NTNU
Det skapende universitet

Self-configured demonstrator of the Cross Entropy Ant System

Tore André Kristiansen

Master i kommunikasjonsteknologi
Oppgaven levert: Juni 2007
Hovedveileder: Poul Einar Heegaard, ITEM
Biveileder(e): Ingebrigt Fuglem, Telenor R&D

Norges teknisk-naturvitenskapelige universitet
Institutt for telematikk

Oppgavetekst

The Cross Entropy Ant System (CEAS) is a distributed, robust and adaptive swarm intelligence system for path management in communication networks. The CEAS is performing adaptive multi-path load sharing and stochastic routing with fast restoration on link failures. Previous work have shown that CEAS are robust and efficient in solving complex optimisation problems like finding primary and backup paths and pcycles in networks, and also finding paths in network with changes in topology and traffic load and patterns.

As a proof-of-concept, a prototype implementation of a swarm based routing in software IP router, named AntPing, demonstrates the principles in a small-scale network. The work on the swarm based routing is based on research at Q2S and ITEM at NTNU, and in EU-IST project BISON (http://www.cs.unibo.it/bison/) where Telenor R&D participated. The demonstrator is a product of the BISON project. The demonstrator should be extended to allow self-configurations of routers and multiple virtual connections.

In a project work different strategies for self-configuration of addresses were considered. In this master assignment the main focus is on implementing a selfconfigured demonstrator on the AntPing platform. This includes:

1. Select and design a scheme for autonomous self-configuration of router addressed.
2. Investigate CEAS to integrate the address scheme in AntPing. Identify and specify necessary extensions to AntPing to include the addressing scheme.
3. Implement lab environment and extend AntPing functionality to enable addressing and multi-vc.
4. Select and describe a demonstrator scenario and discuss possible extensions and corresponding AntPing modifications.
5. Verify and demonstrate the address assignment scheme in AntPing.

Oppgaven gitt: 18. januar 2007
Hovedveileder: Poul Einar Heegaard, ITEM

Forord

Denne rapporten er et resultat av min masteroppgave "Self-configured demonstrator of the Cross Entropy Ant System" ved ITEM NTNU våren 2007. Oppgaven bygger på min prosjektoppgave "Discovery protocol in AntPing" [15] utført høsten 2006 og konseptet AntPing [12] som er et arbeid utført av Poul Heegaard, NTNU og Ingebrigt Fuglem, Telenor. Masteroppgaven har hovedfokus på å implementere forslaget til løsning som ble gitt i prosjektet. Selv om ikke implementasjonen er helt lik den foreslåtte løsningen er tankegangen stort sett den samme. I tillegg ble det underveis i arbeidet med masteroppgaven klart at det ble rom for å utvide denne til også å omfatte implementering av ytterligere en funksjon. Dette står ikke i oppgaveteksten men er beskrevet i problemstillingen. Oppgaven er en praktisk rettet oppgave, i dette ligger det mange utfordringer og såkalte uforutsette problemstillinger. Det er ikke alltid at kart og terreng stemmer helt overens, men disse utfordringene føler jeg har bidratt til å øke min forståelse for problemstillingen. Alt i alt mener jeg å ha lært mye praktisk utviklingsarbeid, jeg har hatt det moro, og ikke minst fått ytterligere bekreftelse på det jeg visste fra før - det er forskjell på teori og praksis.

Bjarne Hellevik og Poul Heegaard inviterte meg til å vise frem demonstratoren i en undervisningstime i faget "TTM4120 Dependable systems" den 8. mai. Demoen startet med et nettverk tilsvarende det scenario som er vist i kapittel 4, linkene ble tatt bort, flyttet og lagt til etter forslag fra salen. Fremvisningen forløp helt uten problemer.

Da oppgaven er skrevet på norsk og all litteratur om emnet har vært på engelsk har utfordringen vært å finne den rette balansen mellom norske og engelske faguttrykk. Jeg har derfor forsøkt å bruke norske ord der jeg synes det er hensiktsmessig og engelske faguttrykk der jeg synes det passer best.

Ellers takk til Poul Heegard og Ingebrigt Fuglem for hjelp og støtte underveis når det har buttet imot.

Trondheim 11. juni 2007.

Tore A. Kristiansen

Innhold

1 Innledning **1**

1.1 Bakgrunn 1

1.1.1 Distance-Vector 2

1.1.2 Link-State ruting [17] 3

1.1.3 MultiProtocol Label Switching (MPLS) 3

1.1.4 AntPing - CEAS 4

1.2 CEAS vs standard rutingprotokoller 6

1.3 CEAS vs MPLS 6

1.4 Problemstilling 7

1.5 Avgrensninger 7

1.6 Oppbygning 7

2 Verktøy **8**

2.1 Click softwareruter 8

2.1.1 Generelt 8

2.1.2 Element 9

2.1.3 Ruterkonfigurasjon 9

2.1.4 Kodingsprinsipper 10

2.2 Hping 11

2.3 Network Animator 12

2.4 OpenWRT 12

2.5 Linksys WRT54GL ruter 13

3 Implementering **15**

3.1 Valg av design . . . 15

3.1.1 Vurderte løsninger . . . 16

3.2 Implementert løsning . . . 19

3.2.1 Pakkeflyt i AntPing click . . . 19

3.3 Pakketyper i AntPing . . . 22

3.4 Oppdatering av AntArpTable . . . 22

3.5 Elementene . . . 26

3.5.1 AntArp . . . 26

3.5.2 AntArpTable . . . 26

3.5.3 UpdateAntArp . . . 27

3.5.4 UpdatePathTable . . . 28

4 Test av implementerte funksjoner **29**

4.1 Oppsett . . . 29

4.2 Testscenario . . . 31

4.3 Kommentarer til testforløpet . . . 33

5 Konklusjon **34**

6 Videre arbeider **35**

A Tcl **37**

B IPv4 Link-lokal adresse [10] **38**

B.1 Oversikt . . . 38

B.2 Adressevalg . . . 38

B.3 Adresseannonsering . . . 39

B.4 Håndtering av adressekonflikter . . . 39

B.5 Konfigurasjonsregler . . . 39

C Loose source routing - Route record [16] **40**

D Proxy-ARP [9] **42**

E Klargjøring av Linksysruter **44**

E.1 Kompilering av Click til Linksys WRT54GL med OpenWRT 44

E.2 Installering av OpenWRT. 45

E.3 Installering av nødvendige biblioteker og script. 48

E.3.1 Konfigurere WAN port. 48

E.3.2 Installering av nødvendige software og script samt konfigurering av vlan-porter. 49

F Oppstartsprosedyrer testoppsett **50**

G Oppstart av demo **53**

H Hping-script **55**

I Click-script **65**

J Innhold på vedlagte arkivfil **69**

Bibliografi **71**

Figurer

1.1 Route record for forwardants og source routing for backwardants [12] 5

2.1 Lagdeling Click vs Linux-ruter i user mode 8

2.2 Et Click element 9

2.3 Visualisering av enkelt click-script med en enkel prosess. Pakken kommer fra grensesnittet (eth0), blir telt og så forkastet. 10

2.4 Push og pull porter i en Click-ruter [14] 10

2.5 Linksys WRT54GL ruter [5] 13

2.6 Linksys WRT54GL ruter skjematisk oversikt [5] 14

3.1 Fysisk topologi og adressering i AntPing [12] 16

3.2 Flytdiagram av foreslått løsning fra prosjektoppgaven [15] 18

3.3 Click flytskjema av implementert løsning, se også tillegg Hmed tilhørende Click-script 21

3.4 Oppdatering av rutingtabell. Innhold i forward-/backwardant-pakken og GWtable i tidspunkt 1 til 6. 25

4.1 Testoppsett av "nye AntPing" 30

4.2 Animasjon og plott av scenario - a) T0 ingen linker plugget inn b) T10 alle noder og linker er oppe c) T37 link 1-3 og link 1-5 fjernes d) T52link 1-2 fjernes e) T55 Link 1-2 legges til igjen f) T90 Link 1-2 flyttes til 1-6 g) Plott av kostverdier og eliteantlimit under testscenarioet 32

C.1 . Source routing [12] I denne figuren representerer # pekeren. Hvis den peker lengst til venstre vil pekerverdi = 4, deretter 8, 12. 41

D.1 Proxy-ARP 42

E.1 Linksys WRT54GL oppstartskonsoll 46

E.2 Firmware oppgradering ved hjelp av Linksys konsoll 47

E.3 OpenWRT konsoll 47

Tabeller

3.2 Pakketyper og innhold 22

4.1 Kostverdier tilknyttet utgående ethernetport 30

Sammendrag

Etter hvert som Internett har vokst har prosessen med å finne rett rutingvei blitt mer og mer komplisert. Fastsetting av metrikker som danner grunnlag for rutingprotokollens beregning av beste rutingvei blir oftest gjort manuelt. Dette kan være svært komplisert og krever en dyp innsikt i problemstillingen. Behovet for algoritmer og protokoller som er i stand til automatisk å løse dette på en tilfredsstillende måte blir større og større etter hvert som nettet vokser. AntPing er et prototypekonsept som er laget for blant annet å kunne demonstrere og visualisere en effektiv og pålitelig metode for å optimalisere rutingveier i et IP-nettverk. AntPing baserer seg på bruk av svermintelligens og er implementert ved hjelp av Click softwareruter som kjører på embedded Linux installert i små hjemmerutere. Jeg har i denne masteroppgaven implementert utvidet funksjonalitet til demonstratoren AntPing for å bedre det visuelle inntrykket demonstratoren vil gi en forsamling. AntPing er utvidet til å bli selvkonfigurerende slik at nye linker kan legges til og fjernes, og at dette blir animert ved hjelp av Nam. I tillegg er funksjonaliteten til ruterne utvidet slik at man har redusert behovet for laptoper under kjøring av demonstratoren fra to til en. Jeg har lagt ved "kokeoppskrifter", nødvendige script, kildekode og binærfiler for å kunne installere den utvidede AntPing på Linksys hjemmerutere. Jeg har også foreslått forbedringer og utvidelser av demonstratoren.

Kapittel 1

Innledning

1.1 Bakgrunn

Ruting, optimalisering av rutingveier og styring av disse er et stort og omfattende tema. Emnet har mange fasetter, og det er publisert og utviklet mange protokoller som har til hensikt å styre rutingveiene samt å gjøre rutingen mer effektiv.

Ruterne har ofte implementert en eller annen dynamisk rutingprotokoll eller så er rutingen basert på manuelt oppsatt statisk ruting for å finne beste vei gjennom nettverket. Selv om rutingmekanismene er dynamiske blir de beste rutingveiene ofte beregnet ut fra statiske metrikker på linkene, fastsatt av nettadministrator. Dette er et enkelt og robust system men ikke nødvendigvis særlig effektivt. Når den eneste måten å drive metningskontroll i linkene på er at senderaten styres fra kilden ved hjelp av TCP, blir resultatet ofte en ikke særlig effektiv måte å utnytte linkkapasiteten på. Etter hvert som Internett har fått enorme dimensjoner ser man at behovet for mer effektive rutingprotokoller er økende. Dette ikke minst fordi det å manuelt beregne metrikkene kan være svært krevende og en nesten umulig oppgave på store nettverk.

Internett er konservativt, med denne påstanden mener jeg at det i bunn og grunn er lite som har forandret seg med hensyn til de grunnleggende prinsippene om hvordan rutingen blir styrt på. De statiske metrikkene på linkene fastsettes ofte ved hjelp av manuelle utregninger og metoder (eksempelvis bruk av ping for å finne delay). Nettet er i dag styrt av semiautonome systemer. Med det mener jeg at man har autonome rutingprotokoller som finner "beste" vei, men dette baserer seg på manuelt fastsatte metrikker beregnet og styrt av nettadministrator. Innføring av nye og mer effektive rutingprotokoller går tregt, og nye metoder må vise klar forbedring og samtidig ikke endre mye på den grunnleggende måten nettet i dag styres på for å få gjennomslag. Det kan synes at man ikke stoler helt på at de autonome protokoller klarer å gjøre jobben fullt ut eller at disse ennå ikke er blitt gode nok. Etter min mening er ikke denne semiautonome måten å styre nettet på helt i tråd med designkravene til det opprinnelige Internett.

TCP/IP ble laget for å knytte sammen mange ulike typer nettverk. IP-nettet skulle kun være en formidler av datapakker mellom disse nettene og det viktigste designkriteriet var overlevelse. Nettet skulle tåle bortfall av rutere eller linker uten at vital informasjon skulle gå tapt. Dette betydde at nodene i nettet måtte være tilstandsløs og at tilstandsdataene for sesjonen måtte ligge i endene. Slik ville ingenting kunne gå tapt uten at en av de kommuniserende parter på endene av nettverket feilet (noe som uansett vil være ødeleggende for enhver kommunikasjon).

For å få best mulig overlevelse i et nett må hver enkelt node være uavhengig av andre noder eller styring fra eksterne systemer for å overleve, altså mest mulig autonom. Nettet er i dag oppbygd av autonome rutere, men metrikken blir satt manuelt og er avhengig av en sentral kilde. Samtidig blir Internett stadig mer komplisert, noe som medfører at det blir stadig vanskeligere å fastsette riktige metrikker manuelt. Behovet for bedre og mer automatiserte rutingmekanismer blir derfor stadig mer påtrengende. Forslag til løsning av denne problemstillingen kan grovt sett basere seg på to hovedprinsipper:

1. Man kan innføre en mer sentral mekanisme som automatisk regner ut beste rutingvei for så å fordele dette ut til ruterne, enten ved å styre metrikkene eller fullstendig ta kontroll over rutingtabellene. Et eksempel på dette er [11]. Disse metodene gir ikke den enkelte node i nettet full autonomitet.
2. Eller man kan innføre nye og bedre distribuerte mekanismer/protokoller i nettet som lar nettet regulere seg selv. Et eksempel på dette er CEAS (Cross Entropy Ant System) [12].

Uansett, utfordringene er at for å kunne konfigurere nettet hurtig og riktig må man hele tiden ha ferske og riktige måledata for å kunne detektere eksempelvis metning eller utfall av linker. Kravet om ferske måledata må ikke føre til at nettet oversvømmes av konfigurasjonsdata, og det blir derfor en selvfølge at protokoller som er ment å effektivisere rutingen ikke overbelaster nettet med "overhead" i form av konfigurasjonsmeldinger. Pr i dag er den foretrukkede måten å styre Internett på den distribuerte med mest mulig autonome rutere selv om krav til QOS (Quality of Service) for enkelttjenester, og enkeltbrukere presser på for en mer sentralisert styring og kontroll.

1.1.1 Distance-Vector

DV (Distance-Vector) [17] algoritmer er et eksempel på dynamiske rutingprotokoller. Disse lar hver enhet i nettverket bygge og vedlikeholde sin egen lokale rutingtabell. Prinsippet bak DV-ruting er enkelt. Hver ruter i nettverket vedlikeholder sin distanse eller kostverdi fra seg selv til alle kjente destinasjoner i nettet. Denne verdien representerer den samlede distansen/kostverdien til denne stien. Den av stiene som har den laveste kostverdi vil bli foretrukket foran de andre. Denne informasjonen er lagret i DV-tabell. DV-tabellen blir periodisk sendt til alle nabonoder, uansett om det har oppstått endringer eller ikke. Hver enkelt ruter prosesserer så alle disse mottatte

tabellene, og ut fra disse regner de ut den beste stien gjennom nettverket. Hovedfordelen med DV-protokollene er at de er svært enkle å implementere samt oversiktlige å feilsøke på, men de fungerer best i små enkle nettverk med liten redundans. Det er uansett mange ulemper med denne typen protokoller. Når det oppstår en feil vil det ta en viss tid til at hele nettet har registrert dette og kalkulert beste rute. Denne tiden kalles konvergenstid. I store kompliserte nettverk vil konvergenstiden ved bruk av en DV-rutingprotokoll være betydelig. Dette medfører at tiden fra en linkfeil har oppstått til at rutingtabellene er rekalkulert vil være alt for stor, og i mellomtiden vil rutingen være upålitelig.

RIP (Route Information Protokoll) er en populær DV-rutingprotokoll.

1.1.2 Link-State ruting [17]

Internett har med årene blitt enormt stort og med dette har kompleksiteten økt. For å bøte på svakhetene ved DV-rutingprotokoller ble det etter hvert nødvendig å utvikle mer robuste rutingprotokoller. Algoritmen som disse mer robuste protokollene bruker til å finne rutingveier baserer seg på link state-informasjonen. En link state er en beskrivelse av et grensesnitt på en ruter og dets forhold til naboruterne f.eks IP-adresse, subnettmaske og type nettverk. Samlingen av disse link state danner en link state-database. Prosessen som link state-algoritmene bruker for å oppdage nettverkstopologien er som følger:

- Hver ruter identifiserer alle sine direkte tilknyttede naboer.
- Hver ruter sender ut en liste til hele nettverket om hvem de direkte tilknyttede naboene er sammen med assosiert link kostverdi gjennom en såkalt link state advertisement (LSA).
- Hver ruter lager så sin egen topologidatabase ut fra de mottatte LSA og sitt kjennskap til eget nabolag. Dette gir identiske topologidatabaser i alle ruterne.
- Ruterne bruker så disse til å kalkulere/oppdatere sine egne rutingtabeller.

Dette gir raskere konvergenstid enn en DV-protokoll, men samtidig krever dette mer prosessorkraft og minne av den enkelte ruter, noe som igjen krever større og kraftigere rutere for å kunne fungere tilfredsstillende.

Open Shortest Path First (OSPF) er en velkjent og mye brukt link state protokoll.

1.1.3 MultiProtocol Label Switching (MPLS)

MPLS [17] er en måte å emulere et linjesvitsjet nett i et pakkesvitsjet nett og kan derfor ikke direkte sammenlignes med IP-ruting. Dette er en mekanisme som netteiere har for å kunne tunnelere og styre trafikken i nettet. Den gir blant annet mulighet til å styre QOS med hensyn

på bruker eller trafikktype. MPLS opererer mellom lag 2 og lag 3 i IP-protokollen, ofte blir dette referert til som lag 2,5.

Kort fortalt om virkemåten:

MPLS gir alle datapakker en ekstra header som inneholder en eller flere "labels". Dette blir kalt en "label stack". Disse pakkene er svitsjet i en "Label lookup" svitsj istedet for oppslag i en rutingtabell. Label-svitsjing vil være en raskere måte å svitsje på enn et oppslag i en rutingtabell. Ruterne i ytterpunktene av et MPLS nettverk kalles Label Edge Routers (LER). Rutere som kun utfører ruting basert på "Label Switching" kalles "Label Switch Routers" (LSR). Når en umerket datapakke ankommer en LER og skal bli tunnelert må ruteren først bestemme hvor pakken skal og deretter sette på en eller flere Labels. Den blir så sendt til neste ruter som basert på den øverste Labelen skifter den ut med en ny Label, og sender deretter pakken videre til neste ruter. Flere lag med Labels kan eksempelvis brukes til hierarkisk ruting eller til MPLS VPN. MPLS bruker vanlig IP-rutingprotokoller for å finne beste sti.

1.1.4 AntPing - CEAS

AntPing [12] er en prototype implementasjon av et distribuert og robust stifinner og monitoreringssystem kalt CEAS (Cross Entropy Ant System). AntPing er realisert ved hjelp av "Click modular router" og Linksys WRT54GL hjemmeruter.

