NTNU
Det skapende universitet

Interdomain Multicast

Robust multikast videresending mellom
mobile ad-hoc - og kablede nettverk.

Simon Dehli

Master of Telematics - Communication Networks and Networked
Innlevert: juni 2016
Hovedveileder: Øivind Kure, ITEM
Medveileder: Marianne Hauge, FFI
Margrete B. Hauge, FFI
Lars Landmark, FFI

Norges teknisk-naturvitenskapelige universitet
Institutt for telematikk

Title:	Interdomain Multicast
Student:	Simon Dehli

Problem description:

Multicast is a routing scheme which can be beneficial for multiple use cases in military communications. Examples are blue force tracking and traditional push-to-talk communication solutions. Many multicast routing protocols exists for both wired networks and Mobile ad hoc networks (MANET). Interconnection of multicast in these two different network environments are needed in order to support multicast traffic spanning both network types. This problem has not received much attention in the research community and few solutions exists. Interconnection via multiple gateways can be one way to ensure adequate robustness of the multicast design spanning the two network types. The candidate is tasked to design and further implement and test a multicast solution for connecting wired and wireless domains with more than one gateway. Tailoring of existing protocols may be done to the extent possible and proprietary solutions may be proposed where deemed necessary.

Responsible professor:	Øivind Kure, ITEM
Supervisor:	Mariann Hauge, FFI
	Lars Landmark, FFI

Abstract

There are multiple multicast routing protocols for Mobile Ad-hoc Networks (MANETs) such as Simplified Multicast Forwarding (SMF), Multicast Ad-Hoc On-Demand Distance Vector (MAODV) and Multicast Optimized Link State Routing (MOLSR). The common denominator for these three (3) is that they route multicast traffic within the MANET, but that none of them embeds gateway functionality considering the connection of other types of networks. Thus, these protocols do not include functionality for interconnecting one wired Internet Protocol (IP) network's multicast service with a MANET's multicast service and support for interoperation between wired and MANET multicast routing protocols.

We propose an implementation - using multiple gateways - which provide this functionality. The proposed implementation supports partitioning in the MANET, link failures in the wired network and provides the ability to set different priorities on gateways with regard to the forwarding of multicast traffic. Moreover, OLSR is used as a unicast routing protocol in the MANET, the MANET itself is based on an 802.11 Ad-hoc network and the wired network is based on Ethernet and IP.

Sammendrag

Det finnes mange multikastruteprotokoller for Mobile Ad-hoc Nettverk (MANET) som for eksempel Simplified Multicast Forwarding (SMF), Multicast Ad-Hoc On-Demand Distance Vector (MAODV) og Multicast Optimized Link State Routing (MOLSR). Fellesnevneren for disse tre (3) er at de sprer multikasttrafikk innad i MANET, men at ingen av dem gir gatewayfunksjonalitet med tanke på tilknytning av andre typer nett. Dette innbefatter blant annet en funksjonalitet for å sammenkoble multikasttjenester i eksempelvis ett kablet Internet Porotocol (IP)-nett med en MANETs multikasttjenester.

Vi presenterer ett løsning - ved bruk av flere gatewayer - som gir denne funksjonaliteten. Løsning støtter partisjonering, linkbrudd i kablede nettverk og gir muligheten til gi gatewayer forskjellig prioritet med tanke på videresendning av multikasttrafikk. Videre er OLSR brukt som unikastruteprotokoll i MANETet, MANET er basert på 802.11 og det kablede nettverket er basert på Ethernet og IP.

Preface

This project is the original, unpublished and independent work by the author. Invaluable input and feedback have been given by supervisors Dr. M. Hauge, Dr. L. Landmark and Professor Ø. Kure during the project.

Simon Dehli
Trondheim, Norway
Juni, 2016

Innhold

Figurer

Tabeller

Algoritmer

Akronymer

AIMF Arbitrated Interior Multicast Forwarding.

AODV Ad hoc On-demand Distance Vector.

BGMP Border Gateway Multicast Protocol.

BGP Border Gateway Protocol.

BIDIR-PIM PIM Bidirectional Mode.

CBT Core Based Tree.

DPD Duplicate Packet Detection.

GW Gateway.

H-DPD Hash Based DPD.

I-DPD Identification Based DPD.

IGMP Internet Group Management Protocol.

KN Kablet node.

MANET Mobile Ad-hoc Network.

MASC Multicast Address-Set Claim.

MBGP Multiprotocol Border Gateway Protocol.

MLD Multicast Listener Discovery.

MN MANETnode.

MOSPF Multicast Extensions to OSPF.

MSDP Multicast Source Discovery Protocol.

NAT Network Address Translator.

OLSR Optimized Link State Routing Protocol.

OSI Open Systems Interconnection Basic Reference Model, OSI BRM.

PIM Protocol independent Multicast.

PIM-DM PIM Dense Mode.

PIM-SM PIM Sparse Mode.

PIM-SSM PIM Source-Specific Multicast.

RFC Request For Comments.

RPF Reverse Path Forwarding.

SMF Simplified Multicast Forwarding.

SPT Source Spesific Three.

TTL Time-To-Live.

Kapittel 1
Introduksjon

1.1 Motivasjon

Multikasting [23] [2] er trolig dataspredningsmetoden som speiler ende trafikken i militære nettverk i høyest grad. Disse endenettverkene er som oftest av Mobile Ad-hoc Network (MANET) [42] type. Det har vært en rivende utvikling innenfor disse endenettverkene. Noe som blant annet har gitt oss muligheten til å transmittere pakkedata og dermed ikke bare taletrafikk. Denne pakkedatatrafikken har videre vært primært av unikast [72] type, mens multikasting har blitt skjøvet til side. Ettersom multikasting heller ikke bare er ende til ende trafikk så skaper det utfordringer når man kobler MANET og kablede nettverk sammen. En av utfordringene er imidlertid at topologien i et MANET lider under konstante linkbrudd, noe kablede nettverk ikke gjør i samme grad. På grunn av dette kan man ikke i samme grad bruke de samme metodene for kontrollere dataflyter i et MANET som i et kablet nett. Spesielt da med tanke på multikasting hvor eksisterende multikasteruteprotokoller ikke fungerer hensiktsmessig i et MANET . Der er laget flere multikast ruteprotokoller for MANET (eks. Simplified Multicast Forwarding (SMF) [49]). Derimot er det få multikast ruteprotokoller som gir MANET ende-rutere evne til å levere interdomene multikastingstjenester på en robust og hensiktsmessig måte .

1.2 Problemstilling

Multikast er en datarute-metode som kan være fordelaktig å bruke i et militært datanettverk ved noen anledninger. Eksempler er blue force tracking og tradisjonelle push-to-talk samband. Det fins allerede flere multikast ruteprotokoller for kablede og Mobile ad-hoc nettverk. En sammenkobling av slike nettverk med tanke på multikast trafikk er nødvendig for å la multikast trafikk entre inn i begge typer nettverk. Dette problemområdet har fått heller lite oppmerksomhet og derfor blitt lite undersøkt. Det å gi en sammenkoblingstjeneste for multikast, ved hjelp av flere rutere, kan kanskje øke robustheten ved videresending av multikastet trafikk.

Kandidaten er gitt oppgaven å designe, utvikle og teste en multikast sammekoblingstjeneste ved hjelp av to eller flere rutere, som knytter kablede og trådløse nettverk sammen med tanke på multikast trafikk. Tilpasning av eksisterende protokoller kan utføres. Produsenteide protokoller kan brukes hvis nødvendig.

1.3 Lignende Arbeider

Som nevnt er det gjort relativt lite med tanke på MANET interdomene/interprotokoll multikastruting, men det er gitt noen forslag til hvor det brukes, eller kan brukes, to eller flere gateway'er (GWer):

1. **MANET Multicast with Multiple Gateways** [20]. utledet av Claudiu Danilov, Thomas R. Henderson, Phillip A. Spagnolo, Thomas Goff og Jae H. Kim.
2. **Connecting MANET Multicast** [12]. utledet av Ian D. Chakeres, Claudiu Danilov, Thomas R. Henderson og Joseph P. Mackert.
3. **A Simplified Approach to Multicast Forwarding Gateways in MANET** [46]. utledet av Yannick Lacharité, Maoyu Wang, Louise Lamont og Lars Landmark.
4. **Multicast Forwarding using Multiple Gateways and Hash for Duplicate Packet Detection in a Tactical MANET** [47]. utledet av Lars Landmark, Yannick Lacharité og Louise Lamont.

Under konsept nummer 1 er det laget en løsning lik den løsningen som blir lagt fram i denne oppgaven. Man kan si at vi velger å implementere en lignende løsning. Her er det lagt frem et scenario hvor man bruker tre (3) GWer som sammenkobler ett MANET med ett tradisjonelt kablet nett. Disse GWene inkorporerer multikastruteprotkoller som er brukt i det kablede nettet samt de i MANETet. Videre bruker hver GW multicast interesse, GWers MANET grenesnitt tilgjengelighet og annen lokal informasjon til å bestemme lokal videresendingstilstand. Det er videre uklart hvordan dette kommuniseres, men det sies at det utarbeides en «distribuert» redundans-variabel som styrer hvilken GW som videresender, men altså ikke hvordan den er kommunisert. De sier heller ikke noe om hva som skjer hvis kildestrømmen opphører grunnet linkbrudd i det kablede nettet.

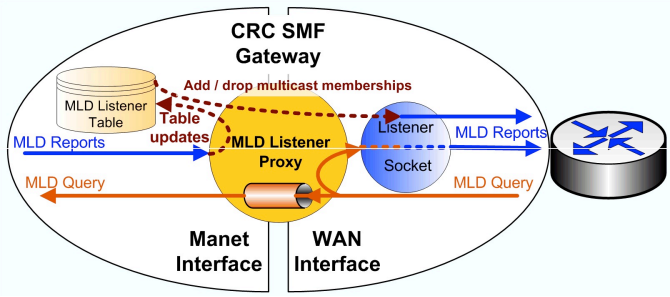
I selve MANETet er Simplified Multicast Forwarding (SMF) [49] brukt.

Under konsept nummer 2 er det laget en litt simplere løsning en gitt i konsept nummer 1. Det brukes SMF i selve MANET, det brukes en unikastruteprotokoll

slik at SMF kan kalkulere ett CDS på en god måte, og i det kablede nettet er det brukt en tradisjonell multikastruteprotokoll. I motsetning til konsept nummer 1 er den ingen informasjonsutveksling mellom GWer vedrørende partisjonering og/eller multikast interesse. Så multikastpakke-duplikatproblematikk vil bli løst av SMF her. Så som en følge av dette vil det forekomme duplikater i område mellom GWer og 1-hops MANET-noder. Det kan ha små eller store konsekvenser sett ut fra den flyktige topologien til MANET. Det skal sies at de forslår løsninger for å løse dette.

Konsept nummer 3 presenterer en løsning hvor man lager en bro mellom SMF og multikastruteprotokollen i det kablede nettet samt en Multicast Listener Discovery (MLD) [22] proxy slik at mulikastinteressen blir videreformidlet fra MANETet til det kablede nettverket. Denne proxy-funksjonaliteten er gjengitt i figur 1.1. I selve MANETet er Simplified Multicast Forwarding (SMF) [49] brukt. Det er videre lagt frem en løsning for bruk av 1 eller flere GWer.

I selve MANET er er Simplified Multicast Forwarding (SMF) [49] brukt.



Figur 1.1: MLD-proxy detaljer. [46]

Konsept nummer 4 ser på ytelsesforskjeller mellom to (2) forskjellige implementasjoner av Duplicat Packet Detection (DPD) [49] funksjonalitet. Den ene implementasjonen er basert på hashing og den andre er basert på en «taggerId». Den første varianten legger en hashverdi inn i «identifier-feltet» i pakken som videresendes, mens den andre legger til et (kilde,sekvens)-felt par på pakken (Det førstnevnte feltet er mindre enn det sistnevnte). De viser til at hash varianten presterer likt den andre metoden.

En tredje metode er også lagt frem hvor man instruerer ett subset av MANET

noder til å bare «høre» på en bestemt GW, men denne «mappingen» er statisk og vil fungere dårlig ved partisjonering og er her brukt som ett sammenligningsgrunnlag.

1.4 Metode

Denne oppgaven består av tre deler; hendholdsvis en teori-, en implemetasjons- og en resultat del. Teori delen gir en innføring i de standarder som gjelder for multikast i dag og i de verktøy som er brukt i implementasjonen. Implementasjonsdelen gir ett innblikk i hvordan multikastruteprotokollen er implementert; altså design, arkitektur og valg av verktøy og rammeverk. Til slutt, i resultatdelen er det endelige arbeidet lagt frem og validert. Her er en liste med delmål for arbeidet.

1. Problemtsillingen er beskrevet.
2. Har oversikt over tidligere utført arbeid innen inter- og intradomene-mulikast og MANET domenet og kjenner til gjeldende praksis.
3. relevant teori som dekker punkt 2 er lagt frem.
4. Et adekvat rammeverk er valgt som videre skal brukes til å simulere og skape nevnte multikastruteprotkoll.
5. Implementasjonen er testet ved hjelp av tester i gitt rammeverk.
6. Evaluer funksjon og finn resultater.

1.5 Omfang og Avgrensninger

> Kandidaten er gitt oppgaven å designe, utvikle og teste en multikast sammekoblingstjeneste, ved hjelp av to eller flere rutere, som knytter kablede og trådløse nettverk sammen med tanke på multikast trafikk. Tilpasning av eksisterende protokoller kan utføres og produsenteide protokoller kan brukes hvis nødvendig. (Fra seksjon 1.2)

Utledet fra problemstillingen skal det lages en mulitkastruteprotokoll som gir multikasttjenester til ett MANET på en robust måte. Denne protokollen skal så testes og resultater skal legges frem. Det vil ikke være IPv6-støtte i protokollen som leveres.

1.6 Struktur

Oppgaven har følgende struktur:

Kapittel 2 gir en innføring i multikast. Da både i kablede IP-nettverk og MANET.

Kapittel 3 vil beskrive design og arkitektur gjeldene vår egenutviklede protokoll Arbiterated Interdomain Multicast Forwarding (AIMF).

Kapittel 4 gjennomgår tester av implementasjonen.

Kapittel 5 deler erfaringer rundt implementasjon og design.

Kapittel 6 inneholder diskusjon og konklusjon.

Kapittel 2
Teori

Dette kapittelet vil gi leseren ett sammendrag av den viktigste teorien rundt multikast som tjeneste. I tillegg vil det det bli gitt er innføring i de verktøyene som er brukt for å implementere selve ruteprotokollen.

2.1 Multikast metoden

Multikasting er en av flere dataflytmetoder. Den skiller seg ut ved at en multikastadresse - i motsetning til en unikastadresse - ikke er låst til en bestemt lokasjon. Multikast bruker adresser til adressering av pakker, men adressen blir tolket av nettverket til å være en «peker» på en gruppe av lokasjoner, eller unikastadresser. Hvis vi ser på kringkastingsmetoden [52] ligner denne i form av at den ikke er låst til en bestemt lokasjon og at den bruker en adresse for adressering av en gruppe av mottakere. Det som til slutt skiller multikasting og kringkasting er på måten disse datatransmisjonene blir behandlet i selve nettverket. Nettverket består da av rutere, svitsjer og eventuelt huber. Kringkastet data blir typisk ikke videresendt av en ruter[1], og svitsjer og huber videresender dataene til alle aktive porter. Ved multikastet data vil en ruter videresende dataene hvis den på forhånd er «fortalt» at den skal gjøre det. Det samme gjelder for smarte[2] svitsjer , mens huber vil behandle det som kringkastet data. Man kan si a kringkasting sender til alle, unikast sender til en bestemt mottaker og multikast er et sted mellom disse. Multikast sender til til en gruppe med mottakere, og multikastruting løser denne oppgaven.

Multikasting ble først skikkelig gjennomgått i doktorgradsavhandlingen til S. Deering [2] [81]. Og den første store implementasjonen av multikasting ble realisert ved MBone [48] prosjektet i løpet av 1992 [81]. To år senere - i 1994 - ble det til og

[1] Denne regelenkan unnvikes ved tilfeller hvor en DHCP-klient befinner seg i ett annet nettverk enn DHCP tjeneren. [67] [24]

[2] Her menes det svitsjer som støtter Internet Group Management Protocol (IGMP) [11] (Multicast Listener Discovery (MLD) [22]) snooping.[13]

med multikastet en sanntidskonsert av Rolling Stones[3] over MBone [87]. Det var ingen god kvalitet men fremtidsutsiktene den gangen for multikasting var store. For å «nå» Mbone med multikastet trafikk ble det brukt tunneler[4] til å frakte multikastet data over de delene av Internett som ikke støttet multikastet data. Altså ble tunnelene brukt til å pakke inn multikastet data inn i en unikastet pakke der hvor nettverk som ikke støttet multikastruting måtte traverseres. Slik oppnådde man «øyer» av multikast støttede nettverk rundt om på Internett. Øyene hadde intradomene-multkasttrær internt og mellom øyene ble det laget interdomene-multikasttrær. Videre utover 90-tallet ble multikast basisnettene Abilene [68] og vBNS [68] utviklet som igjen utvidet MBone [81]. [81] [88]

2.1.1 Roller i multikast

Innunder multikast har man begrepene:

Kilde Data som er multikastet har en kilde i form av en node hvor dataene stammer fra på lik linje med unikastet data.

Mottaker En mottaker er en node i et nettverk som *ønsker* å motta multikastet data.

Gruppe En gruppe er ett sett med mottakere som er gjort kjent i ett til flere nettverk. Kilden trenger da ikke å vite hvem mottaker/ene er .

Eksempler på slike inndelinger er meglerhus, eller børser. En gruppe kan bestå av meglere som vil følge med på obligasjonskurser og en annen kan være for de meglere som er interessert aksjekurser i fiskeindustrien. Kilden kan deretter være en tjener som sender ut en liste med oppdaterte kurser. Dette blir et tilfelle av enveis kommunikasjon hvor det er ikke noe i veien for at meglerne også er aktive kilder i gruppen. Da har de for eksempel mulighet til å spre egen informasjon som omhandler kursene. Et annet eksempel som understreker en stor fordel med multikasting er dynamisk gruppeinnmelding. Se følgende eksempel: La oss si at en organisasjon har plassert ut MANET noder i en utilgjengelig fjellheim for å foreta målinger av været. Organisasjonen som står for sensornettverket har flere ansatte som er interessert i måledataene. På forhånd har man satt opp nodene til å sende sine måledata til

[3] De første til å multikaste en direktesendt konsert var bandet kalt Severe Tire Damage. Dette ble da gjort i 1993.

[4] En tunnel er en metode for å tilslutte noder på en logisk måte. Dette gjørs eksempelvis når eldre infrastruktur skal traverseres. Måten man gjør dette på er å legge en ny IP-header oppå den allerede eksisterende IP-headeren hos en pakke. Konsekvensen av dette er at rutere vil sjekke den nye IP-headeren i stede for den gamle. Når denne pakken ankommer sin destinasjonen - som er bestem at den nye IP-headeren - blir den nye IP-headern plukket av og pakken blir sendt videre som multikastet data.

en multikastgruppe. De ansatte melder seg så bare inn i denne gruppen når de er interessert i motta dataene. Og når de er ferdige med eventuelle kalkulasjoner melder de seg ut. I teorien vil det da aldri brukes unødvendige ressurser i nettverket og det vil være forholdsvis enkelt å legge til nye sensorer.

Denne inn -og utmeldingen av mottakere i grupper har IGMP for Internet Protocol versjon 4 (IPv4) [72] og MLD for IPv6 [35] ansvaret for. Disse to bli vil gjennomgått i seksjon 2.2.1.

2.1.2 Multikastadresser

En IPv4 adresse består av 32bit. Denne adressen består igjen av 4 oktetter og hver oktett kan viselig holde en desimal verdi på opp til 255. Disse adressene er delt opp i såkalte adresse-rom klasser (A, B, C, D, E). Denne inndelingen brukes normalt sett ikke til praktiske formål (eks. ruting[5]), men til å bestemme bruksformål for gitte klasser. Nå deles disse adressene opp ved hjelp av IPv4 adresse prefikser A/N [31] slik at man kan få tilgang på flere adresse-rom. Hvor A er IPv4 nettadressen og N er et desimaltall som forteller mange bits som er satt til 1 i masken fra venstre (MSB) til høyre (LSB). Eksempelvis klasse A adressen 126.168.0.0/14 vil gi en «frittroms-maske» lik 0.3.255.255. Det innebærer at noder som er tilsluttet dette nettet kan ta adresser i rommet 126.(168-171).(0-255).(1-254). Hvem som får bruke hvilke adresser på Internett er det Internet Assigned Numbers Authority (IANA) [37] som bestemmer. [88]

Når en IPv4 node allokerer en adresse eier den et par som består av en adresse og en maske. Adressen er den unike identifikasjonen noden har mens masken bestemmer hvilket nett noden tilhører. Ved hjelp av dette paret vil en node vite om en annen node er en del av nettet eller ei ved å gjøre en bitoperasjon på adressen til den andre noden. Hvis denne sjekken er ok - altså at begge nodene befinner seg i sammen nett - kan datagrammet sendes direkte, mens i motsatt fall må datagrammet sendes til en ruter. Denne ruteren vil da ta ansvaret for å få videresendt datagrammet.

Når det gjelder IPv4 multikastadresser må de første 4 bittene i den første oktetten i IPv4 adressen være satt til 1110. Dette innebærer at en multikastadresse har en «frittroms-maske» lik 15.255.255.255 som videre gir et adresse rom på $(224-239).(0-255).(0-255).(0-255)$. Grunnen til at den siste oktetten ikke går fra 1 til 254, men fra 0 til 255, er at multikast-ruting tjenester ikke trenger nettadresse eller kringkastnings-adresse. Dette rommet av adresser ligger innunder klasse D [17] [45]. Inne i klasse D rommet er det mindre IPv4 adresse rom som har særskilte bruksformål, men ikke funksjonsformål. Adresser fra 224.0.1.0 til 238.255.255.255 er kalt «globally scoped addresses». Dette rommet med adresser bør brukes hvis multikastet trafikk

[5]Derav ordet «classless routing».

skal traversere flere autonome systemer og/eller Internett. Adresser som er i rommet 234.0.0.0/8 er reservert for *Kilde Spesifikk Multikast*, eller bedre kjent som Source Spesific Multicast (SSM) [36]. Til slutt har vi adresserommet 239.0.0.0/8 som er kalt «limited -» eller «administratively scoped addresses». Disse adressene bør brukes i lokale sammenhenger - la seg være ett bediftsnettverk eller i ett Local Area Network (LAN). En IPv4 node som ønsker å sende multikastet data setter destinasjonsadressen til å være en multikastadresse, mens kildeadressen settes til den allerede tilordnende unikastadressen. [45] [81] [88]

For IPv6 er det ikke store forskjellene når det kommer til multikastadresser på ett overordnet nivå. Her må den første oktetten være satt til $0xFF$ (11111111) som er en tilnærming av designvalget som er tatt i IPv4 hvor de 4 første bittene må være satt til 1110 i den mest signifikante oktetten. Andre adresser som bryter med dette er da unikastadresser.

2.1.3 Multikast og logiske trær

Trebaserte metoder distribuerer data langs logiske trær. Et multikastdistibusjonstre - som en multikastruteprotokoll er ansvarlig for - vil sjelden gjenspeile det fysiske nettverket (eks. kobberkabling eller fiberoptisk kabling) ettersom kilder og mottakere varierer [81]. Det er flere slike logiske trær og også helt andre metoder som ikke er trebasert, eksemplevis «full-mesh»-metoden, men det ses her på Steiner trær og videre de to distinkte trærne delte - og kildetrær. Vi begynner med å vise en viktig faktor ved multikasting ved å beskrive hvordan multikast flyter i ett nettverk og skaper et multikasttre:

Node 5 i figur 2.1 ønsker å sende en strøm med data med båndbredde b til m mottakere. Ved unikasting vil da node 5 sende en kopi av strømmen til hver av de m mottakerne ettersom hver IP-pakke bare kan holde en enkelt destinasjons adresse. Unikast er den metoden som antageligvis brukes desidert mest på Internet i dag. En annen metode er å lage ett tre som har roten i node 5 og som brer seg ut til de m andre nodene. Kopiering av strømmen vil skje der hvor treet forgreiner seg, og ikke ved kilden (node 5). [81]

Hvis vi sier at node 5 er kilden, node 6, 7, 8 og 9 er mottakere og vi antar at korteste vei fra node 1 til 4 er via node 3. Ved unikasting vil da node 5 måtte sende 4 kopier av stømmen og lasten på linken mellom node 5 og 1 (1,5) vil da være $4b$. Link (1,3) il være $3b$, link (1,2) vil være b og link (2,4) vil være 0. Lasten på mottakertilknyttende linker vil følgelig være b og resterende linkers last vil være lik 0. Ved multikasting vil derimot ikke node 5 lage noen kopier så link (1,5) vil ha last b. Ved node 1 støter man på en forgrening slik at en kopi lages per gren. Kopien blir deretter sendt videre på hver nye gren. Altså lasten på (1,2) og (1,3) blir da b. Node 3 har bare en mottaker tilsluttet og lasten på link (3,8) blir dermed b. Denne

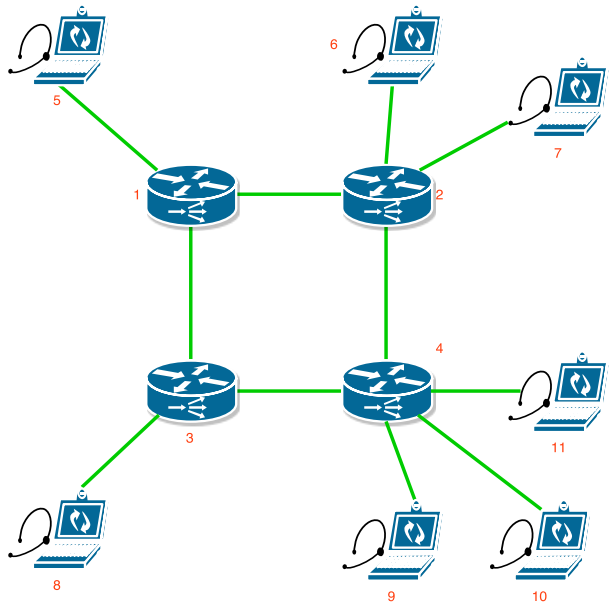
logikken gjelder også de andre mottakerne. Oppsummert vil alle grenene i treet alltid ha en last lik b og de grenene som ikke er med i treet vil ha last 0. Ut fra dette kan man se at det kan være mye å hente med tanke på båndbreddebesparelser. [81] [88]

Over ble det forklart hvordan trafikk traverserer et multkasttre. Det er ett problem som oppstår ved forgreninger: På hvilke linker skal kopiene sendes? Eksempelvis vil man helst ikke at en kopi skal sendes tilbake på den linken hvor dataene kom fra. Ved forbedret flooding[6] så sjekker hver node om den har mottatt en gitt pakke fra en direkte tilsluttet node før. Hvis pakken har blitt sett før så kastes den, hvis ikke blir den videresendt. Men dette løser ikke problemet hvis det er flere veier tilbake til kilden, og man samtidig etterstreber å ha en trestruktur. Reverse Path Forwarding (RPF) [19] løser sistnevte problem ved å sjekke opp kildeadressen mot den sameksisterende unikast rutetabellen. Ankommer en multikastet pakke fra kilde k på ett grensesnitt hos ruter r og det viser seg at dette grensesnittet er en del av den korteste veien fra r til k så er sjekken ok. Videre vil da r videresende pakken på de grensesnittene som er satt aktiv i dens egen multikastruteprotokoll. Følgelig hvis ikke noen grensesnitt er satt aktive eller at sjekken ikke er ok, vil pakken bli kastet. Hvis det viser seg at to, eller flere, grensesnitt er en del av den korteste veien mellom r og k så kan man velge det grensesnittet som har lavest adresse (bittverdi) eller bruke en randomisert plukkemetode. Denne plukkemetoden kan eksempelvis implementeres slik: En «tilfelding nummergenerator» plukker ett tall mellom 1 og 100. Vi antar at det er 4 grensesnitt som er en del av den korteste veien mellom r og k. Da vil grensesnitt 1 eie tallområdet 0-24, grensenitt 2 vil eie område 25-49 og så videre. Er det tilfeldig valgte tallet innenfor tallområdet til grensesnitt 1 så blir grensesnitt 1 valgt. [81] [24] [88]

Minimum Spanning Tree (MST) og Steiner Trær

Problemet man prøver å løse når man vi lage en god måte å spre multikastet data på er å sammenkoble alle kilder $K(g)$ og mottakere $M(g)$ i en gitt gruppe g med lavest mulig linkkost og følgelig antall linker I ett nettverk X. Altså minst mulig linkkost og linksett i område $K(g) \cup M(g)$ i område X - altså ett Minimum Spanning Tree (MST). La oss is at $A_{K(g)ogM(g)} = K(g) \cup M(g)$ og vi vil ha ett Steiner Minimal Tree (SMT) som tilknytter alle nodene i submengden $A_{K(g)ogM(g)}$ i mengden X, altså ett $SMT(A_{K(g)ogM(g)})$ i X [77]. Vi vil altså at mengden $A_{K(g)ogM(g)} \cup X$ er minst mulig. Dette problemet er NP-komplett [43] [77] og denne typen trær - som man ender opp med - ble definert av Jakob Steiner, en Sveitsisk matematiker [81]. Problemet kan løses enten distribuert eller på sentralisert måte. Eksempler på metoder for å løse problemet er Prim[76] sin MST metode, Kruskal [76] sin og den heuristiske MST metoden til Kou, Markowsky, og Berman (KMB) [43] [81]. Den sistnevnte

[6] Også kalt improved flooding. Dette er en metode som SMF [49] bruker. Under SMF er implementasjonen av den kalt Duplicate Packet Detection (DPD) [49].



Figur 2.1: Multikasting

MST-metoden (KMB) lager for eksempel ett tre som ikke differensierer på kilde eller mottaker. Slike trær kalles delte trær og bli forklart i seksjon 2.1.3. [65] [77] [88]

Kildetrær

Et kildetre er et tre med rot i kilden. Når multikastet data blir sendt fra kilden vil treet vokse slik at alle mottakerne i nettverket er en del av treet. Dette treet er også minst mulig med tanke på linkkost og og linkset.[7]. Treet vokser bestandig *bort* fra noden logisk sett og funksjonen som «overser» at dette skjer er RPF. Hvis vi ser på figur 2.1: Kilden k er plassert i node 5 og strømmer multikastet data til gruppe g og mottakerne er node 8, 9 og 10. Vi antar at node 8, 9 og 10 allerede har annonsert sin interesse i denne gruppen (k, g) slik at interessen er kjent i hele nettverket. [8]. Den multikastede trafikken ankommer først node 1. Node 1 har bare tillært (k, g) tilstand på linken (1,3) og RPF sjekken er ok. Node 1 videresender dataene og de ankommer node 3. Node 3 tre har derimot tillært (k, g) tilstand på linken (3,8) og (3,4) og RFP

[7]Dette kommer an på implementasjonen, men vanligvis er det ønskelig med minst mulig kumulativ linkkost og et minst mulig antall linker.

[8]For å trekke en sammenligning tilbake til Steiner trær, så er kilden en del av $K(g)$ og mottakerne er en del av $M(g)$. Ruteprotokollen i aksjon prøver å lage et MST som løser $K(g) \cup M(g)$.

sjekken er følgelig ok. Til slutt mottar node 4 den multikastede trafikken. Node 4 har (k, g) tilstand på linken (4,9) og (4,10). [88] [81]

I dette eksempelet var den bare en enkelt kilde, men flere er mulig. Ved flere kilder så må det lagres enda ett (k, g) par per link per node og dette kan koste mye ressurser. En multikastruteprotokoll kan allokere færre ressurser ved å bruke delte trær, men ved bruk av kilde trær kan man trolig forvente høyest ytelse [9]. Protocol Independent Multicast - Source Spesific Multicast (PIM-SSM) [36] og Protocol Independent Multicast - Dense Mode (PIM-DM) [5] bruker eksempelvis kildetrær.

Delte Trær

Et delt tre er ett distribusjons tre som er delt mellom alle kilder i en gruppe ($K(g)$). Det delte treet ble først definert av D.W. Wall [89] [81]. Denne definisjonen ble videre grunnlaget for Core Based Trees (CBTs) som ble definert av Ballardie, Francis, and Crowcroft [7] [81]. Denne metoden er også brukt i den mye brukte Protocol Independent Multicast - Sparse Mode (PIM-SM) [28] ruteprotokollen. Den annen relevant ruterprotokoll vi nevner her, som også bruker delte trær, er Bidirectional Protocol Independent Multicast (BIDIR-PIM) [33].

I motsetning til kildetrær så kan man si at i delte trær aggregerer man alle multikaststrømkilder k i gruppe g sammen til en og samme kilde og lokasjon som vi kan kalle en *fusjonert kilde* $FK(g)$, eller bare FK [10]. Fra FK bygges det igjen et kildetre som innlemmer alle interesserte mottakere. Hvis vi ser på figur 2.1: Kilden k er plassert i node 5 og strømmer multikastet data til gruppe g og mottakerne er node 8, 9 og 10. Vi antar at node 8, 9 og 10 allerede har annonsert sin interesse i denne gruppen g slik at interessen er kjent i hele nettverket. Node 2 er satt til å ha en FK funksjon og alle nodene er kjent med at node 2 har denne funksjonen. Vi antar også at korteste vei mellom node 2 og 3 er via 4. Den multikastede trafikken ankommer først node 1 fra node 5. Node 1 registrer at den mottar multikastet data og videresender den til node 2 ettersom denne node holder tilstanden FK. Node 2 videresender så trafikken til node 4 ettersom node 4 viser en arvet interesse for gruppe g ($(*, g)$) (RPF sjekken er ok). Node 4 har igjen $(*, g)$ tilstand på link (4,9), (4,10) og (3,4). Til slutt ankommer dataene node 3 som igjen videresender trafikken til node 8. [45] [81] [88]

Dette FK punktet - kalt *Rendezvous point* (RP) i PIM og *core* i CBT - bekjemper ressursbruken i nettverkt betraktelig når det er flere kilder, men hindrer også mottakere i å differensiere dem. Ved bruk av delte trær ser alltid mottakere bare en enkelt kilde og dermed gir det ingen mening for mottakere å melde kilde spesifikk interesse per gruppe.

[9] Ytelse i kb/s. Det da typisk på grunn av dataen vil utføre færre hopp mellom kilde og mottaker
[10] En FK kan holde flere forskjellige grupper, mens i $FK(g)$ er gruppen spesifisert

For å utdype (k, g) og $(*, g)$ bedre med tanke på ressurser:

(k, g) - Dette kreerer «kilde per gruppe per link per ruter» tilstand.

