 15, Number 12 (1972)

Uitslag

Uiteindelijk is de keus gevallen op XORP. XORP bestaat uit losse binaries waardoor de verwachting is dat nieuwe code "inschieten" makkelijker zal gaan bij XORP dan bij BIRD; het idee is dat alle binaries die XORP opmaken kunnen blijven draaien terwijl enkel de binary met het ant-based protocol er in overschreven en herstart wordt.

Ook is XORP geschreven in een objectgeoriënteerde taal, waarvan de verwachting is dat uitbreiden makkelijker gaat.

-Spoiler alert- Beide zouden later tegen blijken te vallen. Ondanks de segregatie van functionaliteit bij XORP waren er een hoop compiletime en linktime afhankelijkheden tussen de binaries en het feit dat de gebruikte taal object georiënteerd is bleek niet perse makkelijker prototypen te zijn.

4.4. Netwerksimulatie mogelijkheden

Tijdens de ontwikkeling van code is het nodig die te kunnen testen. Om daarin te kunnen voorzien is een ontwikkel/testomgeving nodig.

De meest voor de hand liggende methode is een omgeving bouwen door genoeg hardware bij elkaar te zoeken. Op deze manier is dan een kleine maar realistische kopie te bouwen van de omgeving waar het programma uiteindelijk zijn werk in moet doen.

Een alternatief voor een volledige hardwarematige opstelling is simulatie. Er zijn verscheidene netwerksimulatie pakketten ontwikkeld. Zowel proprietary als free software. Hiermee kan flink bespaard worden op de benodigde hardware.

Een andere mogelijkheid voor het simuleren van een netwerk is gebruik maken van virtualisatie. Bijna alle virtualisatiepakketten bieden de mogelijkheid een willekeurige netwerk topologie te definiëren. Via deze methode wordt voor elke instantie van de protocol implementatie een volledig virtueel systeem met operating system aangemaakt.

Elk van de opties heeft zo z'n voor en nadelen. Zie de volgende tabel met vergelijkingen:

Fysiek	Voordeel	Dichtst bij de werkelijkheid.
	Nadeel	Duur en onpraktisch bij grote netwerken.
Simulatie	Voordeel	Resource-vriendelijk.
	Nadeel	Te testen software moet het pakket ondersteunen.
Virtualisatie	Voordeel	Kan praktisch alle software een testomgeving bieden.
	Nadeel	Resource-onvriendelijk. Vereist een compleet OS op elke VM.

Fysiek heeft als er geen restricties waren de voorkeur. Het lijkt immers het meest op de real-world situatie waar de software later in functioneren moet. Maar er zijn restricties. Hardware kost geld, neemt ruimte in en vraagt energie. Met een beperkt aantal machines is het nog haalbaar, maar met tien of meer al bijna onmogelijk. Fysicaliseren valt om die reden af.

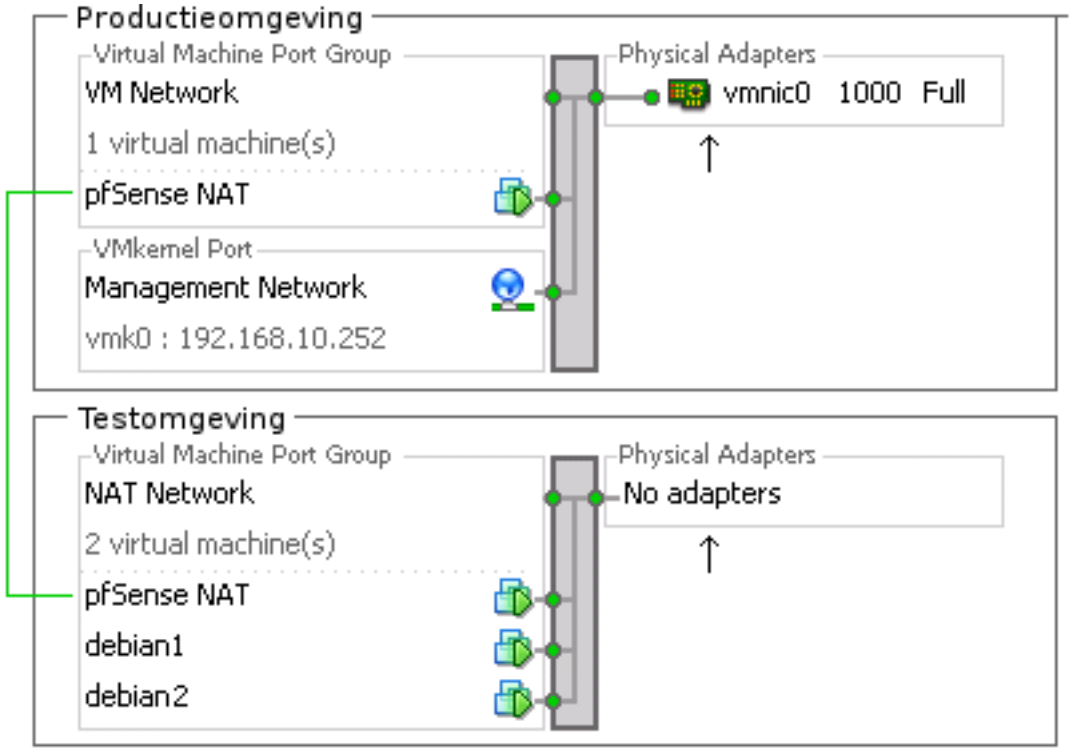
De tweede optie is een simulatiepakket als ns2 of GNS3 gebruiken. Het te testen software pakket moet echter “aan te sluiten” zijn op het simulatiepakket. En dat is met XORP (nog) niet het geval. Hierdoor was ook dit geen optie.

Dit liet virtualisatie als enigste optie over. Restte de vraag welk virtualisatiepakket ik zou gaan inzetten voor het bereiken van mijn doel. De keuze op dat vlak is heel simpel geweest: Ik had zelf geen enkele ervaring op het gebied van virtualisatie, collega's hadden ervaring met VMware. Deze situatie heeft mij bewogen zonder verdere afwegingen voor VMware te gaan daar het voldoende functionaliteit heeft.

VMware bleek echter een niet kritieke maar wel gewenste functionaliteit te missen: Ondersteuning voor NAT en firewalling. Om de rest van het bedrijfsnetwerk te beschermen tegen eventueel netwerkgeweld moest er een bepaalde mate van afscherming van de simulatie zijn. Tevens moet de simulatie niet de beschikbare adressen in het netwerk uitputten.

Om deze afscherming en eigen adresrange in acht te nemen was daarom NAT en simpele firewalling functionaliteit nodig. VMware ESXi echter heeft dat niet standaard aan boord. Om deze functionaliteit toch te implementeren is daarom een VM met pfSense opgenomen in de omgeving. Het pfSense OS is een op FreeBSD gebaseerde router distributie.

Zie het volgende diagram:



Het betreft een diagram gebaseerd op een screenshot van de VMware management software. In dit diagram zijn twee virtuele switches te zien. De bovenste virtuele switch, te zien in het kader “Productieomgeving”, is de switch die verbonden is met de fysieke netwerkpoort naar het bedrijfsnetwerk. De onderste virtuele switch, te zien in het kader “Testomgeving” heeft geen verbinding naar enige fysieke adapter.

Door deze opzet is de testomgeving logisch gezien een afgesloten omgeving. Er is geen weg naar buiten. Geen directe althans. Er is een VM die zowel in de testomgeving als de productieomgeving voorkomt: pfSense.

De pfSense VM heeft twee "aansluitingen" gekregen. De ene aan de virtuele switch van de testomgeving, de ander aan de virtuele switch van de productieomgeving. Hierdoor loopt de enige weg uit de testomgeving via pfSense. In pfSense wordt vervolgens op alle verkeer NAT en een set filterregels toegepast.

Op deze manier is de testomgeving vanaf het bedrijfsnetwerk gezien slechts een enkel IP adres (plus de management interface van VMware) en kan potentieel disruptief netwerkverkeer gevangen gehouden worden in de simulatieomgeving. Hierdoor is eventueel gevaar voor het bedrijfsnetwerk vele malen kleiner gemaakt.

4.5. Prototypen

Deze paragraaf beschrijft het daadwerkelijke prototypen wat is uitgevoerd om ideeën op te doen voor een protocol en een bijbehorende implementatie. Nadat de ontwikkelomgeving was opgezet was hiervoor de weg vrij. De originele verslaglegging hiervan is te lezen in het analyse & prototyping rapport[21].

Build system

Zoals bij §4.3 te lezen is, is XORP geselecteerd als basis. Op deze basis is vervolgens verder gebouwd. De XORP source tree is zodanig gemodificeerd dat er een eigen module is gecreëerd. Dit is bereikt door het build system van XORP aan te passen. XORP's build system is een gecompliceerd mechanisme wat geheel is toegesneden op XORP. De hoofdmoot bestaat uit SCons en Python, aangevuld met een aantal eigen parsers en definitie/scripttalen.

Dit build system genereert code, voert checks uit en laat de code uiteindelijk door de systeem eigen compiler (gcc) en linker (ld) vertalen tot een set binaries. Het start allemaal met de globale SCons files in de XORP root.

Zie het volgende snippet uit één van de globale SCons files:

```
subdirs = [
    'cli',
    'libcomm',
    'libxorp',
    'libxipc',
    'libproto',
    'libfeaclient',
    'xrl/targets',
    'xrl/interfaces',
    'etc/templates',
    'fea',
    'fib2mrib',
    'mld6igmp',
    'mrt',
    'pim',
    'policy',
    'rants',                # Running Ants!
    'rib',
    'rtrmgr',
    'static_routes',
    'utils',
    ]
```

Dit zijn de subdirectories die het build system moet aflopen. Elke representeert een module en/of library. In deze subdirectories staan ook weer één of meerdere SCons scripts die door het hoofd script worden uitgevoerd. Dit werkt vergelijkbaar met de bekende makefiles.

Aan deze lijst is de directory van de eigen module toegevoegd. Op dit moment moest er een naam voor de module en het protocol verzonnen worden. Uiteindelijk heb ik de naam *rants* bedacht, wat een samennemen is van 'running ants'.

In de subdirectory van rants bevinden zich ook een aantal configuratie files die toebehoren aan het build mechanisme. In deze wordt dan onder andere gedefinieerd welke files gecompiled moeten worden. Het is in het XORP project een gebruik om de basis van een programma in een executable te stoppen en de rest van de functionaliteit in een of meerdere libraries. De module is zo doende een losse binary met een library component erbij.

21 Project documentatie: Analyse & prototyping rapport, H6, *Bijlage E*

Zie het volgende snippet uit de SCons file van de rants module:

```
libxorp_rants_srcs = [
    'rants_route.cc',
    'io_udp.cc',
    'rants_node.cc',
    'xrl_rants_node.cc'
    ]

xorp_rants_srcs = [
    'xorp_rants.cc'
    ]
```

Het betreft een lijst met source files die tot de library gelinkt moeten worden en een lijst met source files (van 1 entries lang) die de normale binary moeten worden. Bij aanvang van het prototypen waren deze lijsten nog heel bescheiden in lengte.

Tijdens het functioneren van een module communiceert het met andere modules van het XORP platform. In dit prototype zijn wat communicatie betreft vier andere modules relevant. De rtrmgr, cli, fea en de rib.

De afkorting "rtrmgr" staat voor router manager. Dit programma is de spin in het web die alle andere programma's start, configuratie bepaalt en via waar alle berichten gaan. Het bevat ook standaard zaken als een logger en globale foutafhandeling.

Rants wordt bij het opstarten van het XORP platform door de rtrmgr gestart waarna het via een aantal sockets hiernaar verbinding maakt. Vanaf dat moment kan rants communiceren met andere modules.

De "cli" is de command line interface. Hierop kan de gebruiker aanpassingen maken aan de configuratie en de status van alle onderdelen zien. Veranderingen in waardes gaan via de rtrmgr naar de respectievelijke modules.

De term "fea" staat voor forward engine abstraction. Deze module abstraheert alle functionaliteit die nodig is voor het versturen van data over de netwerk interface. Door de functionaliteiten benodigd voor elk OS of soort hardware in een aparte module te stoppen blijven platform specifieke zaken weg uit de andere modules. Modules die data willen sturen en ontvangen doen dit via deze module.

Tenslotte is er de "rib", wat staat voor routing information base. Deze module aggregeert de door alle routing protocollen gevonden routes in een enkele tabel en manipuleert aan de hand daarvan (via de fea) de routing tabel van het onderliggende platform (bijvoorbeeld de kernel routing table of een stuk dedicated hardware).