- Kjernen i AntPing er en stokastisk rutingalgoritme kalt CEAS (Cross Entropy Ant System).
- AntPing registrerer "forward route" (route records), datagrammene blir deretter sendt tilbake samme vei ved hjelp av source routing.
- AntPing legger til ekstra statistikk i ruterne som er implementert ved hjelp av Click. IP-adressene på hvert hopp er lagt til i IP-header og time stamp er lagt til i payload.
- AntPing sender UDP pakker (ant) på port 51234.
- Det er flere typer ant i AntPing (normal, exploration, forward, backward, update)

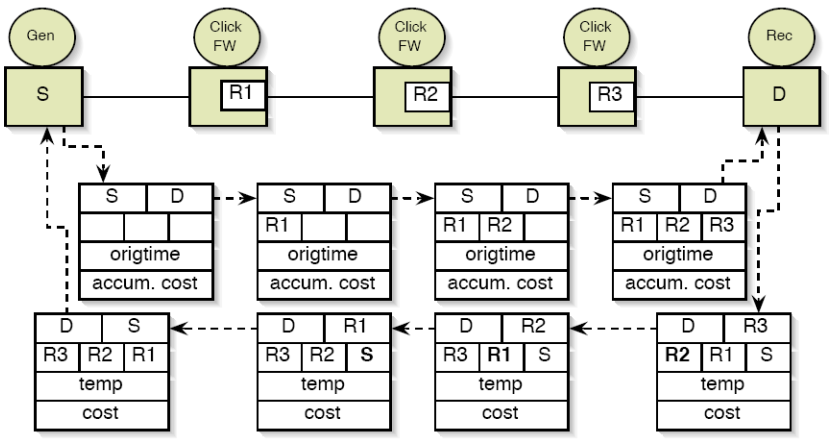
AntPing krever pr i dag en egen generatorenhet som genererer CE ant, det vil si UDP datagrammer, en mottaker av disse samt at det må være implementert støtte for å oppdatere og rute/videresende disse i ruterne. Figur 1.1 viser et eksempel på hvordan pakkeflyten i AntPing virker. En forwardant blir generert i kilden S. Tidspunktet da denne ble generert blir addert til datalasten, kildeadressen blir satt til S og måladressen blir satt til D. Ved ruter R1 blir måladressen lest og neste hopp blir valgt enten i henhold til CE ant rutingalgoritme eller som en uniform fordeling over alle tilgjengelige grensesnitt. Det siste skjer hvis forwardant-typen er explorer. Etter at neste hopp er bestemt vil optionfeltet i IP-headern bli oppdatert med dette grensesnittets adresse, og deretter vil den bli videresendt på valgt grensesnitt. Dette skjer for hver ruter som forwardant er innom helt til den når målet D. Her kan 3 ting skje:

1. Ved D vil det bli undersøkt om dette er den første forwardant som har ankommet fra S. Hvis dette er tilfellet blir dette registrert som en ny forbindelse, kostverdien bli kalkulert, temperaturen addert til datalasten og backward CE ant datagram vil bli sendt tilbake til S som såkalt eliteant. Tilbaketuren, det vil si backward(update)ant, vil benytte source route. Dette for å kunne oppdatere eller skrive nye ferromonverdier (luktestoffverdier) for denne ruten i alle ruterne på tilbakeveien. Når backwardant er ankommet til S igjen blir nødvendig monitoreringsdata skrevet til en loggfil.

2. Hvis dette ikke er første forwardant fra S vil man utføre de samme kalkulasjoner og addere dette til datalasten, det vil så bli gjort en eliteant seleksjon. Dette er nærmere beskrevet i [12]:

   (a) Hvis dette er en eliteant vil den på samme måte som 1. bli sendt tilbake til kilden som eliteant.

   (b) Hvis det derimot ikke er en eliteant vil den bli sendt tilbake på samme måte, men ruterne på tilbakeveien vil ikke få oppdatert sine ferromonverdier. Pakken returneres kun for animasjonsformål.



Figur 1.1: Route record for forwardants og source routing for backwardants [12]

Man har nå ved hjelp av initiell explorerant kartlagt de beste rutingveiene/stiene i nettet. Man vil deretter gå over i en normal fase der man vedlikeholder de oppdagede rutingveiene bare avbrutt av perioder med explorerant for å oppdage eventuelt nye stier. Det essensielle med denne metoden er at man i stedet for bare å bruke den beste stien fra kilde til mål, vil bruke mange stier og man regner ut en vektet fordeling mellom disse basert på kostverdiene. Disse verdiene kan være tid, antall hopp eller en kombinasjon av disse. Eksempelvis har vi ved S to mulige/oppdagede stier

til D med vekting (0,8-0,2). Den første gjennom grensesnitt A, den andre gjennom grensesnitt B. Man vil da sende forwardant med stokastisk fordeling lik 0,8 og 0,2 gjennom hhv A og B. Disse ferromonverdiene vil endres over tid og blir hele tiden oppdatert av backwardant.[1] Som følge av eliteseleksjonen vil de beste stiene bli oppdatert og dermed få en bedre vektet fordeling og vil bli brukt oftere og oftere. For å monitorere og visualisere dette er det i AntPing inkludert støtte for animering gjennom Network Animator (Nam). For en mer detaljert beskrivelse vises det til [12].

1.2 CEAS vs standard rutingprotokoller

Stifinner- og monitoreringssystemet CEAS er et forslag til å gjøre rutingen i Internett raskere og mer robust fordi den fordeler trafikken jevnere ut over hele nettverket samtidig som den gjør bruk av sanntids kostdata. Tradisjonelle rutingprotokoller (RIP/OSPF) har forskjellige strategier på hvordan finne beste vei gjennom et nettverk. Disse protokollene har det til felles at de velger ut **en** beste vei for en rute fra A til B, for deretter å rute all trafikk denne veien, ofte på grunnlag av ikke helt ferske kostdata. Dette i motsetning til CEAS som finner mange veier og sender datapakkene ut på flere av [2] disse basert på en stokastisk vektet fordeling. Dette skjer på grunnlag av kostdata som både er ferske og som i tillegg tar hensyn til kosthistorikken. Mange ting er imidlertid likt med tradisjonell ruting. CEAS produserer en rutingtabell som vil kunne rute en helt vanlig datapakke. Denne datapakken trenger ikke å bli modifisert på noen som helst måte, og man trenger dermed ikke noe ekstralogikk for å bearbeide datatrafikken i endene. Ruteren vil slå opp i tabellen som er basert på både sender- og mottakeradresse (sti), i tillegg har den en ekstralogikk som sørger for en vektet fordeling av pakkene ut på de forskjellige rutingveiene som tabelloppslaget viser. Systemet er distribuert og bryter dermed ikke med de gamle designmålene. Sett fra endebruker vil den altså virke som en vanlig rutingmekanisme og ingen modifikasjoner må gjøres på datapakken.

1.3 CEAS vs MPLS

Hver ruter konfigurert med CEAS inneholder et innslag i en tabell basert på en "label" bestående av sender- og mottakeradresse som tilsammen gir en sti. Dette gjør det nærliggende å sammenligne med MPLS som også lager faste stier fra A til B. Likheten mellom MPLS og CEAS ender imidlertid der. MPLS legger til en ekstra "label" på toppen av en datapakke, mens CEAS ikke trenger å modifisere datapakken. MPLS emulerer linjesvitsjing mens CEAS bare endrer litt på måten man ruter datatrafikken på. CEAS sørger for en bedre fordeling av trafikken i nettet siden man benytter flere veier fra A til B, og ikke bare den beste. Ruting basert på CEAS har ingen andre krav enn at pakken har en IP-header med sender- og mottakeradresse akkurat som vanlige

[1] I et stabilt nett vil disse verdiene etter en stund konvergere mot en stabil verdi.
[2] Prinsippet med eliteant reduserer vektingen av "de dårligste stiene" og øker vektingen på de beste.

rutingprotokoller. MPLS baserer seg på vanlige IGP for å finne beste sti, dermed kan man faktisk tenke seg at CEAS vil kunne brukes til å finne stier for MPLS [13].

1.4 Problemstilling

Demonstratoren AntPing er implementert ved hjelp av 10 små hjemmerutere der linkene er statiske og definert på forhånd. Det er ikke mulig å legge til nye linker underveis under kjøring. Fokuset på denne oppgaven blir derfor i prioritert rekkefølge den fysiske implementasjonen av følgende funksjonalitet:

1) At man under kjøring lett skal kunne legge til nye linker, og at disse automatisk skal bli implementert i nettverket og visualisert ved hjelp av Network Animator (Nam) eller lignende.

2) For å redusere antall hardware-komponenter man trenger for å kjøre demonstratoren ønsker man å implementere mottakeren for forwardant inn i alle ruterne. Til dette formålet trenger man i dag en PC som kommer i tillegg til den som sender ut disse. Ved å implementere denne mekanismen i hver enkelt ruter vil systemet bli mindre utstyrskrevende, og man vil dermed også enklere kunne sende forwardant til flere mål samtidig. Dette for at demonstrasjonen skal kunne gjøres mer realistisk.

Oppgaven bygger videre på en prosjektoppgave "Discovery protocol in AntPing" utført ved ITEM NTNU høsten 2006 av undertegnede [15].

1.5 Avgrensninger

Oppgaven går ikke ut på å endre/forbedre selve konseptet, men å forbedre demonstratoren ved å gjøre den mer fleksibel og dermed bedre kunne demonstrere de gode stifinneregenskapene ved AntPing konseptet. Hele konseptet er laget og testet med Linux Ubuntu 6.06 som OS.

1.6 Oppbygning

Jeg vil først gi en kort oversikt over rutingmekanismer i Internett, AntPing og de verktøyene som er brukt for å realisere demonstratoren. Jeg vil beskrive hvorfor den implementerte løsning ble valgt og deretter vise hvordan den ferdige implementasjonen er bygd opp samt virkemåte. Til slutt vil jeg ved hjelp av et testoppsett vise at implementasjonen fungerer i praksis. I tillegget har jeg lagt til noe grunnleggende teori samt script og "kokeoppskrifter" på hvordan man starter og bruker demonstratoren.

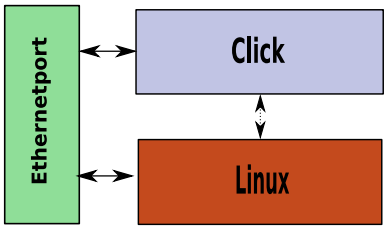
Kapittel 2

Verktøy

2.1 Click softwareruter

2.1.1 Generelt

Click [14] er software arkitektur for oppbygging av fleksible og konfigurerbare rutere. Ideen bak Click er å tilby et verktøy for enkelt å kunne skrive pakkeprosesser. Click er open source og kan dermed fritt benyttes og videreutvikles. Man kan tilføye og endre elementer slik at de kan tilpasses den enkeltes behov. Vanlige rutere er ikke konfigurerbare i den forstand, man kan endre noen settinger men ikke den fundamentale måten ruteren arbeider på. Click kan kjøres i user eller kernel mode. Når man kjører i user mode, som er tilfellet med nåværende utgaven av AntPing, vil Click opptre som en hvilken som helst annen applikasjon. Dette medfører at den innebygde ruteren i Linux vil kjøre som normalt, se figur 2.1. Man kan også kjøre i kernel mode, da vil Click-ruteren erstatte den opprinnelige Linux-ruteren.



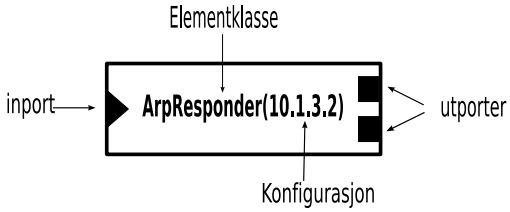
Figur 2.1: Lagdeling Click vs Linux-ruter i user mode

Som vist i figuren over fungerer ruterne uavhengige av hverandre i user mode. For å hindre parallell ruting kan man definere opp forskjellige IP-adresser i Click og Linux eller man kan la

være å definere opp IP-adresser i Linux. Hvis man unnlater å definere opp IP-adresser i Linux vil denne selvsagt ikke motta IP-pakker og dermed ikke fungere som ruter. Det er da kun Click-ruteren som vil stå for rutingen.

2.1.2 Element

En "Click element router" er bygd sammen av pakkeprosessmoduler kalt elementer. Elementene er subklasse av elementklassen. Individuelle elementer implementerer enkle rutingfunksjoner som for eksempel pakkeklassifisering, køing og grensesnitt mot nettverksenheter. Et element kan ha flere innganger og utganger. Dette konfigureres inne i selve elementet eller man kan motta en konfigurasjonsstreng utenfra. Elementet nedenfor er en enkel ARP-responder som svarer på ARP anrop med adressen som er lagt inn som konfigurasjon.



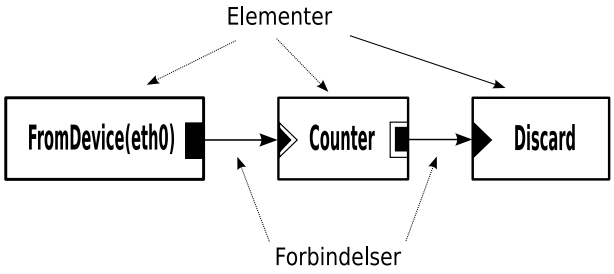
Figur 2.2: Et Click element

2.1.3 Ruterkonfigurasjon

En ruterkonfigurasjon består av elementer satt sammen i system ved hjelp av et click-script der pakker flyter fra element til element, der hvert element har en eller flere innganger og utganger. Elementenes innhold bestemmer hvordan pakkeflyten skal være. Oppdelingen i små atskilte elementer gir muligheten til lett å konfigurere ruteren eller skrive nye elementklasser.

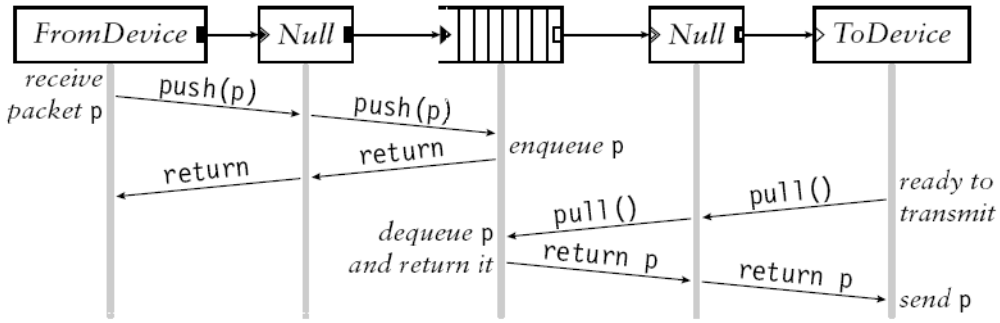
Et enkelt click-script kan se ut som dette:

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



Figur 2.3: Visualisering av enkelt click-script med en enkel prosess. Pakken kommer fra grensesnittet (eth0), blir telt og så forkastet.

Click benytter seg av to typer porter, push og pull. Push-portene er representert ved hjelp av en fyllt trekant eller firkant og pull-portene er åpne trekanter eller firkanter. Man kan også ha såkalte agnostiske porter som kan representere begge typer. En push-port må ha en pull-port i andre enden av forbindelsen.



Figur 2.4: Push og pull porter i en Click-ruter [14]

2.1.4 Kodingsprinsipper

Click tilbyr allerede i utgangspunktet et rikt utvalg av elementer. Men siden dette konseptet er velegnet for såkalte eksperimentelle rutere ligger det i sakens natur at man ofte er avhengig av å lage sine egne spesialtilpassede elementer for å kunne få utført sine forsøk. Click er implementert ved hjelp av et begrenset C++ språk. Hvis ruteren skal kjøres i kjernemodus er det forbudt å implementere klasser fra standard template libraries (STL). Click tilbyr derfor sin egen versjon av ofte benyttede STL klasser. Dette gjør at kodingen kan være en utfordring da ikke alt er like godt dokumentert, men "Clickmiljøet" har en mailingliste der man kan få hjelp. Elementene må inneholde noen faste metoder og være bygget opp på en spesiell måte. Ved pakkemanipulering

må man også benytte egne click-metoder, men ellers er det stort sett fritt frem å kode på vanlig måte, se listingene under.

Headerfil for et enkelt pushelement:

```
#ifndef CLICK_SIMPLEPUSHELEMENT_HH
#define CLICK_SIMPLEPUSHELEMENT_HH

#include <click/element.hh>

CLICK_DECLS
class SimplePushElement : public Element {
   public:
      SimplePushElement();
      ~SimplePushElement();
      const char *class_name() const { return "SimplePushElement"; }
      const char *processing() const { return PUSH; }
      int configure(Vector<String>&, ErrorHandler*);
      void push(int, Packet *);
   private:
      uint32_t maxSize;
};
CLICK_ENDDECLS
#endif
```

Implementasjonsfil for et enkelt pushelement:

```
#include <click/config.h>
#include <click/confparse.hh>
#include <click/error.hh>
#include "simplepushelement.hh"

CLICK_DECLS
SimplePushElement::SimplePushElement(): Element(1, 1){}
SimplePushElement::~ SimplePushElement(){}

int SimplePushElement ::configure(Vector<String> &conf, ErrorHandler *errh) {
   if (cp_va_parse(conf, this, errh, cpKeywords, "MAX_SIZE", cpInteger, "maximum packet
size", &maxSize, cpEnd) < 0) return -1;
   if (maxSize <= 0) return errh->error("maxsize should be larger than 0");
   return 0;
}

void SimplePushElement::push(int, Packet *p){
   if (p->length() > maxSize)  p->kill();
   else output(0).push(p);
}

CLICK_ENDDECLS
EXPORT_ELEMENT(SimplePushElement)
```

2.2 Hping

Hping [2] er en gratis pakkegenerator og analysator for TCP/IP distribuert av Salvatore Sanfilippo. Hping brukes ofte til sikkerhetstesting av brannmurer og nettverk. Den nyeste versjonen av hping, hping3, kan benytte Tcl script. Her kan man bygge opp pakker og pakkesekvenser ved hjelp av et stringbasert lett lesbart språk som gjør "low level" TCP/IP pakkemanipulasjon til en etter forholdene enkel affære. Hping er et godt verktøy som blant annet kan brukes av administratorer (og andre) for avsløre svakheter i sikkerheten.

2.3 Network Animator

Network Animator [4] (Nam) er et Tcl/TK basert animasjonsverktøy for animering av nettverk, både testnettverk og "live". Nam har støtte for topologianimering, pakkeanimering og flere andre verktøy for dataanalyse. Nam brukes ofte sammen med Network Simulator (ns).

2.4 OpenWRT

OpenWRT [5] er beskrevet som Linux distribusjonen for lukkede (embeddede) enheter. I stedet for å lage en enkel statisk firmware tilbyr OpenWRT et fullstendig overskrivbart filsystem med pakkestyring. Dette frigjør deg fra applikasjonsutvalget og konfigurasjonen til leverandøren, og du vil være i stand til både å legge til filer samt bruke andre applikasjoner enn det leverandøren tilbyr. OpenWRT er rammeverket for å kunne bygge en applikasjon uten å måtte bygge en komplett ny firmware, eksempelvis for små hjemmerutere. OpenWRT er også en komplett Linux ruterinstallasjon som har alle vanlige konfigurasjonsmuligheter for slike rutere. For brukerne betyr dette et enkelt og fleksibelt instrument for bruk under utvikling og testing.

