Faculteit
Ingenieurswetenschappen

Departement
Elektrotechniek – ESAT

KATHOLIEKE
UNIVERSITEIT
LEUVEN

Adaptieve video streaming over heterogene netwerken

Eindwerk voorgedragen tot het behalen van het diploma van burgerlijk elektrotechnisch ingenieur, richting elektrotechniek, optie Telecommunicatie en Telematica

Lars Peters
Pieter Wuytens

Promotor:
Prof. dr. ir. Antoine Van de Capelle
Dagelijkse begeleiding:
ir. Dagang Li

2007 – 2008

Voorwoord

Dit woord van dank is gericht aan iedereen die op één of andere manier heeft bijgedragen tot dit afstudeerwerk. Sommige mensen verdienen echter meer dan een gewoon dankwoordje.

Op de eerste plaats wensen wij onze promotor Prof. dr. ir. Antoine Van de Capelle te bedanken. Ondanks zijn drukke bezigheden heeft hij dit werk grondig doorgenomen, zijn opmerkingen en kritiek hebben bijgedragen tot de goede afloop van dit eindwerk.

Ook onze assistent ir. Dagan Li verdient een woordje van dank. Gedurende het hele jaar heeft hij ons eindwerk opgevolgd en heeft hij ons op weg geholpen waar dit nodig was. Met eventuele vragen konden wij steeds bij hem terecht.

Verder dank ik al de mensen die met Click bezig zijn. Zonder de toewijding van deze mensen zou Click niet zo een goede software router geworden zijn. Vooral willen wij Eddie Kohler bedanken, hij is de ontwikkelaar van Click.

Onze vriendinnen Truus en Elke wensen we te bedanken voor het begrip, de steun en het geduld dat zij hebben opgebracht. Verder danken wij ook onze naaste familieleden voor de steun die wij van hen kregen tijdens de realisatie van dit werk.

Bedankt allemaal,

Lars Peters en Pieter Wuytens

Abstract

Vandaag bestaat data trafiek op het Internet voor een aanzienlijk deel uit multimedia-trafiek. Dit soort trafiek stelt soms zware eisen in verband met Quality of Service (QoS) - benodigde bandbreedte, maximale vertraging of maximale jitter - aan het Internet. Bovendien is dit Internet opgebouwd uit verschillende types netwerken, elk met hun eigen karakteristieken inzake de diensten die ze kunnen aanbieden. Daarenboven wordt de aangeboden QoS beïnvloed door alle andere trafiek die zich op deze netwerken bevindt. De juiste afweging maken tussen QoS-beperkingen en de capaciteit van de onderliggende netwerken is een zware opgave, vooral wanneer de multimedia-stromen over verschillende onderliggende netwerktechnologieën - elk met verschillende eigenschappen - gestreamed worden.

Een systeem werd opgezet om de router en de server op de hoogte te houden van veranderingen in het type van netwerk. Hierbij werd gekozen voor de implementatie van een zachte handover. Router en server passen zichzelf aan aan het onderliggende netwerk en proberen zo een aanvaardbare QoS te leveren aan de videostromen.

Om de overgang naadloos te laten verlopen is een mechanisme voorzien om de bestemmeling van één enkel IP te voorzien in plaats van één IP voor elke interface. Pakketten die reeds onderweg zijn naar de bestemmeling moeten zo niet meer gewijzigd worden en bereiken hun bestemming zonder problemen.

De volgende aanpassing om de handover beter te laten verlopen was een omleidingsbuffer. Bij een handover blijven pakketten achter in de uitgaande wachtrij van de daarvoor actieve interface. De omleidingsbuffer leidt pakketten om naar de juiste wachtrij. Dit zorgt ervoor dat bij een handover minder pakketten verloren gaan.

Een eerste techniek aangewend om QoS te implementeren is het voorrang geven van een videostroom ten opzichte van de rest van het verkeer. De tweede gebruikte techniek is een vorm van intelligentie geven aan de videobron, zodat deze weet over welke soort verbinding het videoverkeer loopt. Hier werd besloten om gebruik te maken van een temporeel schaalbare codering. De film wordt hierbij opgedeeld in meerdere lagen. Hier werd de film opgedeeld in twee lagen, de zgn. basislaag (I- en P-frames) en een verbeteringslaag (B-frames). Het idee hierbij is dat de basislaag van een vaste QoS wordt voorzien, terwijl de verbeteringslaag op een “Best-Effort-manier wordt doorgestuurd. Zo werd het mogelijk verschillende niveau’s van prioritering toe te kennen aan verschillende delen van de film. Deze techniek zorgt ervoor dat basislaag van de film weinig tot geen invloed ondervinden aan wisselende verkeersomstandigheden op het netwerk die momenten van congestie en ontoelaatbare vertraging veroorzaken.

De tweede techniek bestaat uit een intelligente server. Bij een overgang naar een ander type van

verbinding bestaat de mogelijkheid dat de capaciteit van de link de video niet kan verwerken. Om dit probleem aan te pakken werd een systeem ontwikkeld dat de server van de nodige informatie voorziet om de bitsnelheid van de video aan te passen naargelang de maximale snelheid. Hier werd gewerkt met twee types van verbindingen en voorziet de server dus ook twee verschillende bitsnelheden. Een mechanisme werd ontwikkeld dat de server via een pakket op de hoogte brengt van de verandering van verbinding. De server neemt dan de nodige maatregelen om een aangepaste bitsnelheid door te sturen. Er werd geopteerd om de server te voorzien van meerdere versies van eenzelfde videostroom aan verschillende bitsnelheden.

Om resultaten te genereren werd een stroom gegenereerd in plaats van een echte video door te sturen. Deze maakt gebruik van zgn. video traces. In deze stroom werd informatie over de pakketten zelf en hun bedoelde inhoud gestoken zodat ze goed konden worden opgevolgd. Het voordeel van een gegenereerde stroom is dat de werkelijke informatie van de verschillende frames niet is gegeven. Wel het aantal bits na codering die worden gebruikt door de verschillende frames. Er is hier dus geen sprake van auteursrechten. De grootte van een trace is ook veel kleiner, typisch enkele megabytes.

Inhoudsopgave

Lijst van Afkortingen

ARP	Address Resolution Protocol
CIF	Common Intermediate Format
CPU	central processing unit: De processor van een computer in het nederlands ook wel CVE centrale verwerkingseenheid genoemd
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DCT	Discrete Cosine Transformation
DHCP	Dynamic Host Configuration Protocol
FDDI	Fiber Distributed Data Interface
FIFO	First In First Out: de pakketten die het eerste in de wachtrij komen worden er ook het eerste uitgehaald.
GOP	Group Of Pictures
ICMP	Internet Control Message Protocol
IP	Internet Protocol
LCD	Liquid Crystal Display
MAC	Media Access Control
MII	media-independent interface
MPEG-4-FGS	Moving Pictures Expert Group 4 Fine Grained Scalability
OSI	Open Systems Interconnection
PAL	Phase Alternating Line
PCM	Pulse Code Modulation
PING	is een hulpprogramma voor computernetwerken, dat wordt gebruikt om de reactietijd in milliseconden tussen twee computers in een netwerk te meten. Hoe lager deze waarde is, hoe sneller je een reactie van de andere computer terugkrijgt. Ping maakt gebruik van het ICMP-protocol
PSNR	Peak Signal-to-Noise Ratio
QCIF	Quarter CIF

QoS	Quality of Service
RED	Random Early Detection
RGB	Rood, geel en blauw
RTCP	Real-time Control Protocol
RTP	Real-time Transport Protocol
SNR	Signal-to-Noise Ratio
SVC	Scalable Video Coding
TCP	Transmission Control Protocol
TTL	Time To Live
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
UTP	Unshielded Twisted Pair
VoIP	Voice over IP
WLAN	Wireless Local Area Network
YUV	Kleurenruimte gebruikt in een PAL televisietoestel. Y staat voor de helderheidscomponent (luminantie), U en V voor de kleurcomponenten (chrominantie).
FTP	File Transfert Protocol

Lijst van figuren

Lijst van tabellen

Hoofdstuk 1

Inleiding

Vandaag bestaat data trafiek op het Internet voor een aanzienlijk deel uit multimedia-trafiek. Dit soort trafiek stelt soms zware eisen in verband met Quality of Service (QoS) - benodigde bandbreedte, maximale vertraging of maximale jitter - aan het Internet. Bovendien is dit Internet opgebouwd uit verschillende types netwerken, elk met hun eigen karakteristieken inzake de diensten die ze kunnen aanbieden. Daarenboven wordt de aangeboden QoS beïnvloed door alle andere trafiek die zich op deze netwerken bevindt. De juiste afweging maken tussen QoS-beperkingen en de capaciteit van de onderliggende netwerken is een zware opgave, vooral wanneer de multimedia-stromen over verschillende onderliggende netwerktechnologieën - elk met verschillende eigenschappen - gestreamed worden.

In deze thesis zal een heterogene netwerkomgeving worden opgezet, bestaande uit veel gebruikte bedrade en draadloze netwerktechnologieën zoals IEEE802.3 Ethernet en IEEE802.11 Wireless LAN. Video zal gestreamed worden over deze testomgeving en de impact van verschillende netwerktopologieën en verschillende patronen van achtergrondtrafiek zal onderzocht worden. Een mechanisme wordt ontworpen tussen de video server/client en het onderliggende netwerk, zodat de server op de hoogte wordt gehouden van veranderingen - zoals in capaciteit of vertraging - in het onderliggende netwerk. Daarvoor zal een cross layer information system (XLIS) geïmplementeerd worden.

Dankzij XLIS zullen de nodes in staat zijn zichzelf aan te passen aan het onderliggende netwerk en zal een aanvaardbare QoS geleverd kunnen worden aan de videostromen. Een intelligent algoritme dat de service parameters in de applicatielaag aanpast aan de hand van informatie, verkregen van onderliggende lagen, wordt ontwikkeld. Dit algoritme verzekert een optimaal gebruikscomfort wanneer geschakeld wordt tussen verschillende netwerkconfiguraties.

Hoofdstuk 2

Achtergrondinformatie

De recente ontwikkelingen in computerhardware en telecommunicatie maken het mogelijk dat de eindgebruiker over multimedia kan beschikken waar en wanneer hij wil. Er wordt verwacht dat videotrafiek in de toekomst een groot, zoniet het grootste, deel van de internettrafiek voor haar rekening zal nemen. Dit zal zowel gebeuren over bedrade als draadloze netwerken. Als consequentie is het imperatief dat de bestaande architectuur en internetprotocols hier extensief op worden getest.

Dit hoofdstuk handelt over het hoe en waarom van streamen. Standpunten worden ingenomen over de gebruikte protocols en methodes. Tevens wordt er hier voldoende achtergrondinformatie voorzien om het vervolg van de thesis te kunnen begrijpen en om eventueel de resultaten te reproduceren.

2.1 Streamen

2.1.1 Definitie

Streamen wordt gedefinieerd als volgt: "Het constant ontvangen en tegelijkertijd afspelen van multimedia door de eindgebruiker terwijl het wordt doorgestuurd door de bron.". Het begrip refereert eerder naar hoe het medium gestreamed wordt, dan naar het medium zelf. Het onderscheid wordt meestal gemaakt voor media, die worden gedistribueerd over een telecommunicatienetwerk, omdat de meeste andere distributiesystemen inherent streamen (bvb. radio en televisie) of niet (bvb. Dvd's, Cd's en boeken).

De vooruitgang in computernetwerken, gecombineerd met krachtige Pc's en moderne besturingssystemen, maken het mogelijk dat streamen praktisch haalbaar en betaalbaar is. Een voorbeeld hiervan is de "niet-computer" oplossingen om te luisteren naar audiostromen.

In het algemeen is de benodigde opslagruimte voor multimedia zeer groot. Dit betekent dat de opslag- en transmissiekosten hoog liggen. Daarom worden de media meestal gecomprimeerd, zowel voor opslag als voor streamen.

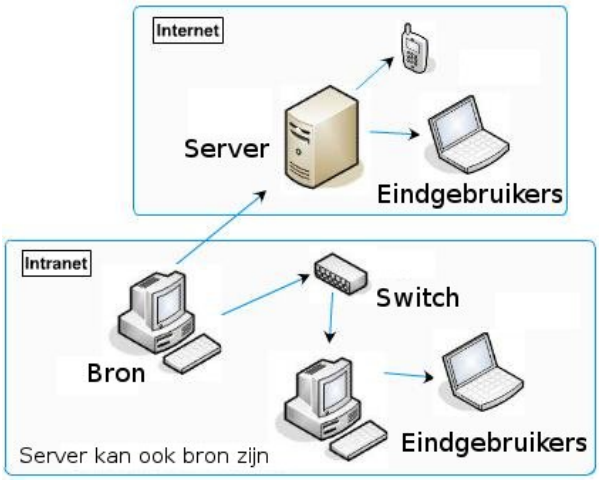
Er bestaan twee grote takken in de manier waarop naar een stroom wordt geluisterd, nl. live streaming en streaming op aanvraag. Live stromen zijn enkel beschikbaar op een bepaald

tijdstip. Bijvoorbeeld het streamen van een voetbalmatch. Stromen op aanvraag zijn, meestal voor langere tijd, opgeslagen op een opslagmedium. De gebruiker kan dan een aanvraag sturen om deze stroom door te sturen.

Onderzoek in deze tak van de telecommunicatie is intens.

2.1.2 Realisatie

Een typische streamingconfiguratie bestaat uit een server, een transportnetwerk en één of meerdere gebruikers. Op figuur 2.1 zijn de verscheidene delen te herkennen.



Figuur 2.1: Typische streamingconfiguratie

De bron bevat de verschillende stromen. De streaming server stuurt deze door naar de internetgebruikers. In dit geval bestaan de stromen uit videobestanden. De gebruikers vragen rechtstreeks via een intranet, of onrechtstreeks via internet, de verscheidene stromen aan of luisteren live mee. Meestal beschikt de bron over een groot aantal harde schijven. Zo kunnen meerdere stromen tegelijkertijd worden opgeslagen en aangevraagd. Ook beschikt de bron over voldoende rekenkracht om de verscheidene stromen tegelijkertijd door te sturen en eventueel bewerkingen hierop uit te voeren.

De streaming server bepaalt het gebruikte protocol en kan eventueel over andere functies beschikken die hem in staat stellen de stroom aan te passen aan wisselende netwerkomstandigheden. Hij regelt, bij aanvraag door een externe eindgebruiker, het transport over het internet en laat het transport naadloos verlopen. Belangrijke zaken zoals het prioritizeren van uitgaande pakketten en het toelaten van eindgebruikers, zoals gebruikelijk bij de voorziening van QoS, zijn hierbij belangrijk. Meer informatie over QoS staat in 2.6.

De derde belangrijke groep zijn de eindgebruikers. Deze zijn de eindbestemmelingen van de media. Zij beluisteren en/of bekijken de stromen. Het is de bedoeling dat deze snel en foutloos door hen ontvangen worden.

Streaming methodes worden in deze context opgedeeld in twee grote takken, nl. unicast en

multicast. Bij unicast krijgt elke eindgebruiker een aparte stroom van de server doorgestuurd. Multicast daarentegen stuurt één enkele stroom over het netwerk totdat het pad naar de verschillende eindgebruikers zich splitst. Neem dat twee mensen die naast elkaar wonen een tenniswedstrijd aan het volgen zijn. Als deze wordt doorgestuurd met unicast, worden twee stromen van deze wedstrijd vanaf de server tot bij elk televisietoestel doorgestuurd. Als gebruik wordt gemaakt van multicast zal er maar één stroom lopen van de server tot aan de router waar het pad tussen beide opsplitst. Multicast heeft dus als grote voordeel dat er minder verkeer over het netwerk loopt. Het nadeel is dat er naar exact dezelfde media moet gekeken worden. Dit komt vooral bij realtime kijken voor.

2.1.3 Problemen

Er zijn enkele veel voorkomende problemen die optreden bij het versturen van informatie over internet. Hieronder de meest voorkomende met de bijhorende oorzaak:

- *Vollopen van de buffer van de router*: netwerkcongestie
- *Te veel vertraging*: te lange wachtrij
- *Pakketten uit volgorde*: fouten in pakketten of vernietigde pakketten
- *Variatie op vertraging*: variatie in de lengte van de wachtrij
- *Beschadigde pakketten*: rekenfout of geheugenfout in de router of server
- *Verandering in beschikbare middelen (bandbreedte/buffer)*: wegvallen van een verbinding of veranderen van netwerkbelasting

Veel problemen zijn aan de variatie in de bezetting en de grootte van de wachtrij gerelateerd. De variatie is op haar beurt te wijten is aan de veranderende internettrafiek. Deze problemen worden aangepakt met de voorziening van QoS.

2.2 Streaming in heterogene netwerken

Hetero stamt af van het Griekse woord heteros, wat “verschillend” betekent. De term heterogene netwerken kan dan ook op verschillende manieren opgevat: hetzelfde type netwerken verbonden met verschillende soorten besturingssystemen of protocols, maar ook de aanwezigheid van verschillende types netwerken. In deze thesis slaat de term op het laatste geval.

In heterogene netwerken wordt het streamen complexer. Op een bepaald moment wordt er overgestapt op een ander netwerk. Dit kan verschillen in o.a. protocol, snelheid en vertraging. De ratio tussen een bekabeld netwerk (±100Mbit) en een draadloos netwerk (±11Mbit) is bijvoorbeeld 10. Bij overgang tussen deze netwerken kan het de streaming-applicatie doen stotteren ten gevolge van congestie. Deze overgang heet een verticale handover. Verschillende types van handovers worden besproken in 2.2.1. Om de media naadloos te streamen is het dus van vitaal belang dat er wordt opgetreden bij zo een overgang.

2.2.1 Handovers

Het onderwerp handovers werd reeds kort aangeraakt. In deze sectie wordt de terminologie rond handovers uitgelegd.

2.2.1.1 Verticale en horizontale handover

Een horizontale handover is een handover tussen twee dezelfde technologieën en protocols. Bijvoorbeeld een Gsm-mast die het overneemt van een andere is een typische horizontale handover. Een verticale handover is een handover in heterogene netwerken, tussen twee verschillende netwerktechnologieën. Bijvoorbeeld een GSM die gebruik maakt van UMTS en WLAN. Bij het wegvallen van de ene dienst, wordt er een overgeschakeld naar de andere.

Snelle ontwikkelingen op vlak van video streaming maken verticale handovers meer en meer gebruikelijk. Iemand die bijvoorbeeld kijkt naar een serie op televisie wil deze serie elders kunnen volgen op GSM. Met een druk op de knop schakelt de video over van een LCD televisie naar de GSM. Hierbij is een overgang van de televisiekabel naar de draadloze technologie van de GSM, bijv. UMTS, nodig.

2.2.1.2 Zachte en harde handover

Een zachte handover ontstaat wanneer er reeds een verbinding aanwezig is met het netwerk waarop wordt overgeschakeld. Als bijvoorbeeld een GSM van de ene cel naar de andere beweegt, kunnen beide cellen al verbinding maken met de GSM om zo het overnemen te versnellen.

Bij een harde handover is er nog geen connectie voorzien met het tweede netwerk. De eventuele authenticatie en verdere initialisatie van de verbinding kunnen een naadloze overgang onmogelijk maken. Verder in de thesis wordt dan ook gebruik gemaakt van een zachte handover. Meer informatie in 3.3.

2.2.2 Aanpassing bij handover

In 2.2 werd opgemerkt dat er bij een verticale handover best wordt opgetreden. Zoniet kan dit, gezien de lange duur van een handover, leiden tot permanente congestie van de verbinding. Verschillende methodes om deze congestie tegen te gaan worden hier besproken. De meeste zijn gebaseerd op het veranderen van bitsnelheden.

2.2.2.1 Meerdere versies

Het idee is dat dezelfde video beschikbaar is in verschillende bitsnelheden. Bij een handover wordt de bron hiervan op de hoogte gebracht. Deze zal dan de stroom aanpassen naargelang de capaciteit van de link verhoogt of verlaagt. De implementatie hiervan is te vinden in 5.1.

Om dit te bewerkstelligen moet de bron dus voorzien zijn van dezelfde stroom aan verschillende bitsnelheden. Deze verschillende bitsnelheden kunnen worden bekomen op diverse manieren. De voornaamste zijn:

- *Verscheidene pixelgroottes*: veel gebruikte groottes zijn: ITU-R/CCIR-601 (standaard TV formaat) 620×480 pixels, CIF 352×288 pixels en QCIF 176×144 pixels. Naargelang er minder pixels worden gebruikt, zakt de bitsnelheid en de kwaliteit. Indien er een groter formaat nodig is bij afspelen, kan men een kleiner formaat interpoleren.

- *Hogere of lagere framesnelheid*: standaard is dit 30 frames per seconde. Hoe lager de framesnelheid, hoe lager de bitsnelheid en de kwaliteit. Bij het afspelen kunnen er enkele frames worden herhaald.
- *Verschillende grootte van quantisatiestap*: deze is afhankelijk van video tot video. Hoe groter de quantisatiestap, hoe lager de bitsnelheid en de kwaliteit.

Deze methode heeft voor- en nadelen. Omdat de video's in verscheidene formaten aanwezig moeten zijn op de bron vereist dit veel schijfruimte. Dit komt de schaalbaarheid niet ten goede. Men kan bijvoorbeeld voor elke video vijf verschillende bitsnelheden ter beschikking hebben. Dit vermenigvuldigt de benodigde schijfruimte met ongeveer een factor drie. Wel is hiervoor een minder krachtige bron, in termen van rekenkracht, nodig als bij bijvoorbeeld transcoding. Zie 2.2.2.2 voor meer informatie.

Een nadeel is dat men moet weten hoe de gebruiker is verbonden met het netwerk. Met WLAN technologie is een veel hogere bitsnelheid haalbaar dan met UMTS technologie.

Een ander nadeel is dat de videokwaliteit plots kan overspringen naar een lagere bitsnelheid, wat zichtbaar kan zijn voor de gebruiker. Of dit storend is of niet, is natuurlijk een subjectief gegeven. De kwaliteit is echter beter dan bij transcoding, mits er wordt gecomprimeerd vanaf een originele video in plaats van een reeds gecomprimeerde.

Bij deze methode treedt weinig vertraging op. De video is direct beschikbaar en kan dus ook direct worden afgespeeld. Samen met een eenvoudige implementatie is deze methode zeer populair.

2.2.2.2 Transcodering

De definitie van transcodering is terug te vinden in [**?**]. Conversie van één codec naar een andere is transcodering. Meer informatie over codecs staat in 2.4. Deze codecs gaan meestal gepaard met een verlies aan informatie. Een tweede belangrijke eigenschap is dat dit realtime gebeurt. Eerst vindt een decompressie van de gecomprimeerde video naar een tussenformaat (PCM voor audio en YUV voor video) plaats. Daarna wordt dit tussenformaat gecomprimeerd naar het gewenste formaat. De gemakkelijkste manier om transcoding te doen, is een bitstroom decoderen naar YUV en dan de data coderen met een andere standaard. Een betere manier is de bitstroom te veranderen van de ene standaard naar de andere zonder volledige decompressie. Hiervoor bestaan verscheidene algoritmes.

Transcoding kan worden samengevat onder twee technieken. Transrating is een proces waarbij de video wordt omgezet naar een lagere bitsnelheid zonder het formaat aan te passen. Transsizing staat bekend als het veranderen van de pixelgrootte.

Een grote beperking van transcoding is de hercompressie van een reeds gecomprimeerd formaat. Omdat codeertechnieken met een hoge compressiefactor verliezen aan informatie met zich meebrengen, introduceert een decodering, gevolgd door een codering, een progressief verlies aan kwaliteit. Dit introduceert extra artefacten bij elke transcodering. Het is dus aan te raden direct te coderen vanaf een verliesvrij formaat of alleszins het keren dat een video getranscodeerd wordt te beperken.

Het grote voordeel van transcoderen is dat de video niet beschikbaar moet zijn in de verscheidene formaten. Dit bespaart opslagruimte. Bij elke overgang naar een verbinding met lagere capaciteit

vindt een transcodering plaats. Dit is nefast voor de kwaliteit van de video, maar zorgt ervoor dat kennis over het netwerk niet meer vereist is.

Het overspringen naar een lagere kwaliteit bij een handover zal nog steeds zichtbaar zijn. Transcoderen introduceert ook een zekere vertraging. Er moet een gedeelte van de video zijn gedecomprimeerd en opnieuw gecomprimeerd vooraleer de bron de video begint te versturen. Ook is er voor transcoderen - zeker voor ingewikkelde codecs zoals H264 - veel rekenkracht nodig.

2.2.2.3 Gelaagde compressie

Een video bitstroom wordt schaalbaar genoemd als delen van de stroom kunnen worden verwijderd, zodat de resulterende bitstroom nog bruikbaar is voor een bepaalde decoder. Deze substroom is een gedeelte van de originele stroom en de gereconstrueerde kwaliteit is lager dan die van de volledige bitstroom. De eerder opgesomde technieken in 2.2.2.1 en 2.2.2.2 hebben deze eigenschap niet en worden niet schaalbare coderingstechnieken of bitstromen met één laag genoemd.

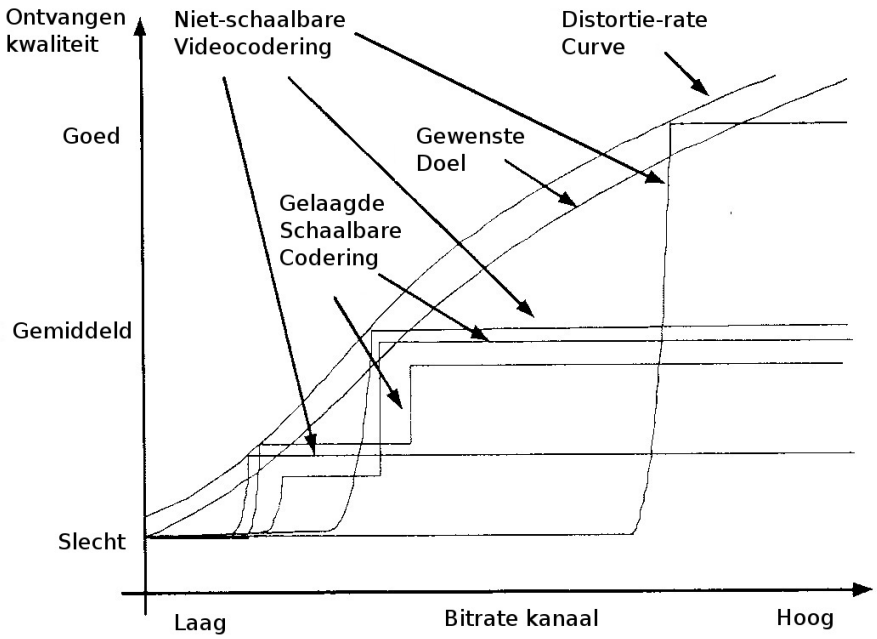
Gelaagde compressie kan worden opgedeeld in twee grote stukken. Codering met een fijne korrel, met vele lagen, en grove korrel, met een beperkt aantal lagen. Codering met fijne korrel kan in de literatuur teruggevonden worden als MPEG-4 FGS en H264 SVC. Grove korrel draagt meestal de naam signaal-ruis-verhouding (SNR), ruimtelijke (spatial) en tijdsgerelateerde (temporal) codering.

Ruimtelijke en tijdsgerelateerde schaalbaarheid zijn technieken gebaseerd op een bitstroom met respectievelijk gereduceerde grootte en lagere framesnelheid. SNR schaalbaarheid geeft een substroom met dezelfde tijds- en ruimteresolutie, maar lagere signaal-ruis-verhouding.

Gelaagde codering met fijne korrel is een techniek waarbij de kwaliteit van de video verbetert naargelang er meer informatie wordt ontvangen. Informatie over deze techniek werd gevonden in [**?**] en [**?**]. De lagen kunnen worden opgedeeld in een basislaag en een verbeteringslaag. De basislaag moet volledig zijn ontvangen voor het afspelen van de video. De kwaliteit van de video verbetert naargelang er meer van de basislaag wordt ontvangen. Dit speelt in op de extra vereiste die internet stelt aan een videocodec, nl. optimaliseren van videokwaliteit over een gegeven bereik van bitsnelheden in plaats van een gegeven bitsnelheid. Conceptmatig is dit terug te vinden op figuur 2.2.

De horizontale as geeft de bitsnelheid van het kanaal aan en de verticale de ontvangen videokwaliteit. De distortie-rate curve geeft een theoretische bovengrens aan voor de behaalbare kwaliteit aan een gegeven bitsnelheid. De drie trapvormige curves geven de performantie aan van een optimale - niet schaalbare - coderingstechniek. Eens er een bitsnelheid is gekozen, probeert de niet schaalbare techniek de optimale kwaliteit te bekomen. Dit is te zien aan de tredes van de trapcurves genaamd niet-schaalbare videocodering die zeer dicht bij de distortie-rate curve liggen. Als de bitsnelheid van het kanaal gelijk aan de video bitsnelheid is, is dit optimaal. Als de bitsnelheid lager ligt, krijgen we het digitaal “alles of niets” verschijnsel, dat zich uit in de zeer lage kwaliteit. Bij hogere bitsnelheid van het kanaal, zien we geen verbetering in de kwaliteit.

Ook technieken met grove schaalbaarheid zijn op figuur 2.2 ook te herkennen. Ze zijn verge-



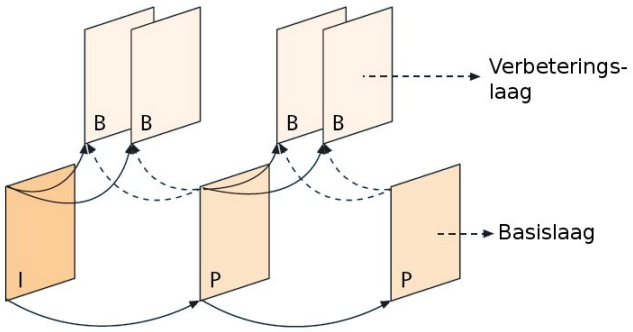
Figuur 2.2: Illustratie van videocompressie performantie

lijkbaar met de niet schaalbare technieken omdat voor een verbetering van de videokwaliteit een laag volledig moet zijn ontvangen. Bij ontvangst van maar een gedeelte van de laag, is er geen verbetering. Dit is te zien aan de trapvormige curves gelaagde schaalbare codering met meerdere trappen. Wat eigenlijk gewenst is, is een codeertechniek die mee evolueert met de bitsnelheid van het kanaal. Dit wordt verwezenlijkt door de technieken met fijne korrel.

Zoals zichtbaar op de curve gewenste doel, heeft FGS een extra redundantie ten opzichte van niet schaalbare technieken. Met andere woorden, om een video te coderen aan dezelfde kwaliteit, is er een hogere bitsnelheid nodig. Dit is het grootste nadeel van de codeertechnieken met fijne korrel. Het grootste voordeel is de graduele aanpassing van de kwaliteit aan de capaciteit van het kanaal. De basislaag moet volledig zijn ontvangen, maar de verbeteringslaag niet. Hoe meer informatie van de verbeteringslaag binnenkomt, hoe beter de kwaliteit. Een tweede pluspunt is dat de videostroom maar één enkele keer moet worden gecomprimeerd.

Uit dit alles kan men concluderen dat de eigenschappen van codeertechnieken met fijne korrel van deze codec een perfecte kandidaat maken voor video streaming. Een implementatie in de thesis is niet gebeurd om de volgende reden. Na enig zoekwerk werden er twee encoders gevonden, nl. MoMuSys[**?**] en Microsoft MPEG-4 Video Reference Software[**?**]. Het probleem met deze encoders is dat het referentie-encoders zijn. Ze vragen dus nog steeds zeer veel parameters. Aangezien het eindwerk niet over codering gaat, is er besloten de fijne korrel codeertechniek niet te gebruiken.

In plaats van technieken met fijne korrel werd er geopteerd voor technieken met grove korrel. De beschikbaarheid van traces maakte het mogelijk de temporeel schaalbare techniek te gebruiken. Deze techniek wordt dan ook verder toegelicht in 2.2.3. Om het verloop te kunnen begrijpen is het aan te raden sectie 2.4 eerst te lezen.



Figuur 2.3: Gelaagde temporele compressie

Bij temporeel schaalbare codering is er plaats gelaten tussen de frames in de basislaag. Hier bevinden zich de frames van de verbeteringslaag. Elk frame van de verbeteringslaag is gecomprimeerd met behulp van inter-frame codering met referentie naar de frames in de basislaag. In het geval van figuur 2.3 gebeurt dit met I-, P- en B-frames. De B-frames behoren tot de verbeteringslaag en worden gecodeerd met behulp van de I- en P-frames uit de basislaag.

De basislaag biedt dus een standaard videokwaliteit aan een lage framesnelheid. Als de basislaag en de verbeteringslaag worden samengevoegd, verhoogt de framesnelheid en dus ook de kwaliteit. Het gevolg hiervan is dat de basislaag volledig moet zijn ontvangen vooraleer gebruik kan worden gemaakt van de verbeteringslaag. Wel kan de basislaag onafhankelijk van de verbeteringslaag worden gedecodeerd. Van deze eigenschap kan handig gebruik worden gemaakt bij prioriterisatie van pakketten, zoals terug te vinden in 2.2.4.

2.2.3 Gegenereerde stroom

Informatie over het gebruik van gegenereerde stromen is terug te vinden in [**?**]. Om onderzoek te doen naar de effecten van een handover en achtergrondverkeer bij het streamen van een video is er een gecodeerde video nodig. Er zijn drie verschillende manieren om dit te doen: een video bitstroom, een video trace en een videotrafiek model.

Een video bitstroom is een werkelijke video die gecodeerd wordt in het gewenste formaat. Deze bitstroom bevat de complete video informatie. Het handige hiervan is dat de veranderende videokwaliteit kan bepaald worden door subjectieve evaluatie - kijken naar de video - of door objectieve methodes - evaluatiealgoritmes -. De grote beperking is dat deze bitstromen zeer groot zijn, nl. enkele gigabytes. Een bijkomend nadeel is dat ze meestal beschermd zijn door auteursrechten. Daardoor is het moeilijk om ze in verschillende formaten te verkrijgen.

Een video trace biedt hiervoor een alternatief. De werkelijke informatie van de verschillende frames is hier niet gegeven. Wel het aantal bits na codering die worden gebruikt door de verschillende frames. Er is hier dus geen sprake van auteursrechten. De grootte van zo een trace is ook veel kleiner, typisch enkele megabytes.

Een videotrafiek model is een statistisch model. Het kan worden bepaald uit een video trace. Belangrijke statistische parameters kunnen worden bepaald en men kan zo een bron maken

die een video genereert met dezelfde statistische eigenschappen. Dit model is meestal bepaald door een aantal video traces. Op de uitvoer van de bron wordt dan een statistische analyse uitgevoerd en vergeleken met die van de traces. Als ze voldoende accuraat wordt beschouwd, kan het model worden gebruikt.

Gezien de eenvoud in gebruik van een video trace, is er besloten deze te gebruiken. Uit de informatie die een video trace bevat, kunnen pakketten worden gegenereerd met informatie die later kan worden gebruikt voor de analyse van het netwerk. Hierover meer in de volgende subsectie.

2.2.3.1 Gebruik van een video trace

De structuur van een video trace wordt weergegeven in tabel 2.1. Dit is een tabel van de basislaag.

Tabel 2.1: Typische structuur van een trace (in dit geval de basislaag)

Framenr.	**Afspeeltijd (s)**	**Frametype**	**Framegrootte (bits)**	**Y-PSNR (dB)**	**U-PSNR (dB)**	**V-PSNR (dB)**
0	0,000	I	62816	39,868	41,239	41,044
3	0,100	P	16480	37,600	40,308	40,027
1	0,033	B	0	33,694	39,074	38,873
2	0,066	B	0	33,245	39,456	39,561
6	0,200	P	11440	37,609	40,334	40,214
...	...	...	...	...	...	...

De eerste kolom is het framenummer. De frames worden doorgestuurd in codecvolgorde. De afspeeltijd wordt gegeven in de tweede kolom. Dit is de tijd waarop een frame in de film te zien is. Hieruit is af te leiden dat de film gecodeerd is aan 30 frames per seconde. Het type van een frame staat in de derde kolom. De vierde kolom geeft de framegrootte weer. Als er een nul staat, betekent dit dat het een frame uit de verbeteringslaag is.

De vijfde, zesde en zevende kolom geven de signaal-ruisverhoudingen van de verschillende componenten van de film. Y staat voor de helderheidscomponent (luminantie), U en V geven de kleurcomponenten (chrominantie). Deze getallen zijn een vaak gebruikte objectieve maat voor de kwaliteit van een film. Er moet worden opgemerkt dat tabel 2.1 de PSNR verhoudingen geeft als er niets anders dan de basislaag wordt ontvangen.

Tabel 2.2 geeft de trace van de verbeteringslaag weer. Hierbij dienen de waardes weergegeven in de PSNR-kolommen opgeteld te worden bij de waardes uit de basislaag om de PSNR van de basislaag gecombineerd met de verbeteringslaag te bekomen.

De PSNR is een objectieve maat voor de kwaliteit van een video. PSNR wordt meestal gedefinieerd aan de hand van de gemiddeld kwadratische fout. Voor twee monochrome beelden I en K van grootte $m \times n$, waarvan één een benadering is die ruis bevat, geldt de volgende formule voor de gemiddeld kwadratische fout:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \| I(i,j) - K(i,j) \|^2$$

De PSNR is gegeven door:

$$PSNR = 10.log_{10} \left(\frac{MAX_I^2}{MSE} \right)$$

Tabel 2.2: Typische structuur van een trace (in dit geval de verbeteringslaag)

Framenr.	**Afspeeltijd (s)**	**Frametype**	**Framegrootte (bits)**	**Y-PSNR (dB)**	**U-PSNR (dB)**	**V-PSNR (dB)**
0	0,000	I	0	0,000	0,000	0,000
3	0,100	P	0	0,000	0,000	0,000
1	0,033	B	8864	4,168	1,283	1,160
2	0,066	B	9784	4,442	0,925	0,807
6	0,200	P	0	0,000	0,000	0,000
...	...	...	...	...	...	...

Hierin is MAX_i de grootst mogelijke pixelwaarde van de figuur. Zo is bijvoorbeeld voor een beeld waar pixels zijn voorgesteld door 4 bits, deze waarde gelijk aan 15. Voor kleurenbeelden zijn er drie RGB waardes per pixel. Hier is de MSE over alle RGB waardes en wordt de deler $3 \times m \times n$.

Met behulp van deze informatie worden pakketten gecreëerd. Om gemakkelijk een bepaald type frame te linken aan een pakket, gebruiken we één frame per pakket. Deze pakketten bevatten het framenummer, frametype, de afspeeltijd en de grootte. Natuurlijk wordt er rekening gehouden met de maximale hoeveelheid informatie die een pakket kan bevatten. Afhankelijk van het gebruikte protocol verandert deze. Dus als de grootte van een frame boven de maximale informatieinhoud van een pakket gaat, wordt een frame opgesplitst in meerdere pakketten. De berekening van de maximale grootte is terug te vinden in 2.3.7.

Om dit te verwezelijken is er een element geschreven in Click. Meer informatie over Click in 3.1. Dit leest de nodige informatie uit het trace-bestand en genereert pakketten aan de gewenste grootte. Het is de naam FakeStream meegegeven. De C++ code is terug te vinden in A.18.

2.2.4 Omgaan met achtergrondtrafiek

Het grote verschil tussen handovers en achtergrondverkeer is dat achtergrondverkeer meestal gepiekt verloopt. In vergelijking met een handover is dit een kort verschijnsel. Omgaan met achtergrondtrafiek kan dus best gebeuren door kleine, snelle aanpassingen. Een goede manier hiervoor is de voorziening van QoS. Zie 2.6 voor meer informatie over QoS.

Alhoewel fijne korrel technieken, zoals besproken in 2.2.2.3, perfect kunnen gebruikt worden bij handovers, zijn deze eerder aangewezen bij het omgaan met achtergrondtrafiek. De grote

schaling[1] in bandbreedte, die gebeurt bij een handover, heeft ook een grote verandering in verstuurd verkeer nodig. Hiervoor zijn de technieken in 2.2.2.1 en 2.2.2.2 aangewezen.

Methodes zoals transcoding toepassen in een fel veranderende omgeving vereist dat de parameters in een hoge frequentie veranderen. Dit legt veel druk op de bron. Deze moet elke keer zijn codering aanpassen. Gelaagde compressie beschikt inherent over een groot bereik. Dit wordt dus ook in het verdere verloop van de thesis gebruikt.

Zoals beschreven in 2.2.2.3 bestaat deze stroom uit een basislaag en een verbeteringslaag. Het idee is om de basislaag een hogere prioriteit te geven dan de verbeteringslaag. Zo kan de router zelf beslissen of hij al dan niet de beschikbare bandbreedte heeft om de verbeteringslaag door te sturen. Op die manier wordt een gedeelte van de bandbreedte gereserveerd voor de basislaag, terwijl de verbeteringslaag op een "Best-Effort"[2] manier wordt doorgestuurd.

2.3 Protocols

Protocols leggen een set standaardregels vast. Deze regels kunnen gaan van datavoorstelling, signalisering, authenticatie tot foutdetectie. Communicatieprotocols hebben vele eigenschappen die ervoor zorgen dat betrouwbare uitwisseling van informatie over een onbetrouwbaar kanaal mogelijk is. Samengevat is een protocol een algoritme dat bepaalde regels volgt zodat het uitwisselingssysteem naar behoren werkt.

In deze sectie wordt een kort overzicht te geven van de protocols die het streamen over internet vergemakkelijken. De verschillende encapsulaties zullen hiërarchisch worden opgesomd. De voor- en nadelen voor streaming worden aangehaald.

2.3.1 OSI lagen

Protocols moeten eerst gesitueerd worden in het lagenmodel. Een laag is een collectie van gerelateerde functies die diensten leveren aan de laag erboven en ontvangen van de laag eronder. Tabel 2.3 geeft een korte beschrijving van de lagen weer. Deze tabel is louter gegeven ter situering van de verschillende protocols.

Het nut van deze lagen kan begrepen worden uit de encapsulatie van gegevens. Om de functies van elke laag te configureren en te voorzien van voldoende informatie, worden er voor elke laag hoofdingen voorzien. Het principe hiervan is te zien in figuur 2.4.

2.3.2 Ethernet

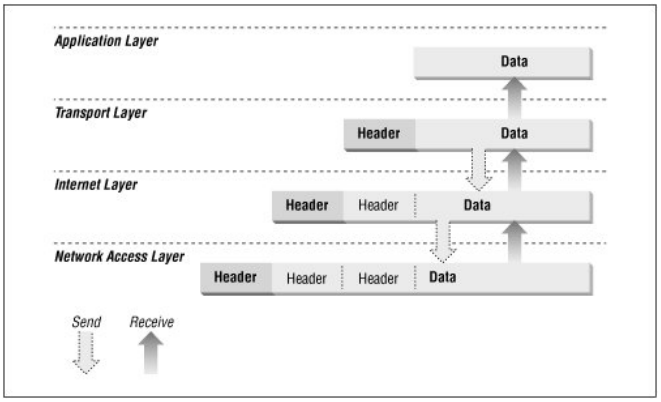
In de fysische laag bestaat Ethernet uit punt-tot-punt verbindingen, verbonden door Ethernet hubs en switchen. Meestal gebeurt dit in een stervormige structuur met behulp van UTP

[1] Ordegrootte is tientallen

[2] Er wordt dan geen onderscheid gemaakt tussen de verschillende soorten trafiek. Alle verkeer krijgt eenzelfde voorrang.

Tabel 2.3: ISO lagenmodel

Data eenheid	Laag	Functie
Data	Applicatielaag	Koppeling proces en applicatie
Data	Presentatielaag	Datarepresentatie en encryptie
Data	Sessielaag	Communicatie tussen computers
Segment/Datagram	Transportlaag (TCP/UDP)	Eindpunt-tot-eindpunt connectie en betrouwbaarheid
Pakket	Netwerklaag (IP)	Bepaling pad en logische adressering
Frame	Datalinklaag (MAC en LLC)	Fysische adressering
Bit	Fysische laag	Signaal en binaire transmissie



Figuur 2.4: Encapsulatie van gegevens

kabels.

Bovenop de fysische laag, in de datalink laag, communiceren Ethernet-stations door pakketten naar elkaar te zenden. Dit zijn blokken data die individueel worden verzonden en ontvangen. Elk Ethernet-station bezit een 48-bit MAC adres om de bestemming en bron van een pakket aan te duiden. Netwerkkaarten ontvangen normaal gezien geen pakketten die een andere bestemming hebben dan zichzelf. Om meerdere apparaten op het medium te kunnen laten communiceren, wordt gebruik gemaakt van een mechanisme dat de kanaaltoegang controleert. Voor Ethernet is dit CSMA/CD[3].

In figuur 2.5 staat de structuur van een Ethernetframe[4]. Deze bevindt zich in de datalinklaag. De volgende elementen zijn te onderscheiden:

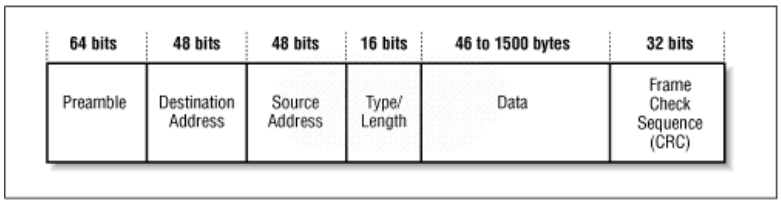
- *Synchronisatie*: zorgt voor functies in de fysische laag.
- *Bestemmingsadres*: het MAC adres van de bestemmeling op het subnetwerk.
- *Bronadres*: het MAC adres van de bron op het subnetwerk.
- *Type, lengte*: bepaalt aan welke hogere laag een frame wordt doorgegeven.
- *Data*: gegevens in een frame; maximaal 1500 bytes, minimaal 46[5].

[3]Dit is een protocol dat ervoor zorgt het kanaal eerlijk wordt verdeeld. In het geval dat twee apparaten tegelijkertijd het kanaal willen gebruiken, wordt aan de hand van dit algoritme bepaald wie eerst mag zenden en wie moet wachten.

[4]Een Ethernetframe is niet te verwarren met een videoframe. Een videoframe is één beeld genomen een sequentie van beelden. Een Ethernetframe is een frame uit een sequentie van frames gezonden over een netwerk.

[5]De minimum framegrootte is nodig om een botsing te ontdekken, cfr. CSMA/CD.

- *32 bit CRC*: ontdekt fouten in een frame.



Figuur 2.5: Structuur Ethernet frame

2.3.3 IP

Deze bevindt zich in de netwerklaag, bovenop Ethernet. IP accepteert data van een hogere laag en encapsuleert deze in één of meerdere pakketten. Vooraf een verbinding opzetten is niet nodig. Een zender zendt de pakketen naar een ontvanger waar hij vooraf niet mee heeft gecommuniceerd. Dit is een connectieloos protocol.

Omdat er wordt geëncapsuleerd in IP kan er een abstractie gemaakt worden van de onderliggende laag. Deze kan een mengeling zijn van verschillende protocols zoals Ethernet, FDDI, token ring, enz.. Elke datalinklaag heeft zijn eigen methode van adressering.

IP voorziet een onbetrouwbare verbinding. Er is dus geen garantie gemaakt dat:

- Er geen fouten in het pakket zijn.
- Het pakket in de juiste volgorde wordt geleverd.
- Er twee of meerdere dezelfde pakketten aankomen.
- Pakketten helemaal niet aankomen.

Deze functionaliteit wordt voorzien door een bovenliggende laag, zoals TCP. Als data, doorgegeven door een hogerliggende laag, groter zijn dan de maximale grootte van de data van een Ethernetframe, dus 1500 bytes, fragmenteert IP de pakketten in kleinere stukken. Bij aankomst van deze gefragmenteerde pakketten zet IP deze wel in de juiste volgorde.

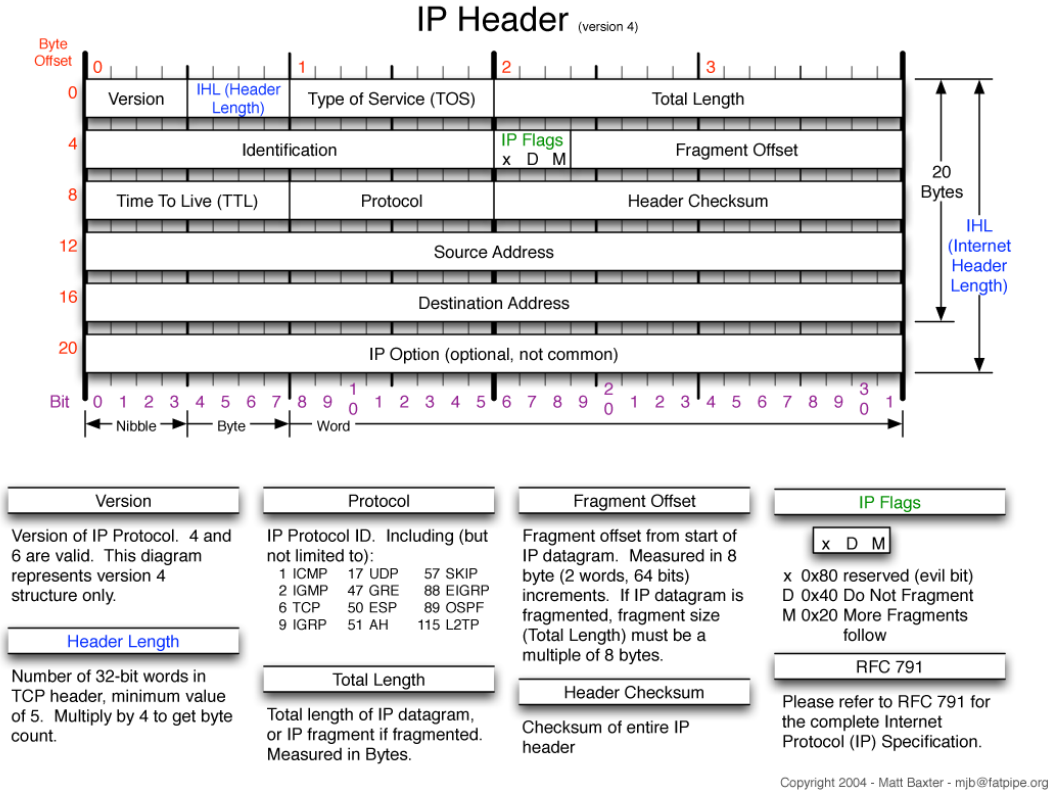
IP heeft een aparte adressering en kan zo routering voorzien. Adressering zorgt voor het geven van IP-adressen en routering zorgt ervoor dat pakketten, aan de hand van de adressen, het juiste pad naar de bestemmeling volgen.

De structuur van een IP-pakket wordt weergegeven in figuur 2.6. Deze hoofding wordt geëncapsuleerd in een Ethernetframe.

Het eerste veld in de hoofding van een IP-pakket is het versie-veld. Voor IPv4 heeft dit de waarde 4.

Het tweede veld bevat het aantal 32-bit woorden in de hoofding. Dit kan variëren omdat een IP hoofding een aantal niet-verplichte opties bevat. De minimale lengte is 5, dus $5 \times 32 = 160$ bits lang, en maximaal 15.

In het Type of Service veld wordt de prioriteit van het IP-pakket aangegeven. Deze prioriteit kan worden gebruikt om dit pakket voorrang te geven in de wachtrijen van de routers die



Figuur 2.6: Structuur IP-pakket

het tegenkomt op zijn pad naar de bestemming. Het veld is dus bruikbaar voor realtime-toepassingen zoals videostreaming en VoIP. In Differentiated Services noemen ze dit DiffServ Code Points. Meer informatie over Differentiated Services is terug te vinden in 2.6.3.

Het veld "totale lengte geeft de grootte van het volledige pakket in bytes weer, inclusief hoofding en data. De minimale lengte is 20 bytes (enkel een hoofding) en maximaal 65535 bytes.

Om verschillende fragmenten van een origineel datagram te detecteren, staat in het Identification veld een uniek nummer.

