NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY, MATHEMATICS AND ELECTRICAL ENGINEERING

PROJECT ASSIGNMENT

Student's name: Tore A. Kristiansen

Course: TTM1 Access and Core Networks, advanced

Title: **Discovery protocol in AntPing**

Text: It is a great challenge to obtain sufficient, necessary, correct, and fresh high quality measurement data for proper network management and provisioning of service guarantees and differentiation in the Internet. As a supplement to current best practice in performance monitoring an idea is to observe temporal variables of a swarm intelligence system designed e.g. for solving reliable path management problems or optimizing routing in the Internet. 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. It is up to the audience to introduce the network dynamics. The demo invites the viewer to plug and unplug the links between nodes and to insert node failures while the performance of virtual paths is monitored. Variable traffic load may also be offered to the demo. A discovery protocol will allow adding new interfaces or links during the execution of the demo. This is currently not yet implemented and the assignment should investigate existing discovery protocols, and discuss how this can be integrated in existing solution. "Routing" (switching) based on MAC addresses instead of IP addresses in AntPing is one possible approach, and another is to have an underlying discovery protocol with IP subnet addressing. The following tasks should be considered;

1. Understand the fundaments the AntPing-routing and the underlying CE ants routing system
2. Investigate existing discovery protocols and evaluate them with respect to realization in the AntPing framework
3. Investigate potential extensions in AntPing to include discovery function
4. Implements lab environment and a discovery protocol (as much as time allows)
5. Verify and demonstrate the (simplified) discovery protocol

Deadline: 20 dec 2006

Handed in:

Carried out at: Department of Telematics

Supervisor: Ingebrigt Fuglem, Telenor R&I

Trondheim, 21 sep 2006

Poul Heegaard

Professor

FORORD

Denne rapporten er resultatet av mitt arbeid på prosjektet ”Discovery protocol in AntPing” ved ITEM NTNU høsten 2006. Prosjektet bygger på konseptet AntPing som er et arbeid utført av Poul Heegaard NTNU og Ingebrigt Fuglem, Telenor. Disse er også hhv faglærer og veileder til prosjektet.

Prosjektet så i utgangpunktet for meg ut til å være forholdsvis snevert men det har åpenbart seg en forholdsvis stor spennvidde av utfordringer. Alt fra praktisk konfigurering i Linux, implementering av C++ kode, forståelse og implementering av softwareruteren Click, ruting, rutingprotokoller og discoveryprotokoller. Prosjektet innholdt en opsjon om å implementere valgt løsning hvis tiden strakk til, noe den dessverre ikke gjorde. Men noe praktisk labarbeid ble allikevel utført. Labarbeidet hjalp meg først og fremst på forståelsen av hvordan Click og Linux ruteren virker hver for seg og i samspill. Jeg har brukt mye tid på å lese om mange typer protokoller og løsninger som ikke har hatt direkte relevans men som har hjulpet meg til å forstå sammenhenger.
Jeg har forsøkt å bruke norske ord der jeg synes det er hensiktsmessig og engelske faguttrykk der jeg føler for det. Blandingen kan man alltids diskutere men jeg har skimtet litt til hva norsk faglitteratur ellers har brukt.

Takk til Poul Heegaard og Ingebrigt Fuglem for gode råd, nyttig diskusjon og veiledning underveis.

Trondheim 20 des. 2006

Tore A. Kristiansen

INNHOLDSFORTEGNELSE

FIGURLISTE

TABELLISTE

SAMMENDRAG

AntPing er en prototypeimplementasjon av et distribuert og robust stifinner og monitoreringssystem, som baserer seg på en algoritme kalt CE (Cross Entropy) ant (maur). AntPing kjører på små hjemmerutere og visualiserer jobben som CE ant-algoritmen utfører med hensyn på å etablere, vedlikeholde og monitorere virtuelle stier. AntPing simulerer Telenors ”backbone” nettverk. Demoen visualiserer hvordan maur beveger seg og hvordan de blir droppet i nettverket. Nettverket er realisert vha Linksys WRT54G router flashet med OpenWrt Linux. Ruting og maurstyring er implementert vha Click software ruter. Oppgaven går ut på å utvide funksjonaliteten til AntPing til dynamisk å kunne addere nye linker mellom nodene som automatisk vil inngå i systemet. Jeg har sett på forskjellige teknologier som er mulige og nødvendige å ta i bruk under en implementering. Jeg har også forklart hva verktøyene som er brukt for å lage demoen har av begrensninger og muligheter. Jeg har skissert ulike løsninger og vurdert disse opp mot faktorer som er utledet under diskusjonen, og til slutt kommet frem til den anbefalte løsning som er kalt ”Statisk ruting – nodeadressering”. Dette er en enkel protokoll som baserer seg på at alle rutere i systemet hele tiden sendes ut en type ”hello” datagrammer til alle grensesnitt. Når linken er oppe registreres dette i motsatt ende og forbindelsen skrives inn i rutingtabellen til motstående ende. Det samme skjer motsatt vei. Når pakkestrømmen brytes tas innslaget bort fra rutingtabellen. Løsningen forutsetter også at alle grensesnitt på forhånd er definert opp med faste IP-adresser.

1 INNLEDNING

1.1 Bakgrunn

Det å fremskaffe nødvendige, korrekte og ferske nok trafikkdata for overvåkning og styring av IP-nettverk er en stor utfordring. I dag bruker man ofte aktiv monitorering vha ping og traceroute til overvåkning og dokumentasjon av tjenester og ytelse. Disse kan synes å ha behov for supplering eller forbedring. En av mulighetene man har til supplement er overvåkning og ruteoptimalisering er vha svermintelligenssystem. Dette systemet er laget for å observere endringer i trafikkbildet og ut fra dette generere pålitelige og mest mulig optimale rutingveier i Internet. For å vise dette har man implementert et prototype konsept basert på svermintelligens ruting i en software ruter. Konseptet er gitt navnet AntPing og demonstrerer prinsippene i et lite nettverk som simulerer Telenors ”backbone” nettverk. Dette arbeidet med svermintelligensruting er basert på forskning gjort ved Q2S og ITEM ved NTNU samt EU-IST prosjekt BISON hvor også Telenor FoU deltok. [2]

1.2 Problemstilling

Det er ikke implementert discovery protokoller i AntPing konseptet. For at man dynamisk under kjøring lett skal kunne addere nye linker eller grensesnitt, må det implementeres en eller flere discovery protokoller. Oppgaven går derfor ut på å lete, finne og sammenligne potensielle protokoller, for deretter å velge ut den eller de mest hensiktsmessige og til slutt implementere disse i eksisterende løsning. Eksisterende løsning er implementert vha Click modular router. Click kjører på små hjemmenettverksrutere Linksys WRT54G som er ”flashet” med uoriginal firmware OpenWRT Linux.

1.3 Avgrensninger

Arbeidet begrenser seg mot å løse et spesifikt problem nemlig implemtasjon av en discoveryprotokoll som kan brukes til autokonfigurasjon av grensesnitt og eventuelt noder i AntPing. Ytelse har så vidt vært berørt i form av at man ønsker minst mulig overhead. Utover dette må ytelsen vurderes ved en testimplementasjon. Andre hensyn eksempeøvis sikkerhet har heller ikke vært vurdert.

1.4 Definisjoner

Grensesnitt – i dette tilfellet Ethernet port
Link – forbindelse innenfor et fysisk nettverk
Subnett – nett som er mulig å nå uten å gå via en ruter
Lokalnett - nett som er atskilt fra det globale nettet og som bruker lokale ikkeunike IP-adresser. Kan kommunisere ut globalt vha NAT.

1.5 Oppbygning

Jeg vil først gi en fremstilling av teknologi som er aktuell for bruk i en implementasjon av de løsninger som jeg foreslår som discoveryprotokoll i AntPing. I kapitlet Laboppsett beskriver jeg verktøyene som brukes for å implementere, dette har leg lagt foran diskusjonen da jeg mener at virkemåten til disse i høy grad legger grunnlag for hvilken løsning som foreslås. Jeg vil deretter gjøre en diskusjon av muligheter og rammebetingelser som løsningen må eksistere i, og ut fra dette vil det fremkomme en del utvalgskriterier. Disse utvalgskriteriene bruker jeg til å vekte ulike løsninger opp mot hverandre som til slutt vil munne ut i en anbefalt løsning.

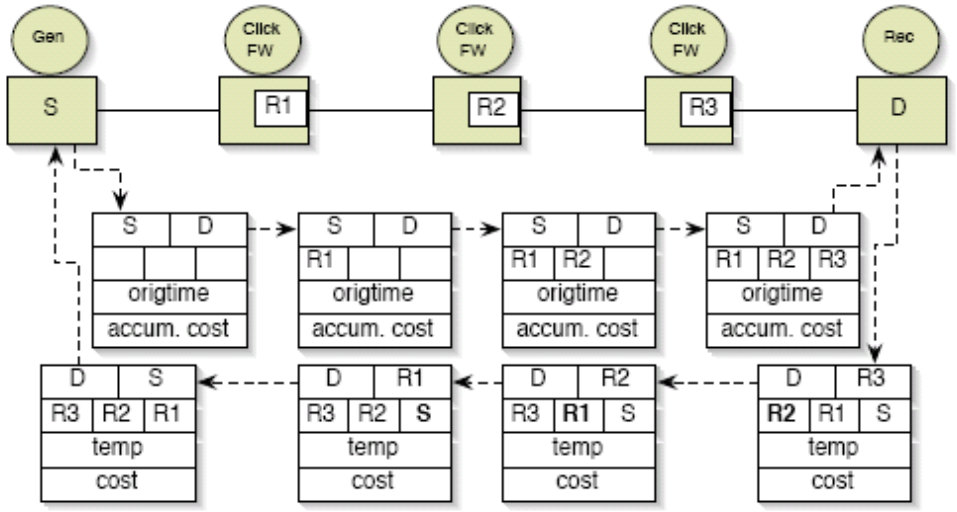
2 AntPing

AntPing [2] er en prototype implementasjon av et distribuert og robust stifinner og monitoreringssystem. Ant ping er realisert vha Click modular router og Linksys

- Kjernen i AntPing er en stokastisk rutingalgoritme basert på såkalte maur kalt CE ant. Se part II i [2]
- AntPing registrerer ”forward route” (route records) datagrammene blir sendt tilbake samme vei vha source routing.
- AntPing legger til ekstra statistikk i ruterene som er implementert vha Click. IP-adressene på hvert hopp er lagt til i IP header og time stamp er lagt til i payload.
- AntPing sender UDP pakker på port 51234.
- Det er flere typer datagram i AntPing (normal, exploration, forward, backward, update)