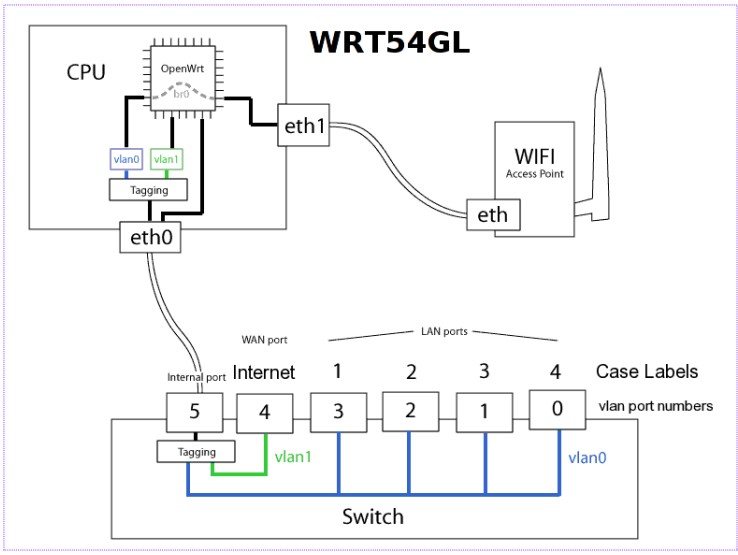
Utfordringen med installering av Linux på såkalte embeddede enheter slik som WRT54GL er at man ikke har muligheten for å kunne kompilere en ny firmware i selve enheten. For å kunne produsere en ny firmware må man derfor bygge opp firmware ved hjelp av en prosess kalt krysskompilering som i sin helhet foregår utenfor den aktuelle enheten, her WRT54GL. Denne prosessen med å bygge opp et slikt krysskompileringsmiljø er ikke helt enkel og derfor er flashing av allerede ferdigkompilert firmware, det vil si et image, den enkleste måten å installere Linux i en slik enhet på. OpenWRT tilbyr ferdig kompilerte imager til mange forskjellige rutere.

2.5 Linksys WRT54GL ruter

Figur 2.5: Linksys WRT54GL ruter [5]

I oppgaven ble ruteren Linksys WRT54GL [3] benyttet, se figur 2.5. Original firmware ble erstattet med OpenWRT [5] Linux White Russian RC5. Linksys WRT54GL består av en Ethernet svitsj, et trådløst aksesspunkt og en ruter som knytter disse sammen, se figur 2.6.

Enheten bruker Linux som operativsystem. Dette gjør at den lett kan flashes med uoriginal firmware, i dette tilfellet OpenWRT Linux. Den er egentlig laget for å dele en høyhastighets datalinje eller internettforbindelse. Ruteren har 5 tilkoblingsmuligheter i form av ethernetutganger på baksiden: En WAN-tilkobling som kobles til en DSL kabel eller annen høyhastighets dataforbindelse samt 4 LAN-utganger som på innsiden er tilknyttet en svitsj. WRT54GL har 4MB flash minne og 16MB RAM.



Figur 2.6: Linksys WRT54GL ruter skjematisk oversikt [5]

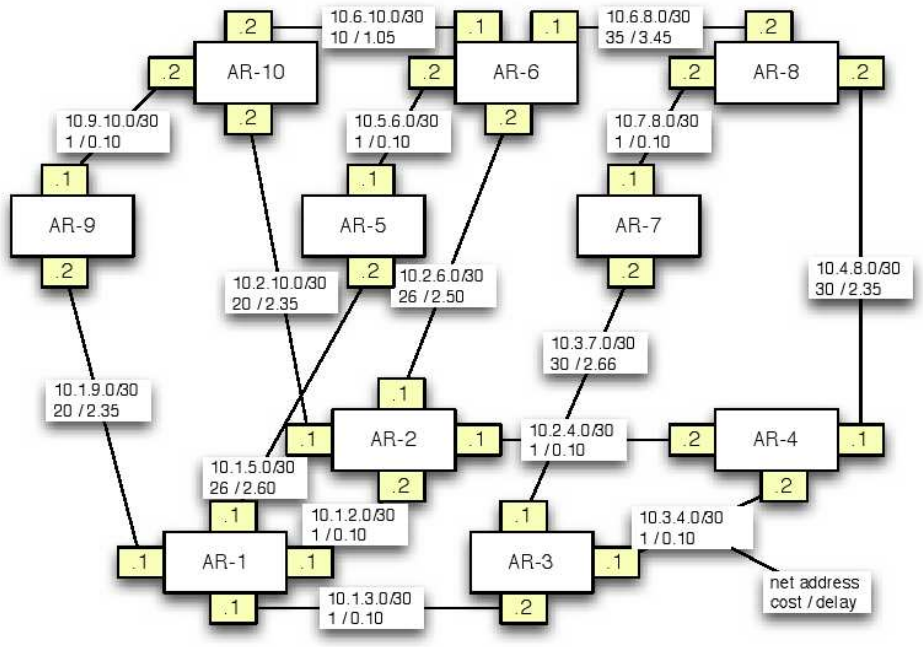
Kapittel 3

Implementering

3.1 Valg av design

Forarbeidet til valg av design ble gjort i den tidligere nevnte prosjektoppgaven [15] som var forløperen til denne masteroppgaven. Oppgaven gikk ut på å utrede en protokoll som oppdaget nye linker og som automatisk registrerte dem i nettverket. AntPing er realisert i Click, se kapittel 2.1, og mye av problemstillingen kokte etter hvert ned til: Hvordan realisere dette i Click? Eller mer detaljert: Hvordan dynamisk kunne endre eller tildele IP-adresser i Click under kjøring? Utfordringen er at Click er realisert ved hjelp av et script som leses inn ved oppstart og dermed er konfigurasjonen (og IP-adressene) statisk etter T0. Det finnes muligheter for å bytte ut script underveis mens Click kjører, men da må man ha mekanismer utenfor Click som kan styre dette. Dette er sikkert mulig men jeg vurderte det til utenfor mulighetsrommet å gjennomføre dette i løpet av en en prosjekt/masteroppgave. Derfor ble alle løsninger som foreslo dynamisk endring eller tildeling av IP-adresse i ruterne forkastet av designmessige årsaker.

Jeg vil derfor bare kort gå gjennom de vurderte løsningene.



Figur 3.1: Fysisk topologi og adressering i AntPing [12]

3.1.1 Vurderte løsninger

1. Statisk ruting - subnettadressering

   AntPing består i dag av et sett noder som er forhåndsprogrammert med navn, statiske IP-adresser og statiske rutingveier. Disse er oppbygd ved hjelp av faste regler som man bruker når man konfigurerer opp IP-adressen på grensesnittene og subnettet mellom to grensesnitt, se figur 3.1. Den tildelte IP-adressen utgår av ruternumrene på begge sider av linken. Hvis vi ser på linken mellom AR-1 og AR-5 vil det her lages et subnett med adresse 10.1.5.0/30, og adressen til AR-1 blir 10.1.5.1 og adressen til AR-5 blir 10.1.5.2.

   For å videreføre denne løsningen i et dynamisk miljø der linker legges til og flyttes kreves det at man er istand til å endre IP-adressene på grensesnittene mens Click kjører. Denne løsningen ble derfor forkastet av de grunner som er forklart over, se kapittel 3.1.

2. Random adresse med rutingprotokoll

   Her går man ut fra konseptet med bruk av autogenererte Link-lokal adresser, se tillegg B. Link-lokal adresser har i denne sammenheng den svakheten at de ikke er rutbare, det vil si at de kun kan brukes innenfor en link. For at denne protokollen skal være mulig å bruke må den utvides og forandres noe. For det første må man skifte ut Link-lokal adressene med

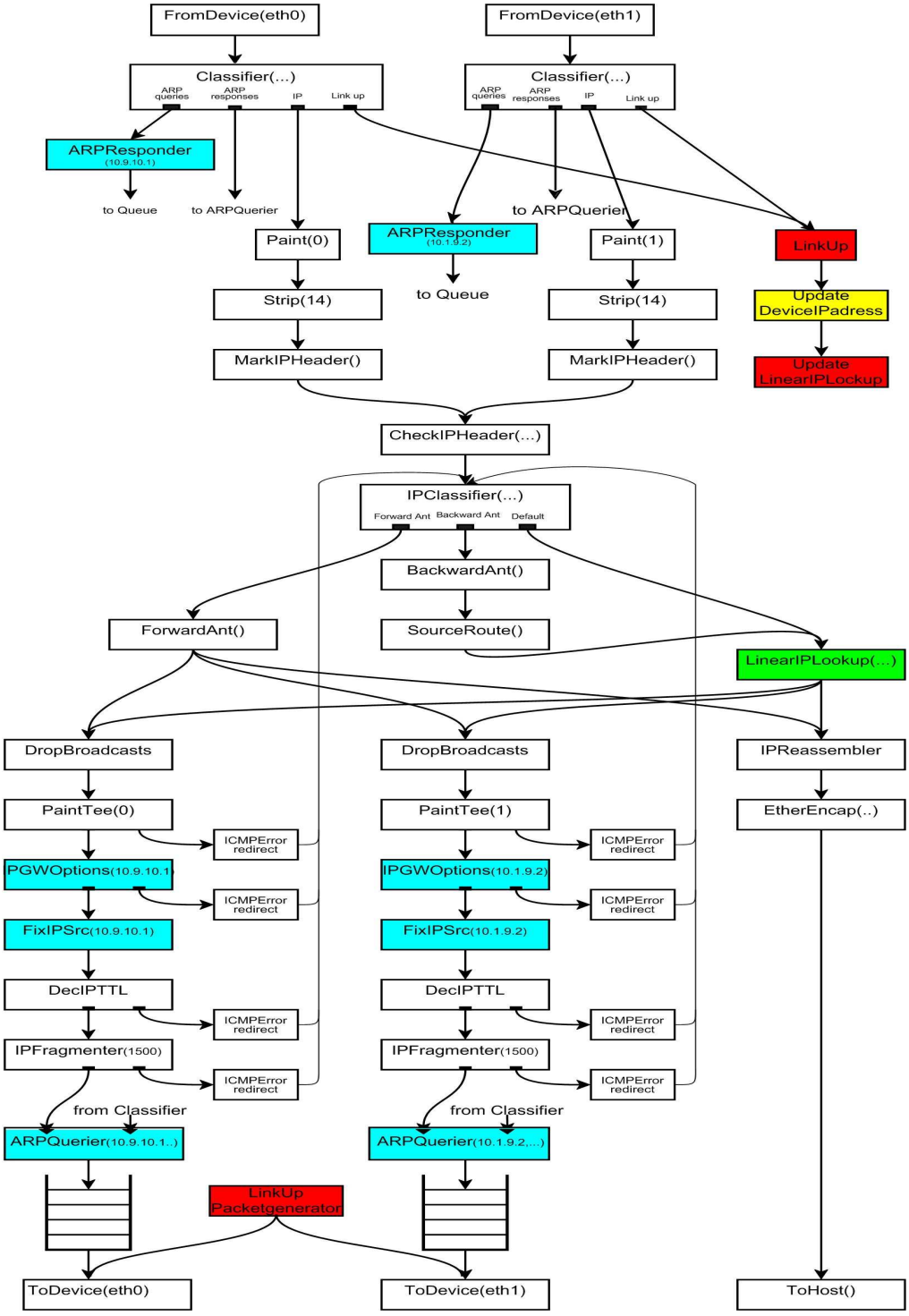
rutbare adresser. Her kan man velge å bruke vanlig lokalnettadresser, det vil si 10.x.x.x. For å kunne fastslå at den autogenererte adressen er unik må en i tillegg bruke en eller annen DAD (duplicat adress detection) protokoll. Denne løsningen ble også forkastet av samme grunn som Statisk ruting - subnettadressering, vanskelig å implementere ved hjelp av Click. Hvis det var mulig å implementere ville denne løsningen vært en meget fleksibel og god løsning som ville gjort ruterne enda mer autonom og selvkonfigurerende.

3. DHCP konfigurering

   I den tredje løsningen ser jeg for meg at man kan konfigurere opp den første ruteren i nettet som en DHCP server. Dette forutsetter at det er plass i RAM til å implementere dette i ruteren, noe jeg skulle anta at det er. Løsningen har flere fordeler, det en velkjent løsning som man vet fungerer og man har full kontroll over tildelte adresser. Løsningen er skalerbar og fleksibel. Løsningen ble også forkastet av samme grunn som de to første, vanskelig å implementere i Click. I tillegg vil den bryte med prinsippet om mest mulig autonomitet.

4. Statisk ruting - nodeadressering

   Denne løsningen er forholdsvis lik Statisk ruting - subnettadressering. Man går ut fra et AntPing-konsept som er bygd opp på nesten samme måte som tidligere. Ruterne er nummerert men adressene på grensesnittene er laget ut fra dette nummeret. I motsetning til løsning 1, der man lager subnettverk med to grensesnitt som har adresse utgått fra begge ruternavn, gir man her grensesnittene adresse i forhold til sitt eget ruternummer og grensesnittnummer. Feks vil ruter AR-25 få adressene 10.1.25.1 til 10.1.25.4. Denne løsningen er skalerbar da man har stort adressespenn å ta av, man kan adressere fra 10.1.1. 1-4 til 10.255.255.1-4. Ergo vil man kunne adressere 255x255 = 65025 noder som skulle være tilstrekkelig for de fleste formål. Her kan også grensesnittene være konfigurert med IP-adressene på forhånd. Dette er den foretrukne løsning og sannsynligvis den eneste implementerbare av de foreslåtte, særlig hvis man tar hensyn til at implementasjonen skal være mulig å gjennomføre i løpet av et semester.



Figur 3.2: Flytdiagram av foreslått løsning fra prosjektoppgaven [15]

3.2 Implementert løsning

Implementert løsning følger i prinsippet løsning nr 4 *Statisk ruting - nodeadressering*. I prosjektoppgaven skisserte jeg et Click-flytdiagram (figur 3.3) som beskrev i elements form den valgte løsning. Den implementerte løsning avviker imidlertid noe fra dette flytdiagrammet. Avviket mellom tidligere anbefalte løsning og den implementerte kan synes stort, men implementert løsning følger som tidligere nevnt den samme tankegangen. Forskjellen ligger i følgende: I den løsningen som ble skissert foreslo jeg å bruke pakkegeneratorer for å sende pakker til alle ethernetporter i ruteren. Disse ville bli detektert i motsatte enden av en link og dermed brukes til å angi at linken var oppe for så å oppdatere "rutingtabellen". Når pakkene uteble over et visst tidsrom ble linken regnet som nede. Jeg har gått bort fra denne metoden og istedet brukt forward- og backwardant til å detektere om linken er oppe og til å oppdatere "rutingtabellen". Grunnen til endringen i forhold til foreslått implementasjon er ønsket om ikke å belaste systemet med mer kontrolltrafikk enn nødvendig. Svakheten er at man sannsynligvis vil bruke noe lengre tid på å oppdage nye linker eller linker som er nede.

3.2.1 Pakkeflyt i AntPing click

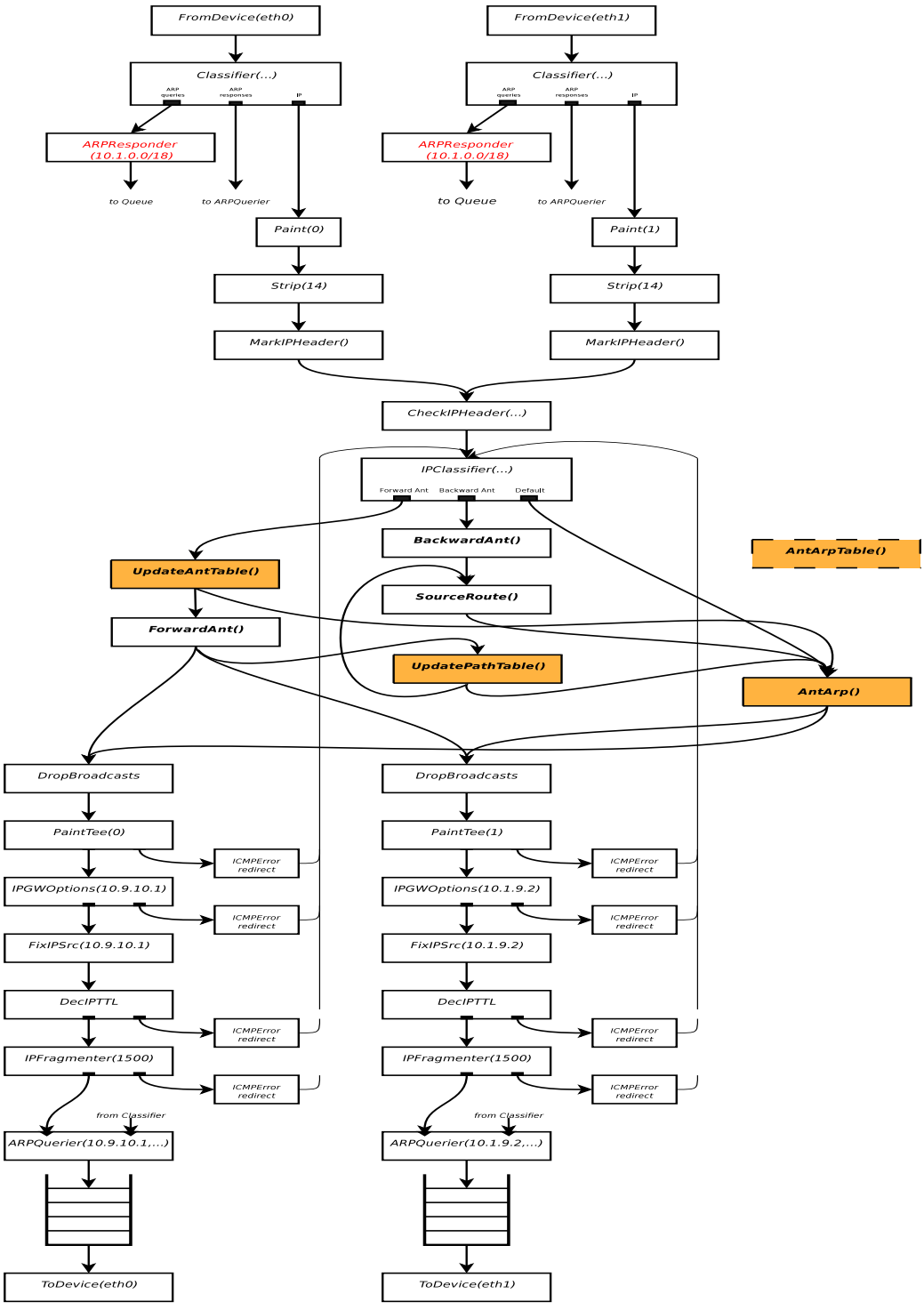
Figur 3.3 viser pakkeflyten i en ruter med 2 utganger. Jeg vil gå gjennom denne og legge vekt på de elementer som jeg har laget i forbindelse med denne masteroppgaven samt de elementer som har betydning for denne konfigurasjonen. Elementene som jeg har implementert har jeg uthevet med farge og blir beskrevet mer detaljert senere i dette kapitlet. Jeg vil ikke legge så mye vekt på beskrivelse av funksjonen til standardelementer i min beskrivelse av pakkeflyten. Konfigurasjonen er kun istand til å rute IP-pakker som enten er forward- eller backwardant samt andre LSR (Loose Source Routing) IP-pakker.

Pakken kommer inn fra en ethernetport og blir deretter klassifisert i en Classifier der ARP-forespørselen blir skilt ut og behandlet for seg. Her har jeg endret litt og omgjort ARP responsen til såkalt Proxy-ARP response (10.1.0.0/18). Dette har jeg gjort for at man fleksibelt skal kunne koble senderen til på alle vlanporter til alle rutere i systemet uten å måtte endre på konfigurasjonen i PC-en. Pakken blir deretter markert med en såkalt Paint som er entydig for den enkelte ethernetport. Dette gjøres for senere å kunne skille hvilken ethernetport på ruteren som pakken kom inn på. Via Strip, MarkIPHeader og CheckIPheader går pakken til IPClassifier. Her skiller man pakkene på UDP/TCP portnummer, dermed kan man skille ut forward- og backwardantpakker som har UDP senderport hhv 50000 og 51234, se tabell 3.2.