Communicatie tussen modules

Hoe de communicatie tussen deze modules verloopt is zeer ingenieus. Dit verloopt via Unix sockets of TCP/IP. Het formaat waarin dit gebeurt is een XORP-specifiek mechanisme. Dit formaat is echter voor een module programmeur niet zichtbaar. De communicatie tussen modules is namelijk geabstraheerd achter een binnen XORP ontwikkelde definitietaal.

Hoe communicatie tussen modules plaatsvindt is gedefinieerd in .xif files. De afkorting XIF staat voor XRL InterFace. XRL op zijn beurt staat voor XORP Resource Locator, wat zoals de naam suggereert, eveneens binnen het XORP project is ontwikkeld. Beide methoden zijn dus XORP specifiek en helemaal gericht op het bieden van functionaliteit die het project nodig heeft.

Elke module heeft één of meerdere bijbehorende .xif files. In een .xif file worden de commando's waarmee de module kan worden aangestuurd vastgelegd. Wat dit bestand bevat lijkt dan ook veel op een API definitie. In het bestand wordt in text vastgelegd wat de naam, argumenten en return van een functie moeten worden. Tezamen met de datatypen voor elk veld. Dit is wederom een goed voorbeeld van hoe XORP gestructureerde modulariteit probeert aan te moedigen, simpelweg door de manier waarop communicatie tussen modules gedefinieerd moet worden.

Zie het volgende snippet uit de rants.xif file:

```
interface rants/0.1 {

	/**
	 * Enable/disable/start/stop rants.
	 *
	 * @param enable if true, then enable rants, otherwise
	 * disable it.
	 */
	enable_static_routes	? enable:bool;
	start_static_routes;
	stop_static_routes;
…
	/**
	 * Enable/disable the rants trace log for all operations.
	 *
	 * @param enable if true, then enable the trace log, otherwise
	 * disable it.
	 */
	enable_log_trace_all	? enable:bool;
}
```

Deze .xif file wordt in het build proces door een speciaal geschreven parser omgezet naar C++ code. De methoden die daar uit komen zijn wat heet pure virtual. Dit houdt in dat ze later, als ze eenmaal hun weg naar de module code hebben gevonden, geïmplementeerd moeten worden. De compiler dwingt dit af.

Nadat de API's gedefinieerd en gecompiled zijn naar .cc en .hh files worden deze nog niet direct opgenomen in de module code. De stap voor opname in de module komt in de vorm van een .tgt ofwel target file.

De target file bepaalt in feite welke API's de module allemaal moet ondersteunen. De meest voorkomende constructie is dat de "common" API, waarmee de rtrmgr de module start en stopt, en de module eigen/specifieke API wordt opgenomen. In het geval van rants komt daar ook nog een API interface voor het ontvangen van gebeurtenissen aan netwerkinterfaces en een interface voor het werken met sockets bij.

```
#include "common.xif"
#include "finder_event_observer.xif"
#include "socket4_user.xif"
#include "rants.xif"

target rants implements	common/0.1,			\
			finder_event_observer/0.1,\
			socket4_user/0.1,		\
			rants/0.1;
```

Het enige wat daarna aan de hand van deze target file gedaan wordt is feitelijk de gegenereerde C++ code van deze API's samenvoegen tot een enkele header (.hh) en source (.cc) file.

Zie hier een sterk ingekorte snippet uit de resulterende header voor rants:

```
virtual XrlCmdError common_0_1_get_version(…);

virtual XrlCmdError common_0_1_get_status(…);

virtual XrlCmdError common_0_1_startup(…) = 0;

virtual XrlCmdError common_0_1_shutdown(…) = 0;

…

virtual XrlCmdError socket4_user_0_1_recv_event(…);

…

virtual XrlCmdError rants_0_1_enable_static_routes(const bool& enable) = 0;

virtual XrlCmdError rants_0_1_start_static_routes();

virtual XrlCmdError rants_0_1_stop_static_routes();
```

Eenmaal de API definities uit de .xif files de keten hebben doorlopen, kan in de code van de module de betreffende header middels een #include statement aan de precompiler worden opgenomen. Hierna zal de compiler (wegens pure virtual) afdwingen dat er in de module code ook invulling wordt gegeven aan de gedefinieerde methoden.

De methoden die vanaf de .xif files hun weg de code in hebben gevonden zijn ook de enige methoden die door andere modules aangesproken kunnen worden. Dit betekent dat alle communicatie tussen modules verloopt via deze methodes. Met uitzondering van eventuele callbacks die meegegeven worden aan een API methode, is de API dus ook echt de enige weg.

Dit gehele mechanisme kan lokaal op dezelfde machine z'n werk doen of over een netwerkverbinding. Voor de module programmeur maakt dit niet uit. Alles wat die ziet is modules die bij elkaar methodes kunnen aanroepen. Echter “onder water” is er een systeem actief die alles over sockets en/of TCP/IP routeert.

Dit “onder water” systeem haalt een verwachting onderuit die bij XORP gesteld was: Tijdens de selectie van een framework/daemon heeft de verwachting dat nieuwe code “inschieten” makkelijker zou gaan wegens de opdeling in binaries mede de keus op XORP doen vallen. Dit bleek echter niet waar. Het is niet mogelijk om een binary uit de ene set te gebruiken met een oudere collectie binaries. Het build system voert tal van checks uit en genereert enorme lappen code voor onderlinge communicatie. Wanneer de ene module verandert, moet de rest daarop mee veranderen.

Commandline interface

Om een module te kunnen bedienen is interactie met de gebruiker nodig. Die interactie loopt via de cli module. Functionaliteit is uiteraard specifiek voor elke module, dus moet elke module ook een eigen set matchende commando's hebben. Binnen XORP worden daarvoor .cmds files gebruikt, die wederom een XORP-specifiek formaat hebben.

Zie hier een snippet van de rants.cmds file:

```
Protocols {
    rants {
      %help: short  "Configure rants";
      %modinfo:    provides rants;
      %modinfo:    depends rib;
      %modinfo:    depends policy;
      %modinfo:    path "xorp_rants";
      %modinfo:    default_targetname "rants";
      %modinfo:    status_method xrl "$(rants.targetname)/common/0.1/get_status-
                   >status:u32&reason:txt";
      %modinfo:    startup_method xrl "$(rants.targetname)/common/0.1/startup";
      %modinfo:    shutdown_method xrl "$(rants.targetname)/common/0.1/shutdown";

      disable {
          %help:          short "Disable the rants";
          %create:;
          %set:           xrl "$(rants.targetname)/rants/0.1/enable_static_routes?
                          enable:bool=`~$(@)`";
          %delete:        xrl "$(rants.targetname)/rants/0.1/enable_static_routes?
                          enable:bool=`~$(DEFAULT)`";
      }

      enabled {
          %deprecated:    "Statement 'enabled: true/false' is replaced with 'disable:
                          false/true'";
          %help:          short "Enable the rants";
          %create:;
          %set:           xrl "$(rants.targetname)/rants/0.1/enable_static_routes?
                          enable:bool=$(@)";
      }
…
}
```

In deze .cmds files kunnen boomstructuren van commando's opgebouwd worden. Deze commando's verwijzen naar de methoden eerder gedefinieerd in de .xif files. Bij het invoeren van een commando wordt de bijbehorende methode uitgevoerd en eventueel de return daarvan weergegeven.

Configuratie

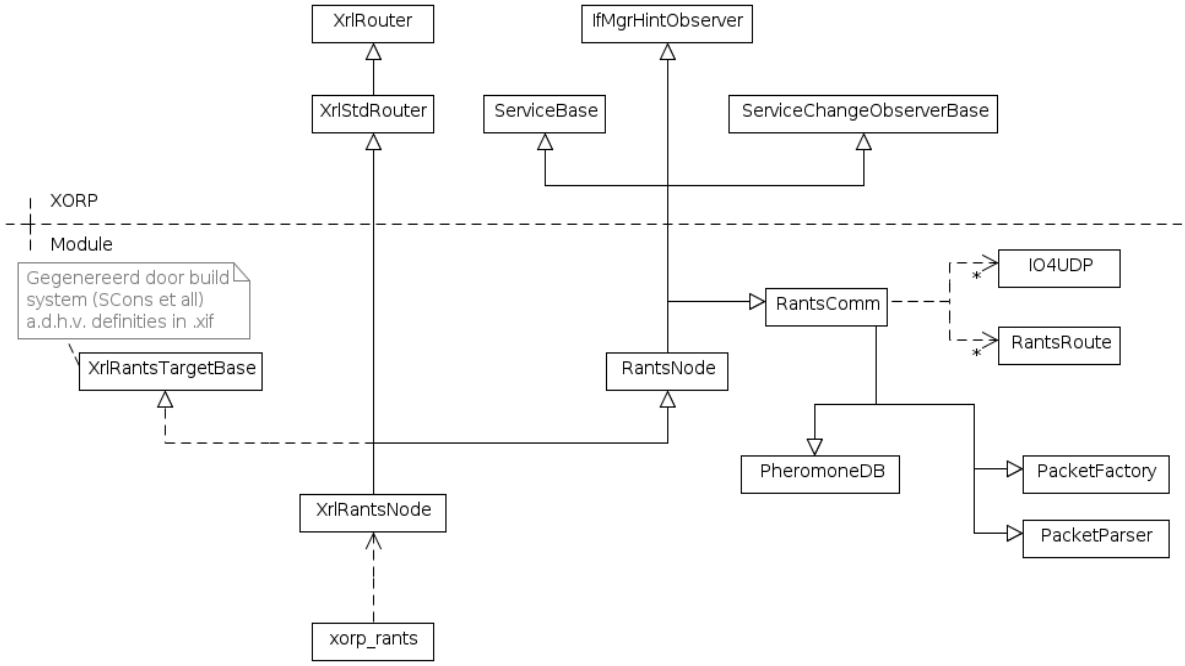
De CLI is niet alleen voor directe interactie via een terminal, hij functioneert ook als configuratie parser. De rtrmgr laadt een configuratiebestand en voert elke regel daarin aan de cli module, alsof het directe input van de gebruiker is. Zowel configuratiebestanden als interactie met de gebruiker vereisen dus commando's in de CLI. Elke module die configurabel moet zijn heeft dus een CLI interface nodig.

Implementatie

Nadat er een basis is die compileert, aanspreekbaar is door andere modules kan er begonnen worden aan de implementatie van functionaliteit.

Gezien XORP geschreven is in C++ wordt een object georiënteerde aanpak gebruikt. Een aantal klassen worden automatisch gegenereerd, een aantal voorgeschreven door “XORP practices” en een aantal zijn reeds in binaire vorm aanwezig.

Zie het volgende diagram:



Dit diagram is een diagram geïnspireerd op een klasse diagram. Hierin zijn de klassen direct betrokken bij de module ontwikkeling opgenomen.

Een automatisch gegenereerde klasse is bijvoorbeeld **XrlRantsTargetBase**. Deze wordt gegenereerd aan de hand van de .xif en .tgt files. Hierin worden dus de API methoden opgenomen. Er worden ook nog andere stukken code gegenereerd (de eerder genoemde “onder water” stukken), maar daar heeft de module schrijver niet direct mee te maken.

Onder voorgeschreven klassen vallen XrlRantsNode en RantsNode. En xorp_rants, maar dat is eigenlijk niet echt een klasse.

XrlRantsNode is de klasse waarin invulling wordt gegeven aan de pure virtual methoden uit XrlRantsTargetBase. Binnenkomende API aanroepen worden in deze klasse verwerkt en/of geforward naar andere stukken code.

De **RantsNode** klasse is waar de meeste van de “andere stukken code” hun plek vinden. Vanuit RantsNode wordt de feitelijke implementatie uitgebouwd. RantsNode kan zelf wat functionaliteit bevatten, maar het merendeel zal via overerving binnenkomen.

De “klasse” **xorp_rants** is eigenlijk geen klasse. Het bevat de main() functie van het programma en initialiseert de eerste objecten in het geheugen. Daarnaast bevat het ook een stuk van de eventloop, die wordt doorlopen wanneer er iets gebeurt wat voor de module relevant is.

De klassen die reeds in binaire vorm aanwezig zijn, staan in het klasse diagram (figuur 1) boven de horizontale gestippelde lijn.

XrlStdRouter en **XrlRouter** zijn de verbinding naar de rtrmgr. Deze klassen werken samen met de klassen betrokken bij de API aanroepen. Als module schrijver heb je hier niet direct mee te doen, anders dan dat ze onderdeel moeten zijn van de module (via overerving).

