

transnationale Universiteit Limburg
School voor Informatietechnologie

NETWERKSIMULATIE MET BEHULP VAN NS-3

Proefschrift voorgelegd tot het behalen van de graad van bachelor in de
informatica, traject ICT aan de transnationale Universiteit Limburg

Bob van der Vleuten

Promotor: Prof. Dr. Wim Lamotte

Co-promotor: Dr. Peter Quax

Begeleider: Takis Issaris

3 september 2010

Academiejaar 2009 - 2010
Universiteit Hasselt

Abstract

In dit eindwerk wordt een poging gedaan tot het inventariseren van de mogelijkheden van de ns-3 netwerksimulator. Er wordt een korte beschrijving gedaan van de manier waarop men simuleert met ns-3 met daarbij een drietal voorbeelden en hun resultaten. Tenslotte wordt er een bespreking gedaan van de voor- en nadelen van ns-3 om tot een besluit te komen.

In het eerste deel van het eindwerk wordt uitgelegd wat ns-3 precies is. Er wordt gekeken naar de opbouw van ns-3 aan de hand van de modules en concepten die het omvat. Er wordt tevens een uitgebreide uiteenzetting gedaan van de mogelijkheden van ns-3, gaande van de verschillende modellen tot meer exotische zaken zoals het maken van een koppeling tussen de reële wereld en de simulator.

In het tweede deel wordt ten eerste getracht de algemene opbouw van een simulatie met ns-3 in code uit te leggen alsook het gebruik en de uitvoering ervan. Hiertoe wordt een basisstructuur uiteengezet, daar meer specifieke zaken teveel afhangen van de noden van de simulatie. Ten tweede worden er drie simulaties uitgelegd die werden gemaakt om de meer specifieke mogelijkheden en de verschillende modellen van ns-3 te tonen. De derde van deze simulaties maakt gebruik van een koppeling tussen reële hosts in Linux containers en een gesimuleerd netwerk in ns-3. Voor die simulatie werden er een aantal scenario's uitgewerkt die eveneens in werkelijkheid werden uitgevoerd om zo de resultaten te kunnen vergelijken.

Tenslotte wordt er een bespreking gemaakt van de ondervonden moeilijkheden tijdens het werken aan dit eindwerk en worden ook de voor- en nadelen die het werken met ns-3 met zich meebrengen uiteengezet. Op basis daarvan alsook van andere literatuur wordt er een besluit gegeven over: de simulatie mogelijkheden, de bijkomende mogelijkheden en de eigen ervaringen alsook een algemeen beknopt besluit.

Woord vooraf

Ik heb voor dit onderwerp gekozen omdat ik al langer een interesse koester voor computernetwerken en deze bachelorproef de ideale gelegenheid was om mij hier verder in te verdiepen. Hoewel mijn keuze voor een masteropleiding HCI al vast lag, leek het mij een goed idee van ook het aspect netwerken prominenter aan bod te laten komen in mijn bacheloropleiding Informatica-ICT.

Ik verwachtte van dit eindwerk dat het mij een beter inzicht zou geven in zowel computernetwerken als de simulatie ervan. Een bijkomende verwachting was het ontdekken van de werking van een opensource project, meer bepaald de manier waarop communicatie en ontwikkeling plaatsvindt. Ns-3 is namelijk een opensource project dat door meerdere organisaties en individuen gesteund en ontwikkeld wordt.

Beide verwachting werden min of meer ingelost. Ik heb een aantal zaken bijgeleerd over computernetwerken, geleerd hoe de ns-3 simulator werkt en waarop deze gebaseerd is en heb tenslotte een inzicht gekregen in de ontwikkeling van en communicatie rond het ns-3 project. Eveneens heb ik ten gevolge van deze bachelorproef ervaring kunnen opdoen met niet alledaagse tools in Linux alsook het opzetten van een testnetwerk in realiteit. Ik ben er dan ook van overtuigd dat dit werk een onmiskenbare bijdrage levert aan mijn opleiding tot informaticus.

Tenslotte wil ik een aantal personen bedanken om hun (in)directe bijdrage tot dit eindwerk. Ten eerste mijn promotor, co-promotor en begeleider om hun advies, deskundige uitleg en een productieve samenwerking. Ook wil ik mijn co-promotor bedanken voor het ter beschikking stellen van de demo-ruimte en hardware voor het test-netwerk, een belangrijk onderdeel van dit werk. Echter is ook morele steun belangrijk en daarvoor kan ik mijn vriendin en mijn ouders niet genoeg bedanken, hun steun en luisterend oor vormden de belangrijkste indirecte bijdrage aan dit eindwerk. Tenslotte wil ik ook de personen die reageerden op mijn berichten in de gebruikersgroep van ns-3 bedanken voor hun uitleg en hulp bij het oplossen van problemen.

Inhoudsopgave

1	Inleiding tot ns-3	5
1.1	Wat is ns-3?	5
1.2	Concepten in ns-3	6
1.2.1	Nodes en nodecontainers	6
1.2.2	Devices en devicecontainers	6
1.2.3	Applicaties	7
1.2.4	Helpers	7
1.3	Modules	8
1.3.1	Simulator module	8
1.3.2	Core module	9
1.3.3	Common module	9
1.3.4	Node module	10
1.3.5	Devices module	10
1.3.6	Internet stack module	11
1.3.7	Routing module	11
1.3.8	Helpers module	11
1.3.9	Applications module	11
1.3.10	Mobility module	12
1.3.11	Contrib module	12
1.3.12	Andere modules	13
1.4	Wat ns-3 te bieden heeft	13
1.4.1	Protocollen	13
1.4.2	Modellen	13
1.4.3	Schaalbaarheid	14
1.4.4	Multi-laag eigenschappen	14
1.4.5	Integratie in de reële wereld	14
1.4.6	Visualisatie	17
1.4.7	Gedistribueerde simulatie met MPI	17
1.4.8	Routing	18
1.4.9	Performantie	19
1.4.10	Willekeurige verdelingen	20
2	Simuleren met ns-3	21
2.1	Ns-3 builden en testen	21
2.2	Simulaties 'runnen'	22
2.3	Voorbeelden van simulaties	22
2.4	Opbouw van een simulatie	22
2.4.1	Preambule	23
2.4.2	Topologie en adressen	24

2.4.3	Applicaties	25
2.4.4	Simulator, Run!	26
2.5	Simulaties en resultaten	27
2.5.1	Eerste simulatie	27
2.5.2	Tweede simulatie	32
2.5.3	Derde simulatie	37
3	Bespreking	50
3.1	Ondervonden moeilijkheden	50
3.1.1	TapBridge en WiFi devices	50
3.1.2	Errormodel en WiFi devices	52
3.1.3	Gebrek aan documentatie	53
3.2	Voordelen van ns-3	53
3.2.1	Performantie	53
3.2.2	Hybride simulaties	54
3.2.3	Commandline parameters	54
3.2.4	Hoeveelheid aan modellen	54
3.2.5	Veelbelovende roadmap en constante ontwikkeling	54
3.2.6	Actieve community	55
3.2.7	Meerdere mogelijkheden voor tracing	55
3.3	Nadelen van ns-3	56
3.3.1	Gebreken bij WiFi simulaties	56
3.3.2	Geen mogelijkheid tot visualisatie	56
3.3.3	Documentatie	56
3.4	Samenvattende tabel van voor- en nadelen	57
4	Besluit	58

1 Inleiding tot ns-3

In deze inleiding wordt besproken wat ns-3 is en welke concepten en werkwijzen er gehanteerd worden. Ns-3 maakt namelijk gebruik van een aantal concepten en werkwijzen die op het eerste zicht vreemd en moeilijk begripbaar kunnen overkomen. Dit is echter het gevolg van het feit dat ns-3 een project is dat vanaf niets werd opgebouwd met bepaalde doelen voor ogen. Men heeft dus gezocht naar een wijze van implementatie die zich het beste leende om deze de doelen te bereiken.

Er wordt ook gekeken naar de mogelijkheden die ns-3 aanreikt aan een ontwikkelaar. Er zijn namelijk heel wat concepten die gerealiseerd zouden kunnen worden bij simulatie maar aangezien ns-3 nog in ontwikkeling is, is de lijst met mogelijkheden nog in zekere zin beperkt. Echter wordt in dit document aangehaald wat al voor handen is en wat er op de 'roadmap' van ns-3 staat.

1.1 Wat is ns-3?

Ns-3 is een raamwerk voor het ontplooiën en uitvoeren van netwerksimulaties. Het biedt een programmeur een set van tools, modellen en implementaties van veelgebruikte netwerkconcepten die deze kan gebruiken om een simulatie op poten te zetten. De talen C++ en Python worden hiertoe aangewend, waarbij Python zich koppelt met de C++ implementatie van ns-3. C++ is dus de taal waarin ns-3 wordt geschreven. Python vormt louter een extra abstractiegraad voor snelle ontwikkeling.

Ns-3 wordt bekeken als de opvolger van ns-2, een populaire netwerksimulatietool. Echter wordt ns-3 niet door het zelfde team ontwikkeld en bouwt het ook niet verder op de code van ns-2 [15]. Een ander groot verschil is het feit dat ns-2 gebruik maakte van C++ en TCL, waar ns-3 gebruik maakt van C++ en eventueel Python. De ontwikkelaars van ns-3 streven naar eigen zeggen een eenvoudige structuur na en willen voorkomen dat ns-3 het zelfde lot is toegeschreven als dat van ns-2, namelijk een onoverzichtelijk log systeem waarvan de modellen niet geverifieerd zijn.

Het ns-3 project ging van start in 2006 en wordt door drie partners gesteund: de Universiteit van Washington, het INRIA en de technologische universiteit van Georgia in Atlanta.

Een belangrijk aspect van ns-3 en de toekomst van het project, is het werken naar een systeem dat in combinatie met de realiteit kan gebruikt worden. Met andere woorden het koppelen van reële applicaties met een gesimuleerd netwerk en vice-versa. Dit aspect staat op het moment van schrijven echter nog in de kinderschoenen. Eén van de manieren om een combinatie te vor-

men tussen reële applicaties en simulaties in ns-3 zijn de zogenaamde Linux Containers in combinatie met een TAPBridge in ns-3, deze worden eveneens getest in dit werk.

Ook het koppelen van gesimuleerde nodes of applicaties met fysieke of virtuele netwerken is mogelijk, ns-3 heeft hiertoe een aantal concepten ter beschikking die de link leggen tussen het gesimuleerde en het fysieke netwerk. Een voorbeeld daarvan is het EmuNetDevice. Deze manier van werken wordt echter niet verder bekeken in dit werk, onder meer door het gebrek aan infrastructuur.

1.2 Concepten in ns-3

In ns-3 worden een aantal concepten gebruikt die, indien goed begrepen, het opzetten van een simulatie eenvoudiger en/of sneller kunnen maken. Eén van de belangrijkste concepten zijn de 'helper' klassen, een ander belangrijk concept in ns-3 is het gebruik van nodes, devices en hun respectievelijke containers. Deze concepten vormen de bouwstenen voor simulaties en zijn dus van groot belang voor het begrijpen van dit eindwerk.

1.2.1 Nodes en nodecontainers

Ns-3 maakt gebruik van zogenaamde nodes om hosts in een netwerk voor te stellen in de code. Een instantie van een nodeklasse is dus vergelijkbaar met een computerbehuizing in de echte wereld. Aan deze instantie kunnen dan apparaten, stacks en applicaties toegevoegd worden door middel van aggregatie.

Omdat bij complexe simulaties het aantal nodes in het netwerk snel kan oplopen voorziet ns-3 een manier om deze te ordenen. Dit gebeurt met behulp van nodecontainers, deze kunnen bekeken worden als een set van nodes waarop een aantal bewerkingen tegelijk gedaan kunnen worden. Zo kan men bijvoorbeeld een stack installeren op alle nodes van een container met behulp één functieaanroep. Nodecontainers maken code ook overzichtelijker aangezien de topologie van een netwerk duidelijker weergegeven kan worden in de code van de simulatie.

1.2.2 Devices en devicecontainers

Waar nodes te vergelijken zijn met hosts, stellen devices de apparaten/adapters voor in de simulatie. Ook deze hebben een corresponderende container om devices te verzamelen en er gezamenlijke bewerkingen op uit te voeren.

Voor elk soort fysiek netwerk zijn er bijgevolg bijhorende devices en devicecontainers. Een voorbeeld van zo'n device is het `CsmaNetDevice`, dat het simulatie-equivalent is van een ethernetkaart.

1.2.3 Applicaties

Een netwerk simuleren heeft weinig nut als er tussen de nodes in het netwerk geen trafiek is. Om die trafiek te genereren worden er applicaties gesimuleerd op de nodes. Deze applicaties worden allen afgeleid van een applicatieklasse die de basisinterface voor een applicatie levert. Applicaties vormen dus een elementair onderdeel van een simulatie. Ook applicaties kunnen verzameld worden in containers.

Een aantal basisapplicaties zoals bijvoorbeeld een echo server en client zijn al beschikbaar in `ns-3`. Een andere interessante applicatie voor simulaties is de `packetsink`, deze doet niet meer dan het aanvaarden van pakketten op de applicatie laag. Voor de rest worden deze pakketten niet meer verwerkt. Echter is het hoofddoel van de applicatieklasse de programmeur toe te laten van eigen applicaties in zijn simulatie in te brengen.

Er is ook een alternatief voor deze manier van simuleren. `ns-3` heeft namelijk de mogelijkheid om zich aan te koppelen aan een virtuele machine. Op die manier kan men 'echte' applicaties draaien op een gesimuleerd netwerk. Meer informatie hierover is te vinden in sectie 1.4.5 op pagina 14 .

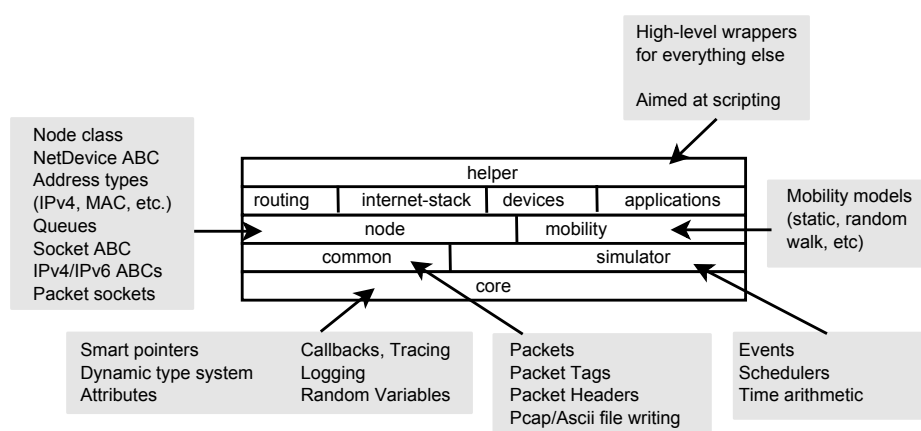
1.2.4 Helpers

Helpers zijn klassen die, zoals de naam al doet vermoeden, de programmeur helpen bij het opzetten van een simulatie. Ze nemen bepaalde veel voorkomende bewerkingen in het opzetten van complexe structuren over. Op deze manier kan er sneller een topologie opgebouwd worden en kan men zich focussen op de prioritaire elementen van een simulatie. Met helper klassen kan men bijvoorbeeld aan een aantal objecten, van dezelfde familie, tegelijk dezelfde attributen toekennen.

Helper klassen zijn beschikbaar voor alle concepten die hierboven al werden beschreven. Helpers kan men echter ook zelf implementeren, dit is bijvoorbeeld nuttig wanneer men een eigen device, model of dergelijke geschreven heeft. De precieze werking en het nut van helpers wordt echter pas echt duidelijk bij het gebruik ervan.

1.3 Modules

NS-3 is in grote lijnen opgebouwd vanuit een aantal modules die elk een bepaald aspect van het simuleren van netwerken op zich nemen. Een aantal van deze modules komen in zo goed als elke simulatie terug omdat zij de basis vormen van het ns-3 raamwerk. Echter zijn er ook modules die enkel voor specifieke doeleinden dienen. Een voorbeeld daarvan is de mobiliteitsmodule, die wordt enkel gebruikt indien men bewegende nodes moet kunnen simuleren. In figuur 1 zijn de verschillende modules van ns-3 te zien.



Figuur 1: De verschillende modules van ns-3, gelaagd weergegeven

De programmeur kan indien hij dat nodig acht zelf extra modules creëren aangezien deze bekeken kunnen worden als een aantal klassen die op een of andere manier gerelateerd zijn. Echter is dit niet voor de hand liggend en bijgevolg niet aangeraden, het wijkt ook sterk af van de originele bedoelingen van het ns-3 framework.

Modules zijn echter niet meer dan C++ header bestanden die een aantal declaraties doen en vooral de onderliggende klassen en code van de module oproepen. Op die manier wordt er gezorgd dat de programmeur minder include statements moet oproepen. Deze manier van werken wordt ook in andere frameworks toegepast, bijvoorbeeld Qt.[22]

1.3.1 Simulator module

De simulator module bevat twee submodules: Time en Scheduler. De simulator module beheert de virtuele tijd en gerelateerd daarmee het opvolgen van gebeurtenissen gedurende de simulatie. Een simulatie is gebaseerd op

gebeurtenissen die zich op bepaalde tijdstippen voordoen en op hun beurt volgende gebeurtenissen oproepen in de toekomst.

1.3.2 Core module

De core module vormt de kern van het hele framework, het definieert het volledig mechanisme waarop ns-3 verder gebouwd is. de core module omvat:

- Een zogenaamde functor klasse voor de callbacks in het systeem, namelijk ns3::Callback.
- Een platformafhankelijke interface om de wallclock van het hostsysteem aan te spreken.
- Een klasse voor het registreren van regressie testen.
- Debugger faciliteiten: asserts en logging.
- Algoritmen voor het berekenen van willekeurige variabelen volgens bepaalde wiskundige verdelingen.
- De basisklassen ns3::Object en ns3::ObjectBase waaruit andere klassen in het ns-3 framework worden afgeleid. Deze klassen voorzien faciliteiten voor aggregatie, trace bronnen, attributen en reference counting.
- Een smartpointer klasse ns3::Ptr, deze klasse wordt in samenwerking met ns3::Object gebruikt.
- Een configuratie klasse die attributen en trace bronnen configureert en centraal beheert tijdens de simulatie.