- Forwardantpakker går til UpdateAntArp. Oppgaven til dette elementet er å oppdatere AntArpTable, som er en slags rutingtabell, og å sende pakker tilbake til senderen for å animere linkopp/linkned. Pakken går deretter til elementet ForwardAnt. Dette elementet har som oppgave å oppdatere datalasten med kostverdier for utgående link, samt å velge veier ut for

forwardantpakkene i henhold til til søkealgoritmen beskrevet i AntPing-dokumentasjonen. Hvis pakken derimot har denne ruteren som endelig destinasjon blir den skilt ut i ForwardAnt og sendt til UpdatePathTable. Dette elementet erstatter den funksjonen som tidligere ble utført i mottaker-PC'en (Ant Rec) ved hjelp av hping og et TCL script. Her blir nye stier detektert og man lager et nytt objekt for denne stien som inneholder en teller og temperaturvariabler. Pakken blir deretter omgjort til en backwardant (det vil si loose source ruting-pakke) for tilbakesending til sender og adressert til neste hopp. Deretter blir datalasten lagt til. Pakken går så til AntArp som slår opp i AntArpTable for å kunne rute pakken til rett ethernetutgang. Pakken følger deretter flyten til den utgangen som er valgt.

- Backwardantpakker går fra IPClassifier til BackwardAnt. Hvis pakken er en eliteant blir stien (eller maurartens) ferromonverdier oppdatert i dette elementet, hvis ikke passerer den bare gjennom. Videre går pakken til SourceRouting, her blir neste hopp hentet ut av LSR feltet i datapakken og brukt til å endre IP-mottakeradressen. Pakken går deretter til AntArp som foruten å rute pakken til riktig utgang også henter ut data i pakken for å oppdatere AntArpTable. Pakken følger deretter samme vei som forwardant.



Figur 3.3: Click flytskjema av implementert løsning, se også tillegg Hmed tilhørende Click-script

3.3 Pakketyper i AntPing

CEAS bruker UDP pakker. Disse er normale UDP pakker som identifiseres ved hjelp av sender /mottakerport og av innholdet i datalasten. Som vi ser av tabell 3.2 har vi fire typer som er i bruk av selve stifinnersystemet CEAS mens tre typer er brukt til rene animasjonsformål. Forwardant har alltid senderport 50000 og mottakerport 51234. Backwardant snur på dette og bruker alltid senderport 51234 og mottakerport 50000. Animasjonspakkene bruker også samme formatet, disse er ikke en del av CEAS men en del av demonstratoren AntPing og brukes i produksjon av Nam tracefil og annen statistikk. Formatet og pakkene er noe endret i forhold til den originale demonstratoren. På backwardant har jeg lagt til en "!" på slutten av pakkedata. Forklaringen på dette finnes i kapittel 6. Jeg har i tillegg laget to nye pakketyper som betegnes type 6 og 7. Disse er rene animasjonspakker. Her har jeg, som forklart i kapittel 3.4, lagt til remote-IP på slutten av datalasten. Dette er IP-adressen til motsatt ende av linken som går opp/ned, denne må være med for at Nam skal kunne detektere korrekte linkopp/ned-meldinger.

Type	Sender-port	Mottaker-port	Datalast	Beskrivelse
0	50000	51234	$0/cost1/cost2/.../costN$	forward normal (maintenance) ant
1	50000	51234	$1/cost1/cost2/.../costN$	forward exploration ant
2	50000	51234	$2/\beta/\rho/cost1.../costN$	forward initiell exploration ant
0	51234	50000	$0/e^{-L(\pi)/\gamma}$/L(π)/γ/β/NAMtime/cost1/cost2/.../costN!	backward oppdaterer ferromonverdien i alle rutere tilhørende denne stien
1	51234	50000	$1/e^{-L(\pi)/\gamma}$/L(π)/γ/β/NAMtime/cost1/cost2/.../costN!	backward brukes til animasjonsformål viser ant drop ved mål
6	51234	50000	6/cost1/$cost2/.../costN/remoteIP$!	backward brukes til animasjonsformål og gir melding om linkned
7	51234	50000	$7/cost1/cost2/.../costN/remoteIP$!	backward brukes til animasjonsformål og gir melding om linkopp

Tabell 3.2: Pakketyper og innhold

3.4 Oppdatering av AntArpTable

Oppdatering av "rutingtabellen" kalt AntArpTable er en sentral funksjon i denne implementasjonen. Det er denne tabellen og oppdateringen av denne som muliggjør at demonstratoren beveger seg fra å være statisk med forhåndsdefinert topologi, til en mer dynamisk utgave der man under

kjøring lett kan legge til og fjerne linker. Dette gjøres enkelt bare ved å plugge inn og ut kabler og deretter oppdaterer systemet seg automatisk.

Figur 3.4 viser hva som skjer i tabellen når pakker flyter gjennom systemet.

Jeg skal her beskrive hva som initielt skjer når man begynner å sende pakker gjennom systemet. For å kunne beskrive dette har jeg vist hva som skjer i AntArpTable elementet, nærmere bestemt GWtable. Det er denne tabellen som AntArp bruker til ruting. For å få fullt utbytte av forklaringen nedenfor bør man forstå "route record" og "loose source routing", se tillegg C. Tallene nedenfor refererer til tallene i figuren. Denne figuren viser forløpet til en forward/backwardant som går frem og tilbake mellom kilden K og målet M, samtidig vises innholdet i R2 "rutingtabellen" GWtable.

1. Pakken er sendt fra K og har M som mål. Dette er en forwardant, og i optionfeltet står det route record. Pakken har passert R1 men er ennå ikke kommet frem til R2. Vi ser at R1 (egentlig IP-adressen til R1/Eth2) er lagt inn i optionfeltet. Rutingtabellen til R2 er helt tom.

2. Når pakken kommer inn i R2 vil rutingtabellen bli oppdatert av UpdateAntArp elementet. Vi ser at R1 er lagt til Eth0 posisjonen i tabellen. Dette foregår på den måten at adressen som pekeren i optionfeltet peker på er den som blir lagt til i tabellen. Hvis pekeren har verdien 4 vil det si at det ennå ikke er blitt lagt til noen IP-adresse i optionfeltet. Da vil adressen som står i kildefeltet bli lagt til i tabellen. Etter at dette er gjort sender det samme elementet en pakke tilbake til kilden for animasjonsformål (type 7). Denne pakken varsler om at det er lagt til en link mellom R2 og R1. Vi ser av datalasten til pakken at det er lagt til "/R1". Mottaker av denne pakken vil da kunne lese at det er opprettet en ny link mellom avsender (R2) av animeringspakken og den adressen som er lagt til bakerst i pakkelasten (R1). Dette kan synes unødvendig da man kan hente ut den samme opplysningen i optionfeltet, men for linknedmeldinga (type 6) vil ikke en slik info finnes i optionfeltet, og for å gjøre det likt og enkelt blir adressen lagt til på slutten av datalasten.

3. Her er pakken kommet ut av R2, og R2 er lagt til i optionfeltet. Det er ingen endring i tabellen.

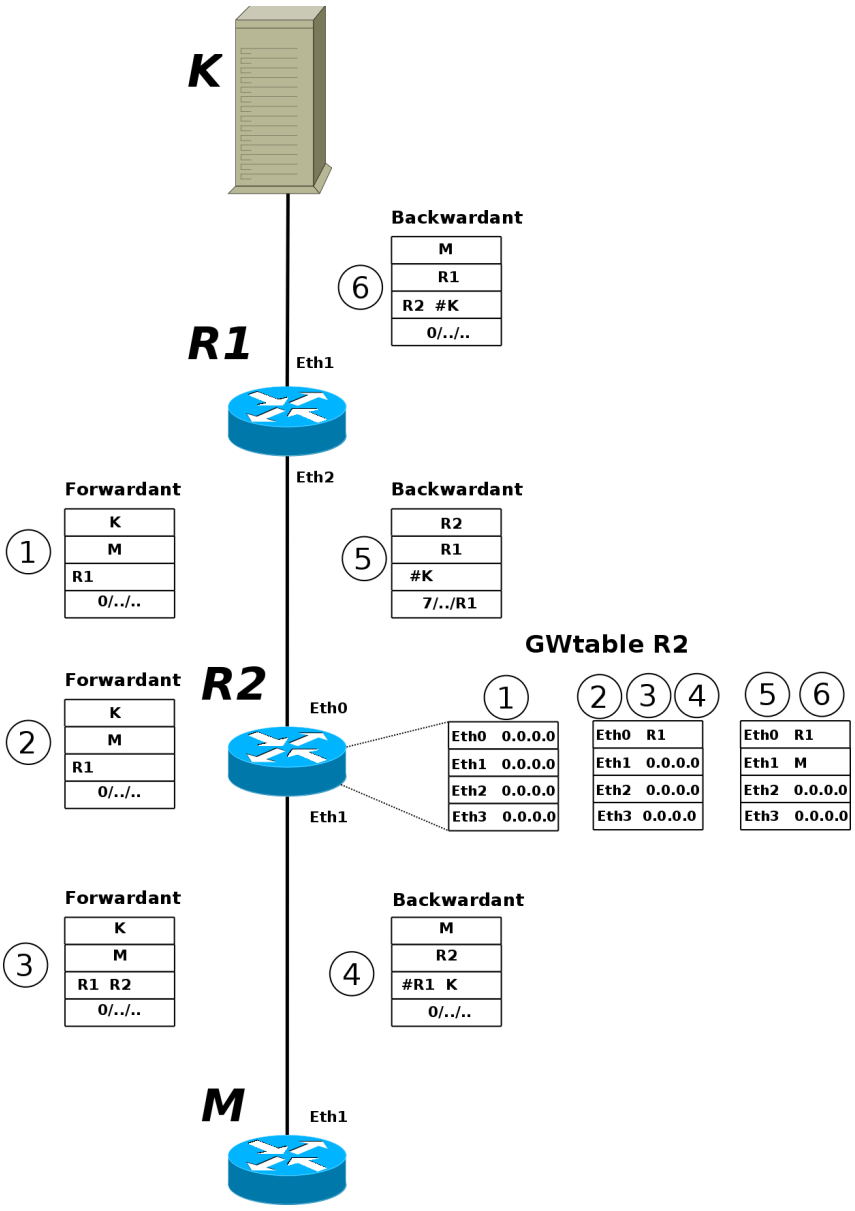
4. Pakken har vært innom ruteren M som er målet. Vi har snudd optionfeltet, regnet ut og lagt til temperaturdata i datafeltet. Pakken er blitt en backwardant og det står loose source routing i optionfeltet. Når pakken kommer inn i ruteren R2 vil adressen som står i målfeltet (her R2) og det som pekeren (#) i optionfeltet peker på bytte plass, og pekeren vil peke på neste adresse i lista.

5. Slik vil pakken se ut når den ankommer AntArp elementet i R2. Her vil posisjonen til pekeren bli sjekket. Hvis pekeren peker på den tredje adressen (som ikke finnes her), vil man gå 2 plasser tilbake, lese adressen og legge den i tabellen på plassen til det grensesnittet

den kom inn på (her Eth1). Hvis pekeren peker på posisjon 2 vil det som står i kildefeltet bli lagt inn i tabellen. Hvis den står i posisjon 1 vil tabellen ikke bli oppdatert.

6. Her har pakken vært gjennom R2, og neste er K. Tabellen er oppdatert.

Som vi ser er man avhengig av at pakken går frem og tilbake på stien for at tabellen skal bli komplett oppdatert. Forklaringen på hvordan linkned håndteres blir forklart under virkemåten til UpdateAntArp, se kapittel 3.5.3.



Figur 3.4: Oppdatering av rutingtabell. Innhold i forward-/backwardant-pakken og GWtable i tidspunkt 1 til 6.

3.5 Elementene

Her vil jeg gi en litt mer grundig beskrivelse av de elementer som er laget spesielt for denne masteroppgaven.

3.5.1 AntArp

AntArp er erstatningen for rutingtabellen. Den inneholder ikke selve tabellen men kaller metoder i AntArpTable. Man har ikke en rutingtabell i tradisjonell forstand, men kun en tabell som holder oversikt over linkene som er oppe og hvilken IP-adresse som da befinner seg i andre enden av disse. Man kan derfor se på denne tabellen som en slags ARP-tabell, derfor navnet AntArp. Elementet er kun i stand til å rute backwardant pakker eller andre pakker med LSR. Forwardant pakker blir rutet i ForwardAnt elementet. Når pakken ankommer AntArp har den allerede vært gjennom SourceRoute elementet, og neste adresse i option feltet ligger allerede som måladresse.

AntArp har to oppgaver:

- **Rute backwardant og andre LSR pakker**. Når LSR IP-pakker ankommer AntArp vil neste hopp være satt som måladresse, dette gjør rutingen enkel. Man slår opp på måladressen i datagrammet, sjekker dette mot AntArpTable og ruter denne til rett utgang.
- **Oppdatere AntArpTable når en backwardant eller andre LSR pakker går gjennom elementet.** Dette gjøres ved å lese i optionfeltet. Her er det en peker som peker på en av adressene i optionfeltet. Når pakken ankommer AntArp vil pekeren alltid peke på hoppet etter det som nå står som måladresse. Leser man en plass tilbake vil den peke på nåværende adresse (det grensesnittet den kom inn på), leser man to plasser tilbake vil man da finne adressen til andre enden av den linken som pakken kom inn på. Hvis pekeren peker på plass nr en i lista befinner pakken seg ennå på kildeadressen og ingen oppdatering av AntArpTable skal foretas. Hvis pekeren peker på andre plass i lista er det kildeadressen som befinner seg på andre enden av linken og vi oppdaterer tabellen med denne. Hvilken ethernetport denne linken er knyttet til finner man ut ved å sjekke verdien som Paint elementet tilførte pakken.

3.5.2 AntArpTable

Dette elementet inneholder en samling tabeller og nødvendige metoder for å manipulere disse.

- **GWtable:** Denne inneholder oversikten over hva som befinner seg på andre enden av en link. Hvis linken er nede inneholder oppslaget 0.0.0.0.

- **vlanTable:** Denne inneholder oversikten over hvilken IP-adresse som er satt til hvilket grensesnitt.
- **pathVector:** Denne vektoren inneholder temperaturdata. Disse verdier er rekursivt regnet ut av data fra alle tidligere pakker.
- **pathVectorElite:** Denne vektoren inneholder de samme data som pathVector, men kun for eliteant.

3.5.3 UpdateAntArp

Her kommer det bare forwardantpakker, og elementet har to oppgaver:

- **Oppdatere AntArp når en forwardant pakke kommer inn.** En forwardant er i route record modus. Det vil si at for hvert grensesnitt en slik pakke rutes ut gjennom registreres dette grensesnittets IP-adresse i optionfeltet. I optionfeltet har man en peker. Når pakken ankommer UpdateAntArp vil pekeren alltid peke på det tomme feltet som neste adresse skal legges inn i. IP-adressen til motsatt ende av linken ligger en plass tilbake. Hvis pekeren peker på første feltet i adresselista betyr det at lista er tom og at adressen på andre enden av linken er kildeadressen. Ut fra disse opplysninger kan AntArpTable oppdateres med hva som befinner seg på motsatt ende av linken.
- **Elementet gir tilbakemeldinger når en link går opp eller ned.** Dette er til bruk når Nam skal animere linkopp/ned. Når en link går opp vil man returnere en pakke av type 7, og når den går ned vil man returnere type 6. Når en slik pakke skal sendes tilbake til mottaker lager man først en kopi av den mottatte pakken. Denne gjøres så om fra RR til LSR, deretter blir det på enden av datalasten lagt til IP-adressen til motstående grensesnitt. Dette gjør det lett å plukke ut animasjonsdata når pakken har returnert til kilden. Det viste seg under test at det var nødvendig å sende de samme animasjonsdataene, det vil si linkopp og linkned, flere ganger for å være sikker på at disse når frem. Dette fordi ARPQuerier elementet i Click ser ut til å forkaste de første pakkene som skal ut på grensesnittet. Måten man detekterer linkopp på er tidligere forklart under kapittel 3.4, men linkned gjøres noe annerledes: Hver gang en pakke ankommer updateAntTable gjør man en sjekk og leser fra driveren om det finnes bærebølge på ethernetportene (det vil si at ethernetportene gir signal om en link er oppe eller ikke). Hvis linken er nede så sjekker man mot rutingtabellen (AntArpTable) om det tidligere er lagt inn en adresse der. Hvis så er tilfellet sender man en linkned pakke (type 6) tilbake til kilden. IP-adressen som sto i tabellen vil bli lagt til bakerst i datalasten tilsvarende slik det ble gjort ved linkoppmeldingen. Deretter blir innslaget i tabellen fjernet. Man beholder statusendringen ved hjelp av en tabell, og man sender en slik linkopp/ned melding 3 ganger for å være sikker på at animasjonsdata når frem.

3.5.4 UpdatePathTable

Dette elementet er erstatning for mottaker Pc'en (AntRec) i AntPing leveransen. Her tar man imot alle forwardant som er adressert til et av grensesnittene på ruteren. Dette gjør at alle ruterne kan fungere som mottakere, noe som gjør systemet mye mer fleksibelt og transporterbart enn tidligere utgaver av demonstratoren.

Her blir nye forbindelser/stier detektert, og for hver ny sti blir det laget to nye objekt.

I det ene objektet vil man rekursivt regne ut og lagre temperaturvariablene til alle forwardant pakkene. I det andre objektet vil den samme operasjonen foregå, men bare på de pakker som skal returneres som backwardant type 0, det vil si eliteant. Hvordan denne seleksjonen foregår er nærmere beskrevet i [12].

Pakken blir deretter omgjort til en backwardant (det vil si loose source ruting-pakke) for tilbakesending til sender og adressert til neste hopp. Deretter blir datalasten lagt til. Denne datalasten inneholder de verdier som er nødvendig for å oppdatere ferromonverdiene til de enkelte ruterne i stien.

Kapittel 4

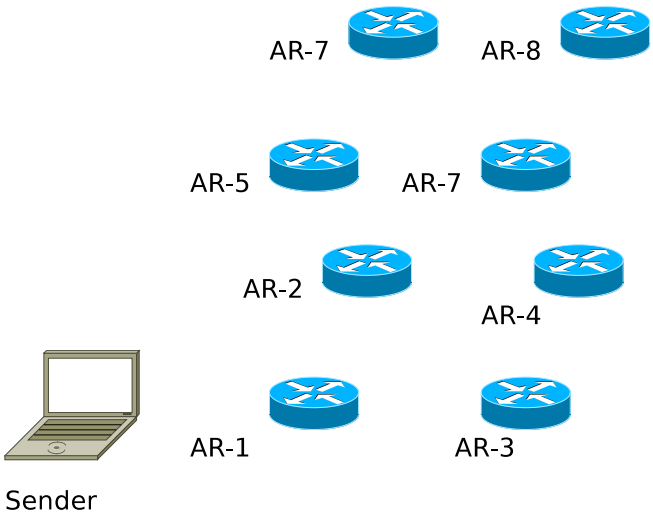
Test av implementerte funksjoner

4.1 Oppsett

Til testoppsett har jeg brukt følgende utstyr:

- 1 laptop PC med Ubuntu Linux 6.06. Denne er brukt til konfigurasjon av rutere og kjøring av hping3 (pakkegenerering samt til mottak av animering og måledata).
- 8 Linksys WRT54GL flashet med Openwrt RC5 Linux.
- Mottakeren av pakkene er implementert i alle ruterne.

Jeg har ikke tegnet opp linker mellom enhetene da dette er fullstendig åpent, og det er opp til hver enkelt tester å definere det enkelte scenario.