ServiceBase, **IfMgrHintObserver** en **ServiceChangeObserverBase** zijn van belang bij het openen van netwerk sockets en versturen/ontvangen van data daaruit. Ook met deze klassen heeft een moduleschrijver niet direct interactie mee. Ze dienen echter wel via overerving onderdeel te zijn van het programma om gebruik te kunnen maken van sockets via de fea.

Naast de door XORP aangeleverde, aanbevolen en reeds bestaande klassen zijn er uiteraard ook de rants eigen klassen. Dit zijn RantsComm, PacketFactory, PacketParser, PheromoneDB, IO4UDP en RantsRoute.

RantsComm is de klasse waar alle functionaliteit van het protocol samenkomt. Genereren en verwerken van packets, feromoonwaarden bijwerken, sockets openen/sluiten en het opslaan van gevonden routes wordt in deze klasse allemaal aan elkaar geknoopt.

De **PacketFactory** en **PacketParser** zijn, zoals de namen al weggeven, elkaars tegenhangers. De PacketFactory gaat een set variabelen in waarna er een stream aan bytes uit komt en de PacketParser zet een stream aan bytes om naar variabelen.

De **PheromoneDB** klasse bevat de feromoontabel van de node, tezamen met de operaties die er op uitgevoerd kunnen worden. Operaties als verdamping, verhoging van feromoonwaarden en het toevoegen van andere nodes die gevonden zijn in het netwerk. Ook kunnen deze waardes opgevraagd worden door andere klassen.

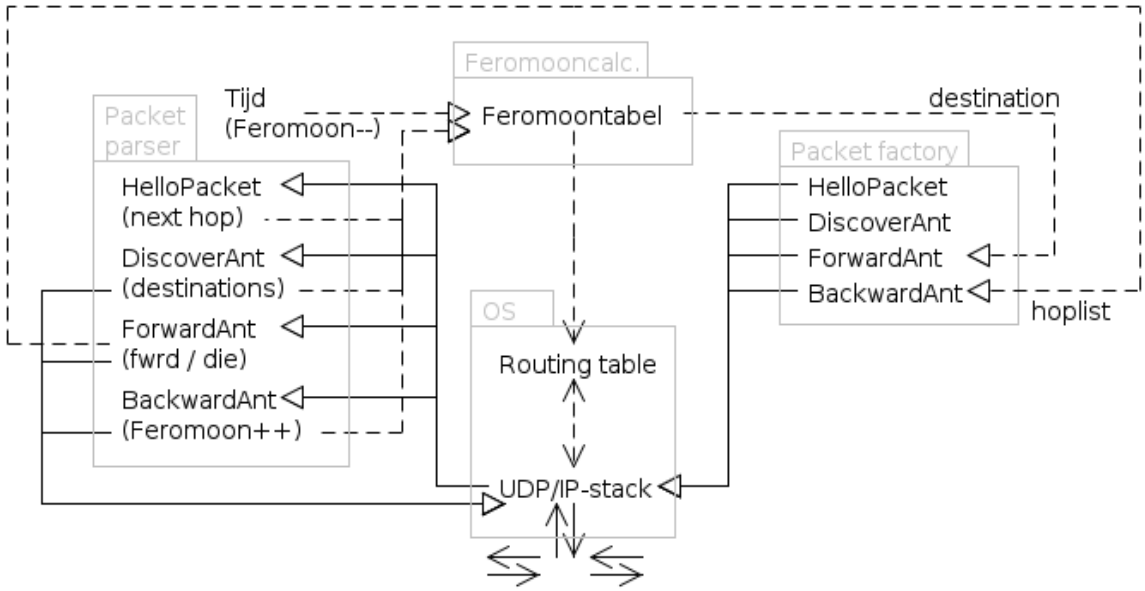
IO4UDP staat voor Input/Output IPv4 UDP. In deze klasse worden de gegevens van een IPv4 socket voor UDP opgeslagen. In de huidige code wordt er slechts één zo'n object aangemaakt, maar in de toekomst kunnen er prima meerdere sockets/poorten gebruikt worden op de UDP/IP stack.

De klasse **RantsRoute** is voor het opslaan van een gevonden route. Objecten naar het type van deze klassen zijn nodig om gevonden routes aan de rib module te leveren. De inhoud van deze klasse is hetzelfde tussen alle routing protocol modules en bevat daarom wat overbodige variabelen.

Functionaliteit

De werking van de rants module loopt, via XORP (fea) samen met het OS. Om de functionaliteit begrijpbaar te houden is deze opgedeeld in “blokken”.

Zie het volgende diagram:



Dit diagram geeft de paden weer waarover gegevens “lopen” tussen de zogenoemde blokken. Het is uiteraard een gesimplificeerde weergave van hoe het programma werkelijk in elkaar zit, maar geeft een goed beeld van hoe het werkt.

Het blok “OS” is in feite de netwerk componenten van het operating system met daar de XORP fea (forward engine abstraction) overheen. De overige drie blokken zijn onderdeel van de module en terug te zien in het klasse diagram.

Om de werking met het diagram uit te leggen, het volgende voorbeeld: Er is een node waarin een bepaalde timer afgaat, en een **HelloPacket** door de factory laat genereren. Deze wordt door het OS een interface uit gestuurd. Een andere node pikt de HelloPacket op, de node's parser verwerkt het pakket, waarna de afzender (indien nog onbekend) als next hop wordt toegevoegd aan de feromoontabel.

Andere voorbeelden zijn alle zeer gelijkend aan die in de vorige alinea. Zo zal het een **DiscoverAnt** hetzelfde afgaan als een HelloPacket, met als afwijking dat deze geen onbekende next hops maar onbekende destinations toevoegt aan de feromoontabel. En hij zal na verwerking (indien TTL nog niet verlopen is) weer doorgestuurd worden via het OS blok.

Of een **ForwardAnt**, die krijgt een willekeurige destination van de feromoontabel en verlaat de node via het OS blok. Met binnenkomen bij een andere node wordt de ForwardAnt naar een volgende node gestuurd of, als de destination het eigen adres is, dan is dat het einde van deze ForwardAnt. De lijst met de afgelegde weg gaat dan naar de packet factory, die de zogenoemde hoplist in een BackwardAnt laadt.

Wanneer een **BackwardAnt** een node bereikt zal het de feromoonwaarden in de feromoontabel beïnvloeden en wederom weer via het OS blok de node verlaten, naar de volgende node in de lijst. Tot de BackwardAnt weer bij de node is aangekomen die initieel de ForwardAnt had gestuurd.

Dan is er nog het verstrijken van tijd wat invloed heeft op de feromoonwaarden in de feromoontabel en het feit dat de feromoontabel de routing tabel van het OS bijwerkt. En daarmee is het diagram compleet.

Technisch zit het op sommige plekken iets ingewikkelder in elkaar. Zo zal in de code bijvoorbeeld te zien zijn dat elk pakket helemaal uit elkaar gehaald wordt en weer helemaal opnieuw opgebouwd in het verwerkingsproces. Het doorsturen van een pakket gaat dus feitelijk weer via de packet factory, in plaats van direct weer terug het OS blok in. Maar dat zou het diagram zo ingewikkeld maken dat het z'n doel voorbij zou schieten. Functioneel geeft het diagram goed weer hoe het proces loopt.

Opgeleverde code

Helaas heeft de analyse & prototyping fase geen werkend programma opgeleverd. De integratie met XORP is veel ingewikkelder gebleken dan gehoopt. Het correct laten functioneren van de module kostte zelfs zoveel tijd dat de ontwikkeling van de feitelijke ant-based code er onder geleden heeft. De implicaties voor het project zijn zowel technisch als projectmatig van aard. De technische kwestie volgt hieronder, de projectmatige zaken worden besproken in §4.6 en hoofdstuk 7: Evaluatie.

De belangrijkste verwachting was dat het gebruik van XORP een hoop tijd zou besparen op het gebied van ontwikkeling. Gezien XORP al een hoop functionaliteit aan boord heeft, lag het in de verwachting dat hier gebruik van maken te implementeren zaken zou schelen. Hierdoor zou weer meer tijd vrij zijn voor het feitelijke protocol werk.

Dit bleek niet het geval. Puntsgewijs:

- Slechte en verouderde documentatie
- Relaties tussen modules veel complexer dan gesuggereerd
- Samenvoegen routes uit verschillende protocollen belemmert rants

XORP profileert zich zelfs als geschikt voor netwerk onderzoek. Door strikte scheiding tussen modules d.m.v. API's en opdelen in processen zou het stabiel zijn en makkelijk om nieuwe functionaliteit aan toe te voegen. De stabiliteit claim heb ik niet aan de tand gevoeld, maar gemak bij het toevoegen van functionaliteit zou ik toch als onwaar willen bestempelen. De relaties tussen XORP modules zijn op code niveau zeer complex. Zowel compiletime als linktime. Dit maakt code toevoegen of verwijderen veel moeilijker dan gesuggereerd wordt. Als er duidelijke documentatie bij geleverd was had werken met XORP nog mogelijk geweest, maar ook hier schoot het XORP project helaas tekort. Veel documentatie is verouderd.

Het is ook gebleken dat de aggregratie van routes uit verschillende protocollen naar een enkele tabel niet uitgeschakeld of omzeild kan worden. Dit wil zeggen dat rants die infrastructuur van XORP moet gebruiken. Alle protocollen binnen XORP werken volgens het geconvergeerd/niet-geconvergeerd paradigma. Omdat rants geen geconvergeerde of niet-geconvergeerde staat kent is de abstractie waarschijnlijk niet berekend op de vele veranderingen die rants maakt.

XORP als geheel is ontworpen en gebouwd naar een ander model dan rants, waardoor zaken waarschijnlijk spaak gaan lopen. De zaken waar nu al tegenaan gelopen is voorspelt niet veel goeds.

Om die redenen heb ik besloten het project na de fase analyseren & prototypen niet voort te zetten met XORP als basis. Bij aanvang van de software architectuur ontwikkelen fase zal daarom opnieuw een keuze worden gemaakt uit de beschikbare routing frameworks. Dit is beschreven vanaf hoofdstuk 6.

Wat de analyse & prototyping fase wel heeft opgeleverd is een hoop code die bijna af is. De structuur van het programma, zoals die in voorgaande hoofdstukken is besproken, is compleet. Alle code is aanwezig, maar mist een werkende verbinding naar onderliggende OS/netwerk functionaliteit. Dit maakt dat alle code ongetest is.

Zie het volgende snippet code:

```
bool PacketParser::parse(
                // Input var
                const vector<uint8_t>& packet,
                // Output vars
                short int& type, short int& ttl, IPv4& source, IPv4& dest,
                short int& origsize, vector<IPv4>& hoplist) {

        /* The common part */
        if ((packet.front() & 0xF0) != 0xB0) {
                // Not a rants packet!
                return false; // Just false for now, might do something more clever with it later
        }
        // type is set later on

        // The second byte in the packet is the ttl
        uint8_t ttl8b = packet.at(1);
        ttl = (short int) ttl8b;


        // 4 source address bytes need to be unchopped to 32bits
        uint32_t source32b = 0;
        for (i = 2; i <= 5; i++) {
                source32b <<= 8;
                source32b += packet.at(i);
        }

        // source is a function argument
        source.set_addr(source32b);

……
```

Dit is een algemeen stuk code daar het niet kritiek op functionaliteit van een bepaald platform leunt. Stukken code als deze zijn prima opnieuw te gebruiken in een volgende iteratie of ander programma.

In hoofdlijnen is het volgende compleet:

- Opbouwen van rants packets (incl. alle payloads)
- Parsen van rants packets (incl. alle payloads)
- Sequentie in het programma

Het volgende is nog halfslachtig/incompleet:

- De feromoontabel
- Verbinding naar het onderliggende OS

Het volgende is nog oningevuld:

- Vertaling van feromoontabel naar standaard IP stack waarden

Ondanks dat de XORP module nooit zal worden afgemaakt is het werk verre van voor niets geweest. De wisselwerking van code schrijven, denken en ontwerpen is de bodem geweest voor tal van aspecten in het protocol, bitpatroon en code opbouw. Een simpel te implementeren protocol met logisch en makkelijk te verwerken bitpatroon is het halve werk.

Tegelijk werken aan een implementatie en protocol tezamen met bitpatroon heeft dus deugd gedaan, ook al zal de eerste poging tot implementatie nooit afgemaakt worden.