AntPing krever pr i dag en egen generatorenhet som genererer CE ant dvs UDP datagrammer, en mottaker av disse samt at det må være implementert støtte for å oppdatere og rute/videresende disse i ruterene. Figur 1 viser et eksempel på hvordan AntPing virker. Et CE ant datagram blir generert i kilden S. Tidspunktet denne ble generert blir addert til payload, 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 CE ant-typen er explorer. Etter at neste hopp er bestemt vil datagram-headern bli oppdatert med dette grensesnittets adresse og deretter vil den bli videresendt på valgte grensesnitt. Dette skjer for hver ruter som CE ant er innom helt til den når målet D. Ved D vil det blir undersøkt om dette er den første CE ant som har ankommet fra S. Hvis dette er tilfellet blir dette registrert som en ny forbindelse, kostnaden bli kalkulert, temperaturen addert til payloaden og backward CE ant datagram vil bli sendt tilbake til S. Hvis ikke vil det blir gjort de samme

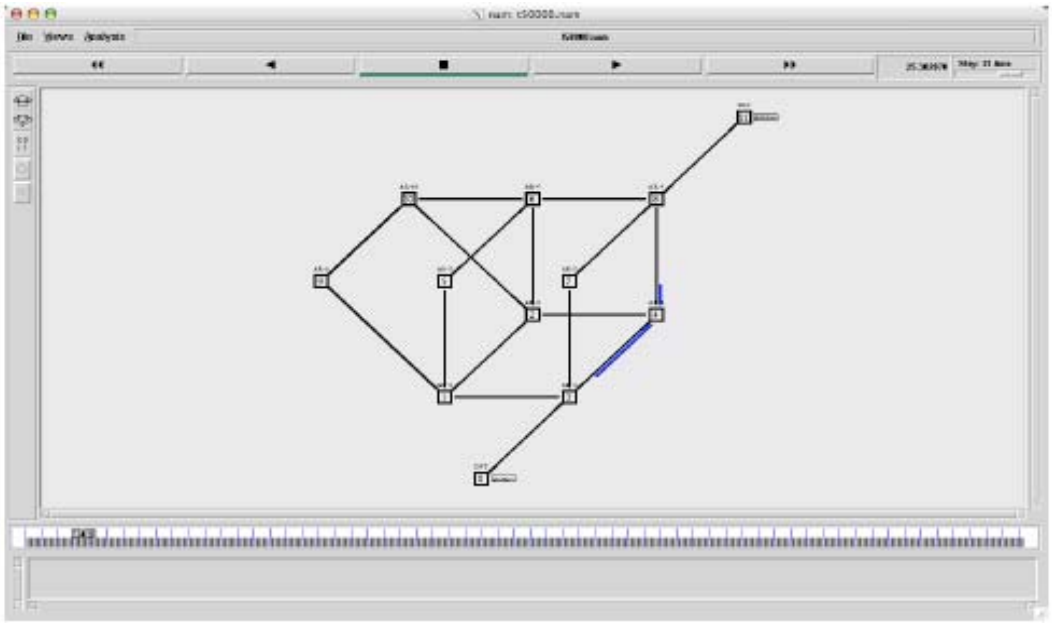
kalkulasjoner og addering til payload med man vil sende et backward update datagram tilbake tilS. Tilbaketuren dvs Backward (update) ant vil benytte source route. Dette for å kunne oppdatere eller skrive nye luktestoffverdiene for denne ruten i alle ruterene på tilbakeveien. Når maur er ankommet til S igjen blir nødvendig monitoreringsdata skrevet til en loggfil, hvis dette ikke allerede er gjort i D.



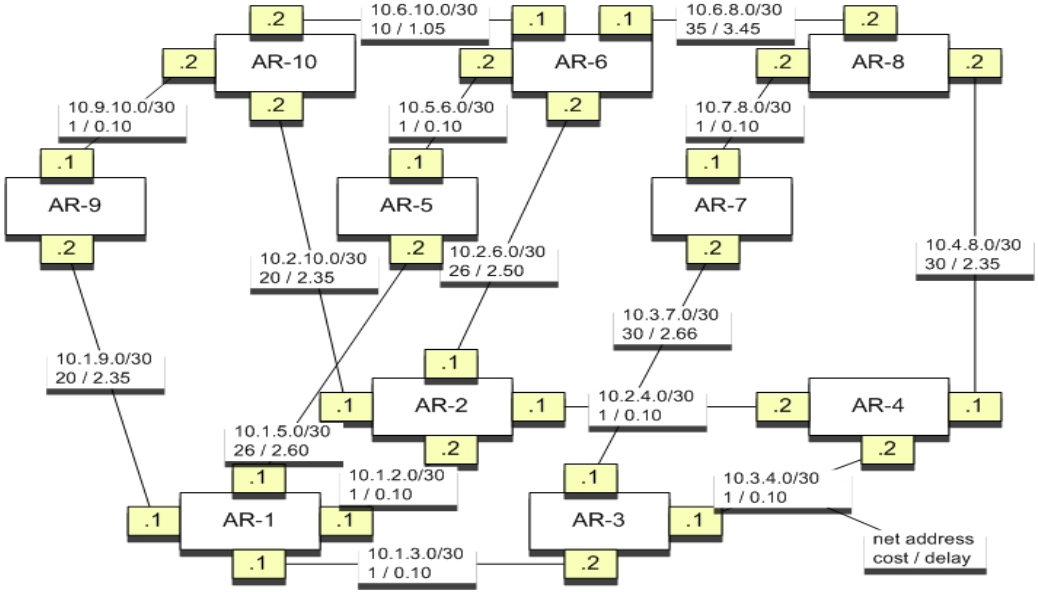
Figur 1 Rute oppdatering og "source routing" for "backward ants" [2]

Man har nå ved hjelp av eksplorer maur kartlagt de beste rutingveiene i nettet. Man vil deretter gå over i en normal fase der man vedlikeholder de oppdagede rutingveiene bare avbrutt av perioder med explorer maur for å oppdage eventuelt nye stier. Det fine med denne metoden er at man i stedet for bare å bruke den beste ruten fra kilde til mål, vil bruke alle stier der man har regnet ut en vektet fordeling basert på tiden som det tar for maurene å finne frem fra S til D på mulige stier. Eksempelvis hvis vi ved S har 2 mulige stier til D, med vekting (0,8-0,2). Den første gjennom grensesnitt A den andre gjennom grensesnitt B. Da vil man sende normal maur (dvs vedlikeholdsmaur) med stokastiskfordeling lik 0,8 og 0,2 gjennom hhv A og B. Det samme vil skje i

ruter nr 2 i stien. Her vil normal mauren bli rutet med stokastiskfordeling (til de eventuelle grensesnitt som fører til D) som følge av de luktestoffene som tidligere backward update ant har lagt igjen. For å monitorere og visualisere dette, er det i AntPing demo inkludert støtte for animering gjennom Network Animator (NAM) [15] som er brukt som et visualiseringsverktøy sammen med Network Simulator (ns).[16]. For en mer detaljert beskrivelse vises det til [2], der man bla beskriver hvordan man eliteselekterer ant for å fjerne stiene med lengst delay.



Figur 2 Snapshot av en NAM animasjon, med kilden tilkoblet til node 8 og målet til node 3. [2]



Figur 3 Fysisk topografi AntPing realisert vha Linksys WRT54G [2]

3 DISCOVERY OG RUTINGMEKANISMER

3.1 TCP/IP Protokollstakk

Applikasjonslaget
Transportlaget
Nettverkslaget
Linklaget

Linklaget

Dette laget er grensesnittet til den virkelige fysiske nettverks-hardware. Dette grensesnittet kan tilby pålitelig eller upålitelig levering, og være pakke- eller bitstrømorientert. TCP/IP spesifiserer ingen bestemt protokoll, noe som gjør det mulig å bruke mange forskjellige teknologier her. Stort sett kan man bruke hvilket som helst nettverksgrensesnitt her. Men den mest vanlige teknologien i dag er Ethernet med tilhørende protokoller.

Nettverkslaget

Gir et virtuelt nett og skjuler den virkelige fysiske nettverkstopologien for lagene over. Internet Protocol (IP) er den viktigste protokollen på dette laget. ARP og ICMP regnes også å tilhøre dette laget. Protokollen tilbyr en forbindelsesløs protokoll som verken tilbyr pålitelighet til lagene over eller forventer pålitelighet fra laget under. Laget tilbyr heller ikke flytkontroll eller feilretting. Laget tilbyr stort sett å rute datagrammer mellom fysiske nett eller mellom noder.

Transportlaget

Transportlaget tilbyr ende til ende forbindelse ved å levere data fra en applikasjon til den andre samarbeidende applikasjon. Man kan ha mange forbindelser oppe samtidig. For det meste blir Transmission Control Protocol (TCP) eller User Datagram Protocol (UDP) brukt. TCP tilbyr en forbindelsesorientert protokoll med pålitelig datalevering, flytkontroll og metningskontroll. UDP tilbyr en upålitelig forbindelsesløs transport. Dette medfører at de applikasjoner som bruker denne, vil selv måtte forestå nødvendig flyt- og metningskontroll. UDP er brukt av applikasjoner der man tåler noe tap av data og der data som ikke kommer frem i tide er uinteressant. Eks sanntidssystemer som video-streaming. Her vil en billedramme som ankommer etter at det skulle vært vist uinteressant og det er bare en belastning for nettet å få denne retransmittert som TCP ville ha gjort.

Applikasjonslaget

Brukes av programvare som bruker TCP/IP for kommunikasjon. En applikasjon her er en brukerprosess som trenger kommunikasjon for å samarbeide med en annen applikasjon vanligvis befinner seg på en annen enhet. Eksempler her er Telnet eller File Transfer Protokoll (FTP). Grensesnittet mellom applikasjon og transportlaget er portnummer og socket.

3.2 Linklaget

3.2.1 Auto-Negotiation

Automatisk konfigurasjon av Ethernet linker tilbys av Auto-Negotiation [3] protokollen som er definert i Ethernet standarden. Denne delen av standarden er implementer i twisted-pair (TP) linker og men har liten utbredelse i Ethernet fiber. Dette fordi fiber har en stor variasjon i bruk av lyskilder og optiske bølgelengder noe som ikke er interoperabele. Unntaket er Gigabit Ethernet fiber automatiske signaleringssystem som virker på alle fiberoptiske linker.
Auto-Negotiation gjør det mulig for Ethernet grensesnitt å utveksle informasjon om grensesnittenes konfigurasjonsmuligheter og kapasiteter over en link. Dette muliggjør at man vha autokonfigurasjon oppnår den best mulige operasjonsmodus over en link.
Auto-Negotiation:
- er designet til å virke kun over et linksegment.
- starter ved link initiering, enten ved at en enhet blir slått på eller ved at en Ethernet kabel blir tilkoblet.
- bruker sitt eget uavhengige signaleringssystem designet for TP kabling. Disse signalene er sendt en gang ved initiering.
- monitorerer hele tiden linkstatus og vil bli trigget hver gang en den får indikasjon på at en link er kommet opp.