Figur 4.1: Testoppsett av "nye AntPing"

For å få en mest mulig realistisk demonstrasjon av CEAS konseptet ved hjelp av hjemmerutere i et labmiljø har man valgt å legge til faste forsinkelsesverdier for å emulere naturlig forsinkelse i ruterne/linkene. Disse verdien er lagt til utgående link i Click-ruterne. Verdiene er gjengitt i tabell 4.1.

	eth0	eth1	eth2	eth3
AR-1	26	1	1	20
AR-2	26	1	1	20
AR-3	1	1	1	1
AR-4	14	30	1	1
AR-5	1	30	1	26
AR-6	10	35	26	1
AR-7	1	1	30	3
AR-8	35	30	1	1

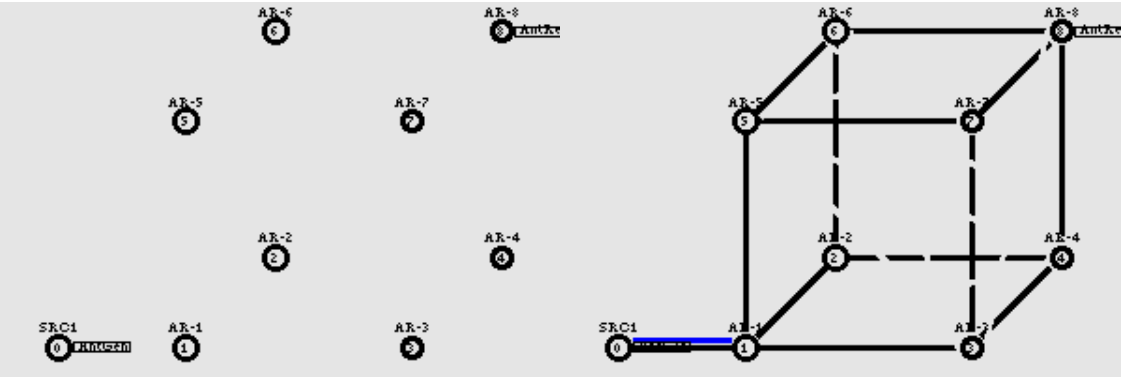
Tabell 4.1: Kostverdier tilknyttet utgående ethernetport

4.2 Testscenario

For å demonstrere at systemet virker etter hensikten vil jeg ved hjelp av screenshot og plott vise et forløp av et tilfeldig oppsatt og utført scenario. I tillegg har jeg i den vedlagte arkivfila lagt ved en video av Nam sekvensen, Nam tracefil og gnuplotfil slik at hele sekvensen kan gjenskapes. Testen ble utført på samme måte og med samme parameter som i tillegg G. Kommandoen som ble gitt i terminalvinduet så slik ut:

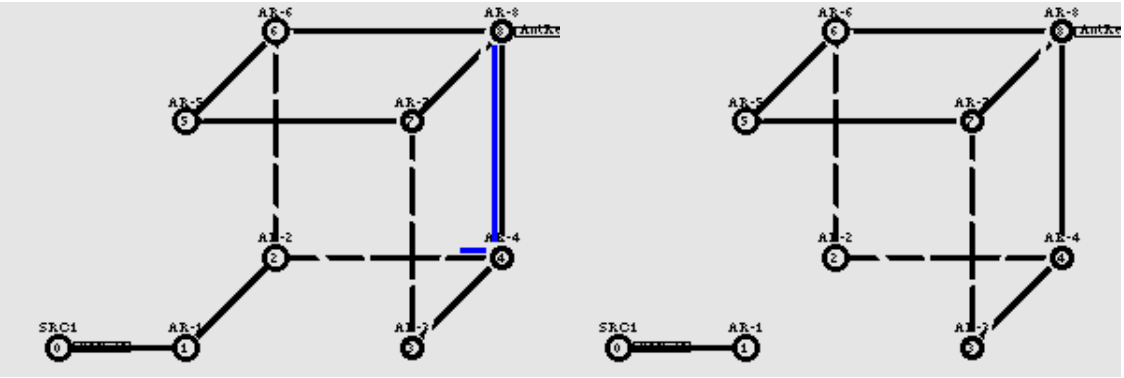
#hping3 exec tx_final.htcl 10.1.8.3 eth0 50000 0 1000 30 0.05 0.95 0.20

Dette vil sende forwardant til 10.1.8.3 på UDP-port 50000 (forwardant) på eth0, med 1000ms mellom hver pakke, og de 30 første er initielle explorerant med ρ=0.05 og β=0.95. Etter de 30 første initielle explorerant vil 20% av påfølgende forwardant være vanlige explorerant.



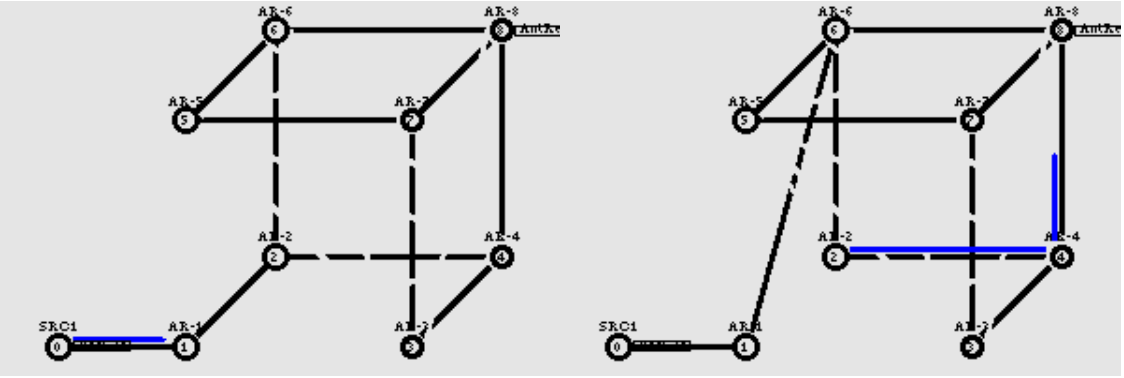
(a) Tidspunkt T0

(b) Tidspunkt T10



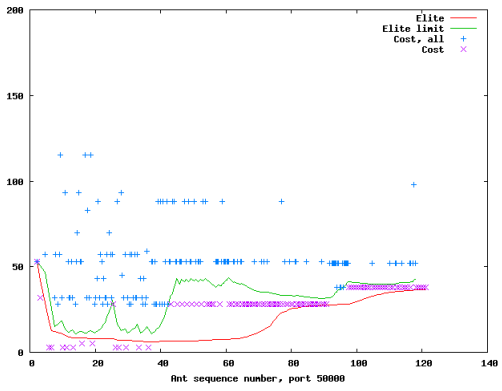
(c) Tidspunkt T34

(d) Tidspunkt T72



(e) Tidspunkt T82

(f) Tidspunkt T98



(g) Plott av hele animasjonen

Figur 4.2: Animasjon og plott av scenario - a) T0 ingen linker plugget inn b) T10 alle noder og linker er oppe c) T37 link 1-3 og link 1-5 fjernes d) T52link 1-2 fjernes e) T55 Link 1-2 legges til igjen f) T90 Link 1-2 flyttes til 1-6 g) Plott av kostverdier og eliteantlimit under testscenarioet

4.3 Kommentarer til testforløpet

Figur 4.2 a-f inneholder screenshot av en Nam animering av et enkelt testscenario. Det blir fjernet og lagt til linker. I tillegg kan vi i figur 4.2 (g) følge kostverdien til alle forwardant som blir sendt ut. De fiolette X viser kostverdien av de backwardant som blir returnert for å oppdatere ferromonverdien i ruterne, såkalte eliteant. De blå + viser kostverdien til alle de andre forwardant som egentlig blir droppet ved målet, men som i demonstratoren blir returnert for animasjon- og statistikkformål. Den grønne grafen viser verdien som eliteant blir selektert ut fra. Denne verdien blir regnet ut fra kostverdien til alle forwardant som når målet. Den røde grafen blir regnet ut på samme måte, men blir kun beregnet ut fra kostverdien til alle eliteant. Disse endrer seg over tid og tar hensyn til både historikk og den ferske kostverdien som den siste forwardant har registrert. Eliteseleksjonen som foregår her sørger for at bare de beste stiene blir vedlikeholdt. Mer utfyllende beskrivelse kan finnes her [12]. I dette testforløpet skal jeg imidlertid bare kommentere animeringen og testresultatet. Av plottet ser vi at vi ved ca T20-T25 har ingen forwardant som blir klassifisert som eliteant. Dette medfører at den grønne grafen (eliteantlimit) stiger i verdi og ved ca T25 klassifiserer den en forwardant med forholdsvis høy kostverdi som eliteant. Dette kommer av at β er satt forholdsvis lavt (0.95). Denne verdien sier noe om vektingen av kosthistorikk opp mot ferske kostverdier som den aktuelle forwardant har registrert. Testscenarioet har altså 95% vekting av historikk. Hvis denne verdien hadde vært høyere ville vektingen på historikk vært høyere, og denne forwardant ville ikke blitt klassifisert som eliteant. Her må man balansere hvor hurtig et nett skal reagere på eventuelle utfall av linker opp mot stabilitet. Hvis nettet reagerer for hurtig vil dette føre til ustabilitet. Vi ser videre at fra ca T37 når linkene 1-3 og 1-5 fjernes får vi et totalt fravær av eliteant. Eliteantlimit stiger helt til det ved ca T42 igjen blir klassifisert eliteant, altså man har igjen funnet gode veier i systemet. Ved T53-T55 fjernes link 1-2 og systemet deles i to. Animeringen registrerer linkned og linkopp, men beste vei har ikke endret seg i systemet og vi fortsetter rutingen som før. Fra T55 og utover får systemet gå uten endringer og vi får forholdsvis flere eliteant. Men siden vi har 20% explorerant vil det alltid være noen forwardant som søker i systemet etter eventuelle nye veier. Når vi har et stabilt system vil disse alltid gå lengre veier enn eliteant. Vi ser at den grønne og den røde grafen nærmer seg hverandre, dette tyder på at systemet har stabilisert seg. Ved ca T90 flytter vi linken 1-2 til 1-6. Dette gjør at vi på nytt må lete etter nye beste veier, og dette finner vi ved ca T98. I et slikt enkelt system som dette, med store avstander i kostverdi mellom beste og nestbeste vei, vil vektingen på beste vei bli stor. Men ved flere veier som hadde hatt tilnærmet lik kostverdi ville flere besteveier blitt funnet og vedlikeholdt. Som en delkonklusjon kan jeg si at testen var vellykket, og derfor kan jeg trekke den slutningen at de implementeringer jeg har gjort her har fungert etter hensikten.

Kapittel 5

Konklusjon

I denne oppgaven har hovedhensikten vært å implementere utvidet funksjonalitet i AntPing demonstratoren. Jeg har utvidet demonstratoren fra kun å kunne simulere et statisk oppsett med faste linker mellom ruterne, til å bli en dynamisk selvkonfigurerende demonstrator som kan legge til nye linker og noder etter behov, uten å måtte gjøre manuelle endringer underveis i demoen. I tillegg har jeg flyttet AntRec funksjonaliteten, som man før måtte bruke en separat PC til, inn i den enkelte ruter. I tillegg til fordelen med å slippe å bruke en egen PC til dette formålet, tilfører dette økt fleksibilitet med eksempelvis å kunne finne stier til flere mål samtidig uten å måtte sette opp ekstra hardware. Konklusjonen må være at målet synes å være oppnådd, og vel så det. Dette kan jeg si da den utvidede AntPing fungerer etter hensikten, og i tillegg har jeg tilført funksjonalitet utover det som var beskrevet i oppgaven. Implementasjonen har tilført demonstratoren en ekstra dimensjon ved å kunne vise en mer dynamisk visuell troverdig oppførsel foran et publikum.

Kapittel 6

Videre arbeider

Videre arbeider kan deles inn i 2 kategorier: 1)Ny funksjonalitet og 2) Forbedring av eksisterende kode.

1. Ny funksjonalitet.

   (a) AntPing bruker i dag fast kostverdi (delay) på linkene. Dette skal simulere den naturlige kostverdien som er i en link (inklusiv prosesseringstid og tid i kø). I et virkelig nettverk vil denne kostverdien variere avhengig av trafikkbelasting eller andre forhold (f.eks feil ved lavere lags utstyr). Da det ligger i konseptet at ruteoptimaliseringen skal være dynamisk og reagere på endringer i kostverdien, vil det være naturlig å se på en implementering av dynamiske kostverdier. Samtidig bør man få til en visualisering av kostverdien i Nam for å se hvordan denne forandrer seg og hvordan rutingen blir påvirket av dette. Kostverdien bør knyttes til link og ikke til utgang slik at man får lik verdi uavhengig av retningen til datapakken.

   (b) Siden Click kjører på OpenWRT Linux som en vanlig applikasjon kan man se for seg å utnytte dette til å kunne kjøre flere softwarerutere parallelt med Click. Dette kan være hensiktsmessig for å kunne sammenligne CEAS med andre rutingprotokoller. Her kan vi eksempelvis bruke XORP (eXtensible Open Router Platform) [8] som er en open source IP-ruter som kompilerer på det meste av plattformer[1]. I denne software ruteren er det allerede implementert flere kjente rutingprotokoller, f.eks RIP, OSPF, BGP. Begrensningen her kan være størrelsen på den ferdigkompilerte XORP binærfila kompilert til OpenWRT og prosessorkapasiteten til Linksys WRT54GL.

2. Demonstratoren opptrer nå stabilt, men enkelte deler av koden kunne helt sikkert vært mer strømlinjeformet. Jeg har hatt fokus på at ting skal virke men har ikke hatt mye

[1] The code should build and run on Linux 2.4.x, Linux 2.6.x, DragonFlyBSD, FreeBSD, NetBSD, OpenBSD, MacOS X, and Windows Server 2003. [8]

tid til å luke ut eventuelle “bugs” ved å stressteste systemet. En kode som det kan være verdt å ta en ny omgang på er UpdatePathTable. For det første foregår det her en del flyttallsberegninger, noe som kan være en feilkilde, i tillegg er prosessen rundt generering av ny pakke ikke helt tilfredsstillende utført. Når ny pakke blir generert følger noe av den gamle datalasten med på kjøpet. Datalasten i den nye pakken blir derfor større enn det den ideelt skal være. For å kunne skille den “nye” datalasten fra den “gamle” måtte jeg legge til en “!” på enden av den nye datalasten (eks 0/45/34/1/6!#%tjy her er 6!#%tjy “søppel”). Det skal være mulig å gjøre dette på en mer elegant måte enn det er gjort i dag. Man bør også lete etter “memory leaks” som også vil kunne føre til ustabilitet. Et eksempel på noe som kan skape memory leak er følgende: For å kunne manipulere datapakker blir det rundt om i koden generert såkalte “WritablePacket”. Dette er datapakker som ofte er en kopi av en innkommende pakke. Noen ganger skal den originale pakken eller innholdet i den ikke brukes mer og bør og kan fjernes, noe som sannsynligvis ikke alltid er gjort.

Tillegg A

Tcl

Tcl [7] ("Tool Command Language", uttales "tikl") er et scriptspråk laget av John Ousterhout. Scriptspråket er mest brukt for hurtig prototyping, scriptede applikasjoner, GUI'er og testing. Tcl kan kjøres på Unix, Windows og Macintosh plattformer.

Tcl har en grafisk verktøykasse kalt Tk, kombinasjonen av disse blir referert til som Tcl/Tk.

Tcl's hovedtrekk inkluderer:

- Alt er en kommando.
- Alt kan dynamisk overskrives og redefineres.
- Alle datatyper kan bli manipulert som string inkludert kode.
- Enkel syntaks.
- Hendelsesbasert grensesnitt mot socket og filer. Tidsbasert og brukerdefinerte hendelser er også mulig.
- Enkel unntaksbehandling bruker unntakskode returnert for alle kommandoutførelser.
- Plattformuavhengig.
- Nær integrasjon med GUI interfacet Tk.
- Lett å vedlikeholde kode. Tcl script er ofte mer kompakt og leservennlig enn funksjonelt ekvivalent kode i andre språk.

Tillegg B

IPv4 Link-lokal adresse [10]

B.1 Oversikt

Link-lokal adresse protokollen [8] er en måte å tildele en IP-adresse i de tilfeller der man har behov for en IP-adresse men ikke har tilgang til DHCP eller at manuell konfigurering er uhensiktsmessig. Adressene tildeles på prefikset 169.254/16 der de 256 første og 256 siste adressene er reservert for spesielle formål. Link-Lokal adressene kan kun kommunisere over en fysisk (eller logisk) link. Protokollen er beregnet brukt i små ad-hoc nettverk på en singel link som inneholder noen få noder [1]. En node som konnekter seg til en link som allerede har 1300 andre noder og som velger seg en tilfeldig adresse har 98% sjanse for å velge en ubrukt adresse, og har 99,96 % sjanse ved to forsøk. Sannsynlighet for å måtte prøve mer enn 10 ganger er ca 10^{-17}. Link-lokal adresser skal bare bli brukt der stabile rutbare adresser ikke er tilgjengelige, eksempelvis i ad-hoc eller isolerte nettverk.

B.2 Adressevalg

Når en node skal konfigurere en Link-lokal adresse, genereres det ved hjelp av en algoritme en tilfeldig adresse. Algoritmen er avhengig av implementasjonen som er valgt. Man kan velge en adresse som genereres mest mulig tilfeldig eller man kan generere en adresse som tar utgangspunkt i MAC-adressen. Den sistnevnte vil da ofte starte med å generere samme adresse hver gang, noe som kan være en fordel hvis adressen tidligere har vist seg å være unik. Adressen må som tidligere nevnt være innenfor 169.254/16 som er Link-lokal adresser.

[1] Her definert til mindre enn 1300.

B.3 Adresseannonsering

Når en adresse er valgt må det gjøres en test for å sjekke om adressen er unik. Hvis adressen er opptatt må det genereres en ny adresse helt til man finner en unik adresse eller til man når et maks antall forsøk. Dette makstallet er bestemt av implementasjonen. Test om adressen er i bruk kan gjøres ved hjelp av en ARP med 0.0.0.0 som avsenderadresse. Avsenderadressen settes til 0.0.0.0 for å unngå at grensesnittene i nettverket oppdaterer sine ARP-tabeller før det har blitt klarlagt at adressen er unik. Dette kalles en ARP-probe. Når dette har blitt gjort et antall ganger uten at man har støtt på adressekonflikt kan man annonsere adressen ved å sende ut en ARP med avsenderfeltet utfylt. Dette vil få de andre grensesnittene til å oppdatere sine ARP-tabeller.

B.4 Håndtering av adressekonflikter

Adressekonflikthåndtering er ikke bare begrenset til adressevalgfasen mens en node sender ARP-prober. Dette er en prosess som pågår hele tiden mens noden bruker en Link-lokal adresse. Hvis en node mottar en ARP-probe som indikerer en adressekonflikt kan den velge to fremgangsmåter:

a)Konfigurer opp en ny Link-lokal adresse som nevnt ovenfor.

b)Hvis man - har aktive TCP forbindelser eller av andre årsaker ikke ønsker å bryte forbindelsen, kan man prøve å forsvare adressen ved å broadcaste med en enkelt ARP-melding med sin egen MAC-adresse og IP-adresse. Dette gjelder kun hvis man ikke har sett andre ARP-pakker innenfor det siste DEFEND_INTERVAL sekunder. Hvis dette ikke er den første ARP-pakke med adressekonflikt innenfor DEFEND_INTERVAL sekunder må noden øyeblikkelig slutte å bruke denne adressen, og konfigurere opp en Link-lokal adresse som beskrevet ovenfor. Dette må gjøres for å unngå en evig loop der to noder vil prøve å forsvare den samme adressen. Denne fremgangsmåten vil bryte eventuelt pågående TCP forbindelser, men vil forekomme meget sjeldent.