4.6. Projectbeheersing

Aan het eind van de 11e projectweek zou volgens de planning de analyseren & prototypen fase aflopen. Op dit moment was echter nog weinig hard resultaat behaald uit het experimenteren met XORP. Om die reden heb ik toen besloten van de planning af te wijken en de geplande tijdsindeling op te rekken.

Het oprekken van de tijdsindeling liet de planning met bijna 4 weken uitlopen. Net voor het begin van de 15e week was er voldoende resultaat om de analyse & prototyping fase te verlaten. Pas vanaf dit punt ging er aan een daadwerkelijk product begonnen worden, in de vorm van het opstellen van een protocol en implementatie.

De normale looptijd van een afstudeerproject is 20 weken. Dit betekende dat ik op dat moment slechts 5 weken over had voor de uitvoer van het project en de bijbehorende documentatie. Dit had zeer waarschijnlijk te weinig geweest voor het opleveren van voldoende materiaal, in het speciaal dit verslag. Op het tussentijds assessment is om die reden besloten het project te verlengen met 10 weken. De duur van het afstudeerproject is zo doende van 20 naar 30 weken opgerekt.

De verandering in duur is terug te zien in de verschillen tussen twee versies van het plan van aanpak. Het plan van aanpak bevat de planning behorende bij dit project. In versie 0.4[22] is de geplande tijd voor de fasen en activiteiten anders dan in die uit laatste versie (0.6)[23].

De meest notabele verschillen zijn de fasen analyseren & prototypen en implementeren & testen.

De fase analyseren & prototypen is van 51 dagen naar 80 dagen opgerekt. Hiermee eindigde volgens de nieuwe planning het analyseren & prototypen in week 16, waardoor het project weer op schema was. De fase implementeren & testen is van 21 naar 41 dagen begroot; een toename van 4 weken.

Deze toename in beschikbare tijd gaf mij weer ruimte om verder te werken aan het project en de kansen op het behalen van mijn doel te vergroten.

22 Project documentatie: Plan van aanpak v0.4, *Bijlage C*
23 Project documentatie: Plan van aanpak v0.6, *Bijlage D*

5. Opstellen protocol

Om programma's onderling te laten communiceren is een taal nodig die aan beide kanten wordt begrepen: Berichten die volgens een bepaalde set regels worden geconstrueerd en geïnterpreteerd aan weerszijden van de verbinding. Dit wordt veelal aangeduid als een protocol. Een protocol is vitaal voor onderlinge compatibiliteit en samenwerking.

Het ant-based routing protocol is een van de hoofdproducten van dit project. Het is de helft van het beoogde resultaat en een voorwaarde voor het behalen van de doelstelling. De doelstelling is namelijk een ant-based routing protocol testen. Het beoogde resultaat daarbij is een ant-based routing protocol getest met een prototype implementatie.

Pas wanneer er een 'concept' (protocol) is, kan een bijbehorende 'proof' (implementatie) geschreven worden. Daarmee is het een belangrijk product en mijlpaal op weg naar de proof.

Het opstellen van het rants protocol is voornamelijk in de analyse & prototyping fase gebeurd. Gedurende deze fase zijn de concepten en structuren bedacht waar het geheel verder rond uitgewerkt zou worden. Direct daarop volgend is de requirements bepalen fase van start gegaan. In de requirements fase zijn de concepten en structuren uitgewerkt en daarna geformaliseerd[24].
Dit hoofdstuk beslaat op die reden niet alleen de requirements bepalen fase, maar ook de staart van de analyse & prototyping fase.

Vertaling van biologie naar digitaal

Mierengedrag is enorm complex en de manier waarop zenuwcentra beslissingen nemen is anders dan hoe een computerprogramma berekeningen uitvoert. Daarom imiteren ant-based algoritmen maar een klein gedeelte van de aanpak die een mierenkolonie heeft. Het rants protocol als geheel is zo'n ant-based algoritme. Het rants protocol is één van de vele vormen en toepassingen waarin gemodelleerd mierengedrag gebruikt wordt als oplossing voor een probleem.

Digitale mieren lopen net als hun biologische voorbeelden willekeurig maar bevooroordeeld door geursterktes rond in hun omgeving. Daarbij hebben ze eveneens een methode om de afgelegde weg te onthouden, die ze terug kunnen volgen als het zover komt.

Wanneer digitale mieren gegenereerd worden krijgen ze een willekeurige knoop als einddoel mee, waarna ze die gaan proberen te bereiken. Dit in tegenstelling tot biologische mieren, die een algemeen doel als “alles wat eetbaar is” hebben. Binnen een instelbare hoeveelheid stappen moet een digitale mier het einddoel bereikt hebben. Is dat niet gelukt, dan houdt de mier op te bestaan (denk aan een TTL).

Een ander groot verschil met de versie van moeder natuur is dat er binnen het rants protocol geen centrale mierenhoop is. Mieren worden door elke knoop in het netwerk gegenereerd en op pad gestuurd.

Ook ziet de omgeving van digitale mieren er anders uit dan die van biologische mieren. De beestjes in de achtertuin hebben een driedimensionale wereld om in te bewegen. De digitale “beestjes” bewegen zich over een graaf van netwerkverbindingen.

De geuren die mieren in de echte wereld volgen en achterlaten in hun omgeving zijn bij rants gemodelleerd als een getal. Elke mogelijke richting vanuit een knooppunt in de graaf (node) heeft een waarde, die de geursterkte, ofwel aantrekkingskracht representeert.

24 Project documentatie: Requirements document, *Bijlage F*

De aantrekkingskracht die een weg heeft is tevens afhankelijk van het doel. Een andere manier om hetzelfde te zeggen zou zijn dat elk doel z'n eigen geur heeft. Op welke geur een mier z'n voorkeur baseert is dus afhankelijk van welk doel hij heeft.

De datastructuur waarin de mogelijke wegen tegen de doelen worden uitgezet heeft het meest weg van een tweedimensionaal matrix. Deze matrix staat bekend als de feromoontabel, naar de algemene biologische benaming voor geurstof; feromoon. Dus in plaats van sporen van feromoon in de tuin, implementeert elk knooppunt in de graaf een geheugenstructuur als de volgende matrix:

Destinations	Next hops: B	Next hops: C
A	0.12	0.13
B	0.99	0.15
C	0.21	0.99
D	0.05	0.01

De digitale mieren worden bij het bepalen van hun volgende stap (next hop) naar het doel wat ze hebben (destination) net als echte mieren beïnvloed door de sterkte van de “geur” (een waarde of kans liggende tussen 0 en 1). Elke stap is willekeurig, maar bevooroordeeld door de geursterkte.

Net als de biologische mier laat een digitale mier een “geurspoor” achter op de weg terug als die succesvol is geweest. Ofwel een mier die succesvol is geweest in het bereiken van z'n doel, gaat exact dezelfde weg terug en maakt daarbij op alle gebruikte wegen de “geur” (0..1) geassocieerd met het bereikte doel sterker.

In tegenstelling tot biologische mieren laat de digitale mier geen geurspoor achter op de weg heen. Echte mieren doen dit wel. Die laten als Hans en Grietje een soort kruimelspoor achter naar huis. Dit geurspoor trekt soms ook wat andere mieren. De digitale mieren van rants doen dit niet. Zij houden in plaats daarvan intern een lijst bij met genomen wegen.

Het geurspoor is bij rants net als bij biologische mieren onderhevig aan “verdamping”, ofwel afname in sterkte. Er is een instelbare procentuele afname per tijdseenheid die alle waarden in de feromoontabel langzaam verlaagt. Wanneer er slechts alleen verhoogd zou worden zouden namelijk de getallen in de feromoontabel op een moment allemaal het plafond raken. Tevens zorgt deze afname voor het vergeten van irrelevant geworden routes.

Tot zover zijn de digitale mieren van rants nog gepresenteerd als iets wat zelf beslissingen kan nemen, zoals de biologische mieren dat kunnen. Maar eigenlijk kunnen ze dat niet. De mieren van rants zijn geen stukjes programma, maar stukjes data die van knooppunt naar knooppunt verstuurd worden. De mieren zijn in feite gewoon packets die over het netwerk gaan en verwerkt worden door de nodes die ze passeren.

Toevoegingen

De biologische mieren konden echter niet als voorbeeld dienen voor alle benodigde functionaliteit. Er waren nog wat zaken in te vullen die in een fysieke, driedimensionale omgeving niet van toepassing zijn. Ofwel waar mieren dus nooit een oplossing voor hebben hoeven vinden.

Zo is in die fysieke wereld een stap naar voren, een stap naar voren. Simpel als dat. In een netwerk is het niet per definitie duidelijk wie of wat er aan de andere kant van de lijn zit en wie en wat er nog meer in het netwerk aanwezig is.

Waar de digitale mieren dus heen kunnen bewegen of heen gestuurd kunnen worden is bij initialisatie onbekend. Met andere woorden: De topologie van de graaf (het netwerk) moet op een manier uitgevonden worden.

Hiervoor zijn twee middelen bedacht: Een signaal om een direct verbonden participerende computer te detecteren en een mier wiens doel het is om alle participerende knopen bij elkaar bekend te maken.

Het signaal om direct verbonden buren te vinden is het zogenoemde “hallo” signaal. Deze is om de buurman te laten weten dat de afzender ook een rants node, ofwel participerende knoop is. Als de andere kant ook het rants protocol ondersteunt wordt de afzender opgenomen in de feromoontabel, als next hop. Uiteraard gaan deze hello-berichten twee kanten op.

De mier die alle participerende computers bij elkaar bekend moet maken doet dit door willekeurig rond te lopen (zonder enig besef van geuren e.d.). Elke node die deze mier langskomt onthoudt hij. Elke node waar zo'n mier voorbij komt loopt deze lijst door om de tot dan toe onbekende nodes in z'n destinations as van de feromoontabel toe te voegen. Na een beperkt aantal stappen (TTL) houdt deze verkenner mier op te bestaan. Elke node stuurt periodiek zo'n mier het netwerk op.

Opzet

Zoals eerder vermeld zijn de mieren in rants feitelijk datapakketjes die over het netwerk verstuurd worden. Er zitten wat adres velden in tezamen met een paar bijkomende variabelen. Alle verwerking daarvan gebeurt in knopen die het rants protocol ondersteunen; de zogenoemde nodes.

De rants mieren bestaan uit een zelf bedacht bitpatroon wat in een UDP/IP pakket verstuurd wordt. Als protocolnummer/poortnummer is voor 54273 gekozen. In deze cijfers kan het woord “rants” gezien worden. Totdat de IANA een poortnummer beneden de 1024 toewijst blijft 54273 het poortnummer waar rants op functioneert.

Er is voor UDP gekozen uit oogpunt van betrouwbaarheidseisen, resources, snelheid en gemak bij implementatie.

Het rants protocol stuurt namelijk voortdurend mieren uit over het netwerk. Ant-based algoritmen zijn meta-heuristics[25], wat inhoudt dat ze uit een poel van mogelijkheden de beste optie proberen te selecteren. Dit houdt in dat het zoekraken van een paar mieren geen probleem is, gezien er nog tal van andere overblijven. Het is ingecalculeerd, bijna onderdeel van de tactiek.

De garanties die bijvoorbeeld TCP dan biedt zijn leuk om te hebben, maar ze komen met een prijs. TCP gebruikt significant meer resources dan UDP of andere transport protocollen die connectionless zijn. Dit kan problemen geven op platforms met weinig resources of in configuraties met een hoge “mierenfrequentie”.

Ook heeft het opzetten van TCP sessies een negatieve impact op performance in termen van snelheid, omdat elke sessie opbouwen en afbreken tijd kost. UDP pakketten kunnen direct uitgestuurd worden. Tevens levert elk zoekgeraakt pakket bij TCP vertraging op voor de rest, omdat TCP om retransmission gaat vragen om de data compleet te krijgen en in dezelfde volgorde te zetten als verstuurd.

Als laatste is TCP waarschijnlijk ook moeilijker bij ontwikkeling. Het is veel complexer dan UDP vanwege alle handige functionaliteit die het biedt. Maar het is functionaliteit die niet nodig is bij een protocol als rants. De functionaliteit van TCP staat die van rants eerder in de weg. Er is simpelweg niet meer nodig dan UDP biedt.