Het volgende veld is het vlaggenveld en bestaat uit drie bits:

- *Reserved*: moet nul zijn
- *Don't Fragment*: de IP-pakketten mogen niet worden gefragmenteerd. Als fragmenteren nodig is, en deze vlag is gezet, wordt het pakket vernietigd.
- *More fragments*: geeft aan of een pakket in een reeks van gefragmenteerde pakketten thuishoort. Voor het laatste pakket in de reeks en voor pakketten die niet zijn gefragmenteerd, is deze vlag niet gezet.

Het Fragmentatie Offset veld geeft de plaats weer van een pakket in een reeks gefragmenteerde pakketten uit hetzelfde datagram.

Time To Live geeft aan hoeveel routers het pakket mag passeren vooraleer het wordt vernietigd. Zo kan men voorkomen dat een pakket blijft circuleren op het netwerk. Elke router vermindert

deze waarde met één. Als de waarde nul wordt, vernietigt hij het pakket. Er wordt dan een boodschap teruggestuurd naar de zender waarin staat dat zijn pakket werd vernietigd.

Het Protocol veld geeft aan welk protocol er in het dataveld van het IP-pakket wordt gebruikt. Enkele hiervan zijn terugvinden in figuur 2.6.

In de Header Checksum wordt de hoofding op fouten gecontroleerd. Bij elke doorgang door een router wordt deze waarde gecontroleerd. Indien ze fout is, wordt het pakket vernietigd.

Het IP adres van de bron wordt gegeven in het Source Address veld. Het IP adres van de bestemmeling in het Destination Address veld. Een voorbeeld van een IP adres is 10.9.8.7. In het veld wordt deze waarde weergegeven in bits.

Het Options veld bevat bijkomende opties. Dit wordt vaak niet gebruikt en en er wordt niet op ingegaan.

2.3.4 TCP

Voor de meest gebruikte internettoepassingen is de volgende laag TCP. Zoals verder wordt besproken, is dit protocol nochthans geen goede keuze voor realtime toepassingen als videostreaming. Hiervoor is UDP aangewezen. Om deze reden worden slechts de eigenschappen die TCP van UDP onderscheiden, besproken.

Een eerste eigenschap is het rangschikken van doorgestuurde gegevens, heruitzending van verloren pakketten en vernietigen van duplicaten. TCP gebruikt een sequentienummer om elke byte in de gegevens aan te geven. Het sequentienummer geeft de volgorde van de bytes aan om zo de data in betrouwbaar en in volgorde te kunnen doorsturen. UDP heeft geen sequentienummer en voorziet ook geen heruitzending.

TCP gebruikt bevestigingsboodschappen om aan te geven dat alle pakketten die voorgaan aan deze boodschap goed zijn ontvangen. Elke eerste data byte in een segment wordt een sequentienummer toegewezen. De ontvanger stuurt een bevestiging van ontvangst met het sequentienummer van de volgende byte die hij verwacht.

Als de zender zo ontdekt dat er data verloren zijn gegaan in het netwerk, stuurt hij de data opnieuw uit. Opnieuw uitsturen van data is niet nodig bij realtime-toepassingen. Als de data te laat aankomen, zijn ze niet meer bruikbaar. Het opnieuw uitzenden, laat andere data wachten totdat de voorgaande is uitgezonden. Het heruitzenden van een paar pakketten kan er dus voor zorgen dat een kettingreactie ontstaat en alle volgende data te laat aankomen.

Een tweede eigenschap is het foutvrije zenden van gegevens. Om in deze functionaliteit te voorzien, is er een controleveld in de hoofding voorzien. Deze controleert zowel de hoofding als de gegevens. UDP voorziet enkel een controleveld voor de hoofding. Realtime-toepassingen hebben geen boodschap aan het heruitzenden van foute gegevens. In het geval van videostreaming, wordt in geval van een foute frame, de vorige herhaald. Vertraging door heruitzending en mogelijkheid dat de rest van de videoframes ook te laat aankomen is dus te vermijden, zoals hierboven aangegeven.

De derde belangrijke eigenschap draagt de naam Flow Control. Deze functie zorgt ervoor dat de zender zijn informatie niet te snel stuurt om in betrouwbare ontvangst en verwerking te

kunnen voorzien. Om dit te verwezelijken wordt gebruik gemaakt van een schuivend venster. Meer informatie over het schuivend venster in [**?**]. Als de ontvanger de zender niet bijhoudt, wordt de zender verzocht te stoppen met zenden. Dit is een probleem bij realtime informatie. Deze wordt namelijk onbruikbaar als ze aankomst na de deadline. Het schuivend venster zorgt er ook voor dat reeds aangekomen informatie nog niet beschikbaar is wanneer ze nog niet is bevestigd. Deze extra vertraging zou ervoor kunnen zorgen dat de informatie onbruikbaar wordt. Ook zorgen de buffers, nodig voor het schuivend venster, voor extra geheugenverbruik.

De laatste eigenschap is congestiecontrole. Congestie treedt op wanneer het netwerk het verkeer op zijn paden niet meer kan verwerken. Een typisch verschijnsel hierbij is het vollopen van buffers van routers. Deze vernietigen de pakketten die ze niet kunnen opslaan. Congestiecontrole past de bitsnelheid van de zender aan zodat dit niet gebeurt. Zo past TCP zich aan aan de omstandigheden van het netwerk. Dit gebeurt voornamelijk door de bevestigingsboodschappen. Meer informatie over congestiecontrole is terug te vinden in [**?**]. In het kort wordt de werking uitgelegd als volgt. Als er te weinig bevestigingen aankomen bij de zender, zal deze zijn bitsnelheid verlagen. Een verhoging zal plaatsvinden na een aantal correct ontvangen bevestigingen. De bitsnelheid is dus variabel. Dit introduceert variatie in vertraging in plaats van pakketten die verloren gaan, zoals het geval is bij UDP. Omdat pakketten met een te grote vertraging niet nuttig zijn, is dit niet nuttig voor realtime toepassingen. Immers, wanneer een pakket niet meer nodig is, maar toch wordt heruitgezonden, is dit onnodig extra verkeer op het netwerk. UDP houdt geen rekening met de congestie van een netwerk. De versturende applicatie bepaalt aan welke snelheid de UDP datagrammen worden doorgestuurd

2.3.5 UDP

Datagram-protocols zoals UDP zenden mediastromen in een serie van kleine pakketten. Dit is eenvoudig en efficiënt. Wel is er hier geen mechanisme, zoals bij TCP, om de levering van de pakketten te garanderen. De applicatie die de pakketten ontvangt, moet verlies detecteren en beschadigde pakketten herkennen. Datagrammen kunnen dus in een verkeerde volgorde aankomen of verloren gaan zonder dat het protocol hier iets aan probeert te doen.

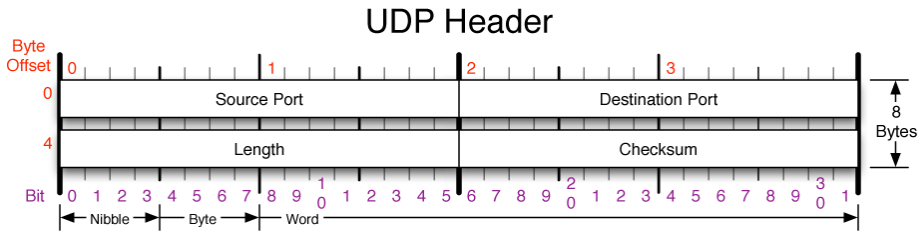
Door de onbetrouwbaarheid van het protocol worden de extra bewerkingen vermeden en dus het protocol sneller en efficiënter. In realtime toepassingen is UDP zeer populair omdat vernietigen van pakketten een betere optie is dan pakketten die te laat aankomen. Late pakketten verbruiken extra netwerkbronnen, vernietigde pakketten niet.

UDP bevindt zich in de transportlaag, maar voorziet enkel multiplexen van applicaties, aan de hand van poorten, en het controleren op fouten in de hoofding en gegevens.

De structuur van de UDP hoofding is dan ook zeer kort en eenvoudig. Bijgevolg creëert UDP weinig overhead. Op figuur 2.7 kan worden gezien dat de belangrijkste velden de poorten zijn. Poorten laten communicatie tussen twee applicaties toe. Poorten kunnen zich bevinden tussen 0 en 65535.

De lengte van het volledige datagram, hoofding en data, is weergegeven in het Length veld. Het Checksum veld wordt gebruikt voor de foutdetectie.

Omdat er geen mechanisme is voorzien voor congestiecontrole, bestaat er gevaar voor oververzadiging. Bronnen met een hoge concentratie aan UDP verkeer kunnen dat teweeg brengen.



Figuur 2.7: Structuur UDP header

Hiervoor moeten voorzieningen worden getroffen door netwerkelementen zoals routers. Zij kunnen bijvoorbeeld UDP pakketten in wachtrijen zetten en de trafiek regelen zodat de rest van de data op het netwerk voldoende bandbreedte ter zijn beschikking krijgen. Dit kan worden verwezenlijkt met de voorziening van QoS.

Vroeger was het totale UDP verkeer op het internet slechts een paar procent. Nu de beschikbare bandbreedte aangeboden door ISP's kijken van video's via internet mogelijk maakt, stijgt dit aanzienlijk. Vele noodzakelijke applicaties maken nochthans gebruik van UDP. Hieronder vallen Domain Name System (DNS), Dynamic Host Configuration (DHCP) en het Routing Information Protocol (RIP). Het is dus een must om deze belangrijke boodschappen een voorrang te geven op de rest van het verkeer.

2.3.6 RTP en RTCP

2.3.6.1 RTP

Informatie over RTP is terug te vinden in [**?**]. RTP is speciaal ontwikkeld voor audio- en video streaming over het internet. Het werkt hand in hand met RTCP en is bovenop UDP gebouwd. Applicaties die gebruik maken van RTP zijn minder gevoelig voor verlies van pakketten, maar meestal wel zeer gevoelig voor te grote vertragingen.

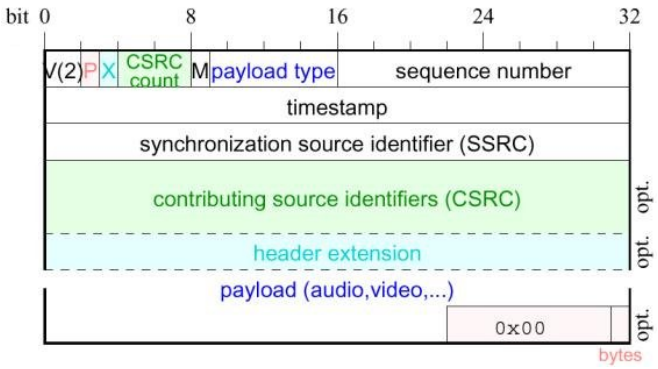
Zelf voorziet RTP geen mechanismen om ervoor te zorgen dat data op tijd aankomen of andere QoS garanties, maar steunt het op de onderliggende lagen om dit te doen. Het voorziet geen garanties voor levering van data of middelen om pakketen die in de verkeerde volgorde aankomen te voorkomen. Het verwacht ook niet dat de onderliggende lagen hiervoor zullen zorgen.

De sequentienummers in RTP zorgen ervoor dat de ontvanger de pakketsequentie van de zender kan reconstrueren. Deze nummers kunnen ook gebruikt worden om de juiste plaats van een pakket te bepalen, bij bijvoorbeeld decompressie van video, zodat het niet nodig is deze pakketten in een bepaalde volgorde te decomprimeren.

Omdat RTP is ontwikkeld om een verscheidenheid aan applicaties te ondersteunen, bevat het een flexibel mechanisme waarmee nieuwe applicaties kunnen worden ontwikkeld zonder het RTP protocol elke keer te moeten herschrijven. Voor elke klasse van toepassingen definieert RTP een profiel en meerdere formaten. Dit profiel voorziet informatie die ervoor zorgt dat de functie van velden in de hoofding voor die klasse kan worden begrepen. Het formaat definieert hoe de data in het RTP pakket moeten worden geïnterpreteerd.

RTP bestaat uit twee delen:

- Het RTP protocol om data met realtime vereisten te vervoeren.
- Het RTCP protocol om de QoS te controleren en informatie door te spelen naar de gebruikers. Zij maken op basis van deze informatie beslissingen om lokale aanpassingen door te voeren.



Figuur 2.8: Structuur RTP hoofding

De structuur van een RTP hoofding is weergegeven in figuur 2.8.

De V staat voor RTP versie, de P voor Padding. Padding zorgt ervoor dat een pakket altijd opgevuld is. Indien er niet genoeg informatie beschikbaar is, wordt er nutteloze informatie toegevoegd. X staat voor Extension bit. Als deze is gezet, volgt er een extensie van de hoofding na de standaard hoofding.

CSRC count bevat het aantal CSRC[6] identificatienummers. M staat voor Marker. De betekenis hiervan wordt bepaald door de implementatie. Meestal wordt deze gebruikt voor het aangeven van belangrijke gebeurtenissen zoals het einde van een videoframe. Payload Type geeft het type informatie in het RTP pakket weer. De applicatie weet dan welke data er in het pakket zit. Zo kan men bijvoorbeeld aangeven met welke codec een bepaalde film is gecomprimeerd.

Het Sequence number veld geeft een sequentienummer. Dit nummer verhoogt met één elke keer er een RTP pakket wordt verzonden. Zo kan de ontvanger het verlies van een pakket detecteren of een pakketsequentie herstellen. Timestamp geeft het moment waarop het pakket werd verzonden weer. Het geeft niet de tijd van de dag weer, maar het aantal tikken[7] van de klok tussen dit frame en het vorige. Het eerste pakket krijgt de timestamp 0. Dit veld kan gebruikt worden voor synchronisatie en berekeningen van variatie in pakketvertraging.

SSRC staat voor Synchronization Source. Dit is een nummer dat één enkele bron binnen een RTP stroom identificeert. Zo krijgen bijvoorbeeld twee microfoons op dezelfde computer een verschillende identificatie toegewezen. Het is willekeurig zodat geen twee bronnen binnen dezelfde RTP sessie hetzelfde nummer hebben. CSRC staat voor Contributing Source identifiers list. Deze wordt enkel gebruikt als een aantal RTP stromen door een mixer zijn gegaan. Een mixer kan stromen van verschillende bronnen decoderen, aan elkaar plakken, weer coderen, samennemen en verder sturen naar de bestemmelingen. De nodige bandbreedte kan zo worden

[6] De betekenis van CSRC volgt.
[7] Tikken zijn eenheden van tijd. Voor de ene toepassing kan die 125ms zijn, voor de andere 1s.

beperkt. De CSRC bevat de SSRC waardes van de verschillende bronnen. Het aantal bronnen in een CSRC kan oplopen van 0 tot 15.

2.3.6.2 RTCP

Meer informatie over RTCP is terug te vinden in [**?**]. Het RTCP protocol is gebaseerd op de periodische verzending van controlepakketten naar alle deelnemers van de sessie. Het gebruikt hetzelfde distributiemechanisme als de datapakketten, nl. RTP. Het onderliggende protocol moet de multiplexing van datapakketten en controlepakketten voorzien.

Het protocol voorziet drie functies:

- Feedback op de kwaliteit van de datadistributie.
- Een manier om verschillende datastromen die van dezelfde zender komen aan elkaar te linken en te synchroniseren.
- Identificatie van de zender om deze te laten zien aan de gebruiker.

Bronnen kunnen met deze informatie hun doorvoersnelheid aanpassen aan wisselende netwerkomstandigheden. Bijvoorbeeld een lagere kwaliteit aan film doorsturen, of juist een hogere ngl. de verandering.

Omdat er botsingen[8] kunnen voorkomen of omdat een programma soms moet worden heropgestart, moeten de ontvangers een andere manier hebben om de zenders te kunnen identificeren. Dit gebeurt aan de hand van een zgn. canonische naam CNAME. De SSRC nummers, uitgezonden door de bron, worden gelinkt aan deze CNAME.

Er bestaan een aantal verschillende soorten pakketten in RTCP:

- *Rapporten van de zender*: hierin staan verzend- en ontvangststatistieken van de zender
- *Rapporten van de ontvanger*: idem, maar bestemd voor ontvangers die geen zenders zijn
- *Bronbeschrijvingen*: hierin staan de CNAMEs en andere informatie over de zender
- *Applicatie-specifieke controlepakketten*

Deze pakketten worden gezonden over UDP. Omdat ze klein zijn, passen er meerdere in één UDP datagram. Daarom is het verplicht minstens twee per datagram te versturen, nl. een rapport en een bronbeschrijving. Omdat deze pakketten klein zijn en elke deelnemer aan de sessie ze uitstuurt, kunnen dit er veel worden. Daarom is er vastgelegd dat ze maximaal 5% innemen van het totale RTP verkeer.

2.3.7 Berekening maximale framegrootte en efficiëntie

In tabel 2.4 staat de berekening van de maximale data in een pakket en de efficiëntie. De eerste kolom geeft het protocol, de tweede en derde het aantal bits die resp. maximaal en minimaal nodig zijn voor de hoofding. Een Ethernetframe heeft minimaal 368 bits data nodig, te zien in kolommen 4 en 5. Hiervan wordt elke keer de plaats nodig voor de minimale, resp. maximale hoofding afgetrokken. In kolommen 6 en 7 gebeurt hetzelfde, maar dan voor de maximale data in een Ethernetframe.

[8] Twee dezelfde SSRC nummers

Nuttige informatie wordt door het maximaal aantal bits gegeven dat het volledig geëncapsuleerde pakket kan dragen. De efficiëntie wordt berekend door de de nuttige informatie te delen door de maximale data in het pakket.

De tabel geeft dus weer dat in geval van een hoofding met minimale grootte, een pakket minstens 48 bytes nuttige informatie moet bevatten. Voor een maximale hoofding is aan deze voorwaarde altijd voldaan. Het maximum aan nuttige informatie dat een pakket kan bezitten, is voor minimale hoofding 11680 bits en 11136 bits voor de maximale. Dit geeft een efficiëntie van resp. 96% en 91%.

Tabel 2.4: Berekening maximale framegrootte en efficiëntie

Protocol	**Min. hoofding (bits)**	**Max. hoofding (bits)**	**Min. data (bits)**		**Max. data (bits)**	
Ethernet	208	208	368	368	12000	12000
IP	160	192	208	176	11840	11808
UDP	64	64	144	112	11776	11744
RTP	96	608	48	-496	11680	11136
Totaal hoofding	528	1072				
Nuttige informatie	11680	11136				
Efficiëntie (%)	96	91				

2.4 Codecs

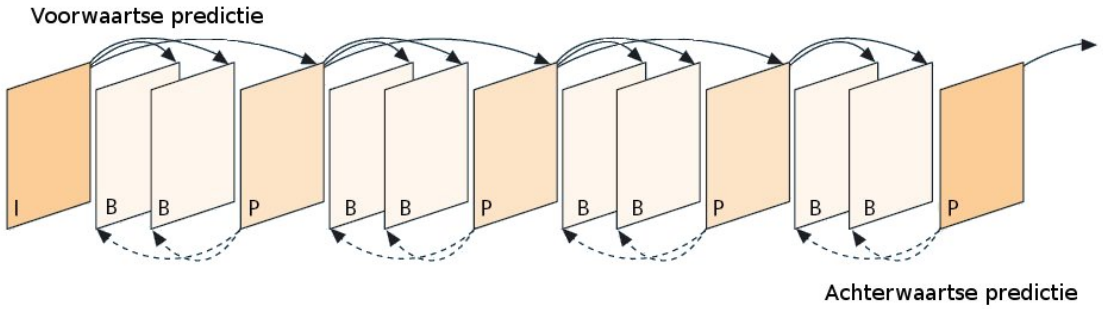
Informatie over MPEG4 kan worden gevonden in [**?**]. Over MPEG4-FGS kan meer informatie teruggevonden worden in [**?**]. De samenvoeging codec staat voor coderen, decoderen. Met de naam codec wordt meestal verwezen naar de gebruikte compressiemethode. Enkele voorbeelden van video-codecs zijn: DivX, Xvid en MPEG. Veel gebruikte audio-codecs zijn: MP3, WMA en AAC. In deze thesis wordt gebruik gemaakt van MPEG voor video en MP3 voor audio. Mits het hier vooral om de video gaat, zal enkel MPEG kort toegelicht worden.

De twee basisprincipes waarop MPEG steunt zijn samen te vatten als intra-frame codering en inter-frame codering. Intra-frame codering is de codering van elke frame op zich en maakt gebruik van DCT. Inter-frame codering is codering tussen verschillende frames en maakt gebruik van bewegingscompensatie.

In intra-frame codering wordt elke videoframe opgedeeld in blokken van 8×8. Deze worden bemonsterd in Y, U en V samples. Elk blok wordt getransformeerd met de DCT transformatie in een blok van 8×8 DCT coëfficiënten. Deze stellen de ruimtelijke frequentiecomponenten van het originele blok voor. Deze DCT coëfficiënten worden dan gequantiseerd. De grootte van de quantisatiestappen worden bepaald door de gewenste bitrate. Een grotere quantisatiestap geeft en lagere kwaliteit en vice versa. De gequantiseerde coëfficiënten worden hierna met behulp van zigzag scanning in een lange rij geplaatst. Hierop wordt run-level codering samen met variabele lengte codering toegepast om verdere compressie te bekomen.

Inter-frame codering wordt bekomen door frames op te delen in verschillende types: I-, P- en B-frames. Dit is weergegeven in figuur 2.9. Een verzameling van frames vertrekkende van een I-

frame tot de frame voorafgaand aan de volgende I-frame noemen we een GOP. De verschillende types van frames worden als volgt gecodeerd. Een I-frame wordt volledig gecomprimeerd gebruik makend van intra-frame codering. Een P-frame wordt gecodeerd met behulp van bewegingscompensatie, gebruik makend van het voorgaande I- of P-frame. We noemen dit achterwaartse predictie. Een B-frame wordt gecodeerd met voorwaartse en achterwaartse predictie en maakt dus ook gebruik van het volgende I- of P-frame.



Figuur 2.9: Hiërarchie van MPEG frames

Om bewegingscompensatie toe te passen wordt een frame opgedeeld in verscheidene blokken. Voor elk blok zal een algoritme zoeken in het voorgaande en volgende frame, afhankelijk van het type, of er een blok hiermee overeenkomt. Wanneer het een blok vindt, wordt hier een bewegingsvector aan toegewezen. Zo moet de informatie van dit blok geen twee keer gecodeerd worden. Als er bijvoorbeeld een rollende bal gecodeerd wordt, zal enkel de bal verplaatsen. De codec herkent de bal in een volgende frame en een bewegingsvector wordt toegekend. Als geen goede overeenkomst wordt gevonden, wordt dat blok gecomprimeerd met behulp van intra-frame codering.

Om een P-frame te kunnen decoderen is er dus het voorafgaande I- of P-frame nodig en voor een B-frame het voorafgaande en volgende I-of P-frame. Het is dus duidelijk dat I-frames belangrijker zijn dan P-frames, die op hun beurt weer belangrijker zijn dan B-frames. Verliezen van een I-frame heeft namelijk tot gevolg dat de volledige GOP niet kan worden gedecodeerd.

De inter-codering van frames heeft ook belangrijke implicaties op het doorsturen van de frames. Stel dat de afspeelvolgorde van de frames IBBPBBPBBPBBIBBP... is. Het eerste frame kan zonder meer worden weergegeven, maar om het tweede frame, een B-frame, af te spelen is het volgende P-frame nodig. Frames in afspeelvolgorde doorsturen is dus een slecht idee. In plaats daarvan worden ze in de zgn. codecvolgorde verzonden, nl. IPBBPBBPBBIBBP.... Dit betekent ook dat - in geval er geen decodervertraging is- er ten vroegste met afspelen wordt begonnen nadat het eerste P-frame ontvangen is.

2.5 Achtergrondverkeer

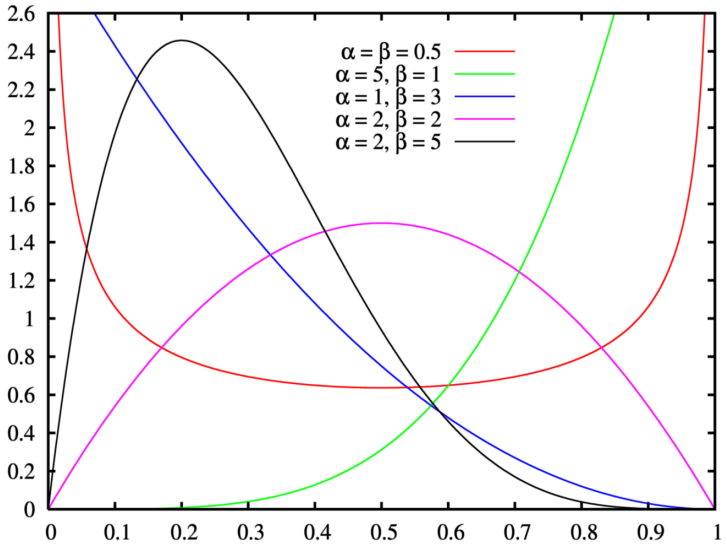
Om een realistisch scenario te creëren is er, buiten de videostroom, achtergrondverkeer nodig. Zonder dit verkeer, treedt er geen congestie op in het netwerk. Dit verkeer bestaat uit

verschillende types van protocols, mits elk protocol een ander gedrag vertoont. Er wordt dus getracht om een variëteit aan protocols verkeer over het netwerk te laten versturen.

Onder achtergrondverkeer wordt hier verstaan dat het niet vanuit de bron vertrekt en ook niet op de bestemmeling van de videostroom aankomt. De lay-out van de gebruikte opstelling is geïllustreerd in figuur 2.11. Het achtergrondverkeer loopt hier van “bron achtergrondtrafiek, langs de hubs en router naar “client achtergrondtrafiek. Hierbij is gekozen voor hubs om de netwerkinterfaces van de bronnen en Clients op de netwerkinterface van de router aan te sluiten.

Er werd geopteerd voor twee belangrijke klassen van verkeer. Eerst en vooral het TCP protocol. Dit wordt ongetwijfeld het meest gebruikt bij versturen van informatie over het internet. Daarnaast zal er nog UDP trafiek worden gegenereerd.

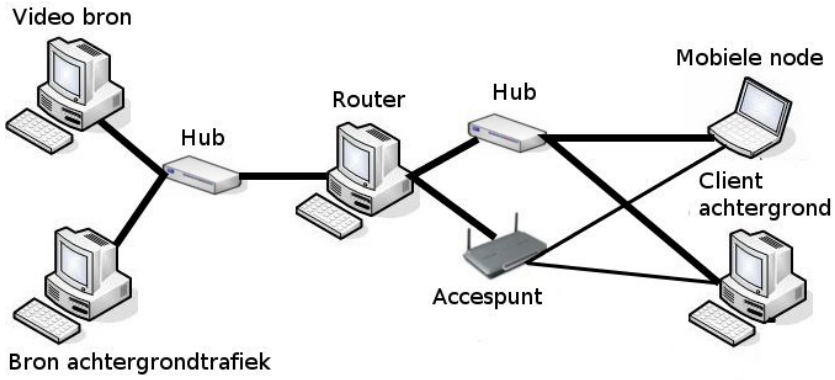
Netwerktrafiek bestaat uit veel kleine en veel grote pakketten. De kleine pakketten zijn meestal controlepakketten en de grote pakketten zijn meestal gefragmenteerde pakketten. Het verkeer heeft dus een U-vormige distributie. Daarom is er besloten om voor de simulatie van achtergrondverkeer vier bronnen te gebruiken. Één voor grote TCP pakketten, één voor kleine TCP pakketten, één voor grote UDP pakketten en één voor kleine UDP pakketten. Zo een distributie is ook verkrijgbaar met behulp van een bèta-distributie met parameters $\alpha = \beta = 0,5$. Dit is te zien in figuur 2.10.



Figuur 2.10: Bèta-distributie

Voor de aankomst van pakketten bij de netwerkkaart is gekozen voor een Poisson-distributie. De simulatiesoftware die hiervoor is gebruikt, is Distributed Internet Traffic Generator. Meer informatie over de generator is terug te vinden in [**?**]. De gebruikte parameters voor het aantal pakketten per seconden en de grootte van de pakketten zijn terug te vinden onder de sectie resultaten van de hoofdstukken 3, 4 en 5. Meer informatie over de parameters gebruikt voor de evaluatie van de implementaties is terug te vinden in 2.7

Algemene informatie over de meetfactoren gebruikt in de verschillende soorten grafieken is te vinden in 2.7.



Figuur 2.11: Lay-out opstelling met achtergrondverkeer

2.6 Quality of Service

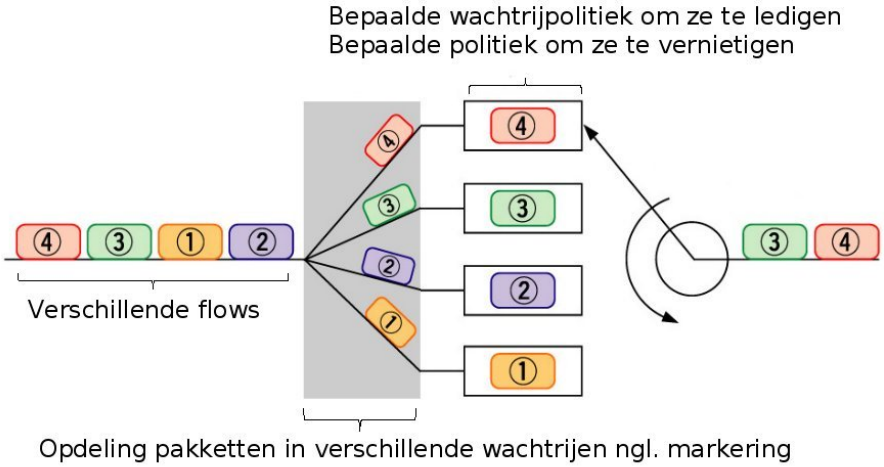
Quality of Service houdt in dat het netwerktrafiek aan bepaalde garanties wordt doorgestuurd. Dit kan met behulp van prioritering. In een normale situatie voorziet het internet enkel in een zgn. "Best-Effort" dienst. Dit houdt in dat er wordt geprobeerd alle data zo snel mogelijk over het netwerk te sturen. Men kan dit bekijken als een klasse met één enkele prioriteit. Belangrijke data zoals controleboodschappen krijgen hier geen voorrang op andere. Met prioritering wordt onderscheid gemaakt tussen verschillende types van verkeer en verzekert dat de belangrijkste gegevens zo snel mogelijk over het netwerk passeren. Men kan zo de kwaliteit van een datastroom garanderen. Garanties worden gemaakt op verscheidene gebieden, nl. bitsnelheid, vertraging, kans op verliezen van pakketten, enz.. Deze garanties zijn vooral belangrijk bij het transporteren van realtime informatie over netwerken die niet voldoende bandbreedte bieden om alle informatie door te voeren.

Er bestaan twee grote takken in QoS, nl. Differentated Services en Integrated Services. Het onderscheid tussen de twee wordt gemaakt op basis van de manier waarop ze de trafiek opdelen in klassen.

2.6.1 Werkingsprincipe

De implementatie van QoS met behulp van prioritering bestaat uit verschillende onderdelen, maar de kern is geïllustreerd in figuur 2.12. Deze afbeelding toont verkeer, opgedeeld in verschillende genummerde stromen, dat aankomt in een netwerkelement dat prioritering voorziet en het in een welbepaalde volgorde verlaat.

Allereerst vindt er een classificatie van de pakketten plaats. Zo kan er een onderscheid worden gemaakt tussen verschillende stromen. Dit kan gebeuren op verscheidene manieren. Zo kan dit op basis van een abonnement, gespecificeerd voor een gebruiker, of op basis van verschillende types van verkeer.



Figuur 2.12: Werkingsprincipe QoS met behulp van prioritering

Na de classificatie worden verschillende prioriteiten aan de pakketten toegekend, bekend als markering. Aan de hand van een studie van het netwerk en het gebruikte ledigingsmechanisme wordt bepaald hoe dit het best gebeurt. Voor de eenvoud kan hier verondersteld worden dat een stroom van een gebruiker die meer betaalt - een duurder abonnement heeft -, een hogere prioriteit krijgt.

De pakketten worden hierna in wachtrijen geplaatst. In welke wachtrij ze terecht komen, is afhankelijk van de prioriteit. Het idee is dat pakketten met dezelfde prioriteit geen voorrang hebben op elkaar en dus samen worden plaatst.

De laatste stap is het ledigen van wachtrijen. De lediging is afhankelijk van het gebruikte mechanisme, de zogenaamde scheduling. FIFO scheduling zal bijvoorbeeld één voor één alle pakketten uit een wachtrij nemen. Hier worden eerst alle pakketten uit de rij met de hoogste prioriteit gehaald. Als deze leeg is, begint het algoritme met de volgende wachtrij. Wanneer er tijdens het leeghalen van een wachtrij met lagere prioriteit, een pakket met hoogste prioriteit binnenkomt, zal dit eerst verstuurd worden. Daarna wordt het ledigen van de wachtrij met de tweede prioriteit weer hervat.

Het kan dus voorkomen dat pakketten met lagere prioriteit niet aan bod komen. Dit heet uithongering. Om dit tegen te gaan, bestaan er verscheidene andere scheduling algoritmes. Een voorbeeld van een algoritme dat rekening houdt met uithongering is Waited Fair Queueing[9].

2.6.2 Integrated Services

Integrated services, afgekort IntServ, wordt omschreven als een QoS systeem met fijne korrel. Het basisidee is dat er een onderscheid wordt gemaakt tussen verschillende stromen. Een stroom is gedefinieerd als een opeenvolging van pakketten met dezelfde bron en bestemmeling en die dezelfde route over een netwerk volgen. Hierbij zijn dus ook stromen die vanuit één computer

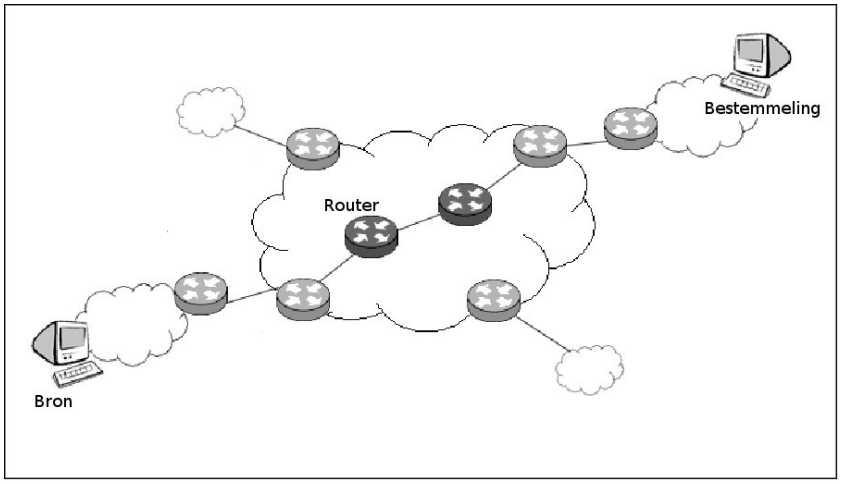
[9]WFQ werkt in cycli. Tijdens één cyclus gaat het alle wachtrijen af. Hierbij wordt er meer tijd aan een wachtrijen met hogere prioriteit dan lagere. Wel komen ze allemaal aan bod.

vertrekken, maar door verschillende applicaties worden doorgestuurd, verschillend. Zo ontstaat een fijne opdeling van het verkeer.

Op figuur 2.13 staat de lay-out van een typisch IntServ systeem. In deze topologie moet elke router in het netwerk voorzien zijn van de IntServ functionaliteit. Elke applicatie die een soort van garantie nodig heeft, moet een individuele reservatie maken bij elke router langs het pad naar de bestemmeling. Dit gebeurt met een protocol genaamd Resource Reservation protocol (RSVP). Meer informatie over RSVP is terug te vinden in [**?**].

Om zo een reservatie te maken, moet de zender laten weten aan de routers wat voor verkeer hij over het netwerk wil sturen en met welke prioriteit. Dit gebeurt door zogenaamde flowspec pakketten door te sturen naar de verschillende routers. Deze kijken dan of een stroom met de gevraagde restricties mogelijk is onder de huidige netwerkomstandigheden. Als het mogelijk is, wordt de stroom toegelaten, anders niet. Dit mechanisme noemt men toelatingscontrole. Eens de stroom is toegelaten, wordt gecontroleerd of ze wel voldoet aan de aangevraagde specificatie van het verkeer. Dit staat bekend als policing.

De routers zorgen er dan voor dat het verkeer met de gewenste restricties over het netwerk wordt gevoerd. Dit gebeurt door prioriteiten toe te kennen en scheduling algoritmes uit te voeren.



Figuur 2.13: Integrated Services topologie

2.6.3 Differentiated Services

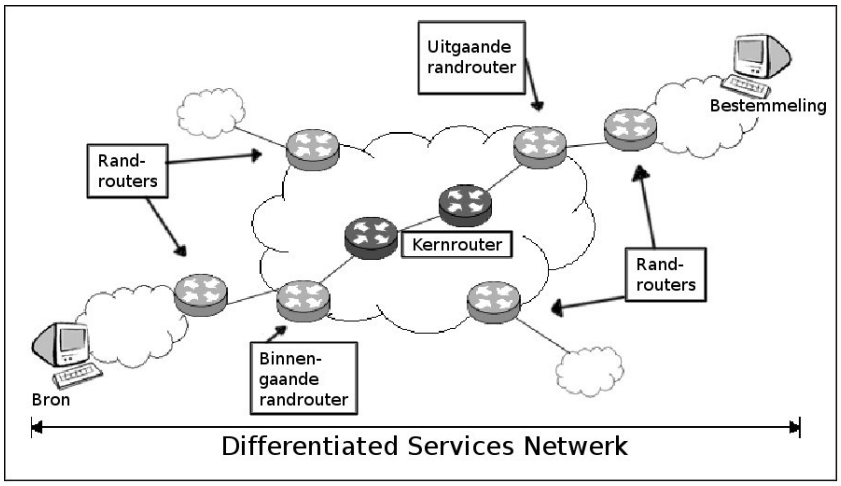
Differentiated services, afgekort DiffServ, wordt omschreven als een QoS systeem met grove korrel. Hier wordt er een onderscheid gemaakt tussen een veel kleiner aantal klassen van verkeer. De verschillende stromen worden geaggregeerd in grotere klassen. Welke stromen worden samengenomen, is eigen aan de implementatie. Zo ontstaat een ruwere opdeling van de trafiek.

Figuur 2.14 geeft een overzicht van de topologie. Er wordt een groter onderscheid gemaakt tussen de verschillende elementen. Het niet nodig dat elke router op het pad voorzien is van

de DiffServ functionaliteit. Enkel de randrouters worden hiervan voorzien. De routers worden opgedeeld in twee grote klassen, nl. de kernrouters en de randrouters.

De kernrouters hebben een zeer beperkte functionaliteit, men noemt dit de Per Hop Behaviour. Dit is dus het gedrag van één enkele router in plaats van het gedrag van een eind-tot-eind dienst. Deze beperkt zich tot het bekijken van de prioriteit en doorsturen van de pakketten naargelang het scheduling algoritme.

De randrouters hebben een uitgebreidere functionaliteit. Zij zullen, net zoals IntServ routers, pakketten een priotirisatie geven, reserveringen maken en toelatingscontrole uitvoeren. Zij vormen een soort van beschermmantel voor het ruggegraatnetwerk, waar enkel kernrouters aanwezig zijn. Ze controleren de ingaande en uitgaande trafiek.



Figuur 2.14: Differentiated Services topologie

2.6.4 Vergelijking

In tabel 2.5 is een overzicht voor de vergelijking tussen DiffServ en IntServ te vinden. De voor- en nadelen worden hier toegelicht.

Het verschil in korrel heeft tot gevolg dat er meer informatie in IntServ routers moet worden bijgehouden. Zij moeten voor elke stroom de kenmerken van het verkeer bijhouden. Bij DiffServ is er een grotere aggregatie en dus zijn er ook minder toestanden. Het groot aantal toestanden aanwezig in IntServ routers moet worden bewaard en periodisch ververst. De router moet elke stroom classificeren, priotiriseren, controleren en in wachtrijen plaatsen. Toelatingscontrole moet elke keer een verversing gebeurt, opnieuw worden uitgevoerd. Dit heeft tot gevolg dat in grote netwerken, waar veel stromen aanwezig zijn, deze routers zeer veel geheugen en rekenkracht nodig hebben. Dit is slecht voor de schaalbaarheid.

De fijne korrel bij IntServ zorgt er wel voor dat het verkeer beter kan worden gecontroleerd. Elke stroom wordt apart behandeld. Zo kan dus elke applicatie apart in het oog worden gehouden. Ook staat de fijnere granulariteit open voor complexere algoritmes voor markering. Daarom is deze vorm van QoS veel nauwkeuriger.

Het bekijken van vele stromen apart en hun effect op de totale trafiek is veel complexer dan het effect van grote geaggregeerde stromen.

Vanwege de problemen met complexiteit en schaalbaarheid van IntServ werd in deze thesis dan ook gekozen voor het DiffServ concept.

Tabel 2.5: Vergelijking DiffServ en Intserv

	IntServ	**DiffServ**
Informatie router	-	+
Complexiteit	-	+
Nauwkeurigheid	+	-
Schaalbaarheid	-	+
Rekenkracht router	-	+

2.7 Resultaten

Bij het streamen van een video worden videoframes opgedeeld in pakketten en doorgestuurd naar de bestemmeling. Deze pakketten vertrekken op een bepaald tijdstip bij de bron en komen iets later aan bij de bestemmeling. De pakketten worden daar gedecapsuleerd, samengevoegd en gedecodeerd naar videoframes. Hierbij moet de videoframe beschikbaar zijn voor decodering vooraleer ze moet worden afgespeeld. Er zijn hier dus drie verschillende tijdstippen te onderscheiden: de vertrektijd (van een pakket), de aankomsttijd (van een pakket) en de afspeeltijd (van een videoframe). In de resultaten worden deze op een grafiek weergegeven.

Zoals reeds vermeld, is het van groot belang voor de kwaliteit van de video dat er minder I- en P-frames verloren gaan dan B-frames. Het type frame dat verloren gaat is dus belangrijk bij de evaluatie van de verschillende implementaties. Deze worden op grafieken weergegeven bij de resultaten. Hierop zijn zowel het aantal verloren frames als het totaal aan frames voor elk type weergegeven.

Een ander type grafiek geeft de pakketvertraging weer. Een realtime-pakket wordt onbruikbaar als het aankomt na het tijdstip van gebruik. Het is dus van belang dat de vertraging laag wordt gehouden en ondertussen de variatie in de pakketvertragingen onder een bepaald peil blijft. De variatie in vertraging moet gebufferd worden om ervoor te zorgen dat de frames aangekomen zijn vooraleer ze nodig zijn om af te spelen. Als de variatie tussen de vertragingen groter is, ligt de maximale waarde ver van het gemiddelde. De buffer wordt wel gedimensioneerd om dit maximale verschil te overbruggen. Dit maximum wordt echter weinig benut. Een stroom wordt voor het afspelen gebufferd om de variaties in vertraging op te vangen. Hoe groter de buffer, hoe langer er moet worden gewacht. Het is dus van belang de variatie laag te houden om de buffer aan de ontvangstzijde te beperken tot een minimum. Op de grafiek staan 5 parameters: de vertraging, het gemiddelde, de maximale vertraging, de minimale vertraging en de variantie van de vertraging.

De maximale vertraging is de maximale van alle vertragingen en de minimale het minimum van alle vertragingen. De pakketvertraging wordt berekend als:

$$Pakketvertraging = Tijdstip\ aankomst\ pakket - Tijdstip\ vertrek\ pakket$$

Het gemiddelde van een steekproef van pakketvertragingen is:

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Hierbij is x_i de vertraging van één pakket en n het totaal aantal pakketten.

De variantie van een steekproef is een statistische maat voor de verschillen in de vertragingen en wordt berekend als volgt:

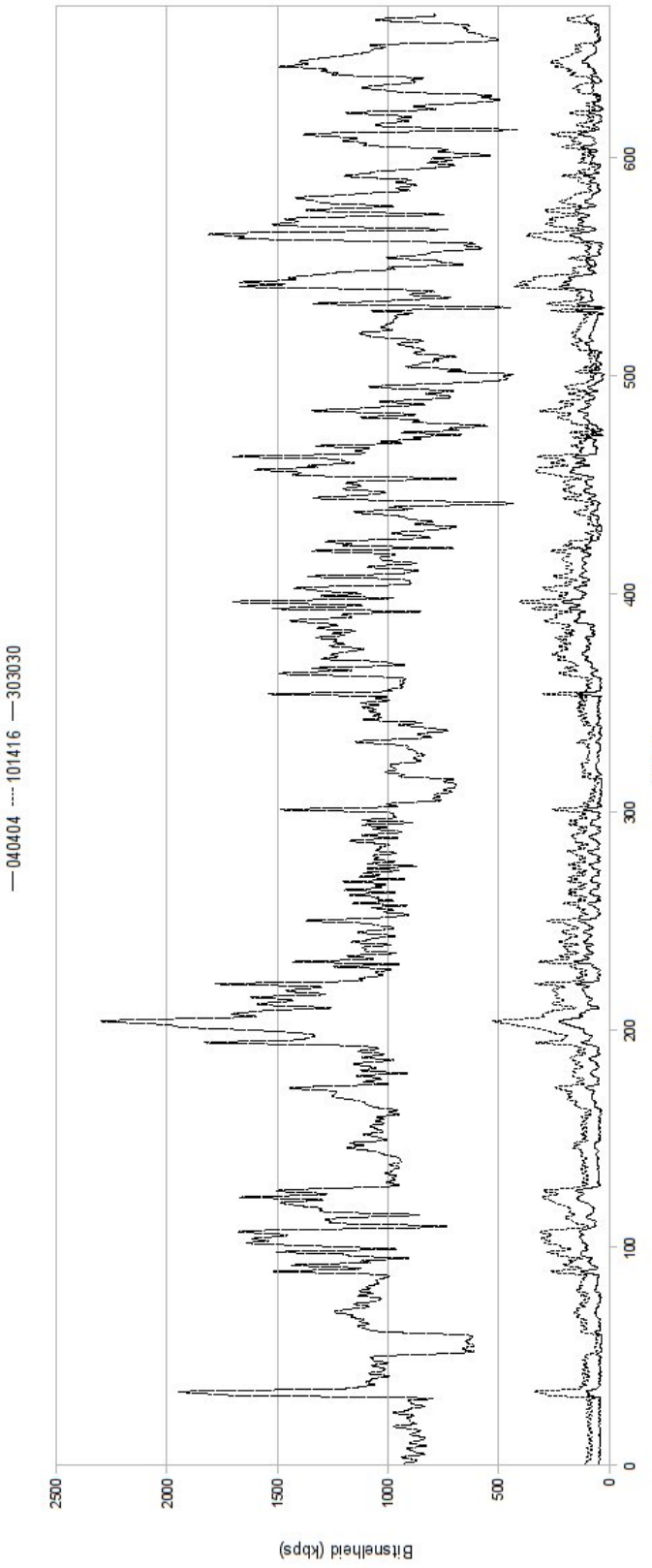
$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \overline{x})^2$$

Om resultaten te genereren, is een trace gekozen, zoals besproken in 2.2.3. Deze trace is verkregen uit een ruim aanbod van traces te vinden in [**?**]. In de testresulaten is de trace "Baseball with commercials"gebruikt. Dit is een videostroom met een variabele bitsnelheid in het QCIF[10]-formaat. De trace gebruikt een gelaagde stroom bestaande uit twee lagen . Een basislaag bestaande uit I- en P-frames en een verbeteringslaag bestaande uit B-frames. Dit is dus een tijdsgebaseerde techniek.

De stroom is verkrijgbaar in verschillende quantisatiegroottes, dus verschillende bitsnelheden. De bitsnelheden zijn te zien op figuur 2.15. Hoe lager de quantisatiefactoren, hoe hoger de bitrate. Bijvoorbeeld de stroom 101416 heeft een quantisatiefactor 10 voor de I-frames, 14 voor de B-frames en 16 voor de B-frames.

Standaard wordt in de testresulaten 040404 gebruikt. Bij aanpassing van de server, meer informatie terug te vinden in 5.1, wordt voor de hoge kwaliteit 040404 gebruikt en voor de lage kwaliteit 303030.

[10] 176×144 pixels



Figuur 2.15: Bitsnelheden testvideo aan verschillende quantisatiegroottes

Hoofdstuk 3

Basisopstelling

Dit hoofdstuk behandelt eerst het softwarepakket waarmee de router wordt geprogrammeerd. Dit softwarepakket is Click [**?**] en wordt besproken in 3.1.

Vervolgens beschrijft 3.2 de opstelling zoals gebruikt in het labo. Dit is de aanwezige hardware in de opstelling.

Het derde deel van dit hoofdstuk, sectie 3.3 gaat over de omgeving waarin gewerkt is. Dit behandelt de gebruikte software en besturingssystemen.

Het hoofdstuk 3.4 besluit met een eerste basisvoorbeeld van de router. Deze basis zal dienen voor de verdere opbouw van de router.

3.1 Inleiding tot Click

Voor de opstellingen werd Click gebruikt. Click is een software-architectuur voor het bouwen van flexibele en configureerbare routers. Iedere Click router is samengesteld uit pakketverwerkingsmodules[1], elementen genaamd. Iedere element voert een eenvoudige bewerking uit op het pakket zoals bijv.:

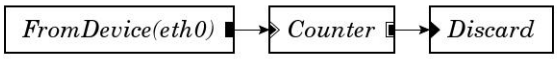
- het pakket classificeren
- het pakket in een wachtrij plaatsen
- het pakket prioriteiten geven
- het pakket naar netwerkkaarten sturen
- het pakket van netwerkkaarten halen

Een Click-configuratiebestand bestaat uit elementen die met elkaar verbonden zijn. De pakketten in de router passeren de elementen in de volgorde zoals ze verbonden zijn. Door de Click-architectuur zijn routers dus modulair uit te breiden door elementen toe te voegen. De elementen zijn geschreven in C++. Dit maakt het mogelijk om zelf elementen te schrijven, waarvan gebruik gemaakt werd.

Op figuur 3.1 staat een simpel voorbeeld dat de basis van Click duidelijk maakt. Links haalt het FromDevice element pakketten van de netwerk interface eth0. Deze pakketten worden

[1] Elementen die bewerkingen op pakketten uitvoeren.

vervolgens door de Counter, die de pakketten telt, gestuurd zodat ze tenslotte worden verwijderd in het Discard element.



Figuur 3.1: Een eerste eenvoudig voorbeeld dat de pakketten van eth0 telt en dan verwijdert.

De Click code zelf voor dit voorbeeld is:

```
FromDevice(eth0) -> Counter() -> Discard;
```

De figuren zeggen dus duidelijk wat er gebeurt. Daarom wordt in het vervolg ook van figuren gebruik gemaakt.

3.1.1 Element

Het element is het belangrijkste deel in Click. Ieder element bevat vijf delen:

- Elementklasse: deze naam geeft aan welke code uitgevoerd wordt als een pakket het element passeert en hoeveel poorten en handlers het element heeft. Ieder element komt overeen met het subklasse element.
- Poorten: geeft aan hoeveel poorten een element heeft. Er zijn geen beperkingen op het aantal uitgangen of ingangen. Iedere uitgang is verbonden met een ingang van een ander element. Ook kan iedere poort een ander functie hebben. Zo zijn er veel elementen die normale pakketten op de eerste uitgang uitsturen en foute pakketten op de tweede. Het aantal poorten van een element kan vast bepaald zijn of kan via de configuratietekst worden meegegeven. Iedere poort moet wel verbonden zijn met minstens één ander element. Er zijn drie verschillende soorten van poorten: push, pull en agnostisch. Het verschil wordt uitgelegd in 3.1.2.
- Configuratietekst: deze tekst geeft extra configuratieparameters mee aan het element. Mogelijkheden zijn: het aantal poorten, de netwerkkaart waar naar geluisterd moet worden, ...
- Methodeinterface: dit is een interface waar ieder ander element toegang toe heeft. Ieder element heeft dus de basisinterface die de methodes beschrijft voor het overbrengen van pakketten. Er kunnen ook ander interfaces gemaakt worden, zoals de interface van een wachtrij die zijn huidige lengte meedeelt aan andere elementen.
- Handlers: dit zijn methodes die ook naar de gebruiker geëxporteerd worden. Ze ondersteunen simpele lees- en schrijfbewerkingen en verlenen zo externe toegang tot de elementen van een configuratie. In de kernel mode is er voor iedere handler een bepaalde bestand in de /Click directory. Omdat de gebruiker makkelijk aan deze handlers kan, worden voornamelijk deze methodes gebruikt.

