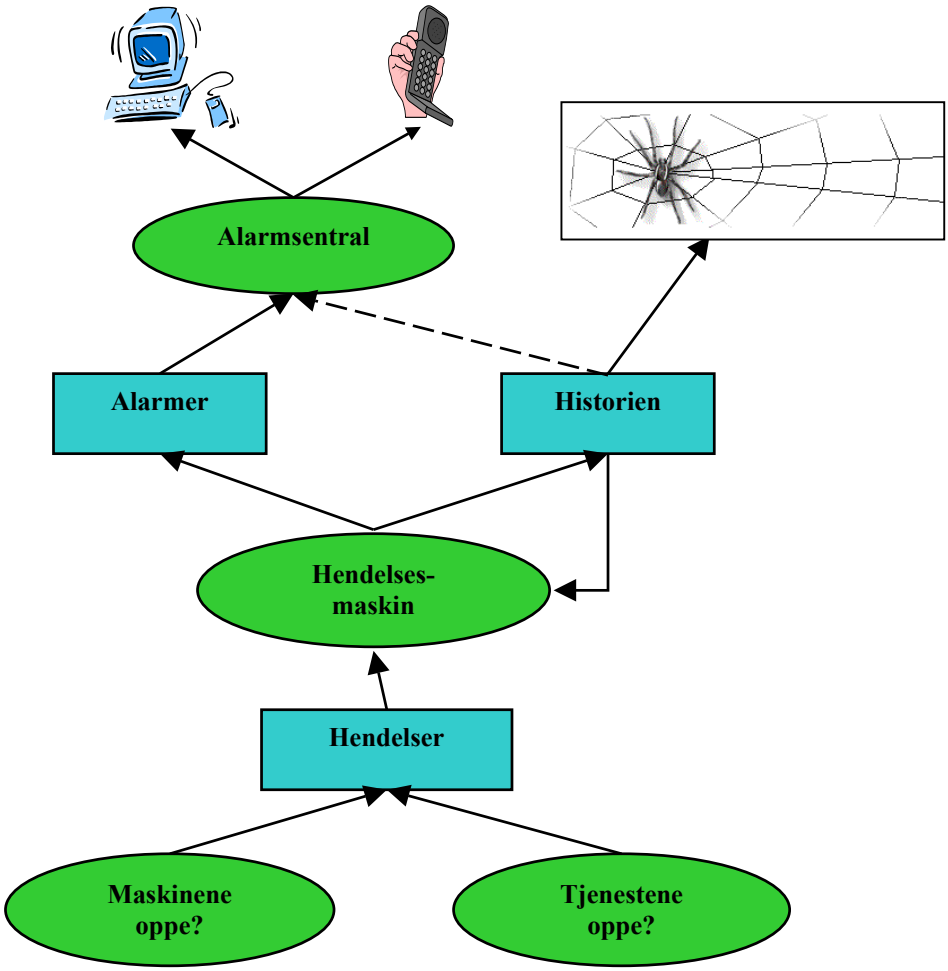
Sluttrapport

NAVMore



Videreutvikling av NAV

UNINETT Prosjekt 353101.12 kontrakt K-04-042

Prosjektleder
Vidar Faltinsen

Prosjektgruppe
John Magne Bredal – Kristian Eide
Sigurd Gartmann – Erik Gorset – Steinar Hamre
Magnus Thanem Nordseth – Daniel Sandvold – Stian Søiland –
Gro-Anita Vindheim – Knut-Helge Vindheim

ITEA, NTNU
12. desember 2002

Sammendrag

Prosjekt NAVMore har pågått fra 27. mai til 12. desember 2002 og har hatt fokus på videreutvikling av NAV[1]. Oppdragsgiver har vært UNINETT og NTNU. Prosjektet er utført som et samarbeid mellom ITEA[2]s nett- og systemdriftgruppe. Prosjektet er også tett koordinert opp i mot UNINETTs egen nettadministrasjonsaktivitet.

Prosjektgruppen har bestått av 11 personer, 6 fra ITEA nett, 5 fra ITEA systemdrift. Totalt timevolum har vært 1451 timer, hovedtyngden av aktiviteten er utført i løpet av sommeren 2002.

NAV er i utgangspunktet et nettadministrasjonssystem, men med NAVMore utvider vi horisonten i retning systemovervåkning. De største funksjonelle nyhetene er tjeneste-overvåker og cricket-statistikk for servere.

Prosjektet favner imidlertid et vidt spekter av deloppgaver, opprinnelig 17, der 5 er utsatt av tidsmessige årsaker. Vi kan dele resultatene i to kategorier; løsninger som inngår i NAV v2.1 (23% av arbeidet) og løsninger som tilhører NAV v3 (77% av arbeidet).

Prosjekt NAVRun har løpt parallelt med NAVMore og har bidratt til at 10 høgskoler/ universitet har installert NAV v2.1. Alle v2.1 resultatene av NAV er altså godt tatt i bruk. Resultater her omfatter:

- Maskinsporing på svitsjeportnivå (gir ”fullverdig” maskinsporing)
- Strukturert håndtering av Cisco syslogmeldinger
- Strukturert system for NAV feilmeldinger
- Styrket topologiavledning, lag2 og lag3
- Utbedringer rundt datainnsamling
- Grafisk visualisering av adresseromsforbruk

Den tyngste aktiviteten i NAVMore har hatt fokus på NAV v3. Målet har vært å få opp en testinstallasjon ved NTNU. Dette er gjennomført, alle v3 nyhetene kjører nå i pilot:

- Sorterte Cricket-data
- Cricket statistikk for servere
- Tjenesteovervåker
- Ny og parallell pinger (statusmonitor)
- Nytt hendelsessystem, event engine, med grensesnitt mot nytt varslingssystem, alert engine.
- UNINETT har selv implementert en alert engine pilot, samt et nytt system for NAV-brukeres varslingsprofiler.

Vi anser oss for å være i en tidlig testfase, med rom for forbedringer og ytterligere videreutvikling. Vi diskuterer i rapporten ideer til videre arbeid og vi foreslår et opp-følgende prosjekt i 2003. Her bør alle sentrale v3-brikker falle på plass. Endelig målsetning bør være å tilby NAV v3 til alle interesserte universiteter og høgskoler høsten 2003.

[1] NAV står for Network Administration Visualized og er NTNUs egenutviklede nettadministrasjonsløsning. Utviklingen startet i 1999 og har pågått i snart 4 år. UNINETT har fra 2001 støttet NAV gjennom tre prosjekter; NAVMe, NAVRun og NAVMore.

[2] ITEA er NTNUs sentrale IT-seksjon.

Innhold

Forord

Prosjekt NAVMore er utført av en prosjektgruppe ved ITEA/NTNU på oppdrag fra UNINETT ved Olav Kvittem og fra NTNU ved Roar Aspli. Prosjektet har et timeforbruk på 1451 timer og er utført i perioden mai-desember 2002. Prosjektdeltagere har vært:

- Vidar Faltinsen (prosjektleder)
- Knut-Helge Vindheim (nett)
- Steinar Hamre (systemdrift)
- Erik Gorset (systemdrift)
- Magnus T. Nordseth (systemdrift)
- Daniel Sandvold (systemdrift)
- Stian Søiland (systemdrift)
- John Magne Bredal (nett)
- Kristian Eide (nett)
- Sigurd Gartmann (nett)
- Gro-Anita Vindheim (nett)

I deler av prosjektet har det vært et nært samarbeid med UNINETT, da med:

- Arne Øslebø
- Andreas Åkre Solberg

Prosjektleder benytter her anledningen til å takke hele prosjektgruppen:

- Til nett – nok en gang – vel blåst!
- Til systemdrift – velkommen om bord – dere har tatt oss med storm :)
- Til UNINETT – takk for samarbeidet – det bør definitivt fortsette!

Vidar Faltinsen
Prosjektleder

Kapittel 1: Innledning

1.1 Hva initierte dette prosjektet?

NAVMore er et samarbeidsprosjekt mellom NTNU og UNINETT. Prosjektet har pågått fra mai til desember 2002 og løpt i parallell med prosjekt NAVRun. Begge prosjektene er oppfølgere etter NAV aktiviteten i 2001 (NAVMe), der vi løsrev NAV fra NTNU og installerte en pilot ved UiTø. NAVRun har fokusert på distribusjon og laget en installasjonspakke, mens dette prosjektet har utviklet ny funksjonalitet basert på ønsker og ideer fra NTNU selv, fra UiTø-piloten og fra UNINETT.

1.2 Hva er prosjektets hovedoppgave?

Målsetningen med NAVMore er å introdusere ny funksjonalitet på en rekke områder. Det har vært ønskelig å dele resultatene i to kategorier:

1. NAV v2.1 funksjonalitet
   Dette er påbygninger og supplement til NAV v2.0 som vi har funnet forsvarlig (og formålstjenlig) å introdusere raskt. Målsetningen har vært å tilby v2.1 utvidelsene med NAVRun sin installasjonspakke høsten 2002.

   Ønskede resultater: fullverdig maskinsporing, syslogdatabase, mer robust datainnsamling, mer intelligent topologiavledning, bedre visualisering av adresseromsforbruk.

2. NAV v3 funksjonalitet
   Dette er mer gjennomgripende forandringer som kan føre til ustabilitet i systemet for øvrig, og som i seg selv må testes og utbedres i en noe lengre betafase. Målsetning her har vært å få en kjørende pilot/prototyp ved NTNU. Langsiktig målsetning er å introdusere NAV v3 ovenfor andre universiteter og høgskoler høsten 2003, da gjennom et oppfølgende prosjekt.

   Ønskede resultater: sorterte cricket-data, cricket server-statistikk, tjenesteovervåker, parallell-pinger, responstids- og pakketapsdata, styrket hendelsessystem (som kan se hendelser i sammenheng og opp i mot nettets topologi), nytt og mer fleksibelt varslingssystem.

1.3 Hvilke effekter vil prosjektet ha for oppdragsgiver?

Nytteverdien av NAV øker ytterligere som en følge av NAVMore. Etterlengtet funksjonalitet er implementert. Horisonten er også utvidet i retning systemovervåkning. ITEAs systemdriftgruppe ser umiddelbar nytte av tjenesteovervåkeren, samt cricket-statistikk for servere.

1.4 Deloppgaver / tidsbruk

Prosjektplanen definerte 16 deloppgaver, med angitt tidsbruk og fremdrift. Vi har i tabellen under gjengitt alle deloppgavene, men sortert dem i fire kategorier; prosjektadministrasjon, NAV v2.1 arbeid, NAV v3 arbeid og ikke utført arbeid. Vi har også lagt til en oppgave (17) for hendelses-systemet, event engine (siden dette har vokst seg til et selvstendig område).