Deze Core module kan dus het hart van het ns-3 framework genoemd worden, de module wordt dan ook in elke simulatie gebruikt, dit is eveneens zo voor de in sectie 1.3.1 beschreven simulator module.

1.3.3 Common module

De common module verpakt enkele basisconcepten van computernetwerken. In geen enkele simulatie van een computernetwerk kan men namelijk zonder pakketten en data rates te werk gaan. Ook twee andere veel voorkomende zaken worden in deze module verpakt, namelijk de foutmodellen en pakket performantie.

1.3.4 Node module

Zoals in sectie 1.2.1 beschreven staat, werkt ns-3 met het concept van nodes in een netwerk. De Node module bevat de abstracte Node klasse waarvan alle nodes worden afgeleid. De module zorgt ook voor de mogelijkheid om een aantal nodegerelateerde concepten toe te voegen aan nodes, namelijk: sockets, queues, channels, adresseringsmogelijkheden, routing protocollen, ipv4, ipv6 en netwerk devices. Een node op zich is namelijk vergelijkbaar met een lege doos, enkel door het toevoegen van de zopas vermelde concepten kan een node nuttig zijn binnen een simulatie en bepaalde taken vervullen. Het toevoegen aan een node gebeurt met behulp van dynamische aggregatie, een werkwijze die veelvuldig gebruikt wordt in ns-3.

1.3.5 Devices module

De devices module bevat een reeks submodules die elk op zich één type device vertegenwoordigen. Deze modules bevatten de benodigde klassen en modellen om een bepaald device te simuleren. De devices module bevat op dit moment volgende submodules:

Bridge Een virtueel netwerk device dat verschillende delen van een LAN onderling verbind.

CSMA Een eenvoudig busmodel naar de idee van Ethernet verbindingen.

Emulated Net Device Een virtueel netwerkapparaat dat de simulatie toelaat te communiceren met een reëel netwerk. Dit device vormt een brug tussen het gesimuleerde en het reële.

MESH Mac laag mobiele mesh netwerken met behulp van 802.11s en FLAME.

Point to Point Point to Point connecties tussen twee dergelijke devices, dit volgens RFC 1661. Echter is het link control protocol niet geïmplementeerd en bijgevolg de bijhorende state machine ook niet. Er wordt uitgegaan van het feit dat de connectie al opgezet is.

TAP Bridge Een virtueel netwerk apparaat voor het integreren van een reële host in het gesimuleerde netwerk.

Wifi Module met alle modellen en apparaten die gerelateerd zijn aan de 802.11 WiFi standaarden.

Wimax Module met modellen en apparaten die gerelateerd zijn aan 802.16 Wimax.

1.3.6 Internet stack module

De internet stack module bevat een aantal klassen die een typische stack voorstellen die gebruikt wordt op het hedendaagse internet. De Module bevat namelijk een ipv4 stack, een module voor het simuleren van het ARP protocol en UDP en TCP implementaties.

1.3.7 Routing module

De routing module bevat een aantal submodules die elk een bepaalde manier van routing in een computernetwerk voorstellen. Elke submodule voorziet dus een aantal klassen die een bepaald routing systeem simuleren. De module bevat ook een aantal klassen om ipv4 routingtabellen te simuleren.

1.3.8 Helpers module

De helpers module bevat klassen en submodules die de verschillende helpers in ns-3 voorstellen. Hierbij valt op te merken dat op het moment van schrijven er geen informatie over deze module te vinden is in de doxygen van ns-3 [16], onder de sectie modules.

1.3.9 Applications module

De applications module bevat de klassen waarvan applicaties worden afgeleid en de door ns-3 reeds geproduceerde basisapplicaties. In de applicatie module zijn volgende applicaties reeds geïmplementeerd:

OnOffApplication Een applicatie die trafiek over het netwerk genereert in een aan-uit patroon. Als de applicatie op de aan stand staat verzend deze data aan een constante datarate. Het omwisselen tussen aan en uit stand kan via een aantal willekeurige verdelingen gebeuren. Deze applicatie wordt gebruikt in sectie 2.5.2 op pagina 32.

PacketSink Een eenvoudige toepassing die alle pakketten die op een bepaalde poort aankomen op de node waarop de applicatie draait opslorpt. Deze applicatie werd ontworpen voor gebruik als client bij de OnOffApplication maar kan ook in combinatie met andere applicaties gebruikt worden.

Ping6 Ipv6 variant van het ping commando.

Radvd Een router advertisement daemon.

Udp Echo client en server Eenvoudige echo applicaties, zowel client en server. Deze maken gebruik van UDP als onderliggend protocol.

1.3.10 Mobility module

De mobility module levert een aantal modellen voor mobiele nodes, deze modellen berekenen de bewegingen van nodes in draadloze netwerken. De beweging van nodes heeft namelijk effect op de ontvangst van draadloze signalen en de kwaliteit ervan. De module voorziet ook een belangrijke bron voor tracing, namelijk het aangeven dat de positie van een node veranderd is. De verschillende modellen in de mobility module zijn:

ConstantPositionMobilityModel Een model waarbij de posities van de nodes constant zijn tot deze door de gebruiker veranderd worden. Dit model is bijvoorbeeld goed voor access points.

ConstantVelocityMobilityModel Een model waarbij de snelheid waarmee de nodes bewegen constant blijft.

HierarchicalMobilityModel Een model dat hiërarchisch wordt opgebouwd uit andere mobiliteitsmodellen.

RandomWalk2dMobilityModel Een model waarbij de nodes at random bewegen in een 2d vlak. De programmeur kan hiervoor een willekeurige verdeling aangeven voor het genereren van de bewegingen.

RandomWaypointMobilityModel De nodes krijgen elk willekeurig en afzonderlijk een eindpunt aangeduid waar zij met willekeurige snelheid en pauzes naartoe bewegen.

RandomDirection2dMobilityModel Elke node krijgt een willekeurige richting en snelheid en beweegt volgens die parameters tot het een grens bereikt, op dat moment krijgt het opnieuw willekeurig een snelheid en richting en herhaalt zich het vorige.

WaypointMobilityModel Een model dat zich op waypoints baseert en waar nodes een bepaald pad door de waypoints volgen.

1.3.11 Contrib module

De contrib module bevat klassen en modules die zijn toegevoegd aan ns-3 maar nog niet zijn opgenomen in de rest van de structuur. Deze modules zijn veelal nog in ontwikkeling of experimenteel. Op het moment van schrijven

zijn er vijf modules en klassen te vinden in de Contrib module: Gnuplot, Congifstore, Gtkconfigstore, EventGarbageClllector en DelayJitterEstimation.

Zaken die in de contrib module te vinden zijn kunnen later migreren naar de rest van het ns-3 systeem of verdwijnen door een gebrek aan gebruik of interesse. Dit is dus een tijdelijke plaats voor nieuwe modules of klassen. De Contrib module is bijgevolg een module die veel voor verandering vatbaar is tussen verschillende iteraties van ontwikkeling.

1.3.12 Andere modules

NS-3 kent ook nog een klein aantal minder belangrijke modules zoals bijvoorbeeld Constants en Utils. Deze modules bevatten een aantal constanten, definities of weinig gebruikte klassen.

1.4 Wat ns-3 te bieden heeft

Hoewel ns-3 nog relatief jong is en men met de ontwikkeling en ondersteuning ervan nog niet volledig klaar is, beschikt het al over een ruime hoeveelheid aan netwerksystemen en protocollen. In deze sectie wordt een overzicht gemaakt van wat ns-3 bij het moment van schrijven, versie 3.8, omvat.

1.4.1 Protocollen

Out of the box kan ns-3 een aanzienlijk aantal protocollen simuleren, dit op de verschillende lagen van het OSI-netwerkmodel. Een deel van deze protocollen zijn een gevolg van de gesimuleerde concepten zoals de hardware, stacks en bepaalde applicaties. Echter kan een programmeur natuurlijk ook zelf protocollen en bijhordende modellen toevoegen indien dit nodig is.

Een aantal frequent voorkomende protocollen die bijvoorbeeld gesimuleerd worden zijn ARP, UDP, TCP, 802.11, ...

1.4.2 Modellen

Om goede simulaties te kunnen verwezenlijken zijn er ook een aantal modellen nodig die met behulp van berekeningen bepaalde 'real life' fenomenen simuleren. Voorbeelden hiervan zijn de mobiliteitsmodellen bij draadloze netwerken en, eenvoudiger, de foutbalans modellen voor het berekend introduceren van fouten in een netwerk. Goede modellen zijn van uiterst belang bij netwerk-simulatie, zij moeten er voor zorgen dat de simulatie het realisme zo goed mogelijk benadert.

Een punt dat vaak gemaakt wordt bij presentaties over ns-3 is dat men bij ns-2 modellen hanteerde die niet getest waren en op geen enkele manier zekerheid konden voorleggen dat ze de realiteit benaderden. Volgens de ontwikkelaars van ns-3 moet dit voorkomen worden, er zijn momenteel dan ook nog minder modellen in de repository van ns-3 maar men geeft wel de garantie dat deze modellen in zekere zin getest zijn.

1.4.3 Schaalbaarheid

Om een goede schaalbaarheid te voorzien, heeft ns-3 een aantal interessante concepten toegevoegd aan ns-3. Zo kan men voor dummy data gebruik maken van zogenaamde virtual zero bytes. Virtual zero bytes nemen, in tegenstelling tot gewone data, zo goed als geen geheugen in. Dit omdat dummy data geen verschil maakt in de simulatie. Op deze manier kan een aanzienlijke hoeveelheid geheugen uitgespaard worden, zonder dat dit effect heeft op de simulatie.

Zoals eerder al beschreven, wordt aan nodes functionaliteit toegevoegd door het aggregeren van objecten in het node object, ook dit zorgt voor beter schaalbaarheid. Sommige nodes kunnen bijvoorbeeld geen nood hebben aan een stack, mobility model en dergelijke. Men kan dus van een besparing van geheugen spreken voor die nodes. Enkel de objecten die effectief gebruikt worden zullen aangemaakt worden. Op die manier spaart men het geheugen uit dat gebruikt zou worden indien nodes op statische manier alle objecten al zouden bevatten. Deze manier van werken verzekert eveneens dat nieuwe concepten op een eenvoudige manier kunnen worden toegevoegd aan nodes.

Ns-3 doet ook in zekere mate aan memory management door middel van reference counting en smart pointers. Op deze manier kan men aan garbage collection doen en het geheugenverbruik tot een minimum beperkt houden.

1.4.4 Multi-laag eigenschappen

Als een van de "interessante eigenschappen van ns-3" haalt Gustavo Carneiro [11] de multilaag eigenschappen aan. Hiermee doelt hij op twee eigenschappen, het tracing systeem in ns-3 en de packet tags. Packet tags zijn kleine hoeveelheden informatie die toegevoegd worden aan de pakketten in functie van de simulatie.

1.4.5 Integratie in de reële wereld

Ns-3 heeft een aantal manieren om de simulaties te integreren in de reële wereld en vice versa. Dit loopt van het opslaan van pakketten naar .pcap

bestanden tot POSIX emulatie en het gebruik van een "network simulation cradle".

Pcap bestanden Om een betere analyse van een simulatie mogelijk te maken, ondersteunt ns-3 de mogelijkheid per device in de simulatie een pcap bestand uit te schrijven. Deze bestanden kunnen bekeken worden met de netwerkanalysesoftware Wireshark. Deze manier van werken legt ook een zeer duidelijke link met de reële wereld waar men met behulp van Wireshark per netwerkadapter een capture kan maken, wat equivalent is aan een pcap bestand.

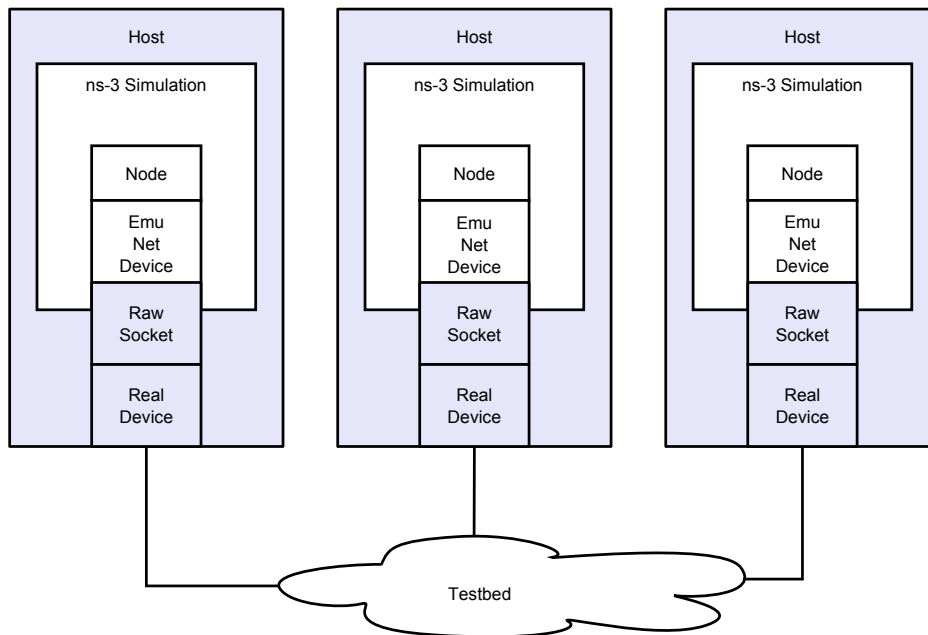
Real-time scheduler Indien een simulatie wordt gekoppeld aan de echte wereld, moeten de gesimuleerde gebeurtenissen eveneens gebeuren volgens de reële tijd in plaats van een gevolg te zijn van louter berekeningen. Om dit te bereiken heeft men in ns-3 een real-time scheduler geïmplementeerd waarin events volgens de wall clock time worden uitgevoerd. Op die manier blijft de simulatie gesynchroniseerd met de reële wereld.

Network Simulation Cradle Een manier om met reële linux sockets te werken gekoppeld aan de simulatie is NSC ofwel network simulation cradle. Dit is een project dat los staat van ns-3 en het wordt ontwikkeld door de WAND network research group. Rr loopt momenteel een project binnen ns-3 om NSC te porten voor bruikbaarheid in de ns-3 simulator. Meer informatie hierover is te vinden op de wiki pagina's van ns-3 [21]. Momenteel zijn er al bepaalde zaken die werken onder linux kernels 2.6.18 en 2.6.26. Echter is de todo- en wishlist nog te lang om verder in te gaan op dit element.

Het doel van NSC is het gebruiken van reële TCP/IP stacks in een netwerk simulator. NSC is een project dat loopt sinds 2005 en een laatste grote update kende in september van het jaar 2009. Meer informatie over NSC kan men vinden op de webpagina [23] die WAND er voor heeft aangemaakt.

POSIX emulatie Ns-3 probeert applicaties die POSIX-compliant zijn te laten werken met ns-3. Op die manier kan men een applicatie die van POSIX-sockets gebruik maakt koppelen aan nodes in een ns-3 netwerk. Wat er achterliggend gebeurt is het omzetten van de POSIX API calls naar ns-3 calls die verder door de simulator verwerkt kunnen worden. Voor de toekomst staat ook het draaien van routing daemons op deze manier in de planning van het project.

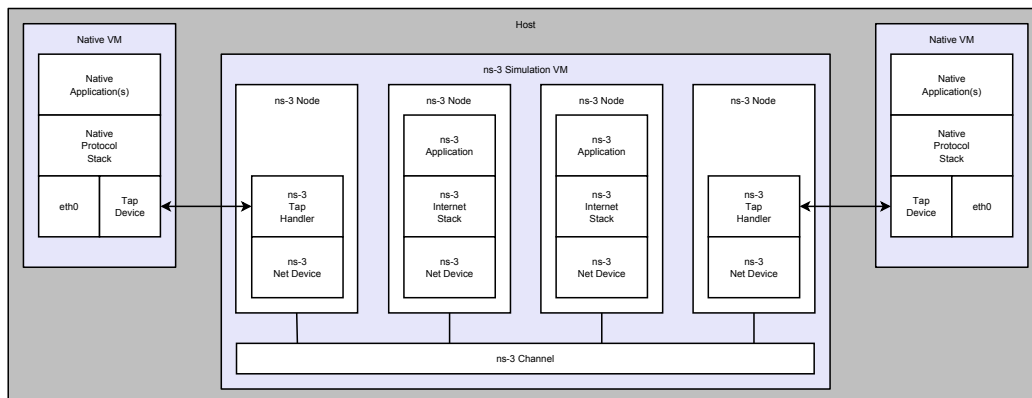
EmuNetDevice Een manier om ns-3 applicaties te draaien in reële netwerken is gebruik te maken van het EmuNetDevice. Dit is een NetDevice en werkt net zoals de andere NetDevices naadloos samen met de nodes. Echter zal het achterliggend de pakketten niet doorsluizen naar een kanaal in de simulator maar naar een socket van het hostsysteem. Op deze manier kan men ns-3 applicaties draaien op een testbed, zoals te zien is in figuur 2.



Figuur 2: Gesimuleerde applicaties op reële hosts in een testbed

TapBridge en Linux Containers Een laatste manier om een hybride simulatie te maken is het gebruik van de TapBridge van ns-3 in combinatie met Linux Containers. Op deze manier worden er virtueel nieuwe hosts gedraaid op een Linux systeem, echter zal enkel de netwerkfunctionaliteit virtueel zijn. De applicaties die op deze virtuele hosts draaien, draaien dus native maar hun netwerkfunctionaliteit zal via een TAP en virtuele bridge naar een TapBridge in de ns-3 simulatie gestuurd worden. Deze TapBridge bevindt zich in een node van het netwerk, deze node bevat enkel de TapBridge en een NetDevice zoals weergegeven in figuur 3.

Deze werkwijze zal eveneens uitgewerkt worden in een simulatie verder beschreven in dit werk in sectie 2.5.3. Bij die simulatie zullen een aantal reële applicaties, VLC server en client, trafiek naar elkaar sturen over een gesimuleerd netwerk.