3.1.2 Push en pull

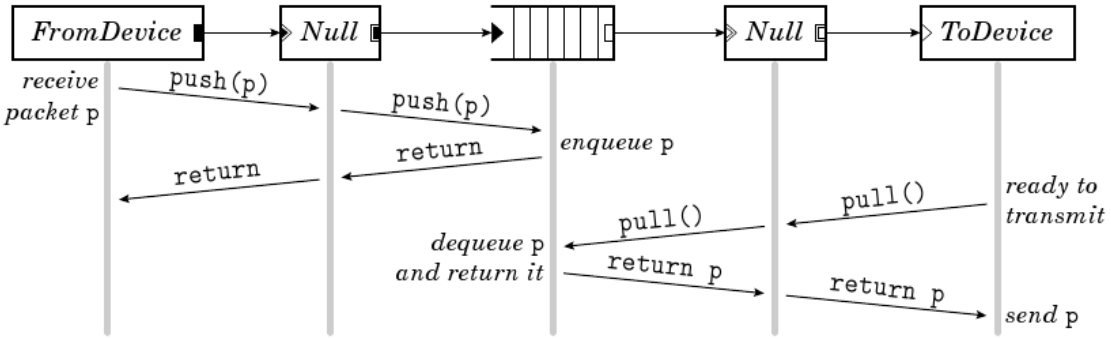
Zoals eerder vermeld in 3.1.1 ondersteunt Click drie soorten van poorten: de push, de pull en de agnostische poort.

Op een push poort starten de pakketten in het bronelement en worden stroomafwaarts geduwd naar de bestemmingselementen. Dit komt overeen met hoe de pakketten in de meeste routers passeren.

Op een pull poort daaraantegen, zal het bestemmingselement een pakkettransfer initialiseren. Het zal dus een pakket bij een element stroomopwaarts vragen. Als er geen pakket aanwezig is op dat tijdstip zal een lege pointer worden teruggegeven.

Een verbinding is bijgevolg alleen geldig als ze tussen 2 push of 2 pull poorten ligt.

Er is ook nog de mogelijkheid om de poorten agnostisch te maken. Click zal dan zelf beslissen of het een push of pull poort wordt, afhankelijk van de verbinding naar dat element. Figuur 3.2 laat ons zien hoe de push en pull verbindingen werken in een simpele router.



Figuur 3.2: Werking van push en pull

Push verwerking is aan te raden wanneer pakketten onaangekondigd arriveren. Dit is het geval bij pakketten van een netwerkkaart. De router moet deze pakketten dan verwerken op het moment dat ze aankomen, ook al zet hij ze maar gewoon in een wachtrij voor verdere verwerking.

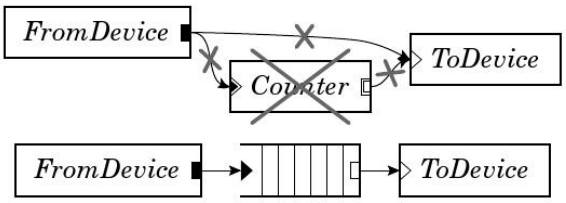
Pull verwerking is aan te raden wanneer de router het tijdstip van aankomst van een pakket moet kunnen controleren. De router mag alleen maar pakketten zenden naar een netwerkkaart als deze daar klaar voor is.

Figuur 3.3 toont enkele fouten op push en pull.

Pull verwerking staat in voor het beslissen over welk pakket er gezonden moet worden. Een Click Scheduler is een simpel element met één pull uitgang en verschillende pull ingangen. Zo een element reageert op een pull aanvraag op zijn uitgang en gaat dan één van zijn ingangen proberen. Als er geen pakket op deze ingang staat, zal hij een andere ingang proberen.

3.1.3 Pakket opslag

Click elementen hebben geen impliciete opslag op hun in- of uitgangen. Om opslag in een router te krijgen, is er een Queue element nodig. Dit is een wachtrij met een push ingang en een pull uitgang. Als er een pakket op de ingang wordt gepushed, zal het eerst in de wachtrij gaan



Figuur 3.3: Configuratie met 4 fouten (1) FromDevice push uitgang is aangesloten op ToDevice pull ingang. (2) Meer dan één verbinding op de FromDevice push uitgang. (3) Meer dan één verbinding naar de ToDevice pull ingang. (4) Een agnostisch element in een push pull omgeving. De onderste opstelling is wel correct omdat de Queue zorgt voor de omzetting van push naar pull. Het is een wachtrij.

totdat het eruit wordt gepulled. Er zijn verschillende soorten wachtrijen. De meest gekende is de FIFO. Op deze manier heeft de router de controle over een zeer belangrijke eigenschap: het opslaan van pakketten. Een Queue kan ook pakketten van verschillende elementen opslaan en kan deze pakketten ook weer vrijgeven aan verschillende elementen. Er kunnen dus verschillende verbindingen naar een Queue gaan en er kunnen er ook verschillende van vertrekken.

3.1.4 CPU planning

Click plant de router CPU met een takenwachtrij. De besturing van de router is een lus die iedere taak op de takenwachtrij uitvoert, één element per keer. Iedere taak is een element dat toegang wil tot de processor. Een element is dus een eenheid van CPU verwerking en van pakketverwerking. Een taak kan push en pull aanvragen initiëren. De meeste elementen worden nooit op een takenwachtrij geplaatst. Ze worden expliciet gepland wanneer hun push en pull methodes worden aangeroepen. Als een element zeer dikwijls push of pull aanvragen doet, wordt het in een takenwachtrij gezet. Dit is door Click voorzien voor elementen zoals FromDevice, ToDevice, ...

Er bestaat ook nog een andere manier in Click om de CPU te plannen en die is tijdgebaseerd. Als er een klok afgaat van een element wordt zijn bijhorende functie opgeroepen.

Click loopt in één thread. Iedere pakketoverdracht-methode (push of pull) moet terugkeren naar zijn oproeper voordat er een volgende taak kan worden uitgevoerd. De router zal dus blijven doorgaan met het verwerken van dat ene pakket totdat het ergens expliciet opgeslaan of verwijderd wordt. De plaatsing van de wachtrijen in een router is daarom zeer belangrijk voor de CPU planning. Als de wachtrijen laat in de router komen zal er eerst veel verwerkingstijd zijn vooraleer een pakket wordt opgeslaan in de wachtrij en dus voordat het volgende pakket aan zijn verwerking begint.

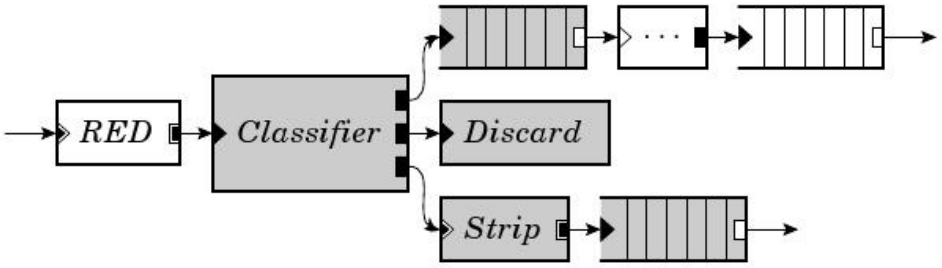
3.1.5 Stroomgebaseerde router context

Als element A de methodes van element B wil gebruiken, dan moet het eerst element B lokaliseren. Verbindingen lossen dit probleem voor pakketoverdracht op, maar niet voor andere

methodeinterface interacties. In plaats daarvan kan A refereren naar B door de naam van B. De configuratie tekst van A kan bijvoorbeeld de naam van B bevatten.

Of A kan een automatisch mechanisme gebruiken dat stroomgebaseerde routercontext heet. Stroomgebaseerde routercontext omschrijft waar pakketten, die op een gegeven element gestart zijn, eindigen na verschillende overdrachten. Het omgekeerde is ook waar. Het kan van pakketten die op een element aankomen kijken van waar deze pakketten vertrokken zijn. Dit bepaalt dus waar een pakket naartoe gaat in een overdracht. Praktisch gezien kan een element vragen waar een pakket overal passeert nadat het op zijn tweede uitgang is uitgestuurd. Het antwoord op deze vraag wordt berekend met stroomgebaseerde routercontext.

Het element RED is een voorbeeld op de stoomgebaseerde routercontext. Iedere Queue exporteert zijn huidige lengte door gebruik te maken van een methodeinterface. Elementen zoals RED zijn geïnteresseerd in deze informatie. Maar om de lengte van een Queue te vinden moet RED eerst weten waar deze queues zich voordoen. Deze informatie wordt verspreid via de stroomgebaseerde routercontext. In figuur 3.4 zullen de plaatsen van de twee grijze queues naar RED teruggegeven worden.



Figuur 3.4: Werking van stroomgebaseerde router context

3.1.6 Click taal

Click gebruikt geen grafieken maar tekstbestanden. De Click configuratiebestanden bestaan uit twee belangrijke delen, de constructies en de verbindingen.

Constructies declareren de elementen en verbindingen zorgen voor de link tussen verschillende elementen. De Syntax is eenvoudig genoeg om dit te leren uit een voorbeeld. De onderstaande code geeft de syntax van een zeer eenvoudige router:

```
// Declare three elements . . .
src :: FromDevice(eth0);
ctr :: Counter;
sink :: Discard;
// . . . and connect them together
src -> ctr;
ctr -> sink;
// Alternate definition using syntactic sugar
FromDevice(eth0) -> Counter -> Discard;
```

De taal bevat ook nog een abstractiemechanisme, samengestelde elementen genaamd. Dit laat de gebruiker toe om zelf een element te schrijven dat zelf uit meerdere elementen bestaat. Een gebruiker kan bijv. een Counter element, gevolgd door een Discard element in een samengesteld element zetten, CountDrop geheten. Op deze manier hoeft hij niet telkens de twee elementen in de Click-configuratie te zetten.

3.1.7 Installeren van Click configuraties

Er zijn twee mogelijke manieren om een router te installeren. De user en de kernel mode

De user mode is goed voor het debuggen van configuraties. De pakketten worden dan door Linux en door Click behandeld. In deze mode is het mogelijk output naar het scherm sturen. Om een configuratie in user mode te installeren is *Click router1.Click* het commando.

De kernel mode wordt gebruikt als de router volledig zonder fouten werkt in een productie-omgeving. Het voordeel van de kernel mode is dat de handlers van een element beschikbaar zijn in het bestandssysteem van Linux. In de rest van de thesis ligt de nadruk dan ook op de kernel mode. Het commando *Click-install router1.Click* installeert deze configuratie in de kernel mode.

Het installeren van een nieuwe configuratie verwijdert de oude configuratie. Al de pakketten in de wachtrijen worden verwijderd, al de handlers worden gereset, ... Er bestaan in Click echter twee methodes om over te schakelen van een configuratie zonder informatie te verliezen.

- Handlers: ieder element kan over een aantal handlers beschikken. Dit zijn toegangspunten voor interactie met de gebruiker. Ze bestaan uit bestanden in de /Click directorie. Als een gebruiker het aantal pakketten dat door een Counter c is gepasseerd wil weten, dan kan hij dat doen door het commando *cat /Click/c/count* uit te voeren. Het commando *echo 0 > /Click/c/reset* reset de counter. Het eerste is een leeshandler terwijl het tweede een schrijfhandler is
- Hotswapping: sommige configuratieveranderingen, zoals het toevoegen van een element, zijn te complex om met handlers uit te voeren. In dit geval moet de gebruiker een nieuwe configuratie schrijven. Als hij wil dat er toch geen pakketten en informatie verloren gaat, kan hij zijn nieuwe configuratie laden met het commando *cp router2.Click /Click/hotconfig*. Dit werkt alleen maar als de structuur van de routerconfiguraties op elkaar lijken.

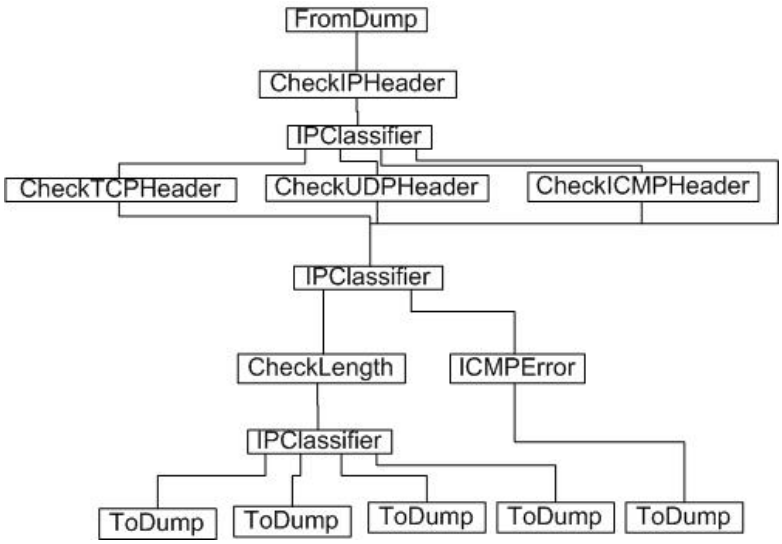
3.1.8 Programmeren van eigen elementen

In de thesis worden elementen op maat gemaakt. Omdat de standaardelementen nodig om het probleem op te lossen niet aanwezig zijn. Ieder Click-element komt overeen met de subklasse van de C++ klasse Element, wat ongeveer 20 virtuele functies heeft. De meeste van deze functies zijn standaard voor ieder element. Een zelfgeschreven element zal maar een zestal van deze functies overschrijven. Enkele functies zijn: push, pull en initialize.

Subklasses van Element zijn vrij makkelijk te schrijven wanneer de syntax van het programmeren in Click en C++ gekend is. De onderstaande code stelt een element voor dat het pakket doorlaat zonder er wijzigingen aan aan te brengen.

```
class NullElement: public Element { public:
NullElement() { add_input(); add_output(); }
const char *class_name() const { return "Null"; }
NullElement *clone() const { return new NullElement; }
const char *processing() const { return AGNOSTIC; }
void push(int port, Packet *p) { output(0).push(p); }
Packet *pull(int port) { return input(0).pull(); }
};
```

3.1.9 Een simpel voorbeeld



Figuur 3.5: Click-voorstelling van virtuele router

De werking van de router uit figuur 3.5 begint bij het element FromDump. Dit element leest pakketten, die eerder opgeslagen zijn met tcpdump, uit het bestand en behandelt ze alsof het echte pakketten waren. Het volgende element dat een pakket dan op zijn pad tegenkomt, is CheckIPHeader. Dit element kijkt of het IP pakket een aanvaardbare[2] lengte heeft en of de IP versie, lengte van de hoofding en checksum juist zijn. Het element controleert ook of de IP-adressen in het pakket geldig zijn. Het zet ook het bestemming IP-adres in de pakket-annotatie. Een annotatie is een extra hoofding die Click aan een pakket meegeeft zonder het eigenlijke pakket te veranderen. Meer informatie over annotaties is te vinden in 3.1.10. Als pakketten geldige IP pakketten zijn, worden ze doorgestuurd naar poort 0. Foute pakketten gaan naar poort 1, tenzij deze niet gespecifieerd is. Dan zullen de pakketen worden verwijderd.

Na de IP-hoofding-controle gaat het pakket naar een IPClassifier, een element dat de pakket-stroom opsplitst, afhankelijk van de inhoud van de pakketten. In dit geval kijkt de IPClassifier of het pakket een TCP, UDP, ICMP of een ander pakket is. Vervolgens gaan de verschillende pakketten een eigen weg. De weg die ze volgen is afhankelijk van de uitgang waarop de classifier zijn pakketten uitstuurt.

[2]Voor de definitie van aanvaardbaar, zie 2.3

Iedere uitgang van de IPClassifier wordt dan onderworpen aan een specifieke controle. Voor TCP bijvoorbeeld, is dit CheckTCPHeader. Dit is duidelijk op Figuur 3.5 te zien.

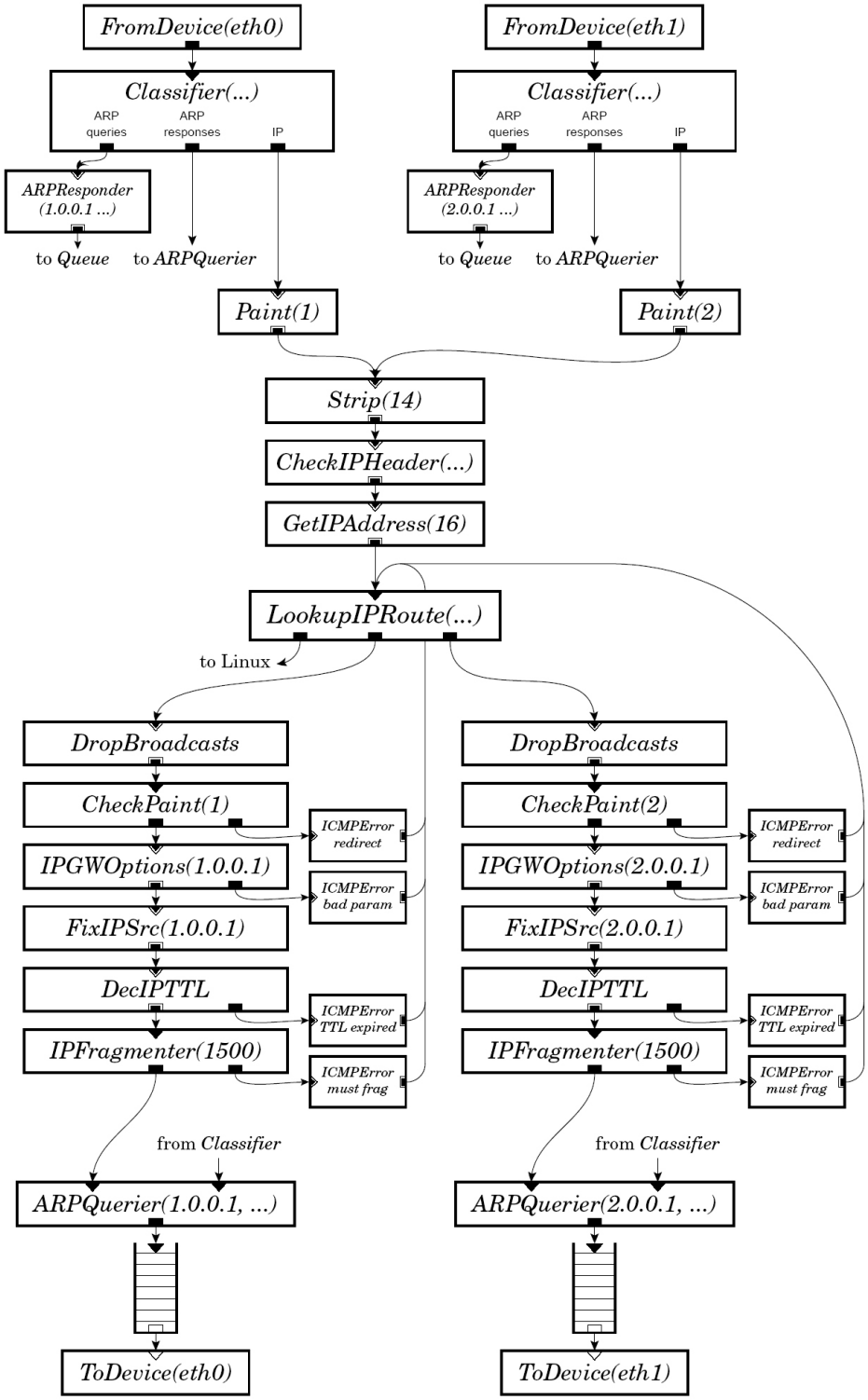
Dan komen de pakketten weer samen en gaan ze naar een IPClassifier. Dit element kijkt of de TTL van het pakket groter is dan 0. Moest dit niet meer het geval, dan wordt het pakket op de tweede poort uitgestuurd. Hier passeert het een element ICMPError, dit zendt een ICMP error pakket uit. Als laatste stap wordt dit in een dumpbestand geschreven. De eerste uitgang van deze IPClassifier zijn elementen met een nog geldige TTL. Deze pakketten worden dan gecontroleerd op hun lengte. Te grote pakketten worden weer op de tweede uitgang uitgezonden als die er is. Als die er niet is, worden deze pakketten verwijderd. Voordat de pakketten in een dumpbestand worden geschreven, gaan ze door een IPClassifier die de pakkettenstroom opdeelt volgens het bestemming-IP-adres. De code van deze router staat in A.1. In hetgeen volgt wordt er met figuren gewerkt.

3.1.10 Een router-voorbeeld

Deze sectie legt uit hoe een geavanceerdere router als in figuur 3.6 wordt gemaakt. (Deze router is compatibel met de standaarden. Dit is niet zo evident.) Stap voor stap wordt uitgelegd wat de verschillende elementen in deze router doen. Om IP pakketten door te sturen hebben we enkele elementen nodig met eenvoudige functies. Een voorbeeld van zo een element in deze router is de DecIPTTL. Dit element vermindert het TTL veld in de hoofding van een IP pakket met één en kijkt of de bekomen waarde nul is of niet. Als de waarde nul is, wordt het pakket naar de tweede uitgang van het element gestuurd. Is de waarde niet nul, dan wordt het pakket op de eerste uitgang uitgestuurd. Deze beslissingen hebben rechtstreeks te maken met de inhoud van het pakket en niet met beslissingen die eerder in de router genomen werden. Sommige elementen hebben informatie, over beslissingen die eerder in de router genomen werden, wel nodig. Maar deze zijn niet toegankelijk in het gewone pakket. Daarom voegt Click nog een extra datapakketje toe aan iedere pakket, dit is te vergelijken met een soort extra hoofding. Deze extra hoofding heet een annotatie. Enkele annotaties zijn:

- Bestemming IP-adres: elementen die bezig zijn met de bestemmig van pakketen gebruiken deze annotatie i.p.v. de echte hoofding. Op deze manier kunnen ze de annotatie veranderen zonder dat ze het eigenlijke pakket veranderen. Dit is nuttig als we het IP-adres van de volgende gateway[3] moeten zetten. GetIPAddress kopieert het bestemmingsadres van het pakket in de annotatie. LookUpIPRoute vervangt de annotatie door het IP-adres van de volgende gateway. ARPQuerier gebruikt de informatie van de annotatie om de MAC adressen van het pakket juist te zetten.
- Paint: het Paint-element markeert pakketten met een kleur. CheckPaint stuurt alle pakketten door naar zijn eerste uitgang, maar de pakketten met de gespecificeerde kleur worden gekopieerd naar zijn tweede uitgang. Deze router gebruikt Paint om te kijken of een pakket dat op een interface binnenkomt ook op diezelfde interface wordt uitgezonden. Als dit het geval is, wordt er een ICMP redirect gegenereerd.

[3]In deze context is een gateway een netwerkelement dat de verbinding tussen twee subnetwerken voorziet. Als een netwerkelement boodschappen wil versturen naar een ander element buiten zijn domein, stuurt het netwerkelement deze naar de gateway.



Figuur 3.6: Een geavanceerdere router

- Link-level broadcast vlag: FromDevice zet deze vlag op alle broadcast pakketten. De router controleert deze vlag zodat geen broadcast pakketten naar de andere interface worden gezonden.
- ICMP parameter problem pointer: dit stuk van de hoofding wordt gezet door het IPGWOptions-element bij slechte pakketten. Deze informatie wordt dan ook gebruikt voor het construeren van ICMP error pakketten.
- Fix IP source vlag: het IP bronadres van een ICMP error pakket moet het adres zijn van de interface waarop de error wordt gegenereerd. Het ICMPError element kan deze interface niet voorspellen. Bij het voorkomen van een fout genereert ICMPError een pakket. De vlag wordt dan gezet en er wordt een standaardwaarde aan het bronadres gegeven. Als geweten is op welke interface het ICMP error pakket wordt uitgezonden, kan het element FixIPSrc het bron IP adres juist zetten.

In de router in figuur 3.6 komen de pakketten binnen op 2 netwerkkaarten. Vervolgens gaan de pakketten naar een classifier die de pakketten opdeelt in: ARP queries, ARP Responses en IP pakketten. Deze opdeling zorgt ervoor dat het pakket de juist Ethernethoofding heeft met de juiste MAC adressen voor bron en bestemming.

De werking van ARP is als volgt. Er komt een pakket binnen op interface1 dat bestemd is voor interface2. De oorspronkelijke Ethernethoofding wordt van het pakket afgehaald door het Strip element. Wanneer het pakket aankomt op de ARPQuerier van de tweede interface moet deze een nieuwe Ethernethoofding met het juist MAC paar op het pakket zetten. De ARPQuerier weet niet welk MAC adres hij moet gebruiken als bestemming. Hij stuurt dus een ARPRequest uit met het IP dat in de annotatie zit. Hierin zit de vraag “Wie weet welk MAC adres dit IP adres heeft”. Dit pakket wordt gebroadcast op de tweede interface. Het pakket komt op een ARPResponder aan en deze antwoordt met juiste MAC adres. Nu kan de ARPQuerier de juiste Ethernethoofding op het pakket zetten en dit uitsturen op de interface.

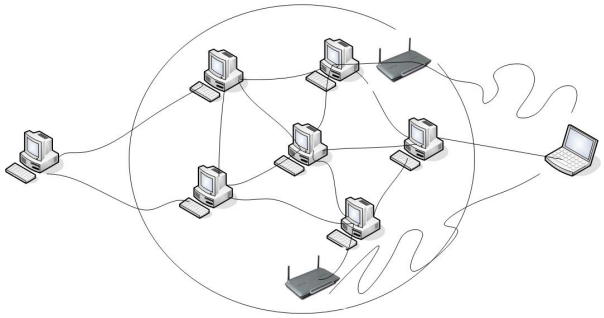
Eens de juiste MAC adressen gevonden zijn, is het pakket een geldig IP pakket. Het pakket komt dan op de derde uitgang van de classifier. Dan wordt de paint-annotatie van het pakket gezet door het Paint element zodat elementen die van en naar dezelfde interface gaan geblokkeerd worden en er een ICMP redirect error wordt gegenereerd door het CheckPaint element. Vervolgens wordt de Ethernethoofding van het pakket gehaald, gekeken of de IP hoofding geldig is en de IP bestemming in de annotatie gekopieerd[4].

LookUpIPRoute kijkt dan op welke uitgang hij het pakket moet sturen, afhankelijk van het bestemmings-IP-adres. Vervolgens doorloopt het pakket het DropBroadCast element wat ervoor zorgt dat broadcast pakketten verwijderd worden.

Dan wordt de paint nagekeken zoals beschreven bij de voorbeelden van annotaties. Het enige element dat nog niet beschreven is, is de IPFragmenter. Dit element splitst pakketten die groter zijn dan 1500 bytes op in pakketten kleiner of gelijk aan 1500 bytes. Als het pakket de Don’t Fragment vlag bezit, worden de pakketten op de tweede uitgang uitgestuurd met de bijhorende foutboodschap. Na dit fragmenteerelement passeren ze de ARPQuerier om vervolgens in een wachtrij terecht te komen. Hier staan ze ter beschikking van de netwerkkaart.

[4]Het kopiëren is nodig omdat LookUpIPRoute de annotatie nodig heeft.

3.2 De opstelling



Figuur 3.7: Schematische voorstelling van het internet

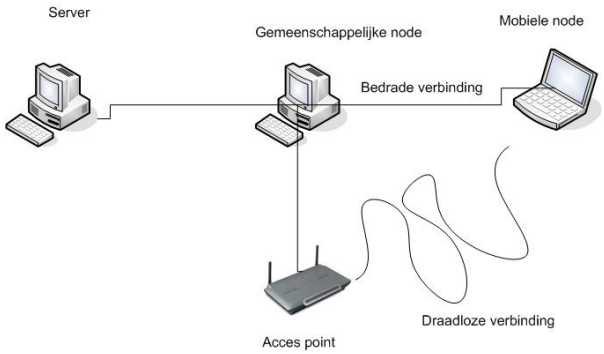
Het doel is verschillende computers te configureren met behulp van Click om zo een testomgeving te bekomen. Het referentiepunt is de structuur van het internet. Deze is te vergelijken met figuur 3.7. Dit is niet anders dan computers die met elkaar verbonden zijn. Naar de gebruiker bestaat een draadloze verbinding en een bedrade verbinding. Voor de beoogde doelstellingen is het voldoende de figuur te beperken. Eens de principes gekend, is het triviaal deze uit te breiden naar grotere opstellingen. De eerste opstelling is dus beperkt tot figuur 3.8. Op deze 3 computers wordt Click geïnstalleerd om de resultaten te genereren.

De server bevat één netwerkkaart (IP 192.168.6.1) via een UTP kabel aangesloten op de eerste netwerkkaart (IP 192.168.6.2) van de gemeenschappelijke node. De tweede netwerkkaart (IP 192.168.1.100) is via het accespunt (IP 192.168.1.1) verbonden met de draadloze netwerkinterface (IP 192.168.1.2) van de mobiele node. De derde netwerkkaart (IP 192.168.0.1) is via een UTP kabel verbonden met de netwerkkaart (IP 192.168.0.2) van de mobiele node.

Bij het gebruiken van Click in kernel mode, worden pakketten niet doorgegeven naar Linux tenzij dit expliciet gebeurt met behulp van een virtuele netwerkinterface. Deze voorziet een extra laag tussen Linux en de Click. Linux denkt te communiceren met een netwerkkaart, maar in feite communiceert Linux met de virtuele interface. Daarom werd iedere node waar Click in kernel mode op draait, van een virtuele netwerkinterface voorzien. Met behulp van deze interface kunnen pakketten naar Linux worden gestuurd en Linux ook pakketten naar de buitenwereld sturen. Voor de mobiele node gebruiken heeft deze interface het IP 192.168.3.1, voor de gemeenschappelijke node 192.168.4.1. Deze virtuele interfaces worden aangemaakt met het FromHost en ToHost element. Meer uitleg over het ToHost-element is terug te vinden in 4.1.2.

Een samenvatting van de gebruikte IPs, standaardroutes[5], namen van de interfaces en types verbinding is terug te vinden in tabel 3.1. Om het overzicht te bewaren is figuur 3.8 ook gegeven. De mobiele node en gemeenschappelijke node zijn, zoals hiervoor aangegeven, bereikbaar via één enkel IP. Dit is merkbaar aan de ingevoerde routes. Voor de gemeenschappelijke node is dit aangegeven met het IP 192.168.4.1. Linux denkt dus te communiceren met dit virtueel apparaat. Zoals te zien in tabel 3.1 zijn er in werkelijkheid zijn er drie netwerkkaarten: een

[5]Een standaardroute is een route waarnaar de netwerkkaart een pakket stuurt wanneer het bestemmings-IP zich niet op het subnetwerk van de kaart bevindt.

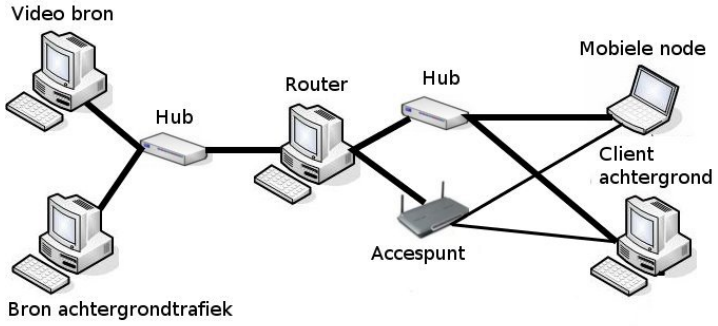


Figuur 3.8: De testopstelling

verbinding met de videobron (server), een verbinding met de mobiele node en een verbinding met het accespunt.

Tabel 3.1: Overzicht van de verschillende gegevens van onze basis opstelling

Node	Bestemming	IP	Interface	Verbinding	Route
Server	Gemeenschappelijke node	192.168.6.1	eth0	draad	192.168.6.2
Gemeenschappelijke node	Server	192.168.6.2	eth0	draad	192.168.4.1
Gemeenschappelijke node	Mobiele node	192.168.0.1	eth2	draad	192.168.4.1
Gemeenschappelijke node	Acces point	192.168.1.100	eth1	draad	192.168.4.1
Mobiele node	Gemeenschappelijke node	192.168.0.2	eth1	draad	192.168.3.1
Mobiele node	Acces point	192.168.1.2	ath1	draadloos	192.168.3.1



Figuur 3.9: Lay-out testopstelling met achtergrondverkeer

De basisopstelling werd uitgebreid om de mogelijkheid te voorzien achtergrondverkeer te versturen over het netwerk. Dit gebeurt zoals eerder beschreven in 2.5 en zichtbaar in figuur 3.9. Tabel 3.2 vat de gegevens samen. Ook hier is een virtuele interface aanwezig voor de communicatie

met de Client voor het achtergrondverkeer. Zoals te zien in de tabel, heeft deze een route naar 192.168.5.1. Dit is het IP van zijn virtuele interface.

Tabel 3.2: Overzicht van de verschillende gegevens van onze totale opstelling.

Node	Bestemming	IP	Interface	Verbinding	Route
Videobron	Hub1	192.168.6.1	eth0	draad	192.168.6.2
Bron achtergrondtrafiek	Hub1	192.168.6.3	eth0	draad	192.168.6.2
Router	Hub1	192.168.6.2	eth0	draad	192.168.4.1
Router	Accespunt	192.168.1.100	eth1	draad	192.168.4.1
Router	Hub2	192.168.0.1	eth2	draad	192.168.4.1
Mobiele node	Hub2	192.168.0.2	eth1	draad	192.168.3.1
Mobiele node	Accespunt	192.168.1.2	ath1	draadloos	192.16..3.1
Mobiele node	Router	192.168.0.2	eth1	draad	192.168.3.1
Client achtergrond	Accespunt	192.168.1.6	ath1	draadloos	192.168.5.1
Client achtergrond	Hub2	192.168.0.6	eth0	draad	192.168.5.1

3.3 De omgeving

Zoals uitgebreid besproken in 3.1 wordt Click gebruikt voor het configureren van de verschillende computers in de testopstelling. Omdat Click een programma is dat alleen in Linux werkt, zijn de testcomputers hierin geconfigureerd. In de thesis is gekozen voor Debian Etch [**?**] als besturingssysteem.

Voor het streamen van video hebben we gebruik gemaakt van VLC [**?**]. Dit is een streaming-applicatie dat meerdere soorten van streams ondersteunt. VLC voorziet de mogelijkheid om te streamen over TCP en UDP, maar ook over RTP. Er zijn ook verschillende coderingsalgoritmes voorzien. Hiervan wordt gebruik gemaakt bij transcoding.

Ook werd bash scripting [**?**] toegepast en voor het schrijven van elementen werd de C++ syntax [**?**] gebruikt.

Om de draadloze netwerkkaart optimaal te kunnen aansturen in Click werd MADWifi geïnstalleerd [**?**]. MADWifi is niet compatibel met alle chipsets. Hiervoor werd een netwerkkaart met de Atheros 5212 chipset gebruikt.

Tot slot hebben we voor het achtergrond verkeer gebruik gemaakt van een FTP server [**?**] en van een trafiekgenerator [**?**].

3.4 Implementatie 1: Één verbinding

Een netwerk dat zich bewust is van zijn omgeving wordt in het vervolg van de thesis geleidelijk opgebouwd. Elke belangrijke wijziging aan de bron (server), router (gemeenschappelijke node) en bestemmeling (Client) wordt opgedeeld in verscheidene implementaties. Elke implementatie

bevat extra functionaliteit en vormt een leidraad doorheen het denkproces van het opbouwen van het netwerk.

De eerste implementatie is het startpunt, een soort prototype. De router heeft nog weinig functionaliteit. Het enige wat hij doet is een video, die de server streamt, via de bedrade verbinding doorsturen naar de mobiele node. Deze speelt de film af.

3.4.1 Server

Op de server is VLC geïnstalleerd. Zoals in 2.3 beschreven, wordt gebruik gemaakt van het UDP protocol. Dit kan in VLC worden ingesteld. De route voor de server wordt geconfigureerd zodat de gateway in het labo niet gebruikt wordt. Het verkeer moet immers doorgestuurd worden naar de gemeenschappelijke node. Dit gebeurt met behulp van de volgende commando's in Linux: *route del default; route add default gw 192.168.6.2;*. Hiervoor is Click niet nodig.

3.4.2 Router (Gemeenschappelijke node)

Op de gemeenschappelijke node wordt wel een Click configuratie gemaakt. Er wordt niet met de standaard Linux routetabellen gewerkt. Zo wordt een maximale vrijheid over de router bekomen. Figuur 3.10 geeft de Click configuratie van de eerste opstelling. Op deze opstelling wordt gedurende de volledige thesis verdergewerkt. Dit is de minimale basis om pakketten via een Click-configuratie op hun juiste plaats te krijgen. De Click-configuratie is terug te vinden in A.2.

Pakketten worden door FromDevice-elementen van de lijn geplukt. Als volgt wordt er een ander MAC-adrespaar aan verbonden. Hierna worden de pakketten via het ToDevice-element op de bedrade verbinding naar de bestemming gestuurd.

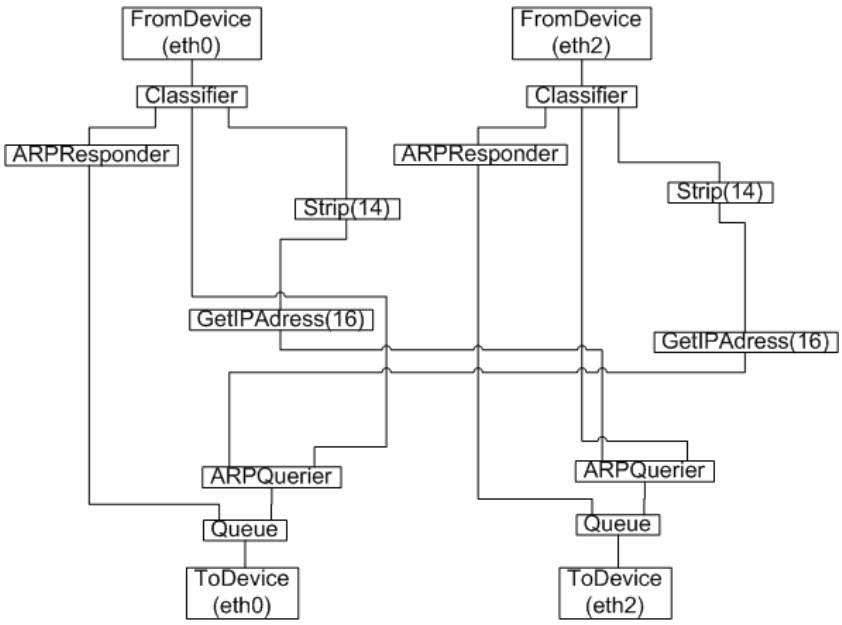
3.4.3 Client (Mobiele node)

Op de Client wordt ook VLC geïnstalleerd om de gestreamde film te kunnen bekijken. Ook worden we hier de IP-adressen van beide netwerkkaarten geconfigureerd.

3.4.4 Hoe resultaten genereren

Om resultaten te genereren wordt er gebruik gemaakt van het Click-element FakeStream. Meer informatie hierover is terug te vinden in 5.1.

Om de pakketten op de server en de mobiele node op te slaan, wordt gebruik gemaakt van Wireshark [**?**]. Dit programma onderschept alle pakketten die op de interfaces worden gezet. Deze pakketten worden voorzien van allerhande informatie zoals o.a. het gebruikte protocol, de aankomsttijd en de grootte van het pakket. De informatie nodig voor de resultaten wordt opgevangen en opgeslaan in twee verschillende dump-bestanden, één bestand staat op de bron en één op de Client.



Figuur 3.10: Eerste configuratie

Uit het dump-bestand op de Client wordt het pakketnummer en de aankomsttijd onttrokken. Naderhand wordt dit pakketnummer vergeleken met het pakketnummer op de bron. Per aangekomen pakket is immers de informatie dezelfde als die er verstuurd is. Er kunnen natuurlijk wel pakketten verloren gaan. Dit kan worden opgemerkt door de pakketnummers te vergelijken. Deze informatie wordt uit de dump-bestanden gehaald met behulp van volgend script:

```
1  #!/bin/bash
2  # what is what?
3  echo start
4  echo packetnumber, arrivaltime > totaal
5
6  #start with first packet
7  COUNTER=1
8  lines=$(cat dump |grep Arrival | wc -l)
9  let lines=lines+1
10 #loop through packets
11 while [ $COUNTER -lt $lines ]
12 do
13 #get packetnumber
14 packetnumber=$(cat dump | grep ^0000 | cut -d "~" -f 3 | cut -c 1- | awk NR==$COUNTER)
15
16 #get arrivaltime
17 arrivaltime=$(cat dump | grep Arrival | cut -d " " -f 11 | awk NR==$COUNTER)
18 uur=$(echo $arrivaltime | cut -d ":" -f 1)
19 minuut=$(echo $arrivaltime | cut -d ":" -f 2)
20 second=$(echo $arrivaltime | cut -d ":" -f 3)
21 time=$(echo $uur*3600+$minuut*60+$second | bc)
22 echo $COUNTER
```

```
23 let COUNTER=COUNTER+1
24 done
```

De code op lijn 4 schrijft in het bestand "totaal" twee kolommen voor de pakketnummers en aankomsttijden. Lijn 8 telt in het dump-bestand het aantal pakketten. Op deze manier kan de lus het juiste aantal keer herhaald worden. De code op lijn 14 zoekt het pakketnummer. De code van lijn 16 tot 21 zet de tijd om in seconden, dit is nodig om de vertrektijd en aankomsttijd te vergelijken.

Op de bron verwerkt een ander script het tweede dump-bestand. Dit script heeft het bestand "totaal" van de client nodig om te kunnen fungeren:

```
1   #!/bin/bash
2   echo nummer packetnumber departtime arrivaltime frametype framenumber
2'  displaytime framesize > goed
3   lines=$(cat alles | wc -l)
4   let lines=lines+1
5   COUNTER=1
6
7   while [ $COUNTER -lt $lines ]
8   do
9   #get frametype
10  frametype=$(cat alles | grep inhoud: | cut -d "~" -f 2| cut -c 1-|awk NR==$COUNTER)
11  #get packetnumber met H of L
12  packetnumber=$(cat alles | grep inhoud: | cut -d "~" -f 3| cut -c 1-| awk NR==$COUNTER)
13  #get packetnumber zonder H of L
14  #packetnumberz=$(cat alles | grep inhoud: | cut -d "~" -f 3| cut -c 2-| awk NR==$COUNTER)
15  #get framenumber
16  framenumber=$(cat alles | grep inhoud: | cut -d "~" -f 4| cut -c 2- |awk NR==$COUNTER)
17  #get displaytime
18  displaytime=$(cat alles | grep inhoud: | cut -d "~" -f 5| cut -c 2- |awk NR==$COUNTER)
19  #get framesize
20  framesize=$(cat alles | grep inhoud: | cut -d "~" -f 6| cut -c 2- |awk NR==$COUNTER)
21  #get departtime
22  departtime=$(cat dump | grep Arrival | cut -d " " -f 11 | awk NR==$COUNTER)
23  uur=$(echo $departtime | cut -d ":" -f 1)
24  minuut=$(echo $departtime | cut -d ":" -f 2)
25  second=$(echo $departtime | cut -d ":" -f 3)
26  departtimeseconds=$(echo $uur*3600+$minuut*60+$second | bc)
27
28  nummer=$(cat totaal | grep $packetnumber, | cut -d "," -f 1| cut -c 1- )
29  echo $packetnumber $nummer
30  if [ $nummer !=NULL ]
31  echo in de lus
32
33  nummer=$(cat totaal | grep $packetnumber, | cut -d "," -f 1| cut -c 1- )
34  arriveltimeseconds=$(cat totaal | grep $packetnumber,| cut -d "," -f 2| cut -c 2- )
35
36
37  echo $nummer $arriveltimeseconds
38  echo $COUNTER $packetnumber $departtimeseconds $arriveltimeseconds $frametype
38' $framenumber $displaytime $framesize >> goed
```

```
39  else
40  echo uit de lus
41  echo $COUNTER $packetnumber $departtimeseconds packetloss $frametype $framenumber
41' $displaytime $framesize >> goed
42  fi
43
44
45  #echo $COUNTER
46  let COUNTER=COUNTER+1
47  done
```

Op lijn 2 worden de kolommen van de gegevens in het bestand "goed"geplaatst. De volgende regel zorgt ervoor dat de lus het juiste aantal keren wordt uitgevoerd. Een tweede teller op regel 6 is om de plaats in "totaal"bij te houden. Deze tellers verwijzen naar dezelfde pakketten. De ene bij vertrek en de andere bij aankomst. Regels 9 tot 26 verzamelen de gegevens in de pakketten. Lijnen 28 tot 42 zetten de gegevens van de Client in "goed".

Zo wordt een bestand "goed"gecreëerd met de gegevens aanwezig in de pakketten gegenereerd door FakeStream. De gegevens zijn: pakketnummer, pakketnummer van de desbetreffende stroom, vertrektijd, aankomsttijd, frametype, framenummer, afspeeltijd en framegrootte.

3.4.5 Resultaten

De eerste resultaten zijn bedoeld om een algemeen beeld te vormen over het netwerk en de wijze waarop de resultaten worden voorgesteld. Er zijn drie soorten resultaten: tijdige levering, verloren frames en vertraging. De meeste zijn besproken in 2.7. Er treden variaties op in de voorstelling naargelang de relevantie van een meting in de geteste implementatie.

Gezien de beperkte functionaliteit van deze implementatie zijn er niet veel testen mogelijk. Op dit moment is er maar één medium voorzien, nl. het bedrade. Er gebeurt dus geen handover bij deze testen. Dit geeft een goed beeld van de karakteristieken van een bekabeld netwerk. Zo een netwerk heeft namelijk een kleine vertraging en een zeer lage variantie op de vertraging in vergelijking met een draadloos netwerk.

Twee testen zijn uitgevoerd op dit testbed. Bij de eerste test wordt video gestreamed over het bekabelde netwerk. Er is hier geen achtergrondtrafiek aanwezig.

De tweede test toont de situatie met achtergrondtrafiek. Een simulatieprogramma voor achtergrondverkeer [**?**] is hiervoor gebruikt. Er werden twee TCP streams en twee UDP streams gegenereerd. Voor iedere stroom zijn de aankomsten van de pakketten poisson verdeeld en hebben een frequentie van 100 pakketten per seconde. Zoals te lezen in 2.7 worden grote en kleine pakketten gegenereerd. De groottes hiervan liggen tussen 46 en 56 bytes -kleine- en 1200 en 1300 bytes -grote-. Om congestie te krijgen, is het Click-element BandwidthShaper toegevoegd. Dit zorgt ervoor dat de netwerkkaart trager pulls naar het vorige element kan sturen. Zo kan met minder dan 11Mbps[6] (snelheid draadloze verbinding) en 100Mbps (snelheid bedrade verbinding) congestie op het netwerk worden gecreëerd. Voor de eerste test werd de bedrade verbinding teruggeschroefd naar 1Mbps.

[6] 1 Mbps = 1000000 bits per seconde

3.4.5.1 Tijdige levering

Tijdige levering is verklaard in 2.7. Samengevat geeft dit type resultaat weer of pakketten tijdig aankomen. Dit is nodig om pakketten te decoderen en vervolgens videoframes weer te geven. Verloren pakketten zijn weergegeven met een omgekeerde piramide. De duur van een handover evenals verloren pakketten ten gevolge van congestie kunnen zo ook afgelezen worden op dit type grafiek. In 3.4.5.2 worden verloren frames meer in detail behandeld.

Figuur 3.11 toont het resultaat van de eerste test. Deze toont dat onder deze omstandigheden geen pakketten verloren gaan bij het streamen, m.a.w. er treedt geen congestie op. Dit is te verwachten aangezien er geen achtergrondverkeer aanwezig is.

De eigenlijke intentie van deze grafiek is te kijken of deadlines worden gehaald. Op deze grafiek is te zien dat de aankomsttijd altijd voor de afspeeltijd ligt. Met andere woorden, alle deadlines worden gehaald. Er kan worden opgemerkt dat de afspeeltijd niet mooi lineair verloopt. Er zitten "bultjes" in het verloop. Dit is te wijten aan de volgorde waarin de frames worden doorgestuurd. In 2.2.3.1, tabel 2.1 is de doorstuurvolgorde te zien. De frames worden dus verzonden in de zogenaamde codecvolgorde, besproken in 2.4.

Het tweede resultaat in figuur 3.12 toont dat het netwerk met achtergrondtrafiek wel degelijk wordt verzadigd. Om de afspeelijd te berekenen werd een constante vertraging van 5 seconden aan de aankomsttijd van het eerste pakket toegevoegd. De grafiek laat dus zien dat de lengte van de wachtrij oploopt en dus ook de vertraging, besproken in 3.4.5.3, vergroot. De frames komen dus te laat aan bij de ontvanger vanaf ongeveer pakketnummer 3000. Even later loopt de wachtrij vol en treedt congestie op.

3.4.5.2 Verloren frames

In 2.7 is uitgelegd dat verschillend types frames een verschillend belang hebben. Tabel 3.3 geeft het aantal verloren frames per type aan. Er wordt een onderscheid gemaakt tussen frames die verloren gaan ten gevolge van een handover en frames die verloren gaan ten gevolge van achtergrondtrafiek. Zo kunnen de effecten van een handover en van het achtergrondverkeer gescheiden worden bekeken.

De verliezen worden procentueel weergegeven. "Per type"geeft het aantal verloren frames gedeeld door het totaal aantal frames van dat type, uitgesloten die tijdens de handover, het procentuele verlies. Bij "Totaal"worden de verloren frames gedeeld door het totaal aantal frames, uitgezonderd die tijdens de handover.

Tabel 3.3 geeft dus weer dat geen frames verloren gaan bij de eerste test. Dit is verwacht, er is geen achtergrondverkeer. De duur van de disconnectie en het aantal pakketten dat hierbij is verloren, zijn hier ook terug te vinden.

Bij de tweede test zijn er wel pakketten verloren gegaan. Er trad dus verzadiging in het netwerk op. De resultaten tonen aan dat in het totaal minder I-frames verloren zijn dan andere. De verklaring hiervoor is dat er minder I-frames zijn in een videostroom. In de gebruikte trace zijn er per GOP namelijk 8 B-frames, 3 P-frames en één I-frame.