Nr	Beskrivelse	Budsjett	Regnskap	NAV ver
1	Prosjektadminstrasjon, sluttrapport	105	175	*adm*
2	Maskinsporing til svitsjeport	70	48	v2.1
3	Bedre topologiavleder lag2 og lag3	70	50	v2.1
5	Sysloganalysator	35	35	v2.1
6	Styrke robusthet,fleksibilitet,logging NAVdb	140	72	v2.1
8	Utvidelser rapportgenerator	70	35	v2.1
	Sum v2.1 (og prosentvis andel, adm innbakt)		*240*	*23 %*
10	Sorterte statistikker trafikk (alle cricket-data)	35	70	v3
11	Styrket statusside, styrket hendelsessystem	70	40	v3
17	Event engine		195	v3
14	Parallellping, pakketap, responstid	35	137	v3
15	Tjenesteovervåker	105	297	v3
16	Systemdrift-statistikk (Cricket)	70	277	v3
	Sum v3 (og prosentvis andel, adm innbakt)		*1016*	*77 %*
4	Nettkart: vlanutbredelse i et vindu	140	0	UT
7	Endringshåndtering, historie, offline-liv	140	20	UT
9	Utvidelser inventardata	35	0	UT
12	Parallellisert RRD collector	105	0	UT
13	Fleksibilitet grenseverdier for terskler	35	0	UT
	Sum NAVMore	**1015**	**1451**	*100 %*

Tabell 1: NAVMore delaktiviteter og tidsbruk (timer)

Som oversikten viser har vi latt 5 deloppgaver utgå fra prosjektet. Til tross for dette har vi overforbrukt i forhold til budsjett. Det er særlig arbeid rundt nytt hendelsessystem, tjenesteovervåker, ny pinger og systemdriftstatistikk som har vært tidkrevende (og som var feilbudsjettert i utgangspunktet).

23% av arbeidet har gitt v2.1 funksjonalitet. Dette er allerede tilgjengelig gjennom NAV sin installasjonspakke og kjører i skrivende stund på 10 installasjoner. v2.1 arbeidet kommer vi nærmere inn på i kapittel 2.

Hovedtyngden av NAVMore (77%) er v3 arbeid. Dette blir behandlet i kapittel 3. Arbeidet har gitt konkrete resultater ved at vi har en fungerende pilot kjørende ved NTNU, riktignok i en tidlig testfase, med rom for forbedringer og utvidelser.

Vi ønsker gjennom et nytt prosjekt å ta opp tråden med de utgåtte deloppgavene. I kapittel 4 drøfter vi videre arbeid.

Med NAVMore utvider NAV for første gang horisonten i retning systemovervåkning. Vi introduserer cricket-statistikk av servere, og vi har implementert en tjenesteovervåker. Arbeidet er utført av studenter ved ITEAs systemdriftgruppe. NAV sin utviklingsgruppe har således også vokst, noe som har vært en utfordring i seg selv.

Samarbeidet har vi også gjort i retning UNINETT, også dette for første gang på "utviklingsnivå". UNINETT selv ønsker et nytt hendelses- og varslingssystem (omtalt som alarmsentral). De har av den grunn fattet stor interesse for vårt event engine design, og de har sågar bidratt med konkrete resultater rundt alert engine og ny web frontend for NAV-brukernes varslingsprofiler. Også dette samarbeidet er svært gledelig.[3]

For å tydeliggjøre de ulike prosjektmedlemmenes bidrag, se tabell 2.

Navn	Opphav	Oppg	Hva
Vidar Faltinsen	ITEA nett		Prosjektleder
Knut-Helge Vindheim	ITEA nett		Vikarierende prosjektleder, faglig koord nett
Steinar Hamre	ITEA sdrift		Faglig koordinater systemdrift
Erik Gorset	ITEA sdrift	15	Tjenesteovervåker
Magnus T. Nordseth	ITEA sdrift	14,15	pping, tjenesteovervåker
Daniel Sandvold	ITEA sdrift	16	Systemdrift cricket
Stian Søiland	ITEA sdrift	14,16	pping, Systemdrift cricket
John Magne Bredal	ITEA nett	10,11	Sortert RRD, ny statusside
Kristian Eide	ITEA nett	17,2,3	Event engine, cam-db, lag2 topologi
Sigurd Gartmann	ITEA nett	3,5,6,8	Lag3 topologi, syslog, robust NAVdb, ragen
Gro-Anita Vindheim	ITEA nett	8	Visualisering av adresseromsforbruk
Andreas Åkre Solberg	UNINETT		Brukerprofiler varsling
Arne Øslebø	UNINETT		Alert engine

Tabell 2: Prosjektdeltagere og oppgaver

1.5 Dokumentasjon

Det er jobbet kontinuerlig med dokumentasjon i hele prosjektet. Alt er linket opp fra NAVMore sin prosjektside, se http://metanav.ntnu.no/NAVMore.

[3] UNINETT aktiviteten defineres formelt sett utenfor prosjektet, men som "samarbeidende resultater".

1.6 Relatert arbeid

Prosjekt NAVRun har løpt parallelt med NAVMore og har hatt fokus på installasjon og distribusjon. NAVRun har gjort NAV tilgjengelig for interesserte høgskoler og universiteter. Gjeldende versjon er 2.1.3, denne inneholder alle resultater nevnt i kapittel 2 (med dertil bugfikser). Pr dato har alle universiteter og 6 høgskoler installert NAV, flere vil komme til. NAVRun leverte sin sluttrapport 13/11-02, rapporten er tilgjengelig på http://metanav.ntnu.no/NAVRun/NAVRun.pdf.

Kapittel 2: NAV versjon 2.1 funksjonalitet

Som tabell 1 i kapittel 1.4 viser, så utgjør deloppgave 2, 3, 5, 6 og 8 NAV v2.1 funksjonalitet. Arbeidet er fullført og allerede innlemmet i kjørende versjon av NAV (pr dato v2.1.3).

Vi beskriver i dette kapitlet v2.1 arbeidet som er gjort.

2.1 Maskinsporing til svitsjeport (deloppgave 2)

I versjon 2.0 gjorde vi kun innsamling av ARP-data, ikke innsamling av brotabelldata. Med ARP-data får vi en oversikt over IP-adresseroms-forbruket, herunder hvilke IP-adresser som er aktive når[4]. ARP-dataene har også bindingen IP-MAC som er "limet" som skal til for å gjøre maskinsporing nyttig.

En svært sentral utvidelse i v2.1 er innsamling av brotabelldata fra alle svitsjer for alle vlan. Med disse dataene har vi sporbarhet av en gitt maskin helt ut til siste svitsjeport vi overvåker. Brotabellene har naturligvis MAC-adresser, men ved å koble dette mot ARP-data kan vi spore IP-adresser. Web frontend gjør også DNS-oppslag for å gjøre dette enda mer brukervennlig.

CAM-loggeren (brotabellinnsamleren) er implementert i Java og kjøres av cron hvert kvarter. Den kjører i parallell (default 24 tråder) og henter brotabelldata for alle vlan (dvs alle vlan som er definert i prefikstabellen, altså alle aktive vlan) for alle svitsjer. Dernest lagres dataene i en tabell der man for hver post har en fra-til levetid for en gitt macadresse på en gitt svitsjeport. Denne lagringen blir helt analog med ARP.

Merk at vi gjør innsamling for alle svitsjer (både kjerne og kantsvitsjer). Merk videre at vi har innebygget logikk for *ikke* å logge CAM-data på uplink-/downlinkporter (dvs porter som er direkte tilkoblet en annen svitsj NAV overvåker, topologiavlederen finner dette). Det er i de fleste tilfeller uinteressant å se alle brotabelldataene på uplink/downlinkporter, dette gir bare forvirrende bidrag til bildet av hvor en maskin faktisk er tilkoblet.

Arbeidet med CAM-loggeren ble påbegynt i NAVMe. CAM-loggeren gir også grunnlagsdata for å avlede topologien. For dette formålet samler CAM-loggeren også inn CDP data m.m. Mer detaljer rundt algoritmen og kjøretidsdata til CAM-loggeren er vist i tekstboksen under. For detaljer om hvordan topologiavledning og vlanavledning er implementert, se system-dokumentasjon på http://metanav.ntnu.no/v2/systemdok/cam.txt.

[4] Man må dog justere for ARP timeout i ruter som default er 4 timer.

Algoritme og kjøretidsdata CAM-logger

CAM-loggeren kjører hvert kvarter. Det som skjer da er at alle svitsjer (SW og KANT) blir hentet fra databasen og lagt i en kø. Så startes et antall tråder (24 er default) som alle jobber mot denne køen. Hver tråd sjekker om det ligger en boks i køen, og hvis så er tilfellet hentes denne ut og tråden foretar spørring med SNMP. Når dette er ferdig sjekker den køen igjen. Dersom køen er tom avslutter tråden. Alle trådene vil altså hente bokser fra køen helt til denne er tom, og da har alle resterende bokser en tråd som jobber mot dem. Dette sikrer at alle trådene hele tiden har arbeid å gjøre selv om noen bokser tar mye lenger tid å hente data fra enn andre.

Hvor fort går det? Siden SNMP spørring ikke er bundet av ressurser på maskinen så minker tidsforbruket lineært med antall tråder som brukes (nesten, skal man være helt nøyaktig forutsetter det at alle boksene tar like lang tid å spørre, noe som dessverre ikke er tilfellet). Hvis vi ser på bb (NTNUs NAV-maskin) så spør vi nå 661 svitsjer, mange av disse er modulære eller stacket (fysisk eller virtuelt). Med seriell camlogger så tok det ca. 2 timer å gå over alle boksene, mens siste kjøring nå tok 4 minutter 45 sekunder. Det er 24 tråder, og 120/24 = 5 minutter, så lineært minkende tidsforbruk stemmer bra (tiden blir litt lavere fordi vi har optimalisert en del siden vi tok tiden på seriell kjøring, spesielt dette med at vi prøver å spørre bare aktuelle vlan hjelper endel). Her er tidene for de 5 tregeste boksene på siste kjøring (de 25 tregeste står i loggfilen til CAM-loggeren):

```
01: 00:01:56.853, sb-354-sw (C4006) (129.241.161.46)
02: 00:01:41.416, sit-sby-ebs21-h (Off8) (129.241.138.13)
03: 00:01:38.398, vm-ans-107-h (PS40) (129.241.46.9)
04: 00:01:32.951, mask-ans-312-h (PS40) (129.241.62.10)
05: 00:01:23.464, mask-stud-310-h (2524) (129.241.65.7)
```

2.2 Forbedringer topologi- og vlanavledning (deloppgave 3)

Det er fikset på en del aspekter rundt topologi- og vlanavledning. Vi hadde blant annet problemer med at avledningen ble for statisk, slik at etter endringer i nettet, så hang gammel topologi-informasjon igjen. Dette er rettet, vi har nå en mekanisme for å slette gammel topologi-informasjon. For å gjøre dette robust sletter vi boksbak-innslag som ikke er sett ved siste x kjøringer av CAM-loggeren, der x kan konfigureres i konfigurasjonsfilen getBoksMacs.conf. Default verdi er 3 (man kan sette denne til 1, men erfaring tilsier at man tidvis mangler data, så 3 er anbefalt). Merk at vi ikke sletter topologi rundt enheter som er nede (erklært nede av statusmonitor).