B.5 Konfigurasjonsregler

Link-lokal adresser kan ikke forwardes av en ruter, og kan kun kommunisere med enheter på samme link. Grensesnitt med disse adressene kan både kommunisere med noder som har Link-lokal adresser og andre med rutbare adresser. Link-lokal prefikset 169.254/16 skal heller ikke subnettes, noe som muliggjør bruk av ARP for å avdekke adressekonflikter. Link-lokal tillater både rutbare og Link-lokal adresser på samme node. Hvis en node har mer enn ett grensesnitt på samme link der alle supporter Link-lokal autokonfigurering, må autogenerering av adresser basere seg på algoritmer som knytter seg til grensesnittet og ikke til noden. Dette for å minske risikoen for å generere like adresser.

Tillegg C

Loose source routing - Route record [16]

0-3	4-7	8-11	12-15	16-19	20-23	24-27	28-31
Version	IHL	TOS		Total Length			
Identifiaction				Flags	Fragment offset		
TTL		Protocol		Header checksum			
Source IP address							
Destination IP address							
Options and padding							

IP-header

Type	length	pointer	route data

Optionfeltet i en IP-header ved LSR og RR

Loose source routing og Route record er en mulighet for kilden til et datagram for enten å gi informasjon hvordan pakken skal rutes eller registrere informasjon om hvilken rute pakken følger.

Optionfeltet begynner med type. Neste felt er lengden på hele feltet, deretter pekeren (pointer) og til slutt route data.

Når type = 131 vil det si at datagrammet er i loose source routing mode (LSR) og hvis type = 7 er datagrammet i route record mode (RR).

Optionfeltet har en lengde på 40 oktetter. Type = 2 oktetter, lengde = 1 oktett, peker = 1 oktett noe som gir 36 oktetter igjen til "route data". Når vi vet at en IP-adresse krever 4 oktetter gir dette rom for maks 9 IP-adresser for å gi eller registrere rutingvei.

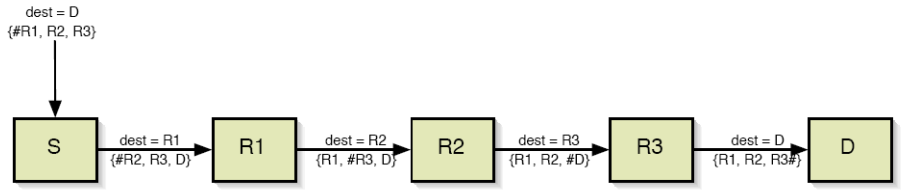
Hvis type = RR vil pekeren peke på der neste IP-adresse skal legges. Pekeren er relativ i forhold til starten på optionfeltet, og laveste lovlige verdi = 4. Hvis vi har lagt inn 0 adresser i route data vil derfor pekeren stå på 4. Når vi har lagt inn 1 adresse vil pekeren ha verdi 8. Vi kan legge til nye IP-adresser helt til pekeren når sin maksverdi som er 40.

Hvis type = LSR skal datagrammet følge ruten som ligger i route data. Man starter å rute på 1. adresse, det vil si pointer = 4. Dette betyr at hvis man skal sende et datagram i retur som har vært i RR mode må man snu route data slik at den siste adderte adresse blir den første.

LSR kan lettest forklares ved hjelp av en figur, men følgende grunnelementer kan forklares først:

1) Hvis pekerverdien er større enn lengden på optionfeltet og måladressen ikke er nådd, må resten av rutingen baseres på vanlig rutingprinsipper.

2) Hvis måladressen er nådd og pekerverdien er mindre enn lengden på optionfeltet så bytter vi neste adresse i source route med måladressen, helt til source route er tom og vi har nådd det endelige målet eller 1).



Figur C.1: . Source routing [12] I denne figuren representerer # pekeren. Hvis den peker lengst til venstre vil pekerverdi = 4, deretter 8, 12.

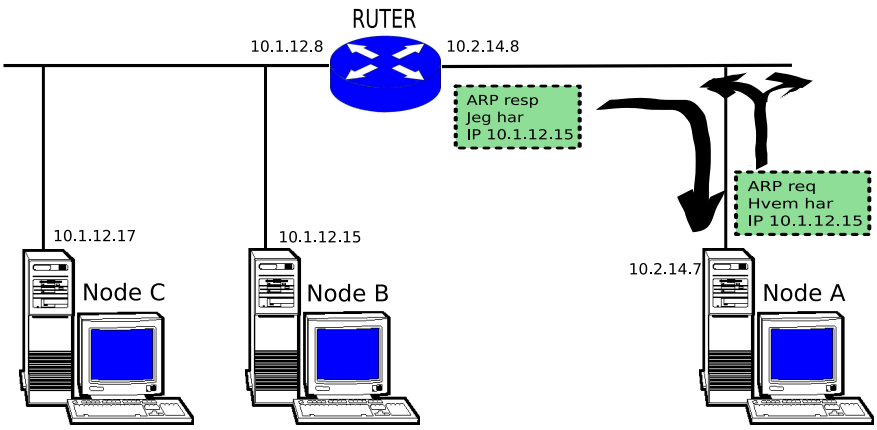
Når pakken sendes ut vil kilden S plassere D til sist i tabellen og sette dest = R1. Her vil fremdeles pekeren ha verdi = 4. Deretter skjer følgende - for hver ny ruter pakken blir rutet gjennom vil adressen som pekeren peker på og innholdet i dest bytte plass, i tillegg vil pekerverdien øke med 4. Dette gjentas helt til kilden D nås eller pekerverdien = 40.

Tillegg D

Proxy-ARP [9]

En vanlig ARP-forespørsel fungerer kun innenfor et fysisk nettverk. Hvis vi har to noder, A (kilden) og B (målet), som befinner seg på forskjellige fysiske nett vil ikke B kunne respondere på en ARP fra A.

Hvis disse to nettene er knyttet sammen med en ruter som er konfigurert med proxy-ARP vil ruteren se ARP forespørselen fra A. Ruteren kan da respondere på vegne av B som om rutergrensesnittet var grensesnittet til B. Da vil A anta at B er på det samme fysiske nettet. Ruteren oppfører seg som en agent for B, dette kalles Proxy-ARP.



Figur D.1: Proxy-ARP

Her kan ruteren eksempelvis fungere som en gateway inn til et ad-hoc nettverk som vil ha subnett 10.1.12.0/24 og vil svare positivt til alle ARP forespørsler til dette nettverket. Ruteren vil her bli ansvarlig for å videresende pakker inn på dette nettverket.

Man kan også utvide et fysisk LAN på denne måten, og nodene på hver sin side av ruteren vil ikke være klar over at en ruter befinner seg imellom.

Tillegg E

Klargjøring av Linksysruter

Alle kommandoer og script her forutsetter at ethernetporten på PC'en heter eth0 og at man kjører Linux OS. Alle nødvendige bibliotek og script er lagt ved denne oppgaven i en arkivfil.

E.1 Kompilering av Click til Linksys WRT54GL med OpenWRT

1. Tilkobling til server

   Oppsett av et krysskompileringsmiljø viste seg å være en komplisert affære. Oppskriften som står i AntPing leveransen [12] virket ikke på de PC'ene jeg hadde tilgjengelig. Jeg fikk derfor tilgang på en server på Telenors pats-lab. Denne serveren er ferdig oppsatt med riktig kompileringsmiljø og kan produsere binære click-filer som kjører på OpenWRT. For å koble meg til denne serveren måtte jeg bruke VPN og SSH. VPN krevde MPPE (Microsofts point-to-point encryption) som det heldigvis er støtte for i Linux kjernen fra og med utgave 2.6.15 Dette gjorde at jeg kunne jobbe direkte på denne serveren fra min Linux PC. Etter mye leting fant jeg software som var brukbar til formålet. PPTP ble installert og kan hentes her [6]. Denne ble konfigurert til å bruke MPPE. Dette fikk meg til pats-labben. For å logge meg på Bison serveren måtte jeg bruke SSH protokollen. Jeg brukte Krusander for grafisk brukergrensesnitt og Putty for kommandolinje.

2. Kompilering av click

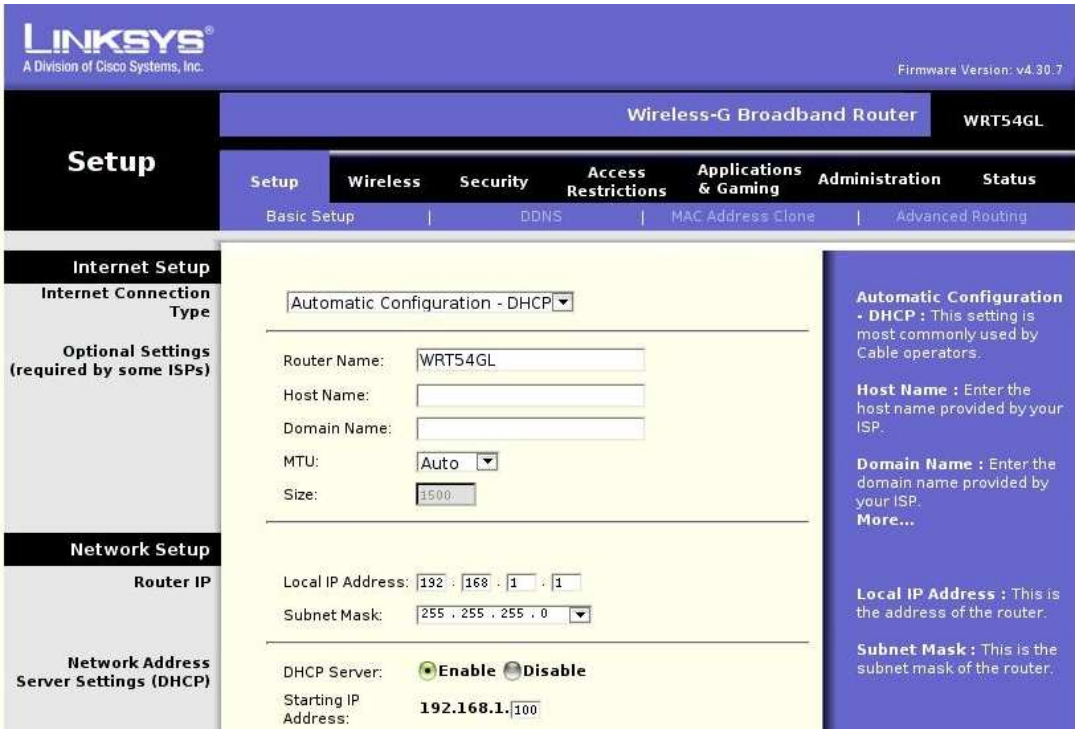
   - Logget meg på Bison serveren (se pkt 1).
   - Kopierte mine elementer pluss de elementer laget i AntPing leveransen til bison0://home/bison/openwrt/obsolete-buildroot/build-mipsel/click/elements/local#/

- For å være sikker at de "nye" elementene ble kompilert måtte jeg fjerne bison0://home/bison/openwrt/obsolete-buildroot/build-mipsel/click/userlevel/ *.mk , *.conf og *.o (VIKTIG !!! Ta backup først !!!)
- Kjør kommandoen #/home/bison/openwrt/obsolete-buildroot/make click
- Den ferdig kompilerte binærfila click kan hentes her bison0://home/bison/openwrt/obsolete-buildroot/build-mipsel/staging_dir/bin/bin#

E.2 Installering av OpenWRT.

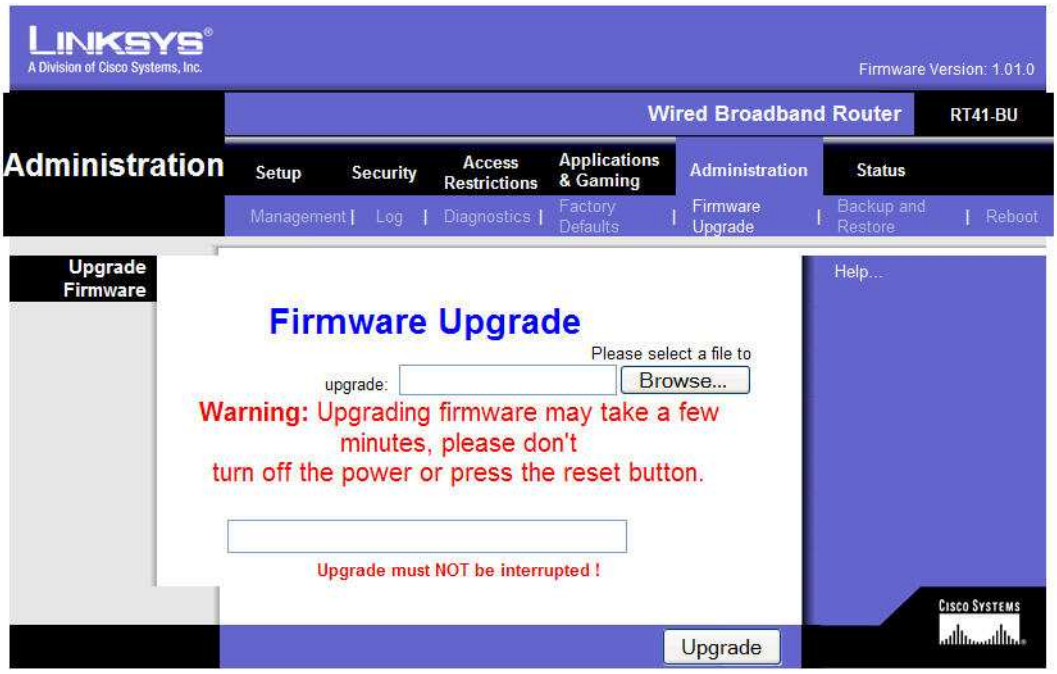
Her følger en "kokeoppskrift" på hvordan flashe Linksys WRT54GL med Openwrt Linux. Alle kommandoer og script her forutsetter at ethernetporten på PC'en heter eth0 og at man kjører Linux OS.

- Koble en av lanutgangene på ruteren til eth0.
- Åpne et konsoll.
- #ifconfig eth0 192.168.1.2
- Åpne nettleseren og skriv inn adressen til ruteren http://192.168.1.1. Du vil da få opp et konsoll tilsvarende dette.



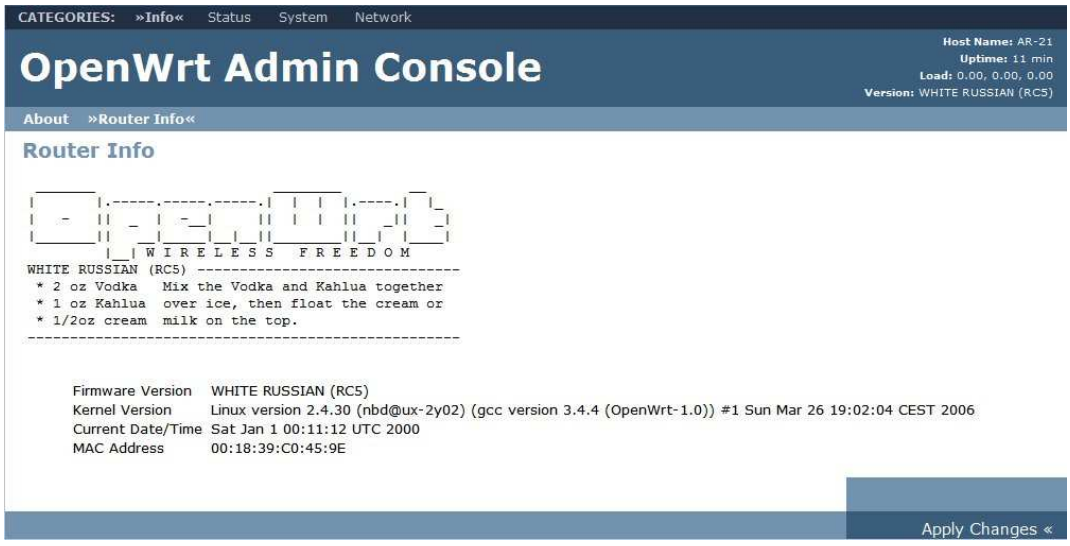
Figur E.1: Linksys WRT54GL oppstartskonsoll

- Klikk på fanen Administrasjon og dette dukker opp.



Figur E.2: Firmware oppgradering ved hjelp av Linksys konsoll

- Man kan da lete fram den vedlagte fila "openwrt-wrt54g-squashfs.bin". Trykk oppdater og ruteren blir flashet med Openwrt RC5 Linux firmware.



Figur E.3: OpenWRT konsoll

- Trykk så på fanen "System" og du vil få beskjed om å legge inn passord. Dette passordet

vil bli brukt senere til å komme inn i ruteren. Så kan du bare lukke dette vinduet. Det enkleste her er å ha felles passord for alle ruterne.

Dette er et vanlig grafisk konsoll som du kan bruke til å sette de vanlige innstillinger på en ruter. Her kan du også flashe tilbake original firmware.

E.3 Installering av nødvendige biblioteker og script.

Her følger en "kokeoppskrift" på hvordan installere nødvendige biblioteker og script på en Linksys WRT54GL med Openwrt Linux:

<HOME> tilsvarer din hjemmekatalog.

Åpne et terminalvindu i hjemmekatalogen.

Kopier AntPing_demo.tar.gz til hjemmekatalogen og pakk ut katalogstrukturen med kommandoen

#tar -zxvf AntPing_demo.tar.gz

Alle nødvendige filer er nå pakket i egen katalogstruktur under <HOME>/demo.

E.3.1 Konfigurere WAN port.

For å kunne konfigurere, styre og legge inn nødvendige script må man gi ruteren et unikt navn og gi WAN porten en unik IP-adresse. Jeg har valgt å følge samme navneregime som AntPing leveransen ved gi ruterne navn fra AR-1 - AR-8 og med WAN IP-adresser 10.100.1.11-18. Default GW til disse WAN portene blir satt til 10.100.1.1 som er adressen man må gi eth0 på den PC som skal brukes til konfigurasjon og styring.

- Koble PC'en til en av lanutgangene på ruteren.
- Åpne et root terminalvindu.
- # ifconfig eth0 192.168.1.2
- #ssh 192.168.1.1 'rm /etc/init.d/S45firewall
  (fjerner brannmur i ruteren)
- #cd /<HOME>/demo/configure/
- #scp /script/nvram-fix.sh root@192.168.1.1:/etc/.

- #ssh 192.168.1.1 '/etc/nvram-fix.sh X'

  (der X byttes ut med nr til ruteren, feks AR-1 , X=1)

- #ssh 192.168.1.1 reboot

Ruteren er nå konfigurert med WAN IP-adresse 10.100.1.1X som vil kunne nås via WAN-porten.

Dette gjentas for alle rutere. Det er mulig at for hver gang du konfigurerer opp en ruter vil du få beskjed om å endre fila /root/.ssh/known_hosts. Da er det bare å slette denne.

E.3.2 Installering av nødvendige software og script samt konfigurering av vlanporter.

Koble alle WAN-porter til eth0 på PC'en via en svitsj. Vi skal nå kjøre noen script som vil automatisere installasjonsprosessen. For at man skal slippe å taste passordet til ruteren for hver gang en kommando skal utføres, må man først generere et nøkkelpar som deretter kopieres til ruteren. Det følgende script vil gjøre dette. Under kjøring av dette scriptet vil du måtte taste inn passordet 3 ganger for hver ruter. Du må også taste "yes" for å godta at ruteren blir lagt inn i /root/.shh/known_host.