3.3 Nettverkslaget

3.3.1 Address Resolution Protocoll (ARP)

ARP [4] er ansvarlig for å konvertere høyere lags protokoll adresse (IP) til fysisk nettverksadresse (MAC). På et enkelt fysisk nettverk er de enkelte noder kjent vha deres fysiske hardwareadresse. En symbolsk adresse, IP i dette tilfellet, er ikke kjent for driverne til de fysiske grensesnittene. Derfor trenger vi en protokoll (ARP) som oversetter IP-adressene til den fysiske adressen på

mottakernes grensesnitt. ARP bruker en tabell som kalles ARP-oppslagstabell eller noen ganger ARP-cache.
Hvis applikasjon vil sende en pakke til en IP destinasjon vil IP-ruting mekanismen først detektere IP-adressen til neste hopp til pakken og til hvilken hardware enhet den skal bli sendt til. ARP prøver først å slå opp i tabellen for å finne MAC-adressen til neste enhet basert på IP-adressen til denne. Hvis den finner denne så blir pakken sendt. Hvis ikke blir pakken forkastet (ARP går ut fra at høyere lags protokoll vil prøve å retransmittere denne pakken) og det blir generert og sendt ut en ARP-broadcast-melding, som kanskje vil resultere i en oppdatert ARP-tabell.

3.3.2 PROXY-ARP

Grunnleggende metode

Som beskrevet i avsnittet ovenfor fungerer en vanlig ARP-forespørsel 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 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.[5] Når en ARP-forespørsel er mottatt kan ruteren ved å slå opp i rutingtabellen fastslå om den vet rutingveien til en node. Hvis den finner noden i rutingtabellen vil den på samme måte som over respondere på denne.
Dette forutsetter at man ikke benytter statisk ruting med default gateway definert da dette alltid vil gi respons på en ARP.
Noden som sender en ARP vil normalt bruke den første responsen som kommer og forkaste de andre.

3.4 Transportlaget

3.4.1 Rutingprotokoller

Statisk Ruting

I statisk ruting [6] er rutingtabellen manuelt konfigurert av nettverksadministratoren. Han er ansvarlig for å finne ruter gjennom nettverket og manuelt å konfigurere alle rutingenheter i nettverket. I små nettverk med lite endring og redundans er dette en forholdsvis enkel sak å administrere, men når nettverket blir større og mer komplisert vil det være en utfordring å finne beste rutingvei. En endring i topologien vil kreve mye manuelt arbeid da statisk ruting ikke har noen mekanismer for å tilpasse seg disse. Statisk ruting har noen fordeler ellers ville det ikke blitt brukt, bla gir statisk ruting ingen overhead og er lite krevende med hensyn på CPU tid i ruteren. I tilegg er det en oversiktlig måte å ordne ruting på i små nettverk.

Distance vector ruting

Distance vektor (DV) [6] 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 kost fra seg selv til alle kjente destinasjoner i nettet. Denne verdien representerer den samlede distansen/kostnaden til denne stien. Andre stier som fører til destinasjonen og som har lavere kostnad vil bli foretrukket og den av alle stier som har lavest kostnad vil bli foretrukket. 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 og oversiktlige å feilsøke. De fungerer best i små enkle nettverk med liten redundans. Uansett, det er 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 konvergeringstid. I store kompliserte nettverk vil konvergenstiden, ved bruk av en DV rutingprotokoll, være betydelig. Dette medfører at tiden etter en linkfeil til at rutingtabellene er rekalkulert vil være alt for stor, og i mellomtiden vil rutingen være upålitelig. (Route Information Protokoll) RIP er en populær DV rutingprotokoll.

Link state routing

[6] Veksten i størrelsen og kompleksiteten i de senere år har gjort det nødvendig med mer robuste rutingprotokoller. Disse protokollene hadde som formål å bøte på svakhetene ved DV rutingprotokoller. Algoritmen som disse 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 naboruterene feks IP-adresse, subnetmaske, 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 kost 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 ruterene.
- Ruterene 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 krever større og kraftigere rutere for å fungere tilfredstillende.

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

3.5 Applikasjonslaget

3.5.1 Dynamic Host Configuration Protocol (DHCP)

DHCP [7] er en mekanisme laget for å tilby unike IP adresser innenfor et nettverk. Man har en server som tildeler IP-adresser innenfor et på forhånd gitt adresseområde. Den holder orden på hvem som har fått tildelt disse adresser og garanterer unikhet innenfor et oppgitt tidsrom.
DHCP er en klient-serverprotokoll der man setter opp en server som inneholder og tilbyr adresse- og konfigurasjonsdata, og der den enkelte node i nettet blir satt opp til å forespørre denne serveren om disse dataene, ved oppstart og eventuelt gjenta dette med visse mellomrom.

DHCP tilbyr to tjenester:

- Varig lagring av nettverksparametre for nettverksklienten. Metoden går ut på at DHCP serveren er gitt en nøkkelverdi som entydig identifiserer klienten. Det naturlige her skulle man tro er MAC adressen men den er ikke alltid unik så her kan eksempelvis IP subnettadresse sammen med MAC-adresse være den unike nøkkelen. En klient kan deretter utføre spørring mot DHCP serveren for å motta nettverksparameterne som er lagret der, eks ved oppstart.

- Midlertidig eller permanent tildeling av IP-adresser til klienter. Grunnmekanismen for tildeling av IP-adresser er enkel: En klient forespør bruken av en adresse for en tidsperiode. DHCP serveren(e) garanterer å ikke reallokere innenfor den aktuelle tidsperioden. Klienten kan også be om forlengelse eller tillate frigivelse av adressen.

3.5.2 IPv4 Link-lokal adresse

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 [1]nettverk på en singel link som inneholder noen få noder. Man bør ikke overstige 1300 noder. 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 [2] adresser ikke er tilgjengelige, eksempelvis ad-hoc eller isolerte nettverk.

Adressevalg

Når en node skal konfigurere en Link-lokal adresse velger vha 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] Små ad-hoc nettverk er i dette tilfellet definert til <1300 noder, noe som er stort nok i denne sammenheng

[2] Rutbare adresser defineres her som alle gyldige unicast IPv4 adresser utenfor 169.254/ prefikset. Dette gjelder alle offentlige og private IP-adresser inkludert eks 10/8 prefikset men ikke loopback adresser for eksempel 127.0.0.0.

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 man når et maks antall forsøk. Dette makstallet er bestemt av implementasjonen. Teste om adressen er i bruk kan gjøres vha en ARP som setter 0.0.0.0 som avsenderadresse, for å unngå at grensesnittet i nettverket oppdaterer sine ARP-tabeller før det har blitt klarlagt at adressen er unik. Dette kalles en ARP-probe. Når dette er 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**.**

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, og man ikke har sett andre ARP-pakker innenfor det siste DEFEND_INTERVAL sekunder, kan den prøve å forsvare adressen ved å broadcaste med en enkelt ARP-melding med sin egen MAC-adresse og IP-adresse. Men hvis dette ikke er den første ARP-pakke med adressekonflikt innenfor DEFEND_INTERVAL må noden øyeblikkelig slutte å bruke denne adressen og konfigurere opp en Link-lokal adresse som beskrevet ovenfor. Dette 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 dette vil forekomme meget sjeldent.

Konfigurasjonsregler

Link-lokal adresser kan ikke forwardes av en ruter og kan kun kommunisere med enheter på samme link både 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 adresse basere seg på algoritmer som knytter seg til grensesnittet og ikke noden. Dette for å minske risikoen for å generere like adresser.

4 Laboppsett

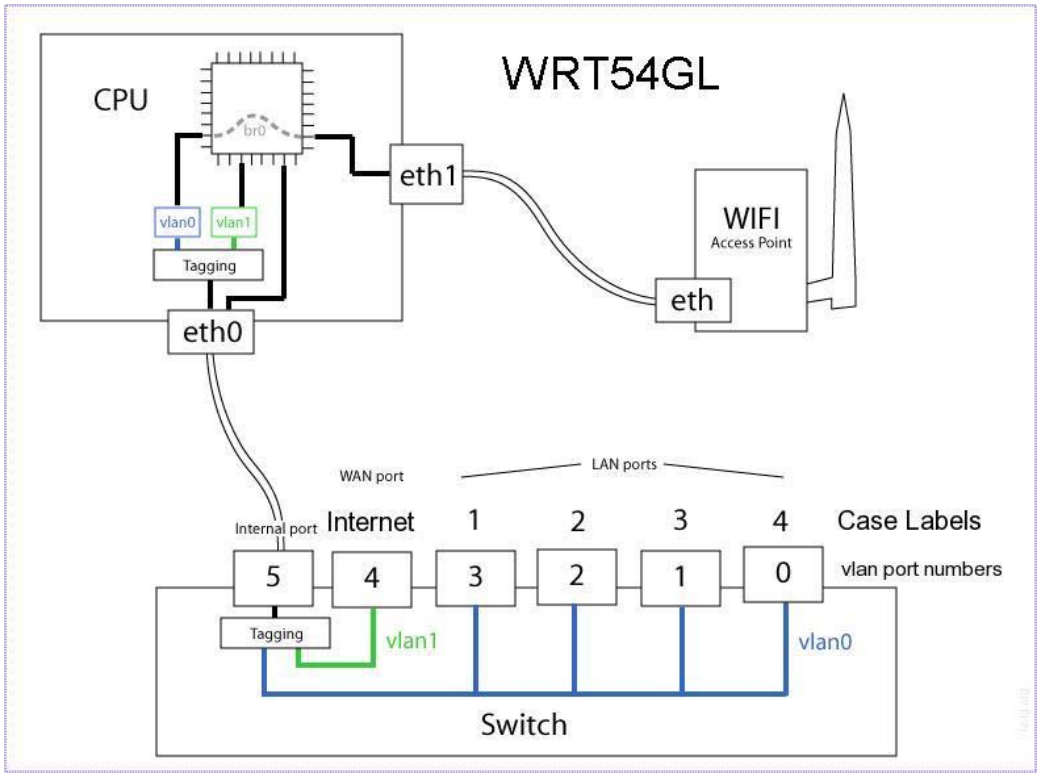
4.1 Hardware

I oppgaven ble det benyttet følgende utstyr:

- **Linksys WRT54GL** ruter der original firmware ble erstattet med OpenWRT Linux White Russian RC5.