En annen utfordring har vært å håndtere gjenbruk av vlan innenfor samme NAV-installasjon. Dette gjøres konsekvent på høgskoler som er spredd på ulike lokasjoner. UNINETT har en innført en standard som tilordner vlan 10 til administrasjon, vlan 20 til ansatte og vlan 30 til studenter. Vi kan også oppleve gjenbruk av vlan på universitetene på tvers av VTP-domener. Vi håndterer nå gjenbruk av vlan ved at vi generelt avleder topologi pr rootgwid pr IP adresserom (rootgwid er ”primær” ruterport for et gitt subnett, det er startstedet for vlanavlederen).

2.3 Topologiavledning lag3 (deloppgave 3)

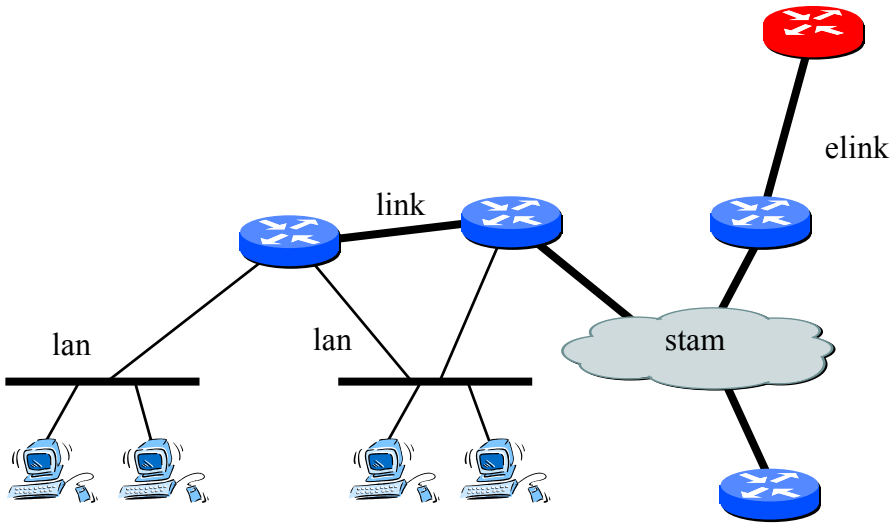
I NAVMe baserte vi oss på konvensjon for description-felt på ruterporter. Med NAVMore er vi nå mer løsrevet fra dette. Vi anbefaler fortsatt NAV-konvensjonen, men man *må* ikke følge den. For høgskoler er det urealistisk å følge NAV-konvensjonen, da UNINETT selv har en svært innarbeidet og divergerende konvensjon for ruterport description.

Se http://metanav.ntnu.no/v2/admindok/navkonv.html for en beskrivelse av konvensjonen. Noen eksempler:

```
lan,idi,ans,glos,111          link,dragv-gw
lan,idi,stud,glos,103         elink,trd-gw,uninett
stam,itea,trlos
```

NAV-konvensjonen har flere formål:

1. Bestemme nettypen: Denne sier noe om lag 3 topologi. Ulike nettyper er illustert på figur 1.
2. Bestemme organisatorisk (org) tilhørighet og anvendelse (anv): Dette sier noe om hva subnettet brukes til.
3. Bestemme koblingen IP adresserom ↔ vlan: Dette kan gi informasjon om lag2 utbredelsen til et subnett.



Figur 1: Nettyper

Et alternativ til å bruke description-konvensjon er å bruke kildefilen vlan.txt. Den vil også kunne gi informasjon på alle 3 punkter.

I NAVMore har vi vært opptatt av å bestemme mest mulig automatisk. Det vi aldri kan avlede er punkt 2 over (org og anv), men vi har gjort tiltak på punkt 1 og 3.

2.3.1 Autoavledning av nettype

Algoritmen vi har innført er ikke vanntett, og den trer først i kraft dersom nettype *ikke* er angitt statisk enten i description-felt eller vlan.txt (så man kan hente seg inn om autoavledning feiler). Avledningen virker slik:

```
1)  Hvis et prefiks har en tilknyttet gwport
    a) Hvis interf = 'loopback%'
             => nettype=loopback
    b) ellers
             => nettype=lan
2)  Hvis et prefiks har mer enn en gwport
    OG
    en av disse har gwport.hsrp=true
    => nettype=lan
3)  Hvis et prefiks har 2 gwportposter fra 2 ulike rutere (altså
    forskjellige gwport.boksid), der begge  disse har hsrp=false
    => nettype=link
    => nettident = $gw1,$gw2
4)  Hvis et prefiks har 3 eller fler gwporter alle med hsrp=false
    => nettype=stam
```

Feilkilder:

- Et `lan` kan ha to eller flere ruterporter uten bruk av `hsrp`. Vi tenker her på tilfeller der subnettet *ikke* er beregnet på transitttrafikk, da blir nettypen `link` eller `stam` feil.
- Man kan kjøre `hsrp` på et `stam` (fordi man har en blanding mellom servere og transittrafikk, litt uryddig men fullt mulig).
- Vi klarer ikke å detektere `elink`. Det er jo en link mot en ekstern ruter/brannmur som vi ikke overvåker. Det vil falle under 1) over og bli kategorisert som `lan`.

2.3.2 Autoavledning av koblingen IP-adresserom ↔ vlan

I NAVMe baserte vi oss ene og alene på kildefilen vlan.txt. Vi liker ikke helt denne tilnærmingen og har sett på forbedringer. Samtidig registrerer vi at flere universiteter synes vlan.txt er en grei løsning, man må uansett ha en sentral fil (eller lignende) for uttak av vlan.

I NAVMore tilbyr vi imidlertid alternativer til vlan.txt som *kan* brukes om man vil. Først; vi har utvidet description-konvensjonen for ruterporter slik at vlanverdien kan angis her. Et eksempel er: `lan,idi,ans,glos,111`. Siste felt er vlanverdien. Se http://metanav.ntnu.no/v2/admindok/navkonv.html for detaljer.

Videre så autoavleder vi vlan-verdien for virtuelle interface, altså når ruterporten heter ”interface VlanXXX”. Den autoavledede verdien overstyrer evt manuelt satte verdier (fra description eller vlan.txt).

Vi ønsker også å autoavlede i tilfeller der man har logiske subinterfacer på en fysisk fastethernet eller gigaethernet ruterport. Da angis jo vlanverdien i

ruter-konfigurasjonen (med kommandoen encapsulation). Pr dato har vi ikke funnet noen måte å hente denne informasjonen med SNMP.

Et tredje tilfelle vi kunne autoavledet er tilfeller der en tradisjonell ruterport er koblet mot en svitsjeport og vi i topologiavlederen (basert på CDP eller MAC-adresser) vet hvilken port. Vi kan da sette vlanverdien til prefikset basert på satt vlanverdi på aktuelle svitsjeport. Dette bør være mulig å implementere.

2.4 Sysloganalysator (deloppgave 5)

Denne deloppgaven er fullført og inkludert i NAV v2.1. En forutsetning er at en syslogdemon kjører på NAV-boksen og logger alle syslogmeldinger fra Cisco-utstyr til en loggfil (NAVRun sin installasjonspakke tar seg av dette). Cisco syslogmeldinger har en iboende struktur. Kort fortalt består en Cisco meldingstype av tre ledd, adskilt med bindestrek (-). De tre leddene er:

1. Tema/område/delsystem som meldingen kommer fra.
2. Prioritet (tall fra 0-7): 0=emergency,1=alert … 6=info,7=debugging.
3. Beskrivelse innen område(1) som gjør meldingstypen unik og beskrivende.

Noen eksempler:

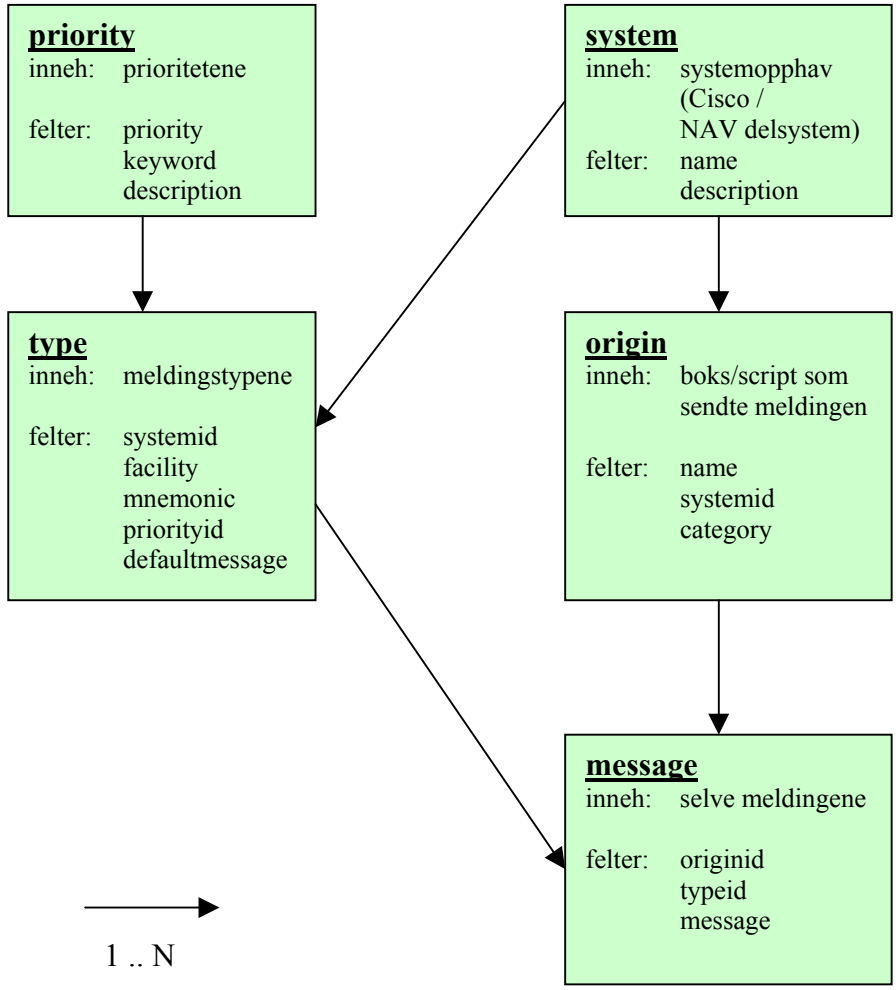
```
IP-3-TCP_BADCKSUM
IP-4-DUPADDR
ISDN-6-CONNECT
LINEPROTO-5-UPDOWN
LINK-4-ERROR
```