25 Maarten van Eeuwijk, Vincent Olsthoorn: Ant-based algoritmen in IP routing (2011) H2, §2, *Bijlage A*

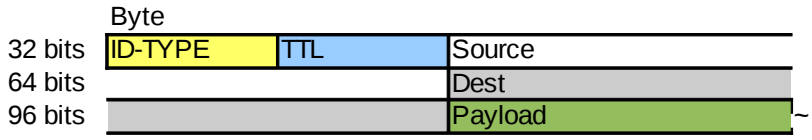
Momenteel is het rants protocol nog slechts als IPv4 variant uitgewerkt. De beslissing om nog geen IPv6 in het ontwerp op te nemen is genomen op basis van het feit dat het niet een vereiste feature is om een proof-of-concept op te kunnen leveren. Een IPv6 versie zou het ontwerp om die reden onnodig ingewikkeld maken en tijd wegnemen bij wel relevante zaken. Ondersteuning voor IPv6 is wel een mogelijke uitbreiding voor toekomstige versies van het rants protocol.

5.1. Bitpatroon

Om data uit te wisselen tussen rants nodes is een bepaald communicatie formaat nodig. Gegevens moeten op een bepaalde manier naar een patroon van bits en bytes gecodeerd worden.

Voor rants is gedurende het prototypen van de analyse & prototyping fase een eigen bitpatroon uitgedacht. In dit bitpatroon worden de benodigde gegevens gecodeerd in nullen en enen. De resulterende stream aan data wordt vervolgens door het OS in een UDP datagram en IP pakket ingepakt.

Zie het volgende schema:



Dit schema geeft de algemene structuur van elk rants packet weer (minus UDP/IP). Alle rants packets gebruiken deze structuur, waarbij de enige verschillen zitten in ID-TYPE en de payload.

De minimum lengte van een packet is 96bits. Wanneer dat niet gehaald wordt omdat de payload te klein is wordt deze ruimte opgevuld (beter bekend als padding). Het packet wordt groter wanneer de payload begint te groeien.

Er is nog geen limiet bepaald voor de maximale grootte van de payload of het packet als geheel, al heeft UDP een effectieve maximale data grootte van 65.507 bytes[26] per pakket.

5.1.1 Ants

De payload is specifiek voor het type packet. Momenteel zijn er vier soorten “mieren” in het protocol opgenomen: HelloPacket, DiscoverAnt, ForwardAnt, BackwardAnt.

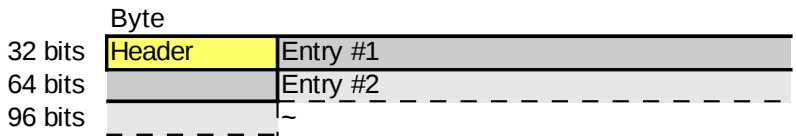
HelloPacket

Het HelloPacket bestaat slechts uit padding. De inhoud maakt niet uit. Het kunnen allemaal nullen zijn. Het enige doel van de HelloPacket is aan de buurman kenbaar maken dat er bij de afzender rants ondersteuning aanwezig is.

26 http://en.wikipedia.org/wiki/User_Datagram_Protocol#Packet_structure

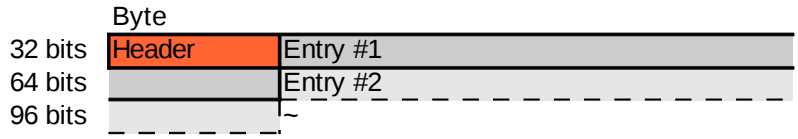
DiscoverAnt

De DiscoverAnt payload bestaat uit een header van 8 bits, die aangeeft hoe lang de daarop volgende lijst van 32bit adressen is. De DiscoverAnt payload is dus de header + de waarde van de header * 32 bits lang.



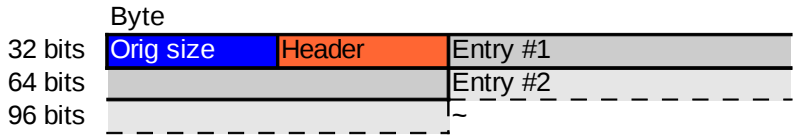
ForwardAnt

De ForwardAnt payload heeft exact dezelfde structuur als die van de DiscoverAnt. De verwerking in de software is uiteraard anders.



BackwardAnt

De BackwardAnt payload is nagenoeg gelijk aan die van de DiscoverAnt en ForwardAnt. Het enige verschil is dat voor de header (die de lengte van de daarop volgende lijst vastlegt) nog een extra veld van een byte is gedefinieerd die de originele lengte van de lijst vasthoudt. Deze waarde is van invloed op berekeningen in de software.



De BackwardAnt is ook meteen de grootste payload in het rants protocol.

Twee bytes header en een lijst maximaal zo lang als geteld kan worden met een enkele byte.

Met een enkele byte kun je tot 255 tellen; 4 bytes per entry in de lijst, maakt dat de lijst maximaal 1.020 bytes lang kan zijn. Tel daarbij op de twee header bytes; 1.022 bytes is momenteel de grootst mogelijke payload in gebruik. Zelfs met de 80 bytes van het algemene deel erbij ruim binnen de limiet van een UDP/IP pakket (65.507 bytes).

5.1.2 Betekenis velden

Elk veld heeft zo zijn betekenis tijdens de verwerking. Een drietal velden echter lijken op het eerste gezicht overbodig. Te weten TTL, source en destination zijn ook al in de IP header te vinden. De velden in het rants gedeelte hebben echter een iets andere betekenis. Ze zijn van een hogere abstractie dan die in de IP header. De velden in een rants packet hebben veelal een net iets andere betekenis dan iemand onbekend met het protocol verwacht.

ID-TYPE

Het ID-TYPE veld bepaalt de verwerking van het packet, gezien elk soort packet een andere betekenis en payload heeft. Het heet ID-TYPE omdat het zowel gebruikt wordt om te herkennen of het een rants packet is (ID) en om te zien van welk type ant (TYPE) het is.

Als eerste vier bits voor dit veld (“ID”) is willekeurig gekozen voor 1101 / 0xD. Samen met de laatste vier bits (“TYPE”) is het 11010000 / 0xB0.

Gezien er nu 4 types ants of packets zijn kan daar nog respectievelijk 1, 2, 3, of 4 bij opgeteld worden. Of 0x1, 0x2, 0x3, 0x4, wat 0xB1 tot en met 0xB4 maakt als mogelijke waardes van het ID-TYPE veld. Als het iets anders is, is het geen rants data of een andere (niet ondersteunde) versie van het protocol.

TTL

De afkorting TTL zal voor de meeste lezers van dit document bekend voorkomen. TTL staat voor Time To Live. Het staat voor het aantal stappen, of hops, dat het packet nog te leven heeft. Bij elke rants node wordt dit veld met 1 verlaagd. Wanneer het de nul bereikt zal het pakket niet doorgestuurd worden.

De TTL in het rants gedeelte is van een hogere abstractie dan die in de IP header. De TTL in de IP header is van toepassing op de weg tussen twee rants nodes. Daar kunnen prima hosts tussen zitten die niets met de rants data doen. De TTL van het rants gedeelte loopt dus alleen maar af bij het passeren van een rants node, omdat daar pas het rants gedeelte verwerkt wordt.

Source

Het source veld is de afzender van het rants packet. Ook hier is de abstractie hoger dan het source veld in de IP header. Het source veld in de IP header is, net als in het verhaal bij TTL, van toepassing op de weg tussen twee rants nodes. Dit betekent dat het source veld van de IP header en die in de rants data maar één keer hetzelfde is: Bij de eerste sprong naar een andere rants node. Het source veld in de rants data verandert niet, terwijl bij elke beweging tussen nodes een nieuwe IP header wordt gemaakt met nieuwe adressen.

Destination

Het destination veld codeert het doel van het rants packet. Het verhaal over abstractie bij source gaat ook hier op voor destination. Echter is het destination veld van de rants data en de IP header niet aan het begin gelijk, maar bij de laatste sprong tussen twee rants nodes. Als deze laatste sprong het bereiken van het doel inhoudt tenminste, niet als het de laatste sprong is vanwege het verstrijken van de TTL.

Payload

Het payload veld is het enige veld wat varieert in lengte. En varieert in betekenis: Het ID-TYPE veld bepaalt de betekenis van de data in dit veld, gezien elke soort packet een andere payload heeft. Hieronder worden de payloads per packet type besproken.

– **HelloPacket**

De payload van een HelloPacket bestaat slechts uit padding. Twee bytes om van 80 bits naar de minimale 96 bits in totale lengte te komen.

– **DiscoverAnt**

De payload van een DiscoverAnt bestaat uit een header en een serie entries.

- Header

De header van de DiscoverAnt payload is een 8 bit getal, waarmee de lengte van de komende lijst met entries wordt aangegeven. 8 bits om mee te tellen betekent dat de komende lijst maximaal 255 entries lang kan zijn.

- Entry

Een entry is een getal van 32 bits lang en staat voor een IPv4 adres. In het geval van de DiscoverAnt representeert een entry een node waar de ant is langs geweest. Elke nieuwe entry wordt achteraan aan de lijst gevoegd. Nodes waar de DiscoverAnt langs komt kunnen uit deze lijst onbekende adressen leren.

– **ForwardAnt**

De ForwardAnt payload heeft dezelfde structuur als de DiscoverAnt payload. De functie echter is anders.

- Header

De header van de ForwardAnt payload is een 8 bit getal, waarmee de lengte van de komende lijst met entries wordt aangegeven.

- Entry

Een entry is een getal van 32 bits lang. Ofwel een IPv4 adres. In het geval van de ForwardAnt representeert een entry een node waar de ant is langs geweest. Elke nieuwe entry wordt achteraan aan de lijst gevoegd. Deze lijst dient als input aan de BackwardAnt als de ForwardAnt succesvol is geweest in het bereiken van z'n doel.

– **BackwardAnt**

De payload van een BackwardAnt is zeer gelijkend aan die van de DiscoverAnt en ForwardAnt. Hij bestaat eveneens uit een header en een serie entries. Deze worden echter vooraf gegaan door het orig size veld.

- Orig size

Het Orig size veld legt de originele lengte van de lijst met entries vast. De lijst met entries slinkt namelijk met het passeren van elke node, terwijl de lengte van de door de ForwardAnt gevonden weg belangrijk is bij berekeningen in de software. Het beïnvloedt namelijk de ophoging van waarden in de feromoontabel. Langere weg = minder ophoging.

- Header

De header is een getal van een byte, waarmee de lengte van de komende lijst met entries wordt aangegeven. Een byte is maximaal 255 entries in de komende lijst van entries.

- Entry

Een entry is een getal van 32 bits lang. Ofwel een IPv4 adres. In het geval van de BackwardAnt representeert een entry een node waar de ant nog langs moet. De lijst met entries is namelijk overgenomen van een ForwardAnt die succesvol is geweest in het bereiken van zijn doel. De BackwardAnt neemt dezelfde weg in omgekeerde volgorde om daarbij feromoonwaarden te verhogen. Bij elke stap wordt de laatste entry van de lijst af gehaald. Ofwel de lijst wordt in omgekeerde richting weer afgebroken als de ForwardAnt die had opgebouwd.

5.2. Concreet

Concreet, in punten, wat een rants node moet doen:

- Als node broadcast je regelmatig HelloPackets netwerkinterfaces uit.
- Als node genereer je regelmatig DiscoverAnts, waarna je die naar een willekeurige next hop stuurt.
- Als node stuur je regelmatig ForwardAnts naar een willekeurige destination.

- Bij ontvangen van een HelloPacket voeg je de afzender toe aan de next hops as in je feromoontabel. Hierna houdt de HelloPacket op te bestaan.
- Bij ontvangen van een DiscoverAnt leer je onbekende adressen uit de lijst die de ant meedraagt en voeg je deze toe aan de destinations as in je feromoontabel.
- Bij ontvangen van een DiscoverAnt voeg je je eigen adres toe aan de lijst met gepasseerde nodes die hij draagt.
- Na verwerken van een DiscoverAnt stuur je deze door naar een willekeurige next hop.

- Bij ontvangen van een ForwardAnt voeg je je eigen adres toe aan de lijst met gepasseerde nodes die hij draagt.
- Bij ontvangen van een ForwardAnt controleer je of de lijst met gepasseerde nodes geen loop bevat. Verwijder die indien nodig.
- Na verwerken van een ForwardAnt stuur je deze door naar een willekeurige next hop, waarbij de keuze bevooroordeeld wordt door de feromoon niveaus geassocieerd met de destination.

- Bij ontvangen van een ForwardAnt met als destination jijzelf voeg je ook je eigen adres toe aan de lijst met gepasseerde nodes die hij draagt.
- Na verwerken van een ForwardAnt met als destination jijzelf genereer je een BackwardAnt. De source van de ForwardAnt wordt de BackwardAnt's destination, de destination van de ForwardAnt is de source van de BackwardAnt, de hoplist wordt payload. Hier houdt de ForwardAnt op te bestaan.