Figur 4 Linksys WRT54GL v1.1 [12]

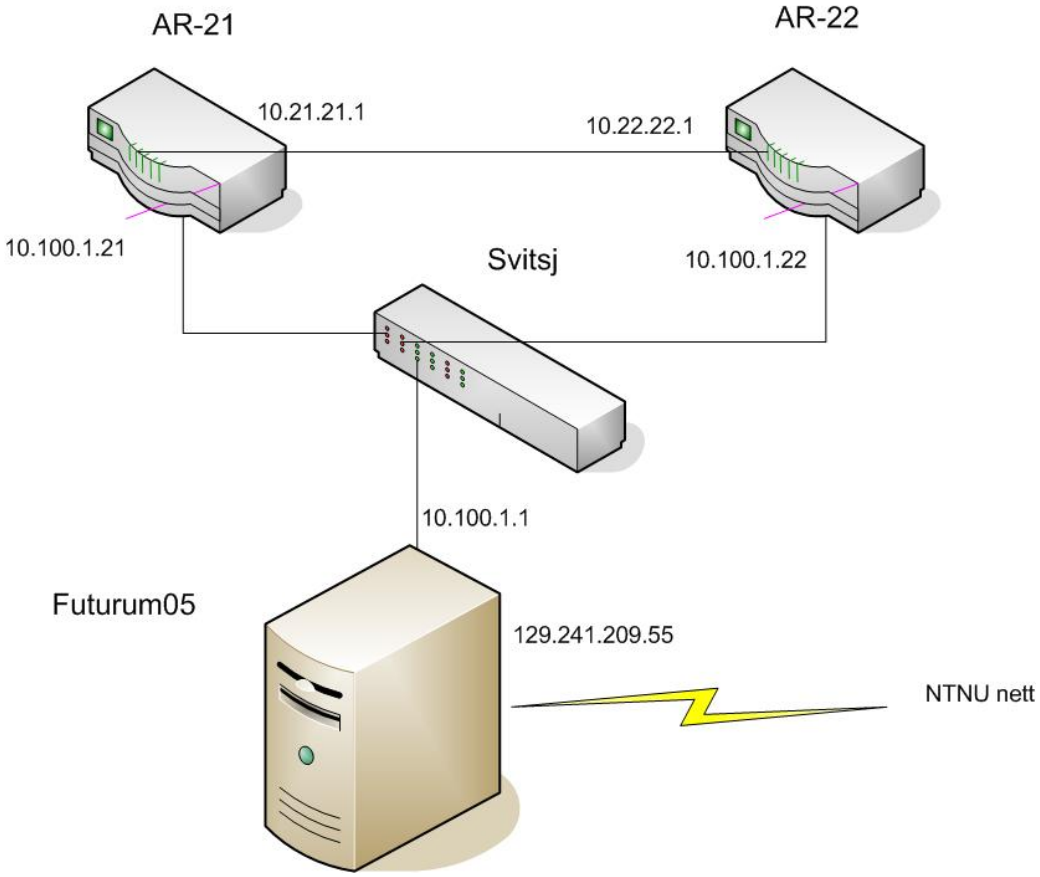
Linksys WRT54GL består av en Ethernet svitsj, et trådløst aksesspunkt og en ruter som knytter disse sammen. Enheten bruker Linux som operativsystem. Dette gjør at den lett kan flashes med uoriginal firmware, i dette tilfellet OpenWRT Linux. Den er egentlig laget for å dele en høyhastighets datalinje eller internettforbindelse. Ruteren har 5 tilkoblingsmuligheter i form av Ethernet utganger på baksiden. WAN 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. Se Figur 5



Figur 5 Linksys WRT54GL router layout [13]

- **Dell Pentium 4 PC** med Debian Linux til konfigurering og styring av ruterene.
- **3com Dual speed 8 port switch.**

Dette ble koblet opp slik som Figur 6 viser.



Figur 6 Testoppsett som er brukt til å teste ut Click-konfigurasjoner

4.2 OpenWRT

OpenWRT [13] er beskrevet som Linux distribusjonen for lukkede (embedded) enheter. I stedet for å lage en enkelt 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 legge til filer, og til å bruke andre applikasjoner en det leverandøren tilbyr. OpenWRT er rammeverket for å kunne bygge en applikasjon uten å måtte bygge en komplett ny firmware, eksempelvis for små hjemmerutere. Det er også en komplett enkel 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 har ikke muligheten for å kunne kompilere en ny firmware i selve enheten. For å kunne produsere en ny firmware må man derfor bygge opp firmware vha en prosess kalt ”cross compiling” som i sin helhet forgår utenfor den aktuelle enheten, her WRT54GL. Denne prosessen er ikke helt enkel og derfor er flashing av allerede ferdigkompilert firmware dvs et image, den enkleste måten å installere Linux i en slik enhet på. Open WRT tilbyr ferdig kompilerte imager til mange forskjellige rutere.[14]

4.3 Click Modular Router

Introduksjon

Click [1]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 protokoller eller endre eksisterende 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.

Man kan kjøre Click i user mode eller i kernel mode. Når man kjører i user mode som er tilfellet med AntPing vil Click kjøre som en hvilket som helst annen applikasjon og den innbygde ruteren i OpenWRT vil kjøre som normalt. Man kan også kjøre i kernel mode, da vil Click ruteren erstatte den opprinnelige Linuxruteren. I user mode kan man eventuelt søke å utnytte rutingen som foregår i Linuxkjernen eller man kan stenge den helt bort ved eventuelt å definere andre IP-adresser i Linuxdriverne eller benytte den innebygde brannmuren.

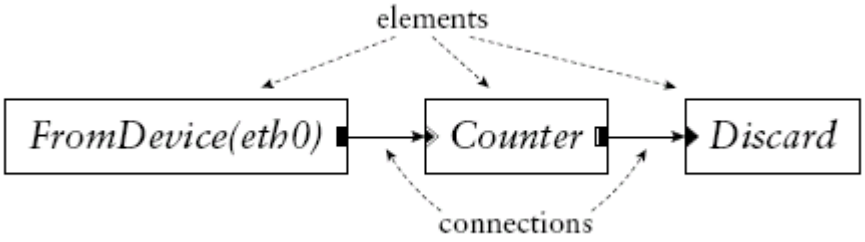
Programmeringsspråk

Click er implementert vha et begrenset C++ språk. Det er forbudt å implementere klasser fra standard template libraries (STL). Click tilbyr sin egen versjon av ofte benyttede STL klasser med lignede interfacec som STL klassene.

Click ruter konfigurasjon

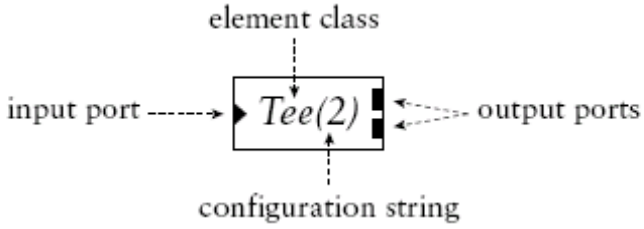
En Click ruter er bygd sammen av pakkeprosessmoduler kalt elementer. Elementene er subklasse av elementklassen. Individuelle elementer implementerer enkle rutingfunksjoner som for eksempel pakkeklassifisering, køing og grensesnitt mot nettverksenheter. En ruterkonfigurasjon består av elementer satt sammen i system vha av et Click-skript 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 som gir muligheten til lett å konfigurere ruteren eller skrive nye elementklasser.
Et enkel Click-skript kan se ut som dette:

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



Figur 7 Visuell presentasjon av Click-skriptet over [1]

Konfigurasjoen over, er en enkel konfigurasjon for å telle pakker som kommer inn på et grensesnitt (eth0).



Figur 8 Et eksempel element [1]

En elementklasse beskriver elementets data layout og oppførsel.
Et element er en subklasse av elementklassen og kan bestå av en eller flere innganger og/eller en eller flere utganger avhengig av datalayout og konfigurasjons stringen.

```
myTee::Tee(2)
```

”myTee” er et element av elementklassen Tee som har konfigurasjonsstrengen 2. Hvis man går inn i dokumentasjonen til klassen Tee ser man at Tee lager kopier av pakkene og fordeler det på et antall utganger gitt av konfigurasjonsstrengen 2. ”myTee” produerer altså 2 identiske pakker av en pakke og leverer disse til 2 utporter.

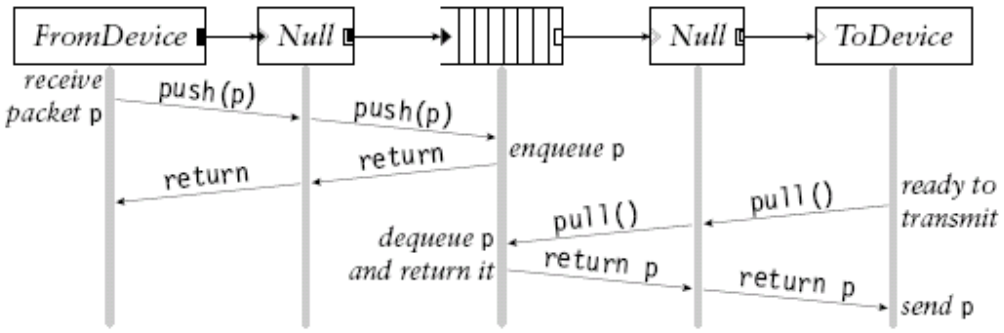
```
myTee::Tee(2);
myTee[0] -> Counter –> Discard;
myTee[1] -> Paint (0) ->……
```

Her ser man hvordan man lett kan duplisere en pakke og fordele den på 2 forskjellige utporter til ulike formål.

Push og Pull

I Click blir pakker ”pushed” inn i ruteren fra en input enhet og ”pulled” ut av ruteren av en utenhet.
Dette innebærer at det er to typer porter. Push og Pull. Push-portene er representert ved hjelp av en fylt trekant eller firkant og Pull-portene er åpne trekanter eller firkanter. Man kan også ha såkalte agnostiske porter som kan

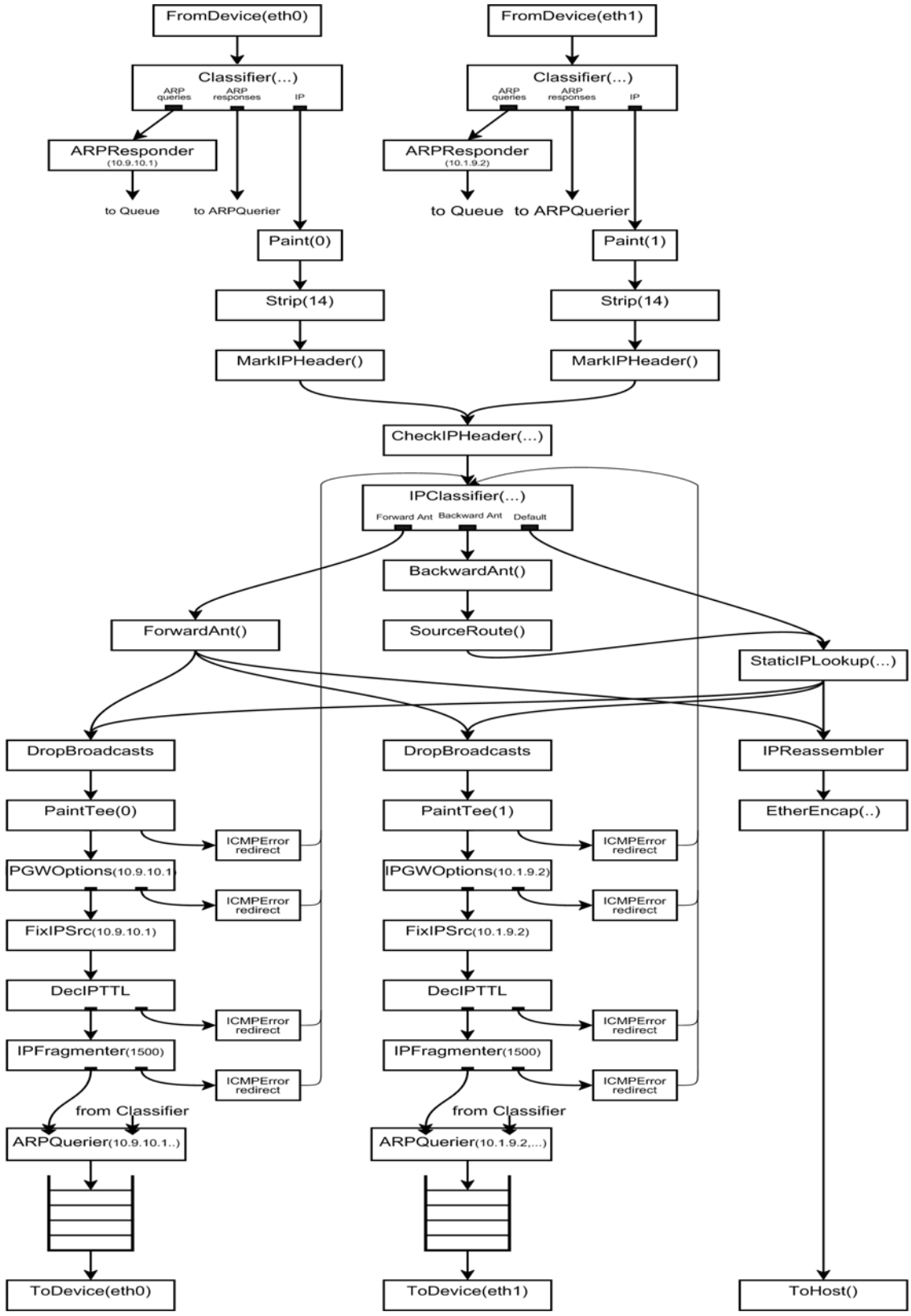
være både Push eller Pull. Push-portene er hendelsesbasert og Pull-portene baserer seg på polling. Push utporter må være koblet til Push innporter og vise versa.