Figuur 3: reële applicaties in reële hosts die via een gesimuleerd netwerk en Tap-Bridges in ns-3 communiceren met elkaar

1.4.6 Visualisatie

Ns-3 beschikt momenteel nog niet over een manier van visualisatie van de netwerktopologieën of de simulaties. Echter zijn er wel al enkele experimentele projecten op poten gezet zoals bijvoorbeeld ns-3-PyViz, een ns-3 visualisatie library en enkele kleinere persoonlijke projecten. In de repository van ns-3 zijn deze echter niet aanwezig.

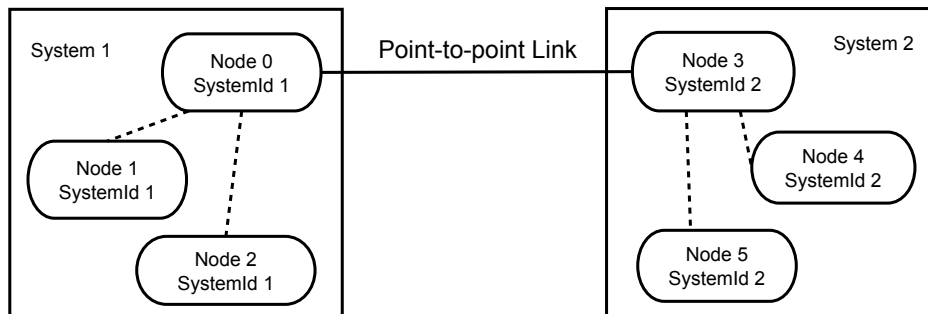
Eveneens is er op de wiki van het ns-3 project een verwijzing te vinden naar een grafische editor voor ns-3 simulaties. Deze editor genereert aan de hand van een grafisch getekend netwerk een C++ file met de topologie. Echter is de gegenereerde code nog vrij slecht leesbaar en zijn er ook nog maar weer beperkte mogelijkheden in het programma. Meer informatie is te vinden op de website van het programma [24]. Het programma werd ontwikkeld door een student aan de universiteit van Straatsburg.

1.4.7 Gedistribueerde simulatie met MPI

Sinds versie 3.8 van het ns-3 framework is het mogelijk gedistribueerde simulaties uit te voeren met behulp van MPI. MPI staat voor Message Passing Interface en omvat een library en protocol ter ondersteuning van gedistribueerde applicaties.

Deze library wordt door ns-3 aangewend om nodes van de simulatie te verspreiden over verschillende systemen in een cluster. Elke node krijgt dan een systemId als eigenschap, deze systemId bepaalt op welk systeem in de cluster de node gesimuleerd zal worden. Echter is er wel een belangrijke beperking, links tussen nodes op een verschillend systeem kunnen enkel van

het type point-to-point zijn zoals aangegeven in figuur 4.



Figuur 4: Zes nodes verspreid over twee systemen in een cluster met behulp van MPI voor gedistribueerde simulatie.

1.4.8 Routing

Ns-3 biedt een aantal verschillende manieren van routing aan voor haar simulaties. Men kan ze opsplitsen in vier categorieën: adhoc, global, nix-vector en list.

Adhoc Ns-3 biedt twee adhoc algoritmen/protocollen aan. OLSR (rfc 3626) wordt ondersteund en kent sinds versie 3.8 volledige HNA ondersteuning. Het andere is AODV (rfc 3561).

Global Het global routing ofwel GOD routing is een eenvoudige manier van routing waarbij de simulator alle routes statisch berekent in het begin van de simulatie. Deze manier van routing zal ook gehanteerd worden in dit werk.

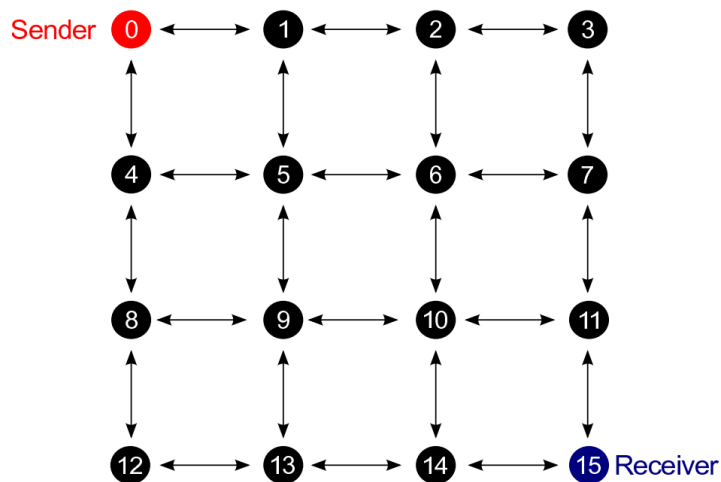
Nix-vector Deze manier van routing is het beste geschikt voor simulaties met een zeer groot aantal nodes die via point-to-point of ethernet verbonden zijn. Nix-vector voorziet een hoge performantie maar is in ruil ook beperkt in mogelijkheden.

List Met list-routing kan men verschillende routing protocollen laten samenwerken in een bepaalde node. zo kan men bijvoorbeeld de combinatie maken van statische routing tabellen met OLSR of AODV. Dit kan goed van pas komen als een deel van het netwerk een bepaald protocol gebruikt terwijl een ander deel dat niet doet.

1.4.9 Performantie

Een belangrijk aspect van een netwerksimulator is de performantie ervan. In 2009 publiceerde een team van de Distributed Systems Group aan de universiteit van Aachen een paper waarin recente netwerksimulators werden vergeleken aan de hand van hun performantie [12]. Het team nam de volgende simulators onder de loep: ns-2 [15], OMNet++ [5], ns-3, SimPy [6] en JiST/SWANS [10]. Een opmerking hierbij is dat ns-2 niet als een recente simulator werd beschouwd, maar toch werd gebruikt. Dit om een vergelijking te kunnen maken met een oudere maar veel gebruikte simulator en deze als ijkpunt te gebruiken: "We include ns-2 here to form a baseline" [12].

De simulators werden getest met een zelfde te simuleren netwerk en alle simulaties werden gedraaid op het zelfde systeem. Het te simuleren netwerk bevatte een willekeurig aantal nodes die onderling verbonden waren volgens het patroon te zien op figuur 5. De nodes in dit netwerk zenden elke seconde een packet uit naar de naburige nodes, elke node zend op zijn beurt de voor het eerst ontvangen pakketten door, op deze manier doet men het netwerk 'overstromen'.



Figuur 5: netwerk topologie gebruikt door DSG, afbeelding toont een netwerk met size=16.

In de conclusie van de paper kan men terugvinden dat, hoewel JiST de snelste simulator is, ns-3 betere algemene resultaten kan voorleggen omdat het een kleinere hoeveelheid geheugen consumeert. Ns-3 komt samen met JiST en OMNet++ als beste uit het onderzoek. Er dient wel opgemerkt te worden dat men hier gebruik maakte van ns-3.1, een sterk verouderde versie.

1.4.10 Willekeurige verdelingen

Ns-3 maakt in een aantal van haar modellen en modules gebruik van willekeurige variabelen, deze worden berekend aan de hand van een wiskundige verdeling. Ter volledigheid worden de mogelijke verdelingen die ns-3 aanbiedt opgesomd: uniforme verdeling, constante verdeling, sequentiële verdeling, exponentiële verdeling, normaalverdeling, logaritmische normaalverdeling, ...

2 Simuleren met ns-3

In deze sectie zal duidelijk gemaakt worden, aan de hand van drie simulaties, hoe men netwerken kan simuleren met ns-3. Echter moet hierbij opgemerkt worden dat niet alle aspecten van ns-3 kunnen behandeld worden. Een aantal concepten voor koppeling tussen simulatie en de reële wereld konden niet worden gebruikt door gebrek aan infrastructuur of wegens de instabiliteit ervan. Bij elke simulatie zal ook een bespreking gemaakt worden van de resultaten van de simulatie.

Voor de volledigheid wordt ook de gebruikte code kort toegelicht waar dit nodig wordt geacht.

2.1 Ns-3 builden en testen

Ns-3 maakt gebruik van een aantal belangrijk software toepassingen waaronder g++, python, en mercurial. Het heeft de mogelijkheid te werken op Unices of een Windows systeem, dit laatste met behulp van Cygwin of Mingw. Er wordt in dit document echter uitgegaan van een Linux omgeving, meer bepaald Ubuntu 10.04 LTS. Er wordt ook gebruik gemaakt van C++ als programmeertaal.

Om aan de basisvereisten voor het downloaden en builden van ns-3 te voldoen kan men op ubuntu het volgende commando uitvoeren:

```
sudo apt-get install build-essential g++ python mercurial
```

Indien dit met succes is beëindigd kan men volgende commando's in de terminal uitvoeren om ns-3 te downloaden en te builden.

```
hg clone http://code.nsnam.org/ns-3-allinone/  
cd ns-3-allinone  
./download.py #downloaden van de vereiste componenten  
./build.py    #builden van ns-3
```

Een optionele stap, maar een die wel aangeraden is, is het testen van het zopas gecompileerde ns-3 pakket. Men kan ns-3 eenvoudig testen met het uitvoeren van het commando

```
./test.py
```

Op deze manier kan men nakijken of alle componenten naar behoren werken. Meer informatie hierover kan men vinden op de wiki [20] van het ns-3 project.

2.2 Simulaties 'runnen'

Het uitvoeren van een simulatie kan op verschillende manieren in gang gezet worden. Wanneer men een nieuwe simulatie schrijft in C++ dient die eerst gecompileerd te worden alvorens deze kan worden uitgevoerd. Het compileren van de simulaties wordt gedaan met behulp van Waf. Waf is een flexibel build systeem dat gebaseerd is op Python [7].

De eenvoudigste manier voor het uitvoeren van een simulatie is gebruik te maken van het volgende commando:

```
./waf --run <naam simulatie>
```

Hierbij wordt `naam simulatie` vervangen door de naam van het c++ bestand waarin de simulatie beschreven staat, zonder de extensie.

Men kan ook gebruikmaken van waf om een nieuwe shell op te roepen en dan zelf de binary aan te roepen. Op deze manier kan men dan ook gebruik maken van tools zoals GDB om de code te debuggen. Om de nieuwe shell op te roepen en de bestanden te compileren roept men volgend commando aan:

```
./waf --shell
```

Nadat dit commando is uitgevoerd kan men de gecompileerde simulatie oproepen, deze bevindt zich in een pad binnen de ns-3 mappenstructuur. Het pad is als volgt, waarbij variant 'debug' of 'optimized' kan zijn naar behoeven van de gebruiker.

```
build/variant/pad-naar-bestand/bestandsnaam
```

Om de shell te verlaten en af te sluiten kan men 'exit' oproepen.

2.3 Voorbeelden van simulaties

Ns-3 voorziet een aantal voorbeeldsimulaties in de map 'examples'. Deze zijn allen gefocust op een bepaald aspect van netwerken en kunnen, zoals in de vorige paragraaf beschreven werd, worden uitgevoerd. Een speciaal geval hierbij is de map 'tutorial' die een aantal genummerde bestanden bevat die overeenkomen met de tutorial [18] van ns-3 die op de website te vinden is.

2.4 Opbouw van een simulatie

Wanneer men naar de voorbeelden kijkt zal men opmerken dat de structuur van deze bestanden veelal hetzelfde is. In de voorbeelden zal eerst een zogenaamde boilerplate lijn staan voor de emacs editor en een stuk commentaar gevuld met juridische informatie. Deze twee delen van de code zijn natuurlijk niet vereist voor zelf geschreven simulaties.

2.4.1 Preamble

De echte code begint pas bij het includen van de juiste modules voor de simulatie. Deze modules zijn eigenlijk gewoon headerfiles die op hun beurt een aantal includes uitvoeren en het zo eenvoudiger maken voor de gebruiker om snel een simulatie te schrijven. Het is aan te raden de volgende include-statements telkens op te nemen bij het schrijven van een nieuwe simulatie.

```
#include "ns3/core-module"  
#include "ns3/simulator-module"  
#include "ns3/node-module"  
#include "ns3/helper-module"
```

Na het includen van de modules wordt de namespace van het ns3 project geselecteerd. Ook hier is het aangeraden dit bij elke simulatie te doen, het levert namelijk code op die beter leesbaar is.

```
using namespace ns3;
```

Om de logging gemakkelijk te kunnen aanspreken voor deze specifieke simulatie wordt een methode opgeroepen die een nieuwe loggingcomponent aanmaakt met een bijbehorende naam. Op deze manier kan men later bijvoorbeeld logging naar de console af en aan zetten door te refereren naar de naam van de component.

```
NS_LOG_COMPONENT_DEFINE ("naam");
```

Bij eenvoudige simulaties volgt nu een uitgebreide main functie. Echter, zoals in complexere voorbeelden te zien is, kan men hier ook nog een aantal callback functies voor tracing en dergelijke declareren. Deze laatste manier van werken wordt verder in dit document toegelicht. De main functie van de simulatie bevat zo goed als alle code, meestal in ongeveer dezelfde volgorde.

```
int main(int argc, char *argv[])  
{  
    ...  
    Code voor simulatie  
    ...  
}
```

De effectieve code voor de simulatie bevindt zich in de main functie. Ook deze code volgt min of meer een volgorde die bijdraagt aan leesbaarheid en structuur ervan. De grote lijnen ervan worden besproken, er zijn namelijk een variërend aantal optionele zaken. Die worden echter duidelijk bij de simulaties die verderop besproken worden.

2.4.2 Topologie en adressen

Bij het opbouwen van een simulatie begint men met een aantal nodes aan te maken en deze in één of meerdere nodecontainers te structureren. Dit kan bijvoorbeeld aan de hand van het subnet waartoe ze zullen behoren of aan de hand van de netwerkadapters die ze zullen bevatten. Het kan bijvoorbeeld handig zijn een nodecontainer te maken waarin alle nodes zitten die een point-to-point connectie hebben. Men kan dan met behulp van een helper in één functieaanroep dit device installeren op alle nodes in de container. Anderzijds kan het op analoge manier handig zijn om adressen toe te kennen aan een aantal nodes. Een voorbeeld:

```
NodeContainer nodes;  
nodes.Create(2);
```

```
PointToPointHelper pointToPoint;  
pointToPoint.Install(nodes);
```

In bovenstaand voorbeeld wordt een nodecontainer aangemaakt die twee nodes bevat. Op deze nodes worden point-to-point apparaten geïnstalleerd en verbonden met behulp van de Install methode van de PointToPointHelper. Het is echter belangrijk in acht te nemen dat dit een vereenvoudigd voorbeeld is. Tussen het aanmaken van de PointToPointHelper-klasse en het aanroepen van de Install methode ervan worden normaal nog een aantal attributen toegekend aan de klasse. Meestal wordt de returnwaarde van de Install methode ook bijgehouden, deze bevat namelijk de devices die geïnstalleerd werden op de nodes. Op analoge wijze kan men andere nodes en devices aan elkaar koppelen. Indien er een node is die men graag in twee containers heeft kan men deze eenvoudig toevoegen door de 'Add' methode aan te roepen op de container waaraan men de node wil toevoegen.

Nadat de nodes en hun bijhorende connecties zijn aangemaakt kan men de stacks installeren. Ook dit gaat op analoge manier aan het voorgaande. Men maakt een InternetStackHelper aan en roept daarvan de Install methode op, met als argument de nodes waarop men de stack wenst te installeren. Als de stacks zijn geïnstalleerd kan men de adressen toekennen aan de devices van de nodes.

```
Ipv4AddressHelper address;  
address.SetBase("10.1.1.0", "255.255.255.0");  
Ipv4Interfacecontainer interfaces = address.assign(devices);
```

In bovenstaande voorbeeldcode wordt er een Ipv4AddressHelper aangemaakt die de IP-adressen aan de devices in de DeviceContainer 'devices' zal

toekennen. De methode `SetBase` wordt gebruikt om de helper te laten weten welk IP het netwerk heeft en het bijhorende subnetmask. Met deze informatie zullen de IP-adressen worden toegekend in oplopende volgorde. Wanneer men opnieuw IP-adressen wil toekennen aan een andere set van devices hoeft men enkel de methode `SetBase` opnieuw op te roepen, het nieuwe adres en subnetmask mee te geven en de `Install` methode te gebruiken. Op deze manier kan men het volledige netwerk van IP-adressen voorzien. Er bestaat eveneens een IPv6 equivalent voor de `IPv4AddressHelper` en `Ipv4InterfaceContainer`.

2.4.3 Applicaties

Als de nodes, connecties, devices en adressen aangemaakt zijn moet er enkel nog trafiek gegenereerd worden op het netwerk. Hiervoor voorziet ns-3 voornamelijk het gebruik van applicaties. Applicaties in ns-3 zijn gesimuleerde applicaties, deze applicaties kan men bekomen door de `Application` klasse af te leiden. Ns-3 beschikt echter zelf over een aantal applicaties. Twee daarvan zijn de `UdpEchoServer` en `UdpEchoClient`, ze vormen het equivalent van de echo servers en clients in de reële wereld.

```
UdpEchoServerHelper echoServer(1024);
```

```
ApplicationContainer servers = echoServer.Install(nodes.Get(1));  
servers.Start(Seconds(2.5));  
servers.Stop(Seconds(30.0));
```

Bovenstaande code maakt een `UdpEchoServerHelper` aan met als attribuut het getal 1024. Het is namelijk een conventie bij ns-3 dat men bij het aanmaken van Helpers de non-optionele attributen aan de constructor meegeeft. In dit geval is dat de poort waarop de echo server pakketten zal ontvangen. Nadat een bepaalde applicatie degelijk geconfigureerd is kan deze geïnstalleerd worden op één of meerdere nodes. Het installeren gebeurt zoals gebruikelijk met de `Install` methode en levert in dit geval als returnwaarde een lijst met de applicaties die geïnstalleerd werden. Deze worden opgevangen in een `ApplicationContainer` waarna hun start- en stoptijd wordt ingesteld. Bij dit voorbeeld zal de server dus aan staan vanaf 2,5 seconden tot 30 seconden na het starten van de simulatie.

```
UdpEchoClientHelper echoClient(interfaces.GetAddress(1), 1024);  
echoClient.SetAttribute("MaxPackets", UIntegerValue(1));  
echoClient.SetAttribute("Interval", TimeValue(Seconds(1.)));  
echoClient.SetAttribute("PacketSize", UIntegerValue(512));
```

```
ApplicationContainer clients = echoClient.Install(nodes.Get(0));
clients.Start(Seconds(3.0));
clients.Start(Seconds(29.0));
```

Op analoge wijze aan het opzetten van de server kan men ook clients opzetten, echter zijn er kleine verschillen te merken. In bovenstaande code kan opgemerkt worden dat er een aantal keer `SetAttribute` wordt opgeroepen en er niet één, maar twee attributen worden meegegeven aan de constructor van `UdpEchoClientHelper`. Dit laatste is een logisch gevolg van het feit dat de client moet weten op welk adres de server te vinden is en op welke poort deze reageert. Het oproepen van `SetAttribute` dient om extra, niet verplichte, attributen aan te passen. In dit geval wordt het maximum aantal pakketten op één gezet, zal er om de seconde een pakket verstuurd worden en is de grootte ervan 512 bytes.