De verbinding werd niet verbroken, dus er zijn geen pakketten verloren ten gevolge van een

Tabel 3.3: Verloren frames implementatie 1

	Zonder achtergrondverkeer		**Met achtergrondverkeer**	
	Per type	Totaal	Per type	Totaal
I-frames(%)	0,00	0,00	12,24	1,76
P-frames(%)	0,00	0,00	9,78	2,53
B-frames(%)	0,00	0,00	5,51	3,29
	Duur(ms)	Totaal(♯)	Duur(ms)	Totaal(♯)
Disconnectie	0,00	0	0,00	0

Tabel 3.4: Kenmerken vertraging implementatie 1

	$\overline{x}$(ms)	s^2	Minimum(ms)	Maximum(ms)
Zonder achtergrondverkeer	180,86	0,05	180,35	181,62
Met achtergrondverkeer	3556,77	1654821,94	557,45	6139,59

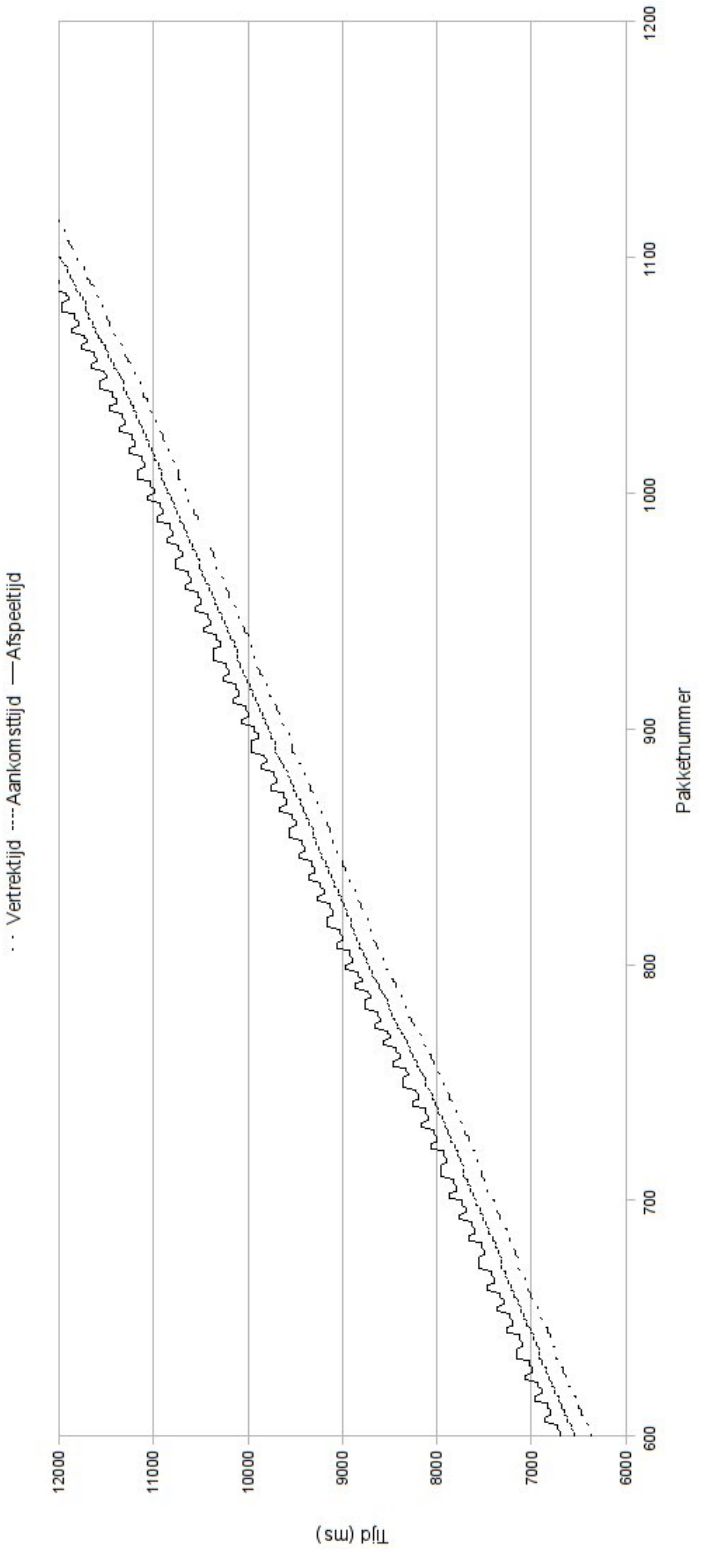
verbroken verbinding.

3.4.5.3 Vertraging

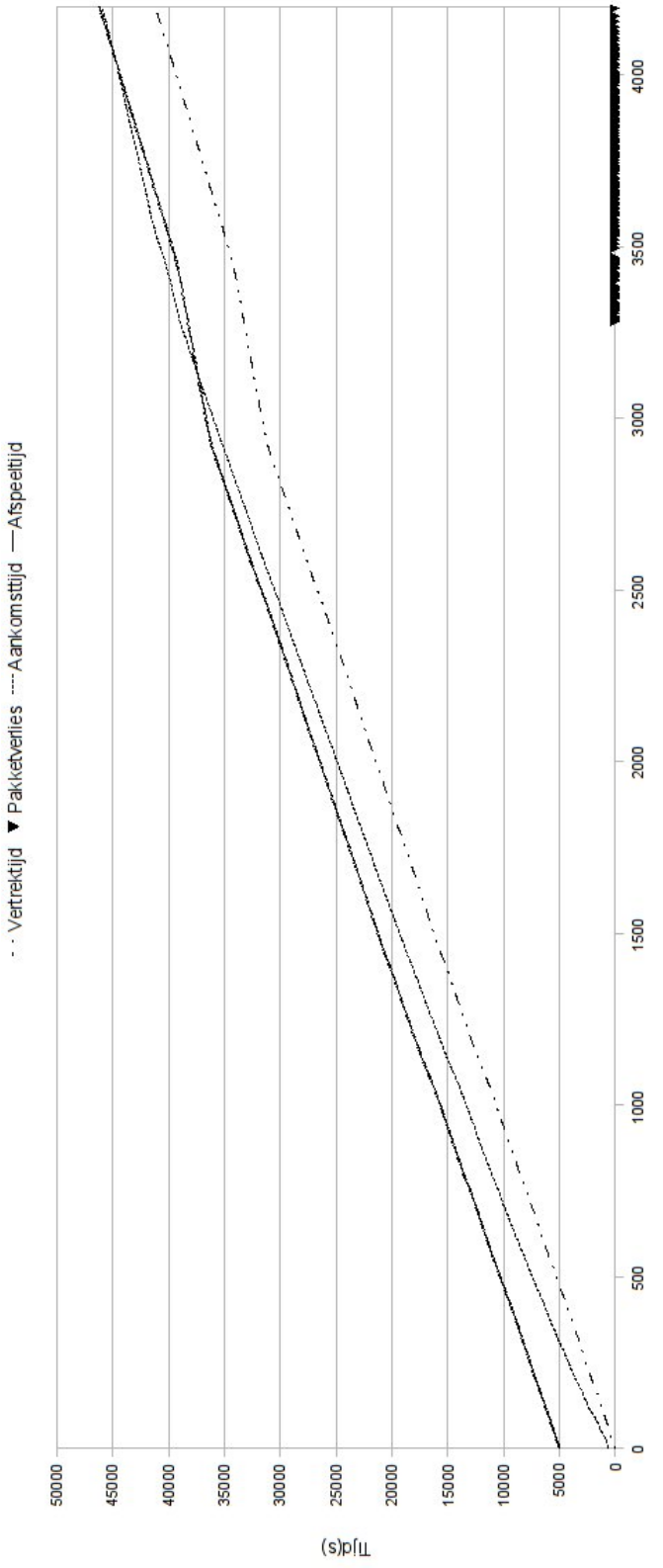
Deze sectie geeft de vertraging van de pakketten weer. De vertraging is belangrijk voor de buffergrootte. Dit werd reeds uitgelegd in 2.7. Figuur 3.13 geeft de vertraging zonder achtergrondverkeer weer en 3.14 de vertraging met achtergrondverkeer. De kenmerken van dit verkeer zijn samengevat in tabel 3.4.

Allereerst kan er worden opgemerkt dat er een duidelijk verschil is tussen de resultaten met en zonder achtergrondverkeer. De variantie in het resultaat met achtergrondverkeer veel groter. Omdat het netwerk met achtergrondverkeer in congestie geraakt, loopt de wachtrij zeer snel op. Dit is te zien in de vertraging in figuur 3.14. De vertraging blijft stijgen totdat er congestie optreedt. Op dat moment begint de router pakketten te vernietigen. De richtingscoëfficient van de stijgende curve wordt bepaald door het trafiek in de film zelf. De codec bepaalt hoeveel informatie er nodig is om de volgende frame te coderen. Als dit veel is, zullen er ook grotere pakketten op de lijn verschijnen en de vertraging sneller oplopen. Dit is dus rechtstreeks gerelateerd aan de karakteristieken van de film.

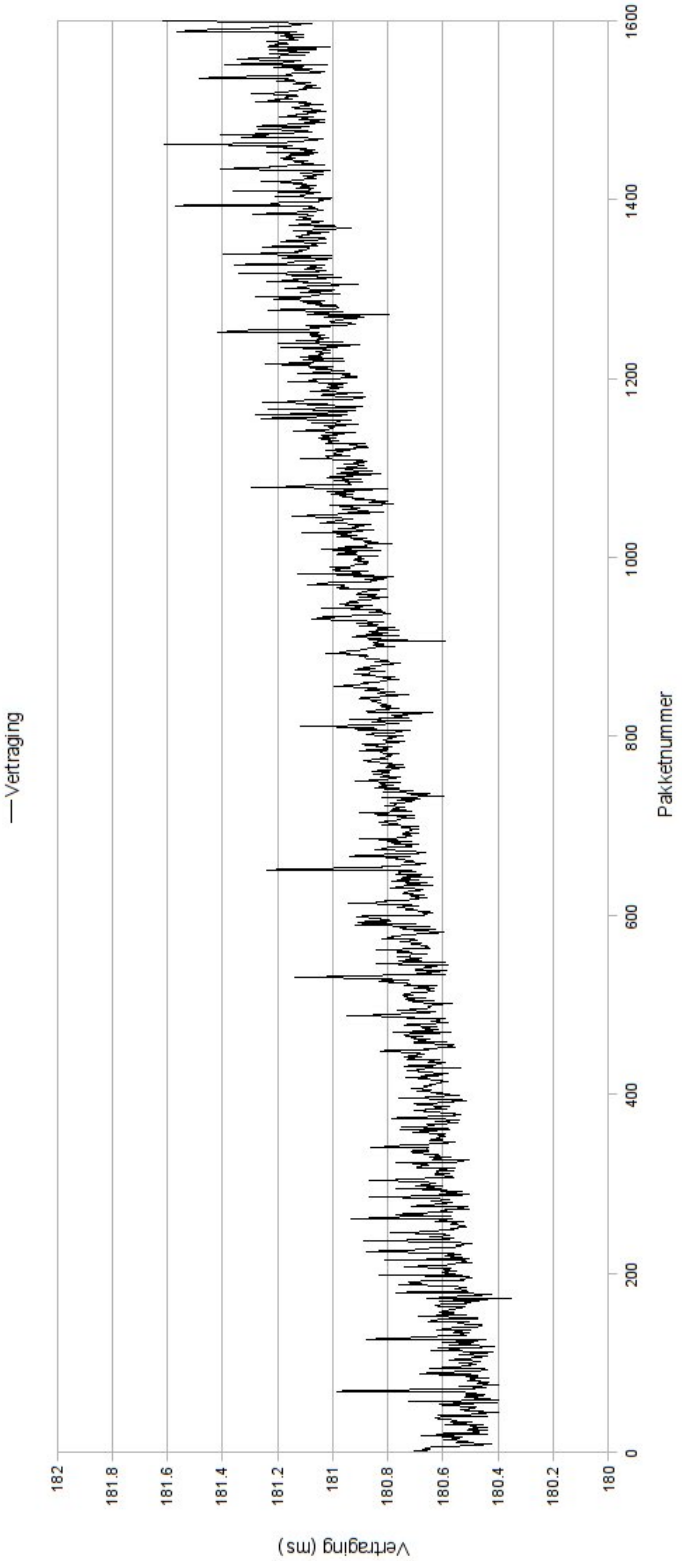
De maximum- en minimumvertraging liggen in het geval zonder achtergrondverkeer dan ook dicht bij elkaar. De wachtrij blijft hier op een constant niveau. Met achtergrondverkeer liggen deze waardes erg uit elkaar.



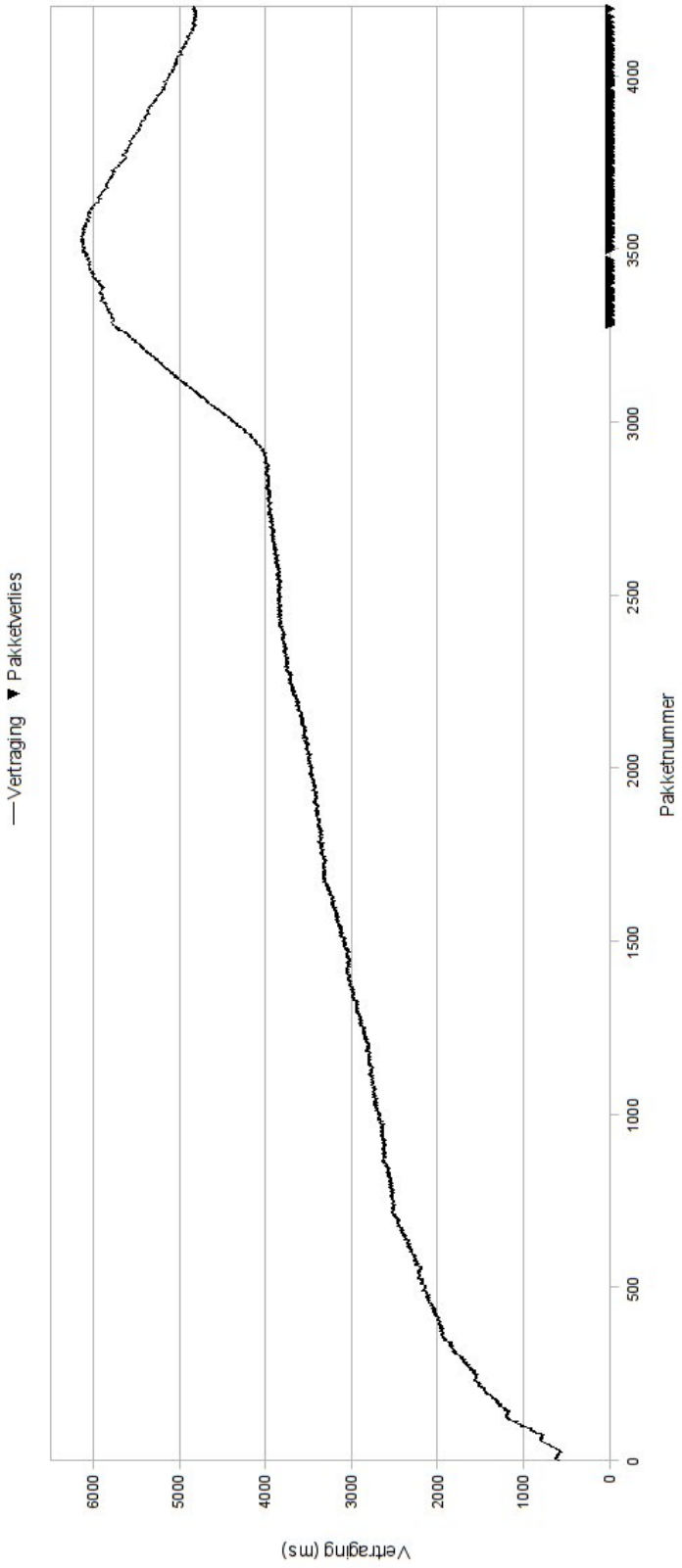
Figuur 3.11: Tijdige levering implementatie 1 zonder achtergrondtrafiek



Figuur 3.12: Tijdige levering implementatie 1 met achtergrondtrafiek



Figuur 3.13: Pakketvertraging implementatie 1 zonder achtergrondverkeer.Kenmerken: $\overline{x} = 180,86$ms; $s^2 = 0,05$; Min= $180,35$ms; Max= $181,62$ms



Figuur 3.14: Pakketvertraging implementatie 1 met achtergrondverkeer. Kenmerken: $\overline{x} = 3556,77$ms; $s^2 = 1654821,94$; Min=$557,45$ms; Max=$6139,59$ms

3.5 Besluit

In dit hoofdstuk 3 werd uitgelegd welke software en hardware gebruikt is voor de opstellingen.

Er zijn ook enkele voorbeelden aangehaald hoe routers geprogrammeerd worden in Click.

In 3.4 bespreekt de basis van het verdere onderzoek. De implementatie 1 router stuurt pakketten die binnenkomen op interface eth0 door naar interface eth2, de bedrade verbinding. Voor de implementaties uit de volgende hoofdstukken wordt steeds vertrokken van implementatie 1. De ARPQuerier en de gemeenschappelijke elementen worden verder niet nog eens verklaard.

In 3.4.5 wordt de manier verklaard waarop de resultaten worden gegenereerd en weergegeven.

Hoofdstuk 4

Router met handover mechanisme

Dit hoofdstuk vertrekt van de opstelling van de gemeenschappelijke node. Dit is implementatie 1 waarmee hoofdstuk 3 werd afgerond (3.4). Stap voor stap worden hieraan wijzigingen aangebracht om een goede handover tussen de bedrade en draadloze verbinding, te bekomen. In dit hoofdstuk gebeurt geen aanpassing aan de server.

Om dit te realiseren wordt eerst de draadloze verbinding in de opstelling ondergebracht. Dit is terug te vinden in 4.1.

Wanneer de twee routes[1] naar de eindbestemming zijn gerealiseerd, wordt een automatisch handover mechanisme ontworpen. Het eerste mechanisme is een script dat het PING progamma gebruikt. Hiervoor moeten de pakketten, zowel op de mobiele node als op de gemeenschappelijke node, ook naar Linux gaan, dit omdat het script in Linux beslist of er een handover gebeurt. Het tweede mechanisme controleert een bestand in Linux, de zogenaamde carrier. Dit gedeelte van de handover wordt beschreven in 4.2 en verder uitgebreid in 4.4.

Bij het versturen van video bleven in eerdere opstellingen pakketten in de verkeerde wachtrij zitten. De oplossing voor dit probleem werd besproken in 4.3.

In de laatste implementatie 4.4 wordt ervoor gezorgd dat ook de signaalsterkte van de draadloze verbinding wordt bekeken. Hiervoor controleert enkel de mobiele node of de verbindingen goed zijn. Deze node meldt dit aan de gemeenschappelijke node en server. Hier wordt ook het handovermechanisme geautomatiseerd.

Tot slot worden de bedenkingen bij de implementaties samengevat in het besluit 4.5.

4.1 Implementatie 2: Twee verbindingen

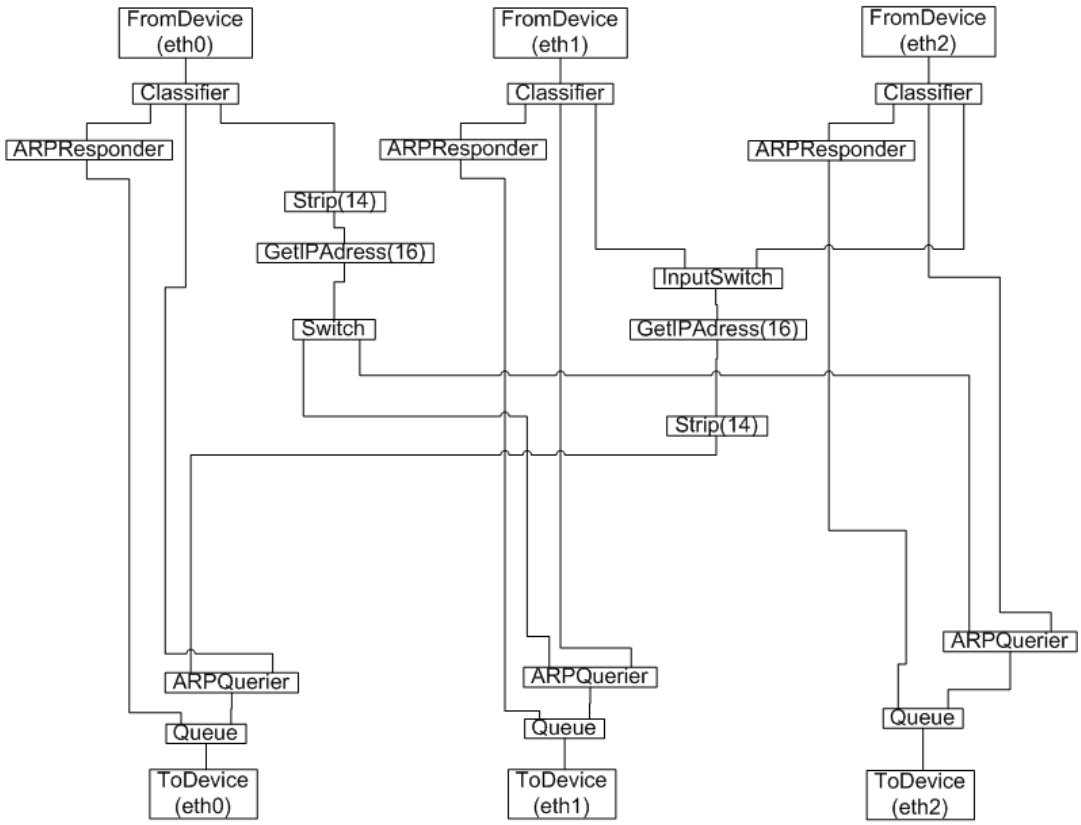
In deze implementatie wordt voor twee mogelijk paden naar de mobiele node gezorgd. Met behulp van Click wordt zachte handover tussen de bedrade en draadloze link verwezenlijkt. Het doel is ervoor te zorgen dat de bron niet op de hoogte moet worden gebracht van een wijziging in het transportmedium.

[1] De twee routes zijn: het pad naar de mobiele node via de bedrade en de draadloze link

Het IP van de draadloze en bedrade interface op de Client is immers niet hetzelfde. Pakketten die dan na een wijziging reeds onderweg waren naar de “oude” interface met een ander IP adres, komen dan niet meer aan op hun bestemming. Ook wordt zo een extra aanpassing, nl. een IP adres verandering, in de bron vermeden. Er werd dus getracht één gemeenschappelijk IP adres te geven aan beide interfaces. Er kan worden opgemerkt dat deze oplossing geen extra functionaliteit in de router vereist. Hiervoor wordt zowel de gemeenschappelijke node als de mobiele node aangepast. De aanpassing in de gemeenschappelijke node is uiteraard enkel voor het opzetten van een extra verbinding en heeft niets te maken met het statische IP adres.

4.1.1 Router (Gemeenschappelijke node)

De gemeenschappelijke node moet nu niet alleen over de bedrade verbinding streamen, maar moet ook de mogelijkheid hebben om over de draadloze verbinding te streamen. Hiervoor wordt de Click-configuratie in de gemeenschappelijke node aangepast. Deze aanpassing worden getoond in figuur 4.1. De Click-configuratie is terug te vinden in A.3.



Figuur 4.1: Configuratie met twee paden. Één over de bedrade verbinding en één over de draadloze.

In deze opstelling is netwerkinterface eth1 toegevoegd. Dit is de netwerkkaart die verbonden is met het accespunt. Dit accespunt brengt de draadloze verbinding met de mobiele node tot stand. Twee elementen werden toegevoegd: een Switch en een InputSwitch. De Switch is een

standaardelement in Click en zorgt ervoor dat in de configuratietekst kan worden meegeven op welke uitgang de switch zijn pakketten stuurt. De switch stuurt de pakketten dus alleen naar de uitgang die in de configuratietekst wordt meegegeven. Op de andere uitgang komen geen pakketten. Het is alsof aan de andere uitgang geen verbinding aanwezig is. De InputSwitch doet hetzelfde, maar dan voor de ingangen. Dit element is geschreven door Dagang Li[**?**]. De code van dit element is terug te vinden in A.14.

Afhankelijk van welke ingang en welke uitgang gekozen wordt, is er een rechtstreekse verbinding van eth0 naar eth1 en omgekeerd of een verbinding van eth0 naar eth2 en omgekeerd. Bij het instellen van de switchen op hun tweede in- en uitgang verkrijgen we het scenario getoond op figuur 3.10.

Het eerste doel, pakketten naar de draadloze- en bedrade verbinding sturen, is dus opgelost.

4.1.2 Client (Mobiele node)

Ook op de Client wordt een handover doorgevoerd. De Client luistert naar de bedrade of de draadloze verbinding. De pakketten moeten nu ook naar Linux omdat ze anders niet gezien worden door de streaming-software. De configuratie van de mobiele node is te zien op figuur 4.2. De Click-configuratie is terug te vinden in A.4.

Op deze configuratie zijn de twee netwerkkaarten van de mobiele node te onderscheiden. In Linux dragen zij de naam eth1 en ath1. Eth1 is de bedrade verbinding en Ath1 is de draadloze verbinding. De InputSwitch kiest het soort verbinding naar Linux, nl. draadloze of bedrade. Ook zijn er twee verschillende ARPResponse elementen, voor iedere netwerkkaart één. Dit omdat iedere netwerkkaart een ander hardwareadres[2] heeft. De ARPResponse pakketten[3] worden naar de juiste interface gestuurd. Dit wordt geregeld door de twee Switchen onder de Classifier.

De IP pakketten, dus niet de ARP pakketten, worden naar ToHost gezonden. ToHost is een element dat ervoor zorgt dat de IP pakketten, dus ook deze van de stream, naar Linux worden gezonden. Het ToHost element zorgt er ook voor dat naar de mobiele node kan worden verwezen met één IP adres. Hiervoor was het nodig om, aan de hand van een IP adres, te verwijzen naar ofwel de bedrade, ofwel de draadloze verbinding. Nu worden deze beide samengevat in één imaginair apparaat.

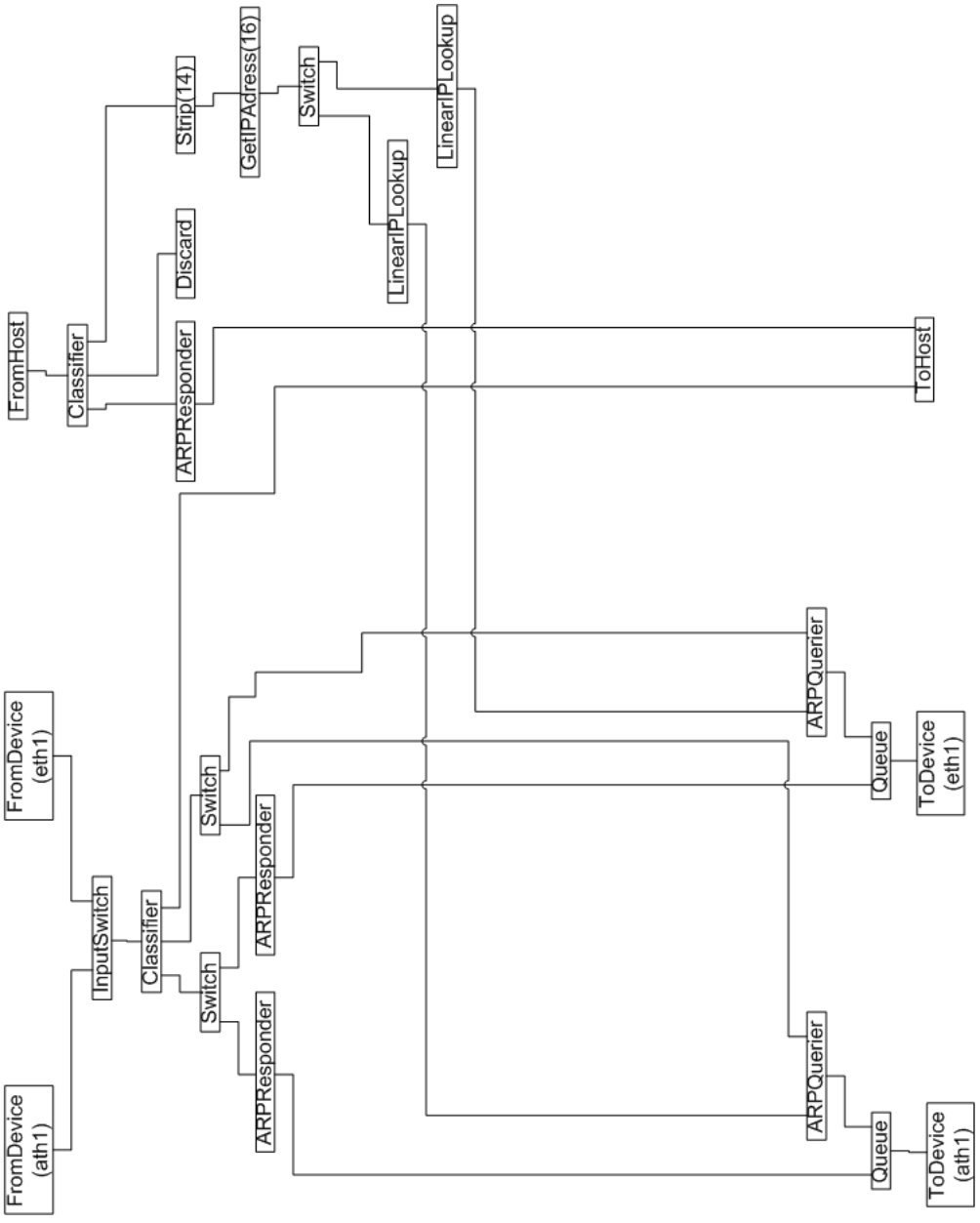
Linux stuurt zelf ook ARP Queries uit. Deze worden opgevangen met een Classifier en een ARPResponder.

Om een bidirectionele verbinding te voorzien, moet Linux pakketen terug naar de juiste router-interface, nl. de draadloze of de bedrade, zenden. Dit vereist extra informatie. Hiervoor staan de LineairIPLookup elementen in.

Dit element voorziet het pakket van het gewenste pad, de zogenaamde volgende hop. In de twee gevallen is dat dus 192.168.0.1 voor de bedrade verbinding en 192.168.1.100 voor de draadloze verbinding. Dit zorgt ervoor dat de netwerkkaarten op de mobiele node vragen naar

[2] Ook naar gerefereerd als MAC adres

[3] Deze bevatten het antwoord op de vraag naar het MAC adres gelinkt aan een IP adres, in de vorm “Het IP adres heeft MAC”. Dit werd beschreven in 3.1.10.



Figuur 4.2: Configuratie met twee routes. Één over de bedrade verbinding en één over de draadloze. De pakketten worden ook naar Linux gestuurd.

de juiste MAC adressen. De server zit op een ander subnetwerk en is dus niet rechtstreeks bereikbaar. Als de vraag "Wie weet het MAC adres van 1-2.168.6.1?" wordt uitgestuurd op een van de twee verbindingen tussen de mobiele node en de gemeenschappelijke node, kan de gemeenschappelijke node niet antwoorden. Dit omdat 192.168.6.1 niet rechtstreeks op een van deze twee netwerkkaarten is aangesloten. Als eerst de volgende hop in de annotatie wordt gezet, kan deze ARP aanvragen wel juist verlopen.

4.1.3 Resultaten

Op dit moment is het mogelijk een manuele handover te verwezelijken. Praktisch gebeurt dit door gelijktijdig op de gemeenschappelijke node en op de mobiele node een hotswap van de configuratiebestanden uit te voeren. De beide commando's om naar de bedrade verbinding te wisselen zijn:

cp gemeenschappelijkenodewire.Click /Click/hotconfig
cp mobielenodewire.Click /Click/hotconfig

Om over te schakelen naar de draadloze verbinding worden de volgende commando's uitgevoerd:

cp gemeenschappelijkenodewireless.Click /Click/hotconfig
cp mobielenodewireless.Click /Click/hotconfig

Het verschil tussen de twee configuraties is dus het getal dat tussen haken achter de verschillende Switchen staat. Een 0 staat voor de draadloze verbinding en een 1 staat voor de bedrade verbinding.

Gelijktijdig uitvoeren van de commando's om te wisselen heeft het verlies van enkele pakketten tot gevolg. Dit zijn pakketten die in de buffer zijn opgeslagen op het moment van de wissel. Zonder achtergrondverkeer zijn dit er zeer weinig tot geen. Maar hoe meer achtergrondverkeer, hoe meer pakketten in de wachtrijen blijven zitten en dus verloren gaan. Dit probleem wordt opgelost in 4.3. Eerst wordt de handover geautomatiseerd zodanig dat als de verbinding tussen de mobiele node en de gemeenschappelijke node wordt verbroken, de handover naar de draadloze verbinding automatisch en zo snel mogelijk gebeurt. Deze verbetering wordt in de volgende sectie 4.2 besproken.

4.2 Implementatie 3: Automatische handover mobiele node

Informatie over een verbroken verbinding wordt niet verzorgd door de transport- of netwerklaag. Daarom wordt het netwerk voorzien van een systeem dat deze informatie voorziet aan de router en de bestemmeling. In eerste instantie wordt dit zogenaamde Cross-Layer informatiesysteem op een gedistribueerde manier geïmplementeerd. Met andere woorden, de verschillende netwerk-elementen houden ieder voor zich de verbinding in het oog. Onder gedistribueerd wordt hier begrepen dat ieder element voor zich een autonome beslissing maakt, zonder daarbij advies of informatie van een ander element te krijgen.

De eerste methode maakt gebruik van PING. Deze methode controleert de volledige verbinding van begin tot einde. Bij het wegvallen van een verbinding (zij het een tussenliggen of directe verbinding) in het netwerk of het verloren gaan van het een PING-pakket ten gevolge van

congestie, voorziet deze methode in een handover naar de andere verbinding. De gemeenschappelijke node wordt aangepast zodat het PING-pakket ook door Linux wordt ontvangen zodat daar de beslissing voor een handover kan worden gemaakt.

De tweede belangrijke methode maakt gebruik van carrier-informatie opgeslagen in een Linux-bestand. Deze methode controleert enkel het bestaan van de directe verbinding. In dit geval het aanwezig zijn van een kabelverbinding of niet. Het wegvallen van een tussenliggende verbinding wordt hier dus niet gedetecteerd. Ook congestie wordt niet opgemerkt met deze techniek.

Om in een overschakeling te voorzien, wordt geen gebruik van hotswap-configuraties gemaakt, maar van de handlers die de switchen bezitten.

4.2.1 Router (gemeenschappelijke node)

Aan de gemeenschappelijke node zijn wel veranderingen aangebracht. De pakketten die over de bedrade verbinding gingen, worden nu ook naar Linux gestuurd. Op deze manier kan in Linux een script geschreven worden dat detecteert of een PING-pakket aankomt bij zijn bestemming of niet. De veranderingen aan de opstelling in 4.1.1 zijn in figuur 4.3 weergegeven. De Click-configuratie is terug te vinden in A.5.

Enkel de pakketten op interface eth2 gaan naar Linux. Dit omdat op dit moment enkel een handover gebeurt wanneer de kabelverbinding verbroken wordt. In 4.4 wordt ook de signaalsterkte van de draadloze verbinding gemeten. Er wordt dan een handover uitgevoerd wanneer de signaalsterkte te laag is.

De pakketten van interface eth2 worden nu ook naar het ToHost element gezonden. Het ToHost element zorgt ervoor dat de pakketten ook naar Linux worden gestuurd voor afhandeling. De verbinding moest hiervoor bidirectioneel zijn omdat een PING een aanvraag stuurt en daarop een antwoord verwacht.

De pakketten die van interface eth2 komen, worden niet allemaal gekopieerd naar Linux. De toevoeging van een IPClassifier zorgt ervoor dat enkel ICMP pakketten naar Linux gaan. Anders zouden alle videopakketten ook naar Linux worden gestuurd.

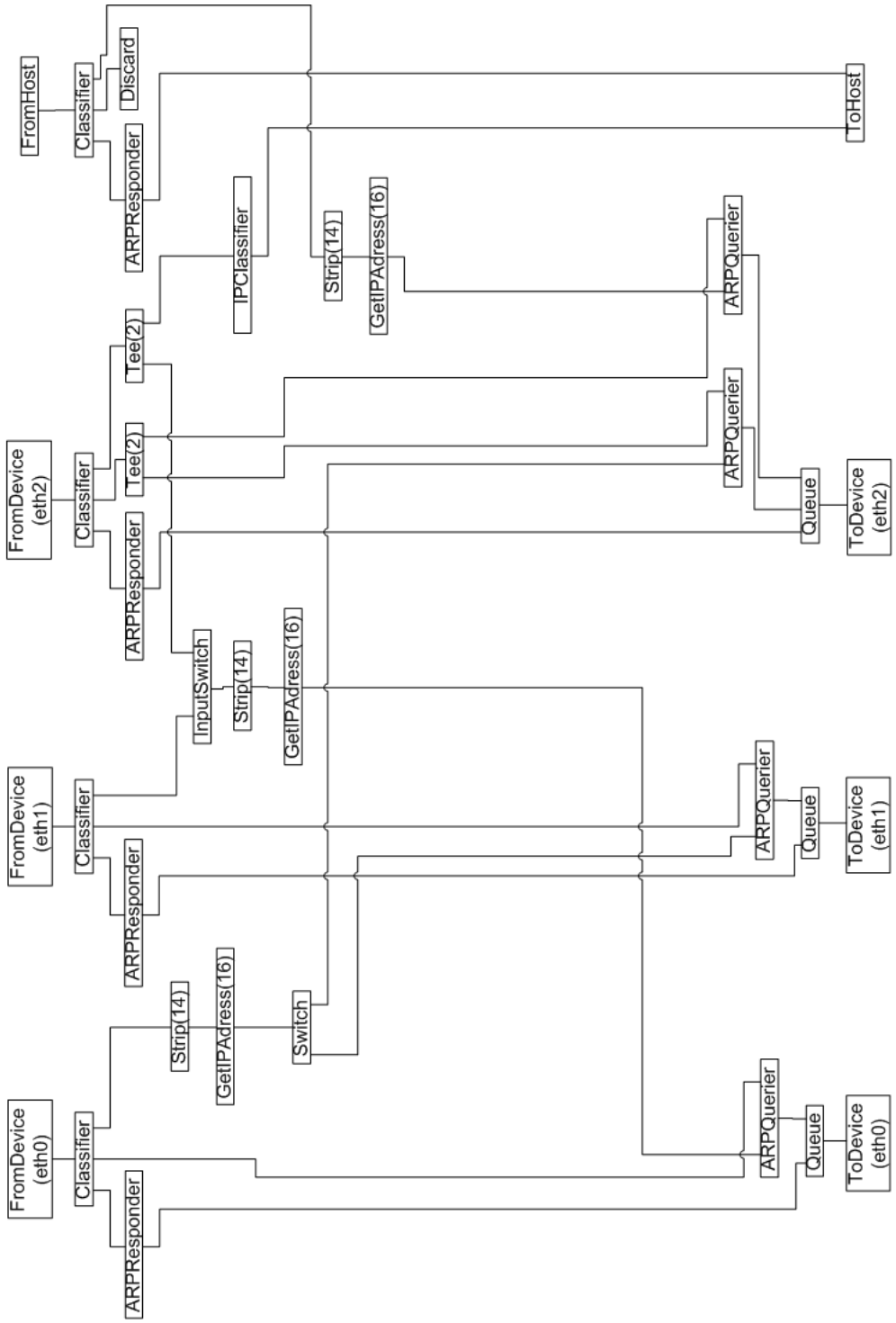
4.2.2 Client (Mobiele node)

De mobiele node blijft hetzelfde als in 4.1.2.

4.2.3 Handover

De automatische handover werd in de eerste fase ontworpen met behulp van een script dat een PING-pakket stuurt en kijkt of dit aankomt of niet. Ook maakt dit script gebruik van handlers i.p.v. hotswap. Het script, dat zorgt voor de automatische handover, draait op de gemeenschappelijke node en op de mobiele node. Later in 4.4 zal enkel de mobiele node een detectie doen. Hij zal de gemeenschappelijke node op de hoogte houden van eventuele veranderingen.

Het script is hieronder te vinden.



Figuur 4.3: Configuratie met twee routes. Één over de bedrade verbinding en één over de draadloze. De pakketten van de bedrade verbinding worden ook naar Linux gestuurd.

```
1  #!/bin/bash
2  i=0
3  while true
4  do
5  sleep 1
6  pinger=‘ping 192.168.3.1 -t 1 -c 1 | grep connected | cut -c 0-9‘
7  echo $pinger
8  if [ "$pinger" != "connected" ]
9  then
10 if [ $i -ne 1 ]
11 then
12 echo  ’install wireless’
13 echo 1 > /proc/Click/verdeler0/switch
14 echo 1 > /proc/Click/combinator0/inputswitch
15 i=1
16 fi
17 else
18 if [ $i -ne 0 ]
19 then
20 echo ’install wire’
21 echo 0 > /proc/Click/verdeler0/switch
22 echo 0 > /proc/Click/combinator0/inputswitch
23 i=0
24 fi
25 fi
26 done
```

Dit script stuurt elke seconde één PING-pakket naar de mobiele of de gemeenschappelijke node. Er wordt dan gekeken of het pakket aankomt of niet. Bij het niet aankomen van het pakket, wordt de verbinding omgezet zodat ze over de draadloze verbinding gaat. Bij het herstellen van de bedrade verbinding, gebeurt geen wissel naar de bedrade verbinding. Dit omdat geen koppeling voorzien is langs de draadloze verbinding (eth2). De omwisseling gebeurt nu niet meer met hotswap, maar er wordt gewerkt met handlers. Een 1 of een 0 wordt gestuurd naar de handlers van beide switchen. Dit is te zien op lijn 13, 14, 21 en 22 van het programma. Op lijn 6 van het programma, wordt de uitvoer van de PING-opdracht bekeken. Een besluit uit het resultaat wordt getrokken op lijn 8.

Het PING-pakket zenden duurt te lang. Zo gaan te veel pakketten verloren voordat de omschakeling gebeurt. Een andere methode is nodig.

De tweede methode gebruikt ifplugd. Dit programma geeft zeer snelle resultaten bij het verbreken van de kabelverbinding. Daarom is de werking van het programma nader bekeken. Hieruit werd ontdekt dat het progamma kijkt naar de carrier van de netwerkkaarten. Deze carrier is te vinden in
/sys/class/net/eth1/carrier.
Als er een 1 in dit bestand zit, is de verbinding aanwezig. Bij 0 is er geen verbinding. Aan de hand van deze informatie werd het volgende script ontworpen voor de automatische handover:

```
1  wire=1
2  while true
3  do
4  onoff=‘cat /sys/class/net/eth2/carrier‘
```

```
5  if [ "$onoff" == "0" ]
6  then
7  if [ "$wire" == "1" ]
8  then
9  echo "wire-wireless"
10 echo 1 > /proc/Click/verdeler0/switch
11 echo 1 > /proc/Click/combinator0/inputswitch
12 wire=0
13 fi
14 fi
15 if [ "$onoff" == "1" ]
16 then
17 if [ "$wire" == "0" ]
18 then
19 echo "wireless-wire"
20 echo 0 > /proc/Click/verdeler0/switch
21 echo 0 > /proc/Click/combinator0/inputswitch
22 wire=1
23 fi
24 fi
25 done
```

Dit programma bevat dezelfde elementen als het vorige, maar hier gebeurt dit op basis van de carrier. Een koppeling langs de draadloze verbinding is dus niet nodig. De toestand van de carrier wordt uitgelezen op lijn 4 van het programma. In lijnen 5 en 15 gebeurt de controle op de waarde. Lijnen 10, 11, 20 en 21 zetten de juiste waarde in de handlers van de switchen, nl. 1 voor de draadloze verbinding en 0 voor de bedrade.

4.2.4 Resultaten

In de eerste twee tests wordt geen achtergrondverkeer op het netwerk geplaatst. Deze tests geven immers enkel de verbetering in de snelheid van de handover weer. Er zijn nog geen aanpassingen gebeurd om congestie tegen te gaan.

De eerste test gebruikt de opstelling uit implementatie 3 met het PING-script. Er gebeurt hier een overgang van bedrade naar draadloze verbinding. De omgekeerde richting kan niet omdat er geen verbinding voorzien is van de draadloze link naar Linux op de gemeenschappelijke node[4]. Er wordt dus geen antwoord gegeven op de PING.

In de tweede test wordt gebruik gemaakt van het carrier-script. Hier gebeurt een overgang van bedrade naar draadloze verbinding, maar ook omgekeerd.

Voor het genereren van deze resultaten werd geen gebruik gemaakt van de BandwidthShaper, de volledige 100Mbps voor de bedrade en 11Mbps voor de draadloze werd benut.

[4]Hiervoor moet het ToHost-element in figuur 4.3 ook verbonden zijn met eth1.

Tabel 4.1: Verloren frames implementatie 3

	Zonder verkeer PING		**Zonder verkeer carrier**	
	Duur(ms)	Totaal(♯)	Duur(ms)	Totaal(♯)
Handover	10199,99	932	773,22+226,52	69+24

4.2.4.1 Tijdsduur handover

De twee verschillende methodes voor het detecteren van een wegvallende verbinding worden hier naast elkaar gelegd. Tabel 4.1 geeft het aantal verloren pakketten, samen met de duur van de handover, weer. Als de handover korter is, gaan minder pakketten verloren. De duur van de handover is hier namelijk gedefinieerd als "het aantal pakketten dat verloren gaat bij een omschakeling". "Het verschil tussen de verstuurtijden van het laatst aangekomen pakket bij de bestemmeling en het eerste nieuwe pakket over de andere verbinding" wordt gedefinieerd als de tijd nodig voor een handover.

Het PING-script laat een handover zien van de bedrade naar de draadloze verbinding. Deze overgang duurt 10199,99ms en hierbij gaan 932 pakketten verloren. Dit is veel meer dan het carrier-script. Hier duurt een handover van bedrade naar draadloze verbinding 773,22ms en gaan slechts 69 pakketten verloren. Het carrier-script detecteert het wegvallen van de bedrade verbinding dus met een factor 10 sneller dan het PING-script.

Andersom, overschakelen naar de bekabelde verbinding, gebeurt met minder pakketverlies. De overgang naar een kabelverbinding gebeurt in 226,52ms. Hierbij gaan 24 pakketten verloren. De tragere overdracht van een draadloze verbinding speelt hierin een grote rol. Het aantal "in-flight-pakketten[5] in een draadloze verbinding is lager en er gaan dus ook minder verloren.

Het is dus duidelijk dat het carrier-script een veel betere performantie heeft. Zelfs de duur van een handover van bekabelde naar draadloze verbinding samengeteld bij de duur van een handover van draadloze naar bekabelde verbinding is korter dan één enkele handover van bekabelde naar draadloze verbinding met het PING-script.

4.2.4.2 Vertraging

De vertraging van een draadloos netwerk is groter dan die van een bekabeld netwerk. Dit komt omdat lucht en zijn omgeving een trager medium is dan koper. Lucht is ook een minder voorspelbaar medium dan koper. Een verandering in de omgeving kan de transferkarakteristieken van de antenne en van het medium (lucht) sterk beïnvloeden. Dit is duidelijk op te merken in figuren 4.4 en 4.5.

Figuur 4.4 geeft de vertraging bij handover met het PING-script weer. De handover van bedrade verbinding naar de draadloze heeft ongeveer tussen pakket 520 tot 1900 plaats. De kenmerken

[5] "In-flight-pakketten zijn pakketten die reeds op de verbinding zijn gezet, maar niet aankomen vooraleer de ontvangstzijde de verbinding heeft afgesloten.

Tabel 4.2: Kenmerken vertraging implementatie 3.

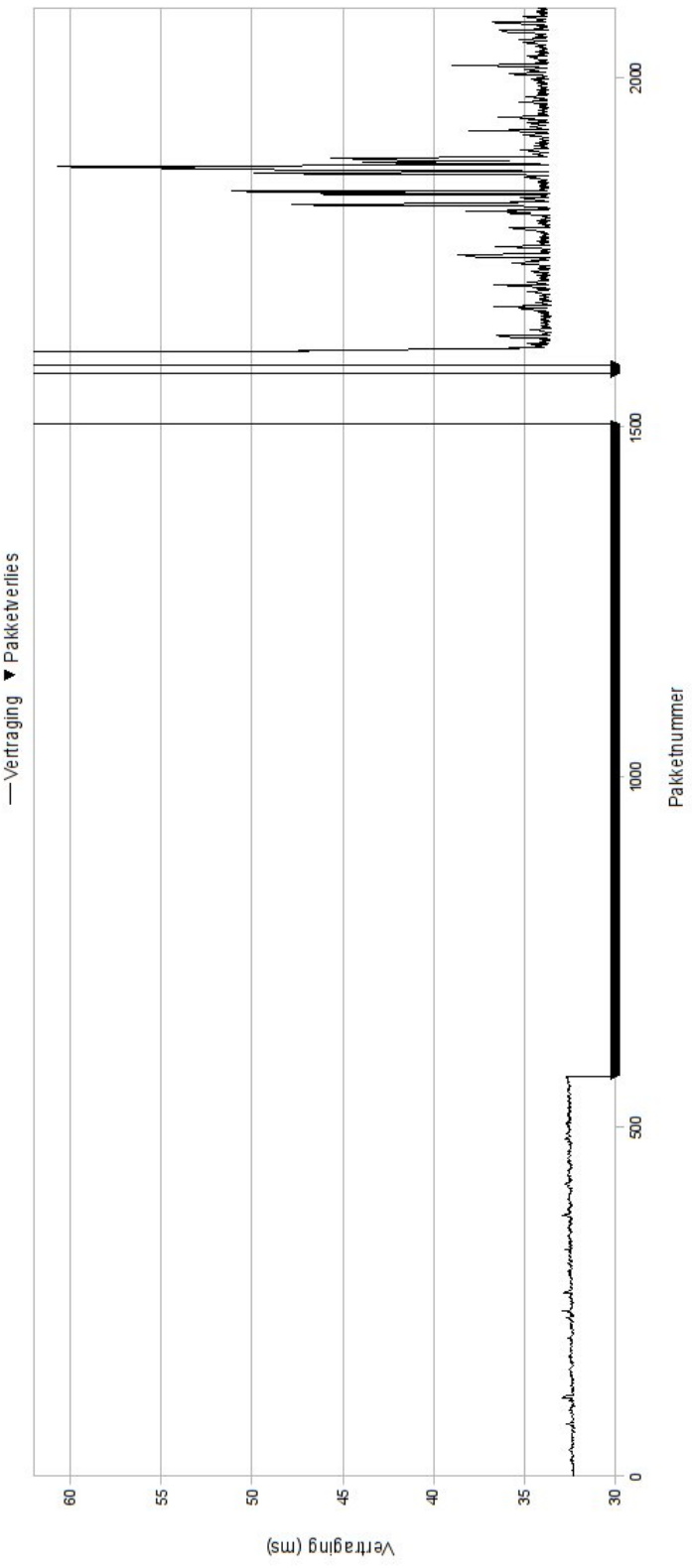
	$\overline{x}$ (ms)	s^2	Minimum (ms)	Maximum(ms)
PING	74,81	27948,54	32,22	1044,38
Carrier	58,67	5847,72	50,08	868,09

zijn terug te vinden in tabel 4.2. De hoge variantie is te wijten aan de uitschieters bij het begin van de overschakeling naar draadloze verbinding. Deze uitschieters zijn een gevolg van een probleem met de synchronisatie van de PING-scripten. Zoals reeds uitgelegd, loopt het PING-script op zowel de gemeenschappelijke node als de Client. Omdat het PING-script een grote vertraging heeft voor de detectie, kan het verschil tussen de detectiemomenten groot zijn.

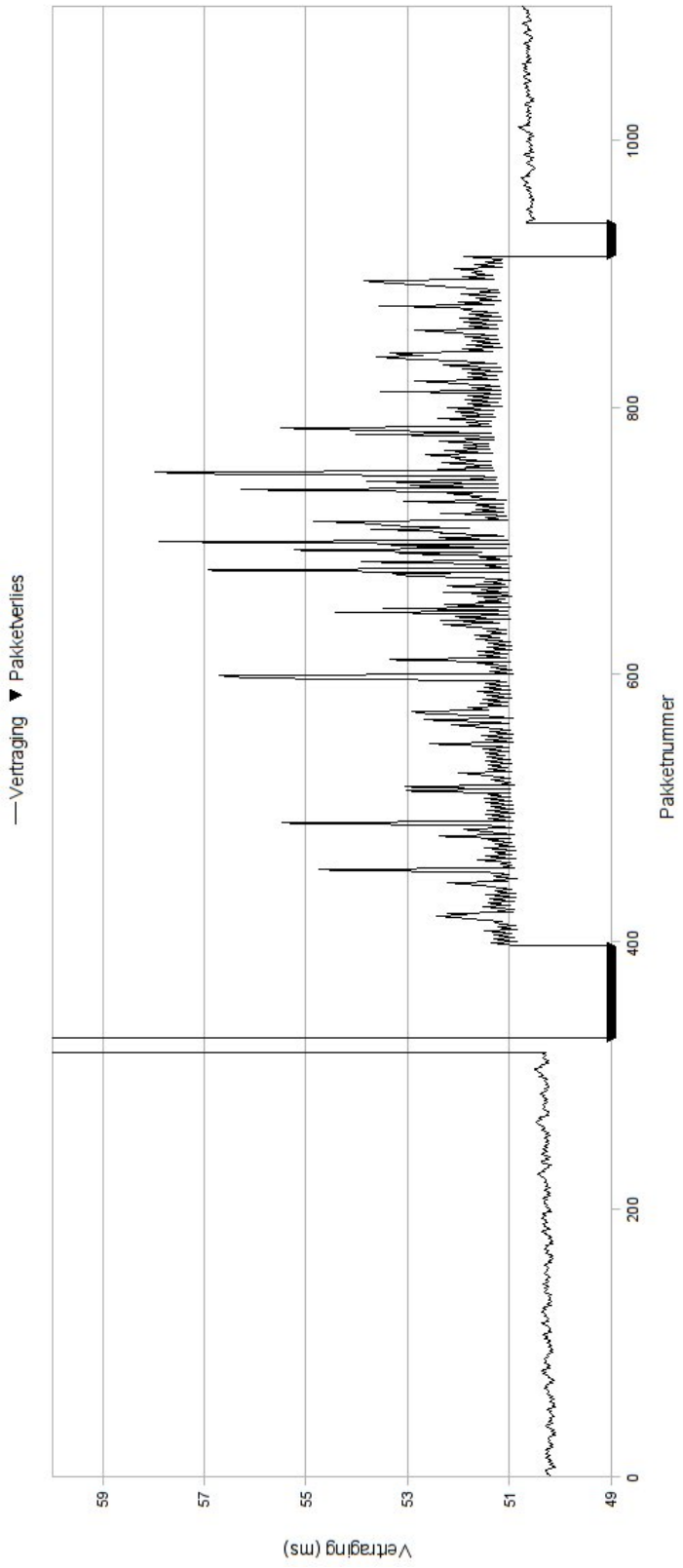
In het testgeval ontdekt de gemeenschappelijke node het verbreken van de kabelverbinding eerder dan de Client. In de Client is dus nog geen omschakeling gebeurd, terwijl de pakketten in de wachtrij richting draadloze link worden gezet op de gemeenschappelijke node. Ze worden er nog niet uitgehaald. De pakketten zijn dus vertraagd totdat de interface met de draadloze link pakketten begint te versturen. In dit geval is het verschil zelfs zo groot dat de uitgaande wachtrij volloopt, te zien aan het dal net achter het de piek. Daarna wordt de wachtrij leeggemaakt en daalt de vertraging totdat het niveau van de wachtrij het minimum - waarschijnlijk één à twee pakketten - bereikt.