Figur 9 kontrollflyten i en software ruter [1]

Hvis vi ser på eksemplet ovenfor vil en pakke ankomme FromDevice og blir levert til Null som er et agnostisk element. Utgangen til Null er koblet til inngangen til et køelement. Køelementet er sentralt her. Den mottar pakker og lagrer dem i en kø og leverer fra seg pakker når den blir ”pollet” fra andre siden.

Som man ser kan man bygge opp rutere og svitsjer og konfigurere disse på mange forskjellige måter. Dette er årsaken til at Click lett kan brukes til eks å implementere eksperimentelle rutingalgoritmer i testnettverk. Her vist vha AntPing se figur 10.



Figur 10 Click flyt skjema av AntPing [2] se også appendiks 1 tilhørende Click-skript

5 Arbeid utført

5.1 Installering og konfigurering av Dell PC

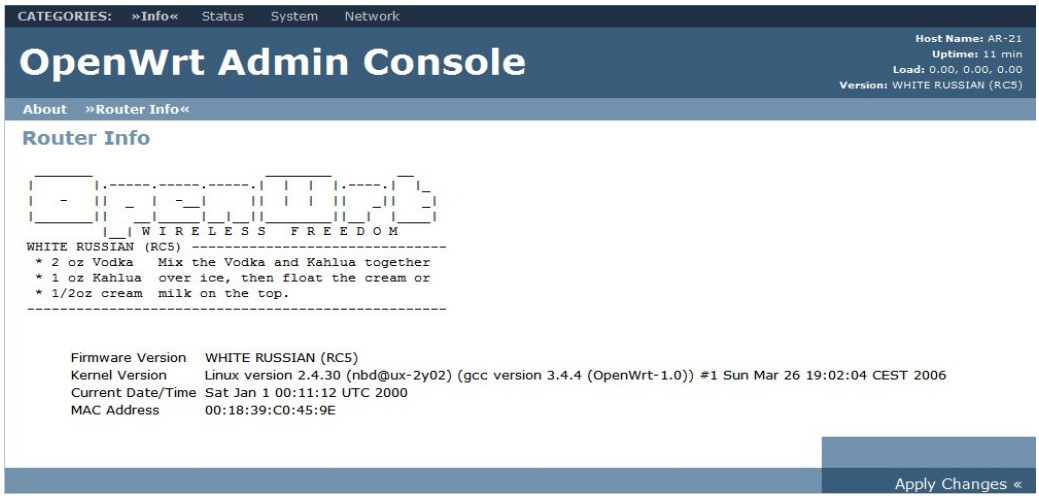
Installerte Debian Linux, serverutgaven med kjerne 2.6.xx . Serverutgaven ble valgt da den hadde ferdig installert SSH server som sørger for enkel remote tilgang til PC'en. PC'en fikk først tilgang til nettet ved NTNU via innbygde ethernetgrensesnittet og den fikk IP-adressen via DHCP. Dette viste seg å være ugunstig da det av ymse årsaker ble utført hyppige omstarter av denne. Det ble da ordnet så med en fast IP-adresse. Installerte deretter to nettverkskort til på denne i tilegg til det ethernetgrensesnittet som allerede eksisterte. For å få alle nettverksgrensesnittene til å komme opp riktig måtte disse bli konfigurert opp i */etc/network/interfaces*. PC'en ble gitt navnet Futurum05.

5.2 Installering av OpenWRT på Linksys WRT54GL

Ved oppstart fikk jeg utlevert to rutere som skulle brukes til prosjektet. Første steg var å installere OpenWRT Linux på disse. Jeg baserte meg på seksjon E i AntPing dokumentasjonen [2].
Lastet først ned OpenWRT RC5 som var den siste stabile versjonen på dette tidspunkt. Firmware kommer i to utgaver og jjfs2. Squashfs gir et rootsystem i ruteren som er read-only og man vil dermed ikke kunne overskrive disse filene i motsetning til jjfs2 som er overskrivbar. Samtidig er squashfs litt komprimert og tar dermed mindre plass. Denne utgaven blir også vurdert som litt mer stabil og lettere å rette opp ved en systemkrasj. Dette er de egenskapene som antas mest gunstig for prosjektet og derfor ble squashfs utgaven valgt. For å kunne starte på oppgraderingen måtte jeg i følge oppskriften gjøre noen forberedelser. Først måtte jeg aktivere boot_wait funksjonen på ruteren. Dette er en funksjon som utsetter boot'en av ruteren noen sekunder. Det er i denne tidsluken fra man slår på ruteren til boot'en kommer i gang at man skal kunne oppgradere ruteren med ny firmware. Samtidig kan denne funksjonen gi en mulighet til å rette opp en eventuell uheldig firmware-oppdatering som har omgjort ruteren til en "dørstopper". Å aktivere denne funksjonen på WRT54GL-utgaven av ruteren

etter de kokeoppskrifter jeg fant i dokumentasjonen, viste seg umulig. Etter å ha lest på diverse lenker på OpenWRT forum [17] og ”googlet” litt fant jeg ut at man kunne bruke grafiske grensenittet til den originale firmware og vha dette, flashe den uoriginale OpenWRT firmware som om den skulle være en ny oppdatert utgave av den originale. [18]. Da var det en enkel oppgave å få dette til. Selve flashingen av firmware via grafisk grensesnitt ble gjort fra en ordinær laptop med Windows XP OS. Se figur 12 og 13. At man på kjøpet fikk oppskriften på en White Russian drink kan jo regnes som en kuriositet.

Figur 11 Konsoll før flashing av firmware i en orginal WRT54GL ruter.



Figur 12 Konsoll etter flashing av OpenWRT

5.3 Konfigurering av ruteren

All konfigurasjon foregikk via SHH fra egen PC til Futurum05, fra denne ble det satt opp ny SSH forbindelse til ruter.
Før WAN-utgangen ble oppkonfigurert måtte jeg koble meg til ruteren via en LAN-utgang. For å konfigurere den ble det først modifisert, så kopiert til ruterens katalog /etc og deretter kjørt et skript kalt nvram-fix.sh. Dette setter IP-adressen på WAN-utgangen og navnet på ruteren som de viktigste dataene. Dette gjorde at jeg nå kunne nå ruteren fra WAN-utgangen og koble denne til Futurum05. Deretter kopierte jeg inn og installerte noen bibliotek og programvare inn på ruteren, uclibc++, mipsel, robocfg og libgcc. Kopierte deretter inn et modifisert skript vlanconf.sh til ruterens katalog /etc og kjørte dette remote. Dette skriptet konfigurerer bla adressene til lan-utgangene (kalt vlan). Kopierte deretter den ferdig kompilerte click fila som inneholder alle nødvendige elementer for å kjøre AntPing. For at ruteren skal bli satt opp som en AntPing-ruter må man kjøre click-skript som definerer opp pakkeflyten i Click og hvordan ruteren skal konfigureres. Se for øvrig kapitlet om Click eller [1].

5.4 Testing

Siden min oppgave hadde som mål at den skal munne ut i en praktisk implementasjon, hadde jeg behov for å få en bedre forståelse av hvordan samspillet mellom Click og OpenWRT fungerte i praksis. For å gjøre dette brukte jeg to enkle click-skript hhv Pakkegenerator og Pakkemottak som jeg kopierte ut på de to ruterene som jeg hadde fått utlevert kalt hhv AR-21 og AR-22 seg Figur 6. Jeg sendte UDP pakker ut på vlan3 AR-21 og mottok disse på AR-22 vlan3. vlan3 tilsvarer Lan port1 i Figur 5.

Pakkemottak

```
FromDevice(vlan3) -> ToDump("mottat") -> Print(ok) -> Discard;
```

Pakkegenerator

```
TimedSource(DATA \<
45 00
00 28 00 00 00 00 40 11 62 bd 0a 03 02 01 0a aa
16 02 13 69 13 69 00 14 c1 3b 55 44 50 20 70 61
63 6b 65 74 21 0a
>, INTERVAL 1)
 -> StoreData(12, \<0a 03 01 01>)
 -> StoreData(28, UDP-pakke)
 -> MarkIPHeader
 -> DecIPTTL
 -> SetIPChecksum
 -> SetUDPChecksum
 -> Queue
 -> EtherEncap(0x0800, 1:1:1:1:1:1, 2:2:2:2:2:2)
 -> ToDump("after")
 -> ToDevice(vlan3);
```

Det viste seg at uansett hvordan jeg varierte MAC-adressen eller IP-adressen så ble disse pakkene mottatt i andre enden. Siden man allerede hadde konfigurert opp disse grensesnittene vha OpenWRT kommandoene roboconfig og nvram, til faste IP-adresser og MAC-adresser, og som ikke stemte med de adressene jeg brukte i Click konfigurasjonen min, trekker jeg den konklusjon at Click og OpenWRT Linux OS opererer helt uavhengig av hverandre. Hva betyr det? For det første så er det unødvendig å konfigurere opp alle grensesnittene i OS med IP-adresser Kun nødvendige IP-adresser for å få kontakt med OS'et er nødvendig. Slik som AntPing er konfigurert nå vil de bare skape problemer med pakker som både blir behandlet i OS og i Click og vi får dublerte pakker. Sett opp mot min oppgave som handler om autokonfigurasjonen i Click, blir det derfor kun adresseinformasjonen som er laget i Click som blir interessant å eventuelt forandre på ved en konfigurering av IP-adresser på grensesnittene. Siden disse adressene leses inn når konfigurasjonsfila kjøres og dette er statisk er disse adressene umulig å endre dynamisk under kjøring. Hvis man ønsker en dynamisk konfigurering av IP-adressene må dette skje ved hjelp av en dynamisk tabell inne i et nyskrevet element.