- Bij ontvangen van een BackwardAnt verhoog je de feromoonwaarde voor de intersectie tussen de laatste node in de hoplist (next hop in je feromoontabel) en de source (destination in je feromoontabel), proportioneel naar de originele lengte (orig size veld van packet) van de hoplist.
- Na verwerken van een BackwardAnt verwijder je de laatste node uit de hoplist en stuur je de BackwardAnt door naar de dan laatste node in de hoplist.
- Bij ontvangen van een BackwardAnt met als destination jijzelf verhoog je de feromoonwaarde voor de intersectie in de feromoontabel. Hier houdt de BackwardAnt op te bestaan.

- Voor alle binnenkomende packets geldt dat de TTL met 1 verlaagd moet worden.
- Voor alle packets geldt dat een packet niet meer doorgestuurd wordt wanneer de TTL 0 bereikt. De data er in wordt echter nog wel verwerkt (wanneer zinvol).

De frequentie waarmee HelloPackets, DiscoverAnts en ForwardAnts uitgestuurd worden moet configurabel zijn.

De initiële feromoonwaarden in de feromoontabel moeten configurabel zijn.

De “verdamping” van feromoon in percentage en met welke frequentie die berekening wordt uitgevoerd op de gehele feromoontabel moet configurabel zijn.

Bij het doorsturen van een ForwardAnt is de keuze voor de next hop willekeurig maar proportioneel naar de feromoonwaarde geassocieerd met die next hop. Dit houdt in als er twee mogelijke next hops zijn, de ene met een fermoonniveau van 0.25 en de ander met 0.75, de eerste 25% kans heeft en de tweede 75% kans om de ForwardAnt toegestuurd te krijgen.

Bij het ophogen van een waarde in de feromoontabel door de BackwardAnt is een feromoonconstante van toepassing. De feromoonconstante wordt gedeeld door de totale lengte van de weg die gevonden was door de ForwardAnt (nu het “Orig size” veld in de BackwardAnt). Het resulterende getal wordt opgeteld bij de huidige feromoonwaarde geassocieerd met de next hop – destination intersectie in de feromoontabel.

```
float constant = 0.2;                    // To be configurable
float origsize = 8;                      // Normally read from a rants packet
uint32_t previoushop = 0xA000001         // “10.0.0.1”
                                         // Normally read from a rants packet
uint32_t source = 0xA000002              // “10.0.0.2”
                                         // Normally read from a rants packet
float pheromone = get_val(previoushop, source);
pheromone += constant / origsize;
set_val(previoushop, source, pheromone);
```

De feromoonconstante gebruikt in de feromoonberekening moet configurabel zijn.

5.3. Mogelijke uitbreidingen

Tijdens het brainstormen en uitproberen zijn nog wat andere variabelen bedacht, die nog geen weg hebben gevonden naar het huidige bitpatroon. Een aantal daarvan zijn echter toch het vermelden waard, omdat ze in een latere protocol versie gebruikt kunnen worden.

Mogelijke toekomstige functies/velden:

- Protocol versie / opties veld

Mogelijk is het handig om een extra versie / opties veld op te nemen. Het gezegde gaat dat software in ontwikkeling blijft tot de laatste gebruiker dood is; dat kan wel eens lang duren. Tot die tijd zullen er altijd uitbreidingen komen, die mogelijk incompatibiliteit veroorzaken. ID-TYPE voorziet hier al deels in, maar een extra veld is alsnog het overwegen waard.

(Een tegenargument hierin is dat elke stap naar “turing completion” een extra mogelijke exploit vector brengt.)

- IPv6 ondersteuning

Alle adressen in het rants protocol zijn nu fixed 32bit lang, ofwel slechts geschikt voor IPv4. IPv6 staat inmiddels voor de deur, wat ondersteuning daarvoor een zo goed als must-have maakt voor toekomstige versies.

- Verwijderen next hops / destinations uit de feromoontabel

Het grootste gat in het ontwerp van rants is dat het alleen maar dingen bijleert. Routes die irrelevant geworden zijn zullen uiteindelijk tot een feromoonniveau van (zo goed als) nul dalen in de feromoontabel. Maar er zijn nog geen regels of criteria opgesteld voor verwijdering uit deze datastructuur. Ofwel in de huidige specificatie is nog nergens opgenomen onder welke omstandigheden een next hop of destination uit de feromoontabel verwijderd wordt.

- Netmask informatie

Nergens in het bitpatroon is iets opgenomen over een netmask of andere variabelen nodig om routering naar niet-rants nodes op een subnets mogelijk te maken. Het protocol heeft zich tot nu toe bijna exclusief gefocust op nodes gezamenlijk een systeem laten opbouwen. Om ze ook nuttig te maken als routers voor gebruikersverkeer zijn nog zaken als VLSM nodig in het protocol.

- Router ID

Er moet nog een expliciete beslissing genomen worden over met welke variabele routers zich identificeren. Dit kan zijn het IP adres of één van de IP adressen (eventueel daar ook nog een verplicht keuze algoritme) of een ander (gegenereerd) ID of nummer.

- AS nummer

Ondersteuning voor onderscheid in verschillende Autonomous Systems kan handig zijn voor systeembeheerders om regionen in het netwerk aan te brengen, elk met eigen rants instellingen.

- Crypto/authenticatie

Om tuig uit het systeem te weren moet de uiteindelijke versie een methode hebben om van andere nodes de authenticiteit te kunnen bepalen. Liefst tezamen met een laag encryptie zodat interessante details over het netwerk moeilijker op te vangen zijn.

6. Realiseren proof-of-concept

Het schrijven van de proof-of-concept is de laatste stap voor het kunnen testen van ant-based algoritmen op netwerkgebied. Dit zou het beoogde resultaat en het behalen van de doelstelling binnen handbereik brengen. Het schrijven van de proof-of-concept is terug te vinden in de fasen 'software architectuur ontwikkelen'[27] en het implementeren gedeelte van 'implementeren & testen'.

De origineel beoogde project aanpak bestond uit een periode experimenteren, gevolgd door consolidatie en een test. Vanaf de requirements fase is het consolideren ingevallen. Deze aanpak krijgt echter vanaf de software architectuur fase een iets andere twist.

Gedurende de analyse & prototyping fase is er gewerkt met het XORP framework. Dit heeft een boel ervaring opgeleverd, waarvan een hoop helaas negatief. Werken met XORP heeft een hoop tijd gekost en simpelweg niet opgeleverd wat er van verwacht werd.[28] Dit motiveert mij XORP in te ruilen voor iets anders. Na deze verandering is er dus iets nieuws waar op aangepast moet worden. Dit aanpassingsproces vraagt zo doende wederom wat experimenteren. De software architectuur fase is dus niet slechts consolideren, er komt ook weer een stuk ontdekking bij kijken. Een kleine afwijking van het originele plan.

Ondanks dat er een ontdekkingselement terug is, is consolideren nog steeds de hoofdmoot in deze fase. Zo is er in de analyse & prototyping fase een niet onaanzienlijk geheel code geschreven wat goed herbruikbaar is. Deze code implementeert tevens het protocol wat in de requirements fase geformaliseerd is.

Het afschrijven van XORP houdt in dat er een nieuwe basis gekozen moet worden. Hierdoor moet een nieuwe (uiteindelijke) keuze worden gemaakt tussen de eerder geanalyseerde frameworks en daemons[29]. De in de analyse & prototyping fase geselecteerde daemon (XORP) wordt in elk geval niet aangehouden als basis voor ontwikkeling.

Dit keer echter wil ik zo min mogelijk beperkt worden door de opzet van de basis. Om dit te bereiken wil ik het ditmaal over een andere boeg gooien: In plaats van een basis kiezen, vanaf de grond af aan zelf bouwen. Bepaalde eerder geanalyseerde frameworks en daemons kunnen hier als voorbeeld in dienen. Er is hierdoor niet meer een enkele keuze van toepassing.

Ook de taal waarin het programma geschreven wordt wil ik veranderen. C++ wil ik verruilen voor C. De taal C heeft een zeer gelijkende naam als C++, maar is conceptueel voor een groot deel anders. C++ heeft een hoop elementen qua notatie afgekeken van C. Echter is niet alle C++ code ook valide in C en andersom.

De reden voor het verruilen van C++ voor C is puur ingegeven op basis van persoonlijke voorkeur.

Het doel is daarom nu om iets in C te bouwen wat gelijkenis heeft aan de XORP module. Het liefst wordt daarbij zoveel code als mogelijk direct hergebruikt dan wel geport. De kern van het ant-based programma is daarmee al aanwezig, waarna de functionaliteit waar het mee moet worden ingepakt afgekeken kan worden van andere projecten.

6.1. Heranalyse beschikbare routing software

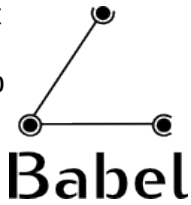
Met de andere projecten wordt bedoeld de eerder onderzochte frameworks en daemons. Deze lijst van projecten is in de analyse & prototyping fase al eens onder de loep genomen. De lijst ga ik nogmaals langs, ditmaal met het oog op hoe goed ze voldoen aan gebruik in de nieuwe aanpak.

27 Project documentatie: Software architectuur rapport, *Bijlage G*
28 Zie §4.5, sectie 'Opgeleverde code'
29 Zie §4.2

De volledige heranalyse is te lezen in het software architectuur rapport[30].

Babel

Babel is net als rants een klein en experimenteel protocol. Dit maakt de code erg gericht op waarvoor het ontworpen is, wat het ongeschikt maakt om op verder te bouwen. De routines die er in gebruikt worden echter kan een hoop van geleerd worden. Zo zal er op verscheidene momenten communicatie naar het OS plaatsvinden of iets met netwerkverkeer gebeuren. Hoe dit gedaan kan worden is daarom prima af te kijken van Babel.



B.A.T.M.A.N.

Batman-adv, de layer-2 incarnatie van het Batman protocol, is de enige nog in gebruik zijnde batman. Batman-adv creëert virtual interfaces en doet daarna frame forwarding. Het doet dus helemaal niets met IP routing. Daarom is er waarschijnlijk weinig tot niets te leren van batman.

BIRD

BIRD is een zeer veelzijdig programma met ondersteuning voor meerdere protocollen. Dit betekent dat er extra abstracties in het programma zitten. Was dit voorheen (in de a&p fase) een pré, nu is het een negatief punt. Waar nu naar gezocht wordt is niet meer code waar het makkelijkste functionaliteit aan toe te voegen is. Wat nu belangrijk is, is hoe leerzaam de code is en hoe makkelijk die is aan te passen c.q. over te nemen.

OpenBGPD & OpenOSPFD

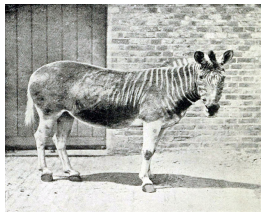
OpenBGPD en OpenOSPFD komen uit de stal van het OpenBSD project. Waar deze mensen om bekend staan is dat ze erg OpenBSD-minded zijn. Heel erg OpenBSD-minded. Dit maakt deze twee daemons praktisch OpenBSD-only. Gezien momenteel Linux als onderliggend OS gebruikt wordt is er niet erg veel te leren van deze twee daemons op dat gebied.

GNU Zebra

Zebra is een collectie deamons die met z'n alle meerdere protocollen ondersteunen. Ze communiceren middels een binary protocol via Unix sockets of TCP/IP verbindingen voor het delen en updaten van informatie. Met andere woorden: Heel veel ingewikkelder dan een klein programma voor een enkel protocol. Hier komt bij dat Zebra al sinds 2005 niet meer (publiek) in ontwikkeling is.

Quagga

Quagga is een GNU Zebra fork. Dit betekent dat Quagga heel veel met Zebra deelt qua ontwerp en code. Dit betekent echter ook dat de bij Zebra genoemde nadelen ook op Quagga van toepassing zijn: Quagga is veel ingewikkelder van opzet dan een programma wat rechttoe rechtaan een enkel protocol implementeert.



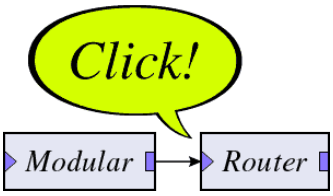
XORP

XORP is in de opening van dit hoofdstuk reeds genoemd als niet geschikt. In de analyse & prototyping fase opgedane ervaring heeft duidelijk gemaakt dat XORP niet de te nemen weg is.