$(*, g)$ - Dette kreerer «gruppe per link per ruter» tilstand.

Ett eksempel på forskjell i ressurs bruk: Vi sier at vi har 40 kilder og 2 grupper. Hver gruppe har 20 kilder hver. Vi har 1 stykk ruter. Denne ruteren har vidresendingstilstand på alle kilder og grupper (Ruteren har to grensesnitt med alle kildene tilkoblet ett av dem).

Ved kildetre: (IPv4 adresse for kilde (32bit) * 20 * IPv4 adresser per gruppe (32bit) * 2) + grensenittid(1-255 grensesnitt = 32bit) + ruterid (16bit[11]). Man kan jo si at det alltid er bare en instans av ruterid, men ved flere instanser av multikast-ruteprotokoller per fysiske ruter blir det fort aktuelt å regne med denne. For eksempel ved bruk av blant annet Multicast Virtual Private Networks (mVPNs) [80] [78] og/eller Multicast Virtual Routing and Forwarding (mVRF) [79] funksjon[12] . Kumulativ tilstand blir $(32bit * 20) * (32bit * 2) + (1 * 32bit) + (1 * 16bit) = 41008bit \approx 41kb$.

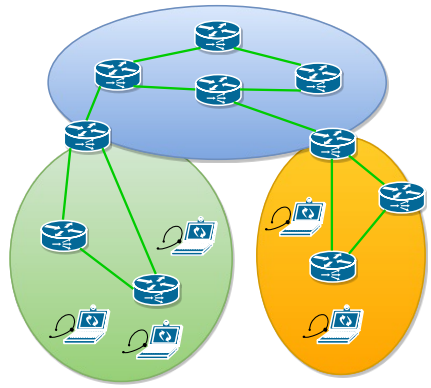
Ved delt tre: (IPv4 adresser per gruppe (32bit) * 2) + grensenittid(32bit) + ruterid (16bit). Her blir da den kumulative tilstanden $41008bit - ((32bit * 2) + (1 * 32bit) + (1 * 16bit)) = 40896bit$ mindre, eller 997,26‰ mindre, enn den ville ha vært ved bruk av kildetre.

[11] Vi antar i eksempelet at det autonome systemnummeret er i bruk til å representere selve ruteren. Multiprotocol Border Gateway Protocol Version-4 (MBGP-4) [8] bruker 16 bit til å representere dette nummeret.

[12] Dette er en veldig forenklet fremstilling av tilstandens størrelse. Ut i fra denne rapporten [14] fra Cisco System kan man få en mer riktig fremstilling på hvilke data som lagres ved bruk av delte trær

2.2 Multikastruteprotokoller

Multikastruteprotokoller blir delt inn etter i hvilket domene de opererer(se figur 2.2). Til eksempel opererer PIM-SM og PIM-DM i intradomenet. Et intradomene inkorporerer vanligvis kun ett autonomt system (AS) med tanke på unikast - la seg være ett OSI Intermediate system to Intermediate system intra-domain routing information exchange protocol (IS-IS) [66] AS eller ett Enhanced Interior Gateway Routing Protocol (EIGRP) [83] AS. I ett interdomene opererer eksempelvis multikastruteprotokollene Multiprotocol Border Gateway Protocol 4 (MBGP) [8], sammen med Multicast Source Discovery Protocol (MSDP)[13] [30], eller Border Gateway Multicast Protocol (BGMP) [86] sammen med Multicast Address-Set Claim (MASC) [73]. Med tanke på unikast blir nesten alltid Border Gateway Protocol (BGP) [75] brukt i interdomenet. Interdomeneprotokoller knytter intradomeneprotokoller sammen (Se parallellen til mBone i seksjon 2.1). [24]



Figur 2.2: Interdomene- og intradomenemultikast. I det blå omrisset opererer interdomeneprotokoller, mens i de resterende omrissene opererer intradomeneprotokollene

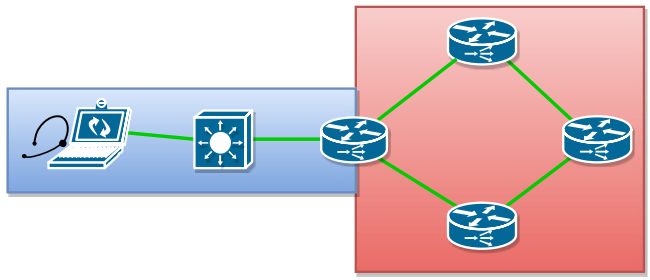
2.2.1 Multikastruteprotokoller for bruk i intradomenemultikast

Protokollene som blir gjengitt her har alle en visjon om lage ett MST, men de differensierer seg med tanke på hvilke tretype (se seksjon 2.2) som blir brukt. Først blir IGMP og MLD gjennomgått, fordi abstrakt sett er disse to protokollene budbæreren mellom multikast interessen i endene av ett nettverk og nettverkets multikastruteprotokoll. Helt til slutt blir SMF gjennomgått som er en MANET multikastruteprotokoll. SMF

[13]MSDP baserer seg at multikastruteprotokollen i intradomenet bygger delte trær, fordi den i praksis utveksler RP- eller kjerne-tilstand mellom domener. (for eksempel PIM-SM)

bruker enten en «full-mesh»-metode for å tjene MANETet med multikast, eller den kan bruke en hybrid av delt tre-metoden og «full-mesh»-metoden for å oppnå det samme. SMF vil bli gjennomgått i seksjon 2.3.1.

IGMP og MLD



Figur 2.3: Operasjonsområde for IGMP (blå) og intradomene-multikastruteprotokoller (rød).

IGMP og MLD brukes til å styre og kontrollere multikastgrupper. Og de er noen av de svært få protokollene i dag som lar noder melde inn, eller av, interesse for gitte multikastgrupper.

Multikast som en tjeneste må også ha en funksjon som melder fra om inn- og utmeldinger av mottakere. Interdomene-multikastruteprotokoller har til nå ikke blitt gitt noen metoder for å få lære seg hvilke mottakere som er aktive til en hver tid. Så for å gi de denne vitale informasjonen er IGMP og MLD inkorporert inn i henholdsvis IPv4 - og IPv6 stakken. Så IGMP er implementert i lag 2 i OSI modellen og MLD på lik måte med tanke på IPv6. MLD er ett derivat av IGMP v3 protokollen - en for hver IP versjon. IGMP(fra nå menes det også MLD når det blir referert til IGMP) lever både i endenoder og i vianoder(rutere) (Se figur 2.3). Rutere som kjører IGMP har en elevert rolle hvor de vedlikeholder en tilstand som innbefatter kontinuerlig kontroll og styring av multikastgruppeinteresse per tilknyttede nettverk. I tillegg er de ansvarlig for å varsle den sameksisterende mulikastruteprotokollen når det forkommer endringer i den sistnevnte tilstanden. Endenoder derimot, bruker IGMP til varsle i fra når de er interessert og når de ikke lenger er interessert[14].

Dataen som trengs for å melde interesse for en multikastgruppe er (kildeadresse, gruppeadresse) for kildetrær og (*, gruppeadresse) for delte trær. Denne dataen

[14]Dette er ikke helt likt implementert i IGMP v1 og v2 og IGMP v3.

kan hentes fra en nedlastet .m3u fil eller skaffes igjennom bruk av Session Initiation Protocol (SIP) [82], eller ved bruk fra Session Announcement Protocol (SAP) [34]. SAP annonserer multikastgrupper ved å multikaste metadata vedrørende gitte multikastgrupper i en enkelt antatt kjent multikastgruppe. Så ved å vite SAP-multikastadressen vil man gjennom denne eventuelt lære seg det/de (k, g) par/ene, eller $(*, g)$par/ene, man er ute etter.

Når det gjelder svitsjer og IGMP så støtter noen svitsjer IGMP snooping [13]. Denne funksjonen lar svitsjer «lytte» på IGMP-trafikk og gir dem dermed evnen til å tilegne seg informasjon om hvilke porter som skal få multikastet trafikk og ikke. Uten denne funksjonen behandler svitsjer multikastet trafikk som om den var kringkastet, noe som man selvfølgelig vil unngå. [24]

Et annen velbrukt funksjon innenfor IGMP-verdenen IGMP-proxying [29]. Ettersom det over er gitt at en enkelt IGMP-melding lever i kun ett kringkastingsdomene så gir dette problemer ved utrulling av multikasttjenester. Nesten alle forbrukerrutere, eller forbukerrutere hvor IP-TV er aktuelt, implementerer IGMP-proxying og Network Adress Transelator (NAT) (Eller Port-NAT ved flere offentlige adresser.) [92], fordi det ville ha vært uforsvarlig og dyrt - sett fra en Internet Service Provider's (ISPs) ståsted - å eksponere eller utruste rutere med multikastrutetilstand til kunder. Kunder med onde hensikter ville kunne annonsere - med litt kløkt - sine egne multikast stømmer og dermed redusere ISPens QOE og eller lettere sagt tjenestekvalitet[15]. En slik ruter med disse funksjonene vil da videresende IGMP-meldingen videre til den «sikre» infrastrukturen hos ISP hvor selve multikastruteprotokollen/ene lever. [24]

Multikasttremedlemskap

Når en IGMP-, eller MLD-node har meldt sin intresse til næreste 1-hopsruter; hva skjer så? Det er to distinkte metoder som brukes for å bli medlem av ett multikasttre; *Push* og *Join* [81] [91] [45].

Push beskriver en metode hvor man ved intervaller kringkaster i utgangspunktet multikastet trafikk i et enkelt intradomene. Hver ruter, eller vianode - for å holde seg til definisjonen, er ansvarlig for kringkastingen. Videre vil da alle endenoder i det gitte domenet - etter gitt stund - være kjent med trafikken, men det er sjeldent at alle endenoder er interessert - denne trafikken er «tvunget» på dem. Så ved uønsket trafikk *beskjærer* (to prune) vianoder linken sin logisk sett ut av «kringkastingsdomenet» hvis det ikke meldt noen interesse hos nedstrøms-endenoder. En vianode vil bare beskjære sin oppstrøms-link hvis, og bare hvis, det er ingen aggregert interesse nedstøms. Når denne prosessen er gjort for alle vianoder vil man stå igjen med ett

[15] En kunde kunne ha «tatt» over multikastgruppen for NRK1

multikast tre, og ikke et heldekkende «kringkastingstre». Multikastruteprotokollen Protocol Independent Multicast - Dense Mode (PIM-DM) [5] bruker denne metoden. Denne metoden passer til bruk hvor interessen for å motta multikastet trafikk vanligvis gjelder brorparten av ett sett med noder i ett intradomene - derav ordet *dense* («tykt», eller «flust» med interesse). [81] [45]

Join beskriver en metode som er langt fra så aggressiv som *push*. Under push-metoden ser man at endenoder ikke trenger å ha kjennskap til multikastgrupper. Ved bruk av join-metoden må derimot endenoder ha kjennskap til dette. Innunder denne metoden må vianoder «ettersøke» ønskede multikastgrupper ved å melde sin interesse *innover* mot kjernen av nettet. Mer detaljert så starter hele prosessen ved at endenodene melder sin interesse hos sin 1-hopsruter(vianode). 1-hopsruteren melder igjen denne «aggregerte» interessen, ved hjelp av en *join*, videre til sin 1-hopsruter og så videre helt til man støter på en RP eller en ruter med tilstand gjeldene denne multikastgruppen man ettersøker ($(*, g)$ eller (k, g)) . PIM-SM, PIM-SSM og BIDIR-PIM bruker denne metoden, men PIM-SSM har ikke en såkalt RP implementert, mens BIDIR-PIM har en «løsere» definisjon av en RP. Multikastruteprotokoller som bruker denne metoden passer som regel til domener hvor interessen er lav relativt til hvor mange noder det er- derav ordet *sparse* («tynt» med interesse). Ett unntak her kan være PIM-SSM som passer til domener hvor det er et lite antall kilder, kildelokasjonene er lite flyktige og hvor det er en stor mengde mottakere (eks. IP-TV hvor mengden $Kilder(gruppe) \ll Mottakere(gruppe)$), men hvor *push*-metoden er uønsket. BIDIR-PIM derimot passer i miljøer hvor mengden $Kilder(gruppe) \approx Mottakere(gruppe)$. [81] [45]

PIM-DM

PIM-DM bruker kildetrær. 1-hopsruteren til en kilde kringkaster kildestrømmen når den oppstår og alle resterende rutere i gjeldene protokolldomene gjør det samme slik at alle entiteter i gjeldene intradomene er dekket. Ut fra endenoders interesse og ruternes aggregerte interesse beskjærer ruterne kringkastnigstreet *bort* fra endenodene og *mot* kilden. Til slutt vil man stå igjen med ett multikasttre.

PIM-SM

PIM-SM bruker en blanding av kildetre og delt tre. Det blir opprettet en RP - plassert ut etter eksempelvis ytelses-preferanser - i det gjeldene nettverket (protokolldomenet). Fra kilden til RP blir det opprettet et kildetre. Og fra RP til endenoder - som eksplisitt har vist interesse for mottak - blir det opprettet ett delt tre. Når endenoder viser interesse for en gitt multikast gruppe g vil deres 1-hopsruter igjen «spørre», eller vise interesse, etter/for denne multikast gruppe g hos sin/e 1-hopsruter/e. Denne prosessen er tidligere forklart under seksjon 2.2.1 (Join). Når denne «rekursive»

prosessen har funnet g sendes den ønskede multikast trafikken den samme veien som prosessen brukte for «spørre» etter g. Når en endenode ikke lengre er interessert så beskjæres multikasttreet slik som kringkasningstreet blir under PIM-DM. Det skal nevnes at PIM-SM kan endre ett «kildetre->RP->delt tre» til ett enkelt kildetre per gruppe g. Dette skjer eksempelvis når multikastpakkeraten (Kb/sek) er stor for gruppe g og RP må avlastes.

BIDIR-PIM

BIDIR-PIM ligner på PIM-SM, men det er noen unntak. Først kan man anta at de som utviklet BIDIR-PIM (1) antok at PIM-SMs RP sjelden ville bli nådd (2) og hvorfor ikke bare ha kontroll på hvem som er 1-hopsruteren på veien mellom en kilde, eller mottaker, og den «imaginære»[16] RPet for en ruter. Da vil man antageligvis få en RP som tilnærmet ligger i det mest sentrale punktet i ett MST for $K(g) \cup M(g)$ noe man egentlig også får i PIM-SM, men dette har fortsatt sine fordeler. Dette er løst ved at hver ruter lagrer $(*, g)$ + $I_{id-gsnitt}$ hvor $I_{id-gsnitt}$ identifiserer det grensesnittet som er en del av unikastveien mot RP for en enkelt ruter. Hvis det er flere rutere i ett subnett så er det unikastrutekosten for veien mot RP per ruter som bestemmer (dette gjelder også for PIM-SM). Da vil de ruterne som ikke hadde lavest kost forbli passiv (de vil ikke lage noen videresendingstilstand).

Dette er alt man trenger for at RPF-funksjonen skal hindre dupliserte sendinger. Man velger altså ett «unikastpunkt»(RP) i ett gitt nettverk - per gruppe g eller for alle grupper - i den hensikt å instruere ruterne til å «lete», eller «søke», etter gruppe g langs unikastveien mot det gitte unikastpunktet (RP). Videre fungerer *join* og beskjæring likt som i PIM-SM, *men* RPet har ingen funksjon i BIDIR-PIM sett bort fra å bli brukt til å implisitt instruere ruterne i ett BIDIR-PIM domene til å velge rett $I_{id-gsnitt}$. [81] [88]

Fordelene blir da at man slipper å ha en «ekte» RP, man kan ha tilnærmet lik konfigurasjon på hver ruter og lignende.

Ved bruk av BIDIR-PIM vil man få $(*, g) + I_{id-gsnitt}$ tilstand, mens hvor PIM-SM har bare $(*, g)$ tilstand per gruppe, men en enkelt, eller noen flere, global/e RP adresse/r per ruter. Denne tilstanden er konstant - i BIDIR-PIM - med tanke på om en multikastgruppe er aktiv eller ikke[17]. BIDIR-PIM kan heller ikke alternere mot ett kildetre som PIM-SM kan[18]. Til slutt skal det nevnes at i BIDIR-PIM kan trafikk flyte begge veiene i det delte treet, derav navnet «BiDirectional» (noe det

[16] En BIDIR-PIM RP kan være en nettadresse eller rett og slett en adresse i det gjeldene protokolldomenet som er kjent av den sameksisterende unikastruteprotokollen.

[17] Tilstanden må ikke være slik; $(*, 239.1.2.3)$ + FastEthernet0/1 (eksakt), men eksempelvis noe som ligner $(*, 235 - 236.0 - 255.0 - 255.0 - 255)$ + FastEthernet0/1.

[18] Dette kildetreet er kalt Source Spesific Tree (SPT) under PIM-SM-terminologi for å unngå missforståelser.

gjør for en gitt gruppe $(*, g)$, men noe det absolutt *ikke* gjør per kilde per gruppe (k, g)). [81] [88]

BIDIR-PIM er godt dekket her inne fordi den antageligvis vil være godt egnet til å bære signaleringen til Arbiterated Inter-domain Multicast Forwarding (AIMF). Da med tanke på at i ett AIMF-system så er $K(g) == M(g)$ alltid sant. Selve trafikken som bæres fra ett kablet nett til ett MANET kan håndteres med en annen multikastruteprotokoll [85].

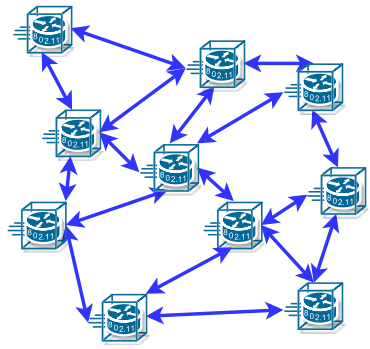
PIM-SSM

PIM-SSM er den rake motsetningen til BIDIR-PIM. Den lager bare kildetrær per gruppe g. All interesse må bli meldt med (k, g) par og alle rutere som er medlem i ett multikasttre for gruppe g har tilstand (k, g). PIM-SSM er også noen av de få multikastruteprotokollene som lar brukere vise spesifikk interesse for kilde k innunder gruppe g. Dette gir flere alternativer innenfor multikasttjenester og øker sikkerheten med tanke på eksempelvis Denial Of Service (DOS).

Statisk -, Auto - [41] og virkårlig [27] (anycast) lokalisert RP

Statisk -, Auto og virkårlig lokalisert (anycast) RP er metoder for å bestemme og for å spre RP identifikasjon (BIDIR-PIM) og faktisk lokasjon (PIM-SM).

Statisk RP impliserer at RP adressen er «hardkodet» i hver ruter. Ved Auto RP velger ruterne i ett mulikastintradomene hvem som skal være RP ved bruk av gitte variabler. Dette valget tas av en *mapping-agent*. Og til slutt ved vilkårligkastet RP så vil den intradomene unikastruteprotokollen bestemme hvilken RP som *faktisk* blir valgt til utføre RP funksjonalitet. Dette kan være en smart måte å fordele RP-last med tanke på PIM-SM. [81] [88]



Figur 2.4: Et MANET med 802.11 standard og halv-dupleks-linker.

2.3 MANET

Ett MANET er ett nettverk (se figur 2.4), eller en klynge, bestående av noder som kommuniserer trådløst med hverandre. En nodes plassering i forhold til andre noder endres kontinuerlig og da vil følgelig topologien til nettet som tilslutter noder i ett MANET være evig i endring. Gitt $\{\forall n \in M\}$ hvor n er en enkelt MANETnode og M er selve MANETet er det prinsipiellt sett ikke transitive «relasjoner» med tanke på dataflyt dem imellom likt som i ett kablet nett. Transitive «relasjonsegenskaper» blir eksempelvis gitt av unikastruteprotokoller og multikastruteprotokoller innenfor unikastflyt og multikastflyt- som også gjelder for kablede nett.

Videre har ett MANET - i følge S. Corson og J. Macker [16] - følgende egenskaper (Da relativt til et kablet nett og negative som sådan):

Dynamisk Topologi Noder har ingen rammer for sitt bevegelsesmønster og linker kan være både dupleks (to frekvenser), halv-dupleks og simpleks.

Båndbredde Båndbredden kan være relativt lav ettersom kommunikasjonen foregår trådløst og på grunn av at man anvender ett delt medium. Når det gjelder båndbredde blir det også nevnt noe viktig av de som siteres: Hva er verst ved sammenkobling av et MANET og et kablet nett; metning, konsekvenser av delt medium, interferens eller lignende? Det hevdes at metning vanligvis er det største problemet. Altså at pakkeraten (Kb/sek) i ett kablet nett normalt sett er større enn i ett MANET. Hvis man prøver å dytte denne trafikken inn i ett MANET får man problemer - man får metning. Dette har store

konsekvenser ettersom ett MANET baserer seg på ett delt medium. Ettersom man i det kablede domenet øker dataratekapasiteten for å møte etterspørselen hos endenoder og deres applikasjoner, så må man i et MANET heller prøve å redusere dataraten som blir sendt fra applikasjoner. Trådløsteknologien vil nok ikke i nær fremtid ta igjen kablede nett med tanke på dataratekapasitet. Det er dermed lettere å gjøre noe med applikasjonene enn med eksempelvis interferens, støy og/eller det faktum at det brukes et delt medium.

Liten energikilde Noen MANETnoder kan ha liten tilgang på energi(strøm i form av batterier).

Informasjons - , transmisjons- og fysisk sikkerhet MANET gir typisk ikke samme sikkerhet som kablede nett. De har trådløse linker som kan avlyttes av tredjepart. Videre er MANETnoder typisk plassert ut uten god fysisk sikring og det er *mye* større sannsynlighet for tap av data i ett MANET enn i ett kablet nett.

MANET har også noen positive sider (som er viktig i denne konteksten): [16]

Dvale ved reaktiv ruting MANET noder kan spare strøm når det ikke er aktivitet.

Geografi Ved bruk av MANET kan man dekke store geografiske områder med relativt lav kost.

Hyppige transfigurasjoner av topologi Man kan tillate seg å endre topologien i ett MANET med relativt svært lav kost.

Flere av disse er viktige faktorer innen eksemplevis «blue-force-tracking» eller annen sivil og/eller militær virksomhet.

2.3.1 Multikastruting i MANET

Det finnes flere multikastruteprotokoller, blant annet Multicast Optimized Link State Routing (MOLSR) [39] og On-Demand Multicast Routing Protocol (ODMRP) [94]. Men her inne dekker vi bare Simplified Multicast Forwarding (SMF) [49]. Primært fordi den er basert på kringkastings-basert videresending (Classic Flooding (CF)) av data og er derfor er relativt lite kompleks - altså er den simpel. Hvis det er ønskelig kan man forbedre multikast videresendingen - med tanke på kumulative båndbreddebesparelser - ved å bruke forskjellige metoder for å velge et relesett (Se seksjon 2.3.1). Sekundært fordi SMF er relativt robust når man *ikke* bruker relesett, fordi alle tilgjengelig noder får tilsendt multikastet data uansett gruppemedlemskap eller ikke (Classic Flooding (CF) [49]).

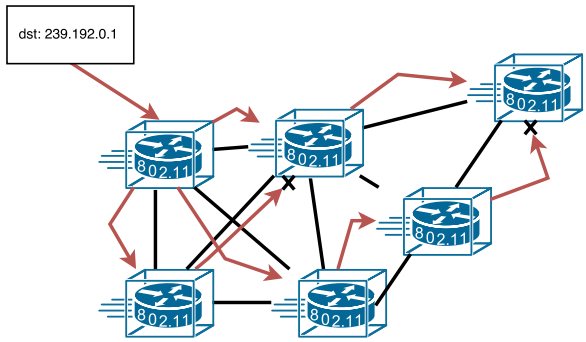
Den primære grunnen til at man eksempelvis ikke kan bruke PIM eller CBT i MANET er at noder i ett MANET bare ha ett grensesnitt og det er i tillegg ett delt medium. Av den grunn kan man ikke velge dette såkalte RPF-grensesnittet og videre vil da ikke dette multikasttreet la seg bygge. Det skal nevnes at det er flere MANET-unikastruteprotokoller som bruker RPF designet for å opparbeide topologibilde; ett eksempel er Topology Dissemination Based on Reverse-Path Forwarding (TBRPF) [64] [9]. Grunnen til at dette er interessant er at RPF her brukes til å spre topologi informasjon og primært for å lage den (hver node brygger ett kildetre der noden selv er kilden). Dette «spredetreet» - brukt til topologiformidling - blir fort ett multikasttre hvis ikke alle noder i ett MANET kjører TBRPF - og på grunn av dette opparbeidede kildetreet hver node lager.

SMF

SMF er en simpel måte å gi multikasttjenester til MANET eller å gi transitive «egenskaper» til ett MANET med hensyn på multikasttjenester[19]. Hver multikastet pakke som ankommer ett SMF-støttet grensesnitt får tilordnet en unik identitet - la seg være en hash[20] av pakken eller en annen identifikasjon[21] - og denne identiteten blir lagret i en viss periode. Dette «lageret» er implementert som et Duplicat Packet Detection (DPD) [49] cache. Så hvis en SMF-node støter på en pakke som generer en identitet lik en som allerede er lagret så vil pakken bli kastet (Se figur 2.5). Slik unngår man da looping av pakker - noe som er svært ødeleggende for et MANET og egentlig for alle typer datanettverk. I figuren som vi refererte til så antar vi at transmisjonstiden mellom hver link er lik - så to hopp tar dobbelt så lang tid som ett. Og videre antar vi at linkene indikerer hvem de ser og at de bare ser 1-hopsnoder.

SMF kan også bruke såkalte Redused Relay Sets for å minimere dette kringkastningstreet - som «bar» SMF med DPD lager - ettersom det følgelig ikke er ønskelig generelt og spesielt ikke i ett MANET. Her gir man MANETet evnen til se om en transmisjon er nødvendig eller ikke - alle noder i ett MANET videresender trafikk for hverandre. Disse settene kalkuleres ved å hjelp av å finne ett lokallinktilknytted dominerende sett (CDS) i ett MANET. Altså ett sett S med minst mulig antall noder i ett MANET som til sammen «ser» alle andre noder i ett gitt sett Y og blir «sett» av alle andre noder i Y. Ett redusert relesett S vil da bestå av mengden $\{\forall x \in Y | RS(x)\}$ hvor $RS(x)$ er funksjonen som kalkulerer dette settet og hvor Y eksempelvis er alle nodene i ett MANET. Denne $RS(x)$ funksjonen bruker typisk antallet naboer n en enkelt node x «ser» og kalkulerer en «score» ut fra dette[22]. Denne «scoren» blir

[19] Når node a når b og b når z så når a z - eksempelvis via b (multikast).
[20] Hash Based DPD (H-DPD) [49]
[21] Identification Based DPD (I-DPD) [49]
[22] $RS(x)$ funksjonen kjøres hos alle MANETnoder.



Figur 2.5: DPD-funksjon i ett MANET. [24]

formidlet av alle MANET noder. Man kan bruke eksempelvis linksett fra OLSR[23] for å fastslå om dette er en 1- eller 2-hops nabo. I Y vil det da typisk være noen noder med lav (i ytterkant av klyngen) og noen med høy (i og/eller nært sentrum av klyngen) «score». S vil da bestå av de som har en relativ høy score. Y trenger heller ikke å være «alle» MANETnoder, men MANETnoder som er interessert i en viss multikastgruppe eller har andre karakteristikker. De noder i $\{\forall x \notin S | x \in Y\}$ vil da ikke videresende data i en, alle eller flere gitt/e multikastgruppe/r,. Det fins også flere metoder for å lage slike sett eller lignende sett som blant annet Essential Connecting Dominating Set (E-CDS) og Source-Based Multipoint Relay (S-MPR) og de er definert i Request for Comments (RFC) for SMF [49].

Denne metoden for å gi multikasttjenester er ofte omtalt som «full-mesh».

[23] Disse MANETnaboene «lærer» SMF av seg selv eller den bruker en sameksisterende unikast-ruteprotokoll til å oppnå det samme (eksempelvis Optimized Link State Routing (OLSR)).

Kapittel 3
Implementasjon

I dette kapittelet vil verktøy-, design- og arkitekturvalg bli forklart.

3.1 NS3

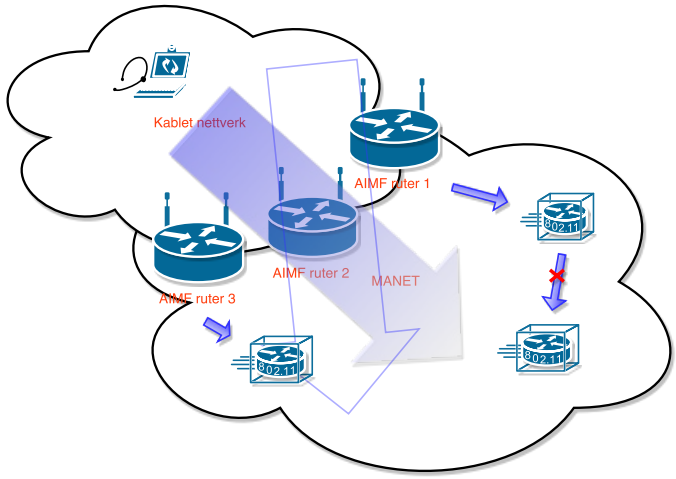
Network Simulator 3 (NS-3) [56] er en diskre eventsimulator og i praksis et rammeverk som inkluderer mange protokoller for bruk i pakkesvitsjede og linjesvitsjede nettverk. Den brukes til å simulere alt fra Wireless Local Area Networks (W-LANs) hvor noder har 802.11 [4] støtte til vanlige kobberkablede LAN hvor tilknyttede noder har alt fra Ethernet [1] støtte til IPv6 støtte, og enkel unikast ruting, da både statisk og dynamisk. Videre er støtten for Optimized Link State Routing (OLSR) [15] og evnen til å simulere MANET, MANET-noder og kablede nett med tilhørende noder viktig i denne konteksten. Sist og ikke minst er også støtten for applikasjoner som sender multikastede User Datagram Protocol (UDP) [70] pakker, og støtten for å simulere mobilitet hos MANET-noder viktig.

NS3 ble valgt fordi det dekker behovet innenfor simulering og for ett rammeverk som understøtter implementering av en multikastruteprotokoll på en god måte. NS3 gir også brukeren god kjennskap - programmatisk sett - til underliggende funksjoner som eksempelvis implementasjonen til IPv4 ettersom kildekoden ligger ved. Dette hjelper bruken når egne protokoller eller funksjoner skal implementeres.

3.2 Abiterated Inter-domain Multicast Forwarding (AIMF)

Under denne seksjonen vil vi beskrive en protokoll som håndterer IP-multikast «handover» i MANET. Vi løser kun «handover» for multikasttrafikk som flyter fra ett kablet IP-nett til ett IP basert MANET. Det vil da ikke bli sett på IGMP eller IP-multikast som traverserer fra ett MANET og inn i ett kablet nett. Det skal nevnes at AIMF har grensesnitt som vil passe fint for en fremtidig implementasjon av IGMP i ns3.

3.2.1 Design



Figur 3.1: AIMF. Multikastet trafikk er illustrert med blå piler.

Så hva er det vi vil løse? (Se figur 3.1) Vi har to svært forskjellige protokolldomener; MANET og kablet nettverk. Fra det kablede nettet vil vi utvide multikasttjenesten til og også gjelde ett eller flere MANET. Vi vil at multikastkilder i det kablede nettet skal bli «hørt» i MANETet. Det neste en ønsker er å gjøre denne utvidelsen robust. MANET har som kjent en meget flyktig topologi slik at partisjonering og linkbrudd forekommer i et meget større omfang enn i kablede nett (se seksjon 2.3). Av den grunn kan vi ikke basere oss på på eksisterende interdomene/interprotokolldomene-multikastruteprotokoller. Denne «usikkerheten» som MANET gir ved tanke på robusthet, tilgjengelighet og pålitelighet kan eksempelvis møtes med flere «inngangs-porter», (GWer) fra det kablede nettverket til MANETet. Slik at ved partisjonering og linkbrudd vil den/de best egnende GW/ene veksle på å videresende multikastet trafikk. For å spesifisere nærmere så lager vi krav til løsningen.

Dette designet er basert på en av flere skisserte løsninger for interdomene/Inter-protokolldomene multikastvideresending gitt i ett forprosjekt [24] gjort ved Norges Teknisk-Naturvitenskapelige Universitet (NTNU).

Krav

Ut i fra teksten over lager vi tre krav.

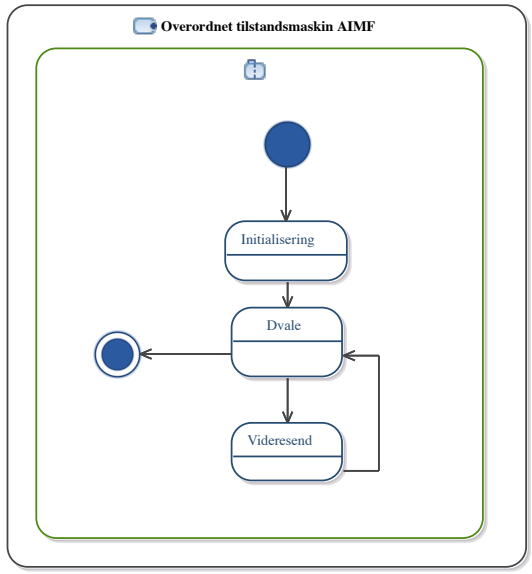
1. Man skal kunne utvide en multikasttjeneste gitt i ett kablet nettverk til og også gjelde ett MANET.
2. Punkt 1 skal oppfylles ved hjelp av 2 eller flere rutere.
3. Løsningen skal gi god robusthet og den bør være «sparsommelig» med tanke på videresending så man slipper unødvendige videresendinger inn i MANETet.