6 DISKUSJON

For å kunne rute en datapakke til en bestemt node må man ha en unik adresse og en mulig rutingvei til denne. For at dette skal være mulig må denne adressen være unik enten globalt eller innenfor sitt lokale rutingdomene og være tilknyttet denne noden statisk eller dynamisk. Adressen og rutingveiene kan enten bli konfigurert manuelt (statisk) eller man kan innføre en autokonfigureringsmekanisme for adresser og en rutingprotokoll for rutingveiene. Hvis adressene er dynamiske må nødvendigvis rutingveiene også være det, men man trenger ikke bruke statiske rutingveier bare fordi adressene er statiske. Årsaken er at det kan oppstå feil og da kan det ofte være en fordel at rutingveiene oppdateres automatisk.

Disse funksjonene kan implementeres på mange måter og tilpasses karakteristikaen til nettverket. Eksempelvis vil ikke et nettverk som er satt opp i kontrollerbare omgivelser i et kontorområde ha de samme behov for automatiserte rutiner som et mobilt ad-hoc nettverk (MANET) har. Man trenger nødvendigvis ikke å tildele dynamiske adresser selv om nettverket som sådan er dynamisk, men linkene og evnen til å oppdage hvilke adresser og linker som er med i nettverket må nødvendigvis være dynamisk. Løsninger på utfordringen rundt det å danne nye adresser har i stor grad konsentrert seg om MANET men heller ikke her har autokonfigurering av nettverksadresser vært en sentral del av problemstillingen. Man har i stor grad konsentrert seg om å finne effektive rutingveier i skiftende omgivelser og i liten grad sett på det å generere gyldige IP-adresser som en problemstilling. Man har forutsatt at IP-adressen var statisk og ferdig konfigurert før noden ble medlem av nettet. [9] Det man finner av litteratur konsentrerer seg naturlig nok om det konseptuelle, mens denne oppgaven skal dreie seg om muligheten og egnetheten til å implementere dette i Click på en enkel måte uten å bruke for mye ressurser. Derfor vil det bli vektlagt at løsningen er forholdsvis **enkel å implementere**. Løsningen bør også være tilpasset den **eksisterende konfigurasjonen** i AntPing.

Linklaget har allerede en velfungerende autokonfigurering/discovery protokoll kalt Auto-Negotiation. Den er stort sett implementert på alt ethernettutstyr som

bruker Twisted-Pair (TP) kabel. Dette er også brukt i Linksysruteren som brukes av AntPing. Denne protokollen starter når enheten blir slått på eller en TP kabel blir koblet mellom to grensesnitt. Siden MAC adressen allerede er tilstede vil oppgaven til protokollen være å fremforhandle riktige fysiske parameterverdier slik at kommunikasjonen skal gå best mulig.
Utfordringen er å autokonfigurere *Nettverkslage*t slik at dette skal kunne tilby kommunikasjon innenfor AntPing etter at den fysiske linken er kommet opp. Her kan man oppleve motstridende krav mellom ønsket funksjonalitet og fleksibilitet på den ene siden og ønsket om enkelhet på den andre siden. AntPing skal fraktes rundt i koffert på forskjellige møter der man ”live” skal kunne demonstrere konseptet. Man ønsker derfor å gjøre dette så enkelt å frakte og å montere opp som overhodet mulig. For at det skal være lett å frakte har man derfor valgt noen enkle hjemmerutere til oppgaven. Disse har sine minne og ytelsesbegrensninger, og derfor vil dette kunne sette en stopper for at mange ønskelige egenskaper blir bygget inn i ruteren.
En enkel tilnærming på problemstillingen er å på forhånd definere opp et nettverk med nødvendige rutingveier. I tillegg kan man konfigurere opp alle porter med IP-adresser, og etter hvert som disse tas i bruk og oppdages vha enkle ”neighbour discovery” mekanismer og deretter addere rutingveier. Dette medfører at AntPing’s IP-adressestruktur må endres. Litt mer avansert men mer tilpasset AntPing’s nåværende adressestruktur vil være å konfigurere grensesnittene med IP-adresse først når linken tas i bruk, for så å addere rutingvei med tilhørende subnettmaske. Dette vil tilfredsstille kravet til enkelhet men er kanskje ikke særlig skalerbart og fleksibelt.
For at AntPing skal være enklest mulig å frakte og koble opp ønsker man et mest mulig **desentralisert** system der hver ruter er mest mulig autonom og nettverket minst mulig avhengig av ekstern maskinvare. Hvis dette skal oppfylles og adressekonfigurasjonen skal være fullstendig automatisk og fleksibel, vil dette fordre bruk av en duplicate address detection (DAD) protokoll i tillegg til en rutingprotokoll. Altså en mer avansert implementasjon. Samtidig vil dette gjøre at nettverkt vil bli svært **skalerbart**, noe som vil kunne

muliggjøre en testing av AntPing i større skala. En ruting og DAD protokoll vil nødvendigvis medføre at det genereres en del overhead noe som negativt kan påvirke ytelsen til de etter forholdene enkle ruterene.
Ved å bruke DHCP vil man kunne redusere kompleksiteten og samtidig **gi lite overhead** i forhold til en DAD funksjon, men dette kan gå ut over ønske om autonome rutere og mest mulig desentraliserte funksjoner.

Løsningene som blir foreslått nedenfor vil som antydet over bli vurdert opp mot følgende faktorer.

- **enkel å implementere**
- **tilpasset eksisterende konfigurasjon**
- **desentralisert**
- **gi lite overhead**
- **skalerbart**

7 MULIGE LØSNINGER

7.1 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 vha faste regler som man bruker når man konfigurer opp IP-adressen på grensesnittene og subnettet mellom to grensesnitt se Figur 3. Disse reglene er entydige og når de følges for å generere adresser, vil sannsynligheten for å få dublerte IP-adresser være lik null. Feilen må i så fall knyttes til en feil enten i algoritmen eller i forhåndsprogrammeringen. Vi kan derfor se bort fra problemet med dupliserte adresser. Disse enkle reglene kan utnyttes for å autogenere adresser på grensesnitt og subnett. Siden man ikke vil få problemer med dupliserte adresser trenger man heller ikke å sende ut discovery meldinger over hele nettet for å fastslå at adressen er entydig. Ulempen er at hvis man følger de nåværende på forhånd oppsatte reglene vil AntPing ikke kunne overstige 255 enheter. En

annen ulempe er at hver ruter må på forhånd være konfigurert opp med et unikt nummer fra 1-255 som man bruker til å generere de unike IP-adressene fra. I tillegg vil ikke denne løsningen være særlig heldig hvis man eksempelvis trekker ut pluggen til default rutingvei og setter den i et annet ledig grensesnitt.

7.2 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 der ruterene er nummerert og adressene på grensesnittene er laget ut fra dette. I stedet for å lage subnettverk med to grensesnitt som har adresse utgått fra begge ruternavn gir man her grensesnittene adresse i forhold til sitt ruternr og grensesnittnr feks ruter AR-25 vil få adressene 10.1.25.til 10.1.25.4. Denne løsningen blir mer skalerbar da man har større adressespenn å ta av da man kan adressere 10.1.1. 1-4 til 10.255.255.1-4. Ergo vil man kunne få 255x255 =65025 noder som skulle være tilstrekkelig for de fleste formål. Her kan grensesnittene være konfigurert med IP-adressene på forhånd eller man konfigurerer etter behov vha en enkel algoritme og ruternr. Dette medfører at så lenge ruternummeret er entydig gitt, vil man ha entydige IP-adresser og man som løsningen over slipper å sende ut discovery-meldinger i hele nettet og dermed vil overheaden holdes på et minimum. Løsningen er mer skalerbar men lider av den samme svakhet som *Statisk ruting – subnettadressering* med hensyn på utplugging av default rutingvei.

7.3 Random adresse med rutingprotokoll

Her går man ut fra konseptet med bruk av Link-lokal adresser. Link-lokal adresser har i denne sammenheng den svakheten at de ikke er rutbare dvs de kan kun 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 har jeg valgt å bruke rutbare lokale adresser dvs 10.x.x.x. I tillegg for å kunne fastslå at den adressen som er generert er unik må en bruke en eller annen DAD mekanisme. En forholdsvis enkel mekanisme er å bruke ARP, som med et tillegg som heter proxy-ARP vil kunne nå hele AntPing. Dette forutsetter at det brukes en ruting protokoll slik at rutingtabellen i nabo noden inneholder en oversikt over hele nettverket. Dette vil spare nettverket for overhead da man benytter seg av allerede innbygde mekanismer. Andre discoveryprotokoller kan også brukes her. Fordelene med denne løsningen er at noden ikke trenger å bli tildelt noe entydig nummer og navn på forhånd. Den er meget fleksibel, og hvis en node svikter er det bare å erstatte den med en tilsvarende som inneholder samme programvare. Ulempen er at den gir mer overhead og at man må implementere en rutingprotokoll, noe som gir en mer avansert test og implementasjonsprosess.

7.4 DHCP konfigurering

I denne fjerde løsning 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 da en slik tabell over tildelte og mulige adresser ikke skulle ta stor plass. 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. Ulempen er at den er sårbar for brudd i rutingveier og har er ”singel piont of failure”. I tilegg må den som forrige løsning ha implementert en rutingprotokoll. I Click er det allerede laget noen elementer til bruk for DHCP, hvis det er plass i ruterens minne skulle dette normalt kunne gå greit å implementere.

7.5 Vurdering av mulige løsninger

Jeg har vurdert opp de mulige løsninger opp mot de faktorene som ble utledet i kapittel 4. Jeg har valgt å bruke en enkel grovskala bedømnings syntaks der – gir -1, 0 gir 0 og + gir 1. Vurderingen er da utelukkende gitt de kriterier som er utledet og disse er igjen basert på rammen for denne oppgaven. Ingen av faktorene er vektet mer enn andre.

	Statisk ruting – subnettadressering	***Statisk ruting - nodeadressering***	***Random adresse med rutingprotokoll***	***DHCP konfigurering***
Desentralisert	+	+	+	-
Enkel å implementere	***0***	+	-	-
Gi lite overhead	+	+	-	***0***
Tilpasset eksisterende konfig	+	-	-	-
Skalerbar	-	***0***	+	+
TOTAL EGNETHET	***2***	***2***	***-1***	***-2***

Tabell 1 Vurdering av mulige løsninger sett opp mot utledede faktorer

7.6 Delkonklusjon

Som vi ser er det to løsninger som ligger alene på toppen og skiller seg klart fra de andre løsningene. Dette gjør at jeg velger å gå videre og se litt nærmere på disse to (Statisk ruting – subnettadressering og Statisk ruting – nodeadressering) og forkaste de andre. For å kunne skille disse to alternativene og gi en anbefaling må jeg utrede begge disse alternativer nærmere.