De manier waarop attributen hun waarde toegekend krijgen kan vreemd ogen, bij de meeste klassen in C++ wordt dit eenvoudigweg gedaan met een set en een get methode specifiek voor het attribuut. Omdat dit voor applicaties en bijkomende helpers een onoverzichtelijke structuur kan veroorzaken en men op de huidige manier beter aan tracing van attributen kan doen, heeft men voor deze aanpak gekozen. Elk attribuut gaat dus gepaard met zijn naam in de vorm van een string, elk attribuut heeft ook een type. De types in bovenstaande code zijn bijvoorbeeld `UInteger` en `Time`, ze moeten dan ook met `UIntegerValue` en `TimeValue` aangepast worden. De verschillende attributen en hun types van een bepaalde klasse zijn te vinden op de doxygen [16] pagina's van ns-3.

2.4.4 Simulator, Run!

Om de simulatie effectief te laten beginnen wordt de functie `Simulator::Run` aangeroepen. Deze methode zal alle informatie verwerken en de simulatie van het netwerk voor handen nemen. De methode eindigt als de simulatie afgerond is.

```
Simulator::Run();
Simulator::Destroy();
return 0;
```

Het bovenstaand stuk code wordt gebruikt om de main functie af te sluiten. De simulatie wordt uitgevoerd door `Simulator::Run` en vervolgens worden alle overblijvende objecten afgebroken door `Simulator::Destroy`. Als dit is afgehandeld wordt de main functie beëindigd en is de simulatie volledig gedaan.

2.5 Simulaties en resultaten

Er werden drie simulaties gedaan in het kader van deze bachelorproef. Een eerste simuleert een eenvoudige ethernetverbinding, een tweede een draadloos netwerk en tenslotte is er een derde complexere opstelling met verschillende soorten verbindingen. Deze simulaties werden telkens in identieke omstandigheden uitgevoerd om de vergelijkbaarheid ervan te behouden. De simulaties werden uitgevoerd op een systeem met een processorsnelheid van 2,4GHz en een werkgeheugen van 4 gigabyte, een Apple MacBook Pro met als besturingssysteem Ubuntu 10.04.

De resultaten van de simulatie worden telkens besproken aan de hand van de verzamelde gegevens met behulp van tracing of pcap-bestanden. Het is namelijk belangrijk te kunnen bepalen in hoeverre het ns-3 framework realiteit of theorie benaderd om de zwakke en sterke punten te ontdekken.

2.5.1 Eerste simulatie

De eerste simulatie bestaat uit vier nodes die elk verbonden zijn aan een switch via een ethernetverbinding. Twee van de vier nodes zullen een constante stroom aan data sturen naar de twee andere nodes in het netwerk. Om deze stroom aan data te sturen werd een eenvoudige zelfgeschreven applicatie aan de simulatie toegevoegd. De simulatie heeft als output een pcap bestand voor elke interface in het netwerk alsook een trace van het congestionwindow van een van de nodes.

De Applicatie bestaat erin pakketten van een opgegeven grootte te versturen aan een opgegeven datarate. De transactie van deze pakketten gebeurt volgens het TCP protocol. De implementatie van de applicatie bestaat uit een aantal functies. De belangrijkste zijn SendPacket en ScheduleTx, deze zorgen namelijk voor het verzenden van de pakketten op het juiste moment en aan de juiste bitrate, zoals te zien is in listing 1.

Listing 1: SendPacket en ScheduleTx methodes voor de applicatie in simulatie 1

```
void TrafficGen::SendPacket()
{
    //packet verzenden en via ScheduleTx volgende verzending
    //van een packet aanvragen
    Ptr<Packet> packet = Create<Packet>(m_packetSize);
    m_socket->Send(packet);

    if(++m_packetsSent < m_nPackets)
        ScheduleTx();
}
```

```

void TrafficGen::ScheduleTx()
{
    if(m_running)
    {
        //bepalen wanneer het volgende packet wordt verzonden
        .
        //tijd wordt berekend aan de hand van de datarate en
        //de packetgrootte in bits.
        Time tNext (Seconds (m_packetSize * 8 / static_cast<
            double> (m_dataRate.GetBitRate ()))));
        //volgende verzending in scheduler van simulator
        //zetten volgens tNext
        m_sendEvent = Simulator::Schedule (tNext, &TrafficGen
            ::SendPacket, this);
    }
}

```

De applicatie wordt als een klasse afgeleid van de ns-3 applicatieklasse en krijgt daarom ook de functies StartApplication en StopApplication mee om deze op een bepaald tijdstip te kunnen starten en stoppen.

De Simulatie is van topologie niet zeer speciaal, echter vergt een switch in het netwerk toch enig werk om een juiste opbouw te hebben. Een switch wordt in ns-3 gesimuleerd als een node die voor elke aansluiting een apart NetDevice heeft, in dit geval zijn dat er dus vier. Telkens moet er een ethernetlink gemaakt worden tussen een device op de switch en een van de nodes. Dit kan men versneld doen met behulp van een for-lus en een aantal node- en devicecontainers. De specifieke code is te zien in listing 2. De laatste drie lijnen in die code maken van de switchNode een node met effectieve switchcapaciteiten.

Listing 2: Het maken van een switch in ns-3

```

//csma devices installeren en opslaan in
//netdevicecontainers voor switch en nodes
NetDeviceContainer csmaDevices;
NetDeviceContainer switchDevices;
for (int i = 0; i < 4; i++)
{
    NetDeviceContainer link = csma.Install (NodeContainer
        (csmaNodes.Get (i), csmaSwitch));
    csmaDevices.Add (link.Get (0));
    switchDevices.Add (link.Get (1));
}

```

```

//switch installeren met behulp van bridgehelper
Ptr<Node> switchNode = csmaSwitch.Get (0);
BridgeHelper bridge;
bridge.Install (switchNode , switchDevices);

```

Zoals eerder al vermeld wordt er tracing gedaan van het congestionwindow van een van de devices in het netwerk. Het tracingsysteem van ns-3 laat namelijk toe dat de programmeur zelf een methode schrijft voor het traceren van een bepaalde trace-source. Die methode wordt dan opgeroepen door middel van callbacks.

Listing 3: callback methode voor het traceren van een congestionwindow

```

static void
CwndChange (uint32_t oldCwnd, uint32_t newCwnd)
{
    NS.LOG.UNCOND ( Simulator::Now () .GetSeconds () << "\t" <<
        newCwnd);
}

```

In listing 3 wordt de methode aangemaakt die wordt opgeroepen als er een verandering plaatsvindt in het congestionwindow. De methode zal dan de tijd van deze verandering en de nieuwe waarde van het congestionwindow loggen. Deze output kan dan achteraf met gnuplot gebruikt worden om een overzicht te krijgen van de evolutie van het congestionwindow.

Het aangeven van de koppeling tussen een trace-source en de callbackmethode gebeurt door middel van een methode in het object dat de trace-source bevat, deze wordt meestal aan het einde van de main functie van de simulatie geplaatst. De code hiervoor is te zien in listing 4

Listing 4: trace-source koppelen aan de callback methode

```

//socket op node 0 met congestwindow tracing (verzsendende
//node)
Ptr<Socket> ns3TcpSocket1 = Socket::CreateSocket (csmaNodes
    .Get (0), TcpSocketFactory::GetTypeId ());
//congestionwindow traceren op socket, analyseerbaar met
//gnuplot.
ns3TcpSocket1->TraceConnectWithoutContext ("
    CongestionWindow", MakeCallback (&CwndChange));

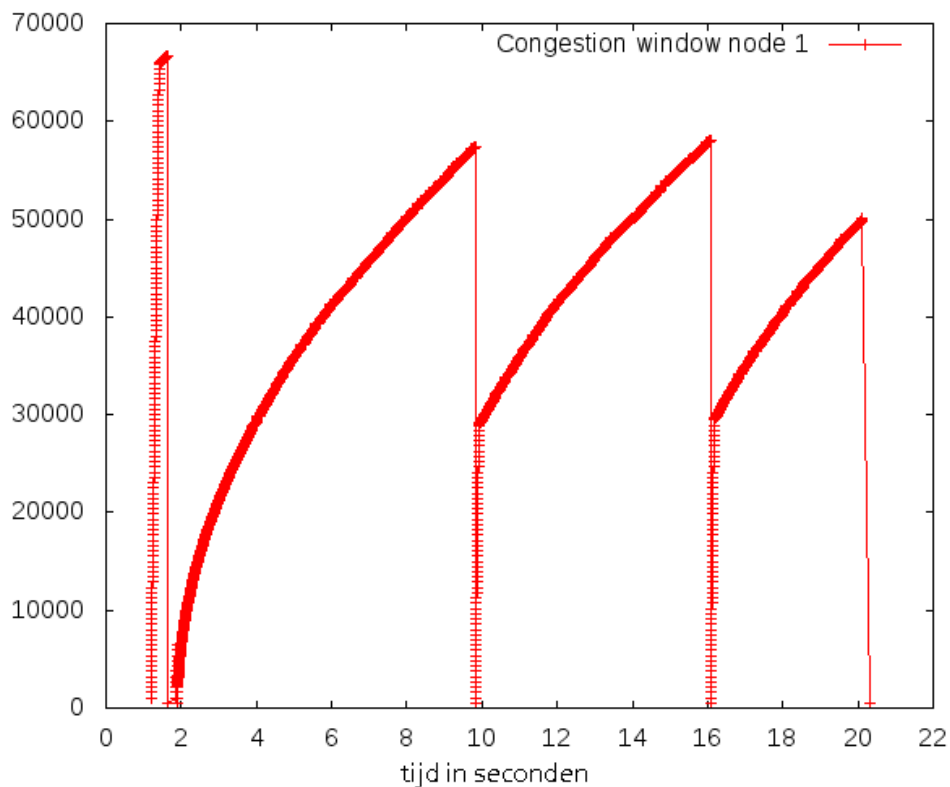
```

De simulatie simuleert 20 seconden op dit netwerk waarbij de applicaties op node 1 en node 3 starten na 1 seconde en stoppen na 19 seconden. Er zijn Packetsinks geïnstalleerd op nodes 2 en 4, deze starten en eindigen gelijktijdig met de simulatie om de trafiek op te vangen. De datarate en

pakketgrootte voor de applicatie zijn respectievelijk 7Mbps en 1024 op node 1 en 5Mbps en 265 op node 3. De snelheid van de ethernetconnecties is 10Mbps.

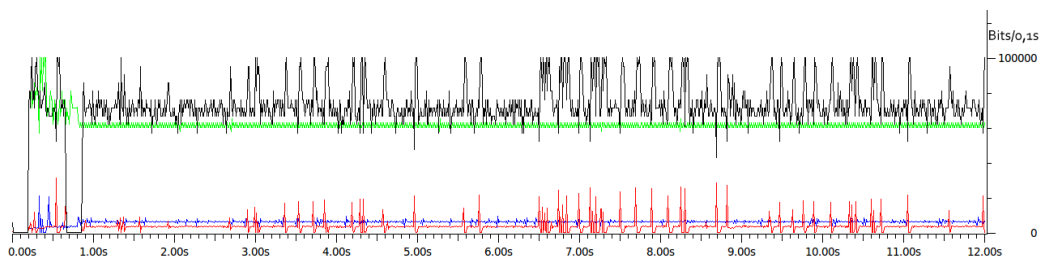
Resultaten

Allereerst wordt gekeken naar het congestionwindow van het device op node 1, dit werd namelijk bijgehouden via logging en kon met behulp van gnuplot in kaart gebracht worden. Het is belangrijk voor de simulatie van TCP-trafiek dat dit congestionwindow geen onrealistische waarden aanneemt en op de juiste manier reageert op het verlies van pakketten en dergelijke.



Figuur 6: Evolutie van het congestionwindow van node 1 gedurende de simulatie

In figuur 6 is de evolutie van het congestionwindow te zien volgens de tijd. Zo is te zien dat het congestiewindow eerst zeer snel stijgt tot het ongeveer de waarde 70000 bereikt (7Mbps). Daarna maakt het een dip naar nul waarna het op tragere manier stijgt. Na de dip daarna kan men herkennen dat er



Figuur 7: IOStats van de simulatie. Zwart = node 1, rood = node 2, groen = node 3, blauw = node 4

eerst een snelle stijging is tot ongeveer de helft van de vorige top, waarna de stijging zich terug trager verder zet. Deze manier van evolutie van het congestionwindow komt overeen met de TCP Tahoe of Reno standaard.

Naast het traceren van het congestionwindow van een van de nodes werd ook een pcap-bestand aangemaakt voor elke node in het netwerk, inclusief de switch. Deze pcap-bestanden kunnen worden geanalyseerd met behulp van Wireshark. Een eerste eenvoudige analyse die men kan maken met wireshark is de hoeveelheid data en de snelheid waarmee elke node die op het netwerk heeft gezet. Deze analyse is grafisch weergegeven in figuur 7. Er is bijvoorbeeld te zien dat, hoewel er op node 2 en 4 enkel packetsinks actief waren, er toch data is verstuurd vanuit deze nodes. Dat is te wijten aan de TCP ACK berichten. Ook de eerste terugval van het congestionwindow in figuur 6 is hier duidelijk zichtbaar.

TCP	[TCP Dup ACK 1272#2]	1024 > 49153	[ACK]	Seq=0	Ack=198296	Win=65535	Le
TCP	[TCP Dup ACK 1313#1]	1024 > 49153	[ACK]	Seq=0	Ack=217592	Win=65535	Le
TCP	[TCP Dup ACK 1313#2]	1024 > 49153	[ACK]	Seq=0	Ack=217592	Win=65535	Le
TCP	[TCP Dup ACK 1313#3]	1024 > 49153	[ACK]	Seq=0	Ack=217592	Win=65535	Le
TCP	[TCP Fast Retransmission]	49153 > 1024	[ACK]	Seq=217592	Ack=0	Win=655	Le
TCP	[TCP Dup ACK 1313#4]	1024 > 49153	[ACK]	Seq=0	Ack=217592	Win=65535	Le
TCP	[TCP Dup ACK 1345#1]	1024 > 49153	[ACK]	Seq=0	Ack=239568	Win=65535	Le
TCP	[TCP Dup ACK 1345#2]	1024 > 49153	[ACK]	Seq=0	Ack=239568	Win=65535	Le
TCP	[TCP Dup ACK 1345#3]	1024 > 49153	[ACK]	Seq=0	Ack=239568	Win=65535	Le
TCP	[TCP Fast Retransmission]	49153 > 1024	[ACK]	Seq=239568	Ack=0	Win=655	Le
TCP	[TCP Dup ACK 1345#4]	1024 > 49153	[ACK]	Seq=0	Ack=239568	Win=65535	Le

Figuur 8: Duplicate ACKs en fast retransmit aangeduid in Wireshark

Om te besluiten wordt nog eens naar de reden voor de drops in het congestionwindow gekeken. Er is namelijk een onderscheid tussen TCP Reno en TCP Tahoe. Bij TCP Tahoe worden duplicate acks op een zelfde manier behandeld als een timeout, namelijk het terugbrengen van het congestionwindow naar de initiële waarde. Indien met kijkt naar de tijdstippen van de

terugval in het congestionwindow en de wireshark data op datzelfde tijdstip, zal men een patroon zien zoals in figuur 8. Hieruit kan men besluiten dat ns-3 simuleert volgens TCP-Tahoe met daarbij fast retransmit toegevoegd.

Er dient ook nog opgemerkt te worden dat de simulatie zich ook correct gedraagt wat betreft de applicatie. Er is een duidelijk onderscheid merkbaar tussen de trafiek van node 1 en node 3, het verschil in pakketgrootte en bitrate is duidelijk op te merken in de statistieken van de pcap-bestanden.

2.5.2 Tweede simulatie

De tweede simulatie werd uitgevoerd om aan te tonen hoe een WiFi-netwerk gesimuleerd wordt in ns-3. Er werd een netwerk opgezet met een access point en vier nodes die volgens een mobiliteitsmodel bewegen. Er werd gekozen voor de 802.11b standaard aangezien dit de recentste is die ns-3 ondersteunt. Om trafiek op het netwerk te simuleren werd in deze simulatie gebruik gemaakt van de on-off applicatie die aanwezig is in het ns-3 framework. Deze applicatie werd op twee van de vier nodes geïnstalleerd, de andere twee nodes kregen packetsinks om de data op te vangen.

WiFi-modellen

Een simulatie met WiFi nodes vergt meer configuratie dan de vorige die met een eenvoudig ethernet netwerk werd uitgevoerd. Er zijn namelijk meer zaken die men kan instellen: de gebruikte 802.11-standaard, het gebruik van een MAC-model met of zonder QoS (waaronder nog eens 3 modellen gekozen kunnen worden), keuze van het Rate-Control algoritme en dergelijke [19]. Het volstaat dus in dit geval niet om enkel een helper te maken die de devices installeert op de nodes. Er moet eveneens een onderscheid gemaakt worden tussen de modellen gebruikt op het access point en de losse nodes, voor access points zijn namelijk aparte modellen nodig in ns-3.