I tillegg til meldingstypen er det med en beskrivelse av den aktuelle saken, samt info om hvilken Ciscoenhet (ruter/svitsj) dette inntraff på og når. Et eksempel er:

```
May 27 08:32:58 mtfs-sw.ntnu.no 2002 May 27 08:32:53 MET +02:00
%CDP-4-NVLANMISMATCH:Native vlan mismatch detected on port 4/2
```

NAV sysloganalysator utnytter strukturen og splitter opp meldingene og legger dem inn i en database (navlog). Vi har også adoptert samme feil-meldingssystem for NAV sine egne feilmeldinger[5]. Databasen inneholder derfor både cisco-meldinger og nav-meldinger (tabellen system indikerer opphav). Det er totalt fem tabeller i navlog, de er vist med felter og relasjoner på figur 2.

[5] Dette er ikke konsekvent gjennomført for alle NAV delsystemer. NAVlog er mer omtalt i kap 2.5.

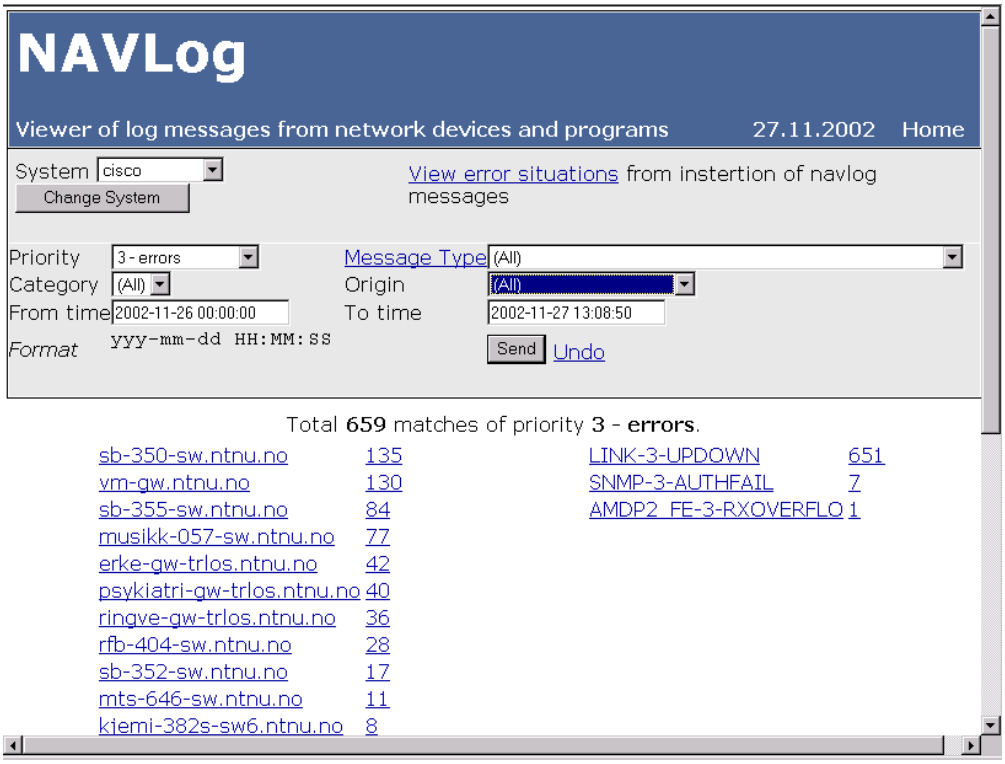


Figur 2: navlog databasen

NAVlog sin frontend er laget i php og har rullegardinger og knapper for enkelt å gjøre søk og uthenting av data i NAVlog. Et eksempel er vist på figur 3, der vi viser alle Cisco syslogmeldinger av prioritet=3 (error) for siste to døgn. Som vi ser er oversikten strukturert i alle involverte meldingstyper og alle involverte bokser. Man kan klikke seg nedover og se de underliggende detaljene for en boks, en meldingstype osv.

Vi har også laget to konfigurasjonsfiler til navlog, der den ene (delete.conf) spesifiserer hvor lenge vi skal ta vare på navlogmeldinger (oppløselig på prioritetsnivå), mens den andre (priority_exceptions.conf) gir mulighet til å omprioritere Cisco sine prioriteter (dersom man er uenig / har andre krav).

Videre er det en cronjobb som regelmessig (hvert 20 minutt) sender alarmer som traps til trapdetect (varslingssystemet). Det er meldinger av prioritet 0, 1 og 2 som sendes som trap. En NAV-bruker kan i neste instans abonnere på disse meldingene.



Figur 3: NAVLog web frontend

2.5 Datainnsamling (deloppgave 6)

Det er gjort mye bugfiksing og utbedring av innsamlingsscriptene. En del er nevnt i kapittel 2.3. Ellers er det særlig håndtering av utfasing/endring som er styrket. I NAV v2.0 hadde vi en tendens til å samle inn riktig data initielt, men vi var ikke *alltid* like god på å håndtere endringer. Dette har naturligvis vært essentielt å utbedre.

Ved svitsjeportinnsamling (som er den mest tidkrevende bortsett fra cam), så har vi bygd inn støtte for SNMP v2 sin getbulk[6]. Dette har redusert kjøretiden betydelig (faktor 1:6).

Vi har også jobbet med nye utstyrstyper og lagt inn støtte for dette. Her nevnes:

- HP kantsvitsjer (HP Procurve 2524 med virtuell stacking).
- Catalyst 6500 native mode (GSW). Medførte mye arbeid, da dette introduserte en ny kategori: GSW, som er en hybrid mellom en ruter (GW) og en svitsj (SW). På en Cat6500 kan man vekselvis sette porter til å være svitsjeporter eller ruterporter.
- Catalyst 4000 sup II, Catalyst 3550, Catalyst 2950. Problemene her var nye OIDer for en del variable vi samler inn.

[6] getbulk mottar en bolk med svar i stedet for ett svar pr SNMP-spørring.

Innsamlingscriptene er videre bygd om til å støtte et pluginsystem for håndtering av typegrupper. Det er nå lettere å utvide støtte for nye utstyrstyper, man trenger "bare" å lage en ny plugin (bestående av en enkelt perlfil) for den nye typegruppen.

Vi har sett på en web-frontend for innlegging av data i kildefiler. Det er laget en pilot, men mer robusthet må inn her før vi kan "slippe" dette. Aktiviteten er nedprioritert, men bør (og vil) bli gjenopptatt til neste år (da også med relativt lav prioritet).

Sist men ikke minst er alle feilmeldingene fra innlesningsscriptene strukturert i navlog (se kap 2.4). Det er således mulig å strukturert søke i feilmeldingene, og det er implisitt gjennomført en kategorisering av feil i prioriteter/delystem/program analogt med Cisco sine 8 nivåer (fra 0 til 7).

Vi ønsker å benytte samme mekanisme for feilrapportering i andre delsystemer, men dette er hittil ikke gjennomført. Det er imidlertid tilrettelagt for det fra navlog sin side. Et annet delsystem trenger bare å logge til en fil i navlog sin loggkatalog (og selvfølgelig følge den veldefinerte syntaksen), så vil meldingene komme inn i navlogbasen!

2.6 Utvidelser i rapportgeneratoren (deloppgave 8)

Rapportgeneratoren er utbedret på en del punkter:

- ragen.conf finnes nå i to varianter: en navme-del (som kommer med NAV) og en local-del for lokale utvidelser. I den lokale ragen.conf kan man supplere med nye rapporter eller man kan om ønskelig overstyre navme-rapporter.
- Det er mulig å lage rapporter som bare vises for brukere med intern tilgang (typisk nettgruppa). NTNU anvender dette for å skjule tilgangen til sensitive romdata.
- Det er nå mer fleksibelt hvordan man kan lage lenker fra et felt i en rapport.
- Vi støtter summering av verdiene i en kolonne.
- Vi har lagt inn mulighet for forklarende tekster, slik at kolonneoverskrifter kan være korte (av hensyn til rapportbredden).

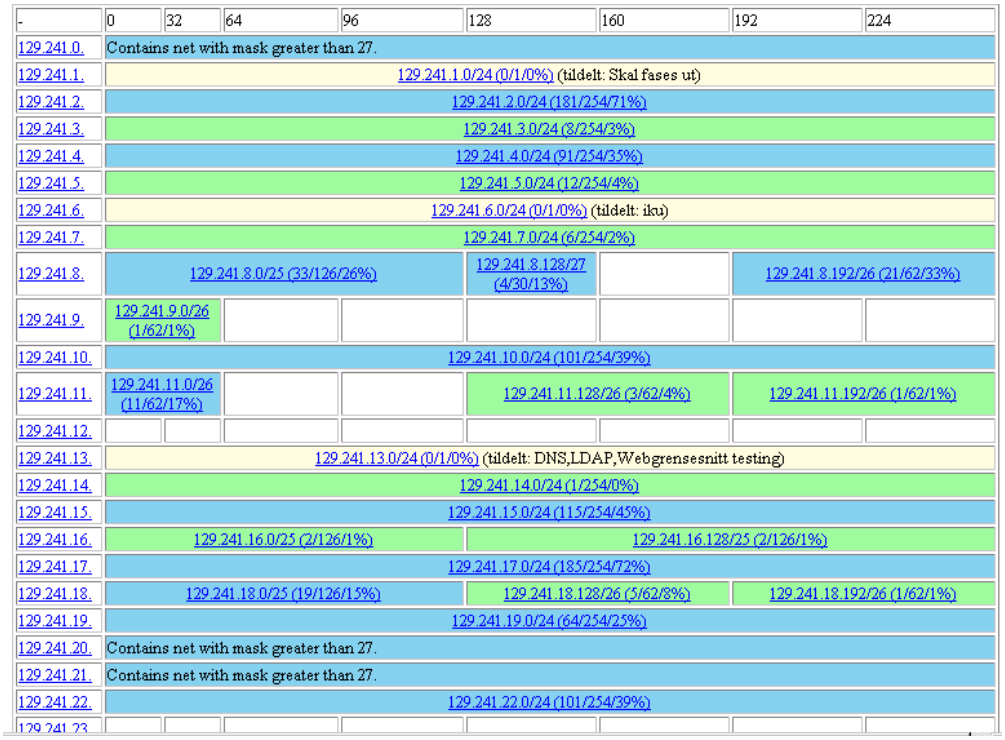
Det er flere ønsker rundt rapportgeneratoren, de nevnes under videre arbeid.