3.2.2 Arkitektur

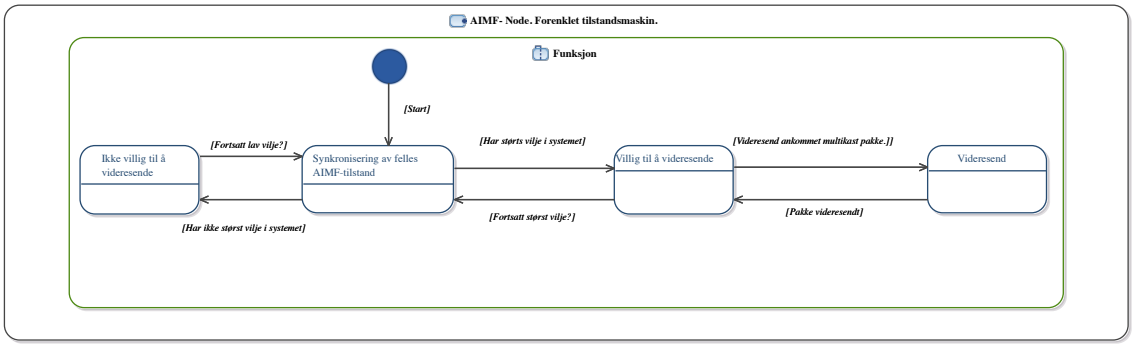
Hva må lages for å løse problemet gitt under design? Primært (1) så må det lages en videresendingsfunksjon per GW som som lar multikastet data entre MANETet. Sekundært (2) må det lages flere funksjoner som styrer disse videresendigsfunksjonene per GW på en distribuert måte slik at det eksempelvis alltid er et minst mulig sett av disse videresendingsfunksjonene som faktisk videresender. Da vil antageligvis systemet opptre sparsommelig. Videre med tanke på robusthet (3) så vil nivået av robusthet være avhengig av hvor godt disse funksjonene er implementert og hvordan de er «tunet». Grunnen til at man går for en distribuert arkitektur er at miljøet rundt hver GW vil være forskjellig med tanke på linktilstanden ut mot MANETet og lignende - noe som vil angå alle GWer i ett AIMF system. Man antar på grunnlag av dette - gitt ett mål om en robust og sparsommelig protokoll - at man må bruke en aggregert og til dels synkronisert tilstand til å bestemme videresendningstilstand per GW og at denne må formidles mellom AIMF-GWer.

Ut fra dette kan man se at man trenger visse datasett per GW til å holde denne tilstanden og man trenger funksjoner som bearbeider tilstanden med tanke på vidresendingstilstand. Videre trenger man funksjoner som formidler tilstand mellom GWene slik at de får ett helhetlig beslutningsgrunnlag med tanke på videresendingstilstand. Til slutt kan vi lage disse funksjonsdiagrammene; en for hele systemet (Se figur 3.2) og en for en enkelt node i systemet (Se figur 3.3).

Her har vi 3 delmål og den påfølgende spesifikasjonen er delt opp etter Model View Controller (MVC) [74] prinsippet. Vi starter med *modellen* hvor vi går igjennom datasettene som AIMF trenger. Neste er *kontroll* som viser hvordan man bruker datasettene definert innunder modell. Til slutt ser vi på brukergrensesnittet (view), eller som vi her kaller *grensesnitt*.



Figur 3.2: En overordnet tilstandsmaskin.



Figur 3.3: En overordnet tilstandsmaskin per AIMF-node.

Modell

AIMF trenger følgende datasett:

($kilde, gruppe$)**- eller** ($*, gruppe$)**-par** Den trenger kjennskap til hvilke multikast-grupper den skal videresende.

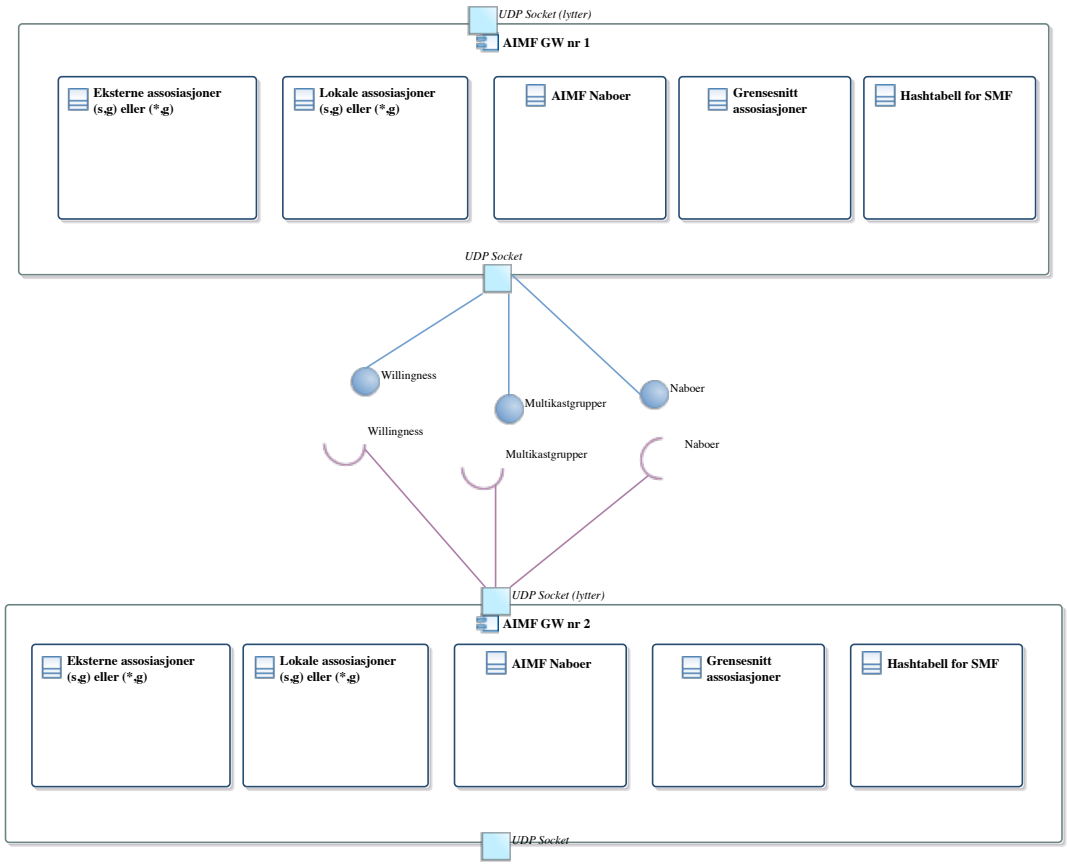
Lokale eller annonserte (k, g)- og/eller $(*, g)$-par Den trenger kjennskap til om gruppeinformasjonen - i punktet over - har ankommet lokalt eller fra en naboinstans av AIMF.

Naboer Den trenger kjennskap til sine tilgjengelige AIMF-naboer og tilhørende variable, slik at hver node kan sammenligne datasett og deretter ta en beslutning når det gjelder videresending.

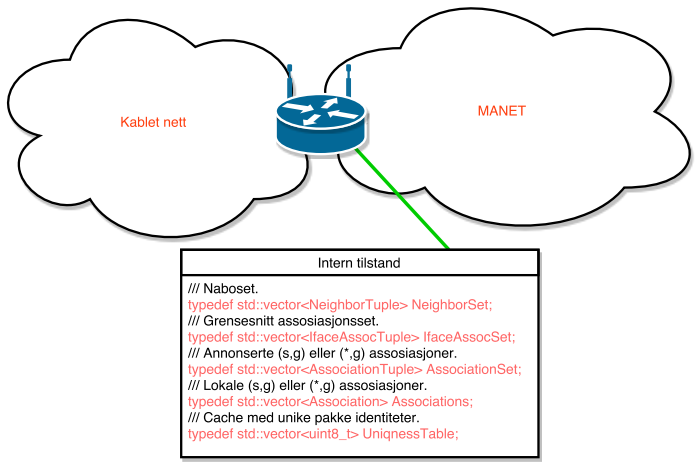
SMF Den trenger en cache med hasher over de seneste videresendte pakker for å være kompatibel med en SMF-støttede MANET-noder.

Hvilke grensesnitt råder man over Den trenger kjennskap til hvilke grensesnitt hver instans har råderett over.

Kommunikasjonskanal Den trenger en kommunikasjonskanal for å formidle sin egen tilstand.



Figur 3.4: Entiteter og grensesnitt som trengs.



Figur 3.5: Tilstand per node.

Disse punktene kan videre bli fremstilt i dette diagrammet (se figur 3.4). Her er entiteten *willingness* lagt til. «Willingness» er en tall fra 0 til 7 som beskriver en GWs vilje til å videresende trafikk. En GW som har høyest «vilje» i ett AIMFsystem vil videresende. Disse punktene er også modellert slik per ruteprotokollinstans:

```
1  /// Naboset (AIMF nabo).
2  typedef std::vector<NeighborTuple> NeighborSet;
3  /// Grensesnitt assosiasjonsset.
4  typedef std::vector<IfaceAssocTuple> IfaceAssocSet;
5  /// Annonserte (s,g) eller (*,g) assosiasjoner.
6  typedef std::vector<AssociationTuple> AssociationSet;
7  /// Lokale (s,g) eller (*,g) assosiasjoner.
8  typedef std::vector<Association> Associations;
9  /// Cache med unike pakke identiteter.
10 typedef std::vector<uint8_t> UniqnessTable;
```

Slik vil det fremstå per AIMF node (Se figur 3.5)

Disse typedefinisjonene er lister som holder visse strukturer. Disse strukturene inneholder visse primitiviteter og klasseobjekter (Se vedlegg A.3).

Her kan man si at det er prøvd å etterligne tilstanden som PIM derivatene [84] og OLSR [57] bruker til å opprettholde sin tilstand, men da uten å ta hensyn til en «redundans- og lastbalanseringsfunksjon». I tillegg til disse datasettene er det implementert funksjoner som forenkler manipulasjonen av disse datasettene.

Kontroll

Her går vi nærmere inn på hvordan vi kontrollerer disse datasettene med tanke på å nå kravene vi har satt. (eks. Hvordan er dataene i datasettene brukt for gjøre valg i pakke-videresendingsfunksjonen).

Kontrollaspektet i NS3 er veldig «låst», så man *må* ta hensyn til hvordan NS3 er implementert. NS3 har for eksempel ikke en form for «rute-buffer» (routecache)[1] per grensesnitt slik at hver IP-pakke *må* bli rutet eksplisitt. Altså det er ingen mellomlagret «regel» som styrer videresending. På grunn av dette er ikke selve rutetabellen så imperativ i arkitekturen, fordi den ikke brukes direkte. Se følgende kode:

```
1  bool RoutingProtocol::RouteInput(Ptr<const Packet> p,
2                  const Ipv4Header &header, Ptr<const NetDevice> idev,
3                  UnicastForwardCallback ucb, MulticastForwardCallback mcb,
4                  LocalDeliverCallback lcb, ErrorCallback ecb) {
5              NS_LOG_FUNCTION(this << p << header << header.GetSource() << ↩
                   header.GetDestination() << idev << &ucb << &mcb << &lcb <<↩
                    &ecb);
6              NS_ASSERT(m_ipv4 != 0);
7              // Check if input device supports IP
8              NS_ASSERT(m_ipv4->GetInterfaceForDevice(idev) >= 0);
9              if (header.GetDestination().IsMulticast()) {
10                 NS_LOG_LOGIC("Multicast destinasjon");
11                 Ptr<Ipv4MulticastRoute> mrtentry = LookupStatic(header.↩
                       GetSource(),
12                         header.GetDestination(), m_ipv4->↩
                               GetInterfaceForDevice(idev), header.GetTtl());
13
14                 if (mrtentry) {
15                     NS_LOG_LOGIC("Multicast rute ok");
16                     if (!forward) {
17                         return false;
18                     }
19                     mcb(mrtentry, p, header);
20
21                     m_txMcastPacketTrace(p->Copy(), m_ipv4, idev->↩
                           GetIfIndex());
22
23                     NS_LOG_DEBUG("Packet routed with destination: " << ↩
                           header.GetDestination() << " and source: " << ↩
                           header.GetSource() << " Will = " << (int) ↩
                           m_willingness << " . It has a TTl of " << int (↩
                           header.GetTtl()) << "↩
                           ----------------------------------");
24
25                     return true;
26                 } else {
27                     NS_LOG_LOGIC("Multicast rute er ikke funnet");
28
29                     return false;
30                 }
31             }
```

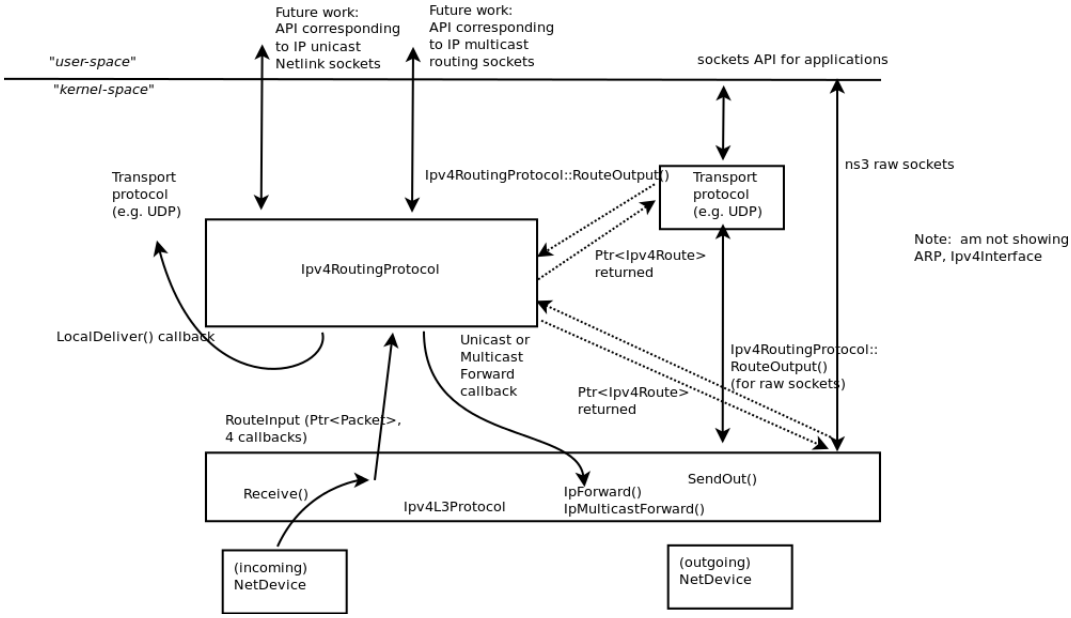
[1] I dag har de færreste linuxkjærner implemetert routecache. [51]

```
32              return false;
33          }
```

Det er flere eksempler på hvor NS3 «låser» arkitekturvalget, så vi nevner bare deklarasjonene:

```
1  virtual ~RoutingProtocol();// arvet destructor
2
3  virtual void DoInitialize(void); // Oppstart av ruteprotokoll.
4
5  virtual Ptr<Ipv4Route> RouteOutput(Ptr<Packet> p,
6                     const Ipv4Header &header,
7                     Ptr<NetDevice> oif,
8                     Socket::SocketErrno & sockerr); // Ruting av lokale IP↩
                          -pakker
9
10 virtual bool RouteInput(Ptr<const Packet> p,
11                    const Ipv4Header &header,
12                    Ptr<const NetDevice> idev,
13                    UnicastForwardCallback ucb,
14                    MulticastForwardCallback mcb,
15                    LocalDeliverCallback lcb,
16                    ErrorCallback ecb); //Ruting av fjerntkommende IP-↩
                          pakker.
17
18 virtual void NotifyInterfaceUp(uint32_t interface);
19
20 virtual void NotifyInterfaceDown(uint32_t interface);
21
22 virtual void NotifyAddAddress(uint32_t interface, Ipv4InterfaceAddress ↩
       address);
23
24 virtual void NotifyRemoveAddress(uint32_t interface, Ipv4InterfaceAddress↩
        address);
25
26 virtual void SetIpv4(Ptr<Ipv4> ipv4); // Faar den tildelte IPv4-↩
       informasjonen.
27 //Dette faar hver node ved hjelp av ns3::InternetStackHelper.
28
29 virtual void PrintRoutingTable(Ptr<OutputStreamWrapper> stream) const;
```

Videre er det en fordel å forstå hvordan disse funksjonene igjen blir kalt fra et ns3:Node perspektiv. Denne oversikten (Se figur 3.6) gir en god pekepinn på hvordan det er tenkt i NS3 med tanke på IPv4 ruting per nodeobjekt. Vår multikastruteprotokoll lever da «inne» i IPv4RoutingProtocol-entiteten og må forholde seg til de funksjonene gitt ovenfor. Og videre ville AIMF ha blitt plassert slik - se i figur 3.7 - med tanke på arv innunder NS3 IPv4 ruting miljøet.

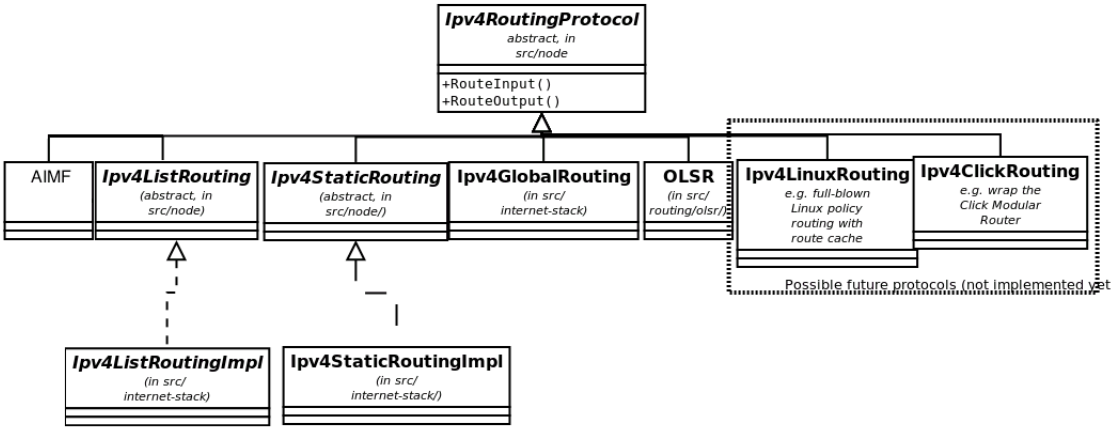


Figur 3.6: NS3s løsning på mottak og sending av pakker innunder IP. [60]

Ettersom NS3 er en hendelsesimulator og ut fra krav så blir funksjonsdiagrammet slik (Se figur 3.8). For videresendings-modulen blir funksjonsdiagrammet slik (Se figur 3.9).

Ut fra disse funksjonsdiagrammene får vi følgenede «regioner»:

- Init og Stopp
- Videresending
  - Videresending intern
  - Videresending ekstern
- Eventer
- Sjekk av OLSRs rutetabell
- Sending av hallomeldinger
- Mottak av hallomeldinger
- Oppdatering av rutetabell



Figur 3.7: Entitetsrelasjoner innunder NS3 IPv4 ruting. [62]

Hvert av de nevnte punkt vil da bli gjennomgått videre:

Init og Stopp

Her inne er det to tilstander hvor Doinit står for «mottak» og videre modellering og initiering av globale - og interne variabler. DoDispose har i oppgave å slette unna tilstand og gjøre protokollen død (da per node).

Videresending

Her forgår sendingen av pakker som skal ut. Denne oppgaven er videre oppdelt med tanke på om pakken er lokal eller fjerntkommende. Ingen av disse videresendingsfunksjonene er overkjørt (overridden) av andre funksjoner i NS3. Altså de er begge aktive. Det er kun multikast som er støttet så ved unikastruting må man legge til en unikastruteprotokoll noe det er gjort her. I denne konteksten er det som sagt brukt OLSR og samkjøringen av AIMF og OLSR er løst ved å bruke NS3s ListRouting-ruteprotokoll. AIMF vil da «kaste» alt annet enn multikast ned til OLSR (se figur 3.10). I det kablede protokolldomenet er ruting generelt løst ved kun bruk av multikastet trafikk (slipper da default gateway ol. per node i det kablede nettet).

Videresending intern Intern ruting er bare aktuelt når det er 2 eller flere grensesnitt - noe det er her. Innunder denne spesifikasjonen måtte det utarbeides en egen rute for lokalsendt multikast slik at bare det ønskelige grensesnittet ble tatt i bruk per AIMF ruter (Her det kablede grensesnittet). Denne lokalsendte multikast trafikken i denne konteksten er AIMFs egen signalering. Det er valgt å la signaleringen gå i det

kablede nettet, fordi dette er mest ressursbesparende kontra å sende signaleringen i MANETet. AIMFs signaleringen blir multikastet med adressen 230.0.0.30.

Videresending ekstern Ekstern ruting ble løst på en enkel måte. En enkelt «volatile» boolsk ($forward$) verdi styrer om en pakke blir videresendt eller ikke.

Eventer

Dette er en tilstand som vi ikke kontrollerer, men som vi kan instruere til å gjøre fremtidige funksjonskall (les eventliste). Rammeverket eier og kontrollerer eventlisten, mens en «bruker» av den kan endre den ved hjelp av ett grensesnitt. En eventliste kan sees på som en linket liste L hvor hvert innslag inneholder paret $(t_{tid}, H_{hendelse})$. Når prosessen som løper igjennom L ankommer ett innslag venter den i t_{tid} før den eksekverer $H_{hendelse}$. Det samme gjelder for neste innslag i L.

Sjekk av OLSRs rutetabell

Her sjekkes det om en AIMF-node er tilgjengelig via det aktuelle MANETet, slik at AIMF kan oppdage partisjonering. Eventlisten styrer når denne sjekken forekommer.

Se følgende kode:

```
1  void RoutingProtocol::OlsrTimerExpire() {
2              double t = 0;
3              uint8_t j = 0;
4              std::vector<olsr::RoutingTableEntry> v = m_olsr_onNode->↩
                   GetRoutingTableEntries();
5              switch (m_willingness) {
6                  case AIMF_WILL_ALWAYS:
7                      forward = true;
8                      break;
9                  case 1:
10                     t++;
11                 case 2:
12                     t++;
13                 case AIMF_WILL_DEFAULT:
14                     t++;
15                 case 4:
16                     t++;
17                 case 5:
18                     t++;
19                 case AIMF_WILL_HIGH:
20                     m_olsrCheck.Schedule(m_olsrCheckInterval + Time(↩
                           Seconds(t)));
21                     for (std::vector<NeighborTuple>::iterator neig = ↩
                           m_state.GetNeighbors().begin(); neig != m_state.↩
                           GetNeighbors().end(); neig++) {
22                         for (std::vector<olsr::RoutingTableEntry>::↩
                               iterator route = v.begin(); route != v.end(); ↩
                               route++) {
23                             if (route->destAddr == neig->neighborMainAddr)↩
                                   {
24                                 if (neig->willingness >= j) {
25                                     j = neig->willingness;
26                                 }
27
28                             }
29                         }
30                     }
31                     if (j <= m_willingness) {
32                         forward = true; //ALERT PIM
33                     } else {
34                         forward = false; //ALERT PIM
35                     }
36                     break;
37                 case AIMF_WILL_NEVER:
38                     forward = false;
39                     break;
40             }
41         }
```

Her vises selve metoden som gir en AIMFnode muligheten til å se om en annen AIMFnode er tilgjengelig via MANETet. På denne måten vil AIMF utlede om det forekommer partisjonering i MANET ved hjelp av å se i OLSRs rutetabell. Hvis en AIMF-GW ikke ser noen ruteoppslag til en, eller flere, AIMF-GW/er så er det partisjonering i MANETet. Vi har et AIMFsystem S. Videre har vi AIMFnode a med en vilje lik $vilje(a) = 6$ og AIMFnodene $vilje(b) = 4$ og $vilje(c) = 3$. Vi antar at a er den første av b og c til å gå igjennom sin funksjon ($a.OlsrTimeExpire()$). Ut fra funksjonen over vil ikke a få noen ekstra tilleggstid utover den tiden som er satt standard. Node b vil få ett kjøringsintervall på $5 + 2sek$ og node c vil få ett på $5 + 3$ sekunder. Følgen av dette er at node a vil kjøre denne metoden med ett intervall på 5 (uten tillegg) og følgelig gitt et intervall som er $2sek$ og $3sek$ mindre enn b og c når det gjelder kjøring av $OlsrTimeExpire()$. Som videre vil resultere is at en AIMFnode med høy vilje vil sannsynligvis reagere fortere på opphør av multikastvideresending enn en med lavere vilje.

Videre vil alle noder som ikke har vilje $vilje(x) = 0$ eller $vilje(x) = 7$ sjekke OLSRs rutetabell opp mot sin egen AIMFnaboliste. Ved treff i begge lister - med tanke på destinasjonsadresse og AIMFnaboadresse - vil hver node sjekke viljen til vær enkelt innslag(AIMFnode). Er det kumulativt sett en en høyere vilje i systemet vil *ikke* AIMFnoden som sjekker sette videresendingstilstand. I motsatt fall vil den følgelig sette videresendingstilstand.

Det er også kommentert at man kunne tenkt seg å varsle PIM her. Ved $forward = true$; send PIM-join hvis sistnevnte førte til endring av $forward$, hvis ikke; forhold seg passiv. Ved $forward = false$; send prune hvis sistnevnte førte til endring av $forward$, hvis ikke; forhold seg passiv.

En kommentar til de som ser at ved vilje `AIMF_WILL_ALWAYS` og `AIMF_WILL_NEVER` vil ikke funksjonen videre bli kalt: Ved endring av vilje på ett senere tidspunkt gitt `AIMF_WILL_ALWAYS` eller `AIMF_WILL_NEVER` vil det ved endringen bli gjort ett kall på funksjonen over.

Det kablede nettet og linkbrudd Når det gjelder linkbrudd i det kablede nettverket så har vi løst det ved bruk av egenskaper hos RP-baserte multikastruteprotkoller. Vi har allerede konkludert med at BIDIR-PIM vil være et godt valg vedrørende AIMF-signalering, men en annen viktig faktor for å bruke RP-baserte multikastruteprotkoller til multikastet signalering i AIMF er følgende:

Hvis vi antar at AIMF node a går ned grunnet linkbrudd eller lignende slik at de andre nodene i AIMF-system S ikke ser signalering fra a. Videre antar vi at a - «før» linkbruddet - eide videresendningsansvaret i S. De andre AIMFnodene i S vil som følge av dette utlede at a ikke kan motta multikastet trafikk overhode ettersom de er

medlemmer i det samme multikastreet. Dette fordrer imidlertid at det er kun en RP konfigurert innenfor ett gitt intradomene og at følgelig all multikastet trafikk bruker denne. Videre er det altså trygt for en av de andre AIMFnodene i S å videresende så lenge de ikke ser signaleringstrafikk fra node a. Mer detaljert så må følgende holde:

$$\text{MST}(KogM(AIMF_{signal}))^2 < MST(K(x) \cup KogM(AIMF_{signal})).$$

GW a vil fortsatt ha videresendingstilstand, men den har ikke noe å videresende, fordi den ikke når RP, eller mer presist - med tanke på BIDIR-PIM - noen rutere som har tilstand $(*, x)$ og som samtidig binder resterende AIMFnoder i S (Dette vil da følgelig gjelde trafikk som går begge veier). Dette er fin måte å løse redundans i det kablede domenet. Denne logikken vil også gjelde hvis alle noder blir stående «alene».

[2] I ett AIMF-system så er $K(g) == M(g)$ alltid sant.

Pakkefelt	**Bruk**
Pakkelengde	Gir en mulighet til å «trace» pakker med tanke på feilsøking. Feltet blir også brukt til å sjekke om en pakke er korrupt eller ikke.
Pakkesekvensnummer	Som punktet over (Pakkelengde), men spesielt med tanke på sporing.
Beskjed type	Sier hvilken type submelding som henger ved. I AIMF er det bare en, men det er da enklere å legge til flere.
Vtime	Ut fra denne tiden setter mottaker «holdetiden» for en nabo.
Beskjedstørrelse	Størrelse på HELLO-HMA beskjed, men vil også gi størrelsen på andre type meldinger hvis dette blir aktuelt.
Avsenderadresse	Denne adressen brukes som id nå det lages ett naboinnslag. (Denne adressen trenger ikke å være lik faktisk avsender adresse)
Time To Live	Det er tenkt at man skal kunne videresende AIMF-hallo meldinger i MANET, men dette er ikke implementert per nå.
Beskjedsekvensnummer	Bruk i feilsøkingsformål, spesielt sporing med tanke på HMA.
Htime	Ut fra denne tiden setter mottaker «holdetiden» for den annonserte multikastgruppen.
Vilje	Viljen til avsender.
Gruppeadresse	Gruppeaddresse for gitt multikastgruppe
Kildeadresse	Kildeadresse for gitt multikastgruppe
Vilje per Gruppe	«Sub»-vilje for gitt multikastgruppe. (Det er tenkt at dette kan være mottat TTL for denne multikastgruppen per avsender.)

Tabell 3.1: Pakkefeltbeskrivelse

Sending av hallomeldinger

Her sendes det hallomeldinger. Dette forkommer enten når viljevariabelen blir endret lokalt eller når eventlisten kaller på denne funksjonen.

En AIMF-pakke inneholder følgende felter (Se figur 3.11 og figur 3.1). En pakke genereres ved bruk av en buffer. C++ har en buffer::Iterator som lar deg lage ett «sett» med bits. Her kan man legge til disse pakkefeltene i serie som i påfølgende eksempel.

Vi har Buffer X. Vi skal eksempelvis legge til destinasjons adresse d (32bit), avsender adresse a (32bit) og data z (135bit). Da kan man gjøre slik (pseudo):

```
Buffer::Iterator X = Buffer(1500bit).getIterator();
            X.Write32bit(d);
            X.Write32bit(a);
            while ((X.size - (d.size + a.size)) < z.size) {
       X.write1bit(z);
    }
}
```

Videre er det fulgt samme prinsipp for pakkehodet:

```
 void
        PacketHeader::Serialize(Buffer::Iterator start) const {
            Buffer::Iterator i = start;
            i.WriteHtonU16(m_packetLength);
            i.WriteHtonU16(m_packetSequenceNumber);
        }
```

Og videre for en hallomelding i AIMF og en, eller flere, Host Multicast Association (HMA) meldinger (samme bufferobjekt som for pakkehodet):

```
void MessageHeader::Serialize(Buffer::Iterator start) const {
            Buffer::Iterator i = start;
            i.WriteU8(m_messageType);
            i.WriteU8(m_vTime);
            i.WriteHtonU16(this->GetSerializedSize());
            i.WriteHtonU32(m_originatorAddress.Get());
            i.WriteU8(m_timeToLive);
            i.WriteHtonU16(m_messageSequenceNumber);
            m_message.hello.Serialize(i);
        }
```

For HMA vil hTime og willingness bare bli sendt en gang per pakke, men man ha en til mange $(group, source, will)$ tupler per pakke.

```
void
        MessageHeader::Hello::Serialize(Buffer::Iterator start) const {
            Buffer::Iterator i = start;
            i.WriteU8(this->hTime);
            i.WriteU8(this->willingness);
            for (size_t n = 0; n < this->associations.size(); ++n) {
                i.WriteHtonU32(this->associations[n].group.Get());
                i.WriteHtonU32(this->associations[n].source.Get());
                i.WriteU8(this->associations[n].willGroupSSM);
            }
        }
```

I AIMF er pakkestrukturen satte opp i gitte bolker. Ett eksempel er at man sier at $d + a = h$ og hvor h blir $64bit$. Videre kan man da få en en annen funksjon til å deserialisere (eller objektivisere) en viss «bolk» (h) av pakken. Dette gir en bedre oversikt, man kan redigere felt inn innunder h på $64bit$ (modularitet) og er dermed i god objektorientert ånd.

Selve prosessen lager en enkelt hallomelding og en, eller flere HMAmedlinger først ettersom pakkelengdefeltet i pakkehodet er avhengig av størrelsen. Det innebærer at hver pakkehodet blir lagt til sist til i bufferen likt som i en stakk. Oppsummert vil det som er lagt til sist bli sendt først (LIFO). Disse serialiseringsmetodene «overkjører» serialiseringsfunksjonen som ns3::Packet::Addheader bruker[3]. Så for å summere opp så arver pakkehode, hallomedling og HMA ns3::Header. Så når man initierer eksempelvis en hallomelding msg og legger til data som nevnt over med «settere» (eks. $msg.SetVtime(x)$) og deretter kaller ns3::Packet::AddHeader(msg) vil serialiseringsfunksjonen til hallomeldingobjektet bli kalt grunnet objekttypen.

Når man så skal sende pakken bruker man en Socket s, buffer X (her gitt som ns3::Packet packet) og nødvendig IP informasjon som parameter. Slik er det gjort i AIMF (Til slutt i funksjonen ser man at utgående pakker blir tracet):

```
1  for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator i =  ↩
       m_socketAddresses.begin(); i != m_socketAddresses.end(); i++) {
2                 Ipv4Address mcast = Ipv4Address(AIMF_MCAST_ADR);
3                 NS_LOG_DEBUG("Using socket with  " << i->second.GetLocal()↩
                        << " as src address.");
4
5                 i->first->SendTo(packet, 0, InetSocketAddress(mcast, ↩
                      AIMF_PORT_NUMBER));
6                 Ptr<Packet> packetCopy;
7                 packetCopy = packet->Copy();
8                 m_txHelloPacketTrace(packetCopy, m_ipv4, i->first->↩
                      GetBoundNetDevice()->GetIfIndex());
9             }
```

Adressen AIMFs hallomeldinger blir sent på er 230.0.0.30 (`AIMF_MCAST_ADR` = 230.0.0.30).

Mottak av hallomeldinger

Denne tilstanden leser en hallomelding og lagrer data assosiert med denne meldingen. Slik er dette løst:

Her mottas altså pakker som blir sendt i AIMF. Det er løst ved at det er initiert en Socket s som er låst til IP-adresse 0.0.0.0 og port 1337. Videre er det «hengt» en

[3]Denne metoden er virtuell(C++) [18] og klassen ns3::Header er en abstrakt klasse så det blir feil å si at den blir brukt, men den er «nærmest» med tanke på arv og *må* dermed overkjøres.

callbackfunksjon på *s* slik at ved mottak blir satt funksjon (RecvAimf) kalt (*this* blir her brukt til å bestemme hvilken instans av RecvAimf funksjonen som skal bli kalt[4] (se vedlegg A.1.) Slik er det gjort i AIMF:

```
1  Ptr<Socket> socket = Socket::CreateSocket(GetObject<Node> (),
2                                  UdpSocketFactory::GetTypeId());
3                  NS_LOG_DEBUG("Trying to bind X on " << addr);
4                  InetSocketAddress inetAddr(Ipv4Address::GetAny(), ↩
                       AIMF_PORT_NUMBER);
5                  socket->SetRecvCallback(MakeCallback(&RoutingProtocol::↩
                       RecvAimf, this));
6                  if (socket->Bind(inetAddr)) {
7                      NS_FATAL_ERROR("Failed to bind() AIMF socket " << addr↩
                           );
8                  }
9                  socket->BindToNetDevice(m_ipv4->GetNetDevice(i));
```

Gitt kodesegmentet over, antar vi at vi har fått en pakke og RecvAimf (se vedlegg A.4 og videre i aimf-routing-protocol.cpp) blir kalt. RecvAIMF sjekker så slike ting som om pakken har gyldig destinasjonsadresse(230.0.0.30) - *s* er «altspisende» med tanke på destinasjonsadresse - og sjekker om pakken faktisk har størrelsen som er oppgitt i *pakkelengde*. Hvis da en av disse «sjekkene» feiler blir pakken kastet. Videre vil den implisitt begynne å deserialisere den ankommende pakken. Slik er det gjort i AIMF:

Pakkehode (en enkelt pakkehode per pakke):

```
1  uint32_t
2          PacketHeader::Deserialize(Buffer::Iterator start) {
3              Buffer::Iterator i = start;
4              m_packetLength = i.ReadNtohU16();
5              m_packetSequenceNumber = i.ReadNtohU16();
6              return GetSerializedSize();
7          }
```

[4]Husk det er like mange instanser av *s* og RecvAimf som det er AIMF-GWer så dette må spesifiseres. Alle AIMF-GWer lever i samme eksekveringsområde (*executionspace*) så derfor må man skille på instanser.