Verder is er een duidelijk verschil in variatie op de vertraging bij bedrade en draadloze verbinding. De piekvertraging is bijna twee maal zo lang als de gemiddelde. Dit vereist een grotere buffer met als gevolg vertraging - zij het enkel in het begin - bij het afspelen van een stroom, zoals vermeld in 2.7.

De kenmerken voor het carrier-script, te zien in tabel 4.2 en op figuur 4.5, zijn gelijkaardig aan die van het PING-script. De uitschieter rond pakketnummer 300 wordt veroorzaakt door pakketten die in de verkeerde buffer zitten, vermeld in 4.1. Als een pakket in de buffer voor de bedrade verbinding zit en deze verbinding wordt afgesloten, blijft deze zitten totdat de bedrade verbinding weer wordt hersteld. Dit geeft een piek in de vertraging op de grafiek.



Figuur 4.4: Pakketvertraging implementatie 3 PING zonder achtergrondverkeer.Kenmerken: $\overline{x} = 74,81$ms; $s^2 = 27948,54$; $Min = 32,22$ms; $Max = 1044,38$ms



Figuur 4.5: Pakketvertraging implementatie 3 carrier zonder achtergrondverkeer. Kenmerken: $\overline{x} = 58,67$ms; $s^2 = 5847,72$; $Min = 50,08$ms; $Max = 868,09$ms

4.3 Implementatie 4: Omleiding buffer

Het probleem met pakketten die al onderweg zijn naar de bestemming werd al ten dele opgelost in 4.1 met een statisch IP adres. In dit hoofdstuk wordt een opslag- en doorsturingssysteem besproken, de zogenaamde omleidingsbuffer waarmee het probleem van pakketten die achterblijven in de verkeerde wachtrij wordt aangepakt.

Dit probleem werd aangehaald in 4.1 en is kort samen te vatten als volgt. Als een handover van bedrade naar draadloze verbinding gebeurt, zitten er nog pakketten in de buffer van de bedrade verbinding die eigenlijk via de draadloze verbinding moeten uitgestuurd worden. In de vorige opstellingen gingen deze pakketten steeds verloren.

4.3.1 Router (gemeenschappelijke node)

In de gemeenschappelijke node is er tussen interface eth1 en interface eth2 een kruislingse koppeling voorzien om de pakketten tijdens de handover naar de juiste wachtrij te verplaatsen. De totale opstelling is te zien in 4.6. De veranderingen zijn in detail te zien op figuur 4.7. Zo kan de werking van dit gedeelte beter worden toegelicht. De Click-configuratie is terug te vinden in A.6.

Bij het versturen over de draadloze verbinding, komen de pakketten van de video in de wachtrij van interface eth2. Hier stapelen pakketten zich op totdat ze door netwerkkaart eth2 eruit worden gehaald. Een omschakeling heeft tot gevolg dat de volgende pakketten van de video in de wachtrij van netwerkkaart eth1 terechtkomen. De pakketten die zich in de wachtrij van eth2 bevinden, zullen niet bij de mobiele node aankomen en dus verloren gaan. Dit is schadelijk voor de videokwaliteit. Er wordt verwacht dat er meer pakketten verloren gaan bij hogere netwerkbelasting.

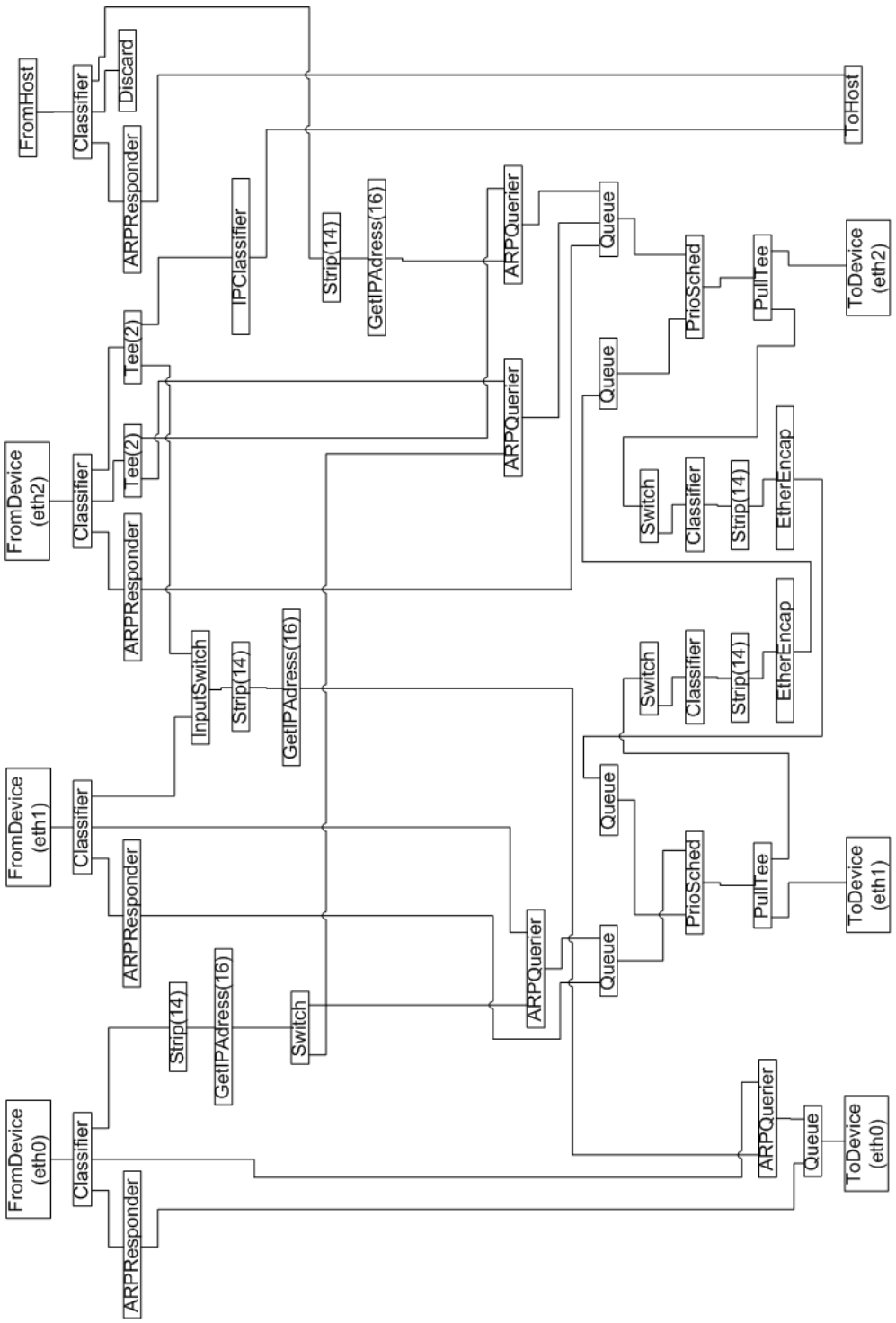
Met een kruislingse koppeling tussen deze twee wachtrijen gaan er geen pakketten meer verloren. De werking wordt uitgelegd aan de hand van figuur 4.7. Neem dat de pakketten op de mobiele node binnenkomen via de bedrade verbinding, dus via interface eth2. De pakketten gaan dan in rechter wachtrij. De linker wachtrij (van het rechter gedeelte) is leeg. In dit geval probeert de pull van de netwerkkaart, doorgegeven naar de juiste wachtrij met behulp van PrioSched[6], pakketten van de linker wachtrij te nemen. Omdat deze leeg is, neemt de kaart pakketten van de rechter wachtrij.

De pakketten worden door PullTee gekopieerd naar de switch. Deze laat de pakketten niet door. PullTee wordt gebruikt omdat na een buffer de pakketten gepulled worden. De tweede uitgang van de PullTee is een push-uitgang omdat deze pakketten in een wachtrij geduwd worden.

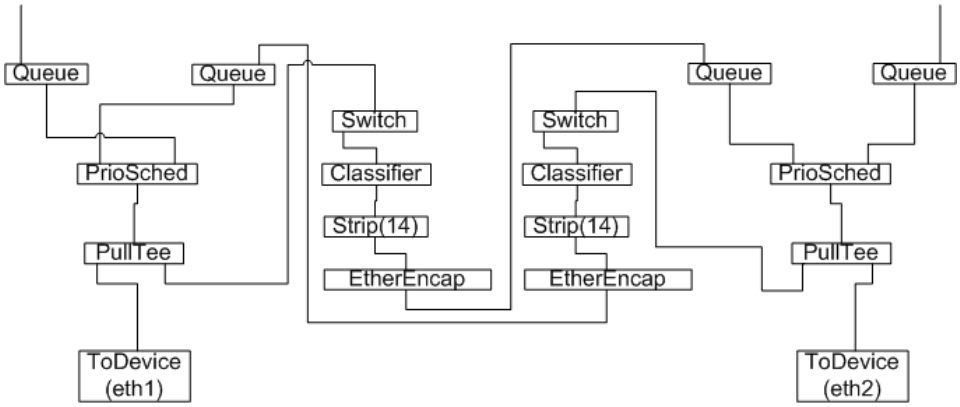
Bij een handover gaat de rechter Switch open. De pakketten gaan nu niet meer naar de rechter wachtrij, maar naar de uiterst linkse. Pakketten in de buffer van interface eth2 worden gekopieerd naar de tweede wachtrij links. Voordat de pakketten in deze wachtrij komen passeren ze eerst langs verschillende elementen.

De Classifier zorgt ervoor dat alleen de videostream-pakketten in de wachtrij belanden. De tweede wachtrij heeft immers voorrang op de eerste. Dus als in de wachtrij te veel pakketten

[6]Het PrioSched element probeert eerst pakketten van de eerste ingang te nemen, daarna van de tweede, enz..



Figuur 4.6: Verbeterde configuratie met kruislingse koppeling tussen de wachtrijen van de bedrade en draadloze verbinding



Figuur 4.7: Detail van de kruislingse koppeling tussen de wachtrijen van de bedrade en draadloze verbinding

zouden komen, komt de eerste (waar nu de videopakketten toekomen) niet zo snel aan bod. Deze vertraging kan het verschil maken tussen een paar pakketten die te laat zijn of nog net op tijd.

Strip(14) verwijdert de Ethernethoofding. EtherEncap zet er weer een nieuwe op. Dit is nodig om de pakketten met de bedrade verbinding als nieuwe bestemming het juiste MAC te geven. Zij hebben immers een ander MAC-paar nodig dan deze voor de draadloze verbinding.

Tot slot komen de pakketten in een wachtrij die voorrang heeft t.o.v. de gewone wachtrij. Deze voorrang is nodig omdat de pakketten die achterbleven in de wachtrij van de andere verbinding eerder aangekomen zijn en dus ook best eerder worden verzonden.

De meest rechtse Switch is op dit moment gesloten om een lus te vermijden. De pakketten worden dan over beide interfaces gestuurd. Dit zou de streaming-toepassing in de war brengen en is ook zeker niet de bedoeling.

4.3.2 Client (mobiele node)

Voor de Client gebruiken we dezelfde configuratie als in 4.1.2.

4.3.3 Resultaten

Slechts één belangrijke verbetering werd getest, nl. de aangepaste buffer om pakketten die in de verkeerde wachtrij staan naar de juiste wachtrij te sturen. De buffer werd aangepast om pakketten die in de verkeerde wachtrij staan naar de juiste wachtrij te sturen.

De resultaten zijn gegenereerd met achtergrondtrafiek. De achtergrondtrafiek loopt nog steeds niet over de draadloze verbinding. Dit gebeurt pas in 4.4. Hiervoor is een FTP-server opgezet, samen met een simulatieprogramma voor achtergrondverkeer [**?**]. Dit programma genereert twee TCP streams en twee UDP streams. De aankomsten van de pakketten zijn poissonverdeeld en hebben een frequentie van 50 pakketten per seconde. Twee streams vertegenwoordigen de grote

Tabel 4.3: Verloren frames implementatie 4

	Buffer		**Carrier**	
	Duur(ms)	Totaal(♯)	Duur(ms)	Totaal(♯)
Handover	566,68+166,68	54+13	773,22+226,52	69+24

Tabel 4.4: Kenmerken vertraging implementatie 4 met achtergrondverkeer.

$\overline{x}$ (ms)	s^2	Minimum (ms)	Maximum(ms)
110,46	8,43	108,74	129,68

pakketten en twee de kleine. De pakketgroottes worden uniform verdeeld tussen de 46 en 56 bytes - kleine pakketten - en 1200 en 1300 bytes - grote pakketten -. De interfaces hebben hier hun volledige snelheid, nl. 100Mbps voor de bedrade en 11Mbps voor de draadloze verbinding.

Deze implementatie wordt vergeleken met die van het carrier-script in sectie 4.2.4 omdat het enige verschil tussen deze twee de omleiding van de buffer is.

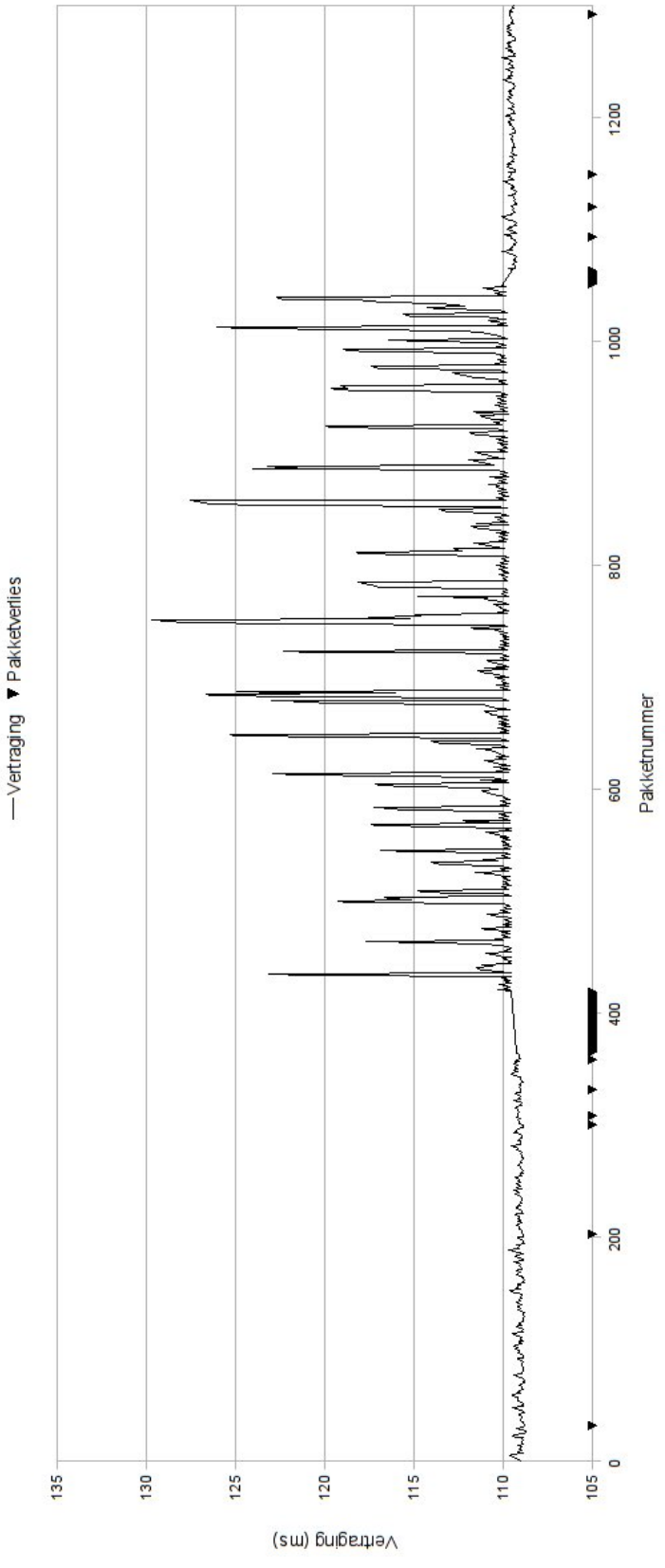
4.3.3.1 Verloren frames

Tabel 4.3 laat zien dat er minder frames verloren gaan. Dit geeft weliswaar een vertekend beeld, mits de duur van de handover bij deze test korter was. Daarom zijn tellers gezet op de verbindingen tussen de wachtrij naar de draadloze en de bedrade verbinding. Zo kan het aantal pakketten dat door de buffer gaat, worden gemeten. Er werden twee pakketten gemeten bij de overgang van draadloze naar bedrade verbinding. Andersom wisselden geen pakketten van wachtrij. Dit is omdat de bedrade link een hogere snelheid heeft en zijn wachtrij dus ook meestal leeg is.

4.3.3.2 Vertraging

De piek in figuur 4.5 bij de overgang naar de bedrade verbinding toonde duidelijk dat de resterende pakketten in de wachtrij naar de bedrade verbinding worden doorgestuurd. De grote vertraging zorgt ervoor dat deze pakketten onbruikbaar zijn aan de ontvangstzijde omdat ze na de deadline aankomen. De pakketten zorgen dus voor onnodig verkeer.

In figuur 4.8 is deze piek niet meer aanwezig. Het probleem van overblijvende pakketten in de uitgaande wachtrij is dus opgelost. Verder is het verloop gelijkaardig aan die van de vorige implementatie. De karakteristieken zijn weergegeven in tabel 4.4.



Figuur 4.8: Pakketvertraging implementatie 4 met achtergrondverkeer.Kenmerken: $\overline{x} = 110,46$ms; $s^2 = 8,43$; $Min = 108,74$ms; $Max = 129,68$ms

4.4 Implementatie 5: Volautomatische handover

In 4.2 werden verschillende manieren van detectie besproken. De snelheid van carrier-detectie gaf de voorkeur aan deze techniek. Er werd ook opgemerkt dat de volledige verbinding hier niet wordt gecontroleerd. Enkel de directe verbinding. Een dergelijke techniek is dus niet aan te raden voor een gedistribueerde manier van detectie. In deze sectie werd daarom een aanpassing gemaakt. Hier maakt niet meer ieder element voor zich een beslissing, maar wordt deze beslissing ook doorgespeeld naar andere netwerkelementen. Deze elementen maken aan de hand van deze informatie de nodige aanpassingen. In deze implementatie wordt enkel de router op de hoogte gebracht van deze beslissing. In hoofdstuk 5 wordt een stap verder gegaan.

Deze techniek merkt congestie niet opmerkt. Om redenen van modulariteit is het beter om het vermijden van congestie en het detecteren van een verbroken verbinding gescheiden te houden.

De eerste aanpassing in deze implementatie is het automatiseren van de handover. Hier draait enkel een script op de mobiele node. Deze detecteert wanneer de bekabelde connectie wordt verbroken. De mobiele node zendt dan een pakket naar de gemeenschappelijke node, naar de server en naar de achtergrondclient. Merk op dat in een uitgebreide netwerkomgeving de router ook het wegvallen van de verbinding kan detecteren en deze informatie doorspelen aan de Client.

De tweede aanpassing is het meten van de signaalsterkte van de draadloze verbinding. Als deze sterkte beneden een bepaald niveau zakt, gebeurt er een horizontale handover[7].

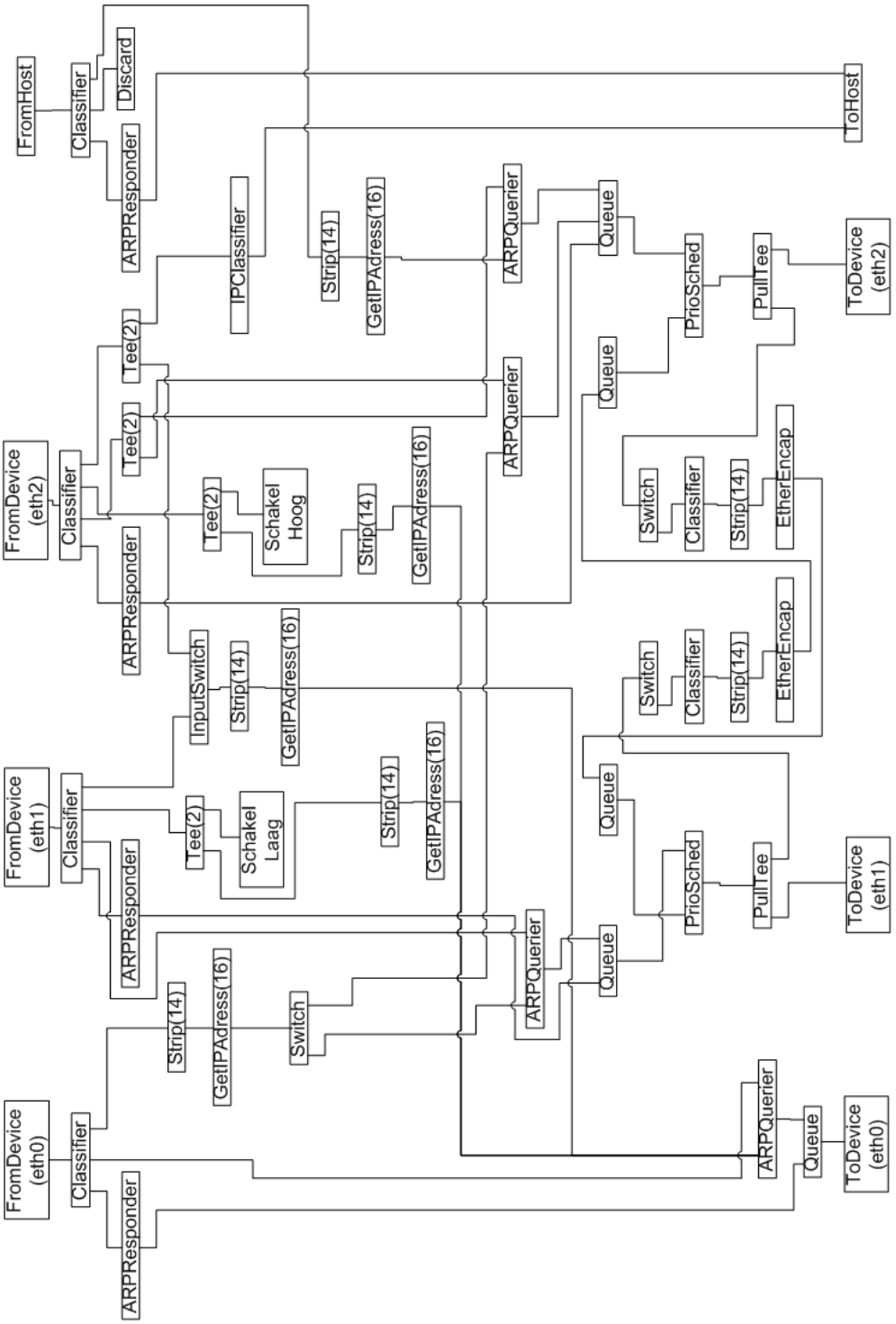
4.4.1 Router (gemeenschappelijke node)

In de gemeenschappelijke node was een aanpassing vereist om, bij aankomst van een bepaald pakket, te switchen en dus ook het gevolgde pad om te zetten. Het element dat deze omschakeling voorziet, is het zelf geschreven element Schakel. De code van dit element is te vinden in A.15.

Om te zorgen dat het juiste pakket bij het juiste Schakelelement komt, worden Classifiers gebruikt. Deze sorteren het pakket op inhoud. In dit geval is de inhoud “hoog” of “laag”. Het sturen van het pakket gebeurt in 4.4.2.

Bij het gebruiken van de bedrade verbinding, zal het wegvallen van de link gedetecteerd worden door de mobiele node. Deze zendt dan een pakket met de inhoud “laag”. Laag wil zeggen dat de bitrate verlaagd moet worden en een omschakeling naar de draadloze verbinding nodig is. De mobiele node stuurt dit pakket ook langs de draadloze verbinding (er is geen kabelverbinding op dat moment). Dit pakket komt binnen in de gemeenschappelijke node op interface eth2. De Classifier zal dit pakket herkennen en doorsturen naar het Schakelelement. Het Schakelelement zorgt ervoor dat de switchen worden omgeschakeld zodat het videoverkeer nu over de draadloze verbinding gaat. De gemeenschappelijke node kopieert het pakket ook naar interface eth0 van de server. De server past dan de bitrate aan afhankelijk van het soort verbinding. Dit is besproken in 5.1. Het kopiëren gebeurt met behulp van het Tee element. Voor de verzending wordt het MAC-paar en het IP adres aangepast. Dit gebeurt met de elementen Strip(14) en GetIPaddress(16).

[7] Een horizontale handover is een handover tussen twee netwerken van hetzelfde type. In dit geval IEEE802.11, dus draadloos naar draadloos.



Figuur 4.9: Automatische omschakeling bij het ontvangen van een bepaald pakket

De implementatie is te zien op figuur 4.9. De Click-configuratie is terug te vinden in A.9.

4.4.2 Client (mobiele node)

Aan de mobiele node zijn twee aanpassingen gebeurd. De eerste aanpassing is het zenden van een pakket bij het verbreken of herstellen van de kabelverbinding.

De tweede aanpassing is het meten van de signaalsterkte van de draadloze verbinding en eventueel overschakelen als de signaalsterkte te zwak wordt.

Beide verbeteringen zijn te zien in figuur 4.10. De Click-configuratie is terug te vinden in A.10.

De vier Sender elementen zijn instanties van het standaardelement ICMPPingSource. De configuratiestring van dit element is:

ICMPPingSource(192.168.3.1, 192.168.6.1, LIMIT 1, ACTIVE false, DATA "laag")
ICMPPingSource(192.168.5.1, 192.168.6.1, LIMIT 1, ACTIVE false, DATA "laag")

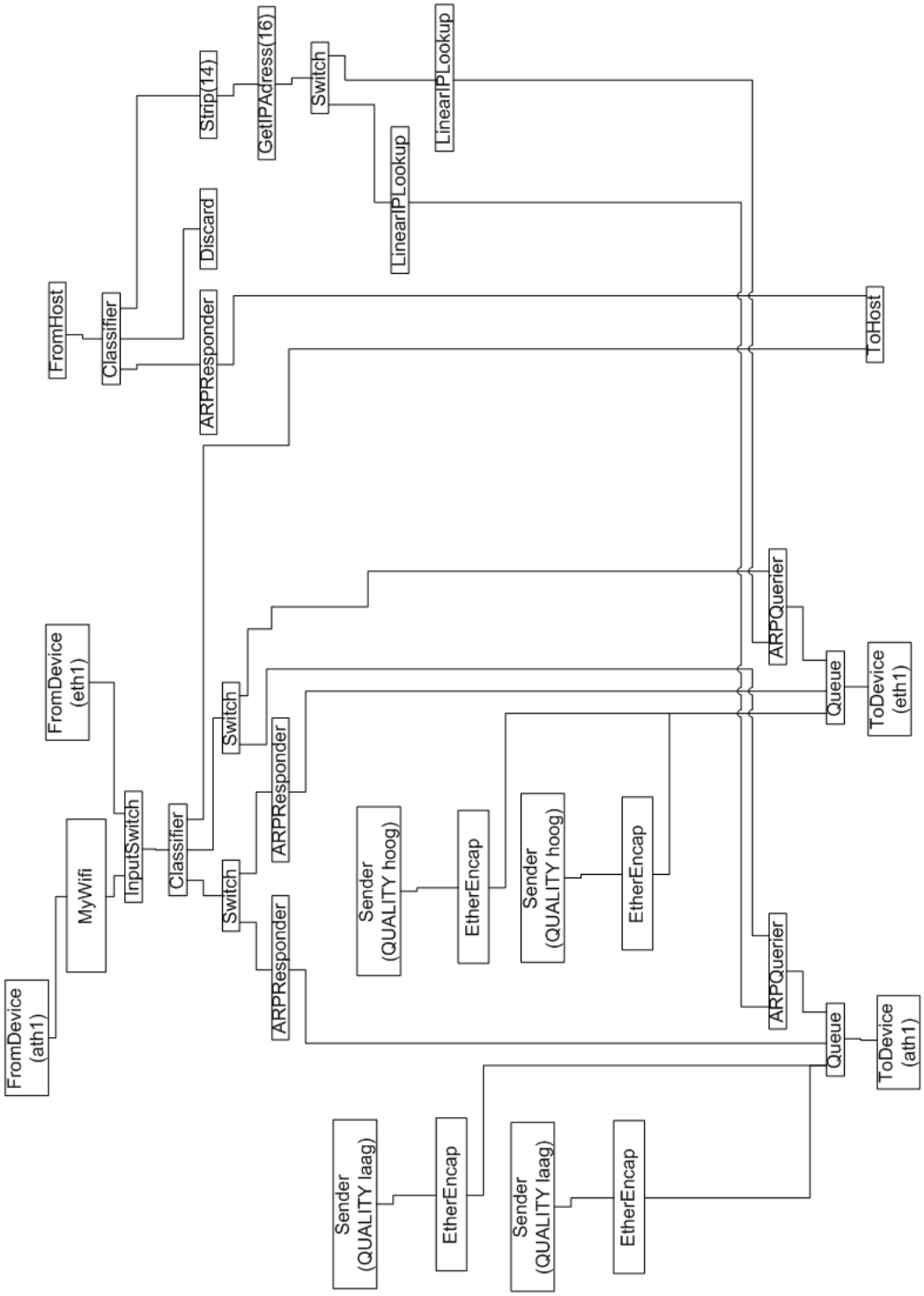
Hier wordt dus het bron- en bestemming-IP adres meegegeven en wordt aangegeven dat er slechts één pakket gezonden moet worden en dat het element niet actief is. Het wordt pas actief bij het activeren van de ACTIVE handler. Ook wordt de informatie "hoog" of "laag" aan de handler DATA meegeven. Het juist zetten van de MAC-adressen gebeurt weer met EthernetEncap. Het pakket wordt zowel naar de server als naar de achtergrondclient gestuurd. De achtergrondclient moet ook weten of hij op de bedrade of draadloze verbinding moet luisteren. Hoe de server dit pakket verwerkt, wordt uitgelegd in 5.1. Hoe de achtergrondclient omgaat met dit pakket wordt besproken in 5.2.

Het element MyWifi is een zelf geschreven element dat de signaalsterkte van de draadloze verbinding meet en een handover initieert indien de signaalsterkte te laag is. De code van dit element is te vinden in A.16.

Om deze aanpassingen te laten werken, is er een monitorprogramma[8] nodig dat de signaalsterkte in een handler zet. Zo kan MyWifi deze waardes gebruiken. Dit gebeurt op lijnen 4 en 5 van het onderstaand script. Buiten het de signaalsterkte wordt ook de kabelverbinding gemonitord. Dit is te zien op lijn 6 van het script. De handler voor het sturen van het ICMP pakket met hoog of laag wordt ook aangesproken op lijnen 16, 17, 18, 19, 32, 33, 34 en 35. De omschakeling van de mobiele node gebeurt hier ook op lijnen 12-15 en 26-29.

```
1  wire=1
2  while true
3  do
4  link=‘cat /sys/class/net/ath1/wireless/link | grep -oE "[[:digit:]]{1,}"‘
5  echo $link > /Click/meet/signal
6  onoff=‘cat /sys/class/net/eth1/carrier‘
7  if [ "$onoff" == "0" ]
8  then
9  if [ "$wire" == "1" ]
10 then
```

[8] Achtergrondproces dat iets, in dit geval de signaalsterkte, continu controleert.



Figuur 4.10: Implementatie meten van de signaalsterkte en het versturen van een pakket bij handover

```
11 echo "wire-wireless"
12 echo 0 > /proc/Click/combinator/inputswitch
13 echo 0 > /proc/Click/verdelerARP2/switch
14 echo 0 > /proc/Click/verdelerARPQ/switch
15 echo 0 > /proc/Click/verdelerROUTEtoDev/switch
16 echo false > /Click/senderlaag/active
17 echo true > /Click/senderlaag/active
18 echo false > /Click/senderlaag1/active
19 echo true > /Click/senderlaag1/active
20 wire=0
21 fi
22 fi
23 if [ "$onoff" == "1" ]
24 then
25 if [ "$wire" == "0" ]
26 then
27 echo "wireless-wire"
28 echo 1 > /proc/Click/combinator/inputswitch
29 echo 1 > /proc/Click/verdelerARP2/switch
30 echo 1 > /proc/Click/verdelerARPQ/switch
31 echo 1 > /proc/Click/verdelerROUTEtoDev/switch
32 echo false > /Click/senderhoog/active
33 echo true > /Click/senderhoog/active
34 echo false > /Click/senderhoog1/active
35 echo true > /Click/senderhoog1/active
36 wire=1
37 fi
38 fi
39 done
```

4.4.3 Resultaten

Dit is het eerste resultaat waarbij de achtergrondtrafiek ook over de draadloze verbinding loopt. Het effect van het sturen van een film met achtergrondverkeer over een link die plots halveert in bandbreedte, wordt hier getest.

De resultaten zijn gegenereerd met achtergrondtrafiek. Enkel het simulatieprogramma voor achtergrondverkeer [**?**] werd gebruikt. De instellingen zijn dezelfde als in 4.3.3. De bandbreedte werd geschaald om congestie op het netwerk te krijgen. De bedrade link werd hiervoor aangepast tot een maximale bitsnelheid van 2000 kbps[9] en de draadloze 1000 kbps.

4.4.3.1 Tijdige levering

Op figuur 4.11 is de tijdige aankomst van de pakketten af te lezen. In het begin loopt de video over de bekabelde verbinding en komen de pakketten op tijd aan. Daarna is er een verandering in de richtingscoëfficient van de aankomsttijd waarneembaar. Hier gebeurt de handover. De eerste pakketten die in de lege buffer terecht komen ondervinden een zeer lage vertraging, vandaar de dip in de aankomsttijd. Pakketten worden overgeheveld naar de andere wachtrij

[9] kbps is kilobits per seconde

Tabel 4.5: Verloren frames implementatie 5

	Per type	Totaal
I-frames(%)	26,57	3,56
P-frames(%)	27,97	7,64
B-frames(%)	28,63	16,97
	Duur(ms)	Totaal(♯)
Handover	133,36+66,89	14+9

terwijl nieuwe pakketten aankomen. De vertraging loopt op totdat de draadloze verbinding de toevloed aan pakketten weer kan verwerken. In dit geval kan de draadloze verbinding in feite de pakketten helemaal niet verwerken. Hierover meer in 4.4.3.2. Er is wel te zien dat de vertraging te hoog ligt, de pakketten komen te laat via de draadloze verbinding. Dit duidt erop dat de gekozen bitrate en achtergrondverkeer te hoog zijn voor de bekabelde verbinding. Deze vertraging is in te perken met behulp van QoS (5.3) en een aanpassing van de bitrate bij de server (5.1).

Bij overschakeling naar de bedrade verbinding, vindt eerst een lage vertraging plaats, pakketten worden overgeheveld. De bedrade verbinding herstelt zich keurig en blijft tussen de grenzen.

4.4.3.2 Verloren frames

In tabel 4.5 is te zien dat er veel frames verloren gaan. Deze gaan verloren tijdens de handover en gedurende de congestieperiode in de draadloze verbinding. Meer over deze congestieperiode in 4.4.3.3. Er wordt geen onderscheid gemaakt tussen de verschillende frames. Per type gaan ongeveer evenveel I-, P-, en B-frames verloren.

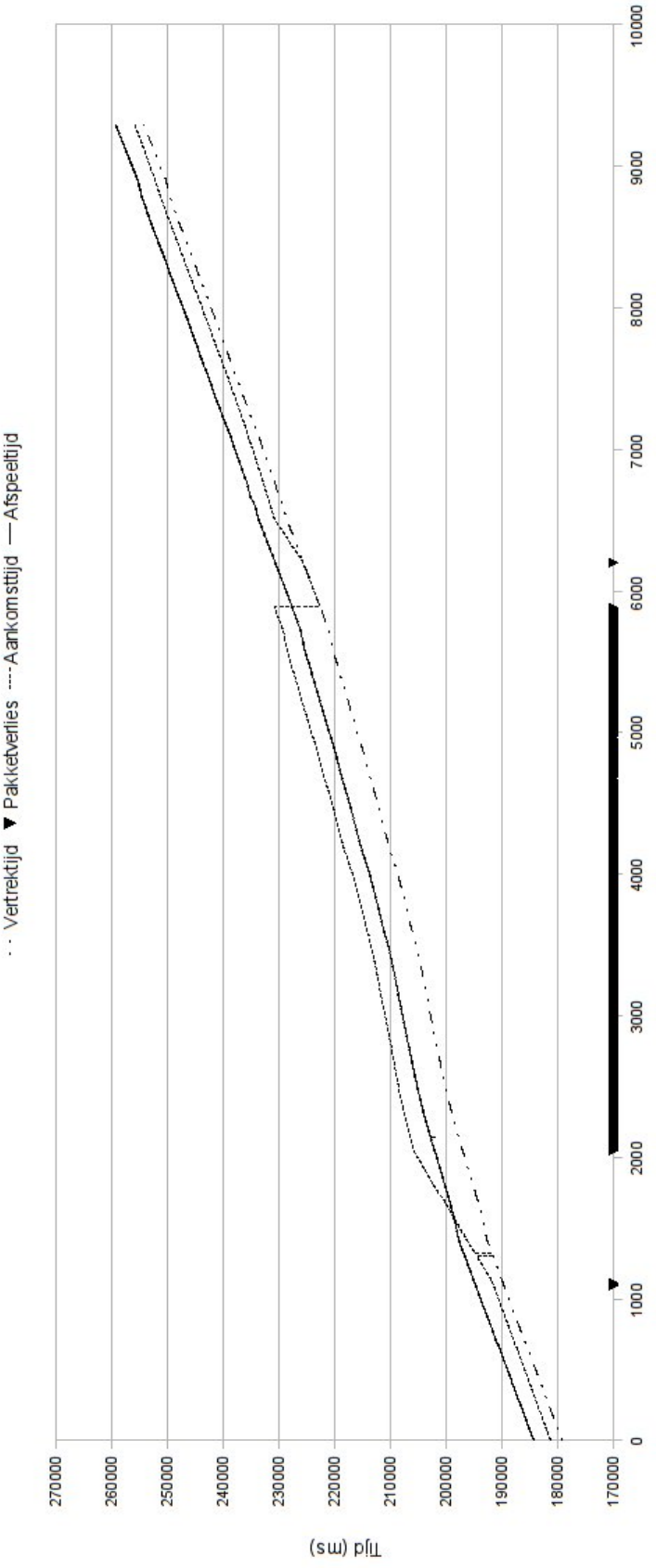
4.4.3.3 Vertraging

De kenmerken van de vertraging zijn weergegeven in tabel 4.6 en te zien op figuur 4.11. Zoals reeds vermeld, vindt vanaf de handover een verhoging van de vertraging plaats. De eerste filmpakketten die in de nieuwe buffer terechtkomen hebben een zeer lage vertraging. De vertraging stijgt hierna snel totdat hij een maximum bereikt. Dit vindt plaats wanneer de wachtrij vol zit. Hier gaan het gros aan pakketten verloren. Dit blijft duren totdat er opnieuw een handover gebeurt naar de bedrade verbinding. Eerst een zeer lage vertraging terwijl de pakketten van de film worden overgeheveld naar de tweede buffer. Eens alle verkeer weer langs de buffer loopt, wordt de vertraging nagenoeg constant.

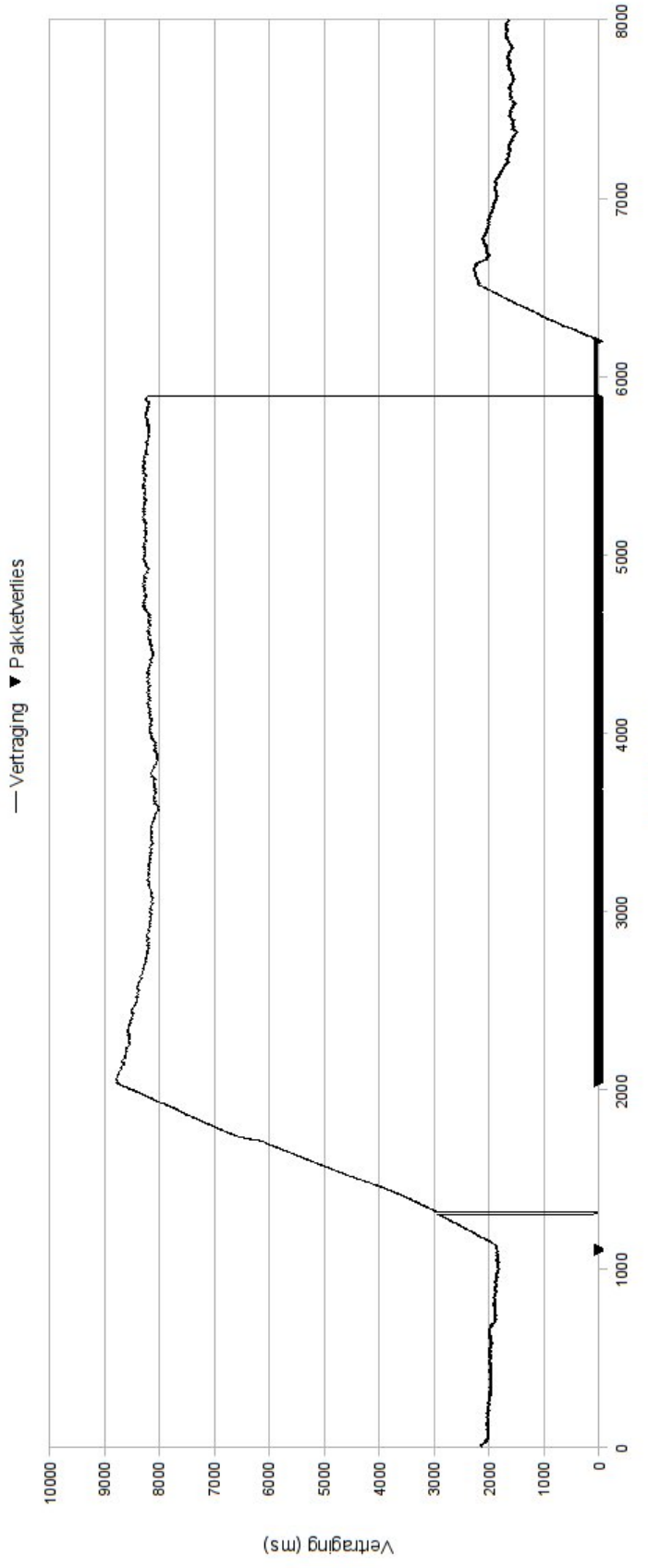
De grote variantie op de vertraging is te wijten aan het verschil tussen de snelheid van de bedrade en draadloze link. Dit is te zien aan het grote verschil tussen de minimum en maximum vertraging.

Tabel 4.6: Kenmerken vertraging implementatie 5 met achtergrondverkeer.

$\overline{x}$ (ms)	s^2	Minimum (ms)	Maximum(ms)
3339,54	7695972,93	43,02	8791,26



Figuur 4.11: Tijdige levering implementatie 5 met achtergrondtrafiek



Figuur 4.12: Pakketvertraging implementatie 5 met achtergrondverkeer.Kenmerken: $\overline{x} = 3339,54$ms; $s^2 = 7695972,93$; Min= $43,02$ms; Max= $8791,26$ms

4.5 Besluit

In dit hoofdstuk zijn verscheidene aanpassingen gebeurd aan de gemeenschappelijke node en de bestemmeling.

Sectie 4.1 behandelde de draadloze verbinding met de bestemmeling. Met behulp van Click werd ervoor gezorgd dat het mogelijk werd een zachte handover te bewerkstelligen tussen de bedrade en draadloze link. Om ervoor te zorgen dat de bron niet op de hoogte moet worden gebracht van eventuele veranderingen, werd ervoor gezorgd dat de bestemmeling één enkel IP adres heeft voor beide interfaces. De router verstuurt enkel over de actieve verbinding en met behulp van Click werd het mogelijk de bestemmeling enkel te laten luisteren naar één interface, de bedrade of de draadloze.

Sectie 4.2 behandelde verscheidene mechanismes om in een handover tussen de twee types verbindingen te voorzien. Het PING-mechanisme bleek zeer traag. Te traag voor de beoogde doeleinden. Wel had het als voordeel dat het volledige pad tussen de router en de bestemmeling wordt nagekeken. Het carrier-script controleert enkel de kabelverbinding van de bestemmeling. Indien er extra netwerkelementen tussen de router en de bestemmeling staan, werkt dit niet meer. De veel snelleren resulaten van het carrier-script waren overtuigend.

Sectie 4.3 behandelde een methode om pakketten om te leiden naar de juiste wachtrij. Dit zorgt ervoor dat bij een handover minder pakketten verloren gaan. De “In-flight”-pakketten zijn een verloren zaak, maar de pakketten in de buffers kunnen gered worden. Dit kan ervoor zorgen dat belangrijke I-frames niet verloren gaan.

Sectie 4.4 behandelde een mechanisme zodat enkel de bestemmeling de toestand van de bedrade verbinding met het volgende netwerkelement controleert en de gemeenschappelijke node hiervan op de hoogte brengt bij een wijziging. Dit mechanisme maakt het mogelijk extra netwerkelementen tussen de bestemmeling en de router te plaatsen. Van dit mechanisme kan ook gebruik worden gemaakt om meerdere netwerkelementen op de hoogte te brengen van deze verandering.

De resultaten tonen aan dat het handovermechanisme snel werkt en dat de omleidingsbuffer werkt als bedoeld.

Hoofdstuk 5

Router met QoS

In dit hoofdstuk wordt QoS aan de router toegevoegd. Op deze manier krijgt het streaming-verkeer op bepaalde manieren voorrang op ander verkeer, ook achtergrondverkeer.

De server krijgt extra functionaliteit. De mogelijkheid om bij een handover een stroom met andere bitrate te versturen, wordt hier voorzien. Dit wordt besproken in 5.1.

In de tweede sectie 5.2 wordt de aanpassing van de achtergrondclient gegeven. Deze moet weten of hij op de draadloze of bedrade verbinding moet luisteren.

In de derde sectie 5.3 van dit hoofdstuk wordt het verkeer opgesplitst in video en achtergrond-trafiek.

In de vierde sectie 5.4 van dit hoofdstuk wordt het verkeer verder opgedeeld. Zowel de I-, P- als B-frames worden op een aparte manier behandeld. De I- en P-frames vormen de basis laag van de film en de B-frames vormen de verbeteringslaag.

De Client (bestemmeling van het videoverkeer) krijgt geen aanpassing in dit hoofdstuk.

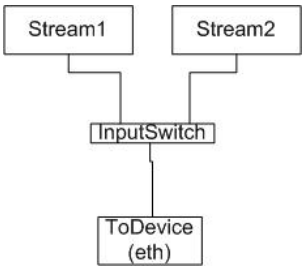
5.1 Aanpassing server

Zoals reeds beschreven in 2.2.2, is het mogelijk om de server te laten inspelen op veranderingen in het netwerk. De toegevoegde functionaliteit in deze implementatie, is deze van meerdere versies. Hierover is meer informatie terug te vinden in 2.2.2.1. Samengevat is het idee dat stromen met een verschillende kwaliteit, en dus ook de bitsnelheid, op de bron ter beschikking zijn.

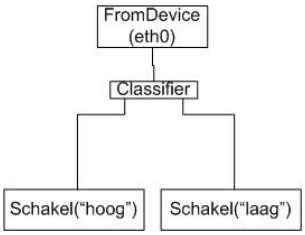
Omdat het om een heterogeen netwerk gaat, werd er besloten de aanpassingen te doen als volgt. De server begint over de UTP verbinding te zenden met hoge kwaliteit. Eens overgeschakeld naar de draadloze interface, verlaagt de zendkwaliteit en vice versa.

Om deze functionaliteit te implementeren zijn verschillende mogelijkheden onderzocht. De eerste mogelijkheid was in Click een element te bouwen met twee ingangen, nl. InputSwitch. Meer informatie over de InputSwitch is terug te vinden in 4.1.1. De ene ingang bevat de stroom met hoge kwaliteit, de andere die met lage kwaliteit. Als er een handover gebeurt, van de

bedrade naar de draadloze verbinding, wordt een pakket met de inhoud “laag” gestuurd vanuit de Client. Een Classifier zorgt ervoor dat dit pakket op de juiste uitgang arriveert. Aan deze uitgang hangt het element Schakel. Deze past de handler aan zodat de juiste ingang wordt geactiveerd. Meer informatie over dit element is terug te vinden in 4.4.2. De configuratie is samengevat in figuren 5.1 en 5.2. Er is niet geopteerd voor deze optie omdat dit de server teveel belast. Deze moet hiervoor nl. twee stromen tegelijkertijd afspelen.

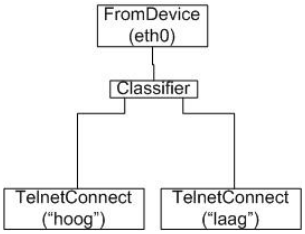


Figuur 5.1: Twee stromen met InputSwitch



Figuur 5.2: Schakelconfiguratie

De volgende optie leek wel valabel. De bedoeling was om excessief overladen van de server te vermijden. Om dit te bekomen kan er van stroom worden gewisseld in plaats van twee stromen tegelijkertijd af te spelen. De werkwijze verloopt identiek aan de vorige methode. Een Classifier om de boodschappen “hoog” en “laag” te filteren, gevolgd door een nieuw element TelnetConnect. De code hiervan is terug te vinden in A.17. Aan dit element kan de gewenste kwaliteit worden meegegeven. De configuratie is samengevat in figuur 5.3.



Figuur 5.3: Schakelen a.d.h.v. telnet interface

Om meer functionaliteit te kunnen voorzien, werd TelnetConnect geïmplementeerd. Deze maakt gebruikt van de telnet interface in VLC[**?**]. De telnetserver van VLC heeft een uitgebreide waaier aan mogelijkheden. Naast het starten en stoppen van een stroom, kan een “video op aanvraag”-server geconfigureerd worden en kunnen verscheidene details van een lopende stroom

opgevraagd worden. Door een socket[1] op te zetten tussen Click en de VLC server, is het mogelijk deze functionaliteit te benutten. De volgende functionele stappen worden achtereenvolgens uitgevoerd door TelnetConnect:

1. De server op voorhand instellen.
2. De tweede stroom opstarten.
3. De tweede stroom synchroniseren.
4. De eerste stroom af zetten.
5. De tweede stroom laten spelen.

Bij het testen van deze functionaliteit, treden er twee problemen op. De reactietijd van de server is te traag en de stroom wordt op de Client niet snel genoeg veranderd.