- Åpne et root terminalvindu.
- #cd /<HOME>/demo/configure/
- #./install_key_files.sh

Deretter kjører du et script for å installere all nødvendige script, bibliotek og programvare for at ruteren skal være en AntPing ruter. Dette scriptet vil gå uten inngripen, men vil bruke noen minutter. Du kan følge forløpet i terminalvinduet.

- #./misc.sh

De nødvendige oppstartscript og programvare er nå installert, og ruterne er nå ferdig konfigurert med nødvendige IP-adresser. Ruterne er nå rebootet og Click kjører.

Tillegg F

Oppstartsprosedyrer testoppsett

Ruter skal automatisk starte Click ved oppstart, men jeg har laget en del script som kan være til hjelp under testing, feilsøking og demo. Man må være tilkoblet WAN porten på ruteren for å utføre de påfølgende script og prosedyrer. Hvis PC'en ikke har vært benyttet tidligere mot disse ruterne, eller man har brukt en annen PC i mellomtiden mot disse, må man for å kunne kjøre SSH uten passord repetere prosedyren som står under E.3.2.

- Åpne et root terminalvindu.

  #ifconfig eth0 10.100.1.1

- For å kjøre Click manuelt må man logge seg inn på ruteren ved hjelp av ssh.

  #ssh 10.100.1.1X

  Der X er nummeret på ruteren.

- Nå er vi inne på ruteren. Her vil vi se et normalt Linux terminalvindu. For å sjekke om Click kjører, gi kommandoen.

  %>ps

  Du vil nå få en liste over alle prosesser. Hvis du ser en linje som inneholder dette:

  '/sbin/click /sbin/antX.click' så kjører clickprosessen.

- Man kan stoppe den ved å skrive

  %>killall click.

- For å starte click igjen gi kommandoen

  %>/sbin/click /sbin/antX.click

- For å gå ut av ruteren gi kommandoen

  %>exit

- For å slippe å starte eller stoppe alle ruterne ved manuelt å gå inn på hver ruter har jeg laget/modifisert noen enkle script.

  Du må være i et terminalvindu med root privilegier. Skift så til riktig katalog med kommandoen.

  #cd /<HOME>/demo/configure

  #./isup.sh

  Dette scriptet vil gi en beskjed om hvilke rutere som kjører og i terminalvinduet vil det samtidig bli vist alle prosesser som kjører på den enkelte ruter. Man vil se at click kjører ved å se etter 'ash -c /sbin/click /sbin/antX.click'.

```
AR-8 IS UP
  PID  Uid     VmSize Stat Command
    1 root        368 S   init
    2 root            SW  [keventd]
    3 root            SWN [ksoftirqd_CPU0]
    4 root            SW  [kswapd]
    5 root            SW  [bdflush]
    6 root            SW  [kupdated]
    8 root            SW  [mtdblockd]
   49 root            SWN [jffs2_gcd_mtd4]
   71 root        348 S   logger -s -p 6 -t
   73 root        368 S   init
   74 root        344 S   syslogd -C 16
   76 root        300 S   klogd
  322 root        308 S   wifi up
  329 root        400 S   /usr/sbin/dropbear
  332 root        368 S   httpd -p 80 -h /www -r OpenWrt
  340 root        260 S   telnetd -l /bin/login
  344 root        368 S   crond -c /etc/crontabs
11831 root        368 S   ash -c /sbin/click /sbin/ant8.click
11832 root       2724 S   /sbin/click /sbin/ant8.click
13950 root        576 S   /usr/sbin/dropbear
13951 root        368 S   ash -c ps
13952 root        360 R   ps
```

- Hvis man ønsker å restarte alle ruterne på nytt for å tømme dem for alle temperaturdata kan man kjøre scriptet

  #./stopall.sh

  og deretter

  #./startall.sh

stopall.sh vil stoppe clickprosessen på hver enkelt ruter.

startall.sh vil starte clickprosessen igjen. Man vil for hver ruter få opp i terminalvinduet:

```
Running on 18
Adding interface 10.1.8.1/24 and cost 35 on physical port 3 to table of interfaces
Adding interface 10.1.8.2/24 and cost 30 on physical port 2 to table of interfaces
Adding interface 10.1.8.3/24 and cost 1 on physical port 1 to table of interfaces
Adding interface 10.1.8.4/24 and cost 1 on physical port 0 to table of interfaces
```

Man ser da at prosessen har startet, man må så trykke Crtl c for at scriptet skal gå til neste ruter for å kjøre opp prosessen der. Dette er svært nyttig da man kan se at prosessen kommer opp og virker. Hvis man har koblet til alle kablene og startet hping prosessen til senderpc'en vil man i terminalvinduet kunne observere at pakker mottas og sendes. Dette gjentas helt til alle ruterne er oppe og man har kontroll på at de kjører.

- Du kan også gi kommandoen

  #./restartall.sh

  som først vil kjøre stopall.sh og deretter startall.sh

Tillegg G

Oppstart av demo

Demoen består av følgende utstyr:

8 Linksys WRT54GL rutere

2 8ports svitsjer

1 PC med Linux OS

+ et antall ethernet kabler

Figur 4.1 viser testoppsettet.

1. Hping3, nam og gnuplot må være installert på PC'en. Kildekode kan finnes på [2, 4, 1]
2. Linksysruterne klargjøres i henhold til tillegg E.
3. Deretter starter man opp Linksysruterne, se tillegg **??**.
4. Kjør AntPing sender

   Koble til Linux PC til en port på en ruter. For å gjøre skjermbildet mest mulig ryddig er det beste å koble den til AR-1.

   Åpne et root terminalvindu

   #ifconfig eth0 10.1.0.1

   #cd /<HOME>/demo/hping/

   #hping3 exec tx_final.htcl 10.1.8.3 eth0 50000 0 1000 30 0.05 0.95 0.20

   Dette vil sende forwardant til 10.1.8.3 på UDP-port 50000 (forwardant) ut på eth0. Dette skjer med 1000ms mellom hver forwardant, og det blir sendt 30 initielle explorerant med ρ=0.05 og β=0.95. Deretter vil det sendes 20% explorerant.

For enkelhets skyld kan man også kjøre scriptet

#./run_hping.sh

Dette vil gi samme resultat.

5. Kjør animasjon

   tx_final.htcl sender alle hendelser som linkopp, linkned og andre animasjonsdata til fila t50000.nam. Nam leser så kontinuerlig denne fila og lager en animasjon av denne.

   Åpne et nytt root konsoll.

   #cd /<*#cd /<HOME>/demo/hping/*

   #tail -f -n 1000 t50000.nam | nam -r 40ms -

   eller for enkelhets skyld

   #./run_nam.sh

6. Plot data

   makeplot kjører makegnuplot som produserer plot.gnu. makeplot kjører gnuplot ved hjelp av plot.gnu som leser fila trace-50000 som inneholder cost og temperaturdata. Åpnet et konsoll.

   #cd /<HOME>/demo/hping/

   #./start-plot-demo

7. Se plotten i en webleser.

   makeplot produserer png filer som kan sees gjennom plot.html.

   Åpne fila /<HOME>/demo/plot.html med webleseren din.

Tillegg H

Hping-script

```
#
# Name:tx_final.htcl
# Purpose: hping tcl script for generating ants in the BISON demo
# Description: The ants are IP packets with dst port 51234 sent to a
# process "recv.htcl" that resides on the destination node.
# This process calculates the temperature based
# on the cost values seen so far and the cost value carried
# by the last ant. For this demo, the TTL is set
# to 8 because this is the maximum number of steps that can
# be observed by the route record in IPv4 optional header.
# The cost and temperature of the returning ants are written
# to a trace file named "trace"
#
# Change the use of tos bit. Cause problems when sending from
# windows machines and through commersial routers.
#
# Extends "send.htcl" by generating NAM trace data from cost values.
#
# Project: BISON
# Documantation: Deliverable D14 of BISON
# Last modified: 2006-03-10
# Author: Poul Heegaard, Telenor R&D
#
# Check arguements, expects $target and $inf and $antId
if {$argc < 5} {
puts stderr "Usage: send <target> <inf> <antid> <data> <btw> \[<explAnts>\] \[<rho>\] \[<beta>\] \[<explRa-
tio>\]"
puts stderr "Example: send 10.7.8.2 eth0 50000 0 1000 \[ 10 0.05 0.98 0.20 \] "
exit
}
# globals: read argument list
```

```
foreach {target inf antId data btw explAnts rho beta explRatio} $argv break
# observe that $data == 0 is ants, $data == 1 is data
# CEants parameters (these might also later be given as arguments)
if {$rho == {}} { set rho 0.05 }
if {$beta == {}} { set beta 0.98 }
# the 10 first ants are explaratory ants
if {$explAnts == {}} { set explAnts 10 }
# Default is 20% of the ants are maintenance (explaration) ants
if {$explRatio == {}} { set explRatio 0.2 }
# btw sets the time between packets in ms, e.g. this means 1 sec. (1000 ms)
# set btw 1000
# Nam topology and initialisations
# test
#set node(127.0.0.1) 0
set node(10.1.0.1) 0
set node(10.1.0.2) 100
# demo net
set node(10.1.1.1) 1
set node(10.1.1.2) 1
set node(10.1.1.3) 1
set node(10.1.1.4) 1
set node(10.1.2.1) 2
set node(10.1.2.2) 2
set node(10.1.2.3) 2
set node(10.1.2.4) 2
set node(10.1.3.1) 3
set node(10.1.3.2) 3
set node(10.1.3.3) 3
set node(10.1.3.4) 3
set node(10.1.4.1) 4
set node(10.1.4.2) 4
set node(10.1.4.3) 4
set node(10.1.4.4) 4
set node(10.1.5.1) 5
set node(10.1.5.2) 5
set node(10.1.5.3) 5
set node(10.1.5.4) 5
set node(10.1.6.1) 6
set node(10.1.6.2) 6
set node(10.1.6.3) 6
set node(10.1.6.4) 6
set node(10.1.7.1) 7
set node(10.1.7.2) 7
set node(10.1.7.3) 7
```

```
set node(10.1.7.4) 7
set node(10.1.8.1) 8
set node(10.1.8.2) 8
set node(10.1.8.3) 8
set node(10.1.8.4) 8
set node(10.1.9.1) 9
set node(10.1.9.2) 9
set node(10.1.9.3) 9
set node(10.1.9.4) 9
set node(10.1.10.1) 10
set node(10.1.10.2) 10
set node(10.1.10.3) 10
set node(10.1.10.4) 10
if { $data == 0 } {
# begin demo nam preamble
set delay 0.060
set delayms 40ms
# open nam file for animations
set namfile [open "t$antId.nam" w]
puts $namfile "V -t * -v 1.0a5"
set ingenting " "
# puts $namfile "n -t * -s 100 -v circle -c black -z 10 -S UP SRC2"
puts $namfile "n -t * -s 0 -v circle -c black -z 10 -S UP -b SRC1"
puts $namfile "n -t * -s 8 -v circle -c black -z 10 -S UP -b AR-8"
puts $namfile "n -t * -s 7 -v circle -c black -z 10 -S UP -b AR-7"
puts $namfile "n -t * -s 6 -v circle -c black -z 10 -S UP -b AR-6"
puts $namfile "n -t * -s 5 -v circle -c black -z 10 -S UP -b AR-5"
puts $namfile "n -t * -s 4 -v circle -c black -z 10 -S UP -b AR-4"
puts $namfile "n -t * -s 3 -v circle -c black -z 10 -S UP -b AR-3"
puts $namfile "n -t * -s 2 -v circle -c black -z 10 -S UP -b AR-2"
puts $namfile "n -t * -s 1 -v circle -c black -z 10 -S UP -b AR-1"
puts $namfile "l -t * -s 6 -d 7 -e RED -r 100kb -D $delayms -O 326.29deg -h 68 -S COLOR -c grey90"
#puts $namfile "l -t * -s 5 -d 10 -e RED -r 100kb -D $delayms -O 45 -h 50 -S COLOR -c grey90 "
#puts $namfile "l -t * -s 5 -d 9 -e RED -r 100kb -D $delayms -O 180deg -h 100 -S COLOR -c grey90"
puts $namfile "l -t * -s 5 -d 8 -e RED -r 100kb -D $delayms -O 15.95deg -h 152 -S COLOR -c grey90"
puts $namfile "l -t * -s 5 -d 7 -e RED -r 100kb -D $delayms -O 0deg -h 100 -S COLOR -c grey90"
#puts $namfile "l -t * -s 4 -d 10 -e RED -r 100kb -D $delayms -O 153.32 -h 241 -S COLOR -c grey90"
#puts $namfile "l -t * -s 4 -d 9 -e RED -r 100kb -D $delayms -O 165.91 -h 282 -S COLOR -c grey90"
puts $namfile "l -t * -s 7 -d 4 -e RED -r 100kb -D $delayms -O 301.08deg -h 62 -S COLOR -c grey90 "
puts $namfile "l -t * -s 4 -d 6 -e RED -r 100kb -D $delayms -O 45deg -h 147 -S COLOR -c grey90 "
puts $namfile "l -t * -s 4 -d 5 -e RED -r 100kb -D $delayms -O 156.66deg -h 159 -S COLOR -c grey90"
#puts $namfile "l -t * -s 3 -d 10 -e RED -r 100kb -D $delayms -O 138,79deg -h 229 -S COLOR -c grey90 "
#puts $namfile "l -t * -s 3 -d 9 -e RED -r 100kb -D $delayms -O 153.42deg -h 242 -S COLOR -c grey90 "
puts $namfile "l -t * -s 3 -d 8 -e RED -r 100kb -D $delayms -O 74d.09deg -h 152 -S COLOR -c grey90"
```

```
puts $namfile "l -t * -s 3 -d 6 -e RED -r 100kb -D $delayms -O 113 -h 160 -S COLOR -c grey90 "
puts $namfile "l -t * -s 3 -d 5 -e RED -r 100kb -D $delayms -O 135deg -h 130 -S COLOR -c grey90 "
#puts $namfile "l -t * -s 2 -d 9 -e RED -r 100kb -D $delayms -O 156deg -h 160.16 -S COLOR -c grey90 "
puts $namfile "l -t * -s 2 -d 7 -e RED -r 100kb -D $delayms -O 45deg -h 82.58 -S COLOR -c grey90 "
puts $namfile "l -t * -s 2 -d 5 -e RED -r 100kb -D $delayms -O 123.66deg -h 67 -S COLOR -c grey90 "
puts $namfile "l -t * -s 2 -d 3 -e RED -r 100kb -D $delayms -O 326.29deg -h 67.93 -S COLOR -c grey90 "
#puts $namfile "l -t * -s 1 -d 10 -e RED -r 100kb -D $delayms -O 113.16 -h 160 -S COLOR -c grey90 "
puts $namfile "l -t * -s 1 -d 6 -e RED -r 100kb -D $delayms -O 74.09deg -h 152.5 -S COLOR -c grey90 "
puts $namfile "l -t * -s 1 -d 4 -e RED -r 100kb -D $delayms -O 16.05deg -h 151 -S COLOR -c grey90 "
# puts $namfile "l -t * -s 9 -d 10 -e RED -r 100kb -D $delayms -o 45deg -h 50 -S COLOR -c grey90"
puts $namfile "l -t * -s 7 -d 8 -e RED -r 100kb -D $delayms -o 45deg -h 50 -S COLOR -c grey90"
# puts $namfile "l -t * -s 6 -d 10 -e BLUE -r 100kb -D $delayms -o 180deg -h 100 -S COLOR -c grey90"
puts $namfile "l -t * -s 6 -d 8 -e BLUE -r 100kb -D $delayms -o 0deg -h 100 -S COLOR -c grey90"
puts $namfile "l -t * -s 5 -d 6 -e RED -r 100kb -D $delayms -o 45deg -h 50 -S COLOR -c grey90"
# puts $namfile "l -t * -s 8 -d 10 -e RED -r 100kb -D $delayms -O 180deg -h 215 -S COLOR -c grey90 "
puts $namfile "l -t * -s 4 -d 8 -e BLUE -r 100kb -D $delayms -o 90deg -h 100 -S COLOR -c grey90"
# puts $namfile "l -t * -s 7 -d 9 -e RED -r 100kb -D $delayms -O 180deg -h 215 -S COLOR -c grey90 "
puts $namfile "l -t * -s 3 -d 7 -e BLUE -r 100kb -D $delayms -o 90deg -h 100 -S COLOR -c grey90"
puts $namfile "l -t * -s 3 -d 4 -e RED -r 100kb -D $delayms -o 45deg -h 50 -S COLOR -c grey90"
# puts $namfile "l -t * -s 2 -d 10 -e BLUE -r 100kb -D $delayms -o 135deg -h 141-S COLOR -c grey90"
puts $namfile "l -t * -s 2 -d 6 -e BLUE -r 100kb -D $delayms -o 90deg -h 100 -S COLOR -c grey90"
puts $namfile "l -t * -s 2 -d 4 -e RED -r 100kb -D $delayms -o 0deg -h 100 -S COLOR -c grey90"
# puts $namfile "l -t * -s 1 -d 9 -e BLUE -r 100kb -D $delayms -o 135deg -h 141 -S COLOR -c grey90"
puts $namfile "l -t * -s 1 -d 5 -e BLUE -r 100kb -D $delayms -o 90deg -h 100 -S COLOR -c grey90"
puts $namfile "l -t * -s 1 -d 3 -e RED -r 100kb -D $delayms -o 0deg -h 100 -S COLOR -c grey90"
puts $namfile "l -t * -s 1 -d 2 -e RED -r 100kb -D $delayms -o 45deg -h 50 -S COLOR -c grey90"
# puts $namfile "l -t * -s 2 -d 8 -e RED -r 100kb -D $delayms -O 45deg -h 147 -S COLOR -c grey90 "
# puts $namfile "l -t * -s 1 -d 8 -e RED -r 100kb -D $delayms -O 45deg -h 200 -S COLOR -c grey90 "
# puts $namfile "l -t * -s 1 -d 7 -e RED -r 100kb -D $delayms -O 45deg -h 147 50 -S COLOR -c grey90 "
puts $namfile "l -t * -s 1 -d 0 -e RED -r 100kb -D $delayms -o 180deg -h 50 -S COLOR -c grey90"
puts $namfile "c -t * -i 0 -n Blue"
puts $namfile "c -t * -i 1 -n Green"
puts $namfile "c -t * -i 2 -n Brown"
puts $namfile "c -t * -i 3 -n Yellow"
puts $namfile "c -t * -i 4 -n Yellow"
puts $namfile "c -t * -i 5 -n Red"
puts $namfile "a -t 0 -s $node([hping outifa $target]) -n AntGen"
puts $namfile "a -t 0 -s $node([hping resolve $target]) -n AntRec"
flush $namfile
# end demo nam preamble
# print ant setup
puts "————————————————————————————————"
puts "Finds connections from [hping outifa $target] to $target ([hping resolve $target])"
puts "sending an ant every $btw ms, with [expr 100*$explRatio]% exploration"
puts "number of initial exploration ants = $explAnts, "
```

```
puts "and rho=$rho and beta=$beta."
puts "————————————————————————————————"
puts " "
} else {
# print data setup
puts "————————————————————————————————"
puts "Data sent on connection from [hping outifa $target] to $target ([hping resolve $target])"
puts "sending data every $btw ms"
puts "————————————————————————————————"
puts " "
}
# Init variables
set accTime 0
# counts number of received ants
set recvind 0
# counts number of sent ants
set sentind 1
#set tcl_precision 4
# Send ants every $btw [ms] to destination as specified in $target
proc genAnts {} {
global target btw sentind recvind explAnts explRatio antId rho beta delay accTime data
# explore = 0 : maintenace ants and data packets
# explore = 1 : exploration ants
# explore = 2 : exploration ants, initialisation phase
# cost = 0: ant packets
# cost > 0: data packets (the value indicates the increment in the receiver counter)
# the ant is sent as exploration ant at startup and later as maintenance
# the initial exploration phase is over when the predefined number of ants are received
if {$recvind <= $explAnts} {
set explore 2
} else {
set explore [expr rand() < $explRatio ]
}
if { $data > 0 } {
set explore 0
}
# construct packet to be sent to $tagret and with route record
set pck "ip(daddr=$target,ttl=10)+ip.rr"
append pck "+udp(sport=$antId,dport=51234)"
set cost $data
if { $explore == 2 } {
# Send initialisation exploration ant to reset
# the variables on receiver for port number $antId
append pck "+data(str=$explore/$rho/$beta/$cost/$btw)"
} else {
# cost values for testing of NAM trace generation
```

```
append pck "+data(str=$explore/$cost)"
}
# send packet
incr sentind 1
if { $data == 0 | $sentind < 20 } {
#puts "Generated: $pck"
if { $explore == 0 } {
puts -nonewline stdout "."
} else {
puts -nonewline stdout "E"
}
flush stdout
}
hping send $pck
# next packet using "bootstrapping"
after $btw genAnts
}
# Read incoming ants and write cost and current temperature to $outfile
proc recAnts {} {
global inf outfile outfile2 namfile recvind antId node delay target accTime rho
# receive packet accoring to filter settings
set p [hping recv $inf 10]
set pck [lindex $p 0]
# read source and destionation ports and data field
set sp [hping getfield udp sport $pck]
set dp [hping getfield udp dport $pck]
set dt [hping getfield data str $pck]
# content of data field:
# 0 = expldata
# 1 = exp(-cost/temp) (for click)
# 2 = cost
# 3 = temp
# 4 = beta
# 5 = accTime (for nam trace)
# 6 = #hops in path
# 7 - (7+#hops-1) = table of costs
# (7+#hops) - end = route addresses
set tabx [split $dt !]
set tab [split [lindex $tabx 0] /]
set tablength [llength $tab]
set explore [lindex $tab 0]
set cost [lindex $tab 2]
set temp [lindex $tab 3]
set beta [lindex $tab 4]
set linkdsrc 0
```

```
if { $explore == 6 | $explore == 7} {
puts "Tabelllengde er $tablength"
set rAdr [expr {$tablength-1}]
set linkCost [expr {$tablength-2}]
puts "Tabelllengde -1 er $rAdr"
set linkdsrc [lindex $tab $rAdr]
}
if { $explore < 0 } {
set tmpaccTime [lindex $tab 5]
} else {
set tmpaccTime 0
}
if { $accTime < $tmpaccTime } {
set accTime $tmpaccTime
}
# determine number of hops
set hops [lindex $tab 6]
# set linkdsrc [lindex [expr {$tablength-1}]]
# is this an ant? (the filter settings did not work when too restrict)
if { $sp == 51234 } {
#puts "Received: $pck"
puts -nonewline stdout "*"
flush stdout
set dt [hping getfield data str $pck]
set lsrr [hping getfield ip.lsrr data $pck]
set lsrl [hping getfield ip.lsrr ptr $pck]
# Update ant received - write cost and temp to trace file
if { $explore == 0 } {
incr recvind 1
# puts "$recvind $temp $temp2 $cost"
# puts "Source route: $lsrr"
# print to "trace"
# puts $outfile "$recvind $temp $cost"
# write temp all also (use beta field)
#puts $outfile "$recvind $temp $cost"
puts $outfile "$accTime [expr -$temp*log($rho) ] $cost"
flush $outfile
if { $recvind == 1 } {
puts $outfile2 "$accTime [expr -$temp*log($rho) ] $cost"
flush $outfile2
}
}
if { $explore == 1 } {
puts $outfile2 "$accTime [expr -$temp*log($rho) ] $cost"
```

```
flush $outfile2
}
# read list of hosts
set sa [hping getfield ip saddr $pck]
set da [hping getfield ip daddr $pck]
#puts $lsrr
set thost [split $lsrr /]
#puts "Hosts, pre-pre: $lsrl $thost $hops"
set hops [expr "$lsrl / 4 - 2"]
#set thost [lrange $thost 0 [expr $hops-2]]
set thost [lrange $thost 0 $hops]
# reverse host list
set ttmp {}
set ind [llength $thost]
#puts "Hosts, pre: $thost $ind $hops"
while {$ind>-1} {
lappend ttmp [lindex $thost $ind]
incr ind -1
}
set thost [lrange $ttmp 1 end]
# append source address of reply packet, i.e. target
lappend thost $sa
# print to check
#puts "Hosts: $thost"
if { $explore == 0 } {
# read the cost of each hop in the path
set tcost [lrange $tab 7 [expr 7 + $hops - 1]]
#puts "Costs: $tcost"
}
# run through the routing table to build nam events
# h = sent on link (leave node)
# d = drop on link
# set size [hping getfield ip length $pck]
set size 1000
set i 0
set shost $da
if { $explore < 3 } {
# forward path to the destination
foreach dhost $thost {
# set fixed time addition on each hop to control animation
set accTime [expr $accTime+$delay]
set dhost [lindex $thost $i]
puts $namfile "h -t $accTime -s $node($shost) -d $node($dhost) -e $size -i 0 -a 0"
flush $namfile
#puts "h -t $accTime -s $node($shost) -d $node($dhost) -e $size -i 0 -a 0"
```

```
set shost $dhost
incr i 1
}
}
# if not leading to an update: dropped at destination or at router
if { $explore > 0 } {
set accTime [expr $accTime+$delay]
set last [expr [llength $thost]-1]
set shost [lindex $thost $last]
if { $last > 1 } {
set dhost [lindex $thost [expr $last - 1]]
} else {
set dhost $da
}
if { $explore == 1 | $explore == 2 } {
puts $namfile "d -t $accTime -s $node($shost) -d $node($dhost) -e $size -i 5 -a $explore"
flush $namfile
#puts "d -t $accTime -s $node($shost) -d $node($dhost) -e $size -i 5 -a $explore"
}
#if { $explore == 3 | $explore == 4 | $explore == 6 | $explore == 7 }
if { $explore == 6} {
puts $namfile "l -t $accTime -s $node($shost) -d $node($linkdsrc) -S COLOR -c grey90 -o black "
flush $namfile
puts "l -t $accTime -s $node($shost) -d $node($linkdsrc) -c grey90 -S -o black DOWN"
}
if { $explore == 7 } {
puts $namfile "l -t $accTime -s $node($shost) -d $node($linkdsrc) -S COLOR -c black -o grey90 "
flush $namfile
puts "l -t $accTime -s $node($shost) -d $node($linkdsrc) -c black -o red -S UP"
}
} else {
set accTime [expr $accTime+$delay]
puts $namfile "h -t $accTime -s $node($shost) -d $node($target) -e $size -i 0 -a 0"
flush $namfile
set ttmp {}
set ind [llength $thost]
while {$ind>-1} {
lappend ttmp [lindex $thost $ind]
incr ind -1
}
set tcost [lrange $ttmp 1 end]
set thost [lrange $thost 0 end-1]
set thost [concat $da $thost]
set i [expr [llength $thost]-1]
```

```
set shost $sa
foreach n $tcost {
set accTime [expr $accTime+$delay]
set dhost [lindex $thost $i]
puts $namfile "h -t $accTime -s $node($shost) -d $node($dhost) -e $size -i 1 -a 1"
flush $namfile
#puts "h -t $accTime -s $node($shost) -d $node($dhost) -e $size -i 1 -a 1"
set shost $dhost
incr i -1
}
set accTime [expr $accTime+$delay]
puts $namfile "h -t $accTime -s $node($dhost) -d $node([hping outifa $target]) -e $size -i 1 -a 1"
flush $namfile
#puts "h -t $accTime -s $node($dhost) -d $node([hping outifa $target]) -e $size -i 1 -a 1"
}
}
# reread $inf after 1 ms
after 1 recAnts
}
# Set filter
hping setfilter $inf "udp"
after 1 genAnts
if { $data == 0 } {
# open trace file
set outfile [open "../trace-$antId" w]
set outfile2 [open "../trace-$antId-all" w]
after 1 recAnts
}
vwait forever
```