Hallomelding (kun en enkelt hallomelding per ankommende pakke):

```
uint32_t
        MessageHeader::Deserialize(Buffer::Iterator start) {
            uint32_t size;
            Buffer::Iterator i = start;
            m_messageType = (MessageType) i.ReadU8();
            NS_ASSERT(m_messageType == HELLO_MESSAGE);
            m_vTime = i.ReadU8();
            m_messageSize = i.ReadNtohU16();
            m_originatorAddress = Ipv4Address(i.ReadNtohU32());
            m_timeToLive = i.ReadU8();
            m_messageSequenceNumber = i.ReadNtohU16();
            size = AIMF_MSG_HEADER_SIZE;
            size += m_message.hello.Deserialize(i, m_messageSize - ↩
                AIMF_MSG_HEADER_SIZE);
       return size;
        }
```

Og sist og ikke minst HMAmelding. Det kan være flere HMAer i en pakke.

```
uint32_t
        MessageHeader::Hello::Deserialize(Buffer::Iterator start, uint32_t↩
             messageSize) {
            Buffer::Iterator i = start;
            this->hTime = i.ReadU8();
            this->willingness = i.ReadU8();
            NS_ASSERT((messageSize-2)
            int numAddresses = (messageSize-2) / IPV4_ADDRESS_SIZE / 2;
            this->associations.clear();
            for (int n = 0; n < numAddresses; ++n) {
                Ipv4Address group(i.ReadNtohU32());
                Ipv4Address source(i.ReadNtohU32());
                uint8_t will(i.ReadU8());
                this->associations.push_back((Association) {
                    group, source, will
                });
            }
return messageSize;
        }
```

Denne prosessen er svært lik prosessen ved mottak av pakker. Disse deserialiseringsfunksjonene bli kalt av ns3::Packet.RemoveHeader - istedenfor Addheader - og man har dermed objektivisert pakkefeltene for videre bruk.

Følgelig vil disse dataene videre brukes til å oppdatere lokal tilstand. Ett eksempel er rutetabellen.

Oppdatering av rutetabell

En oppdatering av rutetabellen skjer når en hallomelding blir mottatt eller når det blir konfigurert ett (k, g), eller $(*, g)$ par i sanntid eller før kjøring. Rutetabellen

baserer seg på lokale og annonserte multikastgruppepar (Se vedlegg A.2).

Kildekode

Se vedlegg A.4 for kildekode for AIMF.

3.2.3 Oppsummering kontroll

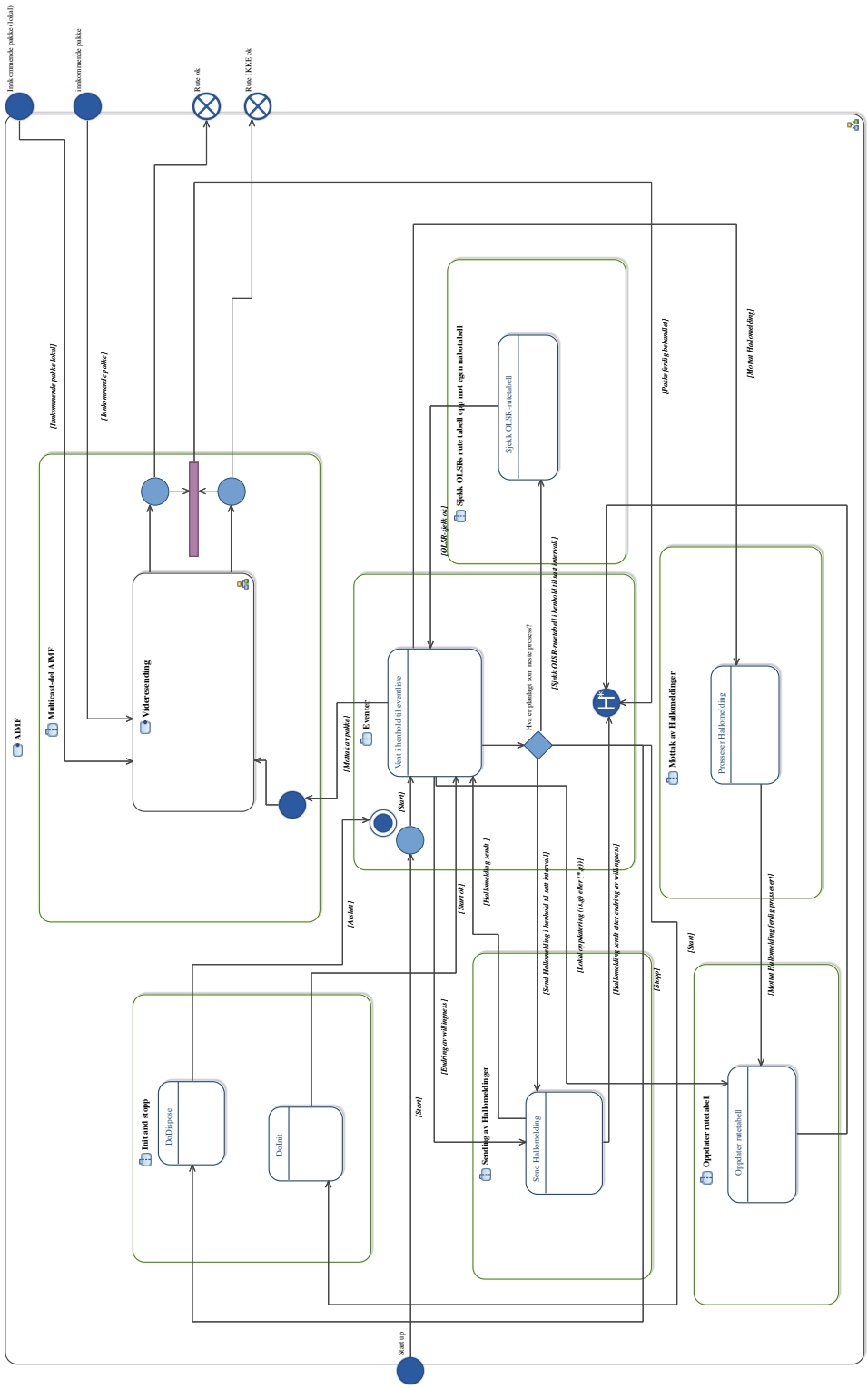
Det skal sies at noen av disse regionene er mere komplekse enn hva som er gjengitt her. Ett eksempel er ekstern videresending hvor det bare nevnes at en boolsk verdi styrer videresending. Grunnen til at det er gjengitt på ett overordnet nivå er på grunn av det ville vært lite hensiktsmessig å forklare alt i detalj ettersom veldig mye av funksjonaliteten allerede er i ns3 (Man ville ha forklart ns3 og ikke AIMF). Så vi anbefaler at man ser i vedleggene for å få mer inngående kjennskap og/eller legger til protokollen i eget ns3 miljø.

3.2.4 Grensesnitt

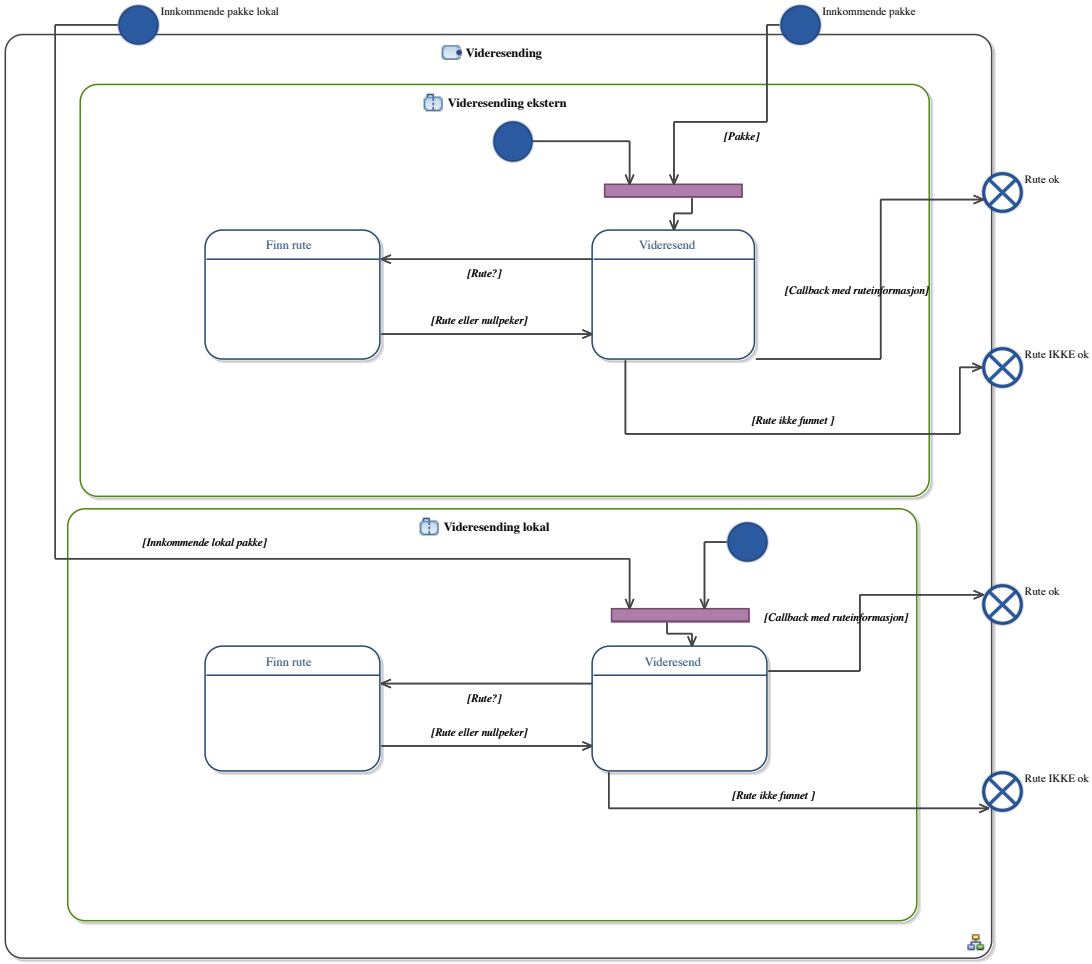
Når det gjelder grensesnitt er NS3 designet slik at man lager tilpassede programmer som *styrer* en simulering. I dette programmet velger man blant annet hvordan node-topologien skal være, hvilke IP-adresser skal tilordnes hvilke noder og lignende (Man setter rett og slett noen av miljøvariablene for å dra en analog til et operativsystem). (Se vedlegg A.1) Videre er dette programmet ikke en del av selve systemet man simulerer, men innehar en viss «master-rolle».

3.3 SMF

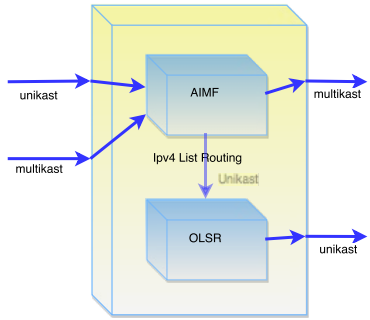
Simplified Multicast Forwarding (SMF) ble implementert ved bruk av en 32*bit* hash-metode tilnærmet likt som designet i [47] og generelt sett i [49]. Videre har implementasjonen arvet ipv4RoutingProtocol i ns3, likt som i AIMF. Det ble ikke valgt å legge denne funksjonaliteten inn i AIMF (SMF er «standalone» i denne oppgaven) ettersom det kan være interessant, og praktisk, å kjøre SMF alene. Hvordan SMF, AIMF og OLSR koopererer innenfor «Ipv4ListRouting» objektet er vist i figur 3.12. Selve funksjonaliteten består av en hash-metode (samme prinsipp som i H-DPD [49]) som «hasher» hver ankommende pakke, uansett grensesnitt, og hvis pakken er «ny» lagrer man denne hashen i en liste(vector). Hvis den ikke er «ny» blir pakken kastet. Videre blir «Hash-listen» skrelt i den «eldre» enden med en tredjedel av listestørrelsen hvert 10. sekund. Manet-grensesnittet må settet eksplisitt i smfHelper (Kontrollskript). Se vedlegg A.4 for kildekode.



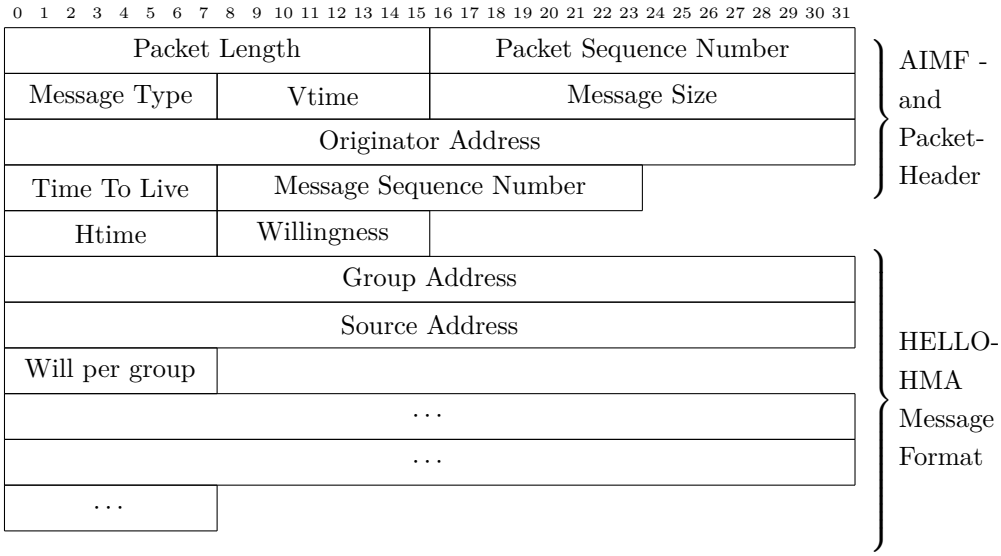
Figur 3.8: Funksjonsdiagram AIMF.



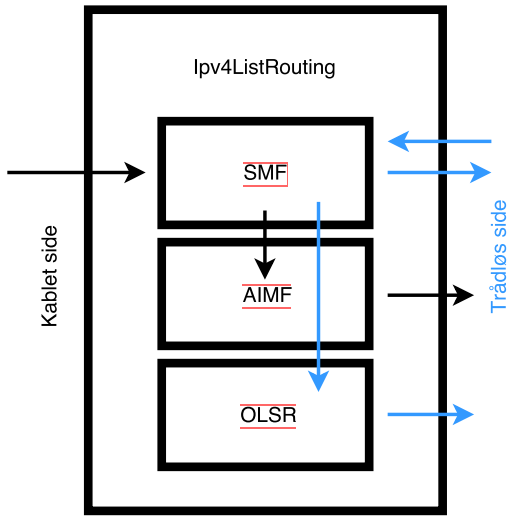
Figur 3.9: funksjonsdiagram for videresending i AIMF.



Figur 3.10: AIMF og OLSR sammen.



Figur 3.11: Pakkefeltdiagram



Figur 3.12: SMF, AIMF og OLSR sammen. Svart pil er multikasttrafikk. Blå pil er multikasttrafikk og kringkastningstrafikk(OLSR).

Kapittel 4
Tester og Resultater

I dette kapittelet vil tester av AIMF bli gjennomgått og testene er gjort innenfor scenariet gitt i seksjon 4.2.

4.1 Testing av AIMF

Lab-materiell og avhengigheter er beskrevet i følgene deployeringsdiagram (Se figur 4.1). Under selve lab-arbeidet er det brukt NetBeans IDE [54] som er spesielt designet for Java. NetBeans har også støtte for C++ og flere andre programmeringsspråk, men det viktige her er *IntelliSense*, eller såkalt *picklist*-støtte [90]. Videre er det brukt GNU Debugger (GDB) [32] og Valgrind [25] til feilsøking og profilering. Dette har blitt gjort på en maskin med Debian 8 (jessie) [21] operativsystem.

4.1.1 Innledning

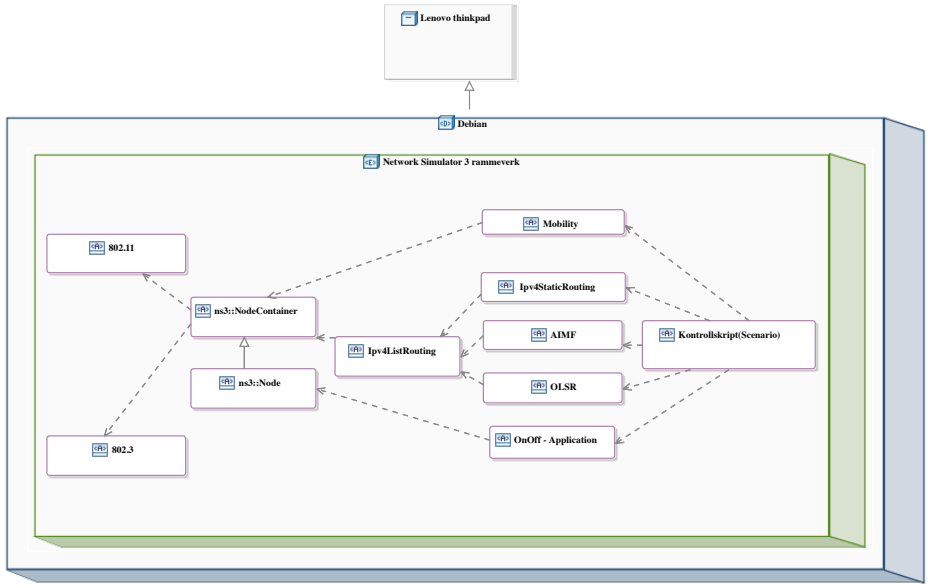
Så hvordan tester vi AIMF? Hva er det vi vil se? Her det en liste over stikkord for testene.

Opptrer AIMF sparsomt? Vil det alltid være ett minimumsett av GWer som videresender multikastet data?

Gir AIMF mer robusthet grunnet bruk av flere GWer? Vil AIMF «tåle» en flyktig topologi?

Utvider AIMF ett kablet multikastdomene til å også innlemme ett MANET? Gir den multikasttjenester innenfor nevnte kriterier?

Dette er da rett og slett en test for å se om kravene i kapittel 3 er overholdt.



Figur 4.1: Deployeringsdiagram for scenario gitt i seksjon 4.2.

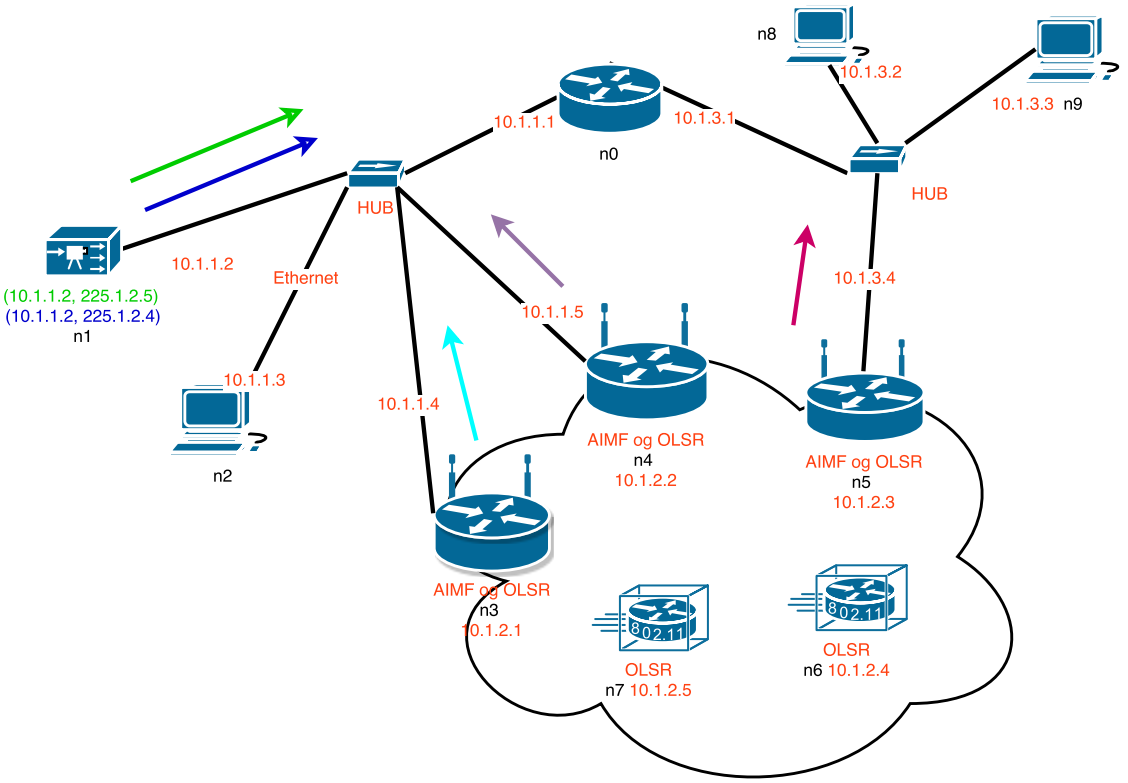
4.2 Sceneria gitt av kontrollskript

Det finnes flere scenarier hvor denne protokollen kan være tenkt brukt, men her vil vi fokusere på ett enkelt scenario til å begynne med.

Antagelser Se figur 4.2.

MANET-nodene (MNne) kjører OLSR. Denne protokollen er valgt fordi den er proaktiv og innehar derfor en mindre flyktig og mer «tilstedeværende» unikastrutetabell enn eksempelvis reaktive Ad hoc On-Demand Distance Vector (AODV) Routing [69], ettersom AIMF gjør bruk av denne og ettersom denne protokollen ble valgt i designene gitt i forprosjektet [24]. Sist, men ikke minst blir det ikke sendt noe unikastet data slik at AODV ville ha vært uten rutetabell. I tillegg er det viktig å nevne at MNene og AIMF-GWene ikke kjører SMF under test 1 til og med test 3 (De videresender altså ikke multikasttrafikk i MANETet). SMF blir først brukt i test 4. Gatewayene (GWene) kjører AIMF, men i tillegg OLSR. Ettersom AIMF er avhengig av OLSR protokollens unikastrutetabell - noe AIMF ikke har. I test 4 blir også SMF lagt til på GWene.

Nodene i det faste nettet $n0$,$n1$,$n2$,$n8$ og $n9$ - som vi kaller kablede noder (KNer) - er tilknyttet et vanlig kobbernett som har grensesnitt hvor Ethernet og IPv4 er implementert, mens MNne $n6$ og $n7$ er tilkoblet ett infrastrukturløst Ad-hoc nettverk



Figur 4.2: Oversikt for scenario. Hvor turkis, lilla og vinrød pil er AIMF signalering. Node $n0$ ruter både AIMF signalering og multikasttrafikk fra $n1$.

(802.11 W-LAN). GWne $n3$, $n4$ og $n5$ er tilsluttet begge de to sistnevnte netttypene. Nodene $n0$, $n1$, $n2$ og $n3$ og $n4$ ligger i adresserom 10.1.1.0/24, mens $n5, n8$ og $n9$ ligger i adresserom 10.1.3.0/24. MNne, samt GWne, ligger i adresserom 10.1.2.0/24.

Når det gjelder nettverkslaget og videre opp i OSI-modellen har noen KNer ekstra oppgaver. Node $n1$ sender to multikaststrømmer. Dataraten er $15kb/s$ og pakkestørrelsen (nyttelasten) er $1400bytes$ for 225.1.2.4-gruppen, mens dataraten er $10kb/s$ og pakkestørrelsen (nyttelasten) er $100bytes$ for 225.1.2.5 -gruppen. Videre er innholdet i nyttelasten bare nuller:

```
1
2       Ipv4Address multicastGroup("225.1.2.4");
3       Ipv4Address multicastGroup2("225.1.2.5");
4
5   OnOffHelper onoff("ns3::UdpSocketFactory",
6               Address(InetSocketAddress(multicastGroup, 9)));
7       onoff.SetConstantRate(DataRate("15kb/s"));
8       onoff.SetAttribute("PacketSize", UintegerValue(1400));
9
10      ApplicationContainer srcC = onoff.Install(c0.Get(1));
11
12      OnOffHelper onoff2("ns3::UdpSocketFactory",
13              Address(InetSocketAddress(multicastGroup2, 1336)));
14      onoff2.SetConstantRate(DataRate("10kb/s"));
15      onoff2.SetAttribute("PacketSize", UintegerValue(100));
16
17      ApplicationContainer srcC2 = onoff2.Install(c0.Get(1));
```

Node $n0$ har satt statiske[1] multikastruter for 225.1.2.4, 225.1.2.5 og 230.0.0.30 som gjengitt her:

```
1       multicast.AddMulticastRoute(c.Get(0), (Ipv4Address("").GetAny()), ↩
            Ipv4Address("225.1.2.4"), senderIf, NetDeviceContainer(nd2.Get(0))↩
            );
2       multicast.AddMulticastRoute(c.Get(0), (Ipv4Address("").GetAny()), ↩
            Ipv4Address("225.1.2.5"), senderIf, NetDeviceContainer(nd2.Get(0))↩
            );
3       multicast.AddMulticastRoute(c.Get(0), (Ipv4Address("").GetAny()), ↩
            Ipv4Address("230.0.0.30"), senderIf, NetDeviceContainer(nd2.Get(0)↩
            ));
4       multicast.AddMulticastRoute(c.Get(0), (Ipv4Address("").GetAny()), ↩
            Ipv4Address("230.0.0.30"), nd2.Get(0), NetDeviceContainer(senderIf↩
            ));
```

Node $n0$ videresender altså begge disse to multikastgruppene fra 10.1.1.0/24 til 10.1.3.0/24 og den videresender AIMF-singnalering begge veier (230.0.0.30). Hubene illustrerer bare et delt medium [3] innfor hvert nett og er derfor ikke entiteter i seg selv.

[1] Dette er grunnen til an Ipv4StaticRouting er lagt til i Ipv4ListRouting objektet.

4.2.1 Mobilitet og utgangsplasseringer

Når det gjelder mobilitet er følgelig de kablede nodene fullstendige statiske, men ikke $n6$ og $n7$. Deres bevegelser er styrt med bruk av en ns3:MobilityHelper. I denne konteksten er det angitt plassering for alle entiteter - da også kablede - på grunn av NetAnimator [55] bruk. Slik er det løst i AIMF: Vi begynner først med den kablede delen - altså GWne og KNne:

- GWne: $n3(140, 8, 0)$, $n4(160, 8, 0)$ og $n5(180, 8, 0)$.
- KNne: $n0(170, 2, 0)$,$n1(145, 3, 0)$,$n2(155, 3, 0)$, $n8(170, 2, 0)$ og $n9(190, 2, 0)$ (Dette vil ikke endre seg i noen av de påfølgende testene).

```
1  MobilityHelper mobility2;
2          Ptr<ListPositionAllocator> positionAlloc = CreateObject<↩
               ListPositionAllocator> ();
3          positionAlloc->Add(Vector(140.0, 8.0, 0.0));//gw1-n3
4          positionAlloc->Add(Vector(160.0, 8.0, 0.0));//gw2-n4
5          positionAlloc->Add(Vector(180.0, 8.0, 0.0));//gw3-n5
6          positionAlloc->Add(Vector(170.0, 2.0, 0.0));//n0
7          positionAlloc->Add(Vector(145.0, 3.0, 0.0));//n1
8          positionAlloc->Add(Vector(155.0, 3.0, 0.0));//n2
9          positionAlloc->Add(Vector(170.0, 2.0, 0.0));//n8
10         positionAlloc->Add(Vector(190.0, 2.0, 0.0));//n9
11         mobility2.SetPositionAllocator(positionAlloc);
12         mobility2.SetMobilityModel("ns3::ConstantPositionMobilityModel");
13         NodeContainer w = NodeContainer(c.Get(3), c.Get(4), c.Get(5));
14         NodeContainer k= NodeContainer(c.Get(0), c.Get(1), c.Get(2), c.Get↩
               (8), c.Get(9));
15         w.Add(k);
16         mobility2.Install(w);
```

Videre er det løst slik for MNne:

Her er hver MN gitt en «sirkel» som setter rammene hvor MNne kan starte. Altså: Deres utgangposisjon er innenfor en sirkel med radius 41 og med senter i $(95, 50)$. Videre kan da MNne «havne» - ut fra en uniformt fordelt fordeling - innenfor arealet $A = \frac{360°}{360°}\pi * 41^2 = \pi * 41^2$).

Når det kommer til forflytting - etter start - er det satt ett nytt forflytningsmønster hvert halve (0.5) sekund og selve forflytingen kan ha en hastighet - som er uniformt fordelt - fra 10 til $100m/s$. Ved forflytting er det også satt rammer for bevegelse hvor forflyttningsrommet er $[0, 190] \times [8, 150]$.

Selve utgangplasseringene og mobiliteten kan under testene endres så betrakt nevnte oppsett som et eksempel.

```
1  MobilityHelper mobility;
2          mobility.SetPositionAllocator("ns3::RandomDiscPositionAllocator",
3                  "X", StringValue("95.0"),
4                  "Y", StringValue("50.0"),
5                  "Rho", StringValue("ns3::UniformRandomVariable[Min=0|Max↩
                       =41]"));
6          mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
7                  "Mode", StringValue("Time"),
8                  "Time", StringValue("0.5s"),
9                  "Speed", StringValue("ns3::UniformRandomVariable[Min=10|↩
                       Max=100]"),
10                 "Bounds", StringValue("0|190|8|150"));
11         mobility.Install(NodeContainer(c.Get(6), c.Get(7)));
```

4.2.2 Trådløs rekkevidde

Den trådløse rekkevidden er løst enkelt ved å bruke en «propageringsmodell» som er gitt i NS3-rammeverket:

```
1
2  ns3::RangePropagationLossModel
```

Denne modellen er en blant mange modeller som bestemmer transmisjonstap og transmisjonsforsinkelse. [58] [59] Modellen vist over er enkel og baserer seg på at det er satt en bestemt rekkevidde signalet kan gå. Er node b innenfor en bestemt rekkevidde gitt a så vil b «høre» a og motsatt (symmetrisk egenskap). I denne konteksten er denne rekkevidden satt til 150 (relativt til tall brukt under seksjon 4.2.1).

4.3 Tidsintervaller i AIMF

Det er flere tidsintervaller i AIMF som er uavhengige av hverandre og dermed flyter i forhold til hverandre. Disse er gitt i følgende tabell (se tabell 4.1) :

Intervall	**Forklaring**	**Definisjon**
Hallomelding-sendeintervall	Ett tidsintervall mellom sending av hallomeldinger.	Har en «defaultverdi» på 2sek, men dette kan justeres ved hjelp av AimfHelper.
AIMF-nabo holdetid	Den tiden en AIMF-instans holder på en nabo uten at denne nabotilstanden blir oppdatert (soft state).	3× Hallomelding-sendeintervall
OLSR-sjekk intervall	Ett tidsintervall mellom sjekker av OLSRs rutetabell.	Har en «defaultverdi» på 5sek+(0-5sek)† hvor sistnevnte intervall avhengig av viljen til hver node, men dette kan justeres ved hjelp av AimfHelper.

Tabell 4.1: Tabell over intervaller i AIMF. († Vilje lik 6 gir ett tillegg på 0sekunder, 5 gir 1sek, 4 gir 2sek, 3 gir 3sek, 2 gir 4sek og 1 gir 5sek. En vilje lik 7 setter en AIMFnode til å alltid videresende, mens ved en vilje lik 0 vil en AIMF node aldri videresende.)

Disse intervallene er ikke de eneste «entitetene» som bestemmer eksekvering av funksjoner vedrørende eksempelvis sending av hallo meldinger. Dette er forklart under kapittel 3 og under eksempelvis sendning av hallomeldinger og OLSR sjekk.

4.4 Statistiske metoder brukt under testing

Under noen av testene i kapittel 4 er det utregnet er korrelasjonskoeffisient for å si noe om forholdet mellom en AIMF-GWs videresendingstilstand og MANET nodenes forflytninger. Dette er løst ved å først hente ut en tidsserie over mottak hos MANET noder og sendt data hos AIMF-GWer. Tidsseriene har følgelig da eksakt lik tidsperiode. Disse tidsseriene er videre sortert på multikast gruppe og MAC-adresse. Videre er det da regnet ut samvariasjonen - se funksjon 4.1 - mellom en AIMF-GWs

sending av en spesifikk multikastegruppe og en MANET nodes mottak av denne. Dette er gjort for alle mulige kombinasjoner av AIMF-GWer og MANET noder. Disse tidsseriene, eller datasettene, er sortert og hentet ut ved hjelp av WireShark [93].

Algorithm 4.1 Funksjon for Korrelasjon.

▷ $ds\_(y,m)$ er MANET node ys datasett for multikastgruppe $m \in M$ hvor $y \in Y$ og hvor Y er alle MANET noder uten GW-funksjonalitet.
▷ $ds\_(x,m)$ er AIMF-GW xs datasett for multikastgruppe $m \in M$ hvor $x \in X$ og hvor X er alle AIMF-GWer.
for all $m \in M$ **do**
  for all $x \in X$ **do**
    for all $y \in Y$ **do**
      $Kor(ds\_(x,m), ds\_(y,m)) = \frac{Kovar(ds\_(x,m), ds\_(y,m))}{\sqrt{Var(ds\_(x,m)) \times Var(ds\_(y,m))}}$
    end for
  end for
end for

4.5 Tester

Her viser vi tester og resultater for vilje, linkbrudd-håndtering i det kablede nettet samt partisjonering i MANET.