Listing 5: Configuratie en installatie van WiFi modellen en Devices

```
// Channel en PHY opzetten
YansWifiChannelHelper channel = YansWifiChannelHelper::
    Default();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default();
phy.SetChannel(channel.Create());

// WiFi helper aanmaken
WifiHelper wifi = WifiHelper::Default();
wifi.SetStandard(WIFI_PHY_STANDARD_80211b);
```

```

wifi.SetRemoteStationManager("ns3::ArfWifiManager");

//MAC helper aanmaken
NqosWifiMacHelper mac = NqosWifiMacHelper::Default();

//SSID instellen en bewaren
Ssid ssid = Ssid("netwerk-2-ssid");

//Configureren en installeren op losse nodes
mac.SetType("ns3::NqstaWifiMac", "Ssid", SsidValue(ssid),
            "ActiveProbing", BooleanValue(false));
NetDeviceContainer staDevices;
staDevices = wifi.Install(phy, mac, wifiStaNodes);

//Configureren en installeren op AP
mac.SetType("ns3::NqapWifiMac", "Ssid", SsidValue(ssid));
NetDeviceContainer apDevices;
apDevices = wifi.Install(phy, mac, wifiApNode);

```

In listing 5 is te zien hoe de devices en modellen worden geconfigureerd en geïnstalleerd op de nodes van het netwerk. Allereerst wordt er een WiFi kanaal aangemaakt wat te vergelijken is met de fysieke ether. Daarna wordt er een Model gekozen voor het simuleren van de PHY laag van het netwerk. In dit geval wordt gebruik gemaakt van het Yans model. Het PHY model moet natuurlijk beschikken over een kanaal en dit wordt toegewezen op regel 4 van de listing.

Dan pas is het tijd om de gebruikelijke helper klasse aan te maken. Optioneel wordt hier ook de standaard ingesteld die men wil gebruiken bij de simulatie, wanneer men dit niet doet zal ns-3 de 802.11a standaard gebruiken. Eveneens moet een remote station manager gekoppeld worden aan de WiFi helper, deze houdt een lijst bij waarin per node de status van die node bewaart wordt. De keuze van de remote station manager hangt van van het gebruikte rate control algoritme (in dit geval het ARF Rate algoritme).

Ook voor de MAC-laag dient een helper aangemaakt te worden, hierbij dient men de keuze te maken tussen een MAC-model dat QoS ondersteunt of niet. Een regel later wordt de SSID van het netwerk aangemaakt en onthouden zodat deze kan gebruik worden bij de configuratie van de nodes.

Voor de installatie van de devices kan gebeuren dient men ook een specifiek mac model te kiezen dat men zal gebruiken bij het installeren. Voor de losse nodes is dit het station model zonder QoS en voor het access point het ap model zonder QoS. Men kan dan uiteindelijk met behulp van de WiFi helper de devices installeren, hierbij dient men het MAC- en PHY-model mee

te geven aan de installatiemethode bovenop de gebruikelijke nodecontainer.

Mobiliteitsmodel

Een simulatie met losse WiFi nodes wordt best ook voorzien van een mobiliteitsmodel om de beweging van mobiele nodes te weerspiegelen. Ns-3 biedt hiertoe een aantal modellen aan en eveneens een helper om die modellen op de nodes te installeren. In listing 6 wordt weergegeven hoe bij deze simulatie het mobiliteitsmodel werd ingesteld en geïnstalleerd op de nodes en het access point.

Listing 6: Configuratie en installatie van het mobiliteitsmodel

```
//Losse nodes op een grid plaatsen en dan random laten  
rondlopen  
MobilityHelper mobility;  
mobility.SetPositionAllocator ("ns3::  
    GridPositionAllocator",  
    "MinX", DoubleValue (0.0),  
    "MinY", DoubleValue (0.0),  
    "DeltaX", DoubleValue (5.0),  
    "DeltaY", DoubleValue (5.0),  
    "LayoutType", StringValue ("RowFirst"));  
  
mobility.SetMobilityModel ("ns3::  
    RandomWalk2dMobilityModel",  
    "Bounds", RectangleValue (Rectangle (0, 100, 0, 100)));  
mobility.Install (wifiStaNodes);  
  
//Plaats van AP constant houden  
mobility.SetMobilityModel ("ns3::  
    ConstantPositionMobilityModel");  
mobility.Install (wifiApNode);
```

Het gebruikte mobiliteitsmodel voor deze simulatie is een tweedimensionaal raster waarop de nodes initieel geplaatst worden en daarna at random bewegen. Het raster heeft punten die vijf eenheden van elkaar liggen. Het random bewegen gebeurt in een beperkt oppervlak dat bepaald wordt door het RandomWalk2dMobilityModel, in dit geval een rechthoek tussen de punten (0,0) en (100,100).

Het installeren van het mobiliteitsmodel op de nodes wordt, zoals gebruikelijk bij een helper, gedaan met de Install methode. In deze simulatie wordt er ook aandacht aan besteed dat een access point normaal niet van plaats

verandert, deze krijgt dan ook het `ConstantPositionMobilityModel` toegewezen.

Tracing

Ook bij deze simulatie werden een aantal manieren van tracing toegepast. Zo werden er vier methodes toegevoegd voor het tracen van eigenschappen van de `WifiNetDevices`, namelijk:

PhyRxOk wordt opgeroepen telkens er een pakket succesvol is ontvangen,

PhyRxError wordt opgeroepen telkens er zich een fout voordoet bij het ontvangen van een pakket,

PhyTxDrop wordt opgeroepen als een pakket niet verzonden werd,

PhyRxDrop wordt opgeroepen als een pakket niet ontvangen werd.

Telkens een van de methodes door middel van tracing wordt aangeroepen zal het dit loggen samen met de volledige informatie van het pakket. Deze traces kunnen in de simulatie worden aan of uitgezet met een globale variabele, daar deze zeer veel logging output kunnen veroorzaken. Ze zijn vooral bedoeld om extra informatie te verkrijgen in het geval dat er iets niet naar behoren functioneert.

Een andere manier van logging die werd gebruikt is specifiek voor WiFi simulaties, men kan dan namelijk files uitschrijven die de output van de `athstats`-tool [1] nabootsen. Deze manier van tracing is even eenvoudig te implementeren als het tracen naar `pcap`-bestanden en is te zien in listing 7. Dit laatste werd eveneens in deze simulatie toegepast.

Listing 7: Tracing naar `athstats`

```
//tracing via athstats files
AthstatsHelper athstats;
athstats.EnableAthstats("athstats-sta", wifiStaNodes);
athstats.EnableAthstats("athstats-ap", wifiApNode);
```

Resultaten

In de bespreking van de resultaten van deze simulatie wordt vooral gekeken naar de `pcap`-bestanden, meerbepaald dat van het accesspoint. Er wordt vooral naar het access point gekeken omdat deze node alle trafiek van het netwerk omvat alsook de 802.11 berichten.

De simulatie bootste een periode na van twintig seconden. De On-off applicaties op node 1 en 3 werden verschillend geconfigureerd. Op node 1 schakelde deze applicatie telkens een seconde aan en dan een halve seconde uit en verstuurd pakketten van 256 bytes aan een snelheid van 6Mbps. Op node 3 schakelde de applicatie aan voor anderhalve seconde en uit voor een seconde, eveneens repetitief. Er werden pakketten van 1024 bytes verstuurd aan 5Mbps. Deze snelheden werden gekozen omdat ze, opgeteld, overeenstemmen met de theoretische maximumsnelheid van de 802.11b standaard [8]. Dit maximum wordt echter zo goed als nooit bereikt in de praktijk. De aan en uit tijdstippen van de applicaties werden zo gekozen dat er periodes van overlap, geen overlap en geen gebruik van het netwerk zouden plaatsvinden. Dit laatste in de context van collisiondetection en -avoidance.

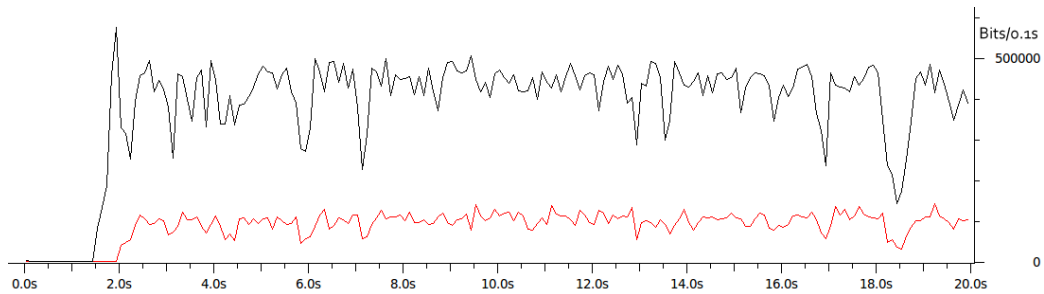
In het begin van het pcap-bestand, figuur 9, is te zien hoe de nodes zichzelf associëren met het netwerk met de correcte SSID bij het access point. Anderhalve seconde nadat de simulatie werd gestart word een eerste ARP request gedaan van node 1 om het MAC-adres van node 2 te achterhalen. Deze anderhalve seconde komt overeen met het feit dat de applicatie pas werd gestart op een seconde na het begin van de simulatie en begint in de off-status, wat dus betekent dat deze pas zijn eerste on-status bereikte na anderhalve seconde. Voor de applicatie op node 3 doet zich hetzelfde voor na twee seconden in de simulatie.

No. .	Time	Source	Destination	Protocol	Info
1	0.000000	00:00:00_00:00:05	Broadcast	IEEE 802.11	Beacon frame, SN=0, FN=0, Flags=0....., BI=25600, SSID="netwe
2	0.001722	00:00:00_00:00:04	00:00:00_00:00:05	IEEE 802.11	Association Request, SN=0, FN=0, Flags=0....., SSID="netwerk-
3	0.001732		00:00:00_00:00:04 (RA)	IEEE 802.11	Acknowledgement, Flags=0.....
4	0.002086	00:00:00_00:00:05	00:00:00_00:00:04	IEEE 802.11	Association Response, SN=0, FN=0, Flags=0....., SSID=Broadcas
5	0.002912		00:00:00_00:00:05 (RA)	IEEE 802.11	Acknowledgement, Flags=0.....
6	0.003586	00:00:00_00:00:03	00:00:00_00:00:05	IEEE 802.11	Association Request, SN=0, FN=0, Flags=0....., SSID="netwerk-
7	0.003596		00:00:00_00:00:03 (RA)	IEEE 802.11	Acknowledgement, Flags=0.....
8	0.004754	00:00:00_00:00:01	00:00:00_00:00:05	IEEE 802.11	Association Request, SN=0, FN=0, Flags=0....., SSID="netwerk-
9	0.004764		00:00:00_00:00:01 (RA)	IEEE 802.11	Acknowledgement, Flags=0.....
10	0.005862	00:00:00_00:00:02	00:00:00_00:00:05	IEEE 802.11	Association Request, SN=0, FN=0, Flags=0....., SSID="netwerk-
11	0.005872		00:00:00_00:00:02 (RA)	IEEE 802.11	Acknowledgement, Flags=0.....
12	0.006406	00:00:00_00:00:05	00:00:00_00:00:03	IEEE 802.11	Association Response, SN=1, FN=0, Flags=0....., SSID=Broadcas
13	0.007232		00:00:00_00:00:05 (RA)	IEEE 802.11	Acknowledgement, Flags=0.....
14	0.007502	00:00:00_00:00:05	00:00:00_00:00:01	IEEE 802.11	Association Response, SN=2, FN=0, Flags=0....., SSID=Broadcas
15	0.008328		00:00:00_00:00:05 (RA)	IEEE 802.11	Acknowledgement, Flags=0.....
16	0.008658	00:00:00_00:00:05	00:00:00_00:00:02	IEEE 802.11	Association Response, SN=3, FN=0, Flags=0....., SSID=Broadcas
17	0.009484		00:00:00_00:00:05 (RA)	IEEE 802.11	Acknowledgement, Flags=0.....
18	0.102370	00:00:00_00:00:05	Broadcast	IEEE 802.11	Beacon frame, SN=1, FN=0, Flags=0....., BI=25600, SSID="netwe

Figuur 9: Begin van de simulatie van netwerk twee in Wireshark, WiFi beacons en associations.

Een ander interessant aspect aan deze simulatie is de omgang van de simulator met de 802.11b standaard en diens theoretische of praktische limieten. In een praktische test met een WiFi access point ingesteld op 802.11b en een WiFi node bleek bij het versturen van een constante stream dat de praktische limiet ongeveer rond de 5Mbps lag. In de simulatie werden eveneens geen grotere snelheden dan 5Mbps aangetroffen, ook niet wanneer de twee applicaties tegelijk probeerden te verzenden aan een opgetelde maximumsnelheid

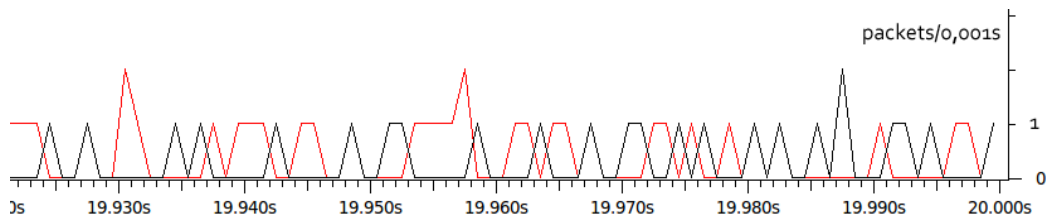
van 11Mbps. Het gemiddelde over de simulatie heen lag op 3,847 Mbps. De grafiek op figuur 10 toont een grafiek van de twee UDP stromen die werden gegenereerd door de on-off applicaties op node 1 en node 3.



Figuur 10: Grafiek met de twee UDP-stromen gegenereerd op node 1 (zwart) en node 3 (rood), de verticale as duidt het aantal bits per 0.1 seconden aan.

Het uit en aanschakelen van de applicaties is op de grafiek moeilijk zichtbaar door de hoge bitrate en het aantal pakketten dat daardoor op de sockets werd gezet die deze volgens de simulator verzonden zo gauw als dit mogelijk was. Hierdoor stopte de trafiek dus niet gelijktijdig met de applicatie.

Figuur 11 maakt de collisionavoidance van het WiFi model duidelijk. Er is te zien hoe beide nodes afwisselend een aantal pakketten verzenden maar zo goed al nooit samen, dit zou immers leiden tot collisions.



Figuur 11: Het aantal pakketten verzonden door node 1 (zwart) en node 3 (rood) per 0,001 seconde.

Het is correct hier op te merken dat het beoordelen en valideren van het WiFi-model van ns-3 zeer uitgebreid werd gedaan in een paper van Baldo et al[9].

2.5.3 Derde simulatie

De derde simulatie heeft als belangrijkste aspect het gebruik van de realtimescheduler van ns-3 in combinatie met lxc nodes. Lxc-nodes zijn nodes die

realtime runnen in een Linux container waarvan de netwerkverbinding 'aangesloten' is op de een TAPBridgeDevice in een node van ns-3. Bij deze simulatie konden dus applicaties zoals VLC worden uitgevoerd binnen de Linux containers, waarbij deze dan data door het netwerk konden zenden. Op die manier kunnen meer reële situaties worden nagebootst met behulp van ns-3.

In het licht van deze mogelijkheid werd gekozen om de simulaties eveneens in realiteit uit te voeren om zo een vergelijking te kunnen maken tussen de realiteit en de simulatie. Hiervoor werd hardware voorzien door het Expertisecentrum voor Digitale Media van de Universiteit Hasselt. Deze tests in realiteit werden eerst uitgevoerd en pas in een later stadium de simulaties in ns-3. Er werden vijf verschillende tests uitgevoerd op een netwerk gelijkend op dat van de simulatie, te zien in figuur 12.

Topologie

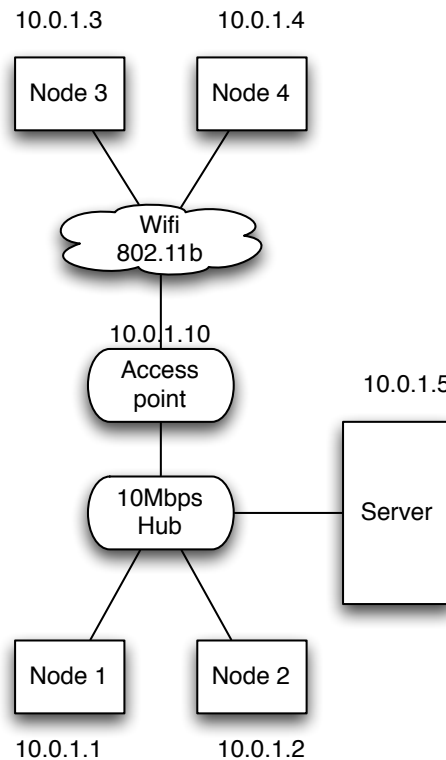
De topologie van dit netwerk bestaat uit twee nodes en een server die verbonden zijn aan een 10Mbps ethernethub, aan die hub is eveneens een WiFi 802.11b access point verbonden. Draadloze nodes verbinden zich met het netwerk via dit access point. De verdeling van de IP-adressen onder de nodes is te zien op figuur 12.

Reële hardware en software

Het reële netwerk werd opgebouwd met de volgende hardware:

- twee vaste computers met Microsoft Windows XP met Service Pack 3 (Intel Core 2 @ 1,86GHz en 2GB RAM),
- een server met Debian Lenny geïnstalleerd (Dell Poweredge R200),
- een 10Mbps ethernethub (D-Link DE-812TP+),
- een Linksys access point met DD-WRT geïnstalleerd en ingesteld op 802.11b,
- een notebook computer met Microsoft Windows XP en WiFi capaciteiten.

Op deze computers werd Wireshark geïnstalleerd, tshark op de server, om informatie over de simulatie te verzamelen. De vergelijking met de simulatieresultaten zal op basis van deze gegevens gedaan worden. Eveneens



Figuur 12: Topologie van netwerk-3.cc

werd een versie van VLC Media Player (1.1.2) geïnstalleerd om mediastreams aan te maken en af te spelen. Op de server werd een oudere versie (0.8.6h) geïnstalleerd om compatibel te zijn met de Debian Lenny repositories. Op de server werd eveneens een HTTP-server geïnstalleerd om HTTP-transfers mogelijk te maken, namelijk Apache 2.2.9 voor Debian.