8 SKISSE TIL IMPLEMENTERING AV LØSNINGER

Det er to løsninger, grovsortert etter faktorene, som skiller seg klart positivt ut og for å kunne anbefale en av disse må jeg sammenligne disse to opp mot hverandre innenfor rammen av AntPing, dvs hva kan best og lettest implementeres i Click som kjører på OpenWRT i en Linksys WRT54G ruter. Jeg vil derfor skissere en litt mer detaljert løsning på begge to for deretter å trekke en endelig konklusjon. For begge løsninger referer man til Figur 13.

8.1 Statisk ruting - subnettadressering

Denne skissen forutsetter at man ikke adderer ekstra rutere til nettverket men kun adderer ekstra linker..

1)Man plugger in TP-kabelen i begge rutere og grensesnittene konfigurerer automatisk opp en link vha ethernet standardprotokoll Auto-Negotiation.

2) Grensesnittene sender ut en IP pakke på linken som mottas av grensesnittet på andre enden som sier jeg er ruter nr *mottatt* Dette skjer peridisk eksempelvis hvert sekund ved hjelp av en pakkegenerator (LinkUpPacketgenerator)

3) Den mottatte pakken fra det andre grensesnittet skilles ut og behandles i *LinkUp*. Hvis dette er første pakken som er mottatt etter at linken er kommet opp vil a) skje hvis ikke b). Hvis pakken uteblir c).

a)

Man vil motta pakken og sjekke om linken er oppe fra før ved å sjekke om det er registrert en timestamp fra tidligere pakker. Hvis ikke

vil man resette en timer som måler hvor ofte man mottar pakker, plukke ut info ruternr ”*mottatt*” og deretter generere en IP-adresse vha syntaksen

If ”*mottatt*” < ”*egen*”

IP-adresse =10.mottatt.egen.2

Netverksadresse = 10.mottatt.egen.0

else

IP-adresse =10.egen.motatt.1

Nettverksadresse=10.egen.motatt.0

Oppdatere nettverksgrensesnittet med IP-adresse.
Deretter vil man oppdatere rutingtabellen LinearIPLookup med en ny rutingvei.
Forkaste pakken.
b)
Man vil motta pakken, sjekke om linken er oppe fra før og hvis så er tilfellet vil man resette timer og forkaste pakken.
c)
Når timer har talt til 3 sek vil man:
Fjerne IP-adresse fra nettverksgrensesnittet
Deretter vil man oppdatere LinearIPLookup dvs fjerne rutingveien til dette grensesnittet.

8.2 Statisk ruting - nodeadressering

Denne skissen forutsetter at man ikke adderer ekstra rutere til nettverket men kun adderer ekstra linker. Her kan man velge å på forhånd konfigurere opp grensesnittene med faste IP-adresser da dette vil forenkle implementasjonen av løsningen. Man slipper da å forholde seg til den dynamiske koblingen mellom IP-adresse og grensesnitt (se kapittel 7.4 om sammenhengen mellom Click og HW grensesnittet), og det vil bli kun innslaget i rutingtabellen som trenger å bli oppdatert når linker går opp og ned.
1)Man plugger in TP-kabelen i begge rutere og grensesnittene konfigurerer automatisk opp en link vha ethernet standardprotokoll Auto-Negotiation.
2) Grensesnittene sender ut en IP pakke på linken som mottaes av grensesnittet på andre enden som sier jeg er ruternr *"mottatt"* jeg sender på portnr *"P"*. Dette skjer periodisk hvert sekund ved hjelp av en pakkegenereingsenhet (LinkUpPacketgenerator)
3) Den mottatte pakken fra det andre grensesnittet skilles ut og behandles i LinkUp-elementet. Hvis dette er første pakken som er mottatt etter at linken er kommet opp vil a) skje hvis ikke b). Hvis pakken uteblir c)

a)

Man vil motta pakken og sjekke om linken er oppe fra før ved å sjekke om det er registrert en timestamp fra tidligere pakker. Hvis ikke
vil man resette en timer som måler hvor ofte man mottar pakker, plukke ut info ruternr *"mottatt"* og *"P"*. Deretter genereres en IP-adresse og rutingvei vha syntaksen:

X=egetruternr/255+1

Y=egetruternr*(X-1)

IP=10.X.Y.P

Gjøre det samme ved hjelp av mottatt info for å generere rutingvei.

X=mottatt/255+1

Y=mottatt*(X-1)

Rutingvei=10.X.Y.P

Deretter vil man oppdatere rutingtabellen LinearIPLookup med en ny rutingvei.

Forkaste pakken.

b)

Man vil motta pakken, sjekke om linken er oppe fra før og hvis så er tilfellet vil man resette timer og forkaste pakken.

c)

Når timer har talt til 3sek vil man
oppdatere LinearIPLookup dvs fjerne rutingveien til dette grensesnittet.

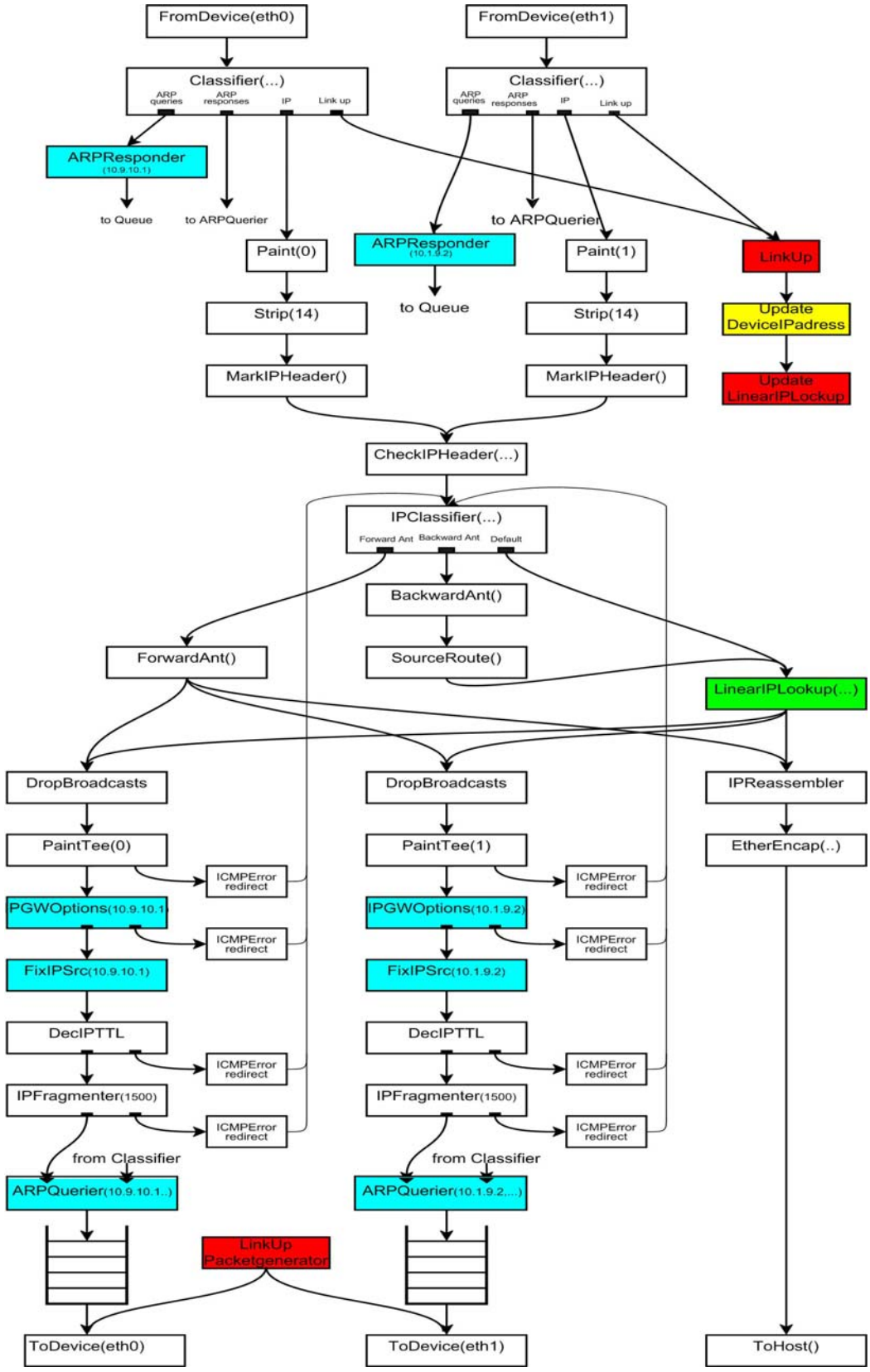
8.3 Vurdering av løsninger

Begge løsninger er forholdsvis enkle løsninger og vil være overkommelig å implementere i Click, men det som gir en utfordring er dynamisk å endre koblingen mellom IP-adresse og grensesnitt, som er foreslått i den første løsningen *Statisk ruting – subnettadressering.* Dette kan implementeres i begge løsningene men som man ved løsning *Statisk ruting – nodeadressering* ikke trenger å implementere. Grunnen til dette er at det ved på forhånd er kjent hvilke IP-adresser de enkelte grensesnitt må få tildelt når linken kommer opp.

Dette er ikke tilfellet for *Statisk ruting – subnettadressering*, som benytter den motstående nodens ruternr til å generere grensesnittets dynamiske IP-adresse. Utfordringen med dynamisk å endre IP-adresse i skyldes at Click skapes vha av en konfigurasjonsfil som kjøres en gang ved oppstart. Slik som AntPing er konfigurert nå benytter den seg av et element som heter AdressInfo der koblingen mellom grensesnitt og IP-adresse defineres opp i forbindelse med initiering av ruteren, dvs kjøring av konfigurasjonsfila.
I enkelte element som trenger dette leser man inn IP-adressen direkte som en konfigurasjonsstring.
Disse bindingene er statiske og kan ikke endres før en ny initiering foretas. Ny initiering kan gjøres mens ruteren kjøres men da trenger man en logikk utenfor Click som styrer dette, noe som ikke er vurdert som en løsning i denne oppgaven. En mulig løsning på dynamisk IP-adressetildeling er at alle element som trenger å forholde seg til koblingen mellom grensesnitt og IP-adresse, leser dette fra en tabell som styres av et element som kun har dette som oppgave. Da kan man skrive til og lese fra denne tabellen som dynamisk opprettholder koblingen mellom IP-adresse og grensesnitt. Siden et slikt element ikke er laget fra før må lage dette samt at de elementer som tidligere har fått IP-adressen inn vha en statisk konfigurasjonsstring også må skrives om til å lese inn denne fra denne tabellen. Hvis vi ser på Figur 13, som viser et Click flytdiagram på hvordan de to løsningene kan se ut kan vi legge merke til følgende: Sammenlignet med den originale AntPing (Figur 10) er elementer markert med rødt (LinkUp, UpdateLinearLookup) nye elementer som må skrives uansett valgt løsning. I tillegg er elementet som er markert med grønt (LinearIPLookup) bare en ny rutingtabell som erstatning for StaticIPLoookup. Den nye rutingtabellen finnes allerede som ferdigskrevet element og har muligheter for dynamisk å endre rutingveiene i motsetnining til StaticIPLookup. De elementer som er markert med blått er de som man må skrive om hvis dynamisk konfigurering av grensesnittene skal implementeres. I tilegg må man lage elementet i gult (UpdateDeviceIPaddress).