4.5.1 Test 1

Her tester vi om multikasttjenester blir gitt til begge MNene over periode på 500sekunder og vi viser at viljen til GWene blir kommunisert og brukt. Følgende instruksjoner blir gitt til AIMF systemet igjennom kontrollskriptet i løpet av simuleringen:

Ved $t = 1$ i simuleringen blir $n3$, $n4$ og $n5$ gitt viljevariablene 2, 3 og 4. Videre blir $n3$ konfigurert med 225.1.2.4 ved $t = 3sek$ og 225.1.2.4 ved $t = 4sek$ ut i simuleringen. Senere får $n3$ endret sin vilje til 5 ($t = 50sek$), $n4$ får endret sin vilje til 6 ($t = 100sek$) og til slutt endres viljen til $n4$ ned til 3 ved $t = 200$.

```
1  Simulator::Schedule(Seconds(1.0), &aimf::RoutingProtocol::↩
       ChangeWillingness, aimf_Gw, 2);
2  Simulator::Schedule(Seconds(1.0), &aimf::RoutingProtocol::↩
       ChangeWillingness, aimf_Gw2, 3);
3  Simulator::Schedule(Seconds(1.0), &aimf::RoutingProtocol::↩
       ChangeWillingness, aimf_Gw3, 4);
4  Simulator::Schedule(Seconds(3.0), &aimf::RoutingProtocol::↩
       AddHostMulticastAssociation, aimf_Gw, multicastGroup, ↩
       multicastSource);
5  Simulator::Schedule(Seconds(4.0), &aimf::RoutingProtocol::↩
       AddHostMulticastAssociation, aimf_Gw, multicastGroup2, ↩
       multicastSource2);
6  Simulator::Schedule(Seconds(50.0), &aimf::RoutingProtocol::↩
       ChangeWillingness, aimf_Gw, 5);
7  Simulator::Schedule(Seconds(100.0), &aimf::RoutingProtocol::↩
       ChangeWillingness, aimf_Gw2, 6);
8  Simulator::Schedule(Seconds(200.0), &aimf::RoutingProtocol::↩
       ChangeWillingness, aimf_Gw2, 3);
```

Det vi er interessert i å se her er hva $n3$, $n4$ og $n5$ - GWene - sender til en hver tid og videre se hva node $n6$ og $n7$ mottar. Til slutt er vi også interessert i om AIMF-systemet reagerer på endring av vilje hos hver GW. Node $n7$ og $n6$ er alltid innenfor dekningen til GWene (Se påfølgende kildekode og figur 4.3).

Node $n6$s og $n7$s utgangposisjon er tildelt en sirkel hver. For $n6$ innebærer dette at utgangsposisjonen er innenfor en sirkel med radius 41 og med senter i $(150, 50)$ og for $n7$ en sirkel med radius 41 og med senter i $(160, 50)$. Forflytningsrommet - etter endt utplassering - er $[130, 190] \times [8, 100]$ og $[130, 190] \times [8, 100]$ for $n6$ og $n7$.

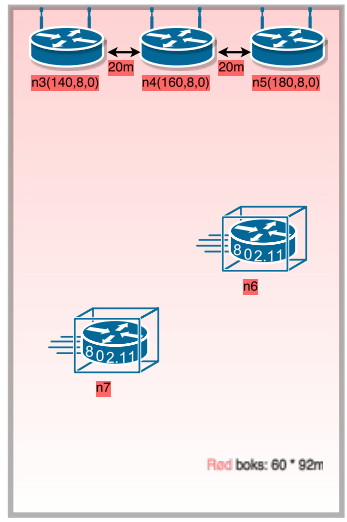
```
1   MobilityHelper mobility;
2       mobility.SetPositionAllocator("ns3::RandomDiscPositionAllocator",
3                "X", StringValue("160.0"),
4                "Y", StringValue("50.0"),
5                "Rho", StringValue("ns3::UniformRandomVariable[Min=0|Max↩
                     =41]"));
6       mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
7                "Mode", StringValue("Time"),
8                "Time", StringValue("0.5s"),
9                "Speed", StringValue("ns3::UniformRandomVariable[Min=10|↩
                     Max=100]"),
10               "Bounds", StringValue("130|190|8|100"));
11      mobility.Install(c.Get(7));
12      MobilityHelper mobility3;
13      mobility3.SetPositionAllocator("ns3::RandomDiscPositionAllocator",
14               "X", StringValue("150.0"),
15               "Y", StringValue("50.0"),
16               "Rho", StringValue("ns3::UniformRandomVariable[Min=0|Max↩
                     =41]"));
17      mobility3.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
18               "Mode", StringValue("Time"),
19               "Time", StringValue("0.5s"),
20               "Speed", StringValue("ns3::UniformRandomVariable[Min=10|↩
                     Max=100]"),
21               "Bounds", StringValue("130|190|8|100"));
22      mobility3.Install(c.Get(6));
```

De kablede nodene har posisjonene (x, y, z) (se også 4.3):

- GWne: $n3(140, 8, 0)$, $n4(160, 8, 0)$ og $n5(180, 8, 0)$.
- KNne: $n0(170, 2, 0)$,$n1(145, 3, 0)$,$n2(155, 3, 0)$, $n8(170, 2, 0)$ og $n9(190, 2, 0)$.

Nodetettheten blir da $\frac{2}{60\times92} = 3.6 \times 10^{-4}$.

Vi starter med å vise hva $n1$ sender. Det forventes at $n1$ sender to multikastsrømmer i samsvar med beskrivelse gitt i seksjon 4.2.



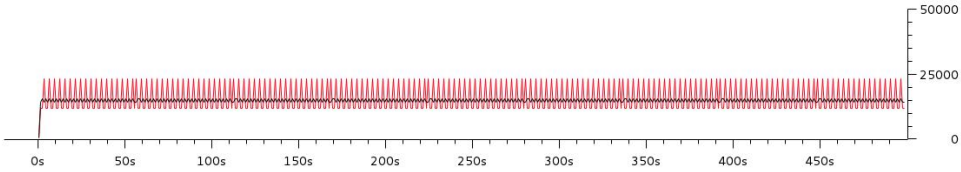
Figur 4.3: Utplassering av noder under simulering. Node $n6$ og $n7$ vil bestanding «høre» $n3$ $n4$ og $n5$. Gjelder for test 1 og test 2.

Forventet resultat

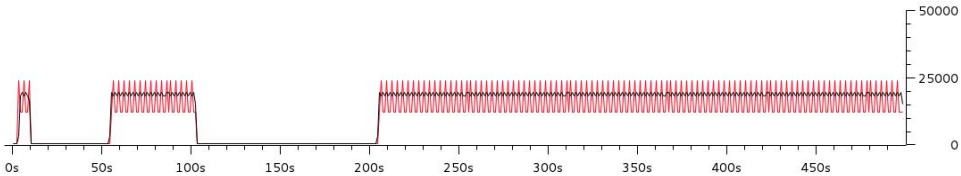
Det var forventet at det ville bli dobbel eller trippel videresendingstilstand i den transiente tilstanden fra start til den tid hvor AIMFnaboer har lært sine naboskaper (den stabile tilstanden). Videre var det forventet at $n5$ ville ha størst videresendingsvilje frem til $50sek$ hvor node $n3$ tar over.Node $n3$ vil videresende frem til $n4$ får en $vilje(n4) = 6$ ved 100sek. Til slutt er det forventet at $n3$ tar over når $n4$s vilje blir redusert til $vilje(n4) = 3$.

Simulering

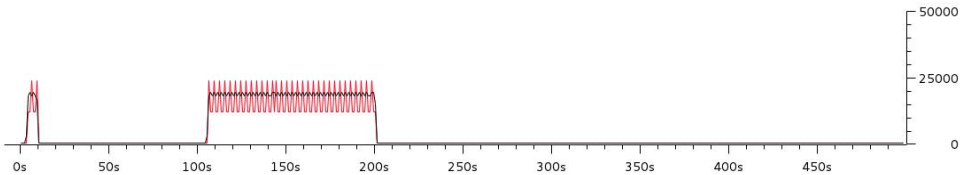
Etter endt simulering ser vi først på *faktisk* videresendt data (se figur 4.4a) og videre hva $n3$ (se figur 4.4b), $n4$ (se figur 4.4c) og $n5$ (se figur 4.4d) sender (multikastgruppe 225.1.2.4 og 225.1.2.5). Til slutt ser vi på mottak hos node $n6$ (se figur 4.4e) og $n7$ (se figur 4.4f).



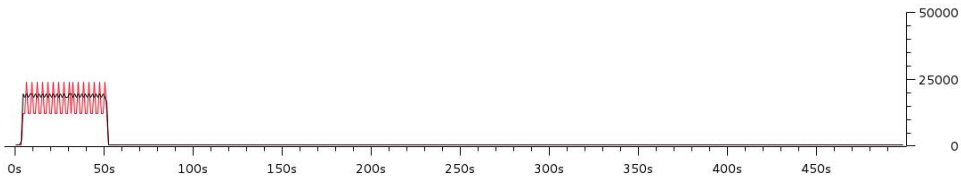
(a) Videresendt pakkerate for $n1$ (kilden). Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)



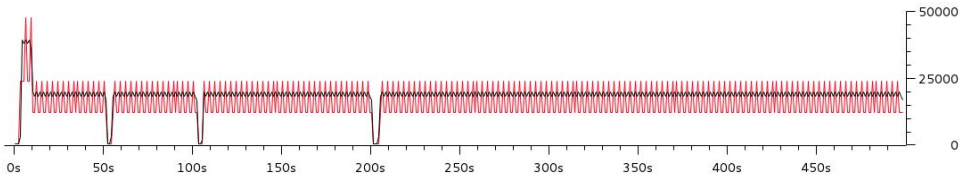
(b) Videresendt pakkerate for $n3$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)



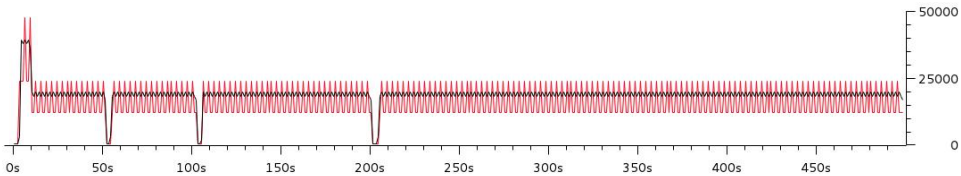
(c) Videresendt pakkerate for $n4$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)



(d) Videresendt pakkerate for $n5$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)



(e) Mottat pakkerate for $n6$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)

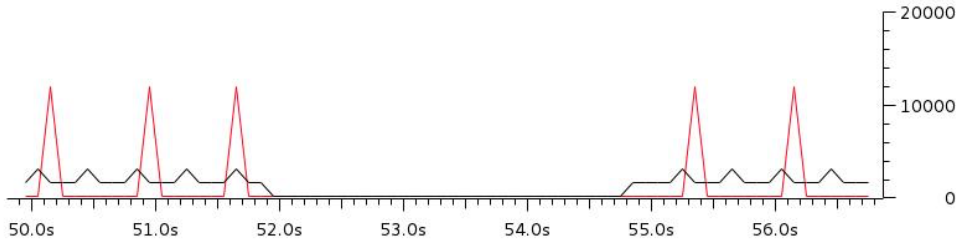


(f) Mottat pakkerate for $n7$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)

Figur 4.4: Data fra test 1.

Faktisk resultat

Som man ser av grafene til $n3$, $n4$ og $n5$ så blir forventingene innfridd. Man ser også at $n7$ og $n6$ mottar to multikaststrømmer som samvarierer positivt med grafen til $n1$ noe som igjen viser at multikasttjenesten blir innfridd i MANETet. Til slutt ser man at også AIMF opptrer sparsomt ettersom $n6$ og $n7$ ikke mottar noen duplikater i den stabile tilstanden (steady state).



Figur 4.5: Mottat pakkerate for $n7$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek) Pakketap ved 50 sekunder.

Ved rundt 50 sekunder ser man an $n5$ slutter og videresende og at $n3$ overtar slik som instruert. Videre ser man da at dette gir utslag hos $n7$ og $n6$. Hos $n7$ (se figur 4.5) varer bortfallet av multikastet trafikk i ca. 3 sekunder (fra 52sek til 55sek.). Denne re-konvergeringstiden skyldes en blandning av en hallomeldings latenstid og en GWs «sjekkintrevalltid» for OLSR-tilgjengelighet ettersom det er her videresendingstilstanden faktisk endres. Så konvergeringstiden ved slike tilfeller er $k = hallomeldinglatenstid + \{x|0sek < x > a\}$ hvor a er OLSR-sjekk intervalltiden (se tabell 4.1). Denne forklaringen gjelder også de andre bortfallene hos $n7$ og $n6$.

4.5.2 Test 2

Her tester vi om multikast tjenester blir gitt til begge MNene over periode på 500sekunder og om GWene tåler at en GWene bli skutt (slutter og fungere). Og følgende instruksjoner blir gitt til AIMF systemet igjennom kontrollskriptet i løpet av simuleringen:

Ved $t = 1$ simuleringen blir $n3$, $n4$ og $n5$ gitt viljevariablene 3, 4 og 2. Videre blir $n3$ konfigurert med 225.1.2.4 ved $t = 3sek$ og 225.1.2.4 ved $t = 4sek$ ut i simuleringen. Node $n4$ stoppes ved $t = 250sek$ og startes igjen ved $t = 350$.

```
1 Simulator::Schedule(Seconds(1.0), &aimf::RoutingProtocol::↩
      ChangeWillingness, aimf_Gw, 3);
2     Simulator::Schedule(Seconds(1.0), &aimf::RoutingProtocol::↩
          ChangeWillingness, aimf_Gw2, 4);
3     Simulator::Schedule(Seconds(1.0), &aimf::RoutingProtocol::↩
          ChangeWillingness, aimf_Gw3, 2);
4     Simulator::Schedule(Seconds(3.0), &aimf::RoutingProtocol::↩
          AddHostMulticastAssociation, aimf_Gw, multicastGroup, ↩
          multicastSource);
5     Simulator::Schedule(Seconds(4.0), &aimf::RoutingProtocol::↩
          AddHostMulticastAssociation, aimf_Gw, multicastGroup2, ↩
          multicastSource2);
6     Simulator::Schedule(Seconds(250.0), &aimf::RoutingProtocol::DoStop, ↩
          aimf_Gw2);
7     Simulator::Schedule(Seconds(350.0), &aimf::RoutingProtocol::DoStart, ↩
          aimf_Gw2);
```

Som sagt er vi interessert i om AIMF systemet «tåler» at en av nodene slutter å operere (vi «dreper[2]» selve aimf-prosessen på en GW). Videre er mobiliteten til node $n6$ og $n7$ lik mobiliteten satt under test 1 og det samme gjelder for multkaststømmene.(se seksjon 4.5.1). Dette gjelder også de kablede GWene og KNene.

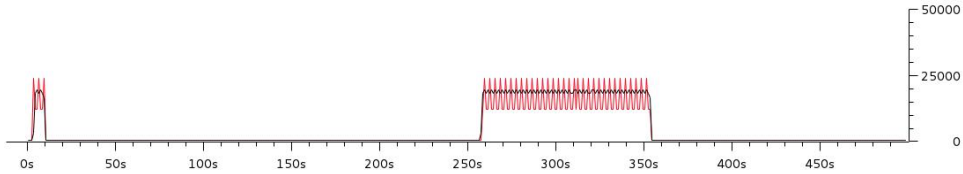
Forventet resultat

Det var forventet at det ville bli dobbel eller trippel videresendingstilstand i den transiente tilstanden fra start til den tid hvor AIMFnaboer har lært sine naboskaper. Videre er det forventet at $n4$ ville ha størst videresendingsvilje frem til $250sek$ hvor node $n3$ tar over. Node $n3$ vil videresende frem til $n4$ får igjen blir starte ved $t = 350sek$.

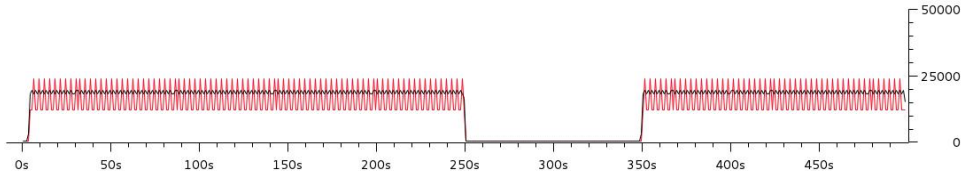
Simulering

Først ser vi på hva $n3$ sender(multikastgruppe 225.1.2.4 og 225.1.2.5)(se figur 4.4b). Videre ser på $n4$ (se figur 4.4c) og $n5$ (se figur 4.4d). Til slutt ser vi på mottak hos node $n6$ (se figur 4.4e) og $n7$ (se figur 4.4f).

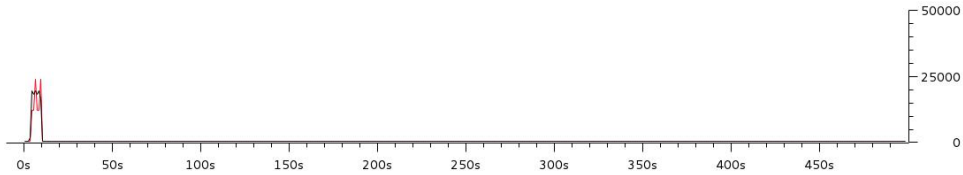
[2]Det blir da ikke sendt noen form for hallomeldinger og eventuell videresending opphører.



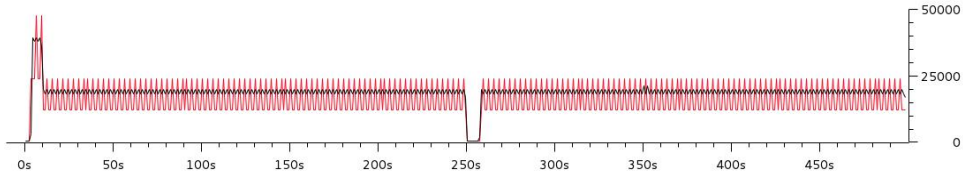
(a) Videresendt pakkerate for $n3$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)



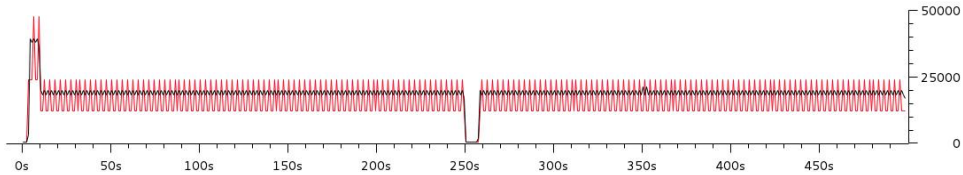
(b) Videresendt pakkerate for $n4$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)



(c) Videresendt pakkerate for $n5$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)



(d) Mottat pakkerate for $n6$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)

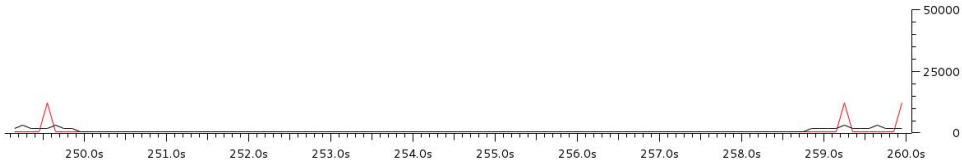


(e) Mottat pakkerate for $n7$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)

Figur 4.6: Data fra test 2.

Faktisk resultat

Som man ser av grafene til $n3$, $n4$ og $n5$ så blir forventingene innfridd. Man ser også at $n7$ og $n6$ mottar to multikaststrømmer som samvarierer positivt med grafen til $n1$ noe som igjen viser at multikasttjenesten blir innfridd i MANETet.



Figur 4.7: Mottat pakkerate for $n7$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek) Pakketap ved 250 sekunder.

Til slutt ser man også at AIMF opptrer robust og sparsomt. Som man ser i figur 4.6e er det ett lengre brudd ved $t = 250sek$ (Se figur 4.7). Dette bruddet blir ikke oppdaget igjennom bruk av OLSR, men ved at de andre nodene ikke mottar hallomeldinger fra $n4$ lengre. Så denne bruddtiden består av AIMFnabo-holdetid + OLSRsjekk-intervalltid hvor sistnevnte kan være alt fra 0 til (5,10) sekunder. AIMFnabo-holdtiden er i denne kjøringen satt til 6 sekunder og hvor OLSRsjekk-intervalltiden også er 6sek (Se tabell 4.1). Til slutt, ettersom AIMFnabo-holdtiden vanligvis blir oppdatert innenfor hallomelding sendeintervallet, så kan denne «soft state'en» - gjeldene en AIMFnabo - bli fjernet innen ca. $6sek - 1.999sek = 4sek$ til 6 sek, etter den sist ble satt. Oppsummert vil det si at denne bruddtiden eller konvergenstiden er $k = \{x|a - b < x > a\} + \{y|0 < y > c\}$ hvor a er AIMFnabo-holdetid, b er hallomelding sendeintervall og c er OLSR-sjekk intervalltiden (se tabell 4.1). Noe som tilsvarer ca. 4 til 12sek med hallomeldinglatenstiden som usikkerhet. Til slutt ved $t = 350$ og $t = 355$ ser man noe overlapp på mottak. Denne overlapptiden følger samme intervall som nevnt. Det skal sies at OLSR kjører på $n4$ slik at overlappen ville vært noe større hvis man i tillegg måtte vente på oppstart av OLSR protokollen på $n4$.

4.5.3 Test 3

Her tester vi om multikasttjenester blir gitt til begge MNene over periode på 500sekunder og om AIMF håndterer partisjonering. Det vi er interessert i her er om AIMF systemet «tåler» at en av AIMF-nodene ikke har rute til hverandre grunnet dekning i MANETet. Altså de har ruter til hverandre i det kablede nettet, men ikke i MANETet. Som nevnt gir OLSR, fordi det er en unikastruteprotokoll, transitive egenskaper med tanke på dataflyt. Så hvis eksempelvis $n3$ har rute til $n7$ og $n7$ har rute til $n4$ så vil $n3$ ha rute til $n4$. Videre er multkaststrømmene lik her som i test 1.(se seksjon 4.5.1).

Ettersom det ikke er implementert SMF i dette eksempelet så er mobiliteten satt opp etter følgende kriterier:

- Node $n6$ og $n7$ er aldri innenfor rekkevidde av hverandre ettersom de ikke videresender multikast.
- Ingen av GWene er innen rekkevidde av hverandre.
- Node $n7$ oppholder seg kun til venstre for $n3$.
- Node $n6$ veksler mellom å være under dekning av $n3$ eller $n4$.

Mobiliteten for MNene og plasseringen for GWene og KNene blir da slik:

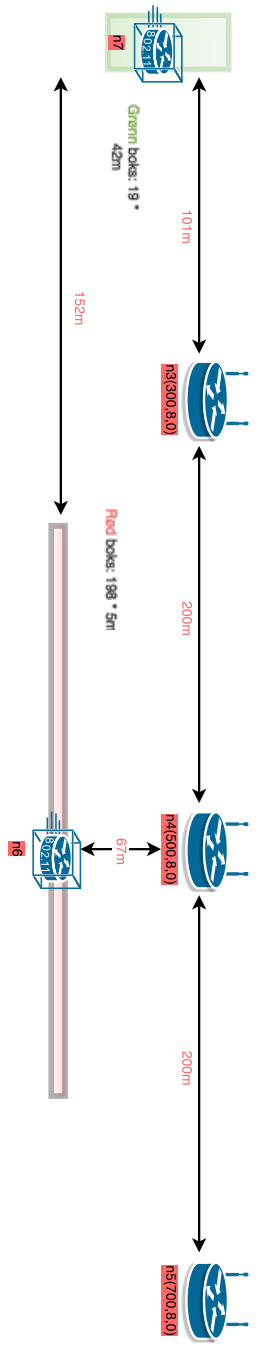
MN $n6$s og $n7$s utgangposisjon er endret på følgene måte: $n6$ beveger seg mellom $n3$ og $n4$ og har aldri kontakt med $n7$. Node $n7$ oppholder seg kun til venstre for $n3$.

Forflytningsrommet deres - etter endt utplassering - er endret til $[351, 450] \times [75, 80]$ og $[180, 199] \times [8, 50]$ for $n6$ og $n7$.

Videre for GWne og KNne:

- GWne: $n3(300, 8, 0)$, $n4(500, 8, 0)$ og $n5(700, 8, 0)$.
- KNne: uendret.

Disse posisjonen er gitt i figur 4.8.



Figur 4.8: Utplassering av noder under simulering (test 3).

```
1
2
3  //----------------MNne----------------
4          int64_t streamIndex = 2356;
5           MobilityHelper mobility;
6           mobility.SetPositionAllocator("ns3::RandomDiscPositionAllocator",
7                   "X", StringValue("190.0"),
8                   "Y", StringValue("25.0"),
9                   "Rho", StringValue("ns3::UniformRandomVariable[Min=0|Max↩
                        =5]"));
10          mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
11                  "Mode", StringValue("Time"),
12                  "Time", StringValue("0.5s"),
13                  "Speed", StringValue("ns3::UniformRandomVariable[Min=10|↩
                        Max=100]"),
14                  "Bounds", StringValue("180|199|8|50"));
15          mobility.Install(c.Get(7));
16          MobilityHelper mobility3;
17          mobility3.SetPositionAllocator("ns3::RandomDiscPositionAllocator",
18                  "X", StringValue("360.0"),
19                  "Y", StringValue("77.0"),
20                  "Rho", StringValue("ns3::UniformRandomVariable[Min=0|Max↩
                        =2]"));
21          mobility3.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
22                  "Mode", StringValue("Distance"),
23                  "Distance", StringValue("200"),
24                  "Direction",StringValue("ns3::UniformRandomVariable[Min=0|↩
                        Max=0]"),
25                  "Speed", StringValue("ns3::UniformRandomVariable[Min=1|Max↩
                        =2]"),
26                  "Bounds", StringValue("351|549|75|80"));
27          mobility3.Install(c.Get(6));
28          ...
29          ...
30
31          ...
32          mobility.AssignStreams(NodeContainer(c.Get(6), c.Get(7)), ↩
                streamIndex);
33
34  //----------------GWne og KNne----------------
35
36          positionAlloc->Add(Vector(300.0, 8.0, 0.0));//gw1-n3
37          positionAlloc->Add(Vector(500.0, 8.0, 0.0));//gw2-n4
38          positionAlloc->Add(Vector(700.0, 8.0, 0.0));//gw3-n5
39          positionAlloc->Add(Vector(170.0, 2.0, 0.0));//n0
40          positionAlloc->Add(Vector(145.0, 3.0, 0.0));//n1
41          positionAlloc->Add(Vector(155.0, 3.0, 0.0));//n2
42          positionAlloc->Add(Vector(170.0, 2.0, 0.0));//n8
43          positionAlloc->Add(Vector(190.0, 2.0, 0.0));//n9
```

Og følgende instruksjoner blir gitt til AIMF systemet igjennom kontrollskriptet i løpet av simuleringen:

Her har vi satt at node $n3$ har lavest vilje lik 2 deretter $n5$ lik 3 og $n4$ lik 4 (alle satt etter 1 sekund). Node $n3$ får konfigurert $(10.1.1.2, 225.1.2.4)$ ved $t = 3sek$ og $n4$ får konfigurert $(10.1.1.2, 225.1.2.5)$ ved $t = 4sek$.

```
1    Simulator::Schedule(Seconds(1.0), &aimf::RoutingProtocol::↩
         ChangeWillingness, aimf_Gw, 4);
2    Simulator::Schedule(Seconds(1.0), &aimf::RoutingProtocol::↩
         ChangeWillingness, aimf_Gw2, 3);
3    Simulator::Schedule(Seconds(1.0), &aimf::RoutingProtocol::↩
         ChangeWillingness, aimf_Gw3, 3);
4    Simulator::Schedule(Seconds(3.0), &aimf::RoutingProtocol::↩
         AddHostMulticastAssociation, aimf_Gw, multicastGroup, ↩
         multicastSource);
5    Simulator::Schedule(Seconds(4.0), &aimf::RoutingProtocol::↩
         AddHostMulticastAssociation, aimf_Gw2, multicastGroup2, ↩
         multicastSource2);
```

Forventet resultat

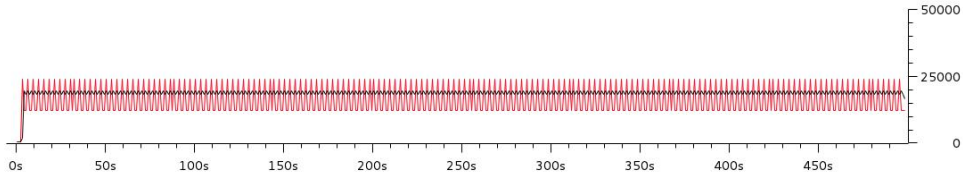
Det er det forventet at GWene har videresendingstilstand når de ikke har rute til en, eller flere, GW/er hvor en, eller flere, GW/er har høyere vilje og når de har en eller flere mottakere i MANETet[3]. Videresendingstilstanden hos den $n4$ vil trolig samvariere med mobiliteten til $n6$ ettersom $n6$ er satt til å veksle mellom $n3$ og $n4$.

Faktisk resultat

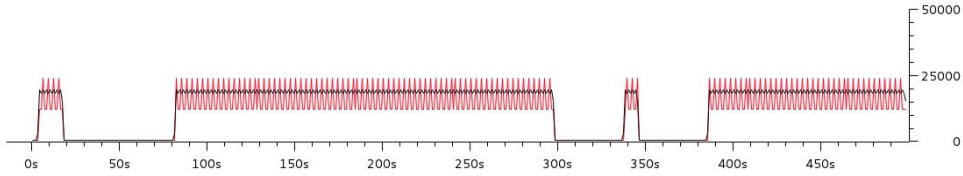
Her ser man at mottaket hos $n6$ har repeterende opphør, mens $n7$ har ett godt mottak. Man ser i figur 4.9 at node $n6$ (se figur 4.9a) endrer $n4$s (se figur 4.9b) videresendingstilstand gjentatte ganger. Videre ser kan man se ut fra figur 4.13 hvem $n6$ mottar data av. Man ser også her ett brudd ved 72 sekunder og 370 sekunder som er konsekvens av at $n6$ beveger seg mellom $n3$ og $n4$.

Simulering

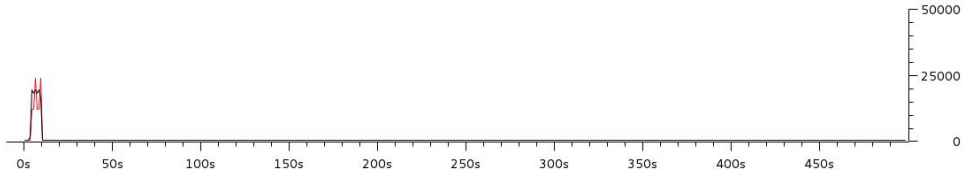
[3]En GW vil ikke videresende hvis OLSRs rutetabell er tom.



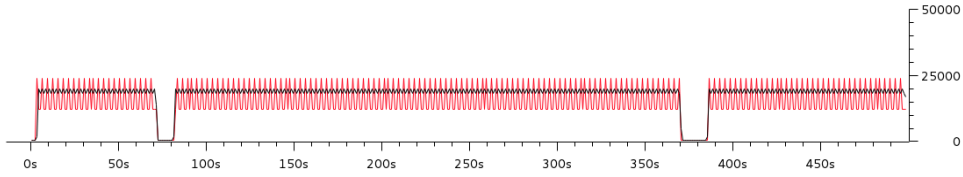
(a) Videresendt pakkerate for $n3$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)



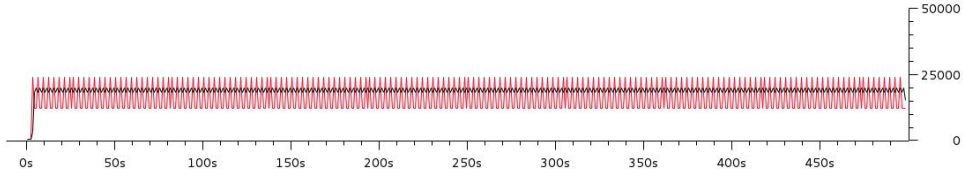
(b) Videresendt pakkerate for $n4$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)



(c) Videresendt pakkerate for $n5$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)



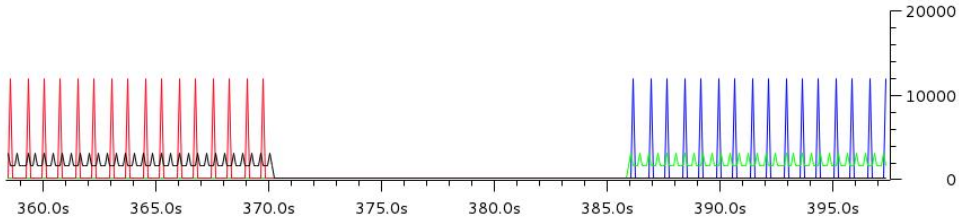
(d) Mottat pakkerate for $n6$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)



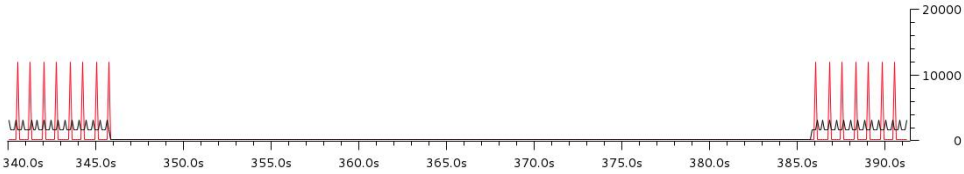
(e) Mottat pakkerate for $n7$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek)

Figur 4.9: Data fra test 3.

Node $n6$ ble plassert mellom $n3$ og $n4$ i starten av simuleringen som resulterer i at $n4$ slutter å videresende etter at transient-perioden (transient state) er passert. Senere ved $t = 72$ ser man at $n6$ beveger seg ut fra dekning av $n3$ og fjerner dermed den transitive relasjonen mellom $n3$ og $n4$. Konsekvensen av dette er at $n4$ ikke har rute til $n3$ lengre og setter dermed videresendingstilstand på de to multikastgruppene. Ved $t = 81$ mottar $n6$ de to multikastgruppene igjen - altså ett opphold på 9sek. Disse 9 sekundene kan forklares med OLSRs konvergenstid og OLSR-sjekk intervalltiden til AIMF (se tabell 4.1. OLSR har en <default» OLSR-naboholdetid på 6sek, et <default» hallomeldingsintervall på 2sek og et «refresh»-intervall på 2sek (Alt er default med tanke på OLSR i simuleringen). Videre er AIMFs OLSR-sjekk-intervalltid i denne konteksten lik $\{y|0 < y > 8\}$sek. Så konvergeringstiden $k = \{x|a - b < x > a\} + \{x|0 < x > c\} + \{y|0 < y > d\}$ hvor a er OLSR-naboholdetid, b er OLSR-hallomeldingsintervall, c er «refresh»-intervallet og d er OLSR-sjekk intervalltiden gjeldene for $n4$ (se tabell 4.1). Det gir en tid fra 4sek+ og opp til 16sek Denne forklaringen gjelder også for bruddet ved $t = 370sek$ (se figur 4.10 og figur 4.11).