Simulatie software

De simulatie werd uitgevoerd in een Ubuntu 10.04 omgeving. De nodes werden gesimuleerd in Linux Container waarvan enkel de netwerkconfiguratie werd aangepast. Ook op dit systeem werd gebruik gemaakt van VLC Media Player voor de mediastreams, meerbepaald versie 1.0.6 uit de Ubuntu Repositories. Op de server werd gebruik gemaakt van Apache 2.2.14 voor Ubuntu om HTTP-transfers mogelijk te maken. Voor het initiëren van HTTP-transfers in commandline werd gebruik gemaakt van Wget 1.12.

Voor het gebruik van lxc en de TAPBridgedevices moeten ook een aantal

softwarepakketten geïnstalleerd worden op het systeem. De vereiste pakketten voor Ubuntu zijn: `lxc`, `uml-utilities` en `bridge-utils`. De versies van de pakketten gebruikt bij de simulatie zijn respectievelijk 0.6.5-1, 20070815-1.1ubuntu2 en 1.4-5ubuntu2.

Aanmaken van de simulatie

De simulatie werd, net zoals de vorige twee besproken simulaties, beschreven in een C++ bestand zoals gebruikelijk bij `ns-3`. Echter werden, voor het simuleren met `lxc` een aantal toevoegingen en aanpassingen gedaan die in eenvoudigere simulaties niet voorkomen. Een belangrijke toevoeging bevindt zich aan het begin van de code, men moet namelijk gebruik maken van de realtime scheduler. Dat moet men aangeven aan het framework aangezien de standaard instelling niet realtime is. De code hiervoor is weergegeven in listing 8. Er werden eveneens twee scripts geschreven om de virtuele taps, `lxc`-containers en dergelijke snel aan te maken of te verwijderen.

Listing 8: Instellen van realtime scheduler met checksum

```
GlobalValue::Bind (" SimulatorImplementationType" ,  
    StringValue (" ns3:: RealtimeSimulatorImpl" ));  
GlobalValue::Bind (" ChecksumEnabled" , BooleanValue ( true ));
```

Listing 9: Vereenvoudigd bash script voor het opzetten van een linux container met `bridge` en `tap` voor `ns-3`

```
#!/bin/bash  
brctl addbr br-node1  
tunctl -t tap-node1  
ifconfig tap-node1 0.0.0.0 promisc up  
brctl addif br-node1 tap-node1  
ifconfig br-node1 up  
pushd /proc/sys/net/bridge  
for f in bridge-nf-*; do echo 0 > \${f}; done  
popd  
lxc-create -n node1 -f lxc-node1.conf
```

De code in listing 9 is een uittreksel uit het script dat de `lxc` containers voor de simulatie aanmaakt. Het script maakt de `bridge`, `tap` en `lxc` container aan voor `node 1` en configureert deze voor gebruik. De naam van de `tunctl tap` die hier in voorkomt moet overeenkomen met die in de code van de simulatie. Dit script dient als `superuser` uitgevoerd te worden omdat voor een aantal bewerkingen zoals het creëren van taps en bridges root privileges benodigd zijn. Eveneens dient een `cgroup` gemount te zijn, dit kan men doen door

```
sudo mount -t cgroup cgroup /cgroup
```

uit te voeren. Het verwijderen van de lxc containers, taps en bridges kan eveneens gedaan worden met een script dat sterk lijkt op dat in listing 9.

Bij een simulatie waar men gebruik maakt van lxc containers moeten deze ook worden voorgesteld als nodes in het netwerk. De Topologie van het netwerk wordt bijgevolg op dezelfde wijze opgezet als in andere ns-3 simulaties. Het verschil bevindt zich na het aanmaken van de topologie. Op elke node die zal worden gesimuleerd met behulp van lxc, wordt een extra device geïnstalleerd, het TAPBridgeNetDevice. Dit device verzorgt de koppeling tussen de ns-3 node en de lxc-container. Het device kan op een aantal manieren worden geconfigureerd, bij deze simulatie werd gekozen voor de UseBridge configuratie. De UseBridge configuratie maakt namelijk gebruik van een bridge tussen de lxc container en de simulatie, op die manier worden MAC-adressen op juiste manier vertaald en gerouteerd over het gesimuleerde netwerk. Een uitgebreide uitleg over de verschillende configuraties is te vinden op de wiki van ns-3 [17]. Een schematische weergave van de werking van UseBridge is te zien in figuur 13.

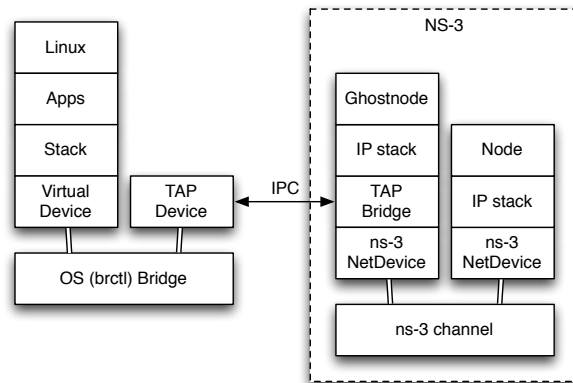
Listing 10: Toevoegen van de connectie tussen een ns-3 node en een lxc container

```
TapBridgeHelper tapBridge;  
tapBridge.SetAttribute("Mode", StringValue("UseBridge"));  
tapBridge.SetAttribute("DeviceName", StringValue("tap-node1  
"));  
// connectie op node 1 met device 0  
tapBridge.Install(csmaNodes.Get(0), csmaDevices.Get(0));
```

Vooraleer men een TapBridgeDevice op een node kan installeren moet men de mode en, in geval van UseBridge, de naam van de TAP opgeven. Dit kan men eenvoudig doen met behulp van de TapBridgeHelper zoals te zien is in listing 10. Bij het installeren moet men aangeven op welke node het device wordt aangebracht en met welk ns-3 NetDevice het connectie maakt, in dit geval het eerste device en de eerste node van de simulatie.

Een bijkomend verschil met eenvoudige simulaties is het feit dat de nodes die worden gesimuleerd met behulp van Linux containers geen adressen krijgen toegewezen in de code van de simulatie. Het toewijzen van adressen gebeurt in het configuratiebestand dat gebruikt wordt bij het opzetten van de lxc container. Het configuratie bestand bepaald ondermeer welk netwerkapparaat er gebruikt wordt door de Linux container en hoe dit geconfigureerd dient te worden.

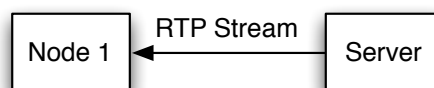
Meer informatie en uitleg over het implementeren en opstarten van een eenvoudige simulatie met lxc containers is te vinden op de wiki van ns-3 [14].



Figuur 13: Schematische weergave van de werking van het Tap Bridge model in UseBridge configuratie

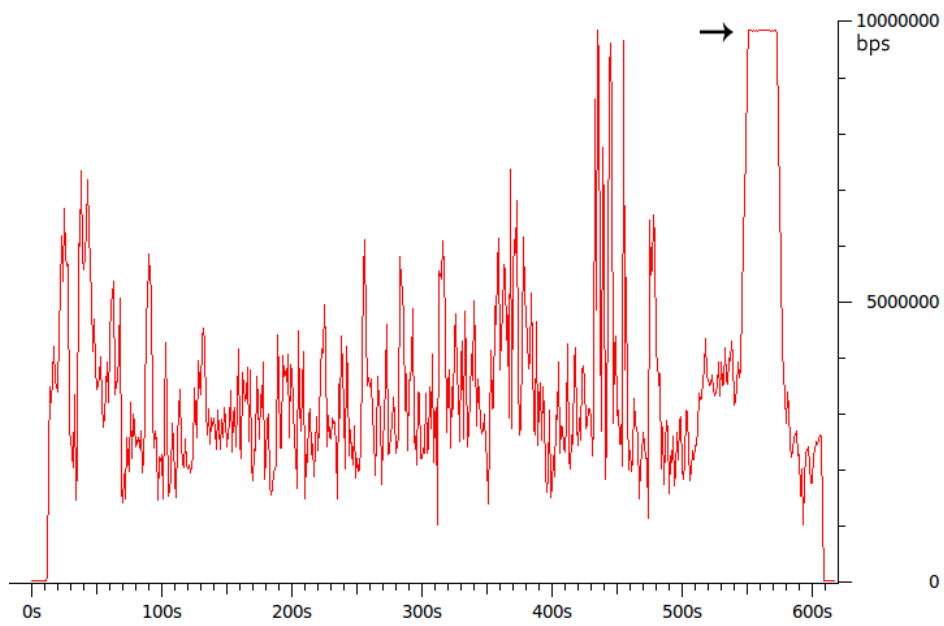
Eerste scenario

Het eerste scenario, weergegeven in figuur 14, is zeer eenvoudig. de server in het netwerk stuurt een RTP-stream met variabele bitrate naar node 1 in het netwerk. Het interessante is echter dat deze stream door zijn variabele bitrate soms pieken maakt van bitrates boven de 10Mbps, wat boven de limiet van de hub in het netwerk ligt. De gebruikte media voor het streamen is een 480p versie van de kortfilm Big Buck Bunny, gedownload van de website [2] in h264-formaat. De video duurt 9 minuten en 56 seconden. Het scenario loopt, als gevolg van de RTP-stream, tien minuten voor deze wordt afgebroken.

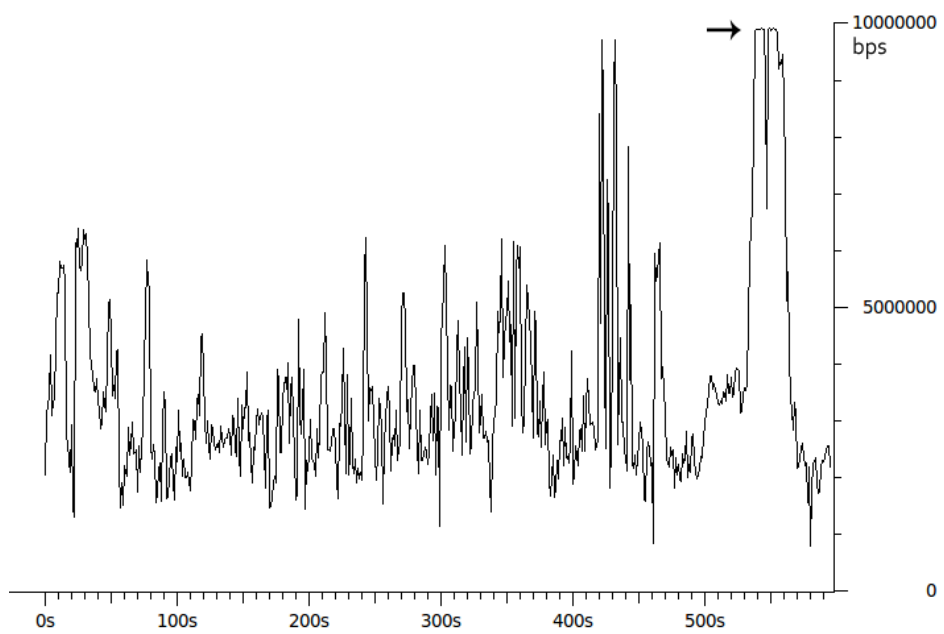


Figuur 14: Eerste scenario schematisch weergegeven

In de test op het netwerk in realiteit werd met behulp van Wireshark vastgesteld dat ongeveer 2000 pakketten van de stream verloren gingen, dat komt overeen met ongeveer 1% van de totale stream. Wanneer men met Wireshark een grafiek maakt (figuur 15) van de pakketten die werden ontvangen op node 1 is duidelijk te zien hoe pieken boven de 10Mbps werden afgeknot als gevolg van de beperkingen van de hub in het netwerk. Dit is vooral duidelijk tussen



Figuur 15: Grafiek van ontvangen bits per seconde op node 1 bij het eerste scenario



Figuur 16: Grafiek van ontvangen bits per seconde op node 1 bij het eerste scenario volgens de simulatie

de 540ste en 600ste seconde. Na het abstraheren van de ontvangen stream uit het pcapbestand, kon dit worden afgespeeld met behulp van VLC Media Player. Er waren, op tijdstippen dat er afknotting zichtbaar is op de grafiek, duidelijke artefacten te zien in het beeld van de stream.

In de test op het netwerk in simulatie werd eveneens met behulp van Wireshark gewerkt, zij het dan de command line variant tshark. Er werden van beide nodes pcap-bestanden bewaard. Ook hier werd verwacht dat bij de pieken aan het einde van de stream, er verlies van pakketten zou optreden.

Na het uitvoeren van de simulatie, kon met behulp van Wireshark geconstateerd worden dat er 3804 pakketten van de stream verloren gingen. Het verlies komt overeen met 1,8% van de totale stream. Er was dus een lichtjes hoger verlies van pakketten in de simulatie t.o.v. de realiteit. Eveneens is op te merken dat wanneer men een grafiek maakt, analoog aan die in figuur 15, er zich een duidelijke afknotting laat opmerken rond de lijn van 10Mbps. Deze grafiek is te zien in figuur 16 en werd gemaakt aan de hand van data van de ontvangende node.

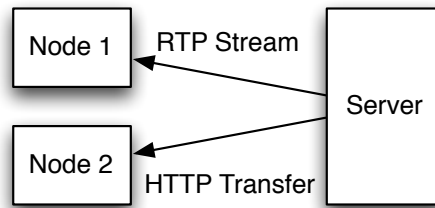
Er kan dus, hoewel er zich kleine verschillen voordoen, besloten worden dat de resultaten van de reële uitvoering sterk aanleunen bij deze van de simulatie. Bijgevolg kan deze simulatie als relatief accuraat worden bevonden. Echter dient opgemerkt te worden dat het hier om een algemeen beeld gaat en minder om details. Bovendien dient te worden verduidelijkt dat de kleine dip (seconde 540) in figuur 16 zich eveneens voordeed aan zendende zijde en waarschijnlijk geen gevolg is van de simulatie maar eerder van de verschillende versies van VLC.

Tweede scenario

Het tweede scenario, weergegeven in figuur 17, bouwt verder op het eerste. Ook hier zal een RTP-stream van de server naar node 1 van toepassing zijn. Echter wordt in dit scenario een tweede datastroom toegevoegd, namelijk een HTTP-transfer van de server naar node 2. De HTTP-transfer werd toegevoegd om het gedrag van UDP t.o.v. TCP en tegenovergesteld te kunnen opmerken, daar RTP over UDP werkt en HTTP over TCP.

Voor de RTP-stream werd dezelfde data gehanteerd als in het eerste scenario. Voor de HTTP-transfer werd Apache 2 geïnstalleerd op de server met standaardinstellingen. Op node 2 werd gebruik gemaakt van de webbrowser Google Chrome voor het downloaden van het bestand van de server.

Na het uitvoeren van de test op het reële netwerk konden een aantal zaken worden geconstateerd met behulp van Wireshark. Ten eerste is er opnieuw een verlies van pakketten op te merken in de RTP-stream ten gevolge van



Figuur 17: Tweede scenario schematisch weergegeven

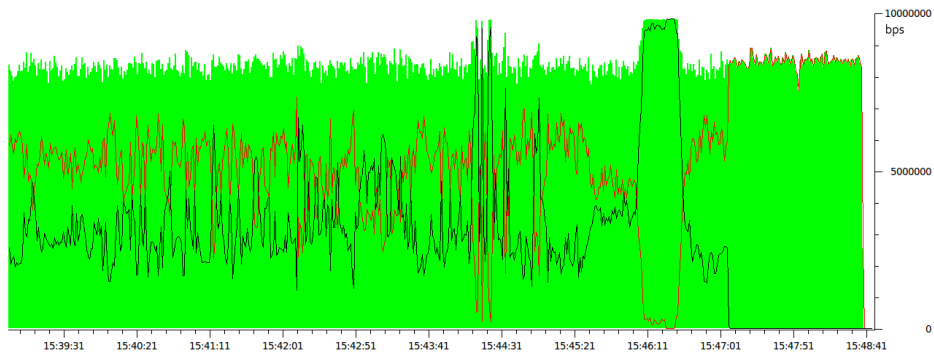
de verbinding van 10Mbps. Dit verlies bedraagt 1,3% van het totaal aantal pakketten, dit is ongeveer gelijk aan dat van het vorige scenario en werd dus niet beïnvloed door de toevoeging van een TCP-stroom. Eveneens kan grafisch zeer goed geconstateerd worden dat de RTP-stream voorrang krijgt op de TCP-traffic, de TCP-traffic neemt enkel de overgebleven bandbreedte in. Dit is zeer duidelijk te zien aan het einde van figuur 18, wanneer de UDP-traffic hoge pieken vertoont duikt de TCP-traffic omlaag, wanneer de RTP-stream gestopt is neemt de TCP-traffic opnieuw de volledige bandbreedte op.

Ook deze test werd uitgevoerd op het gesimuleerde netwerk. De HTTP-transfer werd echter gedaan met Wget aangezien de Linux containers enkel een CLI aanbieden en dus geen Google Chrome kunnen uitvoeren. Uit pcap-bestanden viel te concluderen dat er van de RTP-stream 4,4% van de pakketten verloren ging. Dat is een lichtjes hoger getal dan bij de reële uitvoering van het scenario. Het algemene beeld, te zien op figuur19, toont veel gelijkenissen met dat van de reële uitvoering. Ook hier is te zien hoe de stroom aan UDP pakketten die van TCP pakketten verdrukt en TCP dus afhankelijk is van de hoeveelheid UDP traffic. Er dient bij de figuur ook opgemerkt te worden dat de TCP traffic tijdens de simulatie een aantal keer werd stopgezet om de grootte van de pcap-bestanden te beperken.