8.4 Delkonklusjon

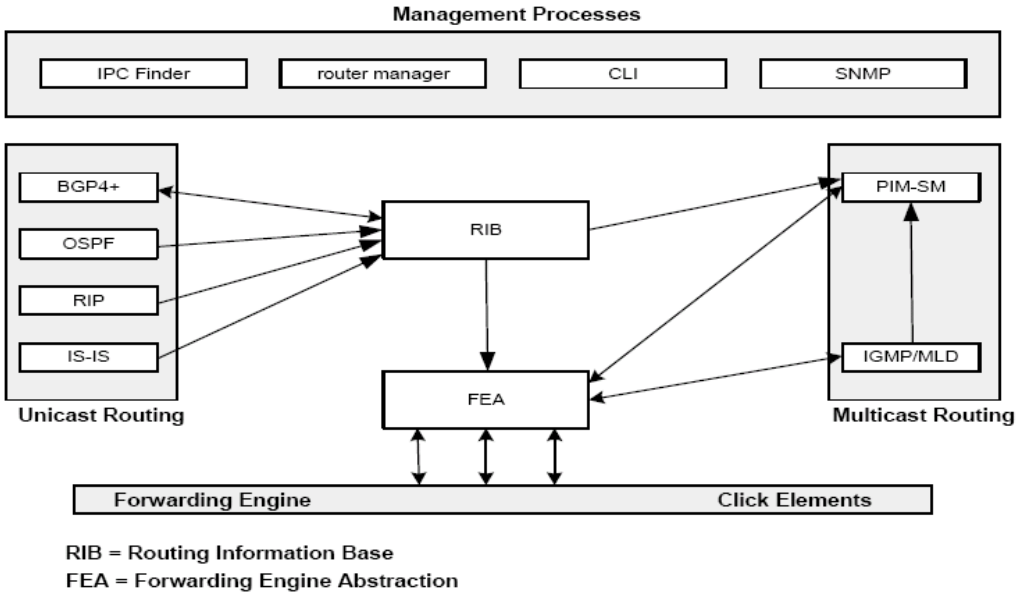
Valgte løsning vil være ***Statisk ruting – nodeadressering.***
Denne løsningen vil være den enkleste å implementere. Selv om den bryter litt med AntPing implementasjonens måte å adressere grensesnittene på anses dette som et lite problem. Det som er vektlagt er at dette er en langt en enklere løsning å implementere enn den *Statisk ruting – subnettadressering.*



Figur 13 Nytt Click flytskjema i henhold til løsningsforslag

9 Videre arbeid

Videre arbeid må bli fysisk å implementere valgt løsning og teste denne for å kunne fastslå at dette virker etter hensikten. Videre vil det være hensiktsmessig å kunne vurdere om det er mulig eller ønskelig å gå videre noen av de andre løsningene med sikte på en implementasjon. Hvis man går videre med løsning 3 og 4 vil disse kreve implementasjon av en rutingprotokoll. Denne kan enten implementeres direkte i Click eller vha annen programvare som ligger på et lag over Click og bruker Click bare som en forwarding-maskin. Se Figur 14.



Figur 14 Programvare med rutingprotokoler som bruker Click som forwarding-maskin. [10]

Eksempler på dette kan være Zebra [11] eller Xorp (eXtensible Open Router Platform) [10].

En annen løsning basert på funnene i kap 5.4 kan være å la Click håndtere kun AntPing og la OpenWRTmed tilleggspakker håndtere resten. Dette kan gjøres ved å la en brannmur i OpenWRT sperre ute alle UDP pakker port 51234 (ant), og kun behandle alle vanlige. Motsatt vil man da la AntPing behandle kun ant pakker. Fordelen med dette er at det allerede er utviklet en del rutingprotokoller som kan kjøre sammen med OpenWRT. Da får man et system der OpenWRT

ruteren behandler vanlige pakker vha forskjellige kjente rutingprotokoller og AntPing behandler sine pakker vha sin stokastiske rutingalgoritme og man kan deretter sammenligne disse.

10 KONKLUSJON

I denne oppgaven har jeg sett på mulige løsninger for å oppdage nye linker som plugges inn mellom noder i AntPing. Det fines et utall måter å tilnærme seg og løse denne oppgaven på. Flere av dem er svært avanserte og vil kreve forholdsvis mye tid å utvikle og implementere. Jeg har landet på en løsning som etter min vurdering er enkel å implementere og samtidig tilfredsstiller de nødvendige krav til funksjonalitet. Det er også mulig at så lenge man bruker denne ruteren som plattform vil man ikke vil kunne implementere for avanserte løsninger, gitt de minne og CPU begrensninger som tross alt denne ruteren har. ***Valgt løsning er derfor Statisk ruting – nodeadressering.***

11 REFERANSER

[1] The Click Modular Router by Eddie Kohler
http://www.read.cs.ucla.edu/click/

[2] Ant-based monitoring on software IP routers
http://www.cs.unibo.it/bison/deliverables/D14.pdf

[3] IEEE 802.3 CSMA/CD (ETHERNET)
http://www.ieee802.org/3/

[4] RFC826 Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware
http://www.faqs.org/rfcs/rfc826.html

[5] RFC1027 Using ARP to implement transparent subnet gateways
http://www.faqs.org/rfcs/rfc1027.html

[6] IBM RedBooks TCP/IP Tutorial and
Technical Overview
http://www.redbooks.ibm.com/redpieces/pdfs/sg247287.pdf

[7] RFC 2131 Dynamic Host Configuration Protocol
http://www.faqs.org/rfcs/rfc2131.html

[8] RFC 3927 Dynamic Configuration of IPv4 Link-Local Addresses
http://www.faqs.org/rfcs/rfc3927.html

[9] MANETconf: Configuration of Hosts in a Mobile Ad Hoc Network
Sanket Nesargi, Ravi Prakash
http://www.utdallas.edu/~ravip/papers/infocom2002.pdf

[10] XORP the eXtensible Open Router Platform.
http://www.xorp.org/

[11] GNU Zebra
http://www.zebra.org/

[12] Linksys WRT54GL router
http://www.linksys.com/servlet/Satellite?c=L_Product_C2&childpagename=US%2FLayout&cid=1133202177241&pagename=Linksys%2FCommon%2FVisitorWrapper

[13] OpenWrt the Linux distribution for embedded devices
http://openwrt.org/

[14] http://wiki.openwrt.org/TableOfHardware

[15] Nam: Network Animator
http://www.isi.edu/nsnam/nam/

[16] The Network Simulator - ns-2
http://www.isi.edu/nsnam/ns/

[17] http://forum.openwrt.org

[18] http://www.scorpiontek.org/portal/content/view/27/36/

APPENDIX 1: Konfigurasjonsskript AntPing

```
// Auto-generated by make-ant-router.pl
// vlan3 10.1.5.1 00:0A:0A:01:05:01
// vlan4 10.1.2.1 00:0A:0A:01:02:01
// vlan5 10.1.3.1 00:0A:0A:01:03:01
// vlan6 10.1.9.1 00:0A:0A:01:09:01

AddressInfo (
       vlan3 10.1.5.1,
       vlan4 10.1.2.1,
       vlan5 10.1.3.1,
       vlan6 10.1.9.1);

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;
};

rt :: StaticIPLookup(
      10.1.5.1/32 0,
      10.1.5.255/32 0,
      10.1.5.0/32 0,
      10.1.2.1/32 0,
      10.1.2.255/32 0,
      10.1.2.0/32 0,
      10.1.3.1/32 0,
      10.1.3.255/32 0,
```

```
      10.1.3.0/32 0,
      10.1.9.1/32 0,
      10.1.9.255/32 0,
      10.1.9.0/32 0,
      10.1.5.0/255.255.255.0 1,
      10.1.2.0/255.255.255.0 2,
      10.1.3.0/255.255.255.0 3,
      10.1.9.0/255.255.255.0 4,
      255.255.255.255/32 0.0.0.0 0,
      0.0.0.0/32 0,
      0.0.0.0/0.0.0.0 10.1.9.2 4);

sr :: SourceRoute(0.0.0.0) -> rt;

ip :: CheckIPHeader(VERBOSE true, DETAILS true)
    -> ant :: IPClassifier(dst udp port 51234, src udp port 51234, -);

ant[0] -> fa :: ForwardAnt(at);
ant[1] -> BackwardAnt(at) -> sr;
ant[2] -> sr;


// 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_10.1.5.1") ->
c0;
//out0 :: Queue(200) -> ToDump("/bison/openwrt/logs/to_10.1.5.1") ->
todevice0 :: ToDevice(vlan3);
arpq0 :: ARPQuerier(10.1.5.1, 00:0A:0A:01:05:01) -> out0;
c0[0] -> ar0 :: ARPResponder(10.1.5.1 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_10.1.2.1") ->
c1;
//out1 :: Queue(200) -> ToDump("/bison/openwrt/logs/to_10.1.2.1") ->
todevice1 :: ToDevice(vlan4);
arpq1 :: ARPQuerier(10.1.2.1, 00:0A:0A:01:02:01) -> out1;
c1[0] -> ar1 :: ARPResponder(10.1.2.1 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_10.1.3.1") ->
c2;
//out2 :: Queue(200) -> ToDump("/bison/openwrt/logs/to_10.1.3.1") ->
todevice2 :: ToDevice(vlan5);
```

```
arpq2 :: ARPQuerier(10.1.3.1, 00:0A:0A:01:03:01) -> out2;
c2[0] -> ar2 :: ARPResponder(10.1.3.1 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_10.1.9.1") ->
c3;
//out3 :: Queue(200) -> ToDump("/bison/openwrt/logs/to_10.1.9.1") ->
todevice3 :: ToDevice(vlan6);
arpq3 :: ARPQuerier(10.1.9.1, 00:0A:0A:01:09:01) -> out3;
c3[0] -> ar3 :: ARPResponder(10.1.9.1 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[0] -> 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[1] -> pvlan3 :: prepareIP2(0, vlan3);
pvlan3[0] -> [0]arpq0;
pvlan3[1] -> rt;

// Forwarding path for vlan4
rt[2] -> pvlan4 :: prepareIP2(1, vlan4);
pvlan4[0] -> [0]arpq1;
pvlan4[1] -> rt;

// Forwarding path for vlan5
rt[3] -> pvlan5 :: prepareIP2(2, vlan5);
pvlan5[0] -> [0]arpq2;
pvlan5[1] -> rt;

// Forwarding path for vlan6
rt[4] -> pvlan6 :: prepareIP2(3, vlan6);
pvlan6[0] -> [0]arpq3;
pvlan6[1] -> rt;

// Routing from ForwardAnt
fa[0] -> SetIPAddress(10.1.5.2) -> pvlan3;
fa[1] -> SetIPAddress(10.1.2.2) -> pvlan4;
fa[2] -> SetIPAddress(10.1.3.2) -> pvlan5;
fa[3] -> SetIPAddress(10.1.9.2) -> pvlan6;
fa[4] -> local;  // Packet is for this host
fa[5] -> rt;
```