30 Project documentatie: Software architectuur rapport, H3, *Bijlage G*

Click router

De Click router is eigenlijk niet echt een routing daemon, maar meer een modulaire toolkit voor packet handling. Onderdelen of functionaliteiten die bij het verwerken van IP packets nodig zijn kunnen als "blokjes" aan elkaar geschakeld worden. Hiermee kan dan een geheel eigen packet handling mechanisme gemaakt worden.



De Click router is daarom meer een alternatief voor de traditionele Unix kernel forwarding dan een routing daemon. Wanneer er ooit de behoefte komt aan een geheel eigen IP stack maken dan die voor een groot deel met de Click router elementen gebouwd worden. Gebruik maken van de Click router is echter nog even een stap verder dan de Unix kernel forwarding gebruiken. Voordat de Click router goed begrepen wordt kan er zo evenveel tijd verstreken zijn als nodig was voor XORP.

6.2. Herkiezen routing framework

Het is ironisch om te zien dat het project wat in de analyse & prototyping fase is weggezet als ongeschikt, wegens het kiezen van een nieuwe koers in deze fase de meeste aandacht trekt. Babel is van alle projecten het meest verwant aan wat in deze fase als doel is gesteld.

Nu er geen framework maar een simpele daemon gezocht wordt is Babel een interessant studieobject geworden. Niet vanwege het protocol of perse vanwege de datastructuren, maar vanwege wat geleerd kan worden over hoe de feitelijke protocol code is ingepakt in bijkomende functionaliteiten.

Een terugkomer is BIRD, die net als in de analyse & prototyping fase op een soort tweede plek eindigt. BIRD is een stuk algemener van opzet dan Babel. Dit maakt het ingewikkelder. Maar BIRD is ook een stuk langer in ontwikkeling. Wat betreft geheugenstructuren kan BIRD daarom wellicht een goed voorbeeld zijn.

Click komt wederom terug als iets voor op de lange baan. Het is wegens tijdsbeperkingen niet direct interessant, maar op de langere termijn een experiment waard.

De keuze komt neer op het volgende: De kern van het rants protocol staat nu min of meer. Het protocol is opgesteld en er is code die het implementeert. Wat nu nodig is is de code er omheen ontwikkelen en het als een geheel afmaken. Hierin kunnen Babel en BIRD een goed voorbeeld zijn. Beide zijn geschreven in C en hebben Linux primair platform. Click is iets voor later.

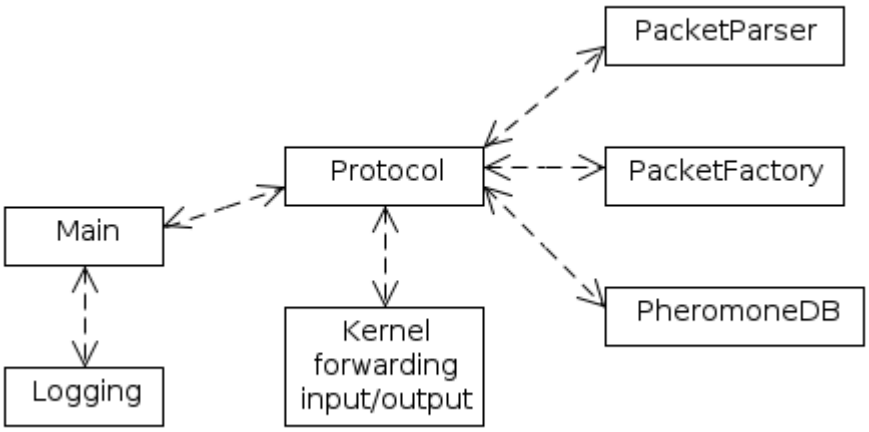
6.3. Programmaontwerp

Bij aanvang van deze fase is besloten het project voort te zetten in de taal C, in plaats van het door XORP gebruikte C++. Dit betekent dat de tot nu toe geproduceerde code vertaald zal moeten worden.

Er is een groot verschil tussen C en C++. De taal C++ is namelijk wat heet een object georiënteerde taal, terwijl C dit niet is. Dit houdt concreet in dat de klassen die weergegeven zijn in het analyse & prototyping rapport niet 1:1 over te nemen zijn in het nieuwe C project. Zowel op code niveau als ontwerp niveau is er een verschil in paradigma wat gebruikt wordt.

Ondanks dat C++ veel explicietere scheiding van code handhaaft dan C is het verstandig om ook vooraf aan programmeren in C een schets te maken van functionaliteiten. Een deel van het klassendiagram uit de analyse & prototyping fase kan dus hergebruikt worden. Echter zal het diagram niet expliciet terug te vinden zijn in de C code.

Het volgende diagram is dan ook vooral een schets om de functionaliteit duidelijk te maken. Aan de hand hiervan wordt de code vorm gegeven worden in bijvoorbeeld logische functienamen.



Een aantal elementen in dit figuur zijn terug te vinden in het klassediagram-achtige figuur te zien in §4.5, sectie implementatie. Dit is, zoals eerder gesteld, omdat zoveel mogelijk werk als mogelijk wordt overgenomen uit de XORP module. In dit geval echter is deze structuur niet 1:1 terug te vinden in de code, omdat de taal C geen klassen kent. De geboden structuur zal door middel van naamgeving aan variabelen en functies zichtbaar gemaakt moeten worden in de te produceren code.

6.4. Implementatie

Van de proof-of-concept is nog geen code opgeleverd. Na het ontwerpen van de architectuur ben ik aan dit document begonnen, waarna het alle resterende tijd heeft opgeëist. Iets wat ik erg jammer vind, gezien ik graag een werkend product had opgeleverd.

Dat ik nog geen werkzaamheden aan de uiteindelijke implementatie heb uitgevoerd betekent niet dat ik geen ideeën en plannen heb hiervoor. Ik heb een stappenplan opgesteld om tot een werkende proof-of-concept te komen:

1. Door gebruik te maken van zaken die ik kan leren uit Babel en BIRD is het mogelijk een basis te ontwikkelen waarop de eerder ontwikkelde code zijn werk kan doen.
2. De eerder ontwikkelde code zou zonder teveel problemen om te zetten moeten zijn naar C, gezien C++ een zeer gelijkende syntax heeft.
3. Hierna is het vooral zorgen dat alles op elkaar aansluit en aan te sturen is. Er is geen CLI faciliteit meer zoals XORP die bood, waardoor er iets bedacht moet worden om het alsnog aan te kunnen sturen.
4. Eenmaal alles naar behoren werkt, kan binnen het programma eerste uitwisseling van data plaatsvinden, om te zien of het bitpatroon systeem daadwerkelijk werkt. Hierbij wordt een set gegevens omgezet naar het bitpatroon en direct weer terug. Als de gegevens na deze twee omzettingen nog hetzelfde zijn, werkt het parsen van data naar behoren.
5. Als dit voltooid is, kan voor het eerst een poging gedaan worden om data te versturen tussen twee nodes. Wanneer deze data naar behoren geinterpreteerd en opgeslagen wordt, is de node bijna compleet.
6. Als laatste moet de logica voor het sturen en verwerken van mieren ingeschakeld worden.

Daarna zou een test op zijn plaats zijn. Door middel van een simulatieomgeving kan dan gekeken worden of het rants protocol en de implementatie daarvan in staat is om routes te vinden in een netwerk.

Dit alles zal zich echter na het afronden van mijn afstuderen voltrekken. Er is niet genoeg tijd meer om dit stappenplan uit te voeren binnen de tijdsbeperkingen.

7. Evaluatie

Dit hoofdstuk is de laatste in dit document. In dit hoofdstuk wordt als afsluiter op het afstudeerproject terug gekeken en geëvalueerd hoe dit verlopen is. Zowel het product als het proces komen hierin aan bod.

7.1. Productevaluatie

Mijn mening over de producten die ik opgeleverd heb verschilt. Niet alle hebben het niveau gehaald wat ik graag had gezien, terwijl ik op andere best trots ben. Ik ga ze nu stuk voor stuk af.

XORP module

De XORP module heeft niet gebracht waar ik op gehoopt had. De module is namelijk niet af gemaakt. Zowel ik als mijn collega's hoopten mijn protocol al in actie te zien. Vroeg in de ontwikkeling al een simulatie kunnen uitvoeren voelt niet alleen vroegtijdig al een aantal concepten aan de tand, het is ook motiverend om eigen werk in actie te zien.

Tevens is het stoppen met XORP een koerswijziging die erg aanvoelt als een verlies. Er is immers een significante hoeveelheid tijd gaan zitten in het leren omgaan met XORP, dit gooi ik met deze keuze in feite weg. Maar de keuze is terecht geweest; XORP heeft niet gebracht wat er van verwacht werd. Integendeel. In plaats van dat XORP het project heeft ondersteund heeft het eerder het ontwikkelproces opgehouden.

Ondanks dat XORP verlaten is heeft het werken ermee toch ook een goede kant gehad. De structuur waarin XORP modules ontworpen en geschreven worden heeft een zekere invloed gehad op de manier waarop de rants code vorm heeft gekregen. Over de rants code ben ik, ondanks de slechte herinneringen aan XORP, wel tevreden. Het is op dit punt nog geen werkend programma, maar er is een goede basis van waaruit doorgewerkt kan worden. Zo is het opbouwen en parsen van data geïmplementeerd alsook de sequentie die het programma moet doorlopen.

De naar mijn mening goede logica van het soon-to-be programma is dus mede te danken aan wat het XORP project te bieden had. De structuur van het programma, de opdeling van functionaliteit, hoe daar doorheen gelopen wordt en hoe de stromen data tussen onderdelen lopen is allemaal bedacht tijdens het programmeren van de XORP module die nooit afgemaakt zal worden.

De XORP module heeft niet gegeven wat ik wilde maar meer als voldoende voor het volgende product. Daarmee heeft het toch z'n doel behaald. De functie van het analyseren & prototypen was ideeën opdoen voor het protocol en de implementatie, wat het werken met XORP toch gebracht heeft.

Ant-based routing protocol

Het routing protocol is één van de hoofdproducten van het project en gelukkig ook één waar ik gelukkig mee ben. Ik ben zelf erg tevreden over het protocol. De beginselen van het protocol zijn tijdens het werken aan de XORP module bedacht. Deze beginselen en ideeën zijn naar mijn mening zeer goed gekristalliseerd in rants, het ant-based routing protocol.

Wel is het protocol nog verre van af, gezien het nog een aantal benodigde functionaliteiten oningevuld laat. Ondanks dit gegeven is het rants protocol zeker iets wat getest kan worden en verder kan groeien tot een product. En, wellicht ooit, actief wordt op netwerken waar ik nog nooit van gehoord heb.

Rants heeft nog een lange weg te gaan, maar wat er nu is is in mijn ogen zeker een goede start. De kernprincipes van ant-based algoritmen zitten er in en de functionaliteit benodigd voor het zijn van een routing protocol is aanwezig. Daarmee heeft ook het protocol z'n doel behaald. De functie van de requirements bepalen was het opstellen van een protocol (concept) waar een implementatie (proof) bij geschreven kan worden. Het concept is genoeg uitgewerkt om deze proof te kunnen schrijven.

Proof-of-concept

Over de proof-of-concept heb ik wat evaluatie betreft een neutraal gevoel. De proof-of-concept is net als de XORP module niet afgemaakt. In tegenstelling tot de eerder besproken XORP module is hier geen technische reden voor. De reden dat de proof-of-concept niet geschreven is, is de tijdsbeperking. Andere prioriteiten (zoals het schrijven van dit verslag) hebben het uitvoeren van deze taak helaas verhinderd. Helaas omdat ik de proof-of-concept graag had geschreven binnen de gestelde tijd.

Omdat ik nog geen resultaat behaald heb met een implementatie van de implementatie & testfase kan ik er ook niet echt een oordeel over vellen. Waar ik wel een oordeel over kan vellen is het ontwerp voor deze proof-of-concept. Voor de uiteindelijke implementatie is namelijk een klein ontwerp gemaakt. Dit ontwerp is geïnspireerd op het ontwerp wat bedacht is in de analyse & prototyping fase. Tevens is er een nieuwe richting uitgezet voor het voltooien van de proof-of-concept.

De nieuw uitgezette richting omvat naast het net genoemde ontwerp een andere aanpak. In plaats van een enkele basis te nemen wil ik from scratch beginnen en andere daemons als voorbeeld gebruiken. Ik heb er vertrouwen in dat deze aanpak mij bij mijn gestelde doel gaat brengen.