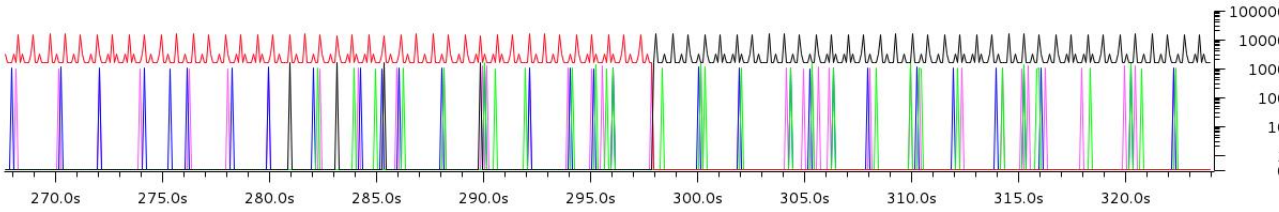


Figur 4.10: Mottat pakkerate for $n6$. Svart linje er 225.1.2.4 og 225.1.2.5 - begge sendt fra $n3$ (bits/sek). Rød linje er 225.1.2.4 og 225.1.2.5 - begge sendt fra $n4$ (bits/sek). Pakketap ved 370 sekunder.



Figur 4.11: Videresendt pakkerate for $n4$. Rød linje er 225.1.2.4, mens svart linje er 225.1.2.5. (bits/sek) Stopp ved 346 sekunder.

Ved $t = 300$, får man ett skifte mellom $n4$ og $n3$ med tanke på $n6$s multikastmottak. Her det da følgelig ingen konvergenstid overhode, med tanke på $n6$s mottak, ettersom $n3$ har høyest vilje i dette AIMFsystemet noe som igjen fører til at når $n6$ beveger seg inn i $n3$s dekningsområde vil den motta disse gruppene «momentant». Med sistnevnte hendelse i minne ser man også at $n4$ stopper sin videresendigstilstand ved $t = 297.6$ så fort $n6$ «binder» $n3$ og $n4$ sammen.



Figur 4.12: Mottatt pakkerate for $n6$. Svart linje er 225.1.2.4 og 225.1.2.5 - begge sendt fra $n3$ (bits/sek (log)). Rød linje er 225.1.2.4 og 225.1.2.5 - begge sendt fra $n4$ (bits/sek (log)). Grønn linje er OLSR-hallomeldinger sendt fra $n3$, blå fra $n4$ og lilla fra $n6$ selv (bits/sek (log)). Node $n6$s bytte fra $n4$ til $n3$.

For å forklare overlappen fra $n4$ til $n3$ nærmere må vi se på OLSRs kontrollmeldinger(se figur 4.12). Ut fra denne figuren - ved lav t og til venstre i figuren - ser man at $n6$ kun mottar OLSR-kontrollmeldinger fra $n4$ noe som igjen betyr at $n6$ trolig ikke annonserer rute for $n4$ til $n3$ og motsatt. Først ved $t = 281$ ser man at multikastrafikk fra $n3$ ankommer $n6$ og ved $t = 282$ den første OLSR-hallomeldingen fra $n3$. Vi kan anta at $n6$ har slettet sin OLSR-assosiasjon vedrørende $n3$ så ved mottak lages det en ny assosiasjon for $n3$. Nå vil $n6$s OLSR-instans først oppdatere sin linkliste før den oppdaterer sin naboliste med $n3$-assosierte data. MNene i denne konteksten genererer[4] Multiple Interface Declaration (MID) [15] meldinger, men ikke Host Network Assocation (HNA) [15] meldinger - grunnet fravær av annonserte nabo-IPnett. Derfor trenger vi bare å bry oss om OLSR-hallomeldinger [15] og noe mindre om MID-meldinger samt Topologi-meldinger (TC) [15]. Ettersom MID-meldinger feilaktig[5] blir sendt i denne konteksten så ignorerer vi de eksplisitt, men implisitt vil de jo følgelig påvirke eksempelvis OLSRs intrevallttider, generelt pakkemottak, generelt pakkestørrelse og lignende. TC meldinger er viktige for å oppnå ruting over flere en 2 hopp, men her er ikke dette aktuelt ettersom det bare er $n6$ som «lever» mellom $n3$ og $n4$.

Den første meldingen - ved $t = 282$ - som $n6$ mottar fra $n3$ er en hallo-melding som inneholder $n3$s linktilstand gjeldene $n7$ - som er symmetrisk. Node $n6$ sender så en hallomelding rett etter mottak[6] av $n3$s ved $t = 282+$ så $n6$ rekker ikke å prosessere sistnevnte hallomelding. Videre har nå $n3$ mottatt en hallomelding fra $n6$ og legger dermed til ett asymmetrisk linksett vedrørende $n6$ i sin neste hallomelding som sendes ved $t = 284$. Node $n6$s hallomelding - etter endt prosessering av $n3s$ første hallomelding - vil inneholde ett symmetrisk linksett gjeldene $n4$ samt ett asymmetrisk linksett vedrørende $n3$ som «svar» på $n3$s fraværende linksett vedrørende

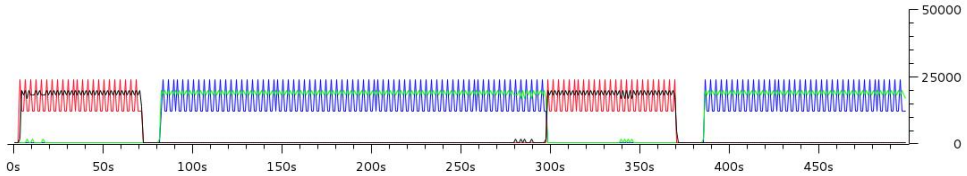
[4]Ns3s OLSR versjon sender MID meldinger - noe som ikke skal skje ettersom hver AIMF-GW har kun ett aktivt MANET-grensesnitt. I kontrollskriptet er OLSR instruert til å ikke bruke det kablede grensesnittet ei heller «loopback»-grensesnittet, men fortsatt blir det sendt MID-meldinger.

[5]Se fotnote 4.

[6]ca. 0.08sek etter mottak av $n3$s hallomelding

$n6$ (hallomeldingen blir sendt ved $t = 284+$). Når denne hallomeldingen ankommer $n3$ vil $n3$ igjen sende en hallomelding etter en viss tid[7], men i motsetning til sistnevnte hallomelding fra $n3$ vil denne også inneholde en symmetrisk linksett,eller rettere sakt ett Multipoint Relay (MPR) [15] linksett, vedrørende $n6$.

Etter at $n6$ har avsluttet denne «hand shake»'en mellom $n3$ og $n6$ ved å sende en hallomelding ved $t = 285$ med ett MPR linksett vedrørende $n3$ har begge disse MNne en symmetrisk relasjon seg imellom og samtidig valgt inn i MPR settet i denne konteksten. Nå har disse to MNne en symmetrisk link seg imellom, men $n3$ har fortsatt ingen transitiv relasjon med $n4$, noe som må til for at $n4$ skal avbryte sin videresending. Ruter som har kost 1 (1-hopp) og 2 (2-hopp) blir i OLSR løst ved at man bruker mottatte hallomeldingers linksett som ikke allerede er 1-hopps ruter for mottakkeren og som er symmetriske til å generere 2-hopps ruter lokalt. Så $n4$ vil utlede fra $n6$s hallomeldinger at $n3$ er tilgjengelig.



Figur 4.13: Mottatt pakkerate for $n6$. Rød linje er 225.1.2.4 og svart linje er 225.1.2.5 - begge sendt fra $n3$ (bits/sek). Blå linje er 225.1.2.4 og grønn linje er 225.1.2.5 - begge sendt fra $n4$ (bits/sek). Pakketap ved 72 og ved 370 sekunder.

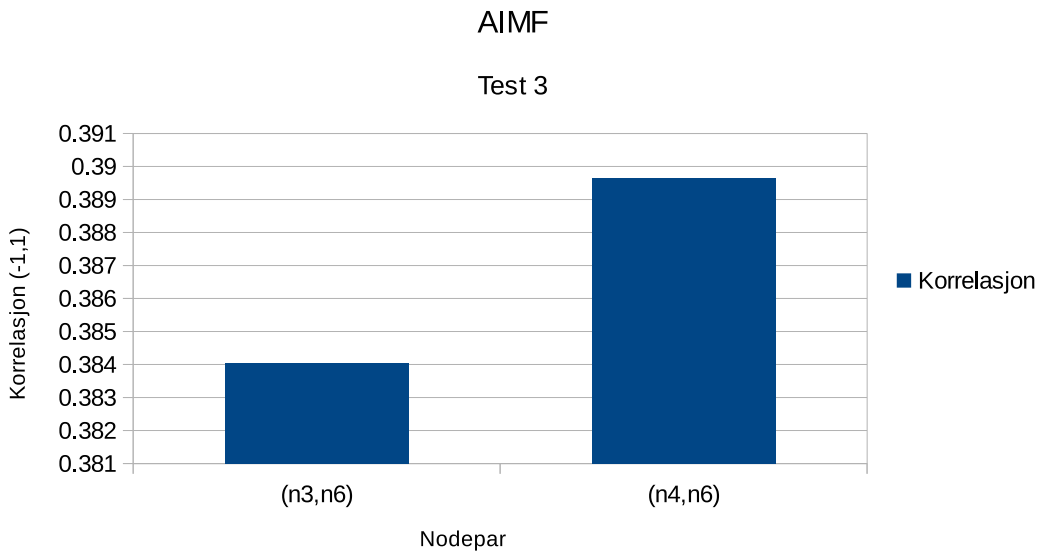
Videre ser man ut fra figur 4.12 og figur 4.13 at $n6$ mottar noe overlapptrafikk ved $(280 < t > 290)$ fra de to multikastgruppene, spesielt svart som er gruppe 225.1.2.5. Egentlig skulle $n4$ avbrutt videresending sin her, men AIMF rekker ikke å reagere. Node $n4$ har som nevnt en OLSR-sjekk intervalltid (AIMF) (se tabell 4.1 på 8sek (5+3). AIMF instansen på $n4$ kan utlede at $n3$ er tilgjengelig ved $t = 285.8+\{x|0 < x > t_r\}+t_p$ hvor t_r er intervallet mellom prosessering av tilstand endret av (hallo,MID,TC ellerHNA)-meldinger. Og hvor t_p er tiden linksett-, nabotabell- og rutetabelloppdatering og lignende i OLSR tar. Videre er AIMFs reaksjons tid med tanke på ruten til $n3$ lik $\{x|0+ < x > 8\}$sek. Vi ser av figur 4.12 at $n6$ ikke lenger får multikasttrafikk sendt fra $n4$ ved $t = 297.84$[8]. En annen faktor kan også være buffere og/eller «backoff»-mekanismer med tanke på Collision Avoidance (CA) [4] i 802.11 stakken generelt og spesielt når det er såpass små pakker som her [40] [26].

I figur 4.13 ser man også at det er overlapp ved $t = 340$sek noe som skyldes at $n6$ veksler mellom $n6$s og $n7$s «OLSR-dekning».

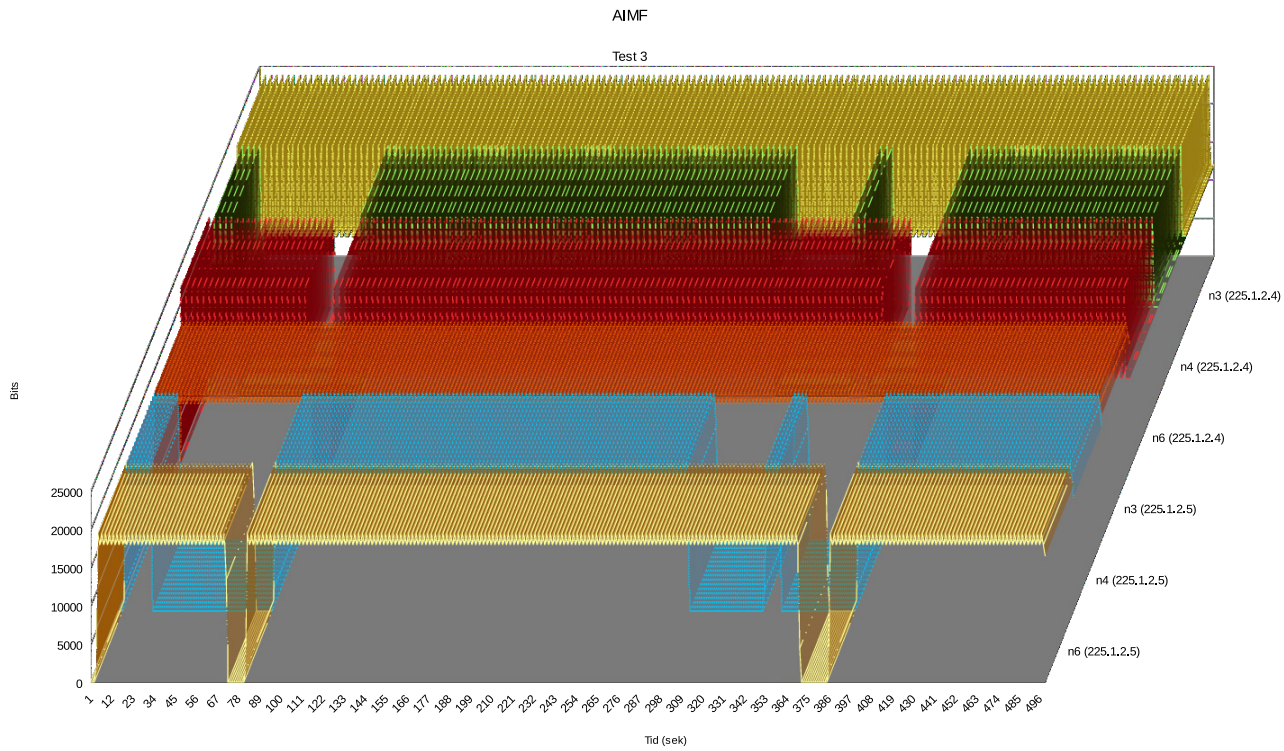
[7]Denne meldingen er ikke nødvendig for å skape denne symmetriske relasjonen mellom $n3$ og $n6$. Dette kan de utlede fra allerede mottatte hallomeldinger.

[8]Node $n4$ sender sin siste multikastpakke - gjeldene denne overlappen - ved $t = 297.77$

Det er regnet ut korrelasjonskoeffisienter for denne testen som viser korrelasjonen mellom $n3$ og $n6$ samt $n4$ og $n6$ for å gi en ekstra sikkerhet rundt resultatet (se figur 4.14). Til slutt er det også laget en oversikt over hele testen som viser mottak(se figur 4.15).



Figur 4.14: Korrelasjon mellom hver (n3,n6) og (n4,n6).



Figur 4.15: Sammenligningsdiagram.

4.5.4 Test 4

Her tester vi om multikasttjenester blir gitt til begge MNene over periode på 500sekunder, om AIMF håndterer partisjonering og vi kjører SMF på GW og MNer. SMF ble implementert helt på slutten av prosjektet så derfor er ikke protokollen brukt i tidligere tester. Det vi er interessert i her er er å se AIMF i samspill med SMF. Altså de har ruter til hverandre i det kablede nettet, men ikke i MANETet. Som nevnt gir OLSR, fordi det er en unikastruteprotokoll, transitive egenskaper med tanke på dataflyt. Så hvis eksempelvis $n3$ har rute til $n7$ og $n7$ har rute til $n4$ så vil $n3$ ha rute til $n4$.

Multikast strømmene er ikke likt konfigurert som i test3. Forskjellen er pakkestørrelse og bittraten. For 225.1.2.4 er pakkestørrelsen 1300$byte$ og bittraten 11.1$Kbit/sek$, og for 225.1.2.5 er pakkestørrelsen 1400$byte$ og bittraten $11,9 Kbit/sek$

Videre har vi også brukt et annen «klasse» for å sende multikasttrafikk slik at hver UDP-pakke som blir sendt er unik. Se kildekode:

```

  UdpClientHelper cl(multicastGroup, multicastPort);
    cl.SetAttribute("MaxPackets", UintegerValue(1000));
    cl.SetAttribute("PacketSize", UintegerValue(1300));
    UdpClientHelper cl2(multicastGroup2, 1336);
    cl2.SetAttribute("MaxPackets", UintegerValue(1000));
    cl2.SetAttribute("PacketSize", UintegerValue(1400));

    ApplicationContainer srcC = cl.Install(c0.Get(1));
    ApplicationContainer srcC2 = cl2.Install(c0.Get(1));

    srcC.Start(Seconds(1.));
    srcC.Stop(Seconds(499.));
    srcC2.Start(Seconds(1.));
    srcC2.Stop(Seconds(499.));
```

GW videresender nå multikasttrafikk som ankommer trådløst, og det samme gjør MNer. Derfor setter vi mobiliteten for MNene og plasseringen for GWene og KNene slik:

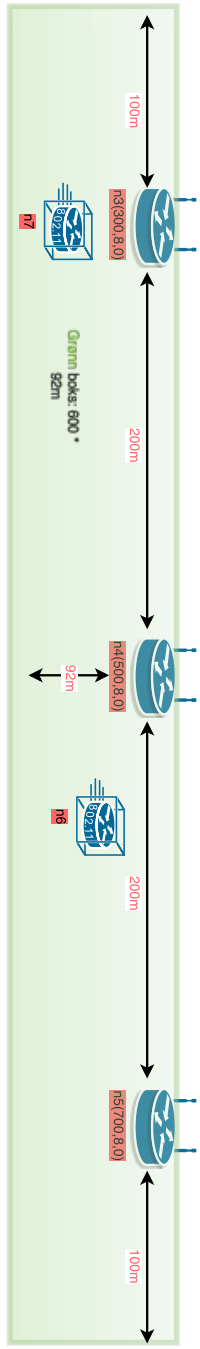
Ettersom vi nå bruker SMF kan vi la testområdet være større, fordi den relative dødsonen vil være mindre med tanke multikastmottak. Forflytningsrommet deres - etter endt utplassering - er dermed endret til $[200, 800] \times [8, 100]$ og $[200, 800] \times [8, 100]$ for $n6$ og $n7$.

Nodetettheten blir da $\frac{2}{600 \times 92} = 3.6 \times 10^{-5}$.

Videre for GWne og KNne:

- GWne: $n3(300, 8, 0)$, $n4(500, 8, 0)$ og $n5(700, 8, 0)$.
- KNne: uendret.

Disse posisjonen er gitt i figur 4.16.



Figur 4.16: Utplassering av noder under simulering (test 4).

```
1
2
3  //----------------MNne----------------
4          int64_t streamIndex = 2357;
5           MobilityHelper mobility;
6           mobility.SetPositionAllocator("ns3::RandomDiscPositionAllocator",
7                   "X", StringValue("400.0"),
8                   "Y", StringValue("25.0"),
9                   "Rho", StringValue("ns3::UniformRandomVariable[Min=0|Max↩
                        =5]"));
10          mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
11                  "Mode", StringValue("Time"),
12                  "Time", StringValue("0.5s"),
13                  "Speed", StringValue("ns3::UniformRandomVariable[Min=10|↩
                        Max=100]"),
14                  "Bounds", StringValue("200|800|8|100"));
15          mobility.Install(c.Get(7));
16          MobilityHelper mobility3;
17          mobility3.SetPositionAllocator("ns3::RandomDiscPositionAllocator",
18                  "X", StringValue("360.0"),
19                  "Y", StringValue("77.0"),
20                  "Rho", StringValue("ns3::UniformRandomVariable[Min=0|Max↩
                        =2]"));
21          mobility3.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
22                  "Mode", StringValue("Time"),
23                  "Time", StringValue("0.5s"),
24                  "Speed", StringValue("ns3::UniformRandomVariable[Min=10|↩
                        Max=100]"),
25                  "Bounds", StringValue("200|800|8|100"));
26          mobility3.Install(c.Get(6));
27          ...
28          ...
29
30          ...
31          mobility.AssignStreams(NodeContainer(c.Get(6), c.Get(7)), ↩
                streamIndex);
32
33  //----------------GWne og KNne----------------
34
35          positionAlloc->Add(Vector(300.0, 8.0, 0.0));//gw1-n3
36          positionAlloc->Add(Vector(500.0, 8.0, 0.0));//gw2-n4
37          positionAlloc->Add(Vector(700.0, 8.0, 0.0));//gw3-n5
38          positionAlloc->Add(Vector(170.0, 2.0, 0.0));//n0
39          positionAlloc->Add(Vector(145.0, 3.0, 0.0));//n1
40          positionAlloc->Add(Vector(155.0, 3.0, 0.0));//n2
41          positionAlloc->Add(Vector(170.0, 2.0, 0.0));//n8
42          positionAlloc->Add(Vector(190.0, 2.0, 0.0));//n9
```

Og følgende instruksjoner blir gitt til AIMF systemet igjennom kontrollskriptet i løpet av simuleringen:

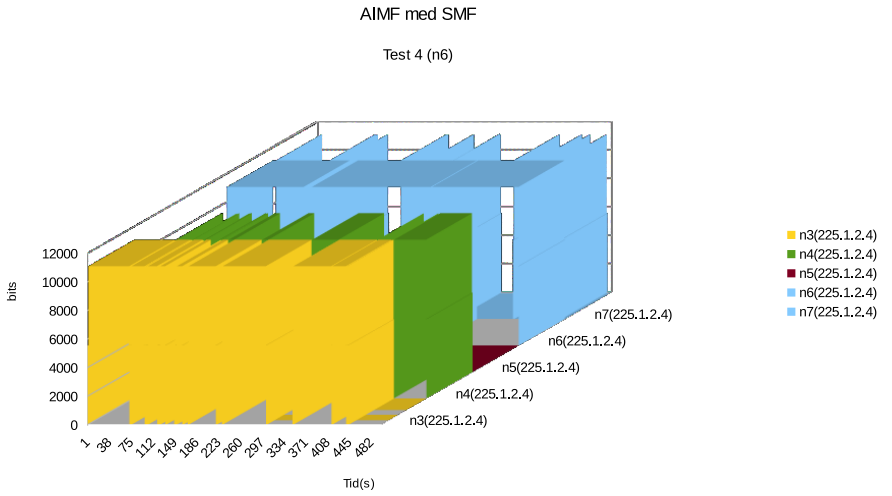
Her har vi satt at node $n3$ har lavest vilje lik 2 deretter $n5$ lik 3 og $n4$ lik 4 (alle satt etter 1 sekund). Node $n3$ får konfigurert $(10.1.1.2, 225.1.2.4)$ ved $t = 3sek$ og $n4$ får konfigurert $(10.1.1.2, 225.1.2.5)$ ved $t = 4sek$.

```
1  Simulator::Schedule(Seconds(1.0), &aimf::RoutingProtocol::↩
       ChangeWillingness, aimf_Gw, 4);
2  Simulator::Schedule(Seconds(1.0), &aimf::RoutingProtocol::↩
       ChangeWillingness, aimf_Gw2, 3);
3  Simulator::Schedule(Seconds(1.0), &aimf::RoutingProtocol::↩
       ChangeWillingness, aimf_Gw3, 3);
4  Simulator::Schedule(Seconds(3.0), &aimf::RoutingProtocol::↩
       AddHostMulticastAssociation, aimf_Gw, multicastGroup, ↩
       multicastSource);
5  Simulator::Schedule(Seconds(4.0), &aimf::RoutingProtocol::↩
       AddHostMulticastAssociation, aimf_Gw2, multicastGroup2, ↩
       multicastSource2);
```

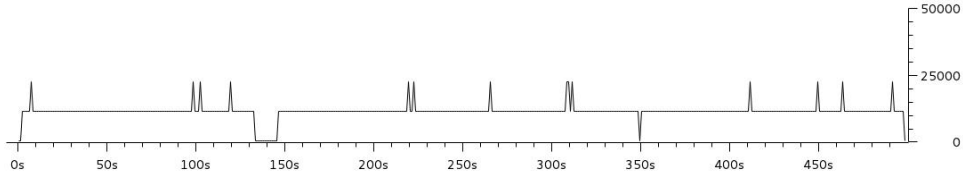
I resultatene som følger tarv vi bare for oss 225.1.2.4 gruppen, ettersom den viser utfallet av testen like godt som om man skulle ha vist begge gruppene. Og vi viser kun mottak og sending for $n6$ og $n7$.

Forventet resultat

Det er det forventet at GWene har videresendingstilstand når de ikke har rute til en, eller flere, GW/er hvor en, eller flere, GW/er har høyere vilje og hvis en GW har en eller flere MNer i sitt dekningsområde. En GW som er ikke når andre GWer via OLSR og som ikke har MNer i sitt dekningsområde vil ikke videresende. Videresendingstilstanden hos $n4$ og $n5$ vil trolig samvariere med mobiliteten til $n6$ og $n7$. Noe overlapp må man forvente når det gjelder mottak hos $n6$ og $n7$ etter som SMF kjører. Det er også forventet at $n6$ og/eller $n7$ har perioder hvor de ikke mottar multikasttrafikk. Grunnen til dette er at denne implementasjonen av SMF «hasher» alle innkommende pakker. Ettersom alle GWer under denne simuleringen mottar multikasttrafikk uansett om de har videresendingstilstand eller ikke. Anta at $n3$ har høyest vilje av $n4$ og $n5$. Disse tre nodene mottar multikast på sine respektive kablede grensesnitt og de tre SMF-instansene «hasher» hver «nye» mottatte pakke. Videre videresender $n3$ pakke x og $n6$ «hasher» x og videresender x på nytt slik at den ankommer $n4$. Node $n4$ vil da ikke videresende x på nytt, fordi $n4$ allerede har mottatt pakke x på sitt kablede grensesnitt. Den løsningen er god hvis det brukes «join»-baserte multikastruteprotokoller i det kablede nettet.



(a) Motttat pakkerate for $n6$. (bits/sek)

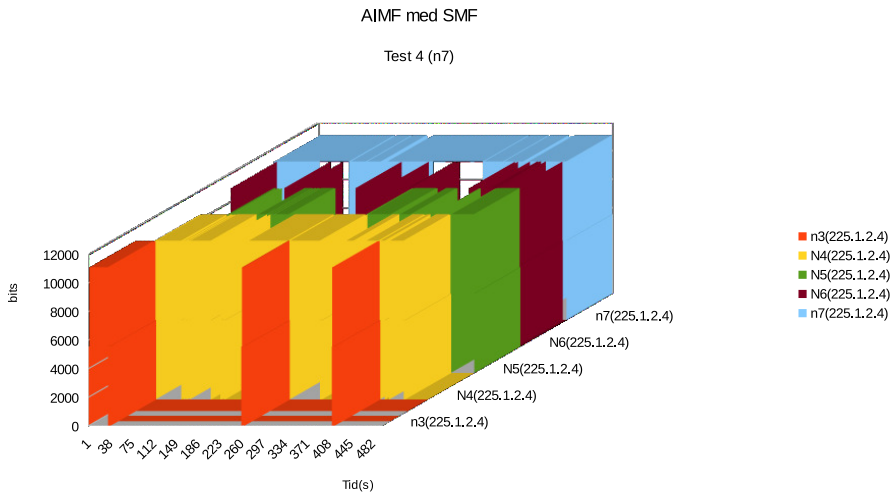


(b) Mottatt pakkerate for $n6$ (kumulativ). (bits/sek)

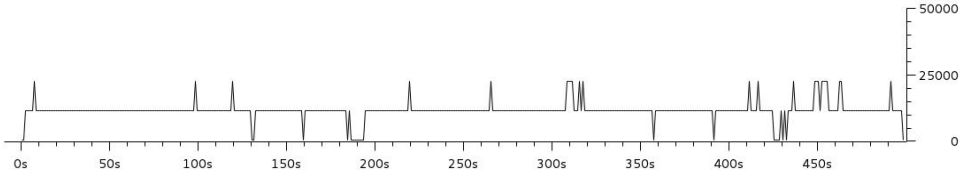
Figur 4.17: Datamottak $n6$ (test 4).

Faktisk resultat

Figur 4.17 og figur 4.18 viser resultatene. I figur 4.17a og figur 4.18a ser man at $n7$ har vandret mellom alle GWene, mens $n6$ bare har vært innom $n3$ og $n4$ (Grafen til n6 i figur 4.17a viser sendt trafikk fra $n6$ som som tilsvarer $n6$s mottak minus duplikater. Det samme gjelder for $n7$ i figur 4.18a). I figur 4.17b og figur 4.18b ser man at $n6$ og $n7$ har noe dobbelt pakkemottak og noe tap. Tap-tiden skyldes prossesser som er forklart under test 3 (jfr. seksjon 4.5.3) og det doble pakkemottaket skyldes SMF (se figur **??**). Dobbelt pakkemottak skjer når $n6$ og $n7$ er innenfor dekning av hverandre og er under dekning av samme GW.



(a) Motttat pakkerate for $n7$. (bits/sek)



(b) Mottatt pakkerate for $n7$ (kumulativ). (bits/sek)

Figur 4.18: Datamottak $n7$ (test 4).

Kapittel 5
Erfaringer

I dette kapittelet går vi igjennom erfaringene som er gjort i løpet av oppgaven. Først går vi igjennom en overordnende erfaringer og deretter følger en mer spesifikk redegjørelse.

5.1 Overordnende erfaringer

Vi startet med å lage en såkalt ns3modul likedan som OLSR, Routing Information Protocol (RIP) [50] og Internet Control Message Protocol (ICMP) [71] er slike moduler. Videre ble det brukt mye tid på hvordan ns3 var satt sammen så vi så mye på de forskjellige protokollene som arvet den abstrakte Ipv4RoutingProtocol klassen. Det ble her erfart at nesten alle protokollene fulgte den strukturen som Ipv4StaticRouting brukte (funksjonskall og lignende). Hvis man tar for seg den statiske rutetabellen og de funksjoner for å oppdatere den, vil man da videre legge til, og/eller endre, funksjoner som gjør den til en dynamisk ruteprotokoll.

Over snakker vi om unikastruting, så ved multikastruting må man eksempelvis fokusere på kilden og ikke destinasjonen (praktisk sett kilderuting). AIMF er en ruteprotokoll som ruter pakker ved hjelp av primært det satte «RPF-grenesnittet» (alltid det kablede grensesnittet som er en del av et multikastintradomene i denne konteksten) og sekundert kilden hvis det er mulig (det er en kjensgjerning at kilden kan være satt til 0.0.0.0).

Det må også nevnes at i ns3 er det «enormt» mange strukturer som *må* og/eller *bør* arves. Eksempelvis noen av disse forholdene - mellom abstrakte klasser og vanlige klasser med arv - bruker tilnærmet alle muligheter - av polymorfisme - gitt innenfor C++ før C++11 [38] standarten. Derfor bør man kjenne C++ godt for og i hele tatt forstå noe av de mest innfløkte løsningene gjort i ns3. Ett eksempel er hvordan man modulerer pakker i ns3 og ett annet er hvordan IPv4-tilstanden per ns3::Node lever og blir brukt i og under en simulering. Praktisk sett ga denne forståelsen oss

muligheten til dedusere hvilke objekter man fikk tak i, eller kunne lage en «hook» på, innefor aimfs og ns3s *namespace*.

Det er grunnlag for å kommentere dokumentasjonen som har vært grunnlag for arbeidet. Dokumentasjonen rundt tracing [63] og generelt ruting [61] synes vi er mangelfull. Førstnevnte nevner bare hvilke trace-muligheter som er tilgjengelige og hvordan de brukes, men ikke noe hvordan man lager egne trace'ere og eventuelt prober. Sistnevnte sier bare noe om hvordan man bruker ruting i ett kontrollskript og noe om tilgjengelige ruteprotokoller - ikke hvordan man går frem for å lage en ruteprotokoll. Alt dette må man selv utlede fra kildekoden.

5.2 Spesifikke erfaringer vedrørende AIMF

De spesifikke delene i dette kapittelet vil være delt inn i regioner likt som i seksjon 3.2.2.

5.2.1 Init og Stopp

Init og stopp var forholdsvis enkelt å implementere ettersom dette er forholdvis likt i alle ns3s ruteprotokoller. Det eneste som brøt med «malen» her var at det ble laget en «ekstra» funksjon som man brukte til å stoppe AIMFprotokollen på en AIMF-GW. Grunnen til dette er at DoDispose funskjonen er satt til «protected» i Ipv4Rouringprotocol - grunnet arv - og kan dermed ikke kalles fra kontrollscriptet[1]. En annen grunn er at DoDispose gjør at protokollen ikke kan startes igjen i en simulering. DoDispose er ment brukt i opprydding - av minne - etter endt simulering (Sletter referanser og så videre). Sammen med DoStop ble også DoStart laget.

5.2.2 Videresending

Selve arkitekturen for videresending i ns3 er godt beskrevet og funksjonskallet er allerede gitt av Ipv4RoutingProtocol.

Videresending intern Når det gjelder intern videresending ble det ble først brukt en linklokalmultikastadresse, men pakker sendt med en slik adresse *skal* gå på «alle» tilgjengelige grensesnitt per ruteprotokoll (eks. PIM og EIGRP bruker slike adresser). Så for å ikke bryte med definisjonen ble en adresse i det administrative multikastadresserommet valgt for å sende hallomeldinger. Og det fordret samtidig at man måtte utarbeide en rute ved oppstart av hver AIMF-GW. Dette er en av grunnene til at man må sette ekskluderte grensesnitt og ett spesifikt grensesnitt mot MANETet.

[1] I teorien er dette fullt mulig hvis man legger kontrollscriptet i sammen mappe som funksjonen lever, men det er klart at dette ikke er meningen.

Videresending ekstern Som nevnt under videresendingsseksjonen er også selve rutingen av fjerntkommende pakker i ns3 relativt lett å sette seg inn i.

5.2.3 Eventer

Eventlisten er godt beskrevet og det var viktig å få med seg at denne gjennomsyrer alt i ns3. Så erfaringen er å bruke den der det er mulig.

5.2.4 Sjekk av OLSRs rutetabell

Vi erfarte at å bruke OLSR til å sjekke tilgjengenelighet mellom AIMF-GWer i MANETet var en god løsning. Man sjekker da om OLSR har en datarute til andre AIMF-GWer. Vi lagde en «hook» på den lokale OLSR instansen i DoInitialize funksjonen og spurte deretter etter rutetabellen i gitte intervaller per AIMF-GW.

Vi ønsket også å lage samme funksjonalitet i det kablede nettet og fant ut at ved å utnytte RP-baserte intradomene multikastruteprotokoller kunne man oppnå dette. (Se seksjon 3.2.2 og under «Sjekk av OLSRs rutetabell».) Vi fikk ikke testet dette i ns3 ettersom slike protokoller ikke er implementert, så det er følgelig bare en antagelse.