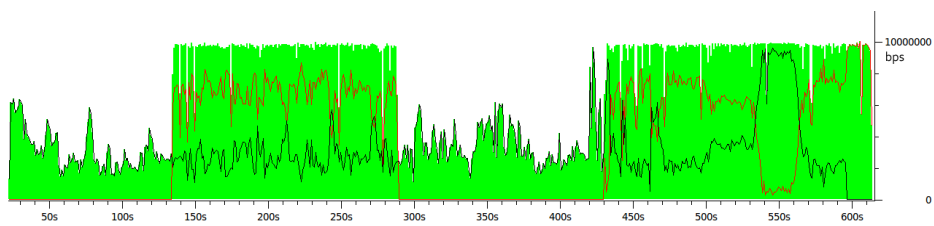
Hoewel de verschillen tussen de realiteit en de simulatie bij dit scenario iets groter zijn, te zien aan het pakketverlies en de totale traffic op het netwerk, kan de conclusie worden gemaakt dat de simulatie redelijk accuraat is. Dit omdat het algemene beeld bij de twee uitvoeringen van het scenario zo goed als gelijk is en er zich geen onoverkomelijke anomalieën voordoen.

Derde scenario

In een derde scenario werd verder gewerkt op basis van het tweede scena-

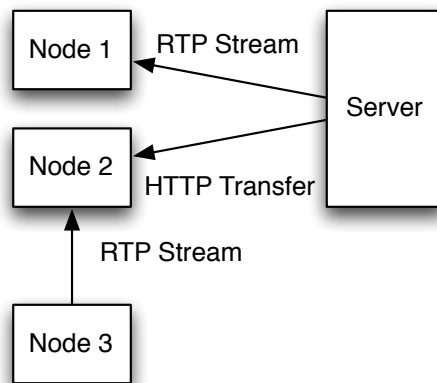


Figuur 18: Grafiek met UDP-traffic (zwart) en TCP-traffic (rood) in het tweede scenario. Het groene oppervlak duidt de totale hoeveelheid traffic op het netwerk aan.



Figuur 19: Grafiek met UDP-traffic (zwart) en TCP-traffic (rood) in het tweede scenario tijdens simulatie. Het groene oppervlak duidt de totale hoeveelheid traffic op het netwerk aan.

rio. Er werd een RTP-stream toegevoegd die werd verstuurd vanaf de WiFi node 3 naar node 2. Op node 2 kwamen dus twee trafieken binnen, enerzijds een RTP-stream en anderzijds een HTTP-transfer. Het scenario wordt weer-gegeven in figuur 20. De RTP-stream die werd gestuurd van node 3 naar node 2 is dezelfde als deze die in de vorige twee scenario's voorkwam.



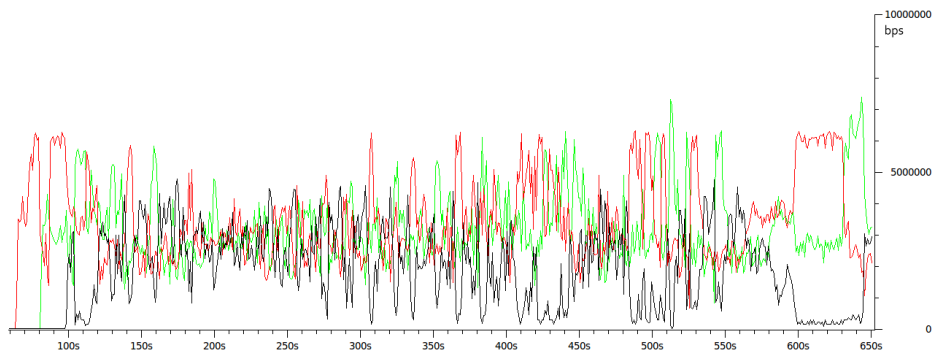
Figuur 20: Derde scenario schematisch weergegeven

In de reële test van dit scenario kwamen dezelfde zaken tot uiting als bij de uitvoering van het tweede scenario, namelijk de onderdrukking van TCP-traffic door UDP-traffic. Echter zijn hier twee RTP-streams aanwezig en is er ook een strijd tussen deze twee stromen om bandbreedte. Deze strijd levert vooral op dat beide streams bandbreedte moeten prijsgeven aan elkaar. Bovendien is op te merken dat de TCP-traffic over nog minder bandbreedte kan beschikken door de verhoogde aanwezigheid van UDP-traffic. Dit alles is te zien in de grafiek in figuur 21

Dit scenario simuleren bleek een moeilijker opgave dan de vorige twee besproken scenario's. Deze moeilijkheid was te wijten aan een probleem met het WifiNetDevice en diens samenwerking met TAPBridge, het probleem is uitgebreid beschreven in sectie 3.1.1.

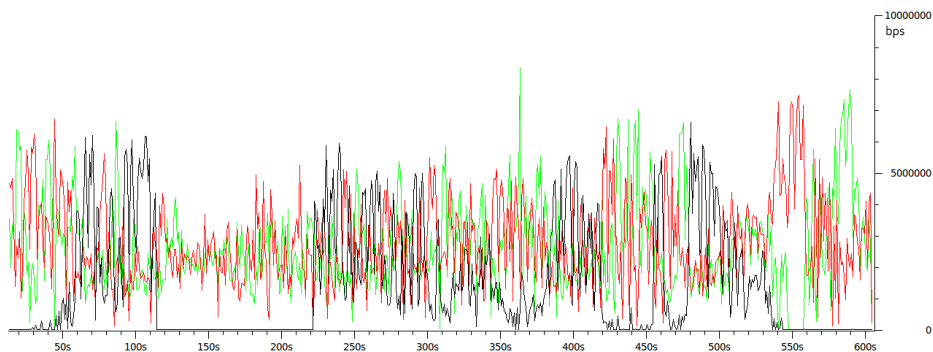
Zoals in die tekst te lezen staat, werd er gezocht naar een workaround om de simulatie uit te voeren en werd deze ook gevonden in de vorm van een gesimuleerde applicatie op de node. Dit heeft vooral tot gevolg dat de RTP-stream van node 3 naar node 2 niet representatief kon worden gesimuleerd aangezien deze werd vervangen door een minder accuraat gesimuleerde UDP-stream gebaseerd op timestamps en pakketgroottes.

De resultaten van de simulatie van dit scenario moeten dus meteen als minder accuraat bekeken worden t.o.v. de vorige simulaties. Er is dan ook



Figuur 21: Grafiek met TCP-traffic (zwart) en RTP-traffic van node 3 naar 2 (rood) alsook RTP-Traffic van server naar node 1 (groen) in het derde scenario tijdens test op het netwerk.

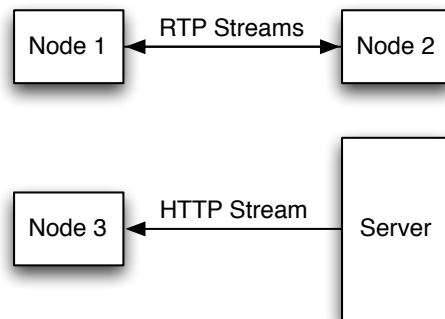
een terechte twijfel over de representativiteit van deze simulatie aangezien de applicatie in de simulatie werkt met de realtime scheduler en er geen zekerheid is over de stiptheid van verzending van de pakketten door deze scheduler. In de resultaten is ook op te merken dat er veel minder UDP pakketten op te merken zijn in de stream van node 3 naar 2 dan deze in realiteit. Een grafiek van de opgenomen bandbreedte door de stromen is te zien in figuur 22.



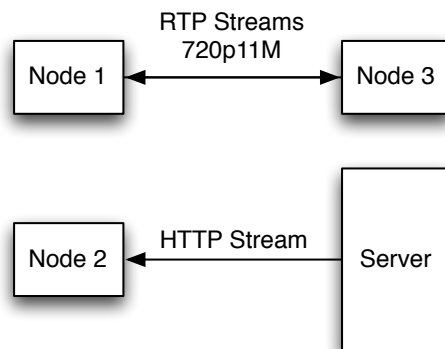
Figuur 22: Grafiek met TCP-traffic (zwart) en UDP-traffic van node 3 naar 2 (rood) alsook RTP-Traffic van server naar node 1 (groen) in het derde scenario tijdens simulatie.

Vierde en vijfde scenario

Op de testopstelling werden nog twee andere scenario's uitgevoerd, weergegeven in figuren 23 en 24. Het belangrijke concept in deze scenario's is de aanwezigheid van RTP-streams heen en terug tussen twee nodes. Echter was de mogelijkheid tot uitvoering van deze scenario's op de simulator niet aanwezig door de beperkingen van de workaround voor het in sectie 3.1.1 beschreven probleem. Er werden echter wel pcap-bestanden bijgehouden van deze scenario's in realiteit en deze zijn te vinden op de digitale kopie die gepaard gaat met dit eindwerk.



Figuur 23: Vierde scenario schematisch weergegeven



Figuur 24: Vijfde scenario schematisch weergegeven

3 Bespreking

In deze sectie zullen de voor- en nadelen van ns-3 worden besproken. Ook problemen die tijdens het werken aan deze bachelorproef naar boven kwamen worden aangehaald. Dit om in sectie 4 tot een conclusie te komen, gebaseerd op informatie over de mogelijkheden en toekomst van ns-3 alsook de eigen ervaringen bij het implementeren en uitvoeren van de simulaties. Het is echter zo dat die conclusie maar beperkt in de tijd geldig zal blijven, gezien de steeds verdere ontwikkeling van het project door diens ontwikkelaars.

3.1 Ondervonden moeilijkheden

Er bestaat geen enkel systeem dat geen kwalen heeft, ook ns-3 heeft dus een aantal minpunten. In dit deel van de conclusie worden minpunten en problemen aangehaald die werden ondervonden tijdens het implementeren, uitvoeren of analyseren van de simulaties. Mogelijke en/of gebruikte oplossingen worden eveneens aangehaald.

Een van de beste manieren voor het zoeken naar een oplossing voor een probleem met ns-3 is het nakijken van voorbeeldcode in de examples map, het opzoeken van klassen in de doxygen en het raadplegen van de usergroup. Deze laatste is een groep personen die allen ns-3 gebruiken en mogelijke oplossingen of antwoorden op vragen kunnen aanrijken die uit de relatief beperkte documentatie niet meteen zijn af te lijden.

3.1.1 TapBridge en WiFi devices

Het grootste probleem dat werd ondervonden tijdens het werken aan deze bachelorproef is de samenwerking, of eerder het gebrek eraan, van het TapBridgeDevice en WifiNetDevices. Zoals te zien is in figuur 12 bevatte de topologie van de derde simulatie een node die via een WiFi kanaal en een access point verbonden was met de rest van het netwerk. Het was de bedoeling dat ook deze node in de simulatie zou gesimuleerd worden met behulp van een lxc-container. Aangezien voorbeelden in de Examples/Tap map samen met een uitleg op de wiki van ns-3 het idee creëerden dat dit zonder enig probleem mogelijk was, werd er vanuit gegaan dat dit ook zou werken in het netwerk van de derde simulatie.

Door planning werden eerst de simulaties op het reële netwerk uitgevoerd, waarbij vanaf het derde scenario een WiFi node werd betrokken. Er werden in totaal vijf scenario's uitgevoerd.

Echter bleek later dat bij het proberen uitvoeren van deze scenario's op het gesimuleerde netwerk, er zich een probleem stelde. Hoewel genereren van

trafiek over het ethernetgedeelte van het netwerk zonder problemen verliep, bleek het onmogelijk om pakketten uit te wisselen tussen de WiFi node en de rest van het netwerk. Dit hoewel er zich geen enkele warning of error had voorgedaan tijdens het compileren of uitvoeren van de simulatie.

Tijdens het zoeken naar een oplossing werd zowel gekeken naar de documentatie van ns-3 alsook de usergroup. Er bleek dat dit probleem en gerelateerde vragen al een aantal keer eerder aangehaald waren op de usergroup. Het probleem zou zich bevinden in de manier waarop TapBridge en de verschillende modes daarvan werken. Zo bleek het gebruik van UseBridge niet verenigbaar te zijn met het koppelen van de TapBridge aan een WifiNet-Device en werd aangeraden UseLocal als mode te gebruiken. Echter bleek ook dit geen degelijke oplossing voor het probleem te zijn, daar er nog steeds geen trafiek mogelijk was tussen de WiFi node en het netwerk.

Er werd toen besloten een vraag te plaatsen in de usergroup [4]:

Hi,

I'm running a simulation with linux containers as explained in the wiki.

It consists out of a few csma nodes connected to a switch and an ap with wifi nodes, the ap is also connected to the switch. When I run the simulation I am able to ping from one csma node to another one but I can't get any connection with the wifi nodes. pinging between the wifi nodes also doesn't work.

Can somebody take a look at this code please, i can't figure out why the wireless doesn't work?

Na wat aandringen kwam er een duidelijk antwoord op de vraag, waaruit bleek dat er geen overduidelijke oplossing was en er zou moeten gekeken worden naar het aanpassen van de code van het ns-3 raamwerk zelf:

I am afraid that you cannot use UseLocal mode when trying to send/receive the packets to/from lxc containers.

UseBridge is needed to bridge the network card in container with the ns-3.

But STA nodes don't support UseBridge very well. This is the dilemma. Maybe the thread that Nikola mentioned can help you to understand this question

Anyway, a code hacking may be needed.

Er werd ook gezocht naar een workaround. Daarmee wordt bedoeld dat het probleem in weze niet werd opgelost maar er naar een manier werd gezocht om de simulatie toch uit te voeren. Dit houdt in dat de datasromen die in realiteit werden gegenereerd op een andere manier worden nagebootst op het netwerk. Een mogelijkheid daartoe is het gebruiken van een statistisch model van de stroom en dit implementeren in een applicatie. Een andere mogelijkheid is het inlezen van een pcap-bestand waarin de stroom vervat zit. Echter zijn dit enkel oplossingen voor het verzenden en niet voor het ontvangen. Voor het ontvangen kan een packetsink een oplossing kunnen zijn.

Er werd uiteindelijk de keuze gemaakt om een applicatie te implementeren op de WiFi node die een bestand kon inlezen waarin timestamps en pakketgroottes staan. Dit bestand kon gegenereerd worden met behulp van tcpdump en awk door het volgende commando uit te voeren:

```
tcpdump -e -l -tt -n -r stream.pcap |  
awk '// { printf "%s %s\n", $1, $9}' | sed 's:///' > stream.txt
```

De applicatie leest vanaf de start telkens een lijn in uit het bestand. Het houdt de laatste timestamp bij en berekent aan de hand van de vorige timestamp en de huidige het verschil in tijd tussen de twee pakketten. Aan de hand van dat verschil en de pakketgrootte zal dan op het juiste moment een pakket op de socket gezet worden voor verzending. Deze workaround werd gebruikt bij het simuleren van het derde scenario (zie pagina 45).

De workaround heeft echter het nadeel dat er geen RTP-stream wordt gegenereerd maar een UDP-stream met dummy data. Ook het gebruik van een socket en de scheduler van ns-3 verzekeren niet dat het verzenden correct is in vergelijking met die op een lxc-node. De workaround laat dus simulatie toe maar de geldigheid van de resultaten is eerder twijfelachtig.

3.1.2 Errormodel en WiFi devices

Een ander probleem dat werd ondervonden is het gebrek aan de mogelijkheid tot toevoegen van een errormodel aan WiFi devices. Dit is bijvoorbeeld mogelijk bij csma en p2p devices, het toevoegen van een errormodel kan interessant zijn om bepaalde anomalieën op te wekken in de gedragingen van het netwerk. Het was de bedoeling bij de tweede simulatie een errormodel toe te voegen aan een van de devices dat voor een verlies van ontvangen pakketten zou zorgen met behulp van een random variabele.

Er blijkt echter geen attribuut te zijn in het WifiNetDevice dat dit toelaat zoals bij andere NetDevices en bijgevolg bleek dit dan ook niet mogelijk te zijn. Ook op de usergroup werd duidelijk gemaakt dat dit niet mogelijk is en men dit enkel kan oplossen door het schrijven van een nieuw PHY model [3].

3.1.3 Gebrek aan documentatie

Dit is eerder een algemeen probleem bij ns-3 en vormde vooral een moeilijkheid in het begin. De documentatie van ns-3 bevindt zich in vier verschillende bronnen: een wiki, een doxygen, een manual en een tutorial. De tutorial is zeer goed voor wanneer men begint met ns-3 maar legt spijtig genoeg maar een deel van de simulator uit, een aantal zaken zoals de bijvoorbeeld de realtime scheduler worden over het hoofd gezien.

De Manual was zeer incompleet, hoewel er een duidelijke structuur is in het document zijn er veel hoofdstukken die tot nu toe simpelweg een placholder zijn en dus geen informatie bevatten. Echter zijn er sinds het begin aan dit werk wel een aantal toevoegingen gedaan.

De doxygen bevat goede informatie over de modules en modellen en is zeer nuttig voor het opzoeken van functies, klassen en dergelijke. Echter dient hierbij opgemerkt te worden dat een zeer groot deel van die klassen en functies niet gedocumenteerd zijn en men dus aangewezen is op louter de structuur ervan.

Tenslotte is er de wiki, deze is op een vreemde manier gestructureerd en daardoor zijn bepaalde zaken moeilijk vindbaar. Het is ook vrij onduidelijk wat men juist in de wiki kan vinden en welke zaken men in de doxygen pagina's moet zoeken. Echter was het een goede hulp om informatie te vinden over het ns-3 project en bepaalde toepassingen ervan.

Het gebrek aan documentatie en de vreemde structuur ervan leidt er soms toe dat er veel langer moest worden gezocht naar bepaalde zaken dan eigenlijk nodig zou moeten zijn. Dit brengt bijgevolg tijdverlies met zich mee. Een bijkomend nadeel is dat men over sommige zaken geen informatie kan vinden en men aangewezen is tot trial-and-error methoden en het bekijken van broncode van het ns-3 raamwerk.

3.2 Voordelen van ns-3

Er zijn een aantal niet te miskennen positieve punten op te merken aan ns-3. In deze sectie worden deze uiteengezet en geargumenteed.

3.2.1 Performantie

Een eerste positief gegeven bij ns-3 is dat het een van de meer performante simulators is van de nieuwere generatie, dit werd eveneens al opgemerkt in sectie 1.4.9 en in de paper van Elias Weingartner [12]. Vooral voor complexere simulaties kan het van groot belang zijn dat de simulator performant genoeg is om deze behoorlijk uit te voeren. Ook voor hybride simulaties zoals de derde

van deze paper is dit van belang, een simulator die de realtime scheduling niet kan volgen door een gebrek aan performantie is in zo'n situatie waardeloos.