De proof-of-concept heeft z'n doel niet behaald. De tijdsbeperking stond het niet toe. Geen implementatie, geen test. Hierdoor is ook de doelstelling van het afstudeerproject niet behaald: Ant-based algoritmen zijn niet getest op netwerkgebied.

7.2. Procesevaluatie

Het proces behorende bij dit project was een unieke. Het project is niet uitgevoerd volgens een gestandaardiseerde methode, maar een aanpak die als onderdeel van dit project is opgesteld. Dit maakt de evaluatie ervan ook temeer interessant; was de aanpak (methode) succesvol?

Aanpak / Methode

Naar mijn menig is de aanpak geslaagd. Het niet behalen van het gestelde doel is het resultaat van een ongelukkige keuze (XORP) en een tekort aan tijd. De aanpak heeft het project goed gediend en is hier dan ook niet verantwoordelijk voor.

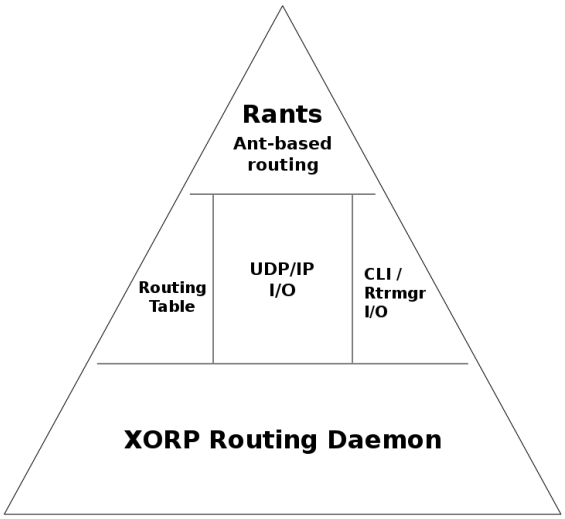
Dit legt een groot deel van het falen dan ook bij XORP. XORP bleek een hele hoge learning curve te hebben als het aankomt op programmeren ervoor. De analyse & prototyping fase was origineel begroot op 51 werkdagen. Na verlenging van het project is dit 80 dagen geworden.

XORP vroeg veel aandacht; het is simpelweg een ingewikkeld stuk software. Dit in combinatie met de sterk wisselende kwaliteit van de documentatie heeft ervoor gezorgd dat het ontwikkelproces zich teveel op XORP gericht heeft. Hier heb ik spijt van. De focus op XORP heeft mijn aandacht bij de feitelijke ant-based functionaliteit weggehouden. Dit betekent dat de implementatie van rants uitgebreider en beter had gekund.

De reden dat het proces zo verlopen is, is de volgende: Gedurende het prototypen was ik er van overtuigd dat “van onderaf omhoog bouwen” de beste methode zou zijn. De basis is XORP, het einddoel een stuk ant-based code. De basis legt bepaalde beperkingen op aan hoe de rest erboven er uit kan zien. Hierdoor leek het mij het verstandigst eerst deze basisvormen uit te vinden dan wel neer te leggen alvorens naar andere zaken te kijken.

Hierdoor heb ik mijn einddoel uit het oog verloren. Het is niet leuk om te concluderen, maar desalniettemin wel waar: Ik heb enigszins geleden aan tunnelvisie.

Zie de volgende diagramustratie:



Mijn focus op integratie en problemen oplossen gerelateerd aan XORP heeft heel veel tijd weg genomen bij zaken die hoger in de piramide liggen. De ant-based code heb ik daarmee tekort gedaan.

Planning

Een andere fout die ik begaan heb is het niet opnemen van het afstudeerverslag in de planning. De reden dat dit niet gebeurd is op zich verklaarbaar: Alle documentatie valt onder de op te leveren producten van een fase en elke fase heeft in de planning een tijdsbudget gekregen.

Het afstudeerverslag behoort toe aan het gehele project en is daarom nooit onder een fase ingedeeld. De inleverdatum staat wel bij de belangrijke data, maar er is geen apart tijdsbestek voor ingeroosterd. De bedoeling was dan ook dat het afstudeerverslag tussendoor geschreven zou worden. Een grove miscalculatie, gezien het ruim een taak op zichzelf is.

Vanaf week 22 heb ik bijna al mijn tijd aan dit verslag besteed. Hiermee is het geheel in de plaats van implementatie en testen gekomen. Wat ik bij een volgend project daarom zeker dien te doen is een beter oog hebben voor bijkomende activiteiten en die expliciet opnemen in de planning.

Buiten de fout met het afstudeerverslag en het uitlopen van de prototyping fase is de planning accuraat geweest. Dat wil zeggen dat in de laatste versie van het plan van aanpak (v0.6, opgesteld nadat er besloten is het afstuderen met 10 weken te verlengen) de planning om en nabij gelijk is aan hoe de uitvoer is geweest.

De tekortkomingen lagen daarmee vooral in de uitvoer van het project en niet in de zelf bedachte aanpak (“methode”) die gehanteerd is. De iteratieve fase gevolgd door de waterval heeft het project goed ondersteund.

7.3. Competenties

De voor dit afstudeerproject gespecificeerde competenties zijn in hun volledigheid te vinden in de bijlagen bundel[31]. Daarin wordt uiteengezet hoe aan de competenties invulling is gegeven en of deze voldoende behaald zijn.

Hier volgt nu kort een lijst van deze competenties, met in enkele regels samengevat hoe ze in het project zijn terug te vinden.

G1 – Praktische aspecten hanteren in (internationale) projecten

Aan deze competentie heb ik invulling gegeven door het maken van een risico analyse.

- Fase: Plan maken.
- Document: Plan van aanpak.

A1 – Analyseren van het probleemdomein

Aan deze competentie is invulling gegeven middels drie activiteiten: Analyseren van het T7 onderzoeksrapport, analyseren van bestaande routing frameworks en beschrijven van het prototype.

- Fase: Analyseren & prototypen.
- Document: Analyse & prototyping rapport.

A3 – Achterhalen van behoeften van belanghebbenden

Behoeften van belanghebbenden zijn normaal gesproken de requirements. De requirements komen ditmaal van een enkele belanghebbende: Ik zelf. De requirements zijn eigenlijk een enkele requirement: Het programma moet voldoen aan een bepaald protocol.

- Fase: Requirements opstellen.
- Document: Requirements document.

C10 – Ontwerpen van een systeemarchitectuur

Voor het schrijven van een implementatie voor het protocol is een ontwerp benodigd. Deze bestaat uit een beschrijving van een systeemarchitectuur met behulp van een diagram met uitleg.

- Fase: Software architectuur ontwikkelen.
- Document: Software architectuur rapport.

D17 – Testen van software systemen

Een systeemtest. Op dit moment wordt getest of het programma in staat is om routes te vinden in een netwerk. Het project heeft dit punt niet bereikt. Deze competentie heeft om die reden een zeer geringe invulling gekregen: Alleen eisen waarop onder andere getest moet worden zijn gespecificeerd: Het protocol.

- Fase: Implementeren & testen.
- Document: *Requirements document*. (Testrapport is niet opgeleverd.)

31 Project documentatie: Competenties verantwoording, *Bijlage I*

7.4. Slot

Als afstudeerproject heeft het ant-based routing protocol mijn inzicht en kunnen flink op de proef gesteld. Ik heb helaas niet al mijn doelstellingen behaald maar ben desondanks toch tevreden. Ik heb met dit project hoog ingezet op zowel organisatorisch vlak als op technisch vlak. Ik ben wateren ingevaren waar ik nog nooit geweest was. Dit maakt dat dit afstudeerproject niet alleen een test is geweest waarop ik mij heb moeten bewijzen maar ook een leerweg.

Het project is nog niet af, maar ik ben gemotiveerd om er later mee door te gaan. Alles wat nu groot is begon ooit klein. Wie weet waar rants naar kan uitgroeien.

Begrippenlijst

- Framework

Een framework is een raamwerk of basis waar functionaliteit bovenop geprogrammeerd kan worden. Een framework is doorgaans zo ingericht dat het op zichzelf weinig doet maar functionaliteiten biedt aan een programmeur. In combinatie met een programma wat ermee geschreven is vervult het dan een taak.

- Daemon

Een "daemon" is een Unix term voor een programma wat z'n werk zonder directe invoer van de gebruiker doet. Veelal lezen deze programma's een bestand met configuratie in waarna ze zonder verbonden te zijn met een terminal hun werk doen.

- NAT

NAT is een netwerk-gerelateerde term en staat voor Network Address Translation. Dit is een technologie om een netwerk met meerdere adressen aan een ander netwerk te presenteren als een enkel adres.

- VMware

VMware is het in dit project gebruikte virtualisatie pakket. Virtualisatie is een veelgebruikte term voor technologieën die het mogelijk maken om een enkel fysiek systeem op te delen in meerdere logische systemen.

- ESXi

De versie van VMware gebruikt in dit ontwikkelproces. ESX zou staan voor "Elastic Sky X". De toevoeging 'i' is gekomen nadat er een nieuwe gratis te downloaden productreeks van is gekomen.

- VM

VM staat voor Virtual Machine. Een Virtual Machine is een logisch systeem binnen het fysieke systeem.

- OS

OS staat voor Operating System. Een operating system is een programma of set programma's die het mogelijk maakt om processortijd, werkgeheugen, invoer/uitvoer en andere hardware te delen tussen meerdere programma's.

- Debian

Debian is een in dit project gebruikt OS. Debian is een Linux distributie en is ingezet op de desktop en als OS op de ontwikkel VM's.

- pfSense

pfSense is een in dit project gebruikt OS. pfSense is een kleine FreeBSD distributie die is gericht op routering en firewalling. Het pfSense OS is in dit project ingezet als NAT tussen het door VMware opgezette virtuele ontwikkelnetwerk en het echte bedrijfsnetwerk.

- SCons

SCons is een in python geschreven build system. Het build system is een term gebruikt voor een combinatie van OS, compiler en tools die het compileren of “builden” van een programma overzien. Hiermee vervult SCons een gelijkende functie als GNU make & autoconf.

- IDE

IDE staat voor Integrated Development Environment. IDE's zijn doorgaans de "tekstverwerkers" van computer code. Ze maken de geschreven regels code beter leesbaar door bepaalde woorden wat kleur te geven en doen onderwijl ook wat spellingscontrole.

- Eclipse

Eclipse is de IDE gebruikt bij de ontwikkeling in dit project.

- CDT

CDT staat voor C/C++ Developers Tool. CDT is een plugin (aanvulling) op Eclipse waardoor Eclipse ook "spellingscontrole" voor C en C++ kan doen.

- Desktop/Server

De termen server, client en desktop zijn soms wat verwarrend omdat ze in een andere context iets anders in kunnen houden. Wanneer de woorden server en desktop met elkaar gebruikt worden gaat het om hardware: Een desktop is doorgaans een lichte computer die bij een gebruiker op het bureau staat. Een server is doorgaans een zware computer die in een aparte ruimte is opgesteld. Een desktop is gericht op directe interactie. Een server heeft doorgaans nauwelijks grafische uitvoer.

- Client/Server

Wanneer de woorden client en server met elkaar gebruikt worden gaat het doorgaans over software. Een server programma biedt een of meerdere functionaliteiten waar client software gebruik van kan maken. Dit concept staat los van de classificatie "server" die aan een computer gegeven kan worden. Er zijn bijvoorbeeld situaties waarin server hardware een client programma uitvoert en de desktop hardware een server programma uitvoert. Server programma's zijn doorgaans tevens daemons.

- X11

X11 of X11R6 is het protocol gebruikt op bijna alle Unix achtige OS'en voor communicatie tussen programma's en het systeem wat de grafische weergave regelt. Dit komt voort uit het op Unix veelgebruikte client-server model. Dit maakt het dan ook mogelijk om het programma van de gebruiker op een zware machine uit te voeren en de invoer/uitvoer daarvan via een andere computer te laten lopen.

- SSH

SSH staat voor Secure SHell. SSH maakt het mogelijk een CLI op afstand te benaderen. Hierin lijkt het veel op het oude telnet, waarbij het belangrijkste verschil is dat SSH de verbinding van encryptie voorziet.

- T7

T7 of TI7 is een lesblok wat onderdeel is van het Technische Informatica lesaanbod op de Haagse Hogeschool. T7 bestaat uit het doen van een onderzoek op een dusdanig niveau dat het feitelijk een voorbereiding is op het afstuderen. Het onderzoek uit T7 is de uitgangspositie geweest voor deze afstudeeropdracht.

- Rants

Rants staat voor Running ants en is de huidige naam van het bij dit project ontwikkelde protocol.