Tillegg I

Click-script

```
// Auto-generated by make-ant-router.pl
// vlan3 10.1.1.1 00:0A:0A:01:05:01
// vlan4 10.1.1.2 00:0A:0A:01:02:01
// vlan5 10.1.1.3 00:0A:0A:01:03:01
// vlan6 10.1.1.4 00:0A:0A:01:09:01
AddressInfo (
vlan3 10.1.1.1,
vlan4 10.1.1.2,
vlan5 10.1.1.3,
vlan6 10.1.1.4);
AlignmentInfo(
c0 4 2,
c1 4 2,
c2 4 2,
c3 4 2);
at :: AntTable(
vlan3/255.255.255.0 3 26,
vlan4/255.255.255.0 2 1,
vlan5/255.255.255.0 1 1,
vlan6/255.255.255.0 0 20);
// {}
elementclass prepareIP1 {$no, $ip |
/* Remove ethernet header, update source routing .. */
//input -> Paint($no) -> Strip(14) -> MarkIPHeader() -> SourceRoute($ip) -> output;
input -> Paint($no) -> Strip(14) -> MarkIPHeader() -> output;
};
elementclass prepareIP2 {$no, $ip |
/* Drops link-level broadcast and multicast packets, update route record, time-to-live etc. */
input -> DropBroadcasts
```

```
-> cp0 :: PaintTee($no)
-> gio0 :: IPGWOptions($ip)
-> FixIPSrc($ip)
-> dt0 :: DecIPTTL
-> fr0 :: IPFragmenter(1500)
-> atm :: AntTrafficMonitor(at, $no, 0.85)
-> output;
dt0[1] -> ICMPError($ip, timeexceeded) -> [1] output;
fr0[1] -> ICMPError($ip, unreachable, needfrag) -> [1] output;
gio0[1] -> ICMPError($ip, parameterproblem) -> [1] output;
cp0[1] -> ICMPError($ip, redirect, host) -> [1] output;
};
aat :: AntARPTable(vlan3, vlan4, vlan5, vlan6);
rt :: AntARP(aat);
ut::UpdateAATbl(aat);
utpath::UpdatePathTable(aat);
sr :: SourceRoute(0.0.0.0) -> rt;
ip :: CheckIPHeader(VERBOSE true, DETAILS true)-> DropBroadcasts
-> ant :: IPClassifier(dst udp port 51234, src udp port 51234, -);
ant[0] -> ut;
ant[1] -> BackwardAnt(at) -> sr;
ant[2] -> sr;
ut[0] -> fa :: ForwardAnt(at);
ut[1] -> rt;
// ARP responses are copied to each ARPQuerier and the host.
arpt :: Tee(5);
// Input and output paths for vlan3
c0 :: Classifier(12/0806 20/0001, 12/0806 20/0002, 12/0800, -);
FromDevice(vlan3) -> c0;
out0 :: Queue(200) -> todevice0 :: ToDevice(vlan3);
//FromDevice(vlan3) -> ToDump("/bison/openwrt/logs/from_10.1.1.1") -> c0;
//out0 :: Queue(200) -> ToDump("/bison/openwrt/logs/to_10.1.1.1") -> todevice0 :: ToDevice(vlan3);
arpq0 :: ARPQuerier(10.1.1.1, 00:0A:0A:01:05:01) -> out0;
c0[0] -> ar0 :: ARPResponder(10.0.0.0/8 00:0A:0A:01:05:01) -> out0;
c0[1] -> arpt;
c0[2] -> prepareIP1(0, vlan3) -> ip;
c0[3] -> Discard;
arpt[0] -> [1]arpq0;
// Input and output paths for vlan4
c1 :: Classifier(12/0806 20/0001, 12/0806 20/0002, 12/0800, -);
FromDevice(vlan4) -> c1;
out1 :: Queue(200) -> todevice1 :: ToDevice(vlan4);
//FromDevice(vlan4) -> ToDump("/bison/openwrt/logs/from_10.1.1.2") -> c1;
//out1 :: Queue(200) -> ToDump("/bison/openwrt/logs/to_10.1.1.2") -> todevice1 :: ToDevice(vlan4);
```

```
arpq1 :: ARPQuerier(10.1.1.2, 00:0A:0A:01:02:01) -> out1;
c1[0] -> ar1 :: ARPResponder(10.0.0.0/8 00:0A:0A:01:02:01) -> out1;
c1[1] -> arpt;
c1[2] -> prepareIP1(1, vlan4) -> ip;
c1[3] -> Discard;
arpt[1] -> [1]arpq1;
// Input and output paths for vlan5
c2 :: Classifier(12/0806 20/0001, 12/0806 20/0002, 12/0800, -);
FromDevice(vlan5) -> c2;
out2 :: Queue(200) -> todevice2 :: ToDevice(vlan5);
//FromDevice(vlan5) -> ToDump("/bison/openwrt/logs/from_10.1.1.3") -> c2;
//out2 :: Queue(200) -> ToDump("/bison/openwrt/logs/to_10.1.1.3") -> todevice2 :: ToDevice(vlan5);
arpq2 :: ARPQuerier(10.1.1.3, 00:0A:0A:01:03:01) -> out2;
c2[0] -> ar2 :: ARPResponder(10.0.0.0/8 00:0A:0A:01:03:01) -> out2;
c2[1] -> arpt;
c2[2] -> prepareIP1(2, vlan5) -> ip;
c2[3] -> Discard;
arpt[2] -> [1]arpq2;
// Input and output paths for vlan6
c3 :: Classifier(12/0806 20/0001, 12/0806 20/0002, 12/0800, -);
FromDevice(vlan6) -> c3;
out3 :: Queue(200) -> todevice3 :: ToDevice(vlan6);
//FromDevice(vlan6) -> ToDump("/bison/openwrt/logs/from_10.1.1.4") -> c3;
//out3 :: Queue(200) -> ToDump("/bison/openwrt/logs/to_10.1.1.4") -> todevice3 :: ToDevice(vlan6);
arpq3 :: ARPQuerier(10.1.1.4, 00:0A:0A:01:09:01) -> out3;
c3[0] -> ar3 :: ARPResponder(10.0.0.0/8 00:0A:0A:01:09:01) -> out3;
c3[1] -> arpt;
c3[2] -> prepareIP1(3, vlan6) -> ip;
c3[3] -> Discard;
arpt[3] -> [1]arpq3;
// Local delivery
toh :: Discard;
arpt[4] -> toh;
rt[4] -> local :: IPReassembler -> ping_ipc :: IPClassifier(icmp type echo, -);
ping_ipc[0] -> ICMPPingResponder -> [0]rt;
ping_ipc[1] -> EtherEncap(0x0800, 1:1:1:1:1:1, 2:2:2:2:2:2) -> toh;
// Forwarding path for vlan3
rt[0] -> pvlan3 :: prepareIP2(0, vlan3);
pvlan3[0] -> [0]arpq0;
pvlan3[1] -> rt;
// Forwarding path for vlan4
rt[1] -> pvlan4 :: prepareIP2(1, vlan4);
pvlan4[0] -> [0]arpq1;
```

```
pvlan4[1] -> rt;
// Forwarding path for vlan5
rt[2] -> pvlan5 :: prepareIP2(2, vlan5);
pvlan5[0] -> [0]arpq2;
pvlan5[1] -> rt;
// Forwarding path for vlan6
rt[3] -> pvlan6 :: prepareIP2(3, vlan6);
pvlan6[0] -> [0]arpq3;
pvlan6[1] -> rt;
// Routing from ForwardAnt
fa[0] -> pvlan3;
fa[1] -> pvlan4;
fa[2] -> pvlan5;
fa[3] -> pvlan6;
fa[4] -> utpath -> rt; // Packet is for this host
fa[5] -> rt;
```

Tillegg J

Innhold på vedlagte arkivfil

demo:

demo/makeplots

demo/plot-50000.gnu

demo/animasjonsfiler:

plot-50000.gnu

plot.html

t50000.nam

trace-50000

trace-50000-all

trace-50000.png

video_av_animering.ogg

demo/click/click_source:

click-1.5.0.tar.gz

demo/click/localElements:

antarp.cc

antarp.hh

anttable.cc

anttable.hh

anttrafficmonitor.cc

anttrafficmonitor.hh

backwardant.cc

backwardant.hh

forwardant.cc

forwardant.hh

sourceroute.cc

sourceroute.hh

updatepathtbl.cc

updatepathtbl.hh

updateaatbl.cc

updateaatbl.hh

aatable.cc

aatable.hh

demo/config:

install_key_files.sh
isup.sh
misc.sh
rebootall.sh
restartall.sh
startall.sh
stopall.sh

demo/config/package:

kmod-cifs_2.4.30-brcm-2_mipsel.ipk
kmod-shfs_2.4.30brcm+0.35-2_mipsel.ipk
libgcc_3.4.4-8_mipsel.ipk
robocfg_0.01-1_mipsel.ipk
uclibc++_0.1.11-2_mipsel.ipk

demo/config/script:

ant10.click
ant1.click
ant2.click
ant3.click
ant4.click
ant5.click
ant6.click
ant7.click
ant8.click
ant9.click
click
make-all-routers.pl
make-router.pl
nvram-fix.sh
S89vlanconf.sh
S98initClick11.sh
S98initClick12.sh
S98initClick13.sh
S98initClick14.sh
S98initClick15.sh
S98initClick16.sh
S98initClick17.sh
S98initClick18.sh

demo/hping:

makegnuplot
makeplots
run_hping.sh
run_nam.sh
start-plot-demo
tx_final.htcl

demo/openwrt_firmware:

openwrt-wrt54g-squashfs.bin

demo/rapport:

demo/rapport/antping:

D14.pdf

demo/rapport/prosjekt:

ProsjektRapport endelig versjon.pdf

Bibliografi

[1] Gnuplot. http://www.gnuplot.info/. Link kontrollert 20. mai 2007.

[2] Hping. http://www.hping.org. Link kontrollert 20. mai 2007.

[3] Linksys WRT54GL router. http://www.linksys.com. Link kontrollert 20. mai 2007.

[4] Nam - Network Animator. http://www.isi.edu/nsnam/nam/. Link kontrollert 20. mai 2007.

[5] OpenWRT, Linux for embedded devices. http://openwrt.org/. Link kontrollert 20. mai 2007.

[6] PPTP - Point-to-Point Tunneling Protocol. http://pptpclient.sourceforge.net/. Link kontrollert 20. mai 2007.

[7] Tcl - Tool Command Language. http://www.tcl.tk/. Link kontrollert 20. mai 2007.

[8] XORP the eXtensible Open Router Platform. http://www.xorp.org/. Link kontrollert 3. juni 2007.

[9] S. Carl-Mitchell and J.S. Quarterman. Using ARP to implement transparent subnet gateways. RFC 1027, October 1987.

[10] S. Cheshire, B. Aboba, and E. Guttman. Dynamic Configuration of IPv4 Link-Local Addresses. RFC 3927 (Proposed Standard), May 2005.

[11] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional IP routing protocols, 2002.

[12] Poul Heegaard and Ingebrigt Fuglem. Demostrator 1: Ant-Based Monitoring On Software Ip Routers. *IST Project Bison Programme Of The 5th Framework (1998-2002)*, 1:69, 2006.

[13] Nina Hesby, Poul E. Heegaard, and Otto Wittner. Robust Connections in IP Networks Using Primary and Backup Paths. 2004.

[14] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[15] Tore A. Kristiansen. Discovery protocol in AntPing. Technical report, NTNU, 2006.

[16] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[17] Rodiques, Gatrell, Karas, and Peschke. *IBM RedBooks TCP/IP Tutorial and Technical Overview*. Prentice Hall, 7th edition edition, 2002.