2.7 Grafisk visualisering av adresseromsforbruk

En forbedring i databasedesignet som kom i NAVMe var at vi skilte ruterporter fra IP adresserom. Det er nå en tabell for ruterporter (gwport) med pekere inn mot IP adresseromstabellen (prefiks). Dette har mange fordeler, en er at det er "enkelt" å gi et bilde av adresseromsforbruket. Hele

adresserommet angis i kildefilen prefiks.txt (NAV håndterer flere parallelle adresserom). For NTNU er dette 129.241.0.0/16.

Vi har laget en tabulær fremstilling av adresseromsforbruket, et ekempel er vist på figur 4. Alle subnett som er i bruk blir vist, ledige blokker kommer oversiktlig til syne. Fargekoder indikerer utnyttelsesgraden av subnettene som er i bruk (basert på ARP-data). Videre har vi lenker fra visningen til rapportgeneratoren for ytterligere informasjon.



-	0	32	64	96	128	160	192	224
129.241.0.	Contains net with mask greater than 27.							
129.241.1.	129.241.1.0/24 (0/1/0%) (tildelt: Skal fases ut)							
129.241.2.	129.241.2.0/24 (181/254/71%)							
129.241.3.	129.241.3.0/24 (8/254/3%)							
129.241.4.	129.241.4.0/24 (91/254/35%)							
129.241.5.	129.241.5.0/24 (12/254/4%)							
129.241.6.	129.241.6.0/24 (0/1/0%) (tildelt: iku)							
129.241.7.	129.241.7.0/24 (6/254/2%)							
129.241.8.	129.241.8.0/25 (33/126/26%)				129.241.8.128/27 (4/30/13%)		129.241.8.192/26 (21/62/33%)	
129.241.9.	129.241.9.0/26 (1/62/1%)							
129.241.10.	129.241.10.0/24 (101/254/39%)							
129.241.11.	129.241.11.0/26 (11/62/17%)				129.241.11.128/26 (3/62/4%)		129.241.11.192/26 (1/62/1%)	
129.241.12.								
129.241.13.	129.241.13.0/24 (0/1/0%) (tildelt: DNS,LDAP,Webgrensesnitt testing)							
129.241.14.	129.241.14.0/24 (1/254/0%)							
129.241.15.	129.241.15.0/24 (115/254/45%)							
129.241.16.	129.241.16.0/25 (2/126/1%)				129.241.16.128/25 (2/126/1%)			
129.241.17.	129.241.17.0/24 (185/254/72%)							
129.241.18.	129.241.18.0/25 (19/126/15%)				129.241.18.128/26 (5/62/8%)		129.241.18.192/26 (1/62/1%)	
129.241.19.	129.241.19.0/24 (64/254/25%)							
129.241.20.	Contains net with mask greater than 27.							
129.241.21.	Contains net with mask greater than 27.							
129.241.22.	129.241.22.0/24 (101/254/39%)							
129.241.23.								

Figur 4: Grafisk visualisering av adresserom

Oversikten er særlig nyttig for de som forvalter store adresseblokker og som til stadighet setter i drift og faser ut subnett. Vi tenker her spesielt på universitetene og på UNINETT selv.

Kapittel 3: NAV v3 arbeid

Mesteparten av arbeidet i NAVMore har hatt fokus på NAV v3. Tabell 1 i kap 1.4 gir en oversikt over deloppgavene som inngår, tabell 2 viser hvem som har bidratt. I dette kapitlet legger vi frem resultatene. Resultatene er oppe og kjører på NTNU sin NAV v3 pilot, følg lenker fra NAVMore-prosjektsiden (http://metanav.ntnu.no/NAVMore).

3.1 Databasen og datainnsamling

Først; vi har besluttet å konvertere hele NAV v2 databasen (manage) til engelsk, en operasjon som naturligvis har gitt bivirkninger for en rekke script. Dette omskrivningsarbeidet er gjennomført. Samtidig er databasen utvidet til å ta hånd om v3 utvidelser. En detaljert oversikt over endringer fra versjon 2 til versjon 3 kan sees under prosjektsiden til NAVMore, nærmere bestemt http://metanav.ntnu.no/NAVMore/v3manage.txt.

En overordnet oversikt over nye tabeller i NAVdatabasen (manage) gis her. Følg kapittelreferansen for detaljer:

Cricket innsamling for servere (se kap 3.3)	
netboxdisk	Diskpartisjoner
netboxinterface	Serverinterface
netboxcategory	Kategoritilhørighet for servere (1-N)
Tjenesteovervåker (se kap 3.5)	
service	Tjenestene som skal overvåkes
serviceproperty	Ekstra parametre / testdata knyttet til en tjenestetest
Utstyrshistorie (se kap 4.1.2)	
device	En fysisk enhet, kan være en modul eller en IP-enhet, eller noe annet. Device er mer generelt enn v2 sin boks (v3 sin netbox)
product	Informasjon om produkter (produktnr, beskrivelse, fabrikat)
order	Bestillinger
Nytt hendelses- og varslingssystem (se kap 3.6 – 3.11)	
eventq	Hendelseskøen (tømmes FIFO av event engine)
eventqvar	Ekstra variable som skal sendes med en hendelse (0-N)
eventprocess	Delsystemene som er leverandør til hendelseskøen
eventtype	Definerte hendelser (boxState, serviceState…)
alertq	Alarmkøen (fylles av event engine, tømmes av alert engine)
alertqvar	Ferdigformatterte alarmtekster klare for utsending. n*m poster tilknyttet hver alarm, gitt n alarmtyper (sms, epost…) og m språk (no, en…)
alerthist	Historisk tabell for alle hendelser. Ivaretar tilstandsinformasjon med start og sluttid. Kan evt utvides med enkel trouble ticket håndtering (alá Zino).
alerthistvar	Analogt med alertqvar. Har også formatering for web.

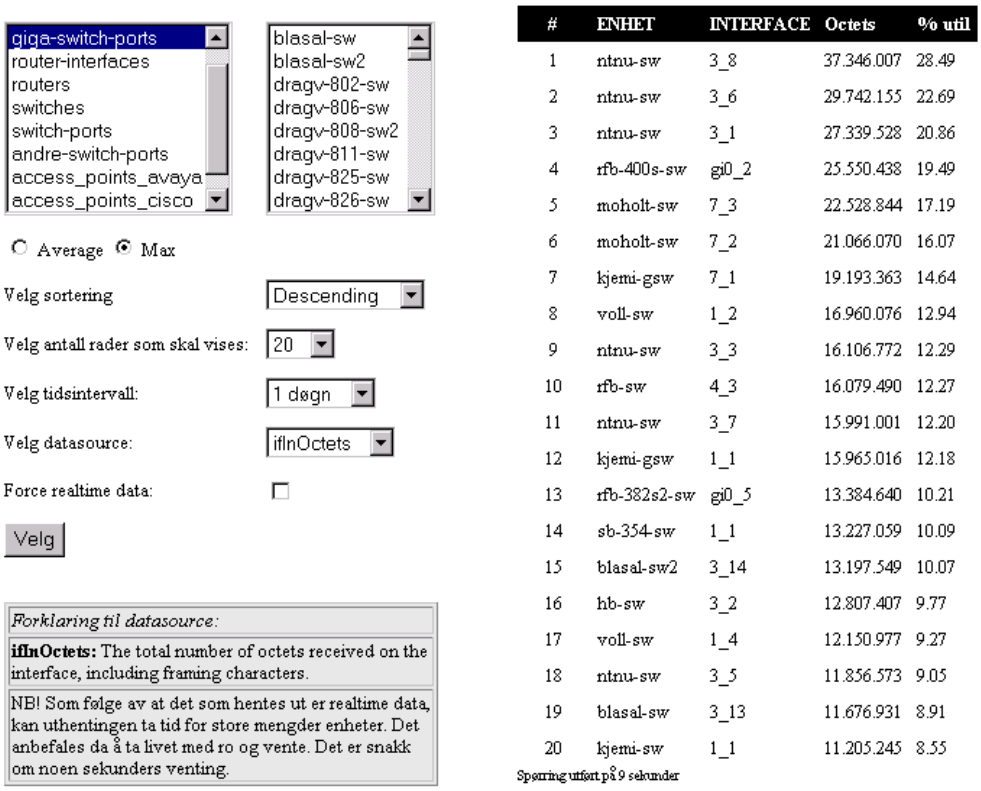
Tabell 3: Nye tabeller i NAV v3

UNINETT har i tillegg definert en helt ny database (navuser) med dertil tabeller for brukerprofiler (arvtager etter v2 sin trapdetectdatabase). Se kap 3.9 for mer.

Innsamlingsscriptene som fyller databasen er modifisert til å jobbe mot de nye tabellene. getPortData som i v2 tok seg av innsamling av svitsjeporter for kantutstyr er erstattet av et mer generelt script getDeviceData (implementert i java). For øyeblikket støtter API'et plugins som henter portdata (device, module og swport tabellene) samt boksdisk og boksinterface (se kap 3.3). getDeviceData kjører ikke fra cron, men gjør sin egen scheduling internt. Når det er tid for å behandle en boks så kalles riktig plugin-modul som gjør selve spørringen, mens koden for å oppdatere databasen er felles. Å lage nye plugin-moduler fungerer på samme måte som i event engine (se kap 3.6).

3.2 Sorterte statistikker (deloppgave 10)

Hensikten er å i tabulær form vise sorterte lister over ulik trafikkstatistikk. Vi har laget et generelt system der brukeren angir datakilde og evt et utvalg av utstyr, samt interessant tidsrom. Basert på dette vises en sortert liste. Et eksempel kan være maks last på alle gigasvitsjeporter siste døgn, se figur 5 under.