Bij het onderzoeken van het probleem, werd het volgende vastgesteld. Bij het ontvangen van een aanpassingspakket[2] laadt de server de nieuwe stroom. Daarna wordt de benodigde data, tijd of positie, via de telnetserver bekomen. Het TelnetConnect element probeert dan zo snel mogelijk de stroom te synchroniseren met de gewenste tijd. Het probleem hierbij is echter dat wanneer de strrom nog niet voldoende is ingeladen, de synchronisatie geen effect heeft. Om deze reden moet het programma gedurende bijna één seconde inactief worden. Dit is te lang voor snelle handover. Om dit euvel te verhelpen, werd geprobeerd twee stromen tegelijkertijd te laden, doch niet samen af te spelen.

Echter bij het beëindigen van de eerste stroom blijven er pakketten van de vorige stroom, achter in de buffer van de speler. Dit zorgt voor een slechte weergave van de stroom. De speler corrigeert dit pas veel later. De speler volledig afsluiten en opnieuw opstarten heeft ook geen zin, mits er dan eerst buffering plaatsvindt vooraleer er kan worden afgespeeld. Dit geeft opnieuw enkele seconden vertraging.

Om deze redenen is de functionaliteit moeilijk in te bouwen in reële stromen, tenzij de code in de server zelf wordt aangepast. Daarom werd er vanaf dit punt overgestapt naar een ander concept, nl. traces. Meer informatie in 2.2.3. Met traces worden pakketten gegenereerd in plaats van een echte stroom te gebruiken.

FakeStream leest een trace-bestand en haalt hieruit informatie: het framenummer, het frametype, de afspeeltijd en framegrootte. Hierna wordt gekeken naar de framegrootte en wordt bepaald of het frame in één pakket past of niet, rekening houdend met de maximale en minimale framegrootte, zoals berekend in 2.3.7. Als de framegrootte groter is, dan wordt het frame opgedeeld in meerdere pakketten. In het andere geval wordt er één pakket per frame gebruikt. De frames worden daarna doorgestuurd aan een snelheid van 30 frames per seconde. Indien een frame uit meerdere pakketten bestaat, worden dus ook meerdere pakketten sneller achter elkaar doorgestuurd.

De informatie uit het trace-bestand wordt dan in een string gezet. Deze string wordt in de informatie van het pakket ondergebracht. Als de string niet lang genoeg is, wordt hij meerdere keren herhaald. Als de string te kort is, wordt hij afgebroken. Er wordt een aanduiding van de kwaliteit en dus van de bitsnelheid in de string gebracht. Zo kan er bij de generatie van resultaten gemakkelijk worden nagegaan wanneer een handover is gebeurd.

[1] Sockets programmeren in C++ [**?**].
[2] ICMP pakket met de boodschap “hoog” of “laag”.

De Click-configuratie voor het gebruik van deze FakeStream is terug te vinden in A.7.

5.2 Aanpassing achtergrondclient

Ook de Client van het achtergrondverkeer past zich aan bij een handover. Hij moet weten op welke verbinding (bedrade of draadloze) de pakketten binnenkomen. De configuratie is afgeleid van deze van de mobiele node en is te zien in 5.4. De Click-configuratie van deze figuur is terug te vinden in A.8. De pakketten van zowel eth0 als ath0 zullen naar Linux worden gezonden. Hij moet alleen zijn gateway juist gaan zetten zodat de pakketen die van de achtergrondclient (TCP-bevestiging) komen op de achtergrondserver terecht komen.

5.3 Implementatie 6: Router met gescheiden verkeer

Deze implementatie is de eerste waarbij de videostroom ontsnapt aan de gevolgen van congestie op het netwerk. Om een QoS te voorzien aan de videostroom wordt gebruik gemaakt van prioritering. In de komende implementaties wordt de videostroom op verschillende manieren geprioriteerd en de gevolgen ervan bekeken.

In deze sectie krijgt het videoverkeer voorrang t.o.v. het andere verkeer. Op deze manier wordt geprobeerd bij congestie zo weinig mogelijk videopakketten te verliezen.

5.3.1 Server

Aanpassingen aan de server zoals besproken in 5.1 zijn gebeurd.

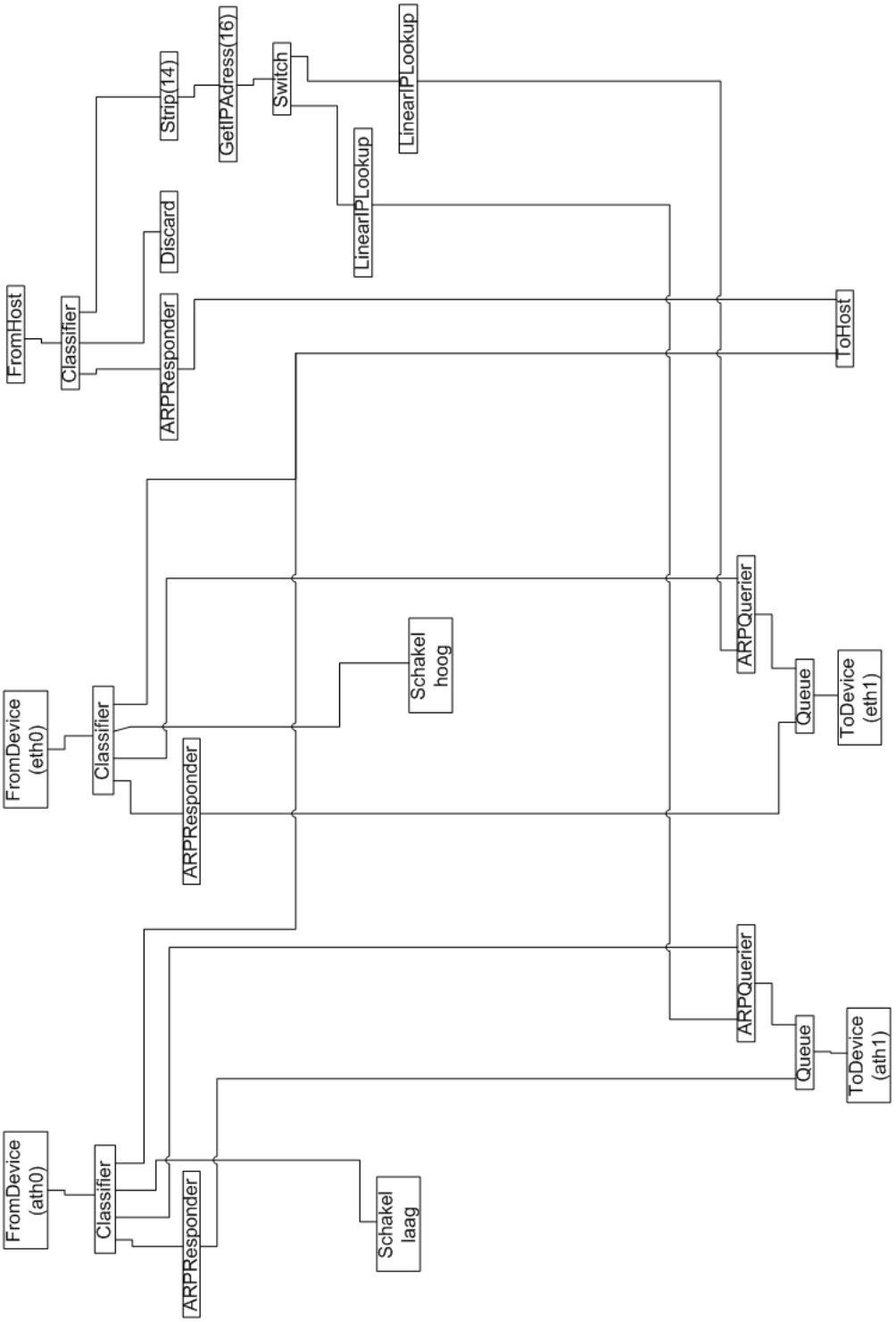
5.3.2 Router (Gemeenschappelijke node)

De aanpassing van de router is te zien in figuur 5.5. Op deze figuur staat enkel het gedeelte na de ARPQuerier elementen. Dit is de enige plaats waar de router wordt aangepast.

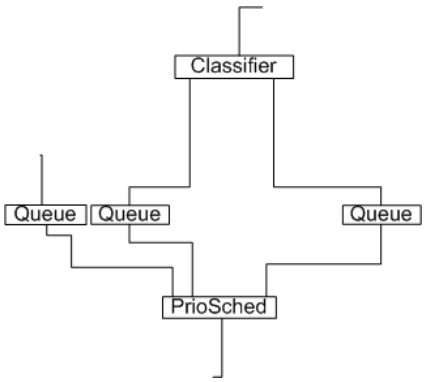
Het verkeer wordt door een Classifier opgesplitst in twee verschillende delen. De eerste uitgang van de Classifier is het videostreaming verkeer. De andere uitgang neemt het andere verkeer voor haar rekening. De meest linker buffer is onderdeel van de omleidingsbuffer. De drie wachtrijen worden op een PrioSched aangesloten. Dit element haalt eerst alle pakketten uit de eerste wachtrij. Als deze leeg is wordt overgegaan naar de volgende wachtrij. De pakketten in de meest linker wachtrij krijgen dus voorrang op de rest van het verkeer. Hier zitten de videopakketten.

Als een congestie op het netwerk ontstaat, gaat eerst het achtergrondverkeer verloren. Dit omdat de meest rechter wachtrij het minst frequent wordt geledigd, namelijk wanneer er geen videoverkeer in de andere wachtrij zit. Pakketten die niet meer in de wachtrij opgeslagen kunnen worden, worden vernietigd door de router.

De router wordt in zijn geheel weergegeven in 5.6. De Click-configuratie van deze opstelling is terug te vinden in A.11.



Figuur 5.4: De client van het achtergrond verkeer



Figuur 5.5: Detail van de router met opsplitsing van het verkeer in 2 soorten: video en ander verkeer

5.3.3 Resultaten

In deze sectie worden de twee aanpassingen apart geëvalueerd: eerst de verbetering aan de server en daarna de effecten van de ingevoerde QoS. Om het effect van de aanpassing van de server te genereren werd voor de gemeenschappelijke node en Client de Click-configuratie uit 4.4 gebruikt.

De resultaten zijn gegenereerd met achtergrondtrafiek. Enkel het simulatieprogramma voor achtergrondverkeer [**?**] werd gebruikt. De instellingen in het eerste resultaat zijn dezelfde als in 4.3.3. Voor het tweede resultaat werd de achtergrondtrafiek verhoogd. Het gemiddeld aantal pakketten per seconden is hier 100.

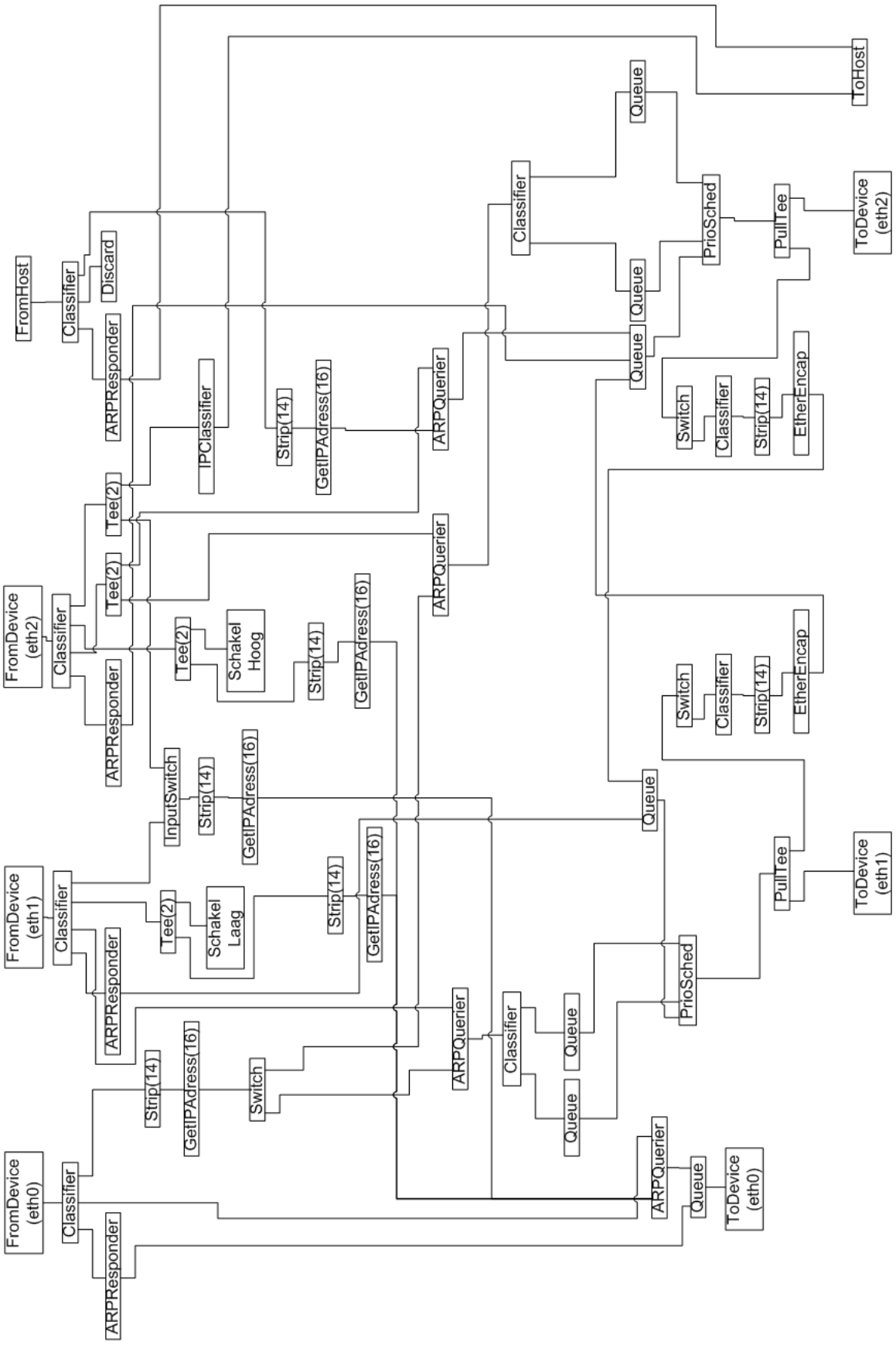
De bandbreedte is geschaald om congestie op het netwerk te krijgen. De bedrade link krijgt een maximale bitsnelheid van 2000 kbps[3] en de draadloze 1000 kbps.

5.3.3.1 Tijdige levering

Op figuur 5.7 is te zien dat de pakketten steeds op tijd aankomen. Na de handover is de helling minder sterk als de helling bij de testen zonder de aanpassing aan de server. De lagere bitrate zorgt ervoor dat de draadloze verbinding het verkeer kan verwerken. Er gaan hier ook geen frames meer verloren tijdens het gebruik van de draadloze verbinding. De val in vertraging na de handover van draadloze naar bedrade link is duidelijk op te merken.

Figuur 5.9 geeft aan dat de video altijd op tijd aankomt bij de implementatie met QoS. De marge van 5s is in dit geval niet meer nodig. Het verschil in helling van de curves is, net zoals het geval met aanpassing van bitrate, te wijten aan de lagere bitrate.

[3]kbps is kilobits per seconde



Figuur 5.6: Router uitgebreid met QoS

Tabel 5.1: Verloren frames implementatie 6

	Met serveraanpassing		**Met QoS**	
	Per type	Totaal	Per type	Totaal
I-frames(%)	0,00	0,00	0,00	0,00
P-frames(%)	0,00	0,00	0,00	0,00
B-frames(%)	0,00	0,00	0,00	0,00
	Duur(ms)	Totaal(♯)	Duur(ms)	Totaal(♯)
Handover	999,97	75	399,99	39

5.3.3.2 Verloren frames

Zoals te zien in tabel 5.1 gaan er in beide gevallen geen frames verloren. Het grote verschil tussen de twee aanpassingen is de vertraging van de film. In het resultaat met aanpassing van de server, loopt het verkeer door dezelfde wachtrij. Met QoS, zal de film een andere wachtrij (met hogere prioriteit) gebruiken. De vertraging hierop is dus lager.

Deze kleinere vertraging heeft zijn prijs. De opgeëiste tol is het aantal pakketten achtergrondverkeer dat verloren gaat. Omdat de film altijd voorrang krijgt, is de bediening van het achtergrondverkeer trager. Om dit effect te tonen, is het aantal verloren pakketten achtergrondverkeer ook bijgehouden. De implementatie met QoS bracht een verlies van 3,46% met zich mee.

5.3.3.3 Vertraging

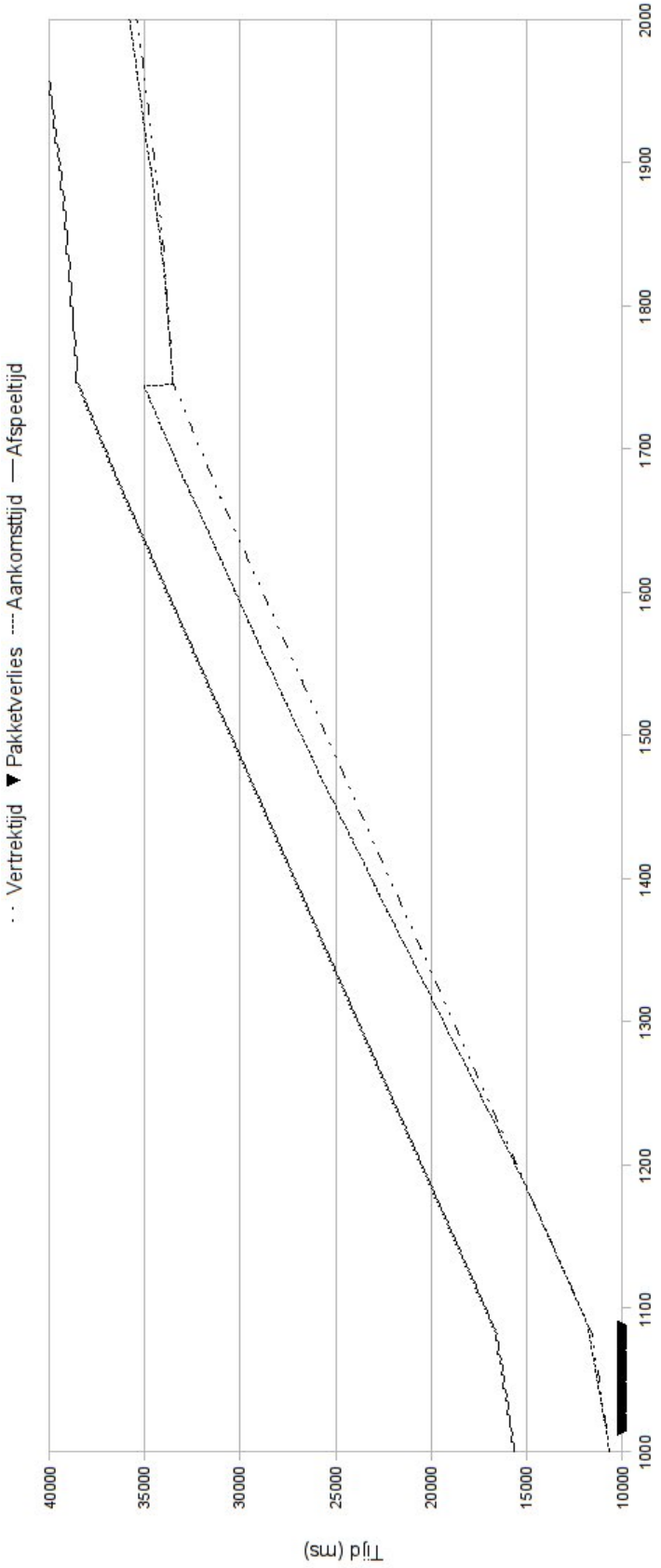
Figuur 5.8 geeft de vertraging met aanpassing van de server weer. Op het moment van de handover, loopt de vertraging sterk op omdat al het achtergrondverkeer nog steeds over de tragere draadloze link moet. Wel is dit veel minder dan zonder aanpassing van de bitrate. De pakketten blijven binnen de grens van 5s. Bij overgang van draadloze naar bedrade verbinding, zakt de vertraging. Daarna stijgt de curve weer. Dit is te wijten aan de variabele bitsnelheid van de video.

Op figuur 5.10 is te zien dat de vertraging vrij constant blijft, ongeacht de verbinding. De video krijgt dan ook voorrang op de rest van het verkeer. De eerste piek is te wijten aan de eerste overgang naar de draadloze verbinding. Vanaf ongeveer pakket 1300 wordt er weer overgeschakeld naar de bedrade verbinding. Hier wordt de vertraging eerst zeer klein totdat er weer meer frames in de buffer lopen. De piek rond frame 1800 is te wijten aan de variabele bitsnelheid van de film. De schijnbaar lagere variantie in de vertraging van de video over de draadloze link is te wijten aan de lagere bitrate. Mits de bitrate kleiner is, is de variabele bitrate van de video gemakkelijker te verwerken door de draadloze link.

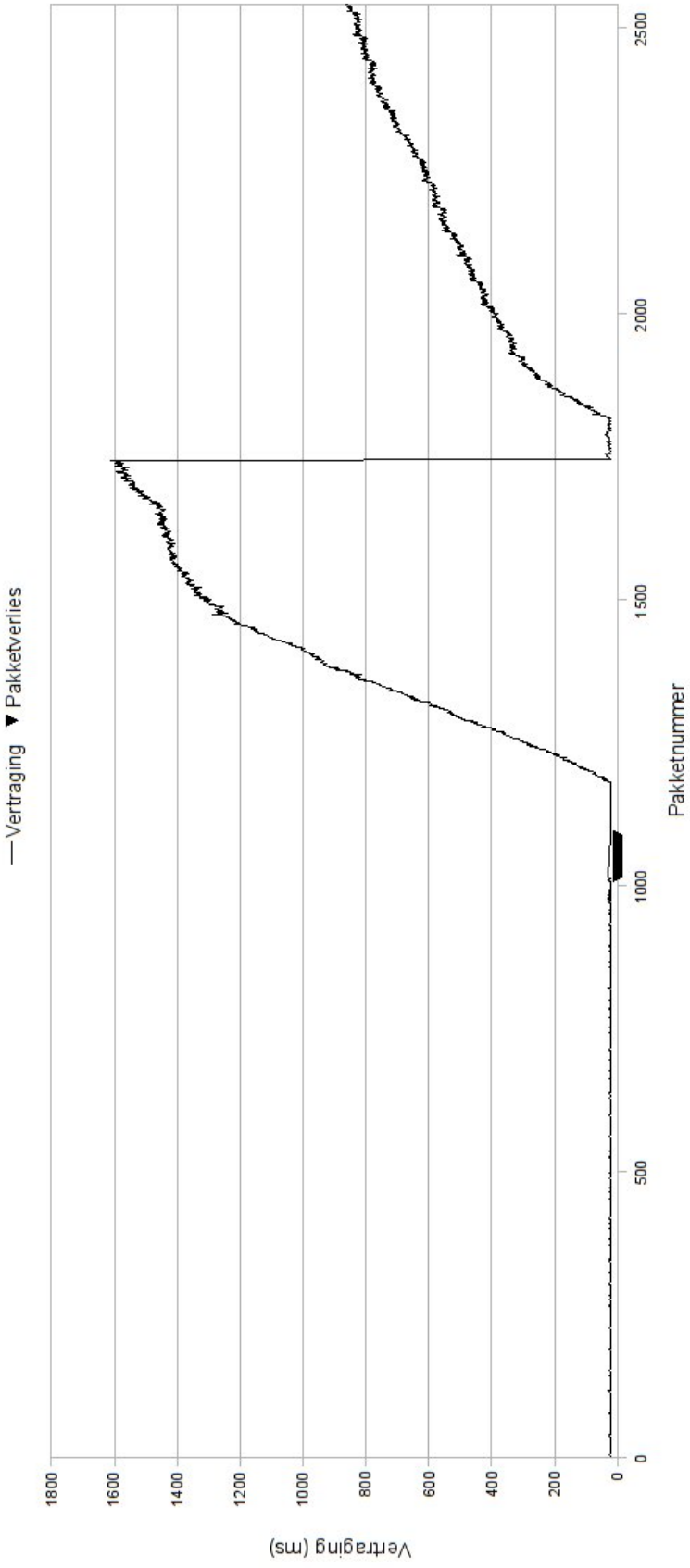
In tabel 5.2 toont aan dat zowel de gemiddelde vertraging als de variantie op de vertraging veel lager liggen bij QoS. Zo kan de buffer aan de ontvangstzijde ingeperkt worden tot een minimum.

Tabel 5.2: Kenmerken vertraging implementatie 6

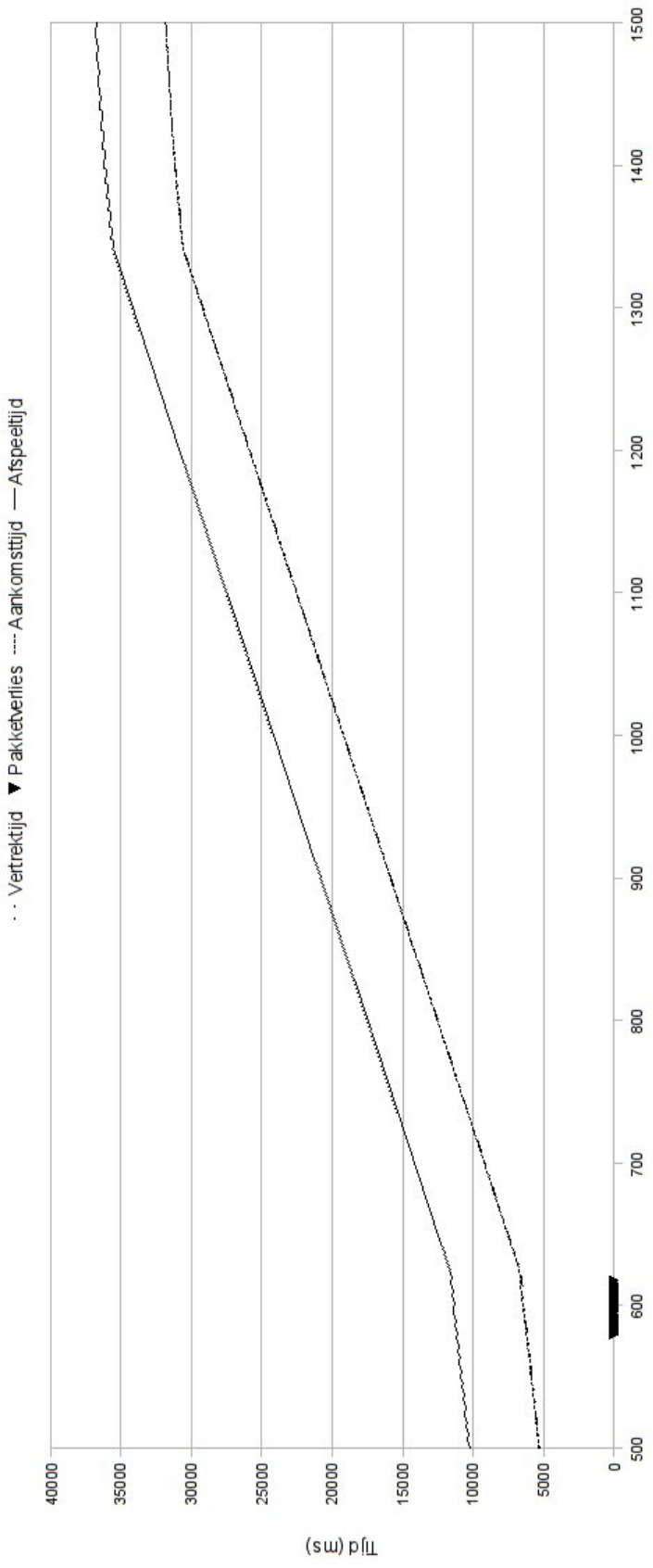
	$\overline{x}$(ms)	s^2	Minimum(ms)	Maximum(ms)
Met serveraanpassing	403,98	228218,54	19,76	1611,42
Met QoS	59,62	228,18	45,85	136,71



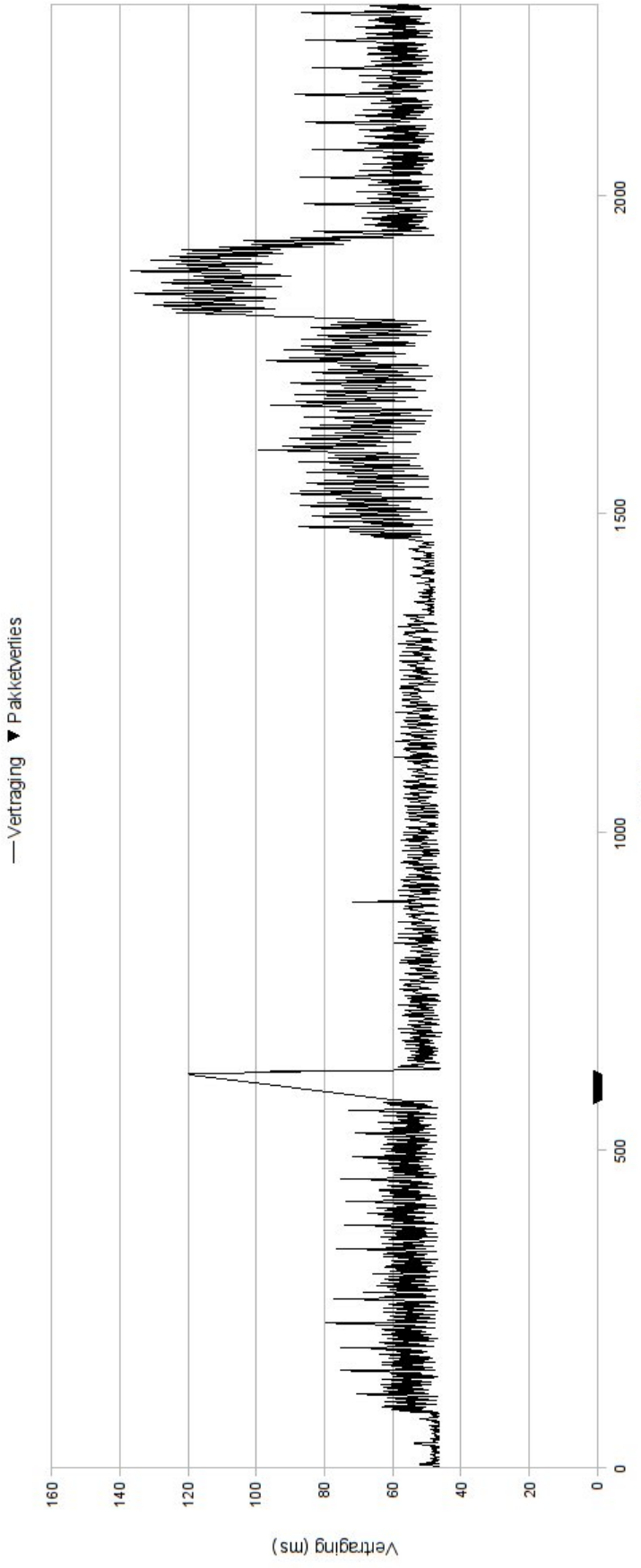
Figuur 5.7: Tijdige levering implementatie 5 met aanpassing en achtergrondtrafiek



Figuur 5.8: Pakketvertraging implementatie 5 met aanpassing en achtergrondverkeer.Kenmerken: $\overline{x} = 3339,54$ms; $s^2 = 7695972,93$; Min= 43,02ms; Max= 8791,26ms



Figuur 5.9: Tijdige levering implementatie 6 met QoS en achtergrondtrafiek



Figuur 5.10: Pakketvertraging implementatie 6 met QoS en achtergrondverkeer.Kenmerken: $\overline{x} = 3339,54$ms; $s^2 = 7695972,93$; Min= $43,02$ms; Max= $8791,26$ms

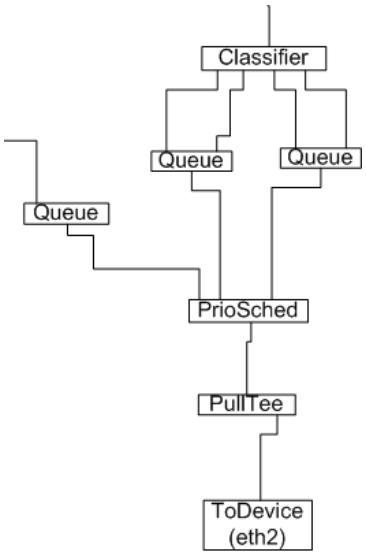
5.4 Implementatie 7: Router met gescheiden frames

In deze implementatie wordt de router verder uitgebreid. Een oplossing voor de stiefmoederlijke behandeling van het achtergrondverkeer in de vorige implementatie wordt hier aangeboden. Pakketten worden opgedeeld in pakketten die I-, P-, en B-frames bevatten zoals in 2.2.2.3 besproken. De I- en P-pakketten worden samen in een wachtrij gezet. Ze vormen de basislaag. De B-pakketten worden met het andere verkeer in een andere wachtrij ondergebracht. De B-pakketten vormen de verbeteringslaag. Zo wordt de verbeteringslaag van een even hoge prioriteit als de rest van het verkeer voorzien.

5.4.1 Server

De server blijft voor deze opstelling dezelfde als in 5.1.

5.4.2 Router (Gemeenschappelijke node)



Figuur 5.11: Detail van het QoS principe

Figuur 5.11 organiseert de QoS. De Classifier zet de pakketten in de juiste wachtrijen. Hij maakt een onderscheid tussen vier soorten pakketten. De pakketten met hun prioriteiten van hoog naar laag zijn:

- I-frame pakketten: hoge prioriteit
- P-frame pakketten: hoge prioriteit
- B-frame pakketten: lage prioriteit
- andere pakketten: lage prioriteit

De wachtrij rechts op de figuur is onderdeel van de omleidingsbuffer en voorziet plaats voor omleiding van pakketten die achterblijven tijdens een handover.

Het PrioSched-element maakt eerst de wachtrij met de hoogste prioriteit leeg voordat hij overgaat naar de volgende. Het principe van een PrioSched is grafisch uitgelegd in figuur 2.12.

Het totaalbeeld van de Click-configuratie is te zien in figuur 5.12. Voor de Click code van deze implementatie kan je terecht in A.12.

5.4.3 Resultaten

In deze sectie krijgt niet meer de hele film prioriteit, maar enkel de basislaag (I- en P-frames). De verbeteringslaag heeft een even hoge prioriteit als de rest van het verkeer, zgn. "Best-Effort". Dit om de bediening van het achtergrondverkeer te verbeteren.

De resultaten zijn gegenereerd met achtergrondtrafiek. Enkel het simulatieprogramma voor achtergrondverkeer [**?**] werd gebruikt. De instellingen zijn dezelfde als in 5.3 met QoS. De bandbreedte werd geschaald om congestie op het netwerk te krijgen. De bedrade link werd hiervoor aangepast tot een maximale bitsnelheid van 2000 kbps[4] en de draadloze tot 1000 kbps.

5.4.3.1 Tijdige levering

Het onderscheid tussen verschillende frames zorgt ervoor dat B-frames laattijdig kunnen aankomen. Dit is te zien op figuur 5.13. De B-frames hebben een grotere vertraging omdat ze samen met het achtergrondverkeer worden vervoerd. De I- en P-frames zijn altijd netjes op tijd bij de bestemmeling. Het gevolg is dat de video altijd met behulp van de basislaag kan worden afgespeeld en zich aanpast aan de hoeveelheid B-frames er ontvangen zijn.

Bij het vervoer over de draadloze link komen er duidelijk veel B-frames te laat aan. Bij het vervoer over de bekabelde verbinding zijn ze allemaal op tijd. Dit is beter te zien op figuur 5.14. Als frames een grotere vertraging hebben dan 5s, zijn ze te laat.

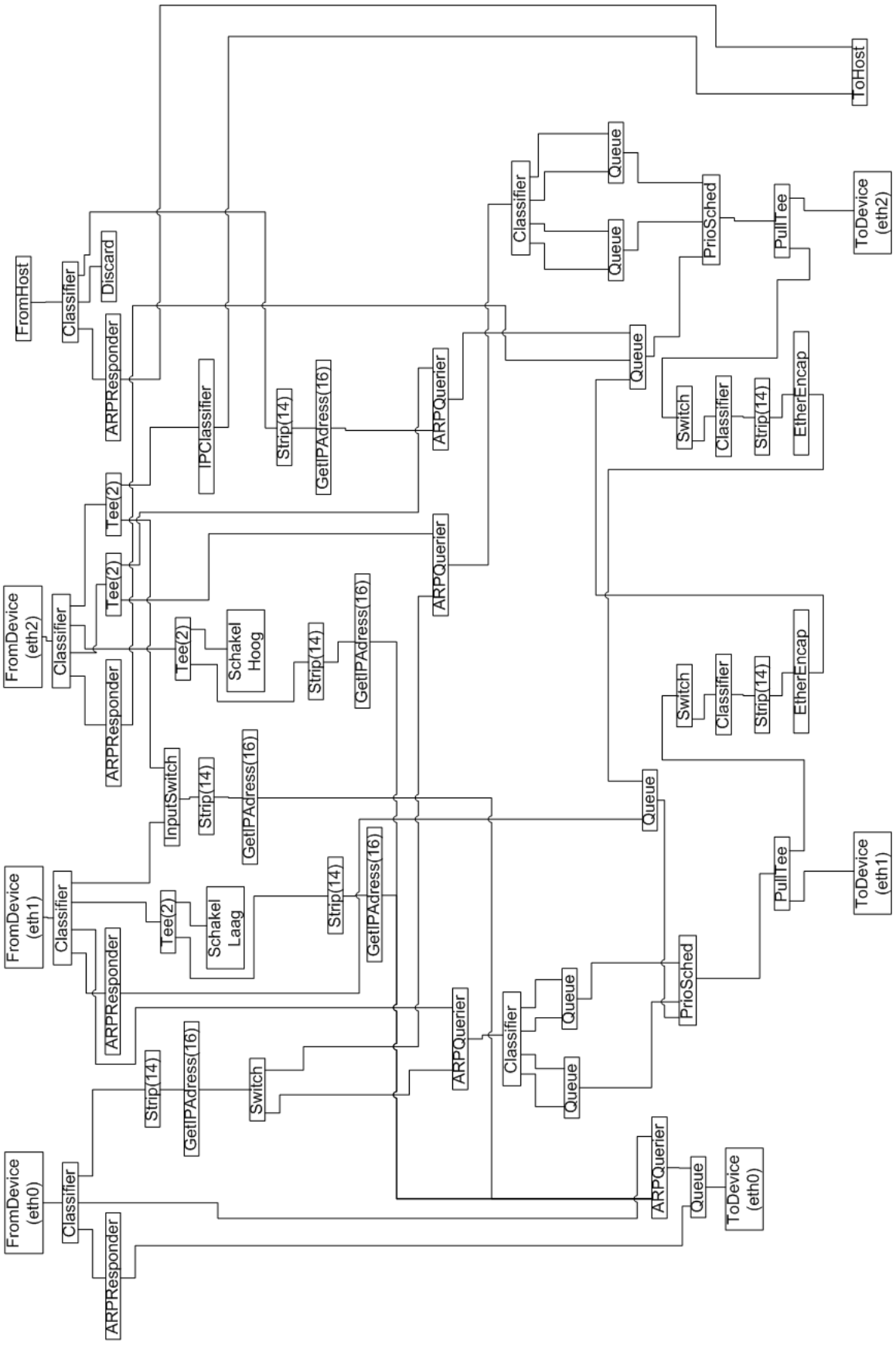
5.4.3.2 Verloren frames

De B-frames krijgen hier geen voorrang op de rest. Daarom gaan er verloren bij congestie. Dit is te zien in tabel 5.3. De I- en P-frames krijgen voorrang en gaan dus niet verloren.

Het achtergrondverkeer krijgt een eerlijkere behandeling dan in 5.3. De data geven aan dat er 4,70% verloren gaan in plaats van 3,46%. Dit is te wijten aan het feit dat de draadloze verbinding langer wordt gebruikt bij de testen van deze implementatie, namelijk 38933,01ms in plaats van 24033,01ms bij de implementatie in 5.3. Tijdens de draadloze verbinding gaan meer pakketten achtergrondverkeer verloren omdat hier de verbindingssnelheid lager ligt en dus ook sneller congestie optreedt. Bij het normaliseren van de duur van de verbinding naar de waarde van de implementatie in 5.3, verliest deze implementatie 2,90% achtergrondverkeer. Dit duidt op de eerlijkere behandeling van deze implementatie.

Er gaan hier weinig pakketten verloren tijdens de handover, zoals weergegeven in tabel 5.3.

[4] kbps is kilobits per seconde



Figuur 5.12: Router uitgebreid met QoS

Tabel 5.3: Verloren frames implementatie 7

	Per type	Totaal
I-frames(%)	0,00	0,00
P-frames(%)	0,00	0,00
B-frames(%)	8,72	5,55
	Duur(ms)	Totaal(♯)
Handover	33,32	3

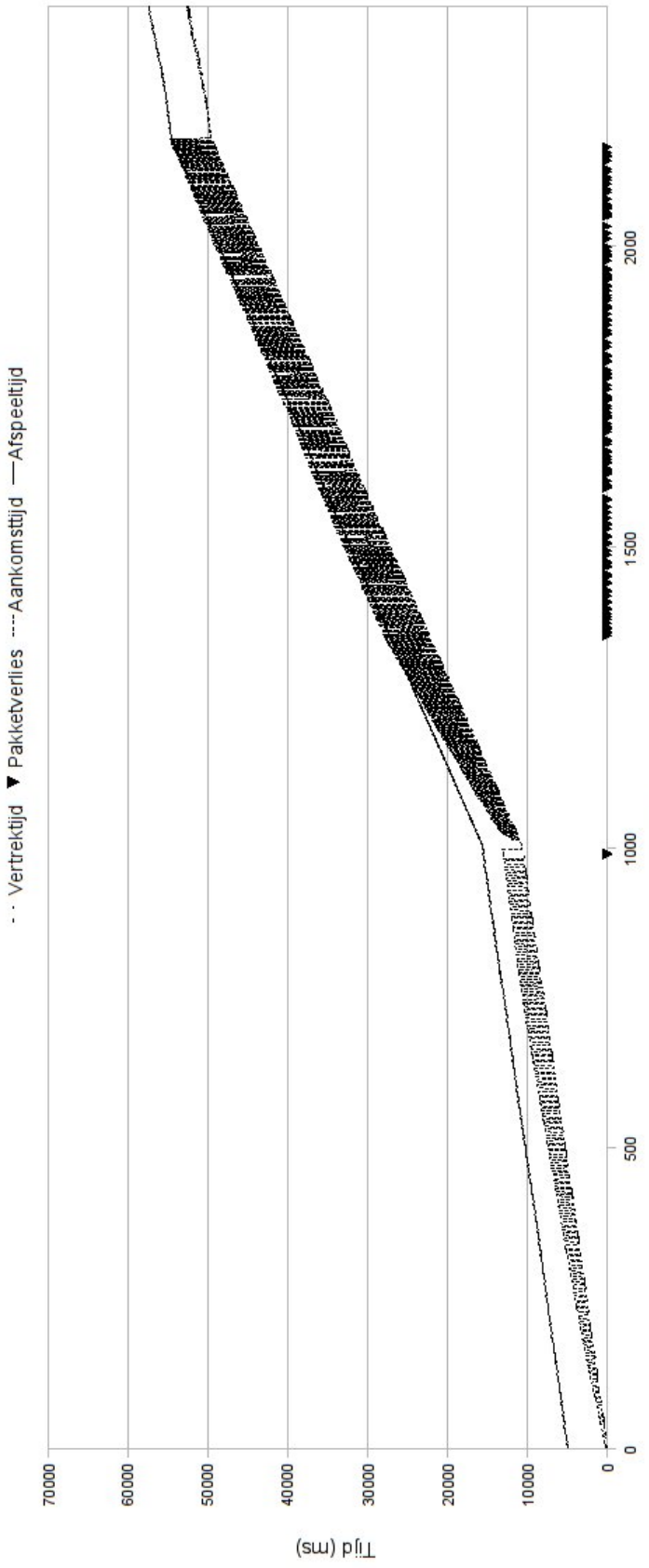
Tabel 5.4: Kenmerken vertraging implementatie 7 met achtergrondverkeer.

	$\overline{x}$ (ms)	s^2	Minimum (ms)	Maximum(ms)
I- en P-frames	82,82	439,11	56,02	270,00
B-frames	2644,03	4471605,41	58,18	5872,67

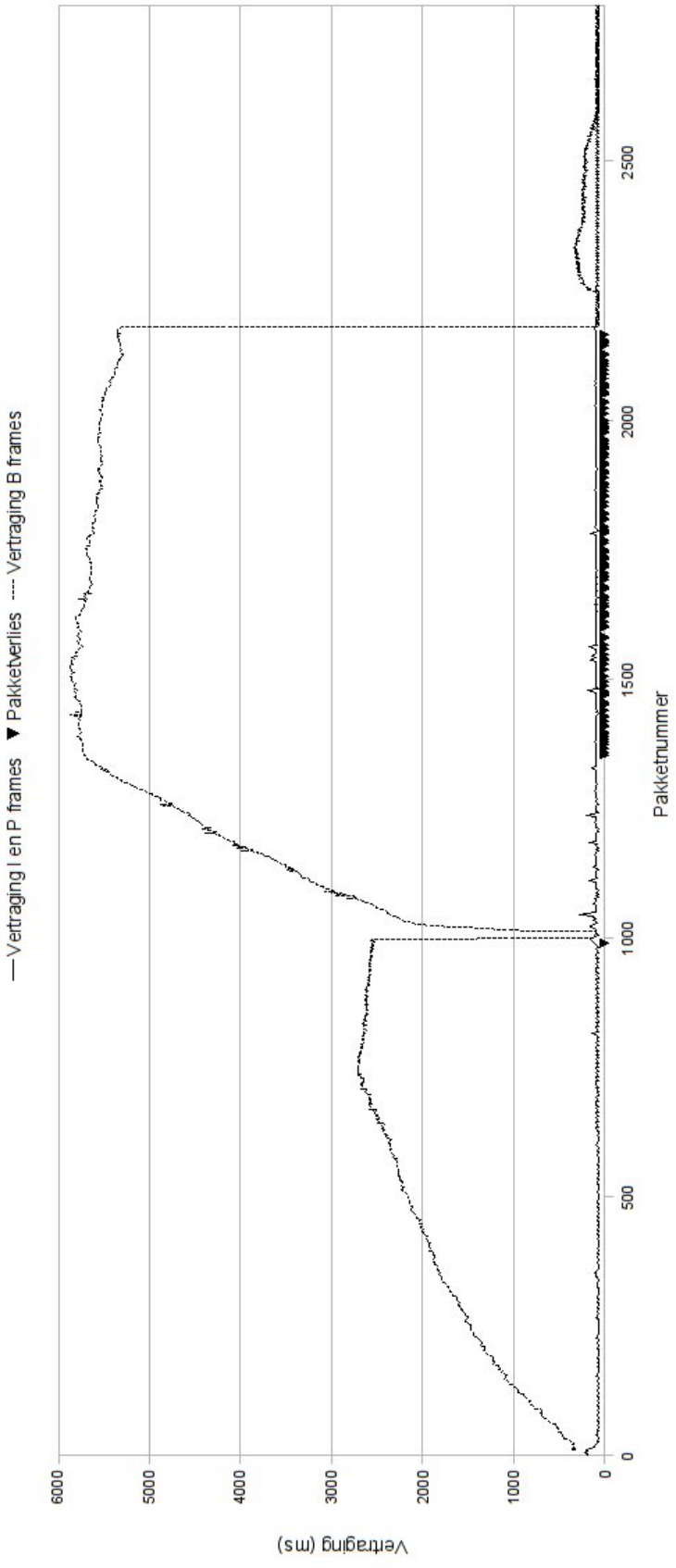
5.4.3.3 Vertraging

Mits de basislaag (I- en P-frames) en de verbeteringslaag (B-frames) in aparte wachtrijen zitten, zijn de vertragingen ook apart uitgezet op figuur 5.14. Hierop is te zien dat de vertraging van de I- en P-frames veel lager is dan deze van de B-frames.

In tabel 5.4 staat dat de variantie van de B-frames gevoelig hoger ligt dan die van de I- en P-frames. De maximum vertraging ligt boven de 5s, wat aangeeft dat sommige B-frames te laat aankomen.



Figuur 5.13: Tijdige levering implementatie 7 met achtergrondtrafiek



Figuur 5.14: Pakketvertraging implementatie 7 met achtergrondverkeer.Kenmerken: I- en P-frames:$\overline{x} = 82,82$ms; $s^2 = 439,11$; Min= $56,02$ms; Max= $270,00$ms; B-frames:$\overline{x} = 2644,03$ms; $s^2 = 4471605,41$; Min= $58,18$ms; Max= $5872,67$ms

5.5 Implementatie 8: Router met gelimiteerde QoS

In deze implementatie wordt de QoS verder uitgewerkt. In de vorige implementaties werd de videostroom niet beperkt. Dit is gevaarlijk omdat de videostroom voorrang krijgt op het gewone verkeer. Er moet dus voor gezorgd worden dat de videostroom niet de volledige bandbreedte benut. Het gebruikte mechanisme fungeert als een soort politieagent en vernietigt alle geprioriseerde videoverkeer dat een vastgezette limiet overschrijdt.

5.5.1 Server

De server blijft voor deze opstelling dezelfde als in 5.1.

5.5.2 Router (Gemeenschappelijke node)

Om ervoor te zorgen dat het videoverkeer het andere verkeer niet blokkeert, is een beperking van de videostroom nodig. Als dit niet gebeurt kan iemand die enkel I-frames stuurt de volledige capaciteit van de router stelen. Omdat de I-frames voorrang krijgen op het andere verkeer bestaat de mogelijkheid dat dit niet meer wordt afgehandeld.

Het bevoorrechte verkeer wordt dus beperkt door een nieuw element aan de router toe te voegen. Dit element is een Meter. De Meter deelt pakketen in naargelang de aankomstfrequentie. De aankomstfrequentie wordt gemeten in pakketten per seconde en gebruikt een gewogen bewegend gemiddelde. Als configuratietekst wordt aan de Meter de aankomstfrequentie meegegeven. Wanneer de aankomstfrequentie groter dan toegelaten is, worden de pakketten op de volgende uitgang uitgestuurd. De pakketten met een te hoge aankomstfrequentie worden in dit geval vernietigd.

De detailfiguur van deze implementatie is te zien in figuur 5.15. De volledige configuratie staat afgebeeld in figuur 5.16. De Click code van deze implementatie vind je in A.13.

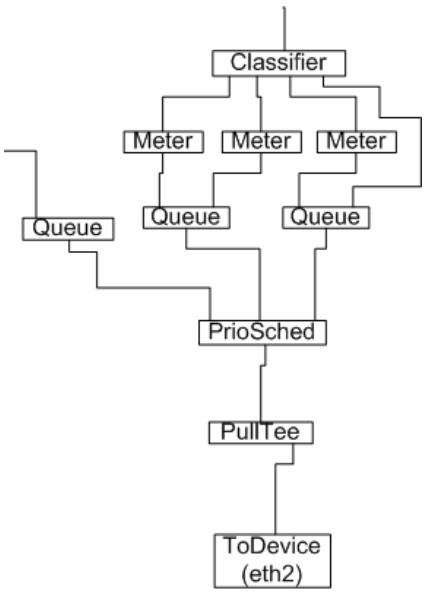
5.5.3 Resultaten

Iemand met verkeerde bedoelingen kan een video doorsturen aan hoge snelheid met enkel I-frames. De wachtrij met I-frames is dan constant bezet en het achtergrondverkeer komt niet meer aan beurt. Om dit tegen te gaan kunnen Meters gezet worden op de wachtrij van de video. Deze controleren of de bevoorrechte wachtrij zijn voorrang niet misbruikt.

Om dit resultaat te genereren werd de videotrace aangepast zodat alle pakketten gemarkeerd worden als I-frames. De snelheid is ook verhoogd naar 60 frames per seconde.

De resultaten zijn gegenereerd met achtergrondtrafiek. Enkel het simulatieprogramma voor achtergrondverkeer [**?**] werd gebruikt. Er werden 75 pakketten per seconde verstuurd door elk van de vier verbindingen. De bandbreedte werd geschaald om congestie op het netwerk te krijgen. De bedrade link werd hiervoor aangepast tot een maximale bitsnelheid van 1000 kbps[5]. Er gebeurde geen handover.