5.2.5 Sending av hallomeldinger

Slik funksjonalitet er ikke dokumentert i ns3 så alt er utledet fra eksisterende kildekode. Vi brukte OLSR som rettesnor og tilpasset OLSRs egen pakkestruktur til vårt eget bruk. I OLSR er det løst på samme måte som i nesten alle andre protokoller eksempelvis RIP, ICMP4 og 6 og alle andre protokoller som sender og mottar pakker seg imellom. En interessant ting her at ns3::Socket - med satt callback - ikke gir programmereren eksplisitt tilgang til IP-hodet. Dette ble erfart ettersom OLSRs pakkeoppsett fremstod som merkelig ettersom mye av dataene her kan hentes ut av ip-hodet. Det kan dog tenkes at man ikke trenger å være låst til IP i fremtidige versjoner, men man har ikke funnet noen referanser som bekrefter dette. Det må komme frem at det trolig ikke er programmererne av OLSR som har valgt å gjøre det slik, de har bare fulgt kodenormen til ns3. Dette gjelder da ikke avsenderadressen ettersom denne ikke trenger å være lik hovedadressen til AIMF-GW (eks. når en AIMF-GW har flere grensesnitt), men kanskje man kunne ha brukt CRC og lignende for å sjekke pakkens integritet.

5.2.6 Mottak av hallomeldinger

Det samme gjelder her som under sending av hallomeldinger. Det er erfart at hvis man først får sendt en hallomelding så er det en «enkel» sak å ta i mot den ettersom

det er samme struktur som blir brukt. Altså har man først serialisert en pakke så er det forholdsvis enkelt å deserailisere, eller objektifisere den.

5.2.7 Oppdatering av rutetabell

Oppdatering av rutetabell er også ganske rett frem ettersom ns3rammeverket inneholder datastukturer som holder på eksempelvis multikastruter og lignende. Så kort oppsummert går man bare igjennom lokale og fjerntkommende multikast par og legger dem til i en multikastrutestruktur - dette blir da rutetabellen. Det skal nevnes at ved fravær av den klassiske *routecach'en* så trenger man egentlig ikke en rutetabell. Man kunne ha regnet, eller funnet, en rute per pakkeankomst. En oppdatering av rutetabellen er avhengig av at ekskluderte grensesnitt og det spesifikke grensesnittet mot MANETet er satt.

5.3 SMF

SMF ble utarbeidet helt i slutten av prosjektet og er derfor ikke spesielt godt dekket under Implementasjonskapittelet. Fremgangsmåten var lik som under bygging av AIMF hvor man arvet Ipv4RoutingProtocol klassen.

5.4 Utvalgte erfaringer under design

Vi følger samme inndeling her som i seksjon 5.2.

5.4.1 Videresending ekstern

Man endte opp med med at en boolsk verdi styrte videresendingstilstanden med hjelp av OLSR, men det ble først prøvd en annen løsning: Først antar vi at en AIMF ruter videresender multikastet trafikk og at alle andre AIMF rutere ser denne trafikken på sitt MANET grensesnitt. I denne situasjonen vil de to AIMF-ruterne som bare «ser» oppdatere og dermed «utsette» ett eventlisteinnslag når en multikastet pakke ankommer. Denne multikastede pakken må også stemme overens med den distribuerte rutetabellen slik at det i praksis er ett eventlisteinnslag per aktive multikaststrøm per rutetabellinnslag. Nå antar vi at multkaststømmen opphører. De to sistnevnte ruterne sine eventlisteinnslag gjeldene den opphørte multikaststrømmen vil da effektueres - ettersom de ikke blir oppdatert og utsatt lengre. Effekten av dette er at en eller to vil starte å videresende den multikastede trafikken.

Her har vi en løsning som muliggjør AIMF-GWer til å ta over videresendingsansvar etter opphørt multikastmottak, men man kan ikke være sikker på om to eller flere «tar over» samtidig. Måten dette ble prøvd løst på var bruke variabler fra selve pakken som ankom, «viljen» til den enkelte AIMF-ruteren, og data fra en Random Number

Generator (RNG). Ønsket effekt av dette var at den AIMF-ruteren som hadde høyest «vilje» blant de som skulle overta videresendingsansvaret ville ta over «først» og at de andre videresendingkandidatene ville motta trafikk før de selv prøvde å ta videresendingsansvaret. Dette fungerte godt ved gitte multikastpakkerater (Kb/sek), men ved andre rater fikk man ikke ventet utfall. Det kunne oppstå hysteresetilfeller over uakseptable tidsperioder ved linkbrudd og/eller partisjonering. Løsningen var rett og slett ikke deterministisk og løsningen ble dermed kastet. Det skal sies at det ble brukt mye tid på dette ettersom vi trodde det kunne gi god funksjonalitet.

Eksempel vist under (Denne funksjonaliteten var spred godt over hele ruteprotokollen, men essensen kommer godt frem her):

```
1  void
2          RoutingProtocol::ReceivingMulticast(Ipv4Address group) {
3              Time * t = m_state.FindTimer(group);
4              if (t == NULL) {
5                  Time k = Simulator::Now() + IS_RECEVING_MCAST + Time::↩
                       FromInteger((int) (7 - m_willingness)*2, Time::S);
6                  m_state.AddTimer(group, k);
7                  Simulator::ScheduleWithContext(group.Get() + GetObject<↩
                       Node> ()->GetId(), DELAY(k), &aimf::RoutingProtocol::↩
                       ReceivingMulticast, this, group);
8                  t = m_state.FindTimer(group);
9              }
10             if (*t < Simulator::Now()) {
11                 NS_LOG_DEBUG(Simulator::Now().GetSeconds() << " " << group↩
                       << " is not spotted. ");
12                 NS_LOG_DEBUG("Old time: " << t->GetSeconds() << ".");
13                 *t = *t + IS_RECEVING_MCAST + Seconds(m_willingness);
14                 NS_LOG_DEBUG("New time: " << t->GetSeconds() << ".");
15             } else if (*t > Simulator::Now()) {
16                 NS_LOG_DEBUG(Simulator::Now().GetSeconds() << " " << group↩
                       << " is spotted. ");
17                 NS_LOG_DEBUG("Old time: " << t->GetSeconds() << ".");
18                 *t = Simulator::Now() + IS_RECEVING_MCAST + Time::↩
                       FromInteger((int) (AIMF_WILL_ALWAYS - m_willingness), ↩
                       Time::S);
19                 NS_LOG_DEBUG("New time: " << t->GetSeconds() << ".");
20             }
21         }
```

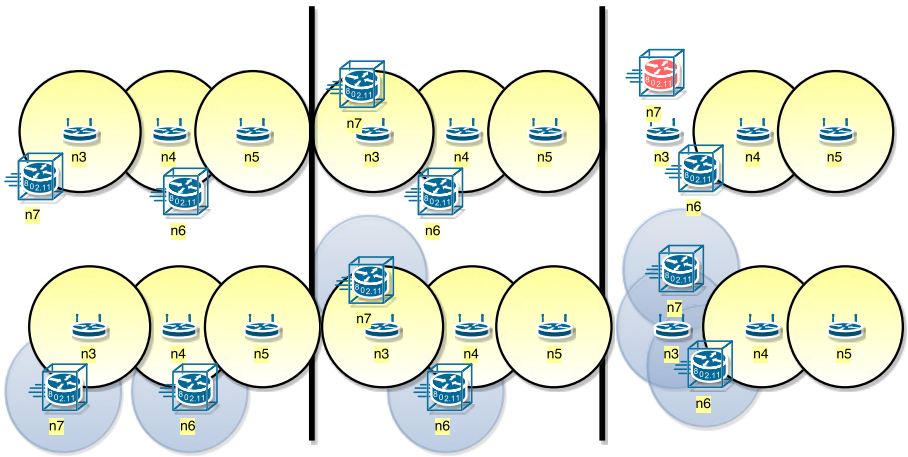
5.4.2 Sending og mottak av hallomeldinger

Kunne man heller brukt unikast-, eller kringkastings-metoden til å formidle disse hallomeldingene? Ved unikast ville hver AIMF-GW ha generert en egen AIMF-melding per AIMF-nabo - noe som skaper unødvendig trafikk sett opp mot multikast - i tillegg måtte man ha hatt en tilstand over alle naboadressene før systemet startet - noe som igjen er tungvint. Videre kan man si at ovennevnte fordrer at det er en støtte for multikasting i det kablede nettverket, men her er det laget en multikastgateway-løsning som er designet for å kunne ligge flere forskjellige IP-nett, så derfor tar vi dette for gitt. En løsning hvor man kringkastet hallomeldingene vil ikke ha gitt den funksjonaliteten som AIMF gir her ettersom man ikke kunne ha kommunisert over flere IP-nett. Ved å bruke multikast kunne man også dra nytte av prinsippet rundt det at all multikastet trafikk blir videresendt i det samme delte treet.

5.5 Resultater ved testing

Det ble vurdert hvordan man skulle vise at løsningen faktisk fungerte. Kunne man for eksempel vise at standardavviket gjeldene videresendingstilstand (Tid i av- og

påtilstand) ble mindre og mindre etter større og større kjøringer? La oss si at man har tre hovedkjøringer med 1000, 750 og 500 kjøringer, ulikt seed per kjøring med tanke på mobilitet og $n6$ og $n7$ lever uavhengig av hverandre i samme boks. Da vil et standardavvik per AIMF-GW over bli mindre og mindre ettersom antall kjøringer øker og ettersom $n3$, $n4$ og $n5$ over tid vil få en lite flyktig videresendingtilstand (mer stabil, repeterende og forutsigbar atferd) som samvarierer sterkt med viljen $n3$, $n4$ og $n5$. Man kan kanskje forvente at n4 vil videresende litt mindre eller mer enn n3 og n5, fordi den er i «midten» - og derfor har en større mulighet til å bli sluttet opp mot enn annen node , men dette har ikke noe med partisjonering å gjøre. Hvis man antar at kjøringen er såpass stor at node $n6$ og $n7$ har vært i alle tilgjengelige posisjoner «flere» ganger (altså en stor n og dermed ett relativt stort konfidensintervall). Så tror vi at standardavviket vil gi en pekepinn på hvilken vilje en AIMF-GW har og ikke si noe om AIMF-løsningen løser partisjonering på en god måte eller økning av MANET-«throughput».



Figur 5.1: Noder uten SMF og «bar» AIMF er illustrert øverst. AIMF *med* SMF støtte og noder *med* SMF er illustrert nederst.

Når det gjaldt å se om det samlet sett alltid er en enkelt multikaststrøm (to i denne konteksten) - i tilfeller uten partisjonering - som entrer MANETet så kan man vise at dette "alltid" skjer ved å eksempelvis lage en $x - y - z$ graf hvor $x = bits$, $y = tid(sek)$ og $x = n(3 - 5).multikastgruppe(1 - 2)$, men at kjøringen er relativt stor (eks. 100000 flyt for $n6$ og $n7$). Ettersom det ikke er noe DPD funksjonalitet i dette systemet må man ved flere tilfeller forvente at $n6$s eller $n7$s multikastmottak opphører (se figur 5.1): Dette kan skje når $n6$, eller $n7$, kun ligger innen for dekning til $n3$ eller $n5$ - som er begge «kantrutere» i denne simuleringen . Hvis da $n6$, eller $n7$ kommer fra «sentrum» og slutter sammen $n3$ og $n4$ eller $n5$ og $n4$ vil den noden som kun lå innenfor dekning til $n3$ eller $n5$ miste multikast mottaket sitt. Her antar vi at $n4$ har større vilje enn $n3$ eller $n5$. Slike tilfeller er unngått i testene gitt i denne oppgaven.

Ved partisjonering kan man også gjøre det samme, men her forklarer man endring av videresendingstilstand per AIMF-GW - med bruk av posisjoner til n6 og n7 i tillegg. Da kan man se eventuell overlapp og opphør i videresending og forklare disse på en god måte. Videre mener vi det er interessant å se på samvariasjonen mellom videresending per AIMF-GW og mobiliteten til n6 og n7 (se test 3, eller test 4.) ved partisjoneringstilfeller.

I test 4 er AIMF testet sammen med SMF og man ser ett godt mottak hos $n6$ og $n7$ og man unngår i tilfeller[2] denne «dødsone»-problematikken som vist i figur 5.1.

[2]Når SMF kjører sammen med AIMF på GWene så vil ikke en «SMF-AIMFgw» videresende pakker som allerede er sett i dert kablede nettet. Dette er noe som skjer hos GW som ikke har høyest vilje.

Kapittel 6
Diskusjon og Konklusjon

Avsluttende kapittel inneholder diskusjon og konklusjon av arbeidet.

6.1 Diskusjon

Spørsmålet er om det er lagt frem en løsning som eksempelvis hever robustheten? For å få svar på dette kan vi prøve å adressere noen av de samme spørsmålene som ble diskutert i det allerede nevnte forprosjektet (disse er gitt på engelsk) [24]. Samt videre supplere disse med de kravene gitt i seksjon 4.1.1 (uthevet i svart):

- Does the proposed solution solve partition merges and occurrences?
  - **Opptrer AIMF sparsomt? Vil det alltid være ett minimumsett av GWer som videresender multikastet data?**
  - **Gir AIMF mer robusthet grunnet bruk av flere GWer?**
  - **Vil AIMF «tåle» en flyktig topologi?**
- Does the proposed solution scale with respect to time and traffic volume. For instance, will the given solution deplete the multicast routing protocol in the wired domain?
- Is there a concern about the complexity of the solution. Difficult to implement and manage?
- How well does the proposed solution handle convergence due to link failures?

Og til slutt stiller vi dette spørsmålet som er det siste gitt under krav i seksjon 4.1.1:

Utvider AIMF ett kablet multikastdomene til å også innlemme ett MANET? Gir den multikasttjenester innenfor nevnte kriterier?

6.1.1 Partisjonering

Hvordan løser AIMF partisjonering og linkbrudd? Ut fra test 3 ser man at AIMF løser partisjonering samtidig som den opererer sparsomt og i test 4 blir mottaket bedre enn i test3 grunnet bruk av SMF. Man ser at AIMF-GWer aldri videresender multikasttrafikk til ett MANET hvis det er en større vilje hos andre OLSR-tilgjengelige AIMF-GWer i samme AIMF-protokolldomene. Videre ser man også i test 2 at AIMF også tåler at en AIMF-GW stopper på grunn av fraværende hallomeldinger og funksjonalitet i hver AIMG-GW. Og til slutt ser man ut fra test 1 at man kan styre hvilke AIMF-GWer som videresender. En annen fordel med AIMF - som konsekvens av jaget etter ett minimalt sett med GW som videresender - er dens evne til å redusere dupliserte multikastpakketransmisjoner mellom AIMF-GWer og 1-hops MANETnoder. Videre antar vi også - ved bruk at RP-baserte multikastruteprotokoller - at det bestandig vil være kun en GW som videresender multikastet trafikk når det ikke forkommer partisjonering. Denne løsningen vil fungere bedre enn «singel-GW»-løsninger med tanke på partisjonering.

6.1.2 Skalerbarhet

Når det gjelder metning og skalerbarhet så er trolig selve MANET teknologien begrensende faktor. AIMF redigerer, eller endrer, *ikke* multikastpakkene som entrer et MANET. Så egentlig kan man si at kilden/ene er problemkilden ettersom de fort kan skape metning. AIMF sender hallomeldinger som igjen kan - i ekstreme situasjoner - skape lastproblematikk for en PIM RP. Anta at man har 6 stykk GWer (a, b, c, d, e, f og g) i ett AIMF-system A og hver av dem sender en hallomelding hvert 2. sekund uavhengig av hverandre. Kun GW a har konfigurert to multikastgrupper. En hallomelding med to gruppepar (k, g), eller $(*, g)$, tar $664bits$ og en hallomelding uten gruppepar tar $512bits$ på datalinklaget. Så over en periode på 2min (120sek) så vil en RP videresende $(644bits \times \frac{120sek}{2sek}) + (5 \times 512bits \times \frac{120sek}{2sek}) = 192240bits \approx 192.24kb/2min \approx 1.602kb/sek$ -noe som er *svært* lite. Hvis vi sier at 5mbit/sek er mye - noe det absolutt er - så måtte det ha vært $\frac{5mbit/sek}{1.602 \times 10^{-3} mbit/sek} \approx 3121$ slike systemer lik A. Så man kan si at for AIMFs del er ikke dette noe problem.

Når det kommer til den viktige multikast trafikken -altså den sendt fra $n1$ (se figur 4.2) - er det inngått noen kompromisser. Som nevnte tidligere er det anvist (jfr. seksjon 3.2.2 og under «Sjekk av OLSRs rutetabell») at man bruker BIDIR-PIM til å rute AIMF signalering, men det er videre sagt at en annen RP-basert ruteprotokoll (PIM-X fra nå av) kan brukes til å rute $n1$s multikasttrafikk. Videre er det også sagt at PIM-X's og BIDIR-PIMs RP må være samlokalisert eller at begge av disse to RPen må være en del av det delte treet til både BIDIR-PIM og PIM-X. RP-baserte multikastruteprotokoller yter vanligvis ikke like godt som eksempelvis PIM-SSM, MOSPF [53] og PIM-DM hvor RP ikke er brukt. PIM-SM vil ei heller kunne alternere gitte multikaststrømmer til ett SPT i denne konteksten. Ett alternativ til det å låse

PIM-X til å være RP-basert er å la kabel-frakoblede GWers AIMF-signaleringen gå i MANETet for å deretter kunne melde om eventuelt[1] opphør av videresendingstilstand. Her må man da overveie om det er ønskelig å sende AIMF-signalering i MANETet. Ett annet alternativ, og kanskje best av de to løsningene gitt her hvis vi hadde fått det til å fungere godt, er at hver GW lytter på multilasttrafikk i MANET som forklart under seksjon refsec:hell. Slik vil da eksempelvis GW x utlede når en GW y - med størst vilje - slutter å videresende og dermed ta[2] over for y likegyldig til om den har rute til y eller ikke ved hjelp av OLSR.

6.1.3 Kompleksitet

Vi vil ikke si at AIMF er komplekst i forhold til andre lastbalanseringsfunksjoner - med tanke på videreseningstilstand og formidling av denne - og/eller andre multikastruteprotokoller med tanke på ruting. AIMF har dog noen initielle verdier som må settes ved oppstart av systemet (eksempelvis valg av «RPF-grensesnittet», eller «beste» grensenitt mot RP, som er vitalt for AIMFs ruting og en eventuelt sameksisterende PIM-løsnng), men vi kan fortsatt ikke hevde at dette øker kompleksiteten i vesentlig grad.

Når det gjelder anvisningen til å bruke RP-baserte multilastruteprotokoller så kan man si at dette hører til under kompleksitet. Hvis man sier at dette hadde vært ett fritt valg kunne man fort ha fått flere multikaststrømmer inn i MANETet, noe som ikke er ønskelig. Det er ingen måte man kan si at en multikaststrøm faktisk er opphørt under feilfri kjøring uten ha noen form for «soft state» eller lignende slik at man slutter å videresende etter ett vist opphør av multikasttrafikk, så derfor er det anvist at man kan bruke det delte treet til å i det minste oppdage linkbrudd. Hvis man eksempelvis brukte PIM-SSM ville ikke AIMF-GWer vite om andre AIMF-GW-naboer mottok multikast eller ikke. I verste fall ville en AIMF-GW med høyest vilje i ett AIMF-system - gitt bruk av PIM-SSM - hindre andre AIMF-GWer i å videresende mottatt multikasttrafikk, uansett om den selv ikke mottok multikasttrafikk.

6.1.4 Konvergens

AIMF håndterer linkbrudd både i MANET og i kablede nettverk, men kovergenstiden varierer. AIMFs maksimale og verst tenkelige konvergeringstid, eller rekonvergeringstid, gjelder linkbrudd i ett MANET. AIMF sjekker GW-«OLSR-tilgjengelighet» ved gitte intervaller slik at AIMF *kan* erverve divergerende tilstand gjeldene partisjonering og/eller linkbrudd i visse tidsperioder. Eksempelvis kan en AIMF-GWs OLSR-instans oscillere mellom å ha en rute mot en AIMF-GW med høyere vilje samtidig som denne OLSRsjekken hos en laverestående AIMF-GW alltid tar en sjekk når OLSR ikke har

[1] Denne noden trenger ikke å ha størst vilje i AIMFsystemet fra før av.

[2] GW x kan kun ta over for y hvis x innehar «nest» størst vilje i det gitte AIMFsystemet

rute - da i en viss periode. Det vil si at hvis tilstanden av partisjonering er meget flyktig kan man få en relativt langvarig ettervirkning, men den vil etterhvert opphøre og kan betraktes som deterministisk. OLSR kunne i stedet ha varslet AIMF om endringer(eks. «callback»), men det er ikke valgt å gjøre det slik på grunn av at vi har ønsket å holde eksisterende «ns3»-protokoller urørt. Disse tidene som bil oppgitt i tabell 4.1 kan også endres og dermed tilpasses gitte miljøer. Da kan tilpasse AIMF for forskjellige scenarioer, la seg være militære eller sivile.

Ved bortfall av GWer i det kablede nettet - ved bruk av «join»-baserte multikastruteprotokoller - er kovergenstiden noe lengre en AIMFs naboholdtid + AIMFs OLSRsjekkintervall-tid ettersom multikastruteprotkollen i det kablede nettet må varsles(Join) *etter* at en nabo, eller ett naboforhold, er slettet[3]. Videre blir konsekvensen av at en nabo blir slettet først effektuert når en enkelt GW fortar en OLSRsjekk. Grunnen til dette er at selve funksjonalitet for å få endret multikastvideresendingstilstanden ligger i denne OLSRsjekken.

Man kan videre sette dette opp mot en løsning hvor man har flere GWer, men hvor man ikke styrer videresendingstilstand. Her ville man ikke ha snakket om disse tidsintervallene ei heller konvergens, noe som er bra. Derimot ville man sett mye duplikater inne i MANETet.

Disse tidene som bil oppgitt i tabell 4.1 kan også endres og dermed tilpasses gitte miljøer. Da kan man tilpasse AIMF for forskjellige scenarioer, la seg være militære eller sivile.

6.1.5 Kvalitet

Kvaliteten på multikastvideresendingen er kun påvirket av MANETets kapasitet. Følgelig vil det være best mottak hos AIMF-GWnære MANETnoder og man vil se en forverring etter hvert hopp utover i MANET. Videre kan man forvente at ved å alltid ha ett minimalt sett av AIMF-GWer som videresender at man ser minimalt med duplikater. Videre kan man jo si at høynet robusthet - med tanke på partisjonering - også høyner kvaliteten på multikastmottak. SMF høyner også mottaket betraktelig og i kooperasjon med AIMF så vil man oppnå en større kumulativ båndbredde besparelse ettersom man slipper duplikater i «ingress»-områdene i MANETet.

6.1.6 Sparsomhet

Sparsomhet har også vært en viktig faktor under design og arkitektur. AIMF-GWer som «ser» en tom OLSR-rutetabell vil ikke videresende multikasttrafikk (OLSRs hallomeldinger går som vanlig). Det er en god ting med tanke på lage minst mulig

[3] soft state

«avtrykk» i det elektromagnetiske spekteret. Man kunne ha hevdet at ved tilfeller hvor MNer vekslet mellom å være innenfor dekning og ikke, ville denne løsningen ha hvert lite hensiktsmessig ettersom AIMF ville ha oscillert, eller stoppet og startet, sin videresending. I tillegg - som allerede forklart- er det alltid ett minst mulig sett med AIMF-GWer som videresender til en hver tid.

6.2 Konklusjon

Vi har igjennom denne oppgaven designet, implementert og testet en interdomene/interprotokoll multilastruteprotokoll kalt AIMF. Og vi har implementert SMF som gitt i [49]. Førstnevnte protokoll gir en robust og sparsom «multikastgateway» -løsning for multikast trafikk som traverserer fra ett kablet IP-nett og inn i ett MANET. Funksjonalitet har blitt vist igjennom å bruke ns3-rammeverket og dets innebygde funksjonalitet. Ingen funksjonalitet i ns3 er skrevet om og følger derfor gitte standarder. Videre er det vist at for MANET så reduserer AIMF dupliserte pakketransmisjoner mellom GWer og 1-hops MANETnoder betraktelig ettersom det alltid er minimalt sett av AIMG-GW som videresender multikasttrafikk til en hver tid. Videre er det vist at AIMF kan styres av en viljevariabel som igjen bestemmer hvem som skal videresende til en hver tid. Denne variabelen kan settes både før kjøring og i sanntid. Det er også vist at SMF fungerer godt sammen med AIMF ettersom MANETnoder generelt kan videresende multitaksttrafikk til MANETnoder som ikke er innfor dekning til ett AIMFsystem

AIMF håndterer partisjonering i MANET og linkbrudd i det kablede nettet på en god måte ved å bruke rutetilstanden til OLSR til å se hvilke AIMF-GW som er tilgjengelige. Og AIMF kan bruke RP-baserte ruteprotokollers atferd samt AIMFs egne hallomeldinger til løse linkbruddproblematikk. Vi tror at dette vil øke kvaliteten og robustheten med tanke på multikastmottak i MANET.

6.2.1 Videre arbeider

Som observert er denne protokollen bare testet for sin funksjonalitet så her gir vi noen interessante gjøremål for videre arbeid.

- Kjøre ytelsestester med SMF inkorporert i AIMF-GWer og MANETnoder.
- Legge til mer funksjonalitet med tanke på Quality of Service (QOS). Eksempelvis gi forskjellige Differentiated services (DiffServ) [6] og/eller Resource Reservation Protocol (RVSP) [10] metoder til å utlede vilje per multilastgruppe I AIMF.
- Implementere AIMF ved hjelp av Software Defined Networking (SDN) [44].

Litteratur

[1] Ieee standards for local area networks: Carrier sense multiple access with collision detection (csma/cd) access method and physical layer specifications. *ANSI/IEEE Std 802.3-1985*, pages 1–30, 1985.

[2] Multicast routing in a datagram internetwork. *Department of Computer Science Stanford University Stanford, California 94305*, pages 1–156, December 1991.

[3] Ieee standards for local and metropolitan area networks: Supplements to carrier sense multiple access with collision detection (csma/cd) access method and physical layer specifications - specification for 802.3 full duplex operation and physical layer specification for 100 mb/s operation on two pairs of category 3 or better balanced twisted pair cable (100base-t2). *IEEE Std 802.3x-1997 and IEEE Std 802.3y-1997 (Supplement to ISO/IEC 8802-3: 1996; ANSI/IEEE Std 802.3, 1996 Edition)*, pages 1–324, 1997.

[4] Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pages 1–2793, March 2012.

[5] A. Adams, J. Nicholas, and W. Siadak. Protocol independent multicast - dense mode (pim-dm): Protocol specification (revised). RFC 3973, RFC Editor, January 2005.

[6] J. Babiarz, K. Chan, and F. Baker. Configuration guidelines for diffserv service classes. RFC 4594, RFC Editor, August 2006.

[7] T Ballardie, P Francis, and J Crowcroit. Core based trees: An architecture ior scalable inter-domain multicast routing. *Proc. oiACM SIGCOMM*, 93, 1993.

[8] T. Bates, R. Chandra, D. Katz, and Y. Rekhter. Multiprotocol extensions for bgp-4. RFC 4760, RFC Editor, January 2007. http://www.rfc-editor.org/rfc/rfc4760.txt.

[9] B. Bellur and R. G. Ogier. A reliable, efficient topology broadcast protocol for dynamic networks. In *INFOCOM '99. Eighteenth Annual Joint Conference of the*

IEEE Computer and Communications Societies. Proceedings. IEEE, volume 1, pages 178–186 vol.1, Mar 1999.

[10] Bob Braden, Lixia Zhang, Steve Berson, Shai Herzog, and Sugih Jamin. Resource reservation protocol (rsvp) – version 1 functional specification. RFC 2205, RFC Editor, September 1997. http://www.rfc-editor.org/rfc/rfc2205.txt.

[11] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet group management protocol, version 3. RFC 3376, RFC Editor, October 2002. http://www.rfc-editor.org/rfc/rfc3376.txt.

[12] I. D. Chakeres, C. Danilov, T. R. Henderson, and J. P. Macker. Connecting manet multicast. In *MILCOM 2007 - IEEE Military Communications Conference*, pages 1–7, Oct 2007.

[13] M. Christensen, K. Kimball, and F. Solensky. Considerations for internet group management protocol (igmp) and multicast listener discovery (mld) snooping switches. RFC 4541, RFC Editor, May 2006.

[14] CA 95134-1706 USA Cisco Systems, inc. 170 West Tasman Drive San Jose. Multi-vrf and ip multicast. http://www.cisco.com/en/US/technologies/tk648/tk828/tk363/technologies_white_paper0900aecd8012033f.pdf. Lastet: 2016-05-03.

[15] T. Clausen and P. Jacquet. Optimized link state routing protocol (olsr). RFC 3626, RFC Editor, October 2003. http://www.rfc-editor.org/rfc/rfc3626.txt.

[16] S. Corson and J. Macker. Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations. RFC 2501, RFC Editor, January 1999.

[17] M. Cotton, L. Vegoda, and D. Meyer. Iana guidelines for ipv4 multicast address assignments. BCP 51, RFC Editor, March 2010.

[18] 2000-2016 cplusplus.com. Polymorphism, c++. http://www.cplusplus.com/doc/tutorial/polymorphism/. Lastet: 2016-05-24.

[19] Yogen K. Dalal and Robert M. Metcalfe. Reverse path forwarding of broadcast packets. *Commun. ACM*, 21(12):1040–1048, December 1978.

[20] C. Danilov, T. R. Henderson, P. A. Spagnolo, T. Goff, and J. H. Kim. Manet multicast with multiple gateways. In *MILCOM 2008 - 2008 IEEE Military Communications Conference*, pages 1–8, Nov 2008.

[21] Debian. About Debian, debian os. https://www.debian.org/intro/about. Lastet: 2016-05-27.

[22] S. Deering, W. Fenner, and B. Haberman. Multicast listener discovery (mld) for ipv6. RFC 2710, RFC Editor, October 1999.

[23] Steve Deering. Host extensions for ip multicasting. STD 5, RFC Editor, August 1989. http://www.rfc-editor.org/rfc/rfc1112.txt.

[24] Simon Dehli. Interdomain multicast. dec 2015.

[25] Valgrind Developers. About Valgrind, valgrind. http://valgrind.org/info/about.html. Lastet: 2016-05-27.

[26] A Duda. Understanding the performance of 802.11 networks. In *2008 IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications*.

[27] D. Farinacci and Y. Cai. Anycast-rp using protocol independent multicast (pim). RFC 4610, RFC Editor, August 2006. http://www.rfc-editor.org/rfc/rfc4610.txt.

[28] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas. Protocol independent multicast - sparse mode (pim-sm): Protocol specification (revised). RFC 4601, RFC Editor, August 2006.

[29] B. Fenner, H. He, B. Haberman, and H. Sandick. Internet group management protocol (igmp) / multicast listener discovery (mld)-based multicast forwarding ("igmp/mld proxying"). RFC 4605, RFC Editor, August 2006. http://www.rfc-editor.org/rfc/rfc4605.txt.

[30] B. Fenner and D. Meyer. Multicast source discovery protocol (msdp). RFC 3618, RFC Editor, October 2003.

[31] V. Fuller and T. Li. Classless inter-domain routing (cidr): The internet address assignment and aggregation plan. BCP 122, RFC Editor, August 2006. http://www.rfc-editor.org/rfc/rfc4632.txt.

[32] GDB. GNU Debugger, gdb. https://www.gnu.org/software/gdb/. Lastet: 2016-05-27.

[33] M. Handley, I. Kouvelas, T. Speakman, and L. Vicisano. Bidirectional protocol independent multicast (bidir-pim). RFC 5015, RFC Editor, October 2007.

[34] M. Handley, C. Perkins, and E. Whelan. Session announcement protocol. RFC 2974, RFC Editor, October 2000.

[35] R. Hinden and S. Deering. Ip version 6 addressing architecture. RFC 4291, RFC Editor, February 2006. http://www.rfc-editor.org/rfc/rfc4291.txt.

[36] H. Holbrook and B. Cain. Source-specific multicast for ip. RFC 4607, RFC Editor, August 2006.

[37] IANA. IANA, internet assigned numbers authority. https://www.iana.org. Lastet: 2016-05-04.

[38] ISO/IEC. ISO/IEC, working draft, standard for programming language c++. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3242.pdf. Lastet: 2016-05-26.

[39] Philippe Jacquet, Pascale Minet, Anis Laouiti, Laurent Viennot, Thomas Clausen, and Cedric Adjih. Multicast optimized link state routing. Internet-Draft draft-jacquet-olsr-molsr-00, IETF Secretariat, IETF MANET Working Group, April 2002. http://www.ietf.org/internet-drafts/draft-jacquet-olsr-molsr-00.txt.

[40] K. Jamshaid, B. Shihada, L. Xia, and P. Levis. Buffer sizing in 802.11 wireless mesh networks. In *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, pages 272–281, Oct 2011.

[41] B. Joshi, A. Kessler, and D. McWalter. Pim group-to-rendezvous-point mapping. RFC 6226, RFC Editor, May 2011.

[42] R.E. Kahn. The organization of computer resources into a packet radio network. *Communications, IEEE Transactions on*, 25(1):169–178, Jan 1977.

[43] Berman L Kou L, Markowsky G. A fast algorithm for steiner trees. pages 15:141–145, 1981.

[44] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

[45] James F Kurose. *Computer Networking: A Top-Down Approach Featuring the Internet, 3/E*. Pearson Education India, 2005.

[46] Y. Lacharit, M. Wang, L. Lamont, and L. Landmark. A simplified approach to multicast forwarding gateways in manet. In *2007 4th International Symposium on Wireless Communication Systems*, pages 426–430, Oct 2007.

[47] L. Landmark, Y. Lacharite, and L. Lamont. Multicast forwarding using multiple gateways and hash for duplicate packet detection in a tactical manet. In *MILCOM 2007 - IEEE Military Communications Conference*, pages 1–7, Oct 2007.

[48] Major Michael R. Macedonia and Lieutenant Commander Donald P. Brutzman. MBONE, the multicast backbone. http://www-mice.cs.ucl.ac.uk/multimedia/projects/mice/mbone_review.html. Lastet: 2016-05-03.

[49] J. Macker. Simplified multicast forwarding. RFC 6621, RFC Editor, May 2012.

[50] Gary Scott Malkin. Rip version 2. STD 56, RFC Editor, November 1998. http://www.rfc-editor.org/rfc/rfc2453.txt.

[51] David Miller. Git, david miller's -next networking tree. http://git.kernel.org/cgit/linux/kernel/git/davem/net-next.git/commit/?id=89aef8921bfbac22f00e04f8450f6e447db13e42. Lastet: 2016-05-23.

[52] Jeffrey Mogul. Broadcasting internet datagrams. STD 5, RFC Editor, October 1984. http://www.rfc-editor.org/rfc/rfc919.txt.

[53] J. Moy. Multicast extensions to ospf. RFC 1584, RFC Editor, March 1994.

[54] NetBeans. NetBeans, c/c++. https://netbeans.org/features/cpp/index.html. Lastet: 2016-05-27.

[55] NSNAM. NetAnimator, ns3. https://www.nsnam.org/wiki/NetAnim. Lastet: 2015-05-16.