Figur 5: Sortert RRD statistikk

For å finne relevante RRD-filer i et gitt søk gjennomgås konfigurasjonstreet til Cricket (med kataloger og targets-filer). Dette gir grunnlaget for hvilke RRD-filer vi skal hente data fra.

Dernest brukes RRDTool for å hente ut tall fra alle RRD-filene. Disse dataene sorteres og vises på skjerm. Vi viser også prosentvis utnyttelse (utilization). For å få til dette henter vi båndbredde-verdier fra swport og gwport-tabellene i NAV sin database.

Et problem for NTNU er at det er så enormt mange datafiler, særlig for svitsjeporter. Dette gjør statistikkvisningen treg. For å bote på problemet har vi implementert en cronjobb som hvert 15. minutt henter data fra alle RRD-filene og lagrer disse til en fil. Frontenden kan da benytte disse dataene og gi et mye raskere resultat. En ulempe er at dataene blir noe foreldet (maksimalt 15 minutt). Ønsker man de helt siste dataene kan man tvinge igjennom et søk i frontenden.

Vi har et teknisk problem med løsningen ved at RRD fra tid til annen returnerer nullverdier. Vi undersøker årsaken til dette problemet.

Vi vurderer også en alternativ overbygning over RRD-filene, ved å ha en metadatabase som et supplement til Cricket sitt konfigurasjonstre. Det vil da bli lettere å utvide sorterte statistikker til å omfatte *alle* RRD-data man har. Se kapittel 4.1.4 for mer.

3.3 Systemdrift-statistikk (deloppgave 16)

Som kjent har vi i NAV v2 laget et script (makecricketconfig) som hver natt bygger Cricket sitt konfigurasjonstre. Basert på dette treet blir det samlet inn statistikk fra alle rutere og svitsjer som overvåkes. Vi ønsket i NAVMore å utvide dette konseptet til også å omfatte servere, noe som er gjort i denne deloppgaven. Arbeidet er oppsummert her, ytterligere detaljer er å lese på http://metanav.ntnu.no/NAVMore/cricket-doc.txt.

Innsamling fra servere er mer komplisert enn innsamling av last og feiltellere fra svitsjer og rutere. Det er langt større proprietære forskjeller, og det er et mer komplisert sett av variable det er interessant å samle inn. En god del arbeid er gjort for å se på klientsiden og ulike snmp demoner. For *nix falt valget på net-snmp[7]. Vi ønsket også å bruke denne for Windows, men fant at den ikke ga fra seg nok informasjon. Valget falt derfor på Windows sin egen snmpd.

For net-snmp har vi sett på muligheten for å lage egne MIBer. Dette viser seg å være en svært nyttig. Et eksempel som henter ut mailkøen fra en mailserver er vist her:

[7] http://net-snmp.sourceforge.net/

Tillegg i `snmpd.conf`:

```
# legger inn en egendef MIB - dog definert i en RFC
pass .1.3.6.1.2.1.28.1.1.1 /usr/local/sbin/mailq.sh
```

Scriptet `mailq.sh` ser slik ut:

```
if  [ "$1" = "-g" ] || [ "$2" = ".1.3.6.1.2.1.28.1.1.1" ] ; then
    echo .1.3.6.1.2.1.28.1.1.1
    echo gauge
    mailq | tail -1 | awk '{print $5;}'
fi
```

For at (det utvidede) makecricketconfig skal vite hvilke serverdata det skal samles data for, så trengte vi tre nye tabeller:

- netboxdisk: Diskpartisjoner
- netboxinterface: Alle interfacene til serverne
- netboxcategory: En server kan være med i flere kategorier

Det er videre gjort utvidelser i kildefilen server.txt for å angi kategoriene. Tabellene netboxdisk og netboxinterface fylles av en Jython-plugin[8] til getDeviceData. GetDeviceData spør altså serveren via snmp hvilke interface og hvilke partisjoner den har. Disse blir grunnlaget for datainnsamling. Det er ønskelig å unnta enkelte mountpoints (for eksempel automounts), disse angis i diskException.conf. Hele modellen er illustrert på figur 6.

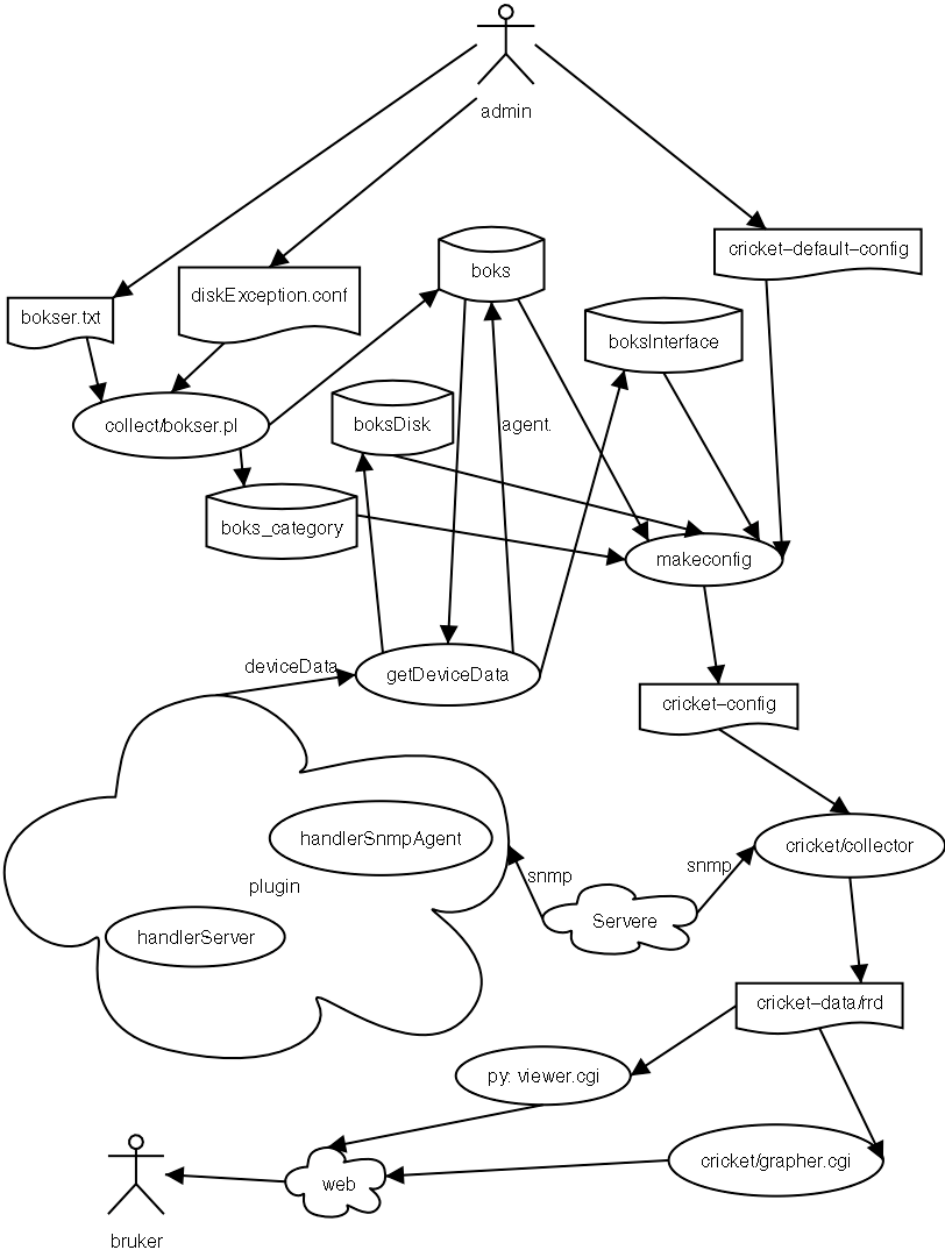
3.4 Parallellpinger (deloppgave 14)

En svakhet med Live (NAV v2 pingeren), er at den er sekvensiell. Den håndterer ikke større utfall på en helt optimal måte. Vi har i NAVMore laget en ny og parallell pinger. Den er implementert i python med relativt lavnivå sockethåndtering (vi bygger bl.a. icmp-pakkene selv).

I motsetning til Live som startes av cron, er den nye ppingeren en demon som går kontinuerlig med sin egen scheduling. Kort fortalt, henter den ved konfigurerbare intervaller en liste over IP-adresser den skal pinge fra NAV-databasen. Den returnerer en liste med pingtider, der man får verdien None for de som ikke svarer innen en gitt timeout (default 5 sekund). Det er laget en konfigurasjonsfil (pinger.conf) der pingeintervall, pakkestørrelse og timeout (m.m.) kan justeres. Algoritmen er forklart i mer detalj i tekstboksen under.

For å ta høyde for at man kan få svar fra et annet interface enn det man sender pakken til, er vi nødt til å lage en skreddersydd icmp-pakke til hver

[8] Jython er Python for Java. Pluginen vi her snakker om heter handlerServer.



Figur 6: Cricket innsamling av servere[9]

[9] Figuren er dessverre ikke helt oppdatert, diskException.conf skal peke på makeconfig, som egentlig skal hete makecricketconfig. Det er ellers brukt v2 navn på tabeller.

host. Dette gjør at utsending av pakker tar noe lenger tid enn ønskelig, men vi er likevel oppe i en rate på mellom 90-100 hoster/sek.

Algoritme pping – *en* pingrunde

Scriptet går i tre tråder:
1. Tråd 1 genererer og sender ut pakker
2. Tråd 2 tar imot, sjekker og lagrer pakkene
3. Hovedtråden står bare og venter på de andre og snurrer på en skråstrek.

Tråd 1 virker slik:
For hver host:
1. Lag en pakke med ping-streng som inneholder: (destinationIP, klokkeslett, programidentifikator)
2. Send ut pakken.
3. Legg til i ”Venter på svar”
4. Sov i 10 millisekunder (hvilket gjør at tråd 1 minimalt kjører i 7.37 sekunder med 737 hoster). Sovingen er viktig for å unngå at alle svar kommer samtidig (hvilket vil gi køing på mottakstråden (tråd 2) og dertil unøyaktige responstidstall).