[5] kbps is kilobits per seconde



Figuur 5.15: Detail van het gelimiteerde QoS principe

Het achtergrondverkeer werd in de gaten gehouden. Zonder meters ging er 27,08% van het achtergrondverkeer verloren, met meters 0,00%.

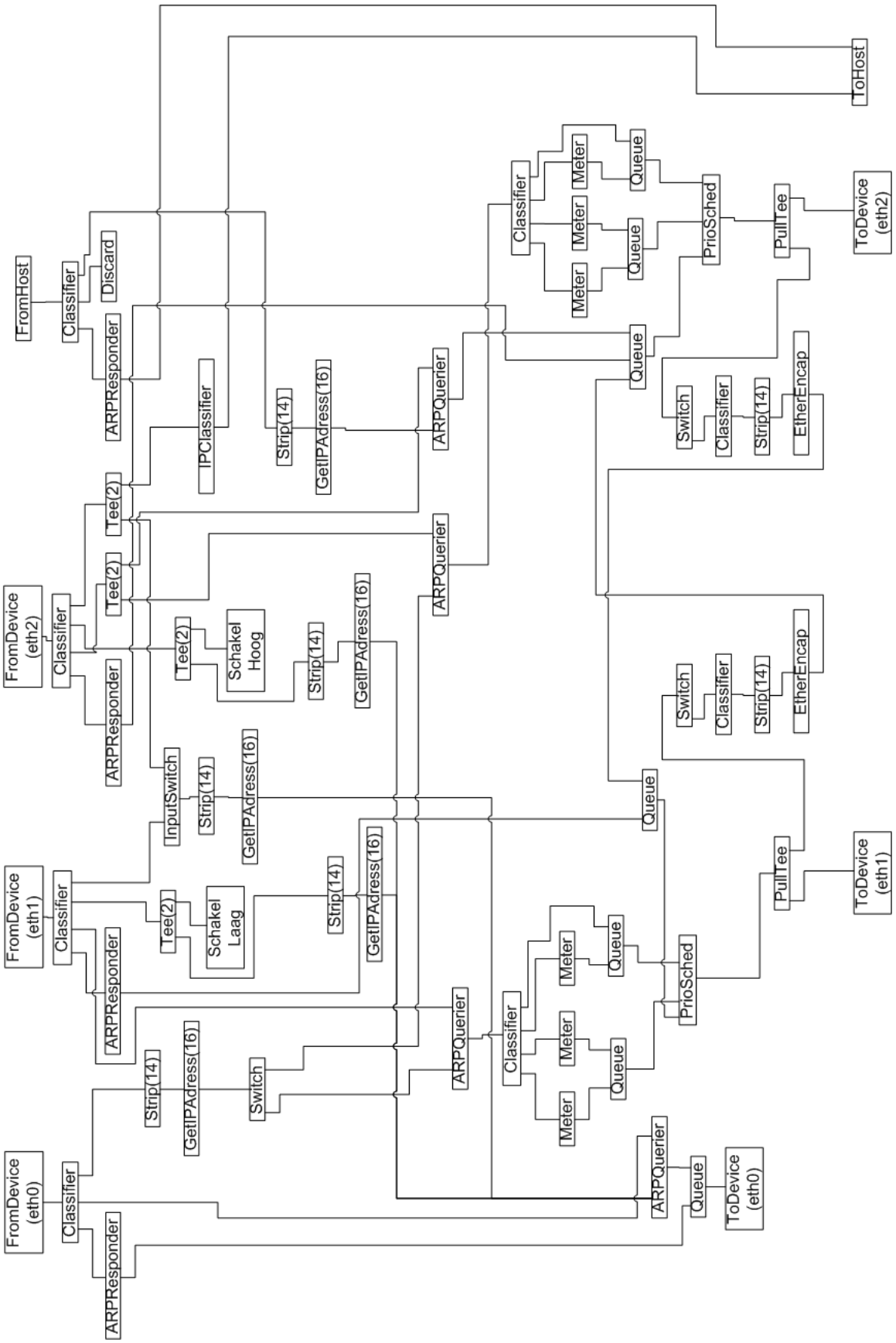
5.6 Besluit

In dit hoofdstuk werd een poging gedaan om het onbetrouwbare UDP-protocol, dat de video vervoert, een grotere zekerheid van tijdige aankomst te bieden. Dit werd bekomen door een verdeling in videoverkeer en in de rest van het verkeer. Een aanpassing in de server werd uitgevoerd om de langdurige effecten van een handover te overbruggen.

Sectie 5.1 behandelt het probleem van de langdurigheid van een handover. Na de handover loopt het verkeer immers over de tragere, draadloze, verbinding. Deze verbinding probeert op een zo goed mogelijke wijze de film samen met de rest van het verkeer te transporteren. Om ervoor te zorgen dat dit beter kan verlopen werd de server intelligenter gemaakt en zal deze bij een verandering naar de tragere link de bitsnelheid van de film aanpassen. Resultaten tonen aan dat bij aanpassing van de film de congestie van het netwerk, die de te hoge bitrate van de film veroorzaakte, volledig verdween. De vertraging werd ook ingeperkt, maar niet zo drastisch als de congestie. Om de variantie op de vertraging omlaag te halen, is QoS aangewezen.

QoS kan op verscheidene manieren worden geïmplementeerd. In deze thesis is gekozen voor verschillende directe aanpakken. Een eerste aanpak werd uitgewerkt in 5.3. Hier wordt de volledige film - dus I-, P- en B-frames - met een hogere prioritering dan het achtergrondverkeer doorgestuurd. Zoals verwacht werd de variatie op de vertraging van de film, zowel als de maximale vertraging drastisch ingeperkt. Het nadeel van deze methode is dat het achtergrondverkeer kan worden uitgehongerd indien het filmverkeer een groot aandeel van de bandbreedte inneemt.

Om de kwestie van uithongering aan te pakken werd in sectie 5.4 niet meer de volledige film



Figuur 5.16: Router uitgebreid met gelimiteerde QoS

geprioriteerd. Omdat het ene type frame van een film belangrijker is als het andere, kan een lagingstechniek worden toegepast. In deze implementie werd gewerkt in twee lagen. De I- en P-frames kregen een hogere prioriteit dan de B-frames. De B-frames werden samen met het achtergrondverkeer doorgestuurd op een “Best-Effort-manier. De resultaten toonden aan dat hier het achtergrondverkeer inderdaad eerlijker werd behandeld. De film verliest hier natuurlijk B-frames. Zij kunnen ook wel eens buiten de tijdslimiet aankomen.

Om de eerlijkheid nog te verhogen, werd in sectie 5.5 de router voorzien van een aantal meters. Deze meters kunnen het totaal aan bevoordeeld videotrafiek beperken om er zo voor te zorgen dat de bandbreedte eerlijk verdeeld wordt tussen de video en de rest van het verkeer. De resultaten wezen dit ook uit.

Hoofdstuk 6

Besluit

In een eerste onderdeel van deze thesis werd getracht een systeem op te zetten om de router en de server op de hoogte te houden van veranderingen in het type van netwerk. Hierbij werd gekozen voor de implementatie van een zachte handover, zoals besproken in 2.2.1.2. Om dit systeem te laten functioneren werd gedacht aan twee mechanismes. Het PING-script en het carrier-script zoals besproken in 4.2. Het carrier-script werd duidelijk de snelste van de twee bevonden. Deze werd dan ook gebruikt in 4.4 om een systeem op te zetten om zo de router, zowel als de server te verwittigen bij de verandering van type verbinding.

In een tweede aanpassing, om de overgang naadloos te laten verlopen, is ervoor gezorgd dat de server zo weinig mogelijk het verkeer moet aanpassen. Daarom werd met behulp van Click een mechanisme voorzien om de bestemmeling van één enkel IP te voorzien in plaats van één IP voor elke interface. Pakketten die reeds onderweg zijn naar de bestemmeling moeten zo ook niet meer gewijzigd worden en bereiken hun bestemming zonder problemen.

Een derde aanpassing om de handover beter te laten verlopen was de omleidingsbuffer, besproken in 4.3. Bij een handover van bijvoorbeeld bedrade naar draadloze verbinding, blijven er pakketten achter in de uitgaande wachtrij van de bedrade interface. De omleidingsbuffer leidt pakketten om naar de juiste wachtrij, in dit geval de uitgaande wachtrij van de draadloze interface. Dit zorgt ervoor dat bij een handover minder pakketten verloren gaan. De “In-flight”-pakketten zijn een verloren zaak, maar de pakketten in de wachtrij kunnen gered worden. Dit kan ervoor zorgen dat belangrijke I-frames niet verloren gaan.

In het tweede onderdeel van deze thesis werden pogingen ondernomen om voor videostreaming in een heterogene netwerkomgeving QoS te voorzien. Er werden hiervoor twee technieken aangewend. Een eerste techniek is het voorrang geven van een videostroom ten opzichte van de rest van het verkeer. De tweede gebruikte techniek is een vorm van intelligentie geven aan de videobron, zodat deze weet over welke soort verbinding het videoverkeer loopt.

Hoofdstuk 2 behandelt de verschillende mogelijke aanpakken. Hier werd besloten om gebruik te maken van een temporeel schaalbare codering zoals besproken in 2.2.2.3. De film wordt hierbij opgedeeld in meerdere lagen. In deze thesis werd de film opgedeeld in twee lagen, de zgn. basislaag (I- en P-frames) en een verbeteringslaag (B-frames). Het idee hierbij is dat de basislaag van een vaste QoS wordt voorzien, terwijl de verbeteringslaag op een “Best-Effort-manier wordt doorgestuurd. Zo werd het mogelijk verschillende niveau’s van prioritering toe te kennen aan

verschillende delen van de film. Deze techniek zorgt ervoor dat basislaag van de film weinig tot geen invloed ondervinden aan wisselende verkeersomstandigheden op het netwerk die momenten van congestie en ontoelaatbare vertraging veroorzaken.

De tweede techniek voorziet een intelligente server. Bij een overgang naar een ander type van verbinding, met eventueel een andere snelheid, is het mogelijk dat de link al het video verkeer niet kan verwerken. Om dit probleem aan te pakken werd een systeem ontwikkeld dat de server van de nodige informatie voorziet om de bitsnelheid van de video aan te passen naargelang de maximale snelheid. In deze thesis werd gewerkt met twee types van verbinding en voorziet de server dus ook twee verschillende bitsnelheden. Een mechanisme werd ontwikkeld dat de server via een pakket op de hoogte brengt van de verandering van verbinding. De server neemt dan de nodige maatregelen om een aangepaste bitsnelheid door te sturen. Er bestaan verschillende mogelijkheden om de nodige maatregelen uit te voeren. In deze thesis werd geopteerd om de server te voorzien van meerdere versies van eenzelfde videostroom aan verschillende bitsnelheden zoals besproken in 2.2.2.1.

Zoals besproken in 2.2.3 werd een stroom gegenereerd in plaats van een echte video door te sturen. Deze maken gebruik van zgn. video traces. In deze stroom werd informatie over de pakketten zelf en hun bedoelde inhoud gestoken zodat ze goed konden worden opgevolgd. Het voordeel van een gegenereerde stroom is dat de werkelijke informatie van de verschillende frames niet is gegeven. Wel gegeven is het aantal bits na codering die worden gebruikt door de verschillende frames. Er is hier dus geen sprake van auteursrechten. De grootte van een trace is ook veel kleiner, typisch enkele megabytes. Om de stroom te genereren werd een trace gebruikt van de video “Baseball with commercials”, gevonden in [**?**].

Het doel van de thesis was om een echte testopstelling te maken zodat onderzocht kan worden wat de effecten van een handover en van voorziening van QoS zijn op een videostroom. Er is al veel onderzoek verricht naar handovers en QoS. Deze onderzoeken baseerden zich meestal op simulatieprogramma’s. Ook werden deze twee effecten nooit samen bekeken. Deze twee effecten zijn in de thesis bekeken in een echte opstelling. In het begin is er gebruik gemaakt van een echte videostroom (dus geen gegenereerde stroom), zodat het effect op de video zichtbaar werd. In het algemeen is te concluderen dat het zeker de moeite is om handovers en voorziening van QoS te onderzoeken bij het streamen van video. De gebruikservaring zal drastisch verbeteren bij een juiste aanpak van de handover en voorziening van QoS.

Voor verder onderzoek kan er alvast de volgende suggestie worden gegeven. In deze thesis werd gebruik gemaakt van een schaalbare techniek met groffe korrel, nl. een temporeel schaalbare codec. Technieken met fijne korrel, zoals MPEG4-FGS, zijn aangeraden voor verder onderzoek. Dit aangezien het kwaliteitsverschil minder zichtbaar is dan bij technieken met grove korrel. Het missen van enkele bits heeft namelijk bij grove korrel het effect dat een volledige frame verloren gaat. Dit verloren frame heeft een niet te verwaarlozen effect op de afgespeelde kwaliteit. Bij fijne korrel heeft dit een veel kleiner effect op de kwaliteit aangezien hier de kwaliteit verbetert naargelang meer informatie aankomt.

Ook het verder onderzoeken van de mogelijkheden van de verschillende protocollen zoals RTP en TCP is een waardevolle uitbreidig aan deze thesis.

Om de opstelling meer bij de realiteit te laten aanleunen, is een uitbreiding van de router tot meerdere clienten aan te raden.

Bijlage A

Appendix

A.1 Click code van de simpele router

```
FromDump(f3a.dump, STOP true)
     -> CheckIPHeader
     -> i1 :: IPClassifier(tcp, udp, icmp, -)
     -> CheckTCPHeader
     -> ttl :: IPClassifier(ttl > 0, -)
     -> cl :: CheckLength(1500)
     -> ip :: IPClassifier(dst 131.179.0.0/16, dst 131.0.0.0/8, dst 18.0.0.0/8, -)
  i1[1] -> CheckUDPHeader -> ttl
  i1[2] -> CheckICMPHeader -> ttl
  i1[3] -> ttl
  ttl[1] -> ICMPError(18.26.7.1, timeexceeded, transit)
     -> ToDump(f3f.dump, ENCAP IP)
  ip[0] ->  ToDump(f3c.dump, ENCAP IP)
  ip[1] ->  ToDump(f3b.dump, ENCAP IP)
  ip[2] ->  ToDump(f3d.dump, ENCAP IP)
  ip[3] ->  ToDump(f3e.dump, ENCAP IP)
```

A.2 Click code van implementatie 1

```
c0 :: Classifier(12/0806 20/0001,   //verbinding met server
                 12/0806 20/0002,
                 12/0800,
                 -);
c1 :: Classifier(12/0806 20/0001,  //bedrade verbinding met laptop
                 12/0806 20/0002,
                 12/0800,
                 -);

//Neem pakketten van eth0 en eth1 en classificeer ze.
FromDevice(eth0)->c0;
FromDevice(eth1)->c1;
```

```
//Is het IP adres in dit pakket dat van mij? Zo ja, stuur een ARP boodschap
//met mijn MAC naar de vrager.
ar0 :: ARPResponder(10.0.0.122 00:0F:1F:8A:06:48);
ar1 :: ARPResponder(192.168.1.100 00:11:6B:31:15:4D);

//de outputs definieren we hebben queues nodig voor de push pull conventie
out0 :: Queue->ToDevice(eth0);
out1 :: Queue->ToDevice(eth1);

//arp queries go to the arp responder en dan naar de queueu
c0[0] -> ar0 -> out0;
c1[0] -> ar1 -> out1;

//Definieren ARP querier. Deze vraagt om het MAC adres van een bepaald
//IP als het dit niet kent.
arpq0 :: ARPQuerier(10.0.0.122,00:0F:1F:8A:06:48);
arpq1 :: ARPQuerier(192.168.1.100,00:11:6B:31:15:4D);

//arp responses gaan naar de arpquerier omdat die responses verwacht
//op zijn queries
c0[1] -> [1]arpq0;
c1[1] -> [1]arpq1;

//zet de ip naar de juiste arpquerier dit is gekruisd we routen niet
//getipaddress sets the annotation for the next hop
//het verandert de standaard headers niet
c0[2] -> Strip(14) -> GetIPAddress(16) -> arpq1;
c1[2] -> Strip(14) -> GetIPAddress(16) -> arpq0;


//uitgang querier naar output sturen
arpq0->out0;
arpq1->out1;

c0[3]->Discard;
c1[3]->Discard;
```

A.3 Click code van implementatie 2: Gemeenschappelijke node

```
c0 :: Classifier(12/0806 20/0001,   //verbinding met server
                 12/0806 20/0002,
                 12/0800);
c1 :: Classifier(12/0806 20/0001,  //draadloze verbinding met laptop
                 12/0806 20/0002,
                 12/0800);

c2 :: Classifier(12/0806 20/0001,  //bedrade verbinding met laptop
                 12/0806 20/0002,
```

```
                12/0800);


//Neem pakketten van eth0, eth1 en eth 2 en classificeer ze.
FromDevice(eth0)->c0;
FromDevice(eth1)->c1;
FromDevice(eth2)->c2;

//Is het IP adres in dit pakket dat van mij? Zo ja, stuur een ARP boodschap
//met mijn MAC naar de vrager.
ar0 :: ARPResponder(10.0.0.122 00:0F:1F:8A:06:48);
ar1 :: ARPResponder(192.168.1.100 00:11:6B:31:15:4D);
ar2 :: ARPResponder(192.168.0.1 00:11:6B:31:11:9C);

//de outputs definieren we hebben queues nodig voor de push pull conventie
out0 :: Queue->ToDevice(eth0);
out1 :: Queue->ToDevice(eth1);
out2 :: Queue->ToDevice(eth2);

//ARP queries worden behandeld door de responder en verdergestuurd naar de uitgang.
c0[0] -> ar0 -> out0;
c1[0] -> ar1 -> out1;
c2[0] -> ar2 -> out2;

//Definieren ARP querier. Deze vraagt om het MAC adres van een bepaald IP als het
//dit niet kent.
arpq0 :: ARPQuerier(10.0.0.122,00:0F:1F:8A:06:48);
arpq1 :: ARPQuerier(192.168.1.100,00:11:6B:31:15:4D);
arpq2 :: ARPQuerier(192.168.0.1,00:11:6B:31:11:9C);

//ARP responses gaan naar de ARP querier omdat die responses verwacht op zijn queries.
c0[1] -> [1]arpq0;
c1[1] -> [1]arpq1;
c2[1] -> [1]arpq2;

/*Dit zijn de standaard IP pakketten. Het IP moet in het 'annotation field' worden
gezet om functionele redenen van de ARP querier. Getipaddress sets the annotation for
the next hop. Het verandert de standaard headers niet. Voeg hier ook de inputswitch en
switch toe*/
verdeler0 :: Switch(0); // Default (0) uitgang is naar de draadloze verbinding.
combinator0 :: InputSwitch(0); // Default (0) ingang is van de draadloze verbinding.

c0[2] -> Strip(14) -> GetIPAddress(16) -> verdeler0;
verdeler0[0] -> arpq1;
verdeler0[1] -> arpq2;
c1[2] -> [0]combinator0;
c2[2] -> [1]combinator0;
combinator0 -> Strip(14) -> GetIPAddress(16) -> arpq0;

//uitgang querier naar output sturen
arpq0->out0;
arpq1->out1;
arpq2-> out2;
```

A.4 Click code van implementatie 2: Client

```
//Kiezen welke input en doorsturen naar classifier.
AddressInfo(wirelessInterface 192.168.1.2 00:13:F7:3B:BE:E9);
AddressInfo(wiredInterface 192.168.0.2 00:15:C5:CD:BF:D6);

combinator :: InputSwitch(1);
c0 :: Classifier(12/0806 20/0001,
                 12/0806 20/0002,
                 12/0800,
                 -
                 );

fromhost_cl :: Classifier(12/0806 20/0001,
                          12/0806 20/0002,
                          12/0800,
                          -
                          );

FromDevice(ath1) -> Print() -> [0]combinator;
FromDevice(eth1) -> [1]combinator;
combinator -> c0;
FromHost(fakedev, 192.168.3.1/32) -> fromhost_cl;

ar2 :: ARPResponder(0.0.0.0/0 1:1:1:1:1:1);
ar3 :: ARPResponder(192.168.3.1 00:13:F7:3B:BE:E9); //wireless
ar4 :: ARPResponder(192.168.3.1 00:15:C5:CD:BF:D6); //wire

verdelerARP2 :: Switch(1);

out0 :: Queue -> ToDevice(ath1);
out1 :: Queue -> ToDevice(eth1);

c0 -> verdelerARP2 -> ar3 -> out0;

verdelerARP2[1] -> ar4 -> out1;

fromhost_cl[0] -> ar2 -> ToHost;

//Opvangen ARP responses
arpq0 :: ARPQuerier(192.168.1.2,00:13:F7:3B:BE:E9);
arpq1 :: ARPQuerier(192.168.0.2,00:15:C5:CD:BF:D6);

verdelerARPQ :: Switch(1);

fromhost_cl[1] -> Discard;
c0[1] -> verdelerARPQ;
verdelerARPQ[0] -> [1]arpq0;
verdelerARPQ[1] -> [1]arpq1;

//Verdelen standaard IP pakketten
verdelerROUTEtoDev :: Switch(1);
```

```
rt3 :: LinearIPLookup(192.168.3.1 0,
                      0/0 192.168.0.1 0); //for wired

rt2 :: LinearIPLookup(192.168.3.1 0, //for wireless
                      0/0 192.168.1.100 0);

c0[2] -> ToHost();

fromhost_cl[2] -> Strip(14) -> GetIPAddress(16) -> [0]verdelerROUTEtoDev

verdelerROUTEtoDev[0] -> rt2;
verdelerROUTEtoDev[1] -> rt3;

rt2 -> [0]arpq0;
rt3 -> [0]arpq1;

arpq0 -> out0;
arpq1 -> out1;

c0[3] -> Discard;
fromhost_cl[3]-> Discard;
```

A.5 Click code van implementatie 3: Gemeenschappelijke node

```
c0 :: Classifier(12/0806 20/0001,   //verbinding met server
                 12/0806 20/0002,
                 12/0800,
                 -);
c1 :: Classifier(12/0806 20/0001,  //draadloze verbinding met laptop
                 12/0806 20/0002,
                 12/0800,
                 -);

c2 :: Classifier(12/0806 20/0001,  //bedrade verbinding met laptop
                 12/0806 20/0002,
                 12/0800,
                 -);
fromhostcl :: Classifier(12/0806 20/0001,  //bedrade verbinding met laptop
                 12/0806 20/0002,
                 12/0800,
                 -);

//Neem pakketten van eth0, eth1 en eth 2 en classificeer ze.
FromDevice(eth0)->c0;
FromDevice(eth1)->c1;
FromDevice(eth2)->c2;
FromHost(fake, 192.168.4.1/32) -> fromhostcl;
```

```
//Is het IP adres in dit pakket dat van mij? Zo ja, stuur een ARP boodschap
//met mijn MAC naar de vrager.
ar0 :: ARPResponder(10.0.0.122 00:0F:1F:8A:06:48);
ar1 :: ARPResponder(192.168.1.100 00:11:6B:31:15:4D);
ar2 :: ARPResponder(192.168.0.1 00:11:6B:31:11:9C);
ar3 :: ARPResponder(0.0.0.0/0 1:1:1:1:1:1);

//de outputs definieren we hebben queues nodig voor de push pull conventie
out0 :: Queue->ToDevice(eth0);
out1 :: Queue->ToDevice(eth1);
out2 :: Queue->ToDevice(eth2);

//ARP queries worden behandeld door de responder en verdergestuurd naar de uitgang.
c0[0] -> ar0 -> out0;
c1[0] -> ar1 -> out1;
c2[0] -> ar2 -> out2;
fromhostcl -> ar3 -> ToHost();
//Definieren ARP querier. Deze vraagt om het MAC adres van een bepaald IP
//als het dit niet kent.
arpq0 :: ARPQuerier(10.0.0.122,00:0F:1F:8A:06:48);
arpq1 :: ARPQuerier(192.168.1.100,00:11:6B:31:15:4D);
arpq2 :: ARPQuerier(192.168.0.1,00:11:6B:31:11:9C);
arpq3 :: ARPQuerier(192.168.0.1,00:11:6B:31:11:9C);

//ARP responses gaan naar de ARP querier omdat die responses verwacht op zijn queries.
c0[1] -> [1]arpq0;
c1[1] -> [1]arpq1;
c2[1] -> tak :: Tee(2) -> [1]arpq2;
fromhostcl[1] -> Discard;

/*Dit zijn de standaard IP pakketten. Het IP moet in het 'annotation field'
worden gezet om functionele redenen van de ARP querier. Getipaddress sets the
annotation for the next hop. Het verandert de standaard headers niet.
Voeg hier ook de inputswitch en switch toe*/
verdeler0 :: Switch(0); // Default (0) uitgang is naar de draadloze verbinding.
combinator0 :: InputSwitch(0); // Default (0) ingang is van de draadloze verbinding.
//UDPCheck :: SetUDPChecksum();
//UDPCheck -> Discard;

c0[2] -> Strip(14) -> verdeler0;
verdeler0[0] -> GetIPAddress(16) -> arpq1;
verdeler0[1] -> GetIPAddress(16) -> arpq2;
-> StoreIPAddress(16)-> UDPCheck[0] -> arpq2;
c1[2] -> Strip(14) -> [0]combinator0;
c2[2] -> tak2 :: Tee(2) -> Strip(14) -> [1]combinator0;
tak2[1] -> ToHost();
combinator0 -> GetIPAddress(16) -> arpq0;
fromhostcl[2] -> Strip(14) -> GetIPAddress(16) -> arpq3;
tak[1] -> [1]arpq3;

//uitgang querier naar output sturen
arpq0->out0;
```

```
arpq1->out1;
arpq2->out2;
arpq3->out2;

c0[3]->Discard;
c1[3]->Discard;
c2[3]->Discard;
fromhostcl[3]->Discard;
```

A.6 Click code van implementatie 4: Gemeenschappelijke node

```
c0 :: Classifier(12/0806 20/0001,   //verbinding met server
                 12/0806 20/0002,
                 12/0800,
                 -);
c1 :: Classifier(12/0806 20/0001,  //draadloze verbinding met laptop
                 12/0806 20/0002,
                 12/0800,
                 -);
c2 :: Classifier(12/0806 20/0001,  //bedrade verbinding met laptop
                 12/0806 20/0002,
                 12/0800,
                 42/737769746368,
  -);

//c2::Tee(5);
fromhostcl :: Classifier(12/0806 20/0001,  //bedrade verbinding met laptop
                 12/0806 20/0002,
                 12/0800,
                 -);

//Neem pakketten van eth0, eth1 en eth 2 en classificeer ze.
FromDevice(eth0)->c0;
FromDevice(eth1)->c1;
FromDevice(eth2)->c2;
FromHost(fake, 192.168.4.1/32) -> fromhostcl;

//Is het IP adres in dit pakket dat van mij?
//Zo ja, stuur een ARP boodschap met mijn MAC naar de vrager.
ar0 :: ARPResponder(192.168.6.2 00:0F:1F:8A:06:48);
ar1 :: ARPResponder(192.168.1.100 00:11:6B:31:15:4D);
ar2 :: ARPResponder(192.168.0.1 00:11:6B:31:11:9C);
ar3 :: ARPResponder(0.0.0.0/0 1:1:1:1:1:1);

puller1::PullTee(2);
puller2::PullTee(2);
prio1::PrioSched;
prio2::PrioSched;
//de outputs definieren we hebben queues nodig voor de push pull conventie
```

```
out0 :: Queue(100)->ToDevice(eth0);
out1 :: Queue(10)->[1]prio1->puller1->ToDevice(eth1);
out2 :: Queue(10)->[1]prio2->puller2->ToDevice(eth2);


pulclas1::IPClassifier(dst udp port 1234,-);
pulclas2::IPClassifier(dst udp port 1234,-);
puller1[1]->suppres1::Switch(1)[1]->pulclas1->Strip(14)->EtherEncap(0x0800,
00:11:6B:31:11:9C, 00:15:c5:cd:bf:d6)->Queue->test2::Counter()->prio2;
puller2[1]->suppres2::Switch(0)[1]->pulclas2->Strip(14)->EtherEncap(0x0800,
00:11:6B:31:15:4D, 00:13:f7:3b:be:e9)->Queue->test1::Counter()->prio1;
suppres1[0]->Discard;
suppres2[0]->Discard;
//ARP queries worden behandeld door de responder en verdergestuurd naar de uitgang.
c0[0] -> ar0 -> out0;
c1[0] -> ar1 -> out1;
c2[0] -> ar2 -> out2;
fromhostcl -> ar3 -> ToHost();
//Definieren ARP querier.
//Deze vraagt om het MAC adres van een bepaald IP als het dit niet kent.
arpq0 :: ARPQuerier(192.168.6.2,00:0F:1F:8A:06:48);
arpq1 :: ARPQuerier(192.168.1.100,00:11:6B:31:15:4D);
arpq2 :: ARPQuerier(192.168.0.1,00:11:6B:31:11:9C);
arpq3 :: ARPQuerier(192.168.0.1,00:11:6B:31:11:9C);

//ARP responses gaan naar de ARP querier omdat die responses verwacht op zijn queries.
c0[1] -> [1]arpq0;
c1[1] -> [1]arpq1;
c2[1] -> tak :: Tee(2) -> [1]arpq2;
fromhostcl[1] -> Discard;

/*Dit zijn de standaard IP pakketten. Het IP moet in het 'annotation field' worden gezet
om functionele redenen van de ARP querier. Getipaddress sets the annotation for the next
hop. Het verandert de standaard headers niet. Voeg hier ook de inputswitch en switch toe*/
verdeler0 :: Switch(1); // Default (0) uitgang is naar de draadloze verbinding.
combinator0 :: InputSwitch(1); // Default (0) ingang is van de draadloze verbinding.

c0[2] -> Strip(14) -> verdeler0;
verdeler0[0] -> GetIPAddress(16) -> arpq1;
verdeler0[1] -> GetIPAddress(16) -> arpq2;
c1[2] -> Strip(14) -> [0]combinator0;
c2[2] -> tak2 :: Tee(2) -> Strip(14) -> [1]combinator0;
tak2[1] -> ToHost();
combinator0 -> GetIPAddress(16) -> arpq0;
fromhostcl[2] -> Strip(14) -> GetIPAddress(16) -> arpq3;
tak[1] -> [1]arpq3;

//uitgang querier naar output sturen
arpq0->out0;
arpq1->out1;
arpq2->out2;
arpq3->out2;
```

```
c2[4]->Discard;
c2[3]->Counter()->ToHost();

c0[3]->Discard;
c1[3]->Discard;
fromhostcl[3]->Discard;
pulclas1[1]->Discard;
pulclas2[1]->Discard;
```

A.7 Click code van implementatie 5: Server

```
FromDevice(eth0)-> class:: Classifier(42/6c616167,
42/686f6f67) -> Schakel(QUALITY "laag");
class[1] -> Schakel(QUALITY "hoog");

output::InputSwitch(1)->UDPIPEncap(192.168.6.1,1234,
192.168.3.1,1234)->EtherEncap(0x800, 00:0B:DB:C6:02:6C,
00:0F:1F:8A:06:48)->Queue()->IPPrint(inhoud, PAYLOAD ASCII)->ToDevice(eth0);

FakeStream("Baseball_303030")->[0]output; // laag
FakeStream("Baseball_040404")->[1]output; // hoog
```

A.8 Click code van implementatie 5: Achtergrond Client

```
//Click configuratie van achtergrond client
c0 :: Classifier( 12/0806 20/0001,
12/0806 20/0002,
42/6c616167, //classifier voor het lage pakket over de draadloze verbinding
12/0800,
-
);

c1 :: Classifier( 12/0806 20/0001,
12/0806 20/0002,
42/686f6f67, //classifier voor het hoge pakket over de bedrade verbinding
12/0800,
-
);
fromhost_cl :: Classifier(12/0806 20/0001,
  12/0806 20/0002,
  12/0800,-);

FromDevice(ath0) -> c0;
FromDevice(eth0) -> c1;
FromHost(fakedev, 192.168.5.1/32) -> fromhost_cl;

ar2 :: ARPResponder(0.0.0.0/0 1:1:1:1:1:1);
ar3 :: ARPResponder(192.168.5.1 00:13:F7:83:66:4E); //wireless
ar4 :: ARPResponder(192.168.5.1 00:0D:56:CE:63:4E); //wire
```

```
out0 :: Counter()->Queue -> ToDevice(ath0);
out1 :: Counter()->Queue -> ToDevice(eth0);

c0->ar3->out0;
c1->ar4->out1;

ToLinux::ToHost();
fromhost_cl[0] -> ar2 -> ToLinux;

//Opvangen ARP responses
arpq0 :: ARPQuerier(192.168.1.6,00:13:F7:83:66:4E);
arpq1 :: ARPQuerier(192.168.0.6,00:0D:56:CE:63:4E);

fromhost_cl[1] -> Discard;
c0[1] ->[1]arpq0;
c1[1] ->[1]arpq1;

//Verdelen standaard IP pakketten
verdelerROUTEtoDev :: Switch(1);

rt3 :: LinearIPLookup(192.168.5.1 0,
                      0/0 192.168.0.1 0); //for wired

rt2 :: LinearIPLookup(192.168.5.1 0, //for wireless
                      0/0 192.168.1.100 0);

c0[3] -> ToLinux;
c1[3] ->ToLinux;

fromhost_cl[2] -> Strip(14) -> GetIPAddress(16) -> [0]verdelerROUTEtoDev

verdelerROUTEtoDev[0] -> rt2;
verdelerROUTEtoDev[1] -> rt3;

rt2 -> [0]arpq0;
rt3 -> [0]arpq1;

arpq0 -> out0;
arpq1 -> out1;

c0[4] -> Discard;
c1[4] -> Discard;

c0[2]->lage::Counter()->Schakel(QUALITY "laag");
c1[2]->hoge::Counter()->Schakel(QUALITY "hoog");
fromhost_cl[3]-> Discard;
```

A.9 Click code van implementatie 5: Gemeenschappelijke node

```
c0 :: Classifier(12/0806 20/0001,   //verbinding met server
                 12/0806 20/0002,
                 12/0800,
                 -);
c1 :: Classifier(12/0806 20/0001,  //draadloze verbinding met laptop
                 12/0806 20/0002,
                 42/6c616167,     //lage kwaliteit
                 12/0800,
                 -);
c2 :: Classifier(12/0806 20/0001,  //bedrade verbinding met laptop
                12/0806 20/0002,
                42/686f6f67,     //hoge kwaliteit
                12/0800,
                -);

fromhostcl :: Classifier(12/0806 20/0001,  //bedrade verbinding met laptop
                 12/0806 20/0002,
                 12/0800,
                 -);

//Neem pakketten van eth0, eth1 en eth 2 en classificeer ze.
FromDevice(eth0)->c0;
FromDevice(eth1)->c1;
FromDevice(eth2)->c2;
FromHost(fake, 192.168.4.1/32) -> fromhostcl;

//Is het IP adres in dit pakket dat van mij? Zo ja, stuur een ARP boodschap
//met mijn MAC naar de vrager.
ar0 :: ARPResponder(192.168.6.2 00:0F:1F:8A:06:48); //zonder switch
ar1 :: ARPResponder(192.168.1.100 00:11:6B:31:15:4D);
ar2 :: ARPResponder(192.168.0.1 00:11:6B:31:11:9C);
ar3 :: ARPResponder(0.0.0.0/0 1:1:1:1:1:1);

//definities voor pakketen om te leiden
puller1::PullTee(2);
puller2::PullTee(2);
prio1::PrioSched;
prio2::PrioSched;
//de outputs definieren we hebben queues nodig voor de push pull conventie
out0 :: Queue->ToDevice(eth0);
out1 :: Queue->[1]prio1->puller1->ToDevice(eth1);
out2 :: Queue->[1]prio2->puller2->ToDevice(eth2);


pulclas1::IPClassifier(udp,-);
pulclas2::IPClassifier(udp,-);
puller1[1]->suppres1::Switch(1)[1]->pulclas1->Strip(14)->
EtherEncap(0x0800, 00:11:6B:31:11:9C, 00:15:c5:cd:bf:d6)->Queue->prio2;
puller2[1]->suppres2::Switch(0)[1]->pulclas2->Strip(14)->
```

```
EtherEncap(0x0800, 00:11:6B:31:11:9C, 00:13:f7:3b:be:e9)->Queue->prio1;
suppres1[0]->Discard;
suppres2[0]->Discard;
//ARP queries worden behandeld door de responder en verdergestuurd naar de uitgang.
c0[0] -> ar0 -> out0;
c1[0] -> ar1 -> out1;
c2[0] -> ar2 -> out2;
fromhostcl -> ar3 -> ToHost();

//Definieren ARP querier. Deze vraagt om het MAC adres van een bepaald IP
//als het dit niet kent.
arpq0 :: ARPQuerier(192.168.6.2,00:0F:1F:8A:06:48);//zonder switch
arpq1 :: ARPQuerier(192.168.1.100,00:11:6B:31:15:4D);
arpq2 :: ARPQuerier(192.168.0.1,00:11:6B:31:11:9C);
arpq3 :: ARPQuerier(192.168.0.1,00:11:6B:31:11:9C);

//ARP responses gaan naar de ARP querier omdat die responses verwacht op zijn queries.
c0[1] -> [1]arpq0;
c1[1] -> [1]arpq1;
c2[1] -> tak :: Tee(2) -> [1]arpq2;
fromhostcl[1] -> Discard;

/*Dit zijn de standaard IP pakketten. Het IP moet in het 'annotation field' worden
gezet om functionele redenen van de ARP querier. Getipaddress sets the annotation
for the next hop. Het verandert de standaard headers niet.
Voeg hier ook de inputswitch en switch toe*/
verdeler0 :: Switch(1); // Default (0) uitgang is naar de draadloze verbinding.
combinator0 :: InputSwitch(1); // Default (0) ingang is van de draadloze verbinding.

c0[2] -> tak6 :: Tee(2) -> Strip(14) -> GetIPAddress(16) -> verdeler0;
verdeler0[0] -> arpq1;//GetIPAddress(16)
verdeler0[1] -> arpq2;
c1[3] -> [0]combinator0;//Strip(14) ->
c2[3] -> [1]combinator0;//Strip(14) ->
combinator0 -> Strip(14) -> GetIPAddress(16) -> arpq0;
fromhostcl[2] -> Strip(14) -> GetIPAddress(16) -> arpq3;
tak[1] -> [1]arpq3;

//ff wa tellen
tak6[1] -> IPClassifier(udp) -> udptellerke :: Counter() -> Discard;

//uitgang querier naar output sturen
arpq0->out0;
arpq1->tellertest2::Counter()->out1;
arpq2->out2;
arpq3->out2;

c2[2]-> tellerWire::Counter() -> tak3::Tee(2) -> Schakel(QUALITY "hoog");
c1[2]-> tellerWireless::Counter() -> tak4::Tee(3) -> Schakel(QUALITY "laag");
tak3[1]-> Strip(14) -> GetIPAddress(16) -> arpq0;
tak4[1]-> Strip(14) -> GetIPAddress(16) -> arpq0;
tak4[2]->Strip(14)->GetIPAddress(16)->tellertest::Counter()->arpq1;
```

```
c2[4]->Discard;
c0[3]->Discard;
c1[4]->Discard;

fromhostcl[3]->Discard;
pulclas1[1]->Discard;
pulclas2[1]->Discard;
```

A.10 Click code van implementatie 5: Client

```
//Kiezen welke input en doorsturen naar classifier.
AddressInfo(wirelessInterface 192.168.1.2 00:13:F7:3B:BE:E9);
AddressInfo(wiredInterface 192.168.0.2 00:15:C5:CD:BF:D6);

combinator :: InputSwitch(1);
c0 :: Classifier(12/0806 20/0001,
                 12/0806 20/0002,
                 12/0800,
                 -
                 );

fromhost_cl :: Classifier(12/0806 20/0001,
                          12/0806 20/0002,
                          12/0800,
                          -,
                          );

FromDevice(ath1) -> Print()-> meet :: MyWifi() -> [0]combinator;
FromDevice(eth1) -> [1]combinator;
combinator -> c0;
FromHost(fakedev, 192.168.3.1/32) -> fromhost_cl;

ar2 :: ARPResponder(0.0.0.0/0 1:1:1:1:1:1);
ar3 :: ARPResponder(192.168.3.1 00:13:F7:3B:BE:E9); //wireless
ar4 :: ARPResponder(192.168.3.1 00:15:C5:CD:BF:D6); //wire

verdelerARP2 :: Switch(1);

out0 :: Counter()->Queue -> ToDevice(ath1);
out1 :: Counter()->Queue -> ToDevice(eth1);

c0 -> verdelerARP2 -> ar3 -> out0;


verdelerARP2[1] -> ar4 -> out1;
ToLinux::ToHost();
fromhost_cl[0] -> ar2 -> ToLinux;

//Opvangen ARP responses
arpq0 :: ARPQuerier(192.168.1.2,00:13:F7:3B:BE:E9);
arpq1 :: ARPQuerier(192.168.0.2,00:15:C5:CD:BF:D6);
```

```
verdelerARPQ :: Switch(1);

fromhost_cl[1] -> Discard;
c0[1] -> verdelerARPQ;
verdelerARPQ[0] -> [1]arpq0;
verdelerARPQ[1] -> [1]arpq1;

//Verdelen standaard IP pakketten
verdelerROUTEtoDev :: Switch(1);


rt3 :: LinearIPLookup(192.168.3.1 0,
                      0/0 192.168.0.1 0); //for wired

rt2 :: LinearIPLookup(192.168.3.1 0, //for wireless
                      0/0 192.168.1.100 0);

c0[2] -> Tak3::Tee(2)->ToLinux;
Tak3[1]->IPClassifier(udp)->linux::Counter()->Discard;

fromhost_cl[2] -> Strip(14) -> GetIPAddress(16) -> [0]verdelerROUTEtoDev

verdelerROUTEtoDev[0] -> rt2;
verdelerROUTEtoDev[1] -> rt3;

rt2 -> [0]arpq0;
rt3 -> [0]arpq1;

arpq0 -> out0;
arpq1 -> out1;

senderlaag::ICMPPingSource(192.168.3.1, 192.168.6.1, LIMIT 1,
ACTIVE false, DATA "laag")->EtherEncap(0x800, 00:13:F7:3B:BE:E9,
00:11:6B:31:15:4D)->tak1::Tee(2)->out0;
senderhoog::ICMPPingSource(192.168.3.1, 192.168.6.1, LIMIT 1,
ACTIVE false, DATA "hoog")->EtherEncap(0x800, 00:15:C5:CD:BF:D6,
00:11:6B:31:11:9C)->tak2::Tee(2)->out1;

senderlaag1::ICMPPingSource(192.168.3.1, 192.168.5.1, LIMIT 1,
ACTIVE false, DATA "laag")->EtherEncap(0x800, 00:13:F7:3B:BE:E9,
00:11:6B:31:15:4D)->tak11::Tee(2)->out0;
senderhoog1::ICMPPingSource(192.168.3.1, 192.168.5.1, LIMIT 1,
ACTIVE false, DATA "hoog")->EtherEncap(0x800, 00:15:C5:CD:BF:D6,
00:0D:56:CE:63:4E)->tak21::Tee(2)->out1;

tak1[1]->ToHost(ath1);
tak2[1]->ToHost(eth1);
tak11[1]->ToHost(ath1);
tak21[1]->ToHost(eth1);
c0[3] -> Discard;
fromhost_cl[3]-> Discard;
```

A.11 Click code van implementatie 6: Gemeenschappelijke node

```
c0 :: Classifier(12/0806 20/0001,   //verbinding met server arp request
                 12/0806 20/0002, //arp response
                 12/0800, //ip pakket
                 -);
c1 :: Classifier(12/0806 20/0001,  //draadloze verbinding met laptop
                 12/0806 20/0002,
                 42/6c616167,     //lage kwaliteit
                 12/0800,
                 -);
c2 :: Classifier(12/0806 20/0001,  //bedrade verbinding met laptop
                 12/0806 20/0002,
  42/686f6f67,     //hoge kwaliteit
  12/0800,
  -);

fromhostcl :: Classifier(12/0806 20/0001,  //bedrade verbinding met laptop
                 12/0806 20/0002,
                 12/0800,
                 -);

//Neem pakketten van eth0, eth1 en eth2 en classificeer ze.
FromDevice(eth0)->c0;
FromDevice(eth1)->c1;
FromDevice(eth2)->c2;
FromHost(fake, 192.168.4.1/32) -> fromhostcl;

//de outputs definieren we hebben queues nodig voor de push pull conventie en
//verschillende prioriteiten
out0 :: Queue->ToDevice(eth0);

//prio queues
out11 :: Queue;
out12 :: Queue;

out21 :: Queue;
out22 :: Queue;

//prioriteits scheduling voor de outputqueues en de bypas
prio1 :: PrioSched();
prio2 :: PrioSched();

//queues aan prioriteiten toekennen
out11->[1]prio1;
out12->[2]prio1;

out21->telleri::Counter()->[1]prio2;
out22->tellerp::Counter()->[2]prio2;

//definities voor pakketen om te leiden
```

```
puller1::PullTee(2);
puller2::PullTee(2);

prio1->puller1;
prio2->puller2;

//omleiden van pakketten
pulclas1::IPClassifier(src 192.168.6.1,-);
pulclas2::IPClassifier(src 192.168.6.1,-);

puller1[1]->suppres1::Switch(1)[1]->pulclas1->Strip(14)
->EtherEncap(0x0800, 00:11:6B:31:11:9C, 00:15:c5:cd:bf:d6)->Queue->prio2;
puller2[1]->suppres2::Switch(0)[1]->pulclas2->Strip(14)
->EtherEncap(0x0800, 00:11:6B:31:11:9C, 00:13:f7:3b:be:e9)->Queue->prio1;
suppres1[0]->Discard;
suppres2[0]->Discard;


//Is het IP adres in dit pakket dat van mij? Zo ja, stuur een ARP boodschap met mijn MAC
//naar de vrager.
ar0 :: ARPResponder(192.168.6.2 00:0F:1F:8A:06:48);
ar1 :: ARPResponder(192.168.1.100 00:11:6B:31:15:4D);
ar2 :: ARPResponder(192.168.0.1 00:11:6B:31:11:9C);
ar3 :: ARPResponder(0.0.0.0/0 1:1:1:1:1:1);


//ARP queries worden behandeld door de responder en verdergestuurd naar de uitgang.
c0[0] -> ar0 -> out0;
c1[0] -> ar1 -> out11;
c2[0] -> ar2 -> out21;
fromhostcl -> ar3 -> ToHost();

//Definieren ARP querier. Deze vraagt om het MAC adres van een bepaald IP als het dit niet kent.
arpq0 :: ARPQuerier(192.168.6.2,00:0F:1F:8A:06:48);
arpq1 :: ARPQuerier(192.168.1.100,00:11:6B:31:15:4D);
arpq2 :: ARPQuerier(192.168.0.1,00:11:6B:31:11:9C);
arpq3 :: ARPQuerier(192.168.0.1,00:11:6B:31:11:9C);

//ARP responses gaan naar de ARP querier omdat die responses verwacht op zijn queries.
c0[1] -> [1]arpq0;
c1[1] -> [1]arpq1;
c2[1] -> tak :: Tee(2) -> [1]arpq2;
fromhostcl[1] -> Discard;

/*Dit zijn de standaard IP pakketten. Het IP moet in het 'annotation field' worden gezet
om functionele redenen
van de ARP querier. Getipaddress sets the annotation for the next hop. Het verandert de
standaard headers niet.
Voeg hier ook de inputswitch en switch toe*/
verdeler0 :: Switch(1); // Default (0) uitgang is naar de draadloze verbinding.
combinator0 :: InputSwitch(1); // Default (0) ingang is van de draadloze verbinding.
//UDPCheck :: SetUDPChecksum();
//UDPCheck -> Discard;
```

```
c0[2] -> tak6 :: Tee(2) -> Strip(14) -> GetIPAddress(16) -> verdeler0;
verdeler0[0] -> arpq1;
verdeler0[1] -> arpq2;
c1[3] -> [0]combinator0;
c2[3] -> [1]combinator0;
combinator0 -> Strip(14) -> GetIPAddress(16) -> arpq0;
fromhostcl[2] -> Strip(14) -> GetIPAddress(16) -> arpq3;
tak[1] -> [1]arpq3;

//ff wa tellen
tak6[1] -> IPClassifier(udp) -> udptellerke :: Counter() -> Discard;

//uitgang querier naar juiste output sturen
arpq0->out0;
cl1 :: IPClassifier(src 192.168.6.1,-)   //video en rest
cl2 :: IPClassifier(src 192.168.6.1,-) //video en rest

arpq1->cl1;
cl1->wirelessfilm::Counter()->out11;
cl1[1]->out12;
//cl1[2]->out13;
//cl1[3]->out14

arpq2->cl2;

cl2->wirefilm::Counter()->out21;
cl2[1]->out22;
//cl2[2]->out23;
//cl2[3]->out24;

arpq3->out21;

//omschakeling

c2[2]-> tellerWire::Counter() -> tak3::Tee(2) -> Schakel(QUALITY "hoog");
c1[2]-> tellerWireless::Counter() -> tak4::Tee(2) -> Schakel(QUALITY "laag");
tak3[1]-> Strip(14) -> GetIPAddress(16) -> arpq0;
tak4[1]-> Strip(14) -> GetIPAddress(16) -> arpq0;

c2[4]->Discard;
c0[3]->Discard;
c1[4]->Discard;

fromhostcl[3]->Discard;
pulclas1[1]->Discard;
pulclas2[1]->Discard;

puller1->BandwidthShaper(1000kbps)->ToDevice(eth1);
puller2->BandwidthShaper(2000kbps)->ToDevice(eth2);
```