[56] NSNAM. NS-3, network simulator 3. https://www.nsnam.org/overview/what-is-ns-3/. Lastet: 2016-05-03.

[57] NSNAM. OLSR, network simulator 3. https://www.nsnam.org/doxygen/group__olsr.html. Lastet: 2016-05-24.

[58] NSNAM. Propagation Loss and Delay, ns3. https://www.nsnam.org/docs/models/html/propagation.html. Lastet: 2015-05-24.

[59] NSNAM. Propagation Models, ns3. https://www.nsnam.org/doxygen/group__propagation.html. Lastet: 2015-05-24.

[60] NSNAM. Routing architecture, ns3. https://www.nsnam.org/docs/release/3.10/manual/html/_images/routing.png. Lastet: 2015-05-16.

[61] NSNAM. Routing, network simulator 3. https://www.nsnam.org/docs/release/3.10/manual/html/routing.html#. Lastet: 2016-05-27.

[62] NSNAM. Routing per protocol, ns3. https://www.nsnam.org/docs/release/3.10/manual/html/_images/routing-specialization.png. Lastet: 2015-05-16.

[63] NSNAM. Tracing, network simulator 3. https://www.nsnam.org/docs/manual/html/tracing.html#. Lastet: 2016-05-27.

[64] R. Ogier, F. Templin, and M. Lewis. Topology dissemination based on reverse-path forwarding (tbrpf). RFC 3684, RFC Editor, February 2004.

[65] Carlos AS Oliveira and Panos M Pardalos. Steiner trees and multicast. In *Mathematical Aspects of Network Routing Optimization*, pages 29–45. Springer, 2011.

[66] D. Oran. Osi is-is intra-domain routing protocol. RFC 1142, RFC Editor, February 1990.

[67] M. Patrick. Dhcp relay agent information option. RFC 3046, RFC Editor, January 2001.

[68] Nathan Penner, Shaowei Lin, and Carly Geehr. Architecture, abilene and vbns. http://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/internet-2/architecture.html. Lastet: 2016-05-03.

[69] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing. RFC 3561, RFC Editor, July 2003. http://www.rfc-editor.org/rfc/rfc3561.txt.

[70] J. Postel. User datagram protocol. STD 6, RFC Editor, August 1980. http://www.rfc-editor.org/rfc/rfc768.txt.

[71] J. Postel. Internet control message protocol. STD 5, RFC Editor, September 1981. http://www.rfc-editor.org/rfc/rfc792.txt.

[72] Jon Postel. Internet protocol. STD 5, RFC Editor, September 1981. http://www.rfc-editor.org/rfc/rfc791.txt.

[73] P. Radoslavov, D. Estrin, R. Govindan, M. Handley, S. Kumar, and D. Thaler. The multicast address-set claim (masc) protocol. RFC 2909, RFC Editor, September 2000.

[74] Trygve Mikjel H Reenskaug. The original mvc reports. 1979.

[75] Y. Rekhter, T. Li, and S. Hares. A border gateway protocol 4 (bgp-4). RFC 4271, RFC Editor, January 2006. http://www.rfc-editor.org/rfc/rfc4271.txt.

[76] J.B. Orlin R.K. Ahuja, T.L. Magnanti. Network flows: theory, algorithms and applications. 1993.

[77] Gabriel Robins and Alexander Zelikovsky. Minimum steiner tree construction. *The Handbook of Algorithms for VLSI Phys. Design Automation*, pages 487–508, 2009.

[78] E. Rosen and R. Aggarwal. Multicast in mpls/bgp ip vpns. RFC 6513, RFC Editor, February 2012.

[79] E. Rosen, Y. Cai, and IJ. Wijnands. Cisco systems' solution for multicast in bgp/mpls ip vpns. RFC 6037, RFC Editor, October 2010.

[80] E. Rosen, IJ. Wijnands, Y. Cai, and A. Boers. Multicast virtual private network (mvpn): Using bidirectional p-tunnels. RFC 7582, RFC Editor, July 2015.

[81] Eric Rosenberg. *A primer of multicast routing.* Springer Science & Business Media, 2012.

[82] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. Sip: Session initiation protocol. RFC 3261, RFC Editor, June 2002. http://www.rfc-editor.org/rfc/rfc3261.txt.

[83] Donnie Savage, James Ng, Steven Moore, Donald Slice, Peter Paluch, and Russ White. Cisco enhanced interior gateway routing protocol. Internet-Draft draft-savage-eigrp-05, IETF Secretariat, February 2016. http://www.ietf.org/internet-drafts/draft-savage-eigrp-05.txt.

[84] Cisco System. PIM Multicast Routing,session 2215. http://www.cisco.com/networkers/nw00/pres/2215_6-28.pdf. Lastet: 2016-05-24.

[85] Cisco Systems. Cisco Systems, ip multicast: Pim configuration guide. multicast group modes. http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipmulti_pim/configuration/12-4t/imc-pim-12-4t-book/imc_basic_cfg.html. Lastet: 2016-05-18.

[86] D. Thaler. Border gateway multicast protocol (bgmp): Protocol specification. RFC 3913, RFC Editor, September 2004.

[87] Inc. Thinking Pictures. MBONE, rolling stones mbone broadcast nov 18, 1994. https://vimeo.com/116409303. Lastet: 2016-05-03.

[88] Xiaohua Tian and Yu Cheng. *Scalable Multicasting over Next-Generation Internet: Design, Analysis and Applications*. Springer Publishing Company, Incorporated, 2012.

[89] David Wayne Wall. *Mechanisms for Broadcast and Selective Broadcast*. PhD thesis, Stanford, CA, USA, 1980. AAI8024752.

[90] Wikipedia. Intelligent code completion, wikipedia. https://en.wikipedia.org/wiki/Intelligent_code_completion. Lastet: 2016-05-27.

[91] Beau Williamson. *Developing IP Multicast Networks*. Cisco Press, 1999.

[92] D. Wing and T. Eckert. Ip multicast requirements for a network address translator (nat) and a network address port translator (napt). BCP 135, RFC Editor, February 2008.

[93] WireShark. Wireshark. https://www.wireshark.org/. Lastet: 2016-05-28.

[94] Yunjung Yi, Sung-Ju Lee, William Su, and Mario Gerla. On-demand multicast routing protocol (odmrp) for ad hoc networks. Internet-Draft draft-ietf-manet-odmrp-04, IETF Secretariat, IETF MANET Working Group, Feb 2003. http://www.ietf.org/internet-drafts/draft-ietf-manet-odmrp-04.txt.

Kapittel A

AIMF. Utvalgt kildekode og linker til eksternt kodelager.

A.1 Kontrollskript

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

// Network topology
//
//       LAN2                    Lan1/WLAN
//       ===                  ==========
//        |                    |    |    |    |    |
//       n0    n1    n2    n3    n4   n5   n6   n7   n8   n9
//        |     |     |     |     |    |              |    |
//        ============================  ================
//             Lan0                      LAN2
// Do your choosing between 802.11 and 802.3 with the wifiok (bool) value.)
// - Multicast source is at node n1;
// - Multicast forwarded by node n3, n4 or n5 onto LAN1/WLAN1;
// - Bidirectional multicast forwarding at node n0 in regard to AIMF datagram forwarding;


#include <iostream>
#include <fstream>

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/csma-module.h"
```

```
38 #include "ns3/wifi-module.h"
39 #include "ns3/applications-module.h"
40 #include "ns3/internet-module.h"
41 #include "ns3/aimf-helper.h"
42 #include "ns3/aimf-routing-protocol.h"
43 #include "ns3/olsr-helper.h"
44 #include "ns3/olsr-routing-protocol.h"
45 #include "ns3/mobility-module.h"
46 #include "ns3/stats-module.h"
47 #include "ns3/netanim-module.h"
48
49
50 using namespace ns3;
51
52 NS_LOG_COMPONENT_DEFINE("AimfMulticast");
53
54 void RoutingTableChangeAIMF() {
55
56 }
57
58 int
59 main(int argc, char *argv[]) {
60     //
61     // Users may find it convenient to turn on explicit debugging
62     // for selected modules; the below lines suggest how to do this
63     //
64     bool verbose = false;
65     bool wifiok = true;
66     //     LogComponentEnable("AimfMulticast", LOG_LEVEL_INFO);
67     //     LogComponentEnable("AimfRoutingProtocol", LOG_LEVEL_INFO);
68     //     //LogComponentEnable("OlsrRoutingProtocol", LOG_LEVEL_INFO);
69     //     LogComponentEnable("AimfHeader", LOG_LEVEL_INFO);
70
71     //
72     // Set up default values for the simulation.
73     //
74     // Select DIX/Ethernet II-style encapsulation (no LLC/Snap header)
75     Config::SetDefault("ns3::CsmaNetDevice::EncapsulationMode", ↩
           StringValue("Dix"));
76     //     Config::Set("ns3::aimf::RoutingProtocol/HelloInterval", ↩
           TimeValue(Seconds(4)));
77
78     // Allow the user to override any of the defaults at
79     // run-time, via command-line arguments
80     CommandLine cmd;
81     cmd.Parse(argc, argv);
82
83     NS_LOG_INFO("Create nodes.");
84     NodeContainer c;
85     c.Create(10);
86     // We will later want two subcontainers of these nodes, for the two ↩
           LANs
87     NodeContainer c0 = NodeContainer(c.Get(0), c.Get(1), c.Get(2), c.Get↩
           (3), c.Get(4));
88     NodeContainer c1 = NodeContainer(c.Get(3), c.Get(4), c.Get(5), c.Get↩
           (6), c.Get(7));
89     NodeContainer c2 = NodeContainer(c.Get(0), c.Get(8), c.Get(9), c.Get↩
           (5));
90
91     NS_LOG_INFO("Build Topology.");
92     CsmaHelper csma;
```

```
93      csma.SetChannelAttribute("DataRate", DataRateValue(DataRate(5000000)))↩
            ;
94      csma.SetChannelAttribute("Delay", TimeValue(MilliSeconds(2)));
95      CsmaHelper csma2;
96      csma.SetChannelAttribute("DataRate", DataRateValue(DataRate(5000000)))↩
            ;
97      csma.SetChannelAttribute("Delay", TimeValue(MilliSeconds(2)));
98
99      // We will use these NetDevice containers later, for IP addressing
100     NetDeviceContainer nd0 = csma.Install(c0);
101     NetDeviceContainer nd2 = csma2.Install(c2); // First LAN
102
103
104     WifiHelper wifi;
105     if (verbose) {
106         wifi.EnableLogComponents(); // Turn on all Wifi logging
107     }
108     wifi.SetStandard(WIFI_PHY_STANDARD_80211b);
109     std::string phyMode("OfdmRate54Mbps");
110
111     YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();
112     // This is one parameter that matters when using FixedRssLossModel
113     // set it to zero; otherwise, gain will be added
114     wifiPhy.Set("RxGain", DoubleValue(0));
115     // ns-3 supports RadioTap and Prism tracing extensions for 802.11b
116     wifiPhy.SetPcapDataLinkType(YansWifiPhyHelper::DLT_IEEE802_11_RADIO);
117
118     YansWifiChannelHelper wifiChannel;
119     wifiChannel.SetPropagationDelay("ns3::↩
            ConstantSpeedPropagationDelayModel");
120     // The below FixedRssLossModel will cause the rss to be fixed ↩
            regardless
121     // of the distance between the two stations, and the transmit power
122     double maxRange = 150;
123     wifiChannel.AddPropagationLoss("ns3::RangePropagationLossModel","↩
            MaxRange",DoubleValue(maxRange));
124     wifiPhy.SetChannel(wifiChannel.Create());
125
126     // Add a non-QoS upper mac, and disable rate control
127     NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default();
128     wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager",
129             "DataMode", StringValue(phyMode),
130             "ControlMode", StringValue(phyMode));
131     // Set it to adhoc mode
132     wifiMac.SetType("ns3::AdhocWifiMac");
133     NetDeviceContainer nd1;
134     AimfHelper aimf;
135     Ipv4StaticRoutingHelper staticRouting;
136     Ipv4StaticRoutingHelper staticRouting2;
137     Ipv4ListRoutingHelper list;
138     Ipv4ListRoutingHelper list2;
139     InternetStackHelper internet;
140     InternetStackHelper internet2;
141     InternetStackHelper internet3;
142
143     if (wifiok) {
144         nd1 = wifi.Install(wifiPhy, wifiMac, c1); // WLAN
145         MobilityHelper mobility;
146         mobility.SetPositionAllocator("ns3::RandomDiscPositionAllocator",
147                 "X", StringValue("95.0"),
148                 "Y", StringValue("50.0"),
```

```
149                 "Rho", StringValue("ns3::UniformRandomVariable[Min=0|Max=41]"));
150         mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
151                 "Mode", StringValue("Time"),
152                 "Time", StringValue("0.5s"),
153                 "Speed", StringValue("ns3::UniformRandomVariable[Min=10|Max=100]"),
154                 "Bounds", StringValue("0|190|8|150"));
155         mobility.Install(NodeContainer(c.Get(6), c.Get(7)));
156         MobilityHelper mobility2;
157         Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
158         positionAlloc->Add(Vector(140.0, 8.0, 0.0));//gw1-n3
159         positionAlloc->Add(Vector(160.0, 8.0, 0.0));//gw2-n4
160         positionAlloc->Add(Vector(180.0, 8.0, 0.0));//gw3-n5
161         positionAlloc->Add(Vector(170.0, 2.0, 0.0));//n0
162         positionAlloc->Add(Vector(145.0, 3.0, 0.0));//n1
163         positionAlloc->Add(Vector(155.0, 3.0, 0.0));//n2
164         positionAlloc->Add(Vector(170.0, 2.0, 0.0));//n8
165         positionAlloc->Add(Vector(190.0, 2.0, 0.0));//n9
166         mobility2.SetPositionAllocator(positionAlloc);
167         mobility2.SetMobilityModel("ns3::ConstantPositionMobilityModel");
168         NodeContainer w = NodeContainer(c.Get(3), c.Get(4), c.Get(5));
169         NodeContainer k= NodeContainer(c.Get(0), c.Get(1), c.Get(2), c.Get(8), c.Get(9));
170         w.Add(k);
171         mobility2.Install(w);
172
173
174
175         NS_LOG_INFO("Add IP Stack. Wireless chosen!");
176         OlsrHelper olsr;
177         OlsrHelper olsr2;
178         olsr.ExcludeInterface(c.Get(3), 1);
179         olsr.ExcludeInterface(c.Get(4), 1);
180         olsr.ExcludeInterface(c.Get(5), 1);
181         aimf.ExcludeInterface(c.Get(3), 2);
182         aimf.ExcludeInterface(c.Get(4), 2);
183         aimf.ExcludeInterface(c.Get(5), 2);
184         aimf.SetMANETNetDeviceID(c.Get(3), 2);
185         aimf.SetMANETNetDeviceID(c.Get(4), 2);
186         aimf.SetMANETNetDeviceID(c.Get(5), 2);
187
188         list.Add(staticRouting, 10);
189         list.Add(aimf, 12);
190         list.Add(olsr, 11);
191         list2.Add(staticRouting2, 0);
192         list2.Add(olsr2, 9);
193
194
195         internet.Install(NodeContainer(c.Get(0), c.Get(1), c.Get(2), c.Get(8), c.Get(9)));
196
197
198         internet2.SetRoutingHelper(list);
199         internet2.Install(NodeContainer(c.Get(3), c.Get(4), c.Get(5)));
200         internet3.SetRoutingHelper(list2);
201         internet3.Install(NodeContainer(c.Get(6), c.Get(7)));
202     } else {
203         nd1 = csma.Install(c1); // Second LAN
204         NS_LOG_INFO("Add IP Stack.");
205         aimf.ExcludeInterface(c.Get(3), 2);
```

```
206         aimf.ExcludeInterface(c.Get(4), 2);
207         list.Add(staticRouting, 0);
208         list.Add(aimf, 10);
209         internet.Install(NodeContainer(c.Get(0), c.Get(1), c.Get(2), c.Get↩
                (5), c.Get(6)));
210         internet2.SetRoutingHelper(list);
211         internet2.Install(NodeContainer(c.Get(3), c.Get(4)));
212     }
213     NS_LOG_INFO("Assign IP Addresses.");
214     Ipv4AddressHelper ipv4Addr;
215     ipv4Addr.SetBase("10.1.1.0", "255.255.255.0");
216     ipv4Addr.Assign(nd0);
217     ipv4Addr.SetBase("10.1.3.0", "255.255.255.0");
218     ipv4Addr.Assign(nd2);
219     ipv4Addr.SetBase("10.1.2.0", "255.255.255.0");
220     ipv4Addr.Assign(nd1);
221
222     std::string probeType;
223     std::string tracePath;
224     std::string probeType2;
225     std::string tracePath2;
226     std::string probeType3;
227     std::string tracePath3;
228     std::string probeType4;
229     std::string tracePath4;
230     probeType = "ns3::Ipv4PacketProbe";
231     tracePath = "/NodeList/*/$ns3::aimf::RoutingProtocol/Tx";
232     probeType2 = "ns3::Ipv4PacketProbe";
233     tracePath2 = "/NodeList/*/$ns3::aimf::RoutingProtocol/Rx";
234     probeType3 = "ns3::Ipv4PacketProbe";
235     tracePath3 = "/NodeList/*/$ns3::aimf::RoutingProtocol/McTx";
236     probeType4 = "ns3::Ipv4PacketProbe";
237     tracePath4 = "/NodeList/*/$ns3::aimf::RoutingProtocol/McRx";
238     NS_LOG_INFO("Configure multicasting.");
239
240
241     Ptr<Ipv4> stack = c.Get(3)->GetObject<Ipv4> ();
242     Ptr<Ipv4RoutingProtocol> rp_Gw = (stack->GetRoutingProtocol());
243     Ptr<Ipv4ListRouting> lrp_Gw = DynamicCast<Ipv4ListRouting> (rp_Gw);
244
245     Ptr<Ipv4> stack2 = c.Get(4)->GetObject<Ipv4> ();
246     Ptr<Ipv4RoutingProtocol> rp_Gw2 = (stack2->GetRoutingProtocol());
247     Ptr<Ipv4ListRouting> lrp_Gw2 = DynamicCast<Ipv4ListRouting> (rp_Gw2);
248
249     Ptr<Ipv4> stack3 = c.Get(5)->GetObject<Ipv4> ();
250     Ptr<Ipv4RoutingProtocol> rp_Gw3 = (stack3->GetRoutingProtocol());
251     Ptr<Ipv4ListRouting> lrp_Gw3 = DynamicCast<Ipv4ListRouting> (rp_Gw3);
252
253
254     Ptr<aimf::RoutingProtocol> aimf_Gw;
255     Ptr<aimf::RoutingProtocol> aimf_Gw2;
256     Ptr<aimf::RoutingProtocol> aimf_Gw3;
257
258
259     for (uint32_t i = 0; i < lrp_Gw->GetNRoutingProtocols(); i++) {
260         int16_t priority;
261         Ptr<Ipv4RoutingProtocol> temp = lrp_Gw->GetRoutingProtocol(i, ↩
                priority);
262         if (DynamicCast<aimf::RoutingProtocol> (temp)) {
263             aimf_Gw = DynamicCast<aimf::RoutingProtocol>(temp);
264         }
265
```

```
266     }
267
268     for (uint32_t i = 0; i < lrp_Gw2->GetNRoutingProtocols(); i++) {
269         int16_t priority;
270         Ptr<Ipv4RoutingProtocol> temp = lrp_Gw2->GetRoutingProtocol(i, ↩
                priority);
271         if (DynamicCast<aimf::RoutingProtocol> (temp)) {
272             aimf_Gw2 = DynamicCast<aimf::RoutingProtocol>(temp);
273         }
274
275     }
276
277     for (uint32_t i = 0; i < lrp_Gw3->GetNRoutingProtocols(); i++) {
278         int16_t priority;
279         Ptr<Ipv4RoutingProtocol> temp = lrp_Gw3->GetRoutingProtocol(i, ↩
                priority);
280         if (DynamicCast<aimf::RoutingProtocol> (temp)) {
281             aimf_Gw3 = DynamicCast<aimf::RoutingProtocol>(temp);
282         }
283
284     }
285     Ipv4StaticRoutingHelper multicast;
286
287     Ipv4Address multicastSource("10.1.1.2");
288     Ipv4Address multicastGroup("225.1.2.4");
289     Ipv4Address multicastSource2("10.1.1.2");
290     Ipv4Address multicastGroup2("225.1.2.5");
291     //     aimf_Gw->AddHostNetworkAssociation(multicastGroup, ↩
            multicastSource);
292     //     aimf_Gw->AddHostNetworkAssociation(multicastGroup2, ↩
            multicastSource2);
293
294
295
296     Ptr<Node> sender = c.Get(0);
297     Ptr<NetDevice> senderIf = nd0.Get(0);
298
299     multicast.AddMulticastRoute(c.Get(0), (Ipv4Address("").GetAny()), ↩
            Ipv4Address("225.1.2.4"), senderIf, NetDeviceContainer(nd2.Get(0))↩
            );
300     multicast.AddMulticastRoute(c.Get(0), (Ipv4Address("").GetAny()), ↩
            Ipv4Address("225.1.2.5"), senderIf, NetDeviceContainer(nd2.Get(0))↩
            );
301     multicast.AddMulticastRoute(c.Get(0), (Ipv4Address("").GetAny()), ↩
            Ipv4Address("230.0.0.30"), senderIf, NetDeviceContainer(nd2.Get(0)↩
            ));
302     multicast.AddMulticastRoute(c.Get(0), (Ipv4Address("").GetAny()), ↩
            Ipv4Address("230.0.0.30"), nd2.Get(0), NetDeviceContainer(senderIf↩
            ));
303
304     Ptr<Node> sender2 = c.Get(1);
305     Ptr<NetDevice> senderIf2 = nd0.Get(1);
306     multicast.SetDefaultMulticastRoute(sender2, senderIf2);
307
308
309
310
311     NS_LOG_INFO("Create Applications.");
312
313     uint16_t multicastPort = 9; // Discard port (RFC 863)
314
315
```

```
316     OnOffHelper onoff("ns3::UdpSocketFactory",
317             Address(InetSocketAddress(multicastGroup, multicastPort)));
318     onoff.SetConstantRate(DataRate("15kb/s"));
319     onoff.SetAttribute("PacketSize", UintegerValue(1400));
320
321
322     ApplicationContainer srcC = onoff.Install(c0.Get(1));
323
324     OnOffHelper onoff2("ns3::UdpSocketFactory",
325             Address(InetSocketAddress(multicastGroup2, 1336)));
326     onoff2.SetConstantRate(DataRate("10kb/s"));
327     onoff2.SetAttribute("PacketSize", UintegerValue(100));
328
329     ApplicationContainer srcC2 = onoff2.Install(c0.Get(1));
330
331
332     srcC.Start(Seconds(1.));
333     srcC.Stop(Seconds(499.));
334     srcC2.Start(Seconds(1.));
335     srcC2.Stop(Seconds(499.));
336
337
338     PacketSinkHelper sink("ns3::UdpSocketFactory",
339             InetSocketAddress(Ipv4Address::GetAny(), multicastPort));
340     ApplicationContainer sinkC = sink.Install(c1.Get(3));
341     sinkC.Start(Seconds(1.0));
342     sinkC.Stop(Seconds(100.0));
343
344     NS_LOG_INFO("Configure Tracing.");
345     //
346     // Configure tracing of all enqueue, dequeue, and NetDevice receive ↩
            events.
347     // Ascii trace output will be sent to the file "csma-multicast.tr"
348     //
349     AnimationInterface anim("animation.xml");
350     anim.SetMobilityPollInterval(Seconds(1));
351     anim.EnablePacketMetadata(true);
352
353     AsciiTraceHelper ascii;
354     csma.EnableAsciiAll(ascii.CreateFileStream("csma-multicast.tr"));
355
356     // Also configure some tcpdump traces; each interface will be traced.
357     // The output files will be named:
358     //     csma-multicast-<nodeId>-<interfaceId>.pcap
359     // and can be read by the "tcpdump -r" command (use "-tt" option to
360     // display timestamps correctly)
361     if (wifiok)csma.EnablePcapAll("aimf-multicastC", false);
362
363     wifiPhy.EnablePcap("aimf-multicastW", c1);
364
365     // Use GnuplotHelper to plot the packet byte count over time
366     GnuplotHelper plotHelper;
367     GnuplotHelper plotHelper2;
368     GnuplotHelper plotHelper3;
369     GnuplotHelper plotHelper4;
370
371
372
373     // Configure the plot.  The first argument is the file name prefix
374     // for the output files generated.  The second, third, and fourth
375     // arguments are, respectively, the plot title, x-axis, and y-axis ↩
            labels
```

```
376     plotHelper.ConfigurePlot("aimf-packet-byte-count",
377             "Hello Packet Byte Count Sent vs. Time",
378             "Time (Seconds)",
379             "Packet Byte Count");
380     plotHelper2.ConfigurePlot("aimf-packet-byte-count2",
381             "Hello Packet Byte Count Received vs. Time",
382             "Time (Seconds)",
383             "Packet Byte Count");
384     plotHelper3.ConfigurePlot("aimf-packet-byte-count3",
385             "Multicast Packet Byte Count Forwarded vs. Time",
386             "Time (Seconds)",
387             "Packet Byte Count");
388     plotHelper4.ConfigurePlot("aimf-packet-byte-count4",
389             "Multicast Packet Byte Count Received vs. Time",
390             "Time (Seconds)",
391             "Packet Byte Count");
392
393
394     // Specify the probe type, trace source path (in configuration ↩
            namespace), and
395     // probe output trace source ("OutputBytes") to plot.  The fourth ↩
            argument
396     // specifies the name of the data series label on the plot.  The last
397     // argument formats the plot by specifying where the key should be ↩
            placed.
398     plotHelper.PlotProbe(probeType,
399             tracePath,
400             "OutputBytes",
401             "Packet Byte Count",
402             GnuplotAggregator::KEY_BELOW);
403     plotHelper2.PlotProbe(probeType2,
404             tracePath2,
405             "OutputBytes",
406             "Packet Byte Count",
407             GnuplotAggregator::KEY_BELOW);
408     plotHelper3.PlotProbe(probeType3,
409             tracePath3,
410             "OutputBytes",
411             "Packet Byte Count",
412             GnuplotAggregator::KEY_BELOW);
413     plotHelper4.PlotProbe(probeType4,
414             tracePath4,
415             "OutputBytes",
416             "Packet Byte Count",
417             GnuplotAggregator::KEY_BELOW);
418
419     // Use FileHelper to write out the packet byte count over time
420     FileHelper fileHelper;
421     FileHelper fileHelper2;
422     FileHelper fileHelper3;
423     FileHelper fileHelper4;
424
425     // Configure the file to be written, and the formatting of output data↩
            .
426     fileHelper.ConfigureFile("aimf-packet-byte-count",
427             FileAggregator::FORMATTED);
428     fileHelper2.ConfigureFile("aimf-packet-byte-count2",
429             FileAggregator::FORMATTED);
430     fileHelper3.ConfigureFile("aimf-packet-byte-count3",
431             FileAggregator::FORMATTED);
432     fileHelper4.ConfigureFile("aimf-packet-byte-count4",
433             FileAggregator::FORMATTED);
```

```
434
435     // Set the labels for this formatted output file.
436     fileHelper.Set2dFormat("Time (Seconds) = %.3e\tPacket Byte Count = %.0f");
437     fileHelper2.Set2dFormat("Time (Seconds) = %.3e\tPacket Byte Count = %.0f");
438     fileHelper3.Set2dFormat("Time (Seconds) = %.3e\tPacket Byte Count = %.0f");
439     fileHelper4.Set2dFormat("Time (Seconds) = %.3e\tPacket Byte Count = %.0f");
440
441     // Specify the probe type, trace source path (in configuration namespace), and
442     // probe output trace source ("OutputBytes") to write.
443     fileHelper.WriteProbe(probeType,
444             tracePath,
445             "OutputBytes");
446     fileHelper2.WriteProbe(probeType2,
447             tracePath2,
448             "OutputBytes");
449     fileHelper3.WriteProbe(probeType3,
450             tracePath3,
451             "OutputBytes");
452     fileHelper4.WriteProbe(probeType4,
453             tracePath4,
454             "OutputBytes");
455
456     //
457     // Now, do the actual simulation.
458
459
460     //
461     NS_LOG_INFO("Run Simulation.");
462
463     Simulator::Schedule(Seconds(1.0), &aimf::RoutingProtocol::ChangeWillingness, aimf_Gw, 2);
464     Simulator::Schedule(Seconds(1.0), &aimf::RoutingProtocol::ChangeWillingness, aimf_Gw2, 3);
465     Simulator::Schedule(Seconds(1.0), &aimf::RoutingProtocol::ChangeWillingness, aimf_Gw3, 4);
466     Simulator::Schedule(Seconds(50.0), &aimf::RoutingProtocol::ChangeWillingness, aimf_Gw, 5);
467     Simulator::Schedule(Seconds(100.0), &aimf::RoutingProtocol::ChangeWillingness, aimf_Gw2, 6);
468     Simulator::Schedule(Seconds(3.0), &aimf::RoutingProtocol::AddHostMulticastAssociation, aimf_Gw, multicastGroup, multicastSource);
469         Simulator::Schedule(Seconds(200.0), &aimf::RoutingProtocol::ChangeWillingness, aimf_Gw2, 3);
470     Simulator::Schedule(Seconds(4.0), &aimf::RoutingProtocol::AddHostMulticastAssociation, aimf_Gw, multicastGroup2, multicastSource2);
471
472     Simulator::Schedule(Seconds(200.0), &aimf::RoutingProtocol::ChangeWillingness, aimf_Gw3, 7);
473
474
475
476
477
478
479     Simulator::Stop(Seconds(500.0));
```

```
480     Simulator::Run();
481
482
483     Simulator::Destroy();
484     NS_LOG_INFO("Done.");
485 }
```

A.2 Routing table computation in AIMF.

```
1  void
2          RoutingProtocol::RoutingTableComputation() {
3              NS_LOG_DEBUG(Simulator::Now().GetSeconds() << " s: Node " << ↩
                   m_mainAddress
4                      << ": RoutingTableComputation begin...");
5
6
7              Clear();
8
9
10             std::vector<uint32_t> outint(m_netdevice.size() - 1);
11             std::copy(m_netdevice.begin(), m_netdevice.end(), std::↩
                   back_inserter(outint));
12             const Associations &localHmaAssociations = m_state.↩
                   GetAssociations();
13             for (Associations::const_iterator assocIterator = ↩
                   localHmaAssociations.begin();
14                     assocIterator != localHmaAssociations.end(); ↩
                           assocIterator++) {
15                 Association const &localHmaAssoc = *assocIterator;
16                 AddEntry(localHmaAssoc.group, localHmaAssoc.source, m_ipv4↩
                       ->GetInterfaceForAddress(m_mainAddress), outint);
17
18                 NS_LOG_DEBUG("Node " << m_mainAddress << ": Adding local (↩
                       " << localHmaAssoc.group << "," << localHmaAssoc.↩
                       source << ") pair to routing table. OuputInterface is:↩
                        " << m_ipv4->GetAddress(outint.front(), 0).GetLocal()↩
                       << " and Input is: " << m_ipv4->GetAddress(1, 0).↩
                       GetLocal());
19             }
20
21
22             const AssociationSet &localHmaAssociationSets = m_state.↩
                   GetAssociationSet();
23             for (AssociationSet::const_iterator assocSetIterator = ↩
                   localHmaAssociationSets.begin();
24                     assocSetIterator != localHmaAssociationSets.end(); ↩
                           assocSetIterator++) {
25                 AssociationTuple const &localHmaAssocSet = *↩
                       assocSetIterator;
26                 AddEntry(localHmaAssocSet.group, localHmaAssocSet.source, ↩
                       m_ipv4->GetInterfaceForAddress(m_mainAddress), outint)↩
                       ;
27
28                 NS_LOG_DEBUG("Node " << m_mainAddress << ": Adding ↩
                       adjacent (" << localHmaAssocSet.group << "," << ↩
                       localHmaAssocSet.source << ") pair to routing table. ↩
                       OuputInterface is: " << m_ipv4->GetAddress(outint.↩
```

```
                    front(), 0).GetLocal() << " and Input is: " << m_ipv4↩
                    ->GetAddress(1, 0).GetLocal());
29          }
30
31
32
33          NS_LOG_DEBUG("Node " << m_mainAddress << ": ↩
                RoutingTableComputation end.");
34          m_routingTableChanged(m_table.size());
35
36      }
```

A.3 Structs used in the modell of AIMF.

```
1   struct NeighborTuple {
2           /// Main address of a neighbor node.
3           Ipv4Address neighborMainAddr;
4           /// Neighbor instance deletion time.
5           Time expirationTime;
6           /// A value between 0 and 7 specifying the node's
7           /// willingness to carry traffic compared to other nodes.
8           uint8_t willingness;
9       };
10
11  struct IfaceAssocTuple {
12          /// Interface address of a node.
13          Ipv4Address ifaceAddr;
14          /// Main address of the node.
15          Ipv4Address mainAddr;
16          /// Time at which this tuple expires and must be removed.
17          Time time;
18      };
19
20   struct AssociationTuple {
21          /// The advertiser of the multicast group association.
22          Ipv4Address advertiser;
23          /// The group which is advertsed.
24          Ipv4Address group;
25          /// Optionally the source of the group which is advertised.
26          /*There is made a */ new /* instance of this
27          */ struct /* when there are multiple sources. */
28          Ipv4Address source;
29          /// Time at which this tuple expires and must be removed
30          Time expirationTime;
31          /// The received TTL of group when SSM is preferred.
32          uint8_t will;
33      };
34
35  struct Association {
36      /// The group which is advertised.
37          Ipv4Address group;
38          /// Optionally the source of the group which is advertised.
39          /// There is made a new instance og this
40          /// struct when there are multiple sources.
41          Ipv4Address source;
42          /// This is not yet implemented, but the idea behind
43          /// it is to save the "IGMP-interface"
44          /// responsible for the injecting.
```

```
45              Ipv4Address advertiser;
46             /// The received TTL of group when SSM is preferred.
47              uint8_t will;
48          };
```

A.4 Kildekode for SMF og AIMF

Kildekoden for SMF og AIMF vil ligge ved oppgaven og vil være tilgjengelig på GitHub.

Link til GitHub-lager for SMF. (https://github.com/debonatis/SMF)
Link til GitHub-lager for AIMF. (https://github.com/debonatis/AIMF)
Kontrollskript ligger under AIMF/examples.

A.4.1 Legge til moduler i ns3

Først, lag en mappe i *src*-mappen i ditt ns3 miljø som heter aimf og/eller smf. Deretter last ned eller bruk Git til å «pull»'e AIMF og/eller SMF inn i respektive mapper. Til slutt, kjør *./waf configure –enable-examples –enable-tests*. Opsjonene *–enable-examples* og *–enable-tests* er ikke nødvendig, men vanlig å ha med.