Tråd 2 virker slik:
Så lenge tråd1 fortsatt går eller så lenge vi har noen i svarkøen:
1. Se om det er kommet data til socketen. (timeout: 5 sek)
2. Hent data
3. Pakk ut icmp-pakken
4. Sjekk at pakken er til vår pid (trådsikkert, "pid" er lagret av hovedtråden ved oppstart)
5. Split icmp-pakken i (destinationIP, klokkeslett, programidentikator)
   Om dette ikke går, dump pakken (ikke vår)
6. Sjekk at identifikatoren er lik vår
7. Dersom IP-adressen er en vi har i svarkø, oppdater 'pingtid' for hosten
   med time.time() – senttime
8. Fjern ifra 'venter på svar'-listen.

Hvis vi kommer hit har vi timet ut eller så er vi ferdige, så hostene vi har igjen har ikke svart i tide. Vi setter pingtid None på disse og rapporter til ringbuffer (for grovfiltrering før eventkøen).

Pingeren rapporter nede og oppemeldinger til event engine (via eventkøen). Event engine vil se nedemeldinger i sammenheng med andre nedemeldinger og bruke topologiinformasjon til å avgjøre skyggeforhold. Event engine vil også ha robusthetskriterier for å avgjøre om nedetilstanden er en transient. Se kap 3.6 for mer.

For å øke fleksibeliteten ønsker vi også muligheten for å grovfiltrere fra pingeren sin side. I pinger.conf kan man angi hvor mange etterfølgende pingmålinger som skal gjennomføres før nedemelding sendes til eventkøen. Dersom denne verdien for eksempel settes til 4, så vil først 4 *etterfølgende* ”None”-svar kvalifisere til nedemelding. Oppemelding vil fortsatt kun kreve et ikke-None svar.

Pingeren får naturlig nok masse responstid- og pakketapsdata. Vi har i prosjektet sett på en måte å ta vare på disse dataene. Det er gjort ved at vi leverer dataene til RRD. Vi har to målesett, ett for responstid, og ett for

pakketap. Nøyaktigheten av målingene blir en funksjon av pingfrekvensen. Vi ser at det er noe divergerende behov ved å pinge for å stadfeste om en enhet er nede versus å pinge for å få gode responstids- og pakketapsdata. Vi tror UNINETT sin MPING-løsning er mer optimal i forhold til gode statistiske verdier (med poisson fordelt pinging m.m.). Like fullt tror vi at pping RRD-verdiene kan gi gode nok data til at driftspersonell får nyttige indisier på trege enheter, store pakketap og andre fenomener som registreres over tid.

Vi mangler fortsatt å strukturere alle RRD-målingene inn mot Cricket. Vi ser dette i sammenheng med en ny mulig overbygning over RRD som potensielt kan supplere Cricket. Dette blir en mulig aktivitet i 2003 (se kapittel 4.1.4).

3.5 Tjenesteovervåker (deloppgave 15)

Tjenesteovervåkeren er implementert i pyton og i likhet med pingeren er den en demon med sin egen scheduling. Den består av en generell logikk-modul som håndterer algoritmen ved å i flere parallelle tråder løpe rundt og sjekke om alle tjenestene er oppe (nærmere omtalt under). Tjeneste-overvåkeren leverer tjeneste oppe og nede meldinger til eventkøen. Disse blir behandlet videre av event engine (som vil korrellere dette mot andre hendelser, så som boks nede/skygge). Mer om event engine i kap 3.6.

Det er skrevet plugin-moduler eller handlere for hver enkelt tjenestetype. Designet gjør det relativt enkelt å utvide løsningen til stadig nye tjenester. De tjenestene vi har implementert støtte for i NAVMore er:

Tjeneste	Testen som gjøres
ssh	Kobler til porten og leser første linje, dvs versjonsnummeret
http	Henter websiden som er spesifisert i serviceproperty, typisk path=~magnun/ vhost=www.ntnu.no
imap	Kobler til med spesifisert bruker og spesifisert passord og lister ut alle mailfoldere
pop3	Samme som imap
smtp	Kobler til angitt port og leser ut versjonsnummer
samba	Lister ut alle share
rpc	Sjekker hvilke tjenester portmapperen sier kjører. Her kan man angi hvilke som er nødvendige for at tjenesten er oppe.
dns	Kobler til og slår opp angitt host. Kan ta både forlengs og baklengs.
dc	Kobler til domenekontrolleren og slår opp angitt brukernavn i brukerdatabasen/passordfilen

Tabell4: Tjenestene vi har implementeret støtte for

3.5.1 Kildeinformasjon

Det er innført en ny kildefil services.txt[10] der vi angir hvilke tjenester som skal overvåkes på hvilke servere. For tjenester der vi trenger testdata oppgis dette i kildefila. Noen eksempler er:

```
#sysname               handler args
brev.stud.ntnu.nossh
paris.stud.ntnu.no      smb    hostname=129.241.56.103
brev.stud.ntnu.no       imap   username=johndoe password=xx
brev.stud.ntnu.no       pop3   username=johndoe password=xx
ludvig.ntnu.no          dns    request=www.ntnu.no
semper2.stud.ntnu.no    http   path=~magnun/ vhost=www.ntnu.no
```

Kildefilen services.txt lastes inn i to nye databasetabeller:

1. service: Beskriver de tjenestene som skal overvåkes på en gitt server. Felter er:
   - active: active=f betyr at tjenesten er på service (og da ikke skal overvåkes).
   - handler: Tjenesten, f.eks. ssh,dns. Dersom standard port *ikke* brukes, sett serviceproperty.property='port'.
   - version. Versjon av handler, f.eks. SSH-1.99-OpenSSH_3.4p1.

2. serviceproperty: Brukes til å sende ekstra parametre (0-N) med en tjeneste. Det kan for eksempel være brukernavn/passord som skal brukes i imap-testen, eller hostoppslaget som skal gjøres i dns test. Serviceproperty brukes også til å angi at en ikke standard port er brukt.

3.5.2 Scheduler

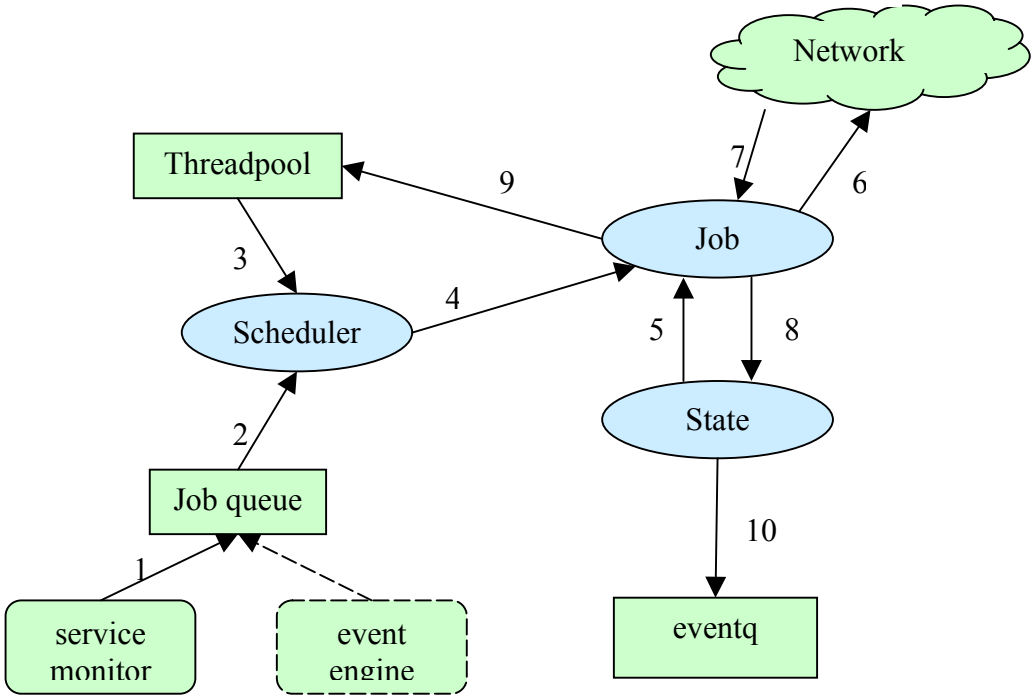
En konfigurasjonsfil (services.conf) angir sentrale parametre for tjenesteovervåkeren. De viktigste er:

Parameter	**Forklaring**
checkinterval	Hvor ofte tjenestene skal sjekkes (default 90 sekund)
maxthreads	Maksimum antall tråder. (default 50)
timeout	Hvor lenge man skal vente på svar fra en tjeneste (default 5 sekund). Denne kan sågar justeres individuelt pr tjeneste.
retry	Hvor mange tester vi skal kjøre før tjenesten erklæres for nede (ved at det postes på eventkø). Default er 3.
retry delay	Hvor stort opphold det skal være mellom Retry-tester (default er 5 sekund).

Tabell 5: Parametre til tjenesteovervåkeren

[10] Det er mulig denne på sikt vil smelte sammen med server.txt.

Overordnet algoritme til servicemonitoren er vist på figur 7. Vi forklarer i teksten under de enkelte tallene som er angitt på figuren.



Figur 7: Tjenesteovervåkeren

1. Jobb plasseres i jobbkøen ved hvert checkinterval. Her unntas tjenester fra servere som er markert som nede av event engine (sees av netbox.up feltet).
   - Vi bruker halve rundetiden på å legge til en ny tjeneste i jobbkøen. Legger altså til en ny tjeneste hvert rundetid/(antall jobber * 2) sekund. Dette for å fordele oppdragene utover den tilgjengelige tiden.
   - I fremtiden kan jobboppdrag komme fra event engine, dette er ikke implementert pr dato.
2. Jobbene i jobbkøen blir kjørt så snart vi har en tråd ledig. I praksis vil dette si at de blir kjørt så fort de blir plassert der.
3. Henter jobb med høyest prioritet.
4. Jobben starter i tråden. AntallTester=1.
5. Jobben sjekker tilstand mot State.
6. Jobben sender forespørsel mot server/tjeneste (ved å bruke aktuell handler plugin).
7. Får respons innen timeout sekund.
   - Dersom man *ikke* får svar og dersom AntallTester < retries:
     - Inkrementer AntallTester.
     - Ligg idle i retry delay sekund.
     - Returner til 6 for en ny test.
8. Rapporterer resultat til State (enten oppe eller nede).

9. Jobben terminerer og tråden markeres som ledig.
10. Dersom tilstanden er endret oppdateres event køen.

Vi ser noen svakheter ved designet. Tjenester som er nede vil unødig holde opp en tråd i påvente av en ny test. En bedre logikk tillater at nye jobber mot samme tjeneste settes tilbake i jobbkøen og tråden frigjøres. Dette er noe vi vurderer å bygge om. Problemstillingen er først aktuell ved et massivt utfall av tjenester (og et massivt utfall av bokser vil ikke påvirke dette, da monitoreringen avgrenses hele tiden til bokser som er oppe).