A.12 Click code van implementatie 7: Gemeenschappelijke node

```
c0 :: Classifier(12/0806 20/0001,   //verbinding met server arp request
                 12/0806 20/0002, //arp response
                 12/0800, //ip pakket
                 -);
c1 :: Classifier(12/0806 20/0001,  //draadloze verbinding met laptop
                 12/0806 20/0002,
                 42/6c616167,     //lage kwaliteit
                 12/0800,
                 -);
c2 :: Classifier(12/0806 20/0001,  //bedrade verbinding met laptop
                 12/0806 20/0002,
  42/686f6f67,     //hoge kwaliteit
  12/0800,
  -);

//c2::Tee(5);
fromhostcl :: Classifier(12/0806 20/0001,  //bedrade verbinding met laptop
                 12/0806 20/0002,
                 12/0800,
                 -);

//Neem pakketten van eth0, eth1 en eth2 en classificeer ze.
FromDevice(eth0)->c0;
FromDevice(eth1)->c1;
FromDevice(eth2)->c2;
FromHost(fake, 192.168.4.1/32) -> fromhostcl;

//de outputs definieren we hebben queues nodig voor de push pull conventie en verschillende
//prioriteiten
out0 :: Queue->ToDevice(eth0);

//prio queues
out11 :: Queue;
out12 :: Queue;

out21 :: Queue;
out22 :: Queue;

//prioriteits scheduling voor de outputqueues en de bypas
prio1 :: PrioSched();
prio2 :: PrioSched();

//queues aan prioriteiten toekennen
out11->[1]prio1;
out12->[2]prio1;

out21->telleri::Counter()->[1]prio2;
out22->tellerp::Counter()->[2]prio2;
```

```
//definities voor pakketen om te leiden
puller1::PullTee(2);
puller2::PullTee(2);

prio1->puller1;
prio2->puller2;

//omleiden van pakketten
pulclas1::IPClassifier(src 192.168.6.1,-);
pulclas2::IPClassifier(src 192.168.6.1,-);

puller1[1]->suppres1::Switch(1)[1]->pulclas1->Strip(14)
->EtherEncap(0x0800, 00:11:6B:31:11:9C, 00:15:c5:cd:bf:d6)->Queue->prio2;
puller2[1]->suppres2::Switch(0)[1]->pulclas2->Strip(14)
->EtherEncap(0x0800, 00:11:6B:31:11:9C, 00:13:f7:3b:be:e9)->Queue->prio1;
suppres1[0]->Discard;
suppres2[0]->Discard;

//Is het IP adres in dit pakket dat van mij? Zo ja, stuur een ARP boodschap met mijn MAC
//naar de vrager.
ar0 :: ARPResponder(192.168.6.2 00:0F:1F:8A:06:48);
ar1 :: ARPResponder(192.168.1.100 00:11:6B:31:15:4D);
ar2 :: ARPResponder(192.168.0.1 00:11:6B:31:11:9C);
ar3 :: ARPResponder(0.0.0.0/0 1:1:1:1:1:1);

//ARP queries worden behandeld door de responder en verdergestuurd naar de uitgang.
c0[0] -> ar0 -> out0;
c1[0] -> ar1 -> out11;
c2[0] -> ar2 -> out21;
fromhostcl -> ar3 -> ToHost();

//Definieren ARP querier. Deze vraagt om het MAC adres van een bepaald IP als het dit niet kent.
arpq0 :: ARPQuerier(192.168.6.2,00:0F:1F:8A:06:48);
arpq1 :: ARPQuerier(192.168.1.100,00:11:6B:31:15:4D);
arpq2 :: ARPQuerier(192.168.0.1,00:11:6B:31:11:9C);
arpq3 :: ARPQuerier(192.168.0.1,00:11:6B:31:11:9C);

//ARP responses gaan naar de ARP querier omdat die responses verwacht op zijn queries.
c0[1] -> [1]arpq0;
c1[1] -> [1]arpq1;
c2[1] -> tak :: Tee(2) -> [1]arpq2;
fromhostcl[1] -> Discard;

/*Dit zijn de standaard IP pakketten. Het IP moet in het 'annotation field' worden gezet
om functionele redenen
van de ARP querier. Getipaddress sets the annotation for the next hop. Het verandert de
standaard headers niet.
Voeg hier ook de inputswitch en switch toe*/
verdeler0 :: Switch(1); // Default (0) uitgang is naar de draadloze verbinding.
combinator0 :: InputSwitch(1); // Default (0) ingang is van de draadloze verbinding.

c0[2] -> tak6 :: Tee(2) -> Strip(14) -> GetIPAddress(16) -> verdeler0;
```

```
verdeler0[0] -> arpq1;//GetIPAddress(16)
verdeler0[1] -> arpq2;
c1[3] -> [0]combinator0;
c2[3] -> [1]combinator0;
combinator0 -> Strip(14) -> GetIPAddress(16) -> arpq0;
fromhostcl[2] -> Strip(14) -> GetIPAddress(16) -> arpq3;
tak[1] -> [1]arpq3;

//teller
tak6[1] -> IPClassifier(udp) -> udptellerke :: Counter() -> Discard;

//uitgang querier naar juiste output sturen
arpq0->out0;
cl1 :: Classifier(43/49,   //i
                  43/50,   //p
  43/42,   //b
  -);    //rest

cl2 :: Classifier(43/49,   //i
                  43/50,   //p
  43/42,   //b
  -);    //rest

arpq1->cl1;
cl1->out11;
cl1[1]->out11;
cl1[2]->out12;
cl1[3]->out12

arpq2->cl2;

cl2->out21;
cl2[1]->out21;
cl2[2]->out22;
cl2[3]->out22;

arpq3->out21;

//omschakeling

c2[2]-> tellerWire::Counter() -> tak3::Tee(2) -> Schakel(QUALITY "hoog");
c1[2]-> tellerWireless::Counter() -> tak4::Tee(2) -> Schakel(QUALITY "laag");
tak3[1]-> Strip(14) -> GetIPAddress(16) -> arpq0;
tak4[1]-> Strip(14) -> GetIPAddress(16) -> arpq0;

c2[4]->Discard;
c0[3]->Discard;
c1[4]->Discard;

fromhostcl[3]->Discard;
pulclas1[1]->Discard;
pulclas2[1]->Discard;
```

```
puller1->BandwidthShaper(1000kbps)->ToDevice(eth1);
puller2->BandwidthShaper(1000kbps)->ToDevice(eth2);
```

A.13 Click code van implementatie 8: Gemeenschappelijke node

```
c0 :: Classifier(12/0806 20/0001,   //verbinding met server arp request
                 12/0806 20/0002, //arp response
                 12/0800, //ip pakket
                 -);
c1 :: Classifier(12/0806 20/0001,  //draadloze verbinding met laptop
                 12/0806 20/0002,
                 42/6c616167,     //lage kwaliteit
                 12/0800,
                 -);
c2 :: Classifier(12/0806 20/0001,  //bedrade verbinding met laptop
                 12/0806 20/0002,
  42/686f6f67,     //hoge kwaliteit
  12/0800,
  -);

fromhostcl :: Classifier(12/0806 20/0001,  //bedrade verbinding met laptop
                 12/0806 20/0002,
                 12/0800,
                 -);

//Neem pakketten van eth0, eth1 en eth2 en classificeer ze.
FromDevice(eth0)->c0;
FromDevice(eth1)->c1;
FromDevice(eth2)->c2;
FromHost(fake, 192.168.4.1/32) -> fromhostcl;

//de outputs definieren we hebben queues nodig voor de push pull conventie en
//verschillende prioriteiten
out0 :: Queue->ToDevice(eth0);

//prio queues
out11 :: Queue;
out12 :: Queue;

out21 :: Queue;
out22 :: Queue;

//prioriteits scheduling voor de outputqueues en de bypas
prio1 :: PrioSched();
prio2 :: PrioSched();

//queues aan prioriteiten toekennen
out11->[1]prio1;
out12->[2]prio1;
```

```
out21->telleri::Counter()->[1]prio2;
out22->tellerp::Counter()->[2]prio2;

//definities voor pakketen om te leiden
puller1::PullTee(2);
puller2::PullTee(2);

prio1->puller1;
prio2->puller2;

//omleiden van pakketten
pulclas1::IPClassifier(udp,-);
pulclas2::IPClassifier(udp,-);

puller1[1]->suppres1::Switch(1)[1]->pulclas1->Strip(14)
->EtherEncap(0x0800, 00:11:6B:31:11:9C, 00:15:c5:cd:bf:d6)->Queue->prio2;
puller2[1]->suppres2::Switch(0)[1]->pulclas2->Strip(14)
->EtherEncap(0x0800, 00:11:6B:31:11:9C, 00:13:f7:3b:be:e9)->Queue->prio1;
suppres1[0]->Discard;
suppres2[0]->Discard;

//Is het IP adres in dit pakket dat van mij? Zo ja, stuur een ARP boodschap met mijn MAC
//naar de vrager.
ar0 :: ARPResponder(192.168.6.2 00:0F:1F:8A:06:48);
ar1 :: ARPResponder(192.168.1.100 00:11:6B:31:15:4D);
ar2 :: ARPResponder(192.168.0.1 00:11:6B:31:11:9C);
ar3 :: ARPResponder(0.0.0.0/0 1:1:1:1:1:1);

//ARP queries worden behandeld door de responder en verdergestuurd naar de uitgang.
c0[0] -> ar0 -> out0;
c1[0] -> ar1 -> out11;
c2[0] -> ar2 -> out21;
fromhostcl -> ar3 -> ToHost();

//Definieren ARP querier. Deze vraagt om het MAC adres van een bepaald IP als het dit niet kent.
arpq0 :: ARPQuerier(192.168.6.2,00:0F:1F:8A:06:48);
arpq1 :: ARPQuerier(192.168.1.100,00:11:6B:31:15:4D);
arpq2 :: ARPQuerier(192.168.0.1,00:11:6B:31:11:9C);
arpq3 :: ARPQuerier(192.168.0.1,00:11:6B:31:11:9C);

//ARP responses gaan naar de ARP querier omdat die responses verwacht op zijn queries.
c0[1] -> [1]arpq0;
c1[1] -> [1]arpq1;
c2[1] -> tak :: Tee(2) -> [1]arpq2;
fromhostcl[1] -> Discard;

/*Dit zijn de standaard IP pakketten. Het IP moet in het 'annotation field' worden gezet
//om functionele redenen
van de ARP querier. Getipaddress sets the annotation for the next hop. Het verandert
//de standaard headers niet.
Voeg hier ook de inputswitch en switch toe*/
verdeler0 :: Switch(1); // Default (0) uitgang is naar de draadloze verbinding.
```

```
combinator0 :: InputSwitch(1); // Default (0) ingang is van de draadloze verbinding.
//UDPCheck :: SetUDPChecksum();
//UDPCheck -> Discard;

c0[2] -> tak6 :: Tee(2) -> Strip(14) -> GetIPAddress(16) -> verdeler0;
verdeler0[0] -> arpq1;
verdeler0[1] -> arpq2;
c1[3] -> [0]combinator0;
c2[3] -> [1]combinator0;
combinator0 -> Strip(14) -> GetIPAddress(16) -> arpq0;
fromhostcl[2] -> Strip(14) -> GetIPAddress(16) -> arpq3;
tak[1] -> [1]arpq3;

//teller
tak6[1] -> IPClassifier(udp) -> udptellerke :: Counter() -> Discard;

//uitgang querier naar juiste output sturen
arpq0->out0;
cl1 :: Classifier(43/49,   //i
                  43/50,   //p
  43/42,   //b
  -);    //rest

cl2 :: Classifier(43/49,   //i
                  43/50,   //p
  43/42,   //b
  -);    //rest

//de meters nog juist zetten
arpq1->cl1;
cl1->out11;
cl1[1]->meter1::Meter(1Bps)->out11;
meter1[1]->Discard;
cl1[2]->out12;
cl1[3]->out12;

arpq2->cl2;

cl2->out21;
cl2[1]->meter2::Meter(1Bps)->out21;
meter2[1]->Discard;
cl2[2]->out22;
cl2[3]->out22;

arpq3->out21;

//omschakeling

c2[2]-> tellerWire::Counter() -> tak3::Tee(2) -> Schakel(QUALITY "hoog");
c1[2]-> tellerWireless::Counter() -> tak4::Tee(2) -> Schakel(QUALITY "laag");
tak3[1]-> Strip(14) -> GetIPAddress(16) -> arpq0;
tak4[1]-> Strip(14) -> GetIPAddress(16) -> arpq0;
```

```
c2[4]->Discard;
c0[3]->Discard;
c1[4]->Discard;

fromhostcl[3]->Discard;
pulclas1[1]->Discard;
pulclas2[1]->Discard;

puller1->BandwidthShaper(1000kbps)->ToDevice(eth1);
puller2->BandwidthShaper(2000kbps)->ToDevice(eth2);
```

A.14 C++ code van de Inputswitch

A.14.1 InputSwitch Header

```
#ifndef CLICK_INPUTSWITCH_HH
#define CLICK_INPUTSWITCH_HH
#include <click/element.hh>
CLICK_DECLS

/*
=c

InputSwitch([K])

=s classification

forwards packet stream from settable input to output

=d

InputSwitch sends every incoming packet from one of its input ports --
specifically, input number K -- to its output. The default K is zero;
negative K means to destroy input packets instead of forwarding them.
You can change K with a write handler. InputSwitch has an unlimited
number of inputs.

=h inputswitch read/write

Return or set the K parameter.

=h CLICK_LLRPC_GET_INPUTSWITCH llrpc

Argument is a pointer to an integer, in which the Switch's K parameter is
stored. (Not supported by now)

=h CLICK_LLRPC_SET_INPUTSWITCH llrpc

Argument is a pointer to an integer. Sets the K parameter to that integer.
(Not supported either)
```

```
=a StaticSwitch, PullSwitch, RoundRobinSwitch, StrideSwitch, HashSwitch,
RandomSwitch, Switch*/

class InputSwitch : public Element { public:

  InputSwitch();
  ~InputSwitch();

  const char *class_name() const { return "InputSwitch"; }
  const char *port_count() const { return "-/1"; }
  const char *processing() const { return PUSH; }
  void add_handlers();

  void notify_ninputs(int);
  int configure(Vector<String> &, ErrorHandler *);
  void configuration(Vector<String> &) const;
  bool can_live_reconfigure() const { return true; }

  void push(int, Packet *);

//  int llrpc(unsigned, void *);

 private:

  int _input;

  static String read_param(Element *, void *);
  static int write_param(const String &, Element *, void *, ErrorHandler *);

};

CLICK_ENDDECLS
#endif
```

A.14.2 InputSwitch Body

```
/*
 * inputswitch.{cc,hh} -- routes packets from one input of several
 * Dagang LI
 *
 * Copyright (c) 2004 ESAT-Telemic, K.U.Leuven.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, subject to the conditions
 * listed in the Click LICENSE file. These conditions include: you must
 * preserve this copyright notice, and you cannot mention the copyright
 * holders in advertising related to the Software without their permission.
 * The Software is provided WITHOUT ANY WARRANTY, EXPRESS OR IMPLIED. This
 * notice is a summary of the Click LICENSE file; the license in that file is
 * legally binding.
```

```
 */

#include <click/config.h>
#include "inputswitch.hh"
#include <click/confparse.hh>
#include <click/error.hh>
#include <click/llrpc.h>

//#include <click/handlercall.hh>
CLICK_DECLS

InputSwitch::InputSwitch()
{
//  MOD_INC_USE_COUNT;
  //add_output();
}

InputSwitch::~InputSwitch()
{
//  MOD_DEC_USE_COUNT;
}

/*
void
InputSwitch::notify_ninputs(int n)
{
  set_ninputs(n);
}
*/

int
InputSwitch::configure(Vector<String> &conf, ErrorHandler *errh)
{
  _input = 0;
  if (cp_va_parse(conf, this, errh,
  cpOptional,
  cpInteger, "active input", &_input,
  cpEnd) < 0)
    return -1;
  if (_input >= ninputs())
    _input = -1;
  return 0;
}

void
InputSwitch::configuration(Vector<String> &conf) const
{
  conf.push_back(String(_input));
}

void
InputSwitch::push(int port, Packet *p)
{
```

```
  if (_input == port)
    output(0).push(p);
  else
    p->kill();
}

String
InputSwitch::read_param(Element *e, void *)
{
  InputSwitch *isw = (InputSwitch *)e;
  return String(isw->_input) + "\n";
}

int
InputSwitch::write_param(const String &in_s, Element *e, void *, ErrorHandler *errh)
{
  InputSwitch *isw = (InputSwitch *)e;
  String s = cp_uncomment(in_s);
  if (!cp_integer(s, &isw->_input))
    return errh->error("InputSwitch input must be integer");
  if (isw->_input >= isw->ninputs())
    isw->_input = -1;
  return 0;
}

void
InputSwitch::add_handlers()
{
  add_read_handler("inputswitch", read_param, (void *)0);
  add_write_handler("inputswitch", write_param, (void *)0);
}

/*
// same as Switch::llrpc, should not be working
int
InputSwitch::llrpc(unsigned command, void *data)
{
  if (command == CLICK_LLRPC_SET_INPUTSWITCH) {
    int32_t *val = reinterpret_cast<int32_t *>(data);
    _input = (*val >= ninputs() ? -1 : *val);
    return 0;

  } else if (command == CLICK_LLRPC_GET_INPUTSWITCH) {
    int32_t *val = reinterpret_cast<int32_t *>(data);
    *val = _input;
    return 0;

  } else
    return Element::llrpc(command, data);
}
*/

CLICK_ENDDECLS
```

```
EXPORT_ELEMENT(InputSwitch)
ELEMENT_MT_SAFE(InputSwitch)
```

A.15 C++ code van Schakel

A.15.1 Schakel Header

```
#ifndef CLICK_SCHAKEL_HH
#define CLICK_SCHAKEL_HH
#include <click/element.hh>
#include <click/string.hh>

CLICK_DECLS
class Schakel : public Element { public:
    Schakel();
    ~Schakel();
    const char *class_name() const { return "Schakel"; }
    const char *port_count() const { return "1/0"; }
    const char *processing() const { return PUSH; }
    void push(int port, Packet *p);

    virtual int configure(Vector<String> &conf, ErrorHandler *errh);

private:

String _quality;

};
CLICK_ENDDECLS
#endif
```

A.15.2 Schakel Body

```
#include <click/config.h>
#include "schakel.hh"
#include <click/handlercall.hh>
#include <click/router.hh>
#include <click/error.hh>
#include <click/confparse.hh>
CLICK_DECLS

Schakel::Schakel()
{
}

Schakel::~Schakel()
{
}
```

```
int
Schakel::configure(Vector<String> &conf, ErrorHandler *errh)
{
   if (cp_va_kparse(conf, this, errh,
    "QUALITY", cpkP+cpkM, cpString, &_quality,
        cpEnd) < 0)
   return -1;
}

void
Schakel::push(int, Packet *p)
{
  if (_quality.find_left("hoog") == 0)
  {
    Element *sw1 = router()->find("verdeler0");
    int succes = HandlerCall::call_write(sw1,"switch","1");
    Element *sw2 = router()->find("combinator0");
    succes = HandlerCall::call_write(sw2,"inputswitch","1");
    Element *sw3 = router()->find("suppres1");
    succes = HandlerCall::call_write(sw3,"switch","1");
    Element *sw4 = router()->find("suppres2");
    succes = HandlerCall::call_write(sw4,"switch","0");
  }
  else if (_quality.find_left("laag") == 0)
  {
    Element *sw1 = router()->find("verdeler0");
    int succes = HandlerCall::call_write(sw1,"switch","0");
    Element *sw2 = router()->find("combinator0");
    succes = HandlerCall::call_write(sw2,"inputswitch","0");
    Element *sw3 = router()->find("suppres1");
    succes = HandlerCall::call_write(sw3,"switch","0");
    Element *sw4 = router()->find("suppres2");
    succes = HandlerCall::call_write(sw4,"switch","1");
  }
p->kill();
}
CLICK_ENDDECLS
EXPORT_ELEMENT(Schakel)
```

A.16 C++ code van MyWifi

A.16.1 MyWifi Header

```
#ifndef CLICK_MYWIFI_HH
#define CLICK_MYWIFI_HH
#include <click/element.hh>
CLICK_DECLS
class MyWifi : public Element { public:
    MyWifi();
    ~MyWifi();
```

```
    const char *class_name() const { return "MyWifi"; }
    const char *port_count() const { return "1/1"; }
    const char *processing() const { return PUSH; }
    void add_handlers();
    void push(int, Packet *);

    private:
    String _signal;
    String _hol;
    static String read_hol(Element *, void *);
    static int write_signal(const String&, Element*, void*, ErrorHandler*);
    static String read_signal(Element *, void *);

};
CLICK_ENDDECLS
#endif
```

A.16.2 MyWifi Body

```
#include <click/config.h>
#include "mywifi.hh"
#include <click/confparse.hh>
#include <click/error.hh>
#include <click/handlercall.hh>
#include <click/router.hh>
#include <click/packet_anno.hh>
#include <click/string.hh>

CLICK_DECLS

MyWifi::MyWifi()
{
_hol="hoog";
_signal="70";
}

MyWifi::~MyWifi()
{
}

void
MyWifi::push(int, Packet *p)
{
char een, twee;
MyWifi *mw = (MyWifi *)this;

_signal=HandlerCall::call_read(mw, "signal");
een=_signal.at(0);
twee=_signal.at(1);
int inteen = int(een)-48;
int inttwee = int(twee)-48;
int totaal=inteen*10+inttwee;
```

```
click_chatter("het totaal is: %d",totaal);


if (totaal<45)
{
Element *sw1 = router()->find("combinator");
int success = HandlerCall::call_write(sw1, "inputswitch", "1");
Element *sw2 = router()->find("verdelerARP2");
success = HandlerCall::call_write(sw2, "switch", "1");
Element *sw3 = router()->find("verdelerARPQ");
success = HandlerCall::call_write(sw3, "switch", "1");
Element *sw4 = router()->find("verdelerROUTEtoDev");
success = HandlerCall::call_write(sw4, "switch", "1");
Element *sw5 = router()->find("senderhoog");
String _active = HandlerCall::call_read(sw5, "active");
if (_active =="false")
success = HandlerCall::call_write(sw5, "active", "true");

_hol=String(totaal);
click_chatter("in kleiner");

}
//else
//{
//
// Element *sw1 = router()->find("combinator");
// int success = HandlerCall::call_write(sw1, "inputswitch", "0");
// Element *sw2 = router()->find("verdelereARP2");
// success = HandlerCall::call_write(sw1, "switch", "0");
// Element *sw3 = router()->find("verdelerARPQ");
// success = HandlerCall::call_write(sw1, "switch", "0");
// Element *sw4 = router()->find("verdelerRoutetoDev");
// success = HandlerCall::call_write(sw1, "switch", "0");
// _hol=String(totaal);
// click_chatter("in groter");
//}
output(0).push(p);
}

String
MyWifi::read_hol(Element *e, void *)
{
  MyWifi *mw = (MyWifi *)e;
  return String(mw->_hol);

}

String
MyWifi::read_signal(Element *e, void *)
{
  MyWifi *mw = (MyWifi *)e;
  return String(mw->_signal);
```

```
}

int
MyWifi::write_signal(const String &in_str, Element *e, void *thunk, ErrorHandler *errh)
{
    MyWifi *mw = (MyWifi *)e;
    String str = cp_uncomment(in_str);
    mw->_signal=in_str;
}

void
MyWifi::add_handlers()
{
  add_read_handler("hol", read_hol, (void *)0);
  add_read_handler("signal", read_signal, (void *)0);
  add_write_handler("signal", write_signal, (void *)0);
}

CLICK_ENDDECLS
EXPORT_ELEMENT(MyWifi)
```

A.17 C++ code van TelnetConnect

A.17.1 TelnetConnect Header

```
// -*- mode: c++; c-basic-offset: 2 -*-
#ifndef CLICK_TelnetConnect_HH
#define CLICK_TelnetConnect_HH
#include <click/element.hh>
#include <click/string.hh>
#include <sys/un.h>
CLICK_DECLS

/*
=c
telnetConnect(IP,PORT,QUALITY)

=s comm

a socket to telnet for custom application (user-level)

=d

Makes a connection to a telnet server running on a video streaming server.
This happens by making a socket connection to the telnet server.
Afterwards it issues "built-in" commands, that change the videostream quality.
The videostreams get synchronized.

Keyword arguments are:
```

```
=item IP

Telnet server IP address.

=item PORT

Telnet server PORT.

=item QUALITY

What quality to change to. Options are "hoog" and "laag"

=back

=e

=a RawSocket Socket*/

class TelnetConnect : public Element {

public:

  TelnetConnect();
  ~TelnetConnect();

  const char *class_name() const { return "TelnetConnect"; }
  const char *port_count() const { return "1/0"; }
  const char *processing() const { return PUSH; }

  virtual int configure(Vector<String> &conf, ErrorHandler *errh);

  bool can_live_reconfigure() const { return false; }
  void push(int port, Packet *p);

private:

String _addr;
int _poort;

int _sockfd; // socket descriptor
int _active; // connection descriptor
struct sockaddr_in _dest_addr;   // will hold the destination addr
String _quality;

void clearbuf(char * buf);
int sendall(int _sockfd, const char *buf, int *len);
void telnetSend(String _command, int _socket);
String telnetRead(String endString);
int extractTime(String time);
double extractPosition(String position);
int extractState(String state);

};
```

```
CLICK_ENDDECLS
#endif
```

A.17.2 TelnetConnect Body

```
// -*- mode: c++; c-basic-offset: 2 -*-
/*
 * TelnetConnect.{cc,hh} -- makes a telnet connection
 * Lars Peters <lw.peters@gmail.com>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, subject to the conditions
 * listed in the Click LICENSE file. These conditions include: you must
 * preserve this copyright notice, and you cannot mention the copyright
 * holders in advertising related to the Software without their permission.
 * The Software is provided WITHOUT ANY WARRANTY, EXPRESS OR IMPLIED. This
 * notice is a summary of the Click LICENSE file; the license in that file is
 * legally binding.
 */

#include <click/config.h>
#include <click/error.hh>
#include <click/confparse.hh>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <arpa/inet.h>
#include "telnetConnect.hh"
#include <time.h>

CLICK_DECLS

TelnetConnect::TelnetConnect()
{
}

TelnetConnect::~TelnetConnect()
{
}

int
TelnetConnect::configure(Vector<String> &conf, ErrorHandler *errh)
{
  if (cp_va_kparse(conf, this, errh,
   "IP", cpkP+cpkM, cpString, &_addr,
   "PORT", cpkP+cpkM, cpInteger, &_poort,
   "QUALITY", cpkP+cpkM, cpString, &_quality,
    cpEnd) < 0)
    return -1;
}
```

```
void
TelnetConnect::push(int port, Packet *p)
{
  p->kill();

  //setup socket file descriptor with some error checking
  _sockfd = socket(PF_INET, SOCK_STREAM, 0);
  if (_sockfd == -1) click_chatter("Initialization failed");

  //set addresses
  _dest_addr.sin_family = AF_INET;          // host byte order
  _dest_addr.sin_port = htons(_poort);   // short, network byte order
  _dest_addr.sin_addr.s_addr = inet_addr(_addr.c_str());
  memset(_dest_addr.sin_zero, '\0', sizeof _dest_addr.sin_zero);

  //some printing for programming error checking
  click_chatter("Effe zien hoe t zit met die pointers = %d", _poort);
  click_chatter("Going to connect to IP ADDRESS = %s", _addr.c_str());
  click_chatter("PORT NR = %d", ntohs(_dest_addr.sin_port));

  //connect to address with some error chechking
  if (connect(_sockfd, (struct sockaddr *)&_dest_addr, sizeof _dest_addr) == -1)
  {
     click_chatter("Connecting failed.\n\n");
  }
  else click_chatter("Connecting succeeded.\n\n");

  //Read login
  String message = telnetRead("Password:");

  //Send password
  telnetSend("admin\r\n", _sockfd);

  //Read welcome message
  message = telnetRead(">");

  //Get channel info
  telnetSend("show channel1\r\n", _sockfd);

  //Get time and change channels
  struct timespec sleepTime;
  sleepTime.tv_sec = 5.0;
  sleepTime.tv_nsec = 0;

  message = telnetRead(">");

if (_quality == "laag")
{
  click_chatter("Omzetting naar lage kwaliteit!");
  telnetSend("show channel1\r\n",_sockfd);
  message = telnetRead(">"); //get updated time
  if (int state = extractState(message) == 0)
```

```
  {
   click_chatter("Staat al op lage kwaliteit");
  }
  else
  {
  double positie = extractPosition(message);
  telnetSend("control channel1 pause\r\n", _sockfd);
  message=telnetRead(">");
  String change = "control channel2 seek ";
  change = change.operator+=(String(positie));
  change = change.operator+=("\r\n");
  telnetSend(change,_sockfd);
  message=telnetRead(">");
  telnetSend("control channel2 pause\r\n", _sockfd);
  message=telnetRead(">");
  click_chatter("De string is : %s",change.c_str());
  }
  click_chatter("************ DONE ***************");
}

if (_quality == "hoog")
{
  telnetSend("show channel2\r\n",_sockfd);
  message = telnetRead(">"); //get updated time
  if (int state = extractState(message) == 0)
  {
   click_chatter("Staat al op hoge kwaliteit");
  }
  else
  {
  double positie = extractPosition(message);
  telnetSend("control channel2 pause\r\n", _sockfd); //Apply changed output settings
  message=telnetRead(">");
  String change = "control channel1 seek ";
  change = change.operator+=(String(positie));
  change = change.operator+=("\r\n");
  telnetSend(change,_sockfd);
  message=telnetRead(">");
  telnetSend("control channel1 pause\r\n", _sockfd);
  message=telnetRead(">");
  click_chatter("De string is : %s",change.c_str());
  }
  click_chatter("************ DONE ***************");
}

// some function I "borrowed" from Beej's programming guide
// to get all the data out of the socket if it doesnt get out all at once.
int
TelnetConnect::sendall(int _sockfd, const char *buf, int *len)
{
    int total = 0;        // how many bytes we've sent
    int bytesleft = *len; // how many we have left to send
    int n;
```

```
    while(total < *len) {
        n = send(_sockfd, buf+total, bytesleft, 0);
        if (n == -1) break;
        total += n;
        bytesleft -= n;
    }

    *len = total; // return number actually sent here
    return ( (n== -1) ? -1:0); // return -1 on failure, 0 on success
}

void
TelnetConnect :: telnetSend(String _command, int _socket)
{
  int len = strlen(_command.c_str());

  click_chatter("Sending data...");
  if (sendall(_socket, _command.c_str(), &len) == -1) \
      click_chatter("We only sent %s bytes because of the error!\n\n", len);
  else click_chatter("All bytes sent\n\n");
}

String
TelnetConnect :: telnetRead(String endString)
{
  char buf [50] = " ";
  String lineBuf;
  int len;
  len = strlen(buf);
  int _num_recvd;

  click_chatter("Reading data...");
  while (lineBuf.find_left(endString) == -1)
  {
   if((_num_recvd=recv(_sockfd, buf, len, 0))==-1) {
     click_chatter("Receiving failed.\n\n");
   }
   else {
          click_chatter("We managed to receive %d bytes : %s",_num_recvd, buf);
        }
   lineBuf = operator+(lineBuf, String(buf));
   clearbuf(buf);
  }

click_chatter("We've received : %s\n\n", lineBuf.c_str());
return lineBuf;
}

void
TelnetConnect::clearbuf (char * buf)
{
memset(buf, 0, sizeof(buf));
```

```
}

int
TelnetConnect::extractTime(String time)
{
  int beginTijd;
  int eindTijd;
  int int_time=0;

  beginTijd = time.find_left("time") + 7;
  eindTijd = time.find_left(" " ,beginTijd);

  time = time.substring(beginTijd,eindTijd-beginTijd);

  if (beginTijd >=7) { // If a time is found
    int_time = atoi(time.c_str())/1000;  // time in ms
  }
  else {
    click_chatter("Geen tijd gevonden");
    int_time = -1; // return -1 if no time is found
  }
  return int_time;
}

double
TelnetConnect::extractPosition(String position)
{
  int beginPositie;
  int eindPositie;
  double double_position=0.0;

  beginPositie = position.find_left("position") + 11;
  eindPositie = position.find_left(" " ,beginPositie);

  position = position.substring(beginPositie,eindPositie-beginPositie);

  if (beginPositie >=11) { // If a position is found
    double_position = atof(position.c_str())*100.0; // position
  }
  else {
    click_chatter("Geen positie gevonden");
    double_position = -1; // return -1 if no time is found
  }
  return double_position;
}

int
TelnetConnect::extractState(String state)
{

  int beginPositie;
  int eindPositie;
```

```
  beginPositie = state.find_left("state") + 8 ;
  eindPositie = state.find_left(" " ,beginPositie);

  state = state.substring(beginPositie,eindPositie-beginPositie -1);

  if (beginPositie >=8) { // If a position is found
    if (state.find_left("playing") != -1){
    click_chatter("De state is %s" , state.c_str());
    return -1;
     }
    if (state.find_left("paused") != -1) {
    click_chatter("De state is %s", state.c_str());
    return 0;
     }
  }
  else {
    click_chatter("Geen state gevonden, kanaal speelt waarschijnlijk niet.");
    return -100;
  }
}
CLICK_ENDDECLS
EXPORT_ELEMENT(TelnetConnect)
```

A.18 C++ code van FakeStream

A.18.1 FakeStream Header

```
#ifndef CLICK_FakeStream_HH
#define CLICK_FakeStream_HH
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <click/element.hh>
#include <click/string.hh>
#include <sys/un.h>
#include <click/gaprate.hh>
#include <click/task.hh>

CLICK_DECLS

/*
=c
FakeStream(FILENAME)

=s comm

Generates a fakestream with data out of FILE.

=d
```

```
Reads out FILE. Gets multiple arguments out of it and puts it into packets.
Also takes into account maximum payload size.

Keyword arguments are:

=item FILE

Filename with necessary data.

=back

=e

=a */

class FakeStream : public Element {

public:

  FakeStream();
  ~FakeStream();

  const char *class_name() const { return "FakeStream"; }
  const char *port_count() const { return "0/1"; }
  const char *processing() const { return AGNOSTIC; }

  virtual int configure(Vector<String> &conf, ErrorHandler *errh);
  void add_handlers();

  bool can_live_reconfigure() const { return false; }
  int initialize(ErrorHandler *);
  void cleanup(CleanupStage);

  bool run_task(Task *);
  Packet *pull(int);

  int T_fread(FILE *input, float lDataSize []);
  int T_fread(FILE *input, String sDataSize []);
  long getFileLength(FILE *input);

private:

#define CR 13             /* Decimal code of Carriage Return char */
#define LF 10             /* Decimal code of Line Feed char */
#define EOF_MARKER 26     /* Decimal code of DOS end-of-file marker */
#define MAX_REC_LEN 1024  /* Maximum size of input buffer */
#define TAB 09     /* Decimal code of Tab char*/
#define MAX_PACKET_SIZE 1448  // Maximum packet size

  int   iReadReturn;             /* Result of read function */
  int   isFilePosErr;            /* Boolean indicating file offset error */
  long  lFileLen;                /* Length of file */
```

```
  long  lLastFilePos;            /* Byte offset of end of previous line */
  long  lLineCount;              /* Line count accumulator */
  long  lLineLen;                /* Length of current line */
  long  lThisFilePos;            /* Byte offset of start of current line */
  char  szReadLine[MAX_REC_LEN]; /* Input buffer */
  FILE *inputFilePtr;            /* Pointer to input file */
  String _filename;
  long lFileLength;

  bool setup_packet(float dFrames [], float dDispTime [], String sType [], float dSize []);

  GapRate _rate;
  long _count;
  long _limit;
  int _datasize;
  bool _active : 1;
  bool _stop : 1;
  Packet *_packet;
  Task _task;
  String _data;
  bool _multiplePackets;
  long _packetRemain;
  double _packetCount;

  float *fFrames;
  float *fDispTime;
  String *sType;
  float *fSize;

  static String read_param(Element *, void *);
  static int change_param(const String &, Element *, void *, ErrorHandler *);

};
CLICK_ENDDECLS
#endif
```

A.18.2 FakeStream Body

```
// -*- mode: c++; c-basic-offset: 2 -*-
/*
 * FakeStream.{cc,hh} -- generates a trace video stream
 * Lars Peters <lw.peters@gmail.com>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, subject to the conditions
 * listed in the Click LICENSE file. These conditions include: you must
 * preserve this copyright notice, and you cannot mention the copyright
 * holders in advertising related to the Software without their permission.
 * The Software is provided WITHOUT ANY WARRANTY, EXPRESS OR IMPLIED. This
 * notice is a summary of the Click LICENSE file; the license in that file is
 * legally binding.
```

```
 */

#include <click/config.h>
#include <click/error.hh>
#include <click/confparse.hh>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/un.h>
#include <arpa/inet.h>
#include "fakeStream.hh"
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <click/element.hh>
#include <click/string.hh>
#include <click/router.hh>
#include <click/straccum.hh>
#include <click/standard/scheduleinfo.hh>
#include <click/glue.hh>

CLICK_DECLS

FakeStream::FakeStream()
  : _packet(0), _task(this)
{
}

FakeStream::~FakeStream()
{
}

int
FakeStream::configure(Vector<String> &conf, ErrorHandler *errh)
{
  unsigned rate = 30;
  bool active = true, stop = false;
  _multiplePackets = false;
  _packetCount = 0;

  if (cp_va_kparse(conf, this, errh,
   "FILENAME", cpkP+cpkM, cpString, &_filename,
    cpEnd) < 0)
    return -1;

  int iLengthFilename = _filename.length(); //do it before appending strings

//Get number of lines in files = number of packets
  String _filenameFrames = _filename.operator+=("_frames");
  inputFilePtr = fopen(_filenameFrames.c_str(), "r");  // Open in TEXT mode

  if (inputFilePtr == NULL )             // Could not open file
```

```
  {
    click_chatter("Error opening %s: %s (%u)\n", _filename.c_str(), strerror(errno), errno);
  }

  lFileLength = getFileLength(inputFilePtr);

//initialise some variables
  _rate.set_rate(rate, errh);
  _limit = lFileLength-1;
  _active = active;
  _stop = stop;
  _count = 0;

  fFrames = NULL;
  fDispTime = NULL;
  sType = NULL;
  fSize = NULL;

//  setup_packet();


//Create queues with DATA
//Get frames

  fFrames = new float[lFileLength];

  iReadReturn = T_fread(inputFilePtr, fFrames); // Read the file and print output
  fclose(inputFilePtr);                // Close it

//Get playtime
  _filename = _filenameFrames.substring(0,iLengthFilename);
  String _filenameDispTime = _filename.operator+=("_disptime");

  inputFilePtr = fopen(_filenameDispTime.c_str(), "r");  // Open in TEXT mode

  if (inputFilePtr == NULL )             // Could not open file
  {
    click_chatter("Error opening %s: %s (%u)\n", _filenameDispTime.c_str(), \
   strerror(errno), errno);
  }
  fDispTime  = new float[lFileLength];

  iReadReturn = T_fread(inputFilePtr, fDispTime); // Read the file and print output
  fclose(inputFilePtr);                // Close it

//Get frametype

  _filename = _filenameFrames.substring(0,iLengthFilename);
  String _filenameType = _filename.operator+=("_type");

  inputFilePtr = fopen(_filenameType.c_str(), "r");  // Open in TEXT mode

  if (inputFilePtr == NULL )             // Could not open file
```

```
  {
    click_chatter("Error opening %s: %s (%u)\n", _filenameType.c_str(), strerror(errno), errno);
  }
  sType = new String[lFileLength];

  iReadReturn = T_fread(inputFilePtr, sType); // Read the file and print output
  fclose(inputFilePtr);                // Close it

//Get size layer

  _filename = _filenameFrames.substring(0,iLengthFilename);
  String _filenameSize = _filename.operator+=("_size");

  inputFilePtr = fopen(_filenameSize.c_str(), "r");  // Open in TEXT mode

  if (inputFilePtr == NULL )             // Could not open file
  {
    click_chatter("Error opening %s: %s (%u)\n", _filenameSize.c_str(), strerror(errno), errno);
  }
  fSize = new float[lFileLength];

  iReadReturn = T_fread(inputFilePtr, fSize); // Read the file and print output
  fclose(inputFilePtr);                // Close it
return 0;
}

int
FakeStream::initialize(ErrorHandler *errh)
{
  if (output_is_push(0))
    ScheduleInfo::initialize_task(this, &_task, errh);
  return 0;
}

void
FakeStream::cleanup(CleanupStage)
{
  if (_packet)
    _packet->kill();
  _packet = 0;
}

bool
FakeStream::run_task(Task *)
{
    if (!_active)
return false;
    if (_count >= _limit) {
if (_stop)
    router()->please_stop_driver();
return false;
    }
    Timestamp now = Timestamp::now();
```

```
    if (_rate.need_update(now) && !_multiplePackets) {
if ( setup_packet(fFrames,fDispTime,sType,fSize) ) {
// click_chatter("Nieuwe frame");
_rate.update();
Packet *p = _packet->clone();
p->set_timestamp_anno(now);
output(0).push(p);
_packetCount++;
_task.fast_reschedule();
return true;
}
else {
_task.fast_reschedule();
return false;
}
    }
    else if (_multiplePackets) {
// click_chatter("Vervolg frames");
setup_packet(fFrames,fDispTime,sType,fSize);
Packet *p = _packet->clone();
p->set_timestamp_anno(now);
output(0).push(p);
_packetCount++;
_task.fast_reschedule();
return true;
}
        else{
  _task.fast_reschedule();
  return false;
        }
}

Packet *
FakeStream::pull(int)
{
    if (!_active)
return 0;
    if (_count >= _limit) {
if (_stop)
    router()->please_stop_driver();
return 0;
    }
    Timestamp now = Timestamp::now();
    if (_rate.need_update(now) && !_multiplePackets) {
// click_chatter("Nieuwe frame");
if ( setup_packet(fFrames,fDispTime,sType,fSize) ) {
_rate.update();
Packet *p = _packet->clone();
p->set_timestamp_anno(now);
_packetCount++;
return p;
}
else return 0;
```

```
    }
    else if (_multiplePackets) {
// click_chatter("Vervolg frames");
setup_packet(fFrames,fDispTime,sType,fSize);
Packet *p = _packet->clone();
p->set_timestamp_anno(now);
_packetCount++;
return p;
}
      else{
return 0;
      }
}

bool
FakeStream::setup_packet(float fFrames [], float fDispTime [], String sType [], float fSize [])
{
    if (_packet)
_packet->kill();

    // note: if you change ‘headroom’, change ‘click-align’
    unsigned int headroom = 16+20+24;
//    click_chatter ("Fout voor make pakket met count: %d en size %f", _count, fSize[_count]);

    if (fSize [_count] != 0.0) // Is there a frame being sent?
    {
    //Create packetdata FORMAT: Frame/dispTime/Type/Size
//    click_chatter("Na eerste if");
    _data="";
    int BUFFER_SIZE = 50;
    char _buf[BUFFER_SIZE];
    snprintf(_buf, BUFFER_SIZE, "~%s~N%d~F%d~T%f ~S%d ", sType[_count].c_str(),\
     int(_packetCount), int(fFrames[_count]), fDispTime[_count], int(fSize[_count]));
    _data = String(_buf);
//    click_chatter("Buffer bevat = %s", _buf);
//    click_chatter("Setuppakket = %s", _data.c_str());

    //Is datalength enough or is there padding required?
      if (int(fSize[_count]/8) <= _data.length()) { //No padding required
  _packet = Packet::make(headroom, (unsigned char *) _data.data(), int(fSize[_count]/8), 0);
  click_chatter("%s", _data.c_str());
  _count++;
         return true;
      }
      else if (fSize[_count] <= MAX_PACKET_SIZE){  //Padding is required AND
//frame size is smaller than maximum packet size
  // make up some data to fill extra space //Vul hier max aantal bytes
  StringAccum sa;
  while (sa.length() < int(fSize[_count]/8))
      sa << _data;
  _packet = Packet::make(headroom, (unsigned char *) sa.data(), int(fSize[_count]/8), 0);
  click_chatter("%s", _data.c_str());
         _count++;
```

```
          return true;
      }
         else { //Padding is required AND
//packet size is larger than maximum size
            if (!_multiplePackets) {
//       click_chatter("Start creating multiple packets");
          _multiplePackets = true;
      _packetRemain = long(fSize[_count]/8);
    }

    if (_packetRemain >= MAX_PACKET_SIZE) //Make packets of maximum packet
//size as long as needed
    {
      StringAccum sa;
      while (sa.length() < MAX_PACKET_SIZE) sa << _data;
      _packet = Packet::make(headroom, (unsigned char *) sa.data(), MAX_PACKET_SIZE, 0);
              click_chatter("%s", _data.c_str());
      _packetRemain -= MAX_PACKET_SIZE;
//       if (_multiplePackets ) click_chatter ("Aantal keer dat ik maximale pakketten heb");
      if (_packetRemain == MAX_PACKET_SIZE) {
// click_chatter("Last packet is maximum size");
_multiplePackets = false;
        _count++;
      }
            return true;
    }
    else { //Fill the last packet with remainder of frame size
      StringAccum sa;
      while (sa.length() < _packetRemain) sa << _data;
      _packet = Packet::make(headroom, (unsigned char *) sa.data(), _packetRemain, 0);
              click_chatter("%s", _data.c_str());
      _count++;
//       click_chatter("Last packet is less than maximum size");
      _multiplePackets = false;
              return true;
    }
          }
        }
    else {
    _count++;
    return false;
    }
}

String
FakeStream::read_param(Element *e, void *vparam)
{
  FakeStream *rs = (FakeStream *)e;
  switch ((intptr_t)vparam) {
   case 0: // data
    return rs->_data;
   case 1: // rate
    return String(rs->_rate.rate());
```

```
   case 2: // limit
    return String(rs->_limit);
   case 3: // active
    return cp_unparse_bool(rs->_active);
   case 4: // count
    return String(rs->_count);
  case 6: // datasize
    return String(rs->_datasize);
   default:
    return "";
  }
}

int
FakeStream::change_param(const String &in_s, Element *e, void *vparam,
  ErrorHandler *errh)
{
  FakeStream *rs = (FakeStream *)e;
  String s = cp_uncomment(in_s);
  switch ((intptr_t)vparam) {

   case 1: { // rate
     unsigned rate;
     if (!cp_integer(s, &rate))
       return errh->error("rate parameter must be integer >= 0");
     if (rate > GapRate::MAX_RATE)
       // report error rather than pin to max
       return errh->error("rate too large; max is %u", GapRate::MAX_RATE);
     rs->_rate.set_rate(rate);
     break;
   }

   case 2: { // limit
     long limit;
     if (!cp_integer(s, &limit))
       return errh->error("limit parameter must be integer");
     rs->_limit = limit;
     break;
   }

   case 3: { // active
     bool active;
     if (!cp_bool(s, &active))
       return errh->error("active parameter must be boolean");
     rs->_active = active;
     if (rs->output_is_push(0) && !rs->_task.scheduled() && active) {
       rs->_rate.reset();
       rs->_task.reschedule();
     }
     break;
   }

   case 5: { // reset
```

```
    rs->_count = 0;
    rs->_rate.reset();
    if (rs->output_is_push(0) && !rs->_task.scheduled() && rs->_active)
      rs->_task.reschedule();
    break;
   }
  }
  return 0;
}

void
FakeStream::add_handlers()
{
  add_read_handler("data", read_param, (void *)0, Handler::CALM);
  add_read_handler("rate", read_param, (void *)1);
  add_write_handler("rate", change_param, (void *)1);
  add_read_handler("limit", read_param, (void *)2, Handler::CALM);
  add_write_handler("limit", change_param, (void *)2);
  add_read_handler("active", read_param, (void *)3, Handler::CHECKBOX);
  add_write_handler("active", change_param, (void *)3);
  add_read_handler("count", read_param, (void *)4);
  add_write_handler("reset", change_param, (void *)5, Handler::BUTTON);
  add_read_handler("length", read_param, (void *)6);
  // deprecated
  add_read_handler("datasize", read_param, (void *)6);

  if (output_is_push(0))
    add_task_handlers(&_task);
}

long
FakeStream::getFileLength(FILE *input)
{
  int   isNewline;              /* Boolean indicating we've read a CR or LF */
  long  lFileLen;               /* Length of file */
  long  lIndex;                 /* Index into cThisLine array */
  long  lLineCount;             /* Current line number */
  long  lStartPos;              /* Offset of start of current line */
  long  lTotalChars;            /* Total characters read */
  char  cThisLine[MAX_REC_LEN]; /* Contents of current line */
  char *cFile;                  /* Dynamically allocated buffer (entire file) */
  char *cThisPtr;               /* Pointer to current position in cFile */

  fseek(input, 0L, SEEK_END);  /* Position to end of file */
  lFileLen = ftell(input);     /* Get file length */
  rewind(input);               /* Back to start of file */

  cFile = (char *) calloc(lFileLen + 1, sizeof(char));

  if(cFile == NULL )
  {
    click_chatter("\nInsufficient memory to read file.\n");
  }
```

```
  fread(cFile, lFileLen, 1, input); /* Read the entire file into cFile */

  lLineCount  = 0L;
  lTotalChars = 0L;

  cThisPtr    = cFile;              /* Point to beginning of array */

  while (*cThisPtr)                 /* Read until reaching null char */
  {
    lIndex    = 0L;                 /* Reset counters and flags */
    isNewline = 0;
    lStartPos = lTotalChars;

    while (*cThisPtr)               /* Read until reaching null char */
    {
      if (!isNewline)               /* Haven't read a CR or LF yet */
      {
        if (*cThisPtr == CR || *cThisPtr == LF) /* This char IS a CR or LF */
          isNewline = 1;                        /* Set flag */
      }

      else if (*cThisPtr != CR && *cThisPtr != LF) /* Already found CR or LF */
        break;                                     /* Done with line */

      cThisLine[lIndex++] = *cThisPtr++; /* Add char to output and increment */
      ++lTotalChars;

    } /* end while (*cThisPtr) */

    cThisLine[lIndex] = '\0';     /* Terminate the string */
    ++lLineCount;                 /* Increment the line counter */
  }
return lLineCount;
}

int
FakeStream::T_fread(FILE *input, float dDataSize []) /* Use: Read text file using fread()
{

  int   isNewline;              /* Boolean indicating we've read a CR or LF */
  long  lFileLen;               /* Length of file */
  long  lIndex;                 /* Index into cThisLine array */
  long  lLineCount;             /* Current line number */
  long  lStartPos;              /* Offset of start of current line */
  long  lTotalChars;            /* Total characters read */
  char  cThisLine[MAX_REC_LEN]; /* Contents of current line */
  char *cFile;                  /* Dynamically allocated buffer (entire file) */
  char *cThisPtr;               /* Pointer to current position in cFile */

  fseek(input, 0L, SEEK_END);  /* Position to end of file */
  lFileLen = ftell(input);     /* Get file length */
  rewind(input);               /* Back to start of file */
```

```
  cFile = (char *) calloc(lFileLen + 1, sizeof(char));

  if(cFile == NULL )
  {
    click_chatter("\nInsufficient memory to read file.\n");
  }

  fread(cFile, lFileLen, 1, input); /* Read the entire file into cFile */

  lLineCount  = 0L;
  lTotalChars = 0L;

  cThisPtr    = cFile;              /* Point to beginning of array */

  while (*cThisPtr)                 /* Read until reaching null char */
  {
    lIndex    = 0L;                 /* Reset counters and flags */
    isNewline = 0;
    lStartPos = lTotalChars;

    while (*cThisPtr)               /* Read until reaching null char */
    {
      if (!isNewline)               /* Haven't read a CR or LF yet */
      {
        if (*cThisPtr == CR || *cThisPtr == LF) /* This char IS a CR or LF */
          isNewline = 1;                        /* Set flag */
      }

      else if (*cThisPtr != CR && *cThisPtr != LF) /* Already found CR or LF */
        break;                                     /* Done with line */

      cThisLine[lIndex++] = *cThisPtr++; /* Add char to output and increment */
      ++lTotalChars;

    } /* end while (*cThisPtr) */

    cThisLine[lIndex] = '\0';     /* Terminate the string */
    ++lLineCount;                 /* Increment the line counter */
    dDataSize[lLineCount-1]=atof(cThisLine);

//    click_chatter("Array is op plaats %d : %f", lLineCount-1, dDataSize[lLineCount-1]);
//    sleep(1);
  } /* end while (cThisPtr <= cEndPtr) */

//  lDataSize[0] = 1;
  return 1;

} /* end T_fread() */

int
FakeStream::T_fread(FILE *input, String sDataSize []) /* Use: Read text file using fread()
{
```

```
int   isNewline;              /* Boolean indicating we've read a CR or LF */
long  lFileLen;               /* Length of file */
long  lIndex;                 /* Index into cThisLine array */
long  lLineCount;             /* Current line number */
long  lStartPos;              /* Offset of start of current line */
long  lTotalChars;            /* Total characters read */
char  cThisLine[MAX_REC_LEN]; /* Contents of current line */
char *cFile;                  /* Dynamically allocated buffer (entire file) */
char *cThisPtr;               /* Pointer to current position in cFile */

fseek(input, 0L, SEEK_END);  /* Position to end of file */
lFileLen = ftell(input);     /* Get file length */
rewind(input);               /* Back to start of file */

cFile = (char *) calloc(lFileLen + 1, sizeof(char));

if(cFile == NULL )
{
  click_chatter("\nInsufficient memory to read file.\n");
}

fread(cFile, lFileLen, 1, input); /* Read the entire file into cFile */

lLineCount  = 0L;
lTotalChars = 0L;

cThisPtr    = cFile;             /* Point to beginning of array */

while (*cThisPtr)                /* Read until reaching null char */
{
  lIndex    = 0L;                /* Reset counters and flags */
  isNewline = 0;
  lStartPos = lTotalChars;

  while (*cThisPtr)              /* Read until reaching null char */
  {
    if (!isNewline)              /* Haven't read a CR or LF yet */
    {
      if (*cThisPtr == CR || *cThisPtr == LF) /* This char IS a CR or LF */
        isNewline = 1;                        /* Set flag */
    }

    else if (*cThisPtr != CR && *cThisPtr != LF) /* Already found CR or LF */
      break;                                     /* Done with line */

    cThisLine[lIndex++] = *cThisPtr++; /* Add char to output and increment */
    ++lTotalChars;

  } /* end while (*cThisPtr) */

  cThisLine[lIndex] = '\0';     /* Terminate the string */
  ++lLineCount;                 /* Increment the line counter */
```

```
    sDataSize[lLineCount-1]=String(cThisLine).trim_space();

//    click_chatter("Array is op plaats %d : %s", \
     lLineCount-1, sDataSize[lLineCount-1].c_str());
//    sleep(1);
  } /* end while (cThisPtr <= cEndPtr) */

  return 1;

} /* end T_fread() */
CLICK_ENDDECLS
EXPORT_ELEMENT(FakeStream)
\begin{verbatim}
```