3.2.2 Hybride simulaties

De mogelijkheden die ns-3 kan voorleggen voor interactie met de reële wereld en op die manier hybride simulaties mogelijk te maken vormen een sterk pluspunt voor het project. Het laat toe van een mix te maken van zowel reële software als hardware met de simulator. Op die manier kunnen bijvoorbeeld software of hardware getest worden zonder al te hoge kosten aan testbeds en dergelijke. Het schept eveneens de mogelijkheid om realistischere simulaties uit te voeren door het gebruik van reële applicaties.

3.2.3 Commandline parameters

Een kleine doch zeer nuttige feature van ns-3 is de mogelijkheid om in een simulatie een aantal parameters te definiëren die aanpasbaar zijn met commandline argumenten. Op die manier kan men bijvoorbeeld met behulp van een script een bepaalde simulatie veelvuldig uitvoeren met verschillende parameters om de resultaten hiervan te vergelijken. Op die manier hoeft men niet telkens de code van de simulatie aan te passen. Voorbeelden van dergelijke parameters zijn het aantal nodes, de bandbreedte van een verbinding, de foutverdeling, ...

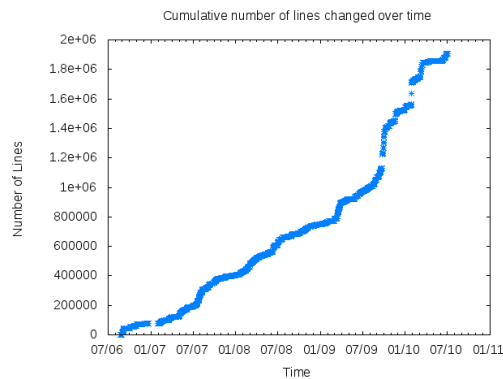
3.2.4 Hoeveelheid aan modellen

De hoeveelheid aan verschillende modellen die beschikbaar zijn na acht iteraties van ontwikkeling zijn een pluspunt voor ns-3. Naast traditionele netwerkinfrastructuur kan met met ns-3 eveneens zaken zoals Wimax en MESH netwerken simuleren. Het feit dat er nog volop toevoegingen worden gedaan aan de lijst met modellen kan enkel betekenen dat er nog meer mogelijkheden zullen zijn in de toekomst. Echter dient wel opgemerkt te worden dat sommige modellen eerder oudere protocollen of hardware simuleren, een voorbeeld daarvan is bijvoorbeeld de achterstand op de 802.11 standaarden waarbij de g en n standaard nog niet in ns-3 vervat zijn.

3.2.5 Veelbelovende roadmap en constante ontwikkeling

Zoals in de vorige sectie al is vermeld, is er nog steeds ontwikkeling aan de gang in het ns-3 project. Op de homepage van het ns-3 project wordt er tevens een goed beeld van gegeven met behulp van een grafiek, te zien in

figuur 25. Deze grafiek duidt het cumulatief het aantal lijnen code aan in functie van de tijd.



Figuur 25: Cumulatief aangepast of toegevoegd aantal lijnen code in ns-3 t.o.v. de tijd

Op de wiki van het ns-3 project is eveneens een roadmap te vinden alsook een pagina met de huidige ontwikkelingen. Dit zijn tevens aanzienlijke lijsten met features zoals visualisatie, grafische editors alsook nieuwe modellen en simulatiemogelijkheden. Dit alles wijst erop dat ns-3 een gezond project is en bijgevolg zal verbeteren en vergroten naar de toekomst toe. Wat natuurlijk als positief kan bekeken worden.

3.2.6 Actieve community

Dit is niet zo zeer een eigenschap van de simulator zelf maar draagt toch bij aan het gebruik ervan. Indien met met vragen en/of problemen zit die men niet met behulp van de documentatie kan oplossen is er altijd de usergroup. Hier kan men vragen stellen en heeft men redelijk snel een behelpende reactie of verwijzing naar een eerder geopende thread die het probleem kan verklaren.

3.2.7 Meerdere mogelijkheden voor tracing

De idee van tracing is bij simulaties van groot belang indien men informatie wilt onttrekken om een analyse te kunnen maken. Ns-3 biedt hier een aantal verschillende mogelijkheden aan die elk hun voor- en nadelen hebben, het beschikken over een pakket van meerdere mogelijkheden tot het verzamelen van gegevens kan de analyse enkel ten goede komen en is dus een troef. Ns-3 biedt namelijk een volledig tracing systeem met callbackfuncties

aan alsook het uitschrijven van netwerktrafiek naar een aantal verschillende bestandsindelingen zoals pcap-bestanden en athstats.

3.3 Nadelen van ns-3

Buiten de opgelopen moeilijkheden opgesomd en besproken in sectie 3.1 zijn er nog een aantal andere zaken die als nadelen aan ns-3 kunnen beoordeeld worden. In deze sectie worden de belangrijkste nadelen aan ns-3 besproken.

3.3.1 Gebreken bij WiFi simulaties

Aangezien WiFi een complexer model is dan bijvoorbeeld ethernet of p2p, zou men kunnen verwachten dat hier ook meer kans is op problemen bij simulaties. Hoewel de paper van Baldo [9] zich licht positief uitspreekt over de validatie van het model aanwezig in ns-3, dienen toch enkele belangrijke opmerkingen gemaakt te worden.

Zoals te lezen is in secties 3.1.1 en 3.1.2 zijn er zaken die, hoewel ze met andere devices en modellen soepel werken, niet werken met WiFi devices. Dit kan enkel als negatief aanzien worden omdat ze de mogelijkheden bij het simuleren van WiFi netwerken serieus inperken.

Ook het ontbreken van modellen voor de recentere 802.11-standaarden kan als een ernstig gebrek aanzien worden bij het ns-3 project. Zeven jaar na de introductie van 802.11g is deze standaard niet beschikbaar in de simulator, voor het simuleren van moderne infrastructuur is dit nefast.

3.3.2 Geen mogelijkheid tot visualisatie

Een groot gebrek aan ns-3 is het, tot op heden, ontbreken van een eenvoudige manier voor het visualiseren van simulaties. Hoewel er een extern python project bestaat dat dit probeert te verwezenlijken is er momenteel nog geen enkele manier om dit te doen. Het visualiseren van netwerkstromen en dergelijke kan een betere kijk geven op de situatie in een netwerk en dus een positieve invloed hebben op de correctheid van de analyse ervan.

3.3.3 Documentatie

In sectie 3.1.3 werd reeds een bespreking gedaan over de documentatie van ns-3 en het gebrek aan een duidelijke structuur en volledigheid ervan. Echter moet hier toch bij worden opgemerkt dat gedurende de tijd vorderde er wel verbeteringen zijn aangebracht en er dus mag verwacht worden dat deze

situatie verder verbetert. Echter is de documentatie op dit moment nog niet volledig in orde en blijft het een serieus nadeel bij het gebruik van ns-3.

3.4 Samenvattende tabel van voor- en nadelen

Een samenvattende tabel van de voor- en nadelen aan ns-3 wordt weergegeven in tabel 1.

Tabel 1: Voor- en nadelen aan ns-3

Voordeel	Nadeel
Performantie Hybride simulaties Commandline parameters Verschillende modellen Constance ontwikkeling Tracingmogelijkheden Actieve community	WiFi gebreken Geen visualisatie Gebrek aan documentatie Verouderde WiFi standaarden Geen mogelijkheid tot debuggen

4 Besluit

In de vorige sectie werden een aantal problemen en gebreken aangehaald alsook positieve eigenschappen van ns-3. Een ultiem besluit over ns-3 en deze bachelorproef wordt dan ook gemaakt op basis van de resultaten van de simulaties, eigen ervaringen en de beschreven voor- en nadelen.

Simulatiemogelijkheden

Wanneer men kijkt naar de eerste twee simulaties alsook de twee eerste scenario's van de derde simulatie, kan men over het algemeen besluiten dat ns-3 redelijk accuraat simuleert. Echter blijven deze simulaties bij eenvoudige scenario's en ethernet of een eenvoudig WiFi netwerk. Voor Ethernet is de simulator dus accuraat te noemen.

Wat WiFi betreft werden er in dit werk toch een aantal problemen en gebreken opgemerkt zoals het ontbreken van modellen voor de recentste standaarden, het ontbreken van een error-model en de onverzoenbaarheid met lxc-containers. Indien men dus complexere zaken wil simuleren met WiFi apparaten, is ns-3 misschien niet aangewezen. Echter werd er een diepgaander onderzoek gedaan door Nicola Baldo [9] en is het aangewezen deze besluiten eveneens in rekening te nemen.

Andere modellen zoals deze voor Wimax en MESH werden niet besproken in deze bachelorproef, echter is de aanwezigheid van deze meer exotische netwerkfaciliteiten een meerwaarde voor ns-3.

Bijkomende mogelijkheden

Ns-3 biedt een groot aantal mogelijkheden bovenop het simuleren van netwerken op zich. De verschillende mogelijkheden van tracing en de mogelijkheden tot interactie met reële applicaties of apparaten vormen een echte troef voor de simulator. Een van deze laatste mogelijkheden werd getest in dit eindwerk, namelijk het gebruik van lxc-containers, en kon als accuraat en positief beschreven worden ondanks de incompatibiliteit met WiFi apparaten.

De mogelijkheid van het traceren van specifieke eigenschappen alsook het uitschrijven van de netwerkactiviteit naar pcap-bestanden werden eveneens besproken in dit werk en als positief bevonden. Hoewel moet opgemerkt worden dat er geen eenvoudige mogelijkheid is tot het debuggen en dit dan weer negatief is voor complexere opstellingen.

Ns-3 biedt buiten deze twee mogelijkheden nog een groot aantal aan specifieke zaken die echter niet werden gebruikt of besproken in dit werk. In

dit besluit kan dus enkel opgemerkt worden dat de aanwezigheid ervan een positief feit is. Aan de keerzijde moet opgemerkt worden dat de afwezigheid van visualisatie en een grafische editor een nadeel betekenen voor personen die netwerken willen simuleren zonder uitgebreide kennis van C++ of Python. Het gebrek aan visualisatie betekent vooral een gebrek aan mogelijke inzichten die hier uit volgen en beperkt dus de mogelijkheid tot analyse.

Eigen ervaringen

Uit eigen ervaringen kan besloten worden dat het gebruik van ns-3 in eerste instantie vreemd is, echter is ns-3 handig in gebruik wanneer men de tutorial heeft gevolgd en de structuur van de simulaties kent. Er dient daarbij wel opgemerkt te worden dat voor meer gedetailleerde zaken geregeld documentatie ontbreekt en dit het gebruik minder makkelijk maakt.

Algemeen besluit

Over het algemeen kan ns-3 als een degelijk simulatieframework beoordeeld worden aan de hand van dit eindwerk alsook andere bronnen zoals de papers van Nicola Baldo [9] en Elias Weingartner [12]. Echter hangt de voldoening van het gebruik van ns-3 vooral af van de situatie waarvoor deze wordt aangewend. Voor elke specifieke situatie kan een verder onderzoek naar ns-3 en andere simulatoren dus nodig zijn om te besluiten of ns-3 de gewenste simulator is.

Referenties

- [1] Athstats man page. <http://linux.die.net/man/8/athstats>, Aug. 2010.
- [2] Big buck bunny » download. <http://www.bigbuckbunny.org/index.php/download/>, Aug. 2010.
- [3] error model for wireless devices - ns-3-users. http://groups.google.com/group/ns-3-users/browse_thread/thread/2def7c53a3dbae16/f4375e70f5877af9?f4375e70f5877af9, Aug. 2010.
- [4] Linux container simulation, csma ok, wifi ko - ns-3-users. http://groups.google.com/group/ns-3-users/browse_thread/thread/b49a9e9eb91c73eb, Aug. 2010.
- [5] Omnet++ community site. <http://www.omnetpp.org/>, June 2010.
- [6] Simpy simulation package homepage. <http://simpy.sourceforge.net/>, June 2010.
- [7] waf - project hosting on google code. <http://code.google.com/p/waf/>, June 2010.
- [8] Wi-fi alliance: Glossary. http://www.wi-fi.org/knowledge_center_overview.php?docid=3762, Aug. 2010.
- [9] BALDO, N., REQUENA-ESTESO, M., NÚÑEZ-MARTÍNEZ, J., PORTOLÉS-COMERAS, M., NIN-GUERRERO, J., DINI, P., AND MANGUES-BAFALLUY, J. Validation of the ieee 802.11 mac model in the ns3 simulator using the extreme testbed. In *SIMUTools '10: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques* (ICST, Brussels, Belgium, Belgium, 2010), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 1–9.
- [10] BARR, R., AND HAAS, Z. J. Jist/swans. <http://jist.ece.cornell.edu/>, June 2010.
- [11] CARNEIRO, G. J. A. M. Ns-3: Network simulator 3. <http://www.nsnam.org/tutorials/NS-3-LABMEETING-1.pdf>, 2010.
- [12] ELIAS WEINGARTNER, H. V. L., AND WEHRLE, K. A performance comparison of recent network simulators. In *Proceedings of the IEEE International Conference on Communications* (2009).

- [13] KUROSE, J. F., AND ROSS, K. W. *Computernetworks, a top down approach*. Pearson Education, 2008.
- [14] NSNAM. How to use linux containers to set up virtual networks. http://www.nsnam.org/wiki/index.php/HOWTO_Use_Linux_Containers_to_set_up_virtual_networks, Aug. 2010.
- [15] NSNAM. The network simulator - ns-2. <http://www.isi.edu/nsnam/ns/>, June 2010.
- [16] NSNAM. ns-3 documentation. <http://www.nsnam.org/doxygen-release>, Aug. 2010.
- [17] NSNAM. Ns-3: Tap bridge model. http://www.nsnam.org/doxygen/group__tap_bridge_model.html, Aug. 2010.
- [18] NSNAM. ns-3 tutorial. <http://www.nsnam.org/tutorials.html>, July 2010.
- [19] NSNAM. ns-3: Wifi models. http://www.nsnam.org/doxygen-release/group__wifi.html, Aug. 2010.
- [20] NSNAM. ns-3 wiki. <http://www.nsnam.org/wiki/index.php>, Aug. 2010.
- [21] NSNAM. ns-3 wiki: Network simulation cradle integration. http://www.nsnam.org/wiki/index.php/Network_Simulation_Cradle_Integration, June 2010.
- [22] TROLLTECH. Qt's modules. <http://doc.trolltech.com/4.5/modules.html>, June 2010.
- [23] WAND. Wand network research group: Network simulation cradle. <http://research.wand.net.nz/software/nsc.php>, June 2010.
- [24] WEISS, P. Announcement ns-3-generator. <http://mailman.isi.edu/pipermail/ns-developers/2010-May/007998.html>, Aug. 2010.

Lijst van figuren

1	De verschillende modules van ns-3, gelaagd weergegeven . . .	8
2	Gesimuleerde applicaties op reële hosts in een testbed	16
3	reële applicaties in reële hosts die via een gesimuleerd net- werk en TapBridges in ns-3 communiceren met elkaar	17
4	Zes nodes verspreid over twee systemen in een cluster met behulp van MPI voor gedistribueerde simulatie.	18
5	netwerk topologie gebruikt door DSG, afbeelding toont een netwerk met size=16.	19
6	Evolutie van het congestionwindow van node 1 gedurende de simulatie	30
7	IOStats van de simulatie. Zwart = node 1, rood = node 2, groen = node 3, blauw = node 4.	31
8	Duplicate ACKs en fast retransmit aangeduid in Wireshark . .	31
9	Begin van de simulatie van netwerk twee in Wireshark, WiFi beacons en associations.	36
10	Grafiek met de twee UDP-stromen gegenereerd op node 1 (zwart) en node 3 (rood), de verticale as duidt het aantal bits per 0.1 seconden aan.	37
11	Het aantal pakketten verzonden door node 1 (zwart) en node 3 (rood) per 0,001 seconde.	37
12	Topologie van netwerk-3.cc	39
13	Schematische weergave van de werking van het Tap Bridge model in UseBridge configuratie	42
14	Eerste scenario schematisch weergegeven	42
15	Grafiek van ontvangen bits per seconde op node 1 bij het eerste scenario	43
16	Grafiek van ontvangen bits per seconde op node 1 bij het eerste scenario volgens de simulatie	43
17	Tweede scenario schematisch weergegeven	45
18	Grafiek met UDP-trafiek (zwart) en TCP-trafiek (rood) in het tweede scenario. Het groene oppervlak duidt de totale hoe- veelheid trafiek op het netwerk aan.	46
19	Grafiek met UDP-trafiek (zwart) en TCP-trafiek (rood) in het tweede scenario tijdens simulatie. Het groene oppervlak duidt de totale hoeveelheid trafiek op het netwerk aan.	46
20	Derde scenario schematisch weergegeven	47
21	Grafiek met TCP-trafiek (zwart) en RTP-trafiek van node 3 naar 2 (rood) alsook RTP-Trafiek van server naar node 1 (groen) in het derde scenario tijdens test op het netwerk.	48

22	Grafiek met TCP-trafiek (zwart) en UDP-trafiek van node 3 naar 2 (rood) alsook RTP-Trafiek van server naar node 1 (groen) in het derde scenario tijdens simulatie.	48
23	Vierde scenario schematisch weergegeven	49
24	Vijfde scenario schematisch weergegeven	49
25	Cumulatief aangepast of toegevoegd aantal lijnen code in ns-3 t.o.v. de tijd	55

Listings

1	SendPacket en ScheduleTx methodes voor de applicatie in simulatie 1	27
2	Het maken van een switch in ns-3	28
3	callback methode voor het traceren van een congestionwindow .	29
4	trace-source koppelen aan de callback methode	29
5	Configuratie en installatie van WiFi modellen en Devices . . .	32
6	Configuratie en installatie van het mobiliteitsmodel	34
7	Tracing naar athstats	35
8	Instellen van realtime scheduler met checksum	40
9	Vereenvoudigd bash script voor het opzetten van een linux container met bridge en tap voor ns-3	40
10	Toevoegen van de connectie tussen een ns-3 node en een lxc container	41