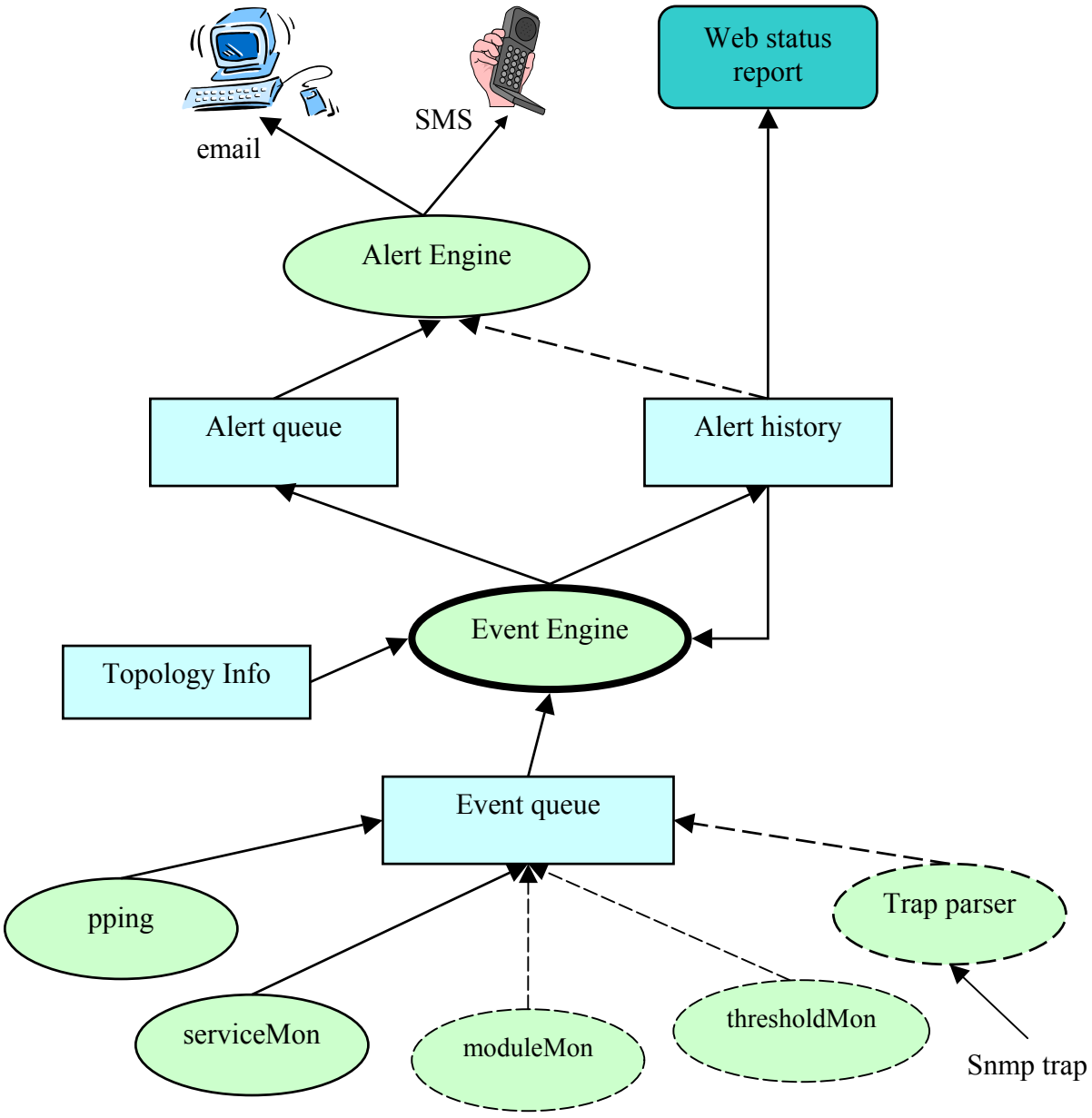
3.6 Event engine (deloppgave 17)

I NAV versjon 2 utgjør Trapdetect hendelses- og varslingssystemet. Grensesnittet for å rapportere hendelser er snmp trap. I NAVMore har vi ønsket å endre på dette. Vi ser nødvendigheten av å skille hendelses-håndtering fra varsling. Videre er snmp trap et unaturlig grensesnitt for kommunikasjon mellom NAV-interne delsystemer. I NAVMore har vi i stedet innført en eventkø implementert som en databasetabell. Det nye hendelses- og varslingssystemet er vist på figur 8.

Hendelsessystemet, heretter omtalt som event engine, er hjertet i det nye systemet. Hovedoppgaven er å ta i mot hendelser, se disse i sammenheng og se dem opp imot topologi informasjon. Event engine skal ideelt sett se helheten og basert på dette rapportere kvalifiserte alarmer til alarmsystemet, heretter kalt alert engine. Alarmene som rapporteres kan ha ulik alvorlig-hetsgrad (severity). Alvorlighetsgraden kan være satt (evt endret) i hendelsessystemet eller i det opprinnelige systemet som rapporterte hendelsen. Merk at det ikke nødvendigvis er et 1:1 forhold mellom events og alerts. Events kan bli slått sammen eller forkastet i event engine.

3.6.1 Eventkøen

Som figur 8 viser så rapporterer pingeren (pping) og tjenesteovervåkeren (serviceMon) til eventkøen. Flere systemer vil komme til etter hvert. Det vil også bli mulig å rapportere hendelser fra eksterne system via snmptrap (evt også epostmottak). Det blir således mange leverandører av hendelser. De rapporterer alle på samme måte, ved å lage en ny post i eventq-tabellen (og evt lage tilhørende eventqvar poster). Event engine lytter i sin ende jevnlig på denne tabellen og tømmer den (sletter poster).



Figur 8: Event engine og alert engine

Eventq har følgende felter:

Felt	Forklaring
source	Hvilken prosess som er opphavet til eventen, for eksempel pping, serviceMon etc.
target	Normalt event engine, men kan for eksempel brukes av event engine for å gi oppdrag til ppingeren m.fl.
eventtypeid	Hva slags hendelse (boxState, serviceState etc)
time	Når hendelsen inntraff.
deviceid	Hvilken device/boks vi her har med å gjøre.
subid	Sekundær variabel som beskriver opphavet til hendelsen. Subid gis ulik tolking for ulike hendelser. For serviceState er subid tjenesten.
value	Verdien til hendelsen. 100 er en oppemelding, 0 en nedemelding. Verdier i mellom kan angi gråsoner (ikke benyttet for pping og servicemon).
state	Angir om hendelsen er tilstandsløs (x) eller om det er initiell sykmelding (s=start) eller friskmelding (e=end)
severity	Angir alvorlighetsgrad, verdi fra 0 til 100. Infomeldinger, info om softwareoppgradering m.m. vil få med lav severity. Boks nede får høy severity.

Tabell 6: Felter i eventq-tabellen

I tillegg er det mulig å knytte en eller flere eventqvar poster til en hendelse. Denne tabellen har to felter: var og val. var er variabelnavn, val er verdien. På denne måten kan et sett med variable følge hendelsen. En anvendelse er feilmeldinger som tjenesteovervåkeren har fanget opp.

3.6.2 Event Engine programmet

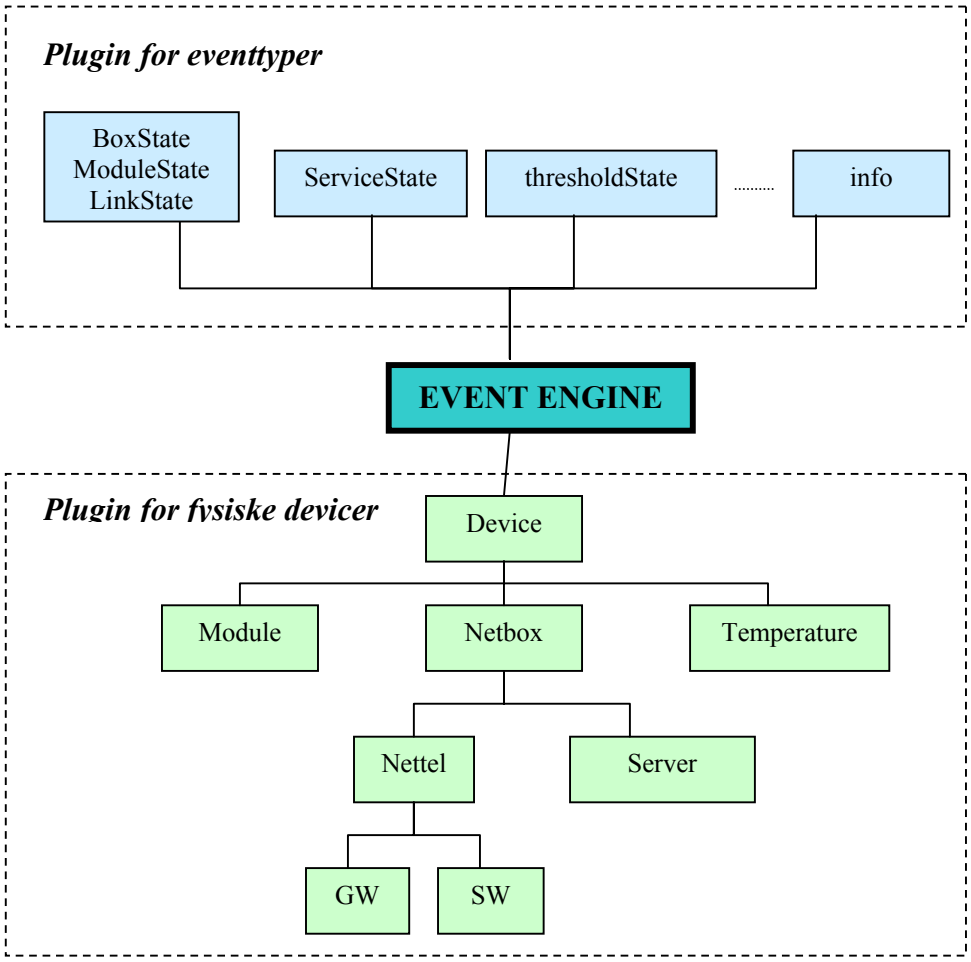
Event engine er implementert i java med en objektorientert design. Selve programmet har ingen innebygd logikk, den tar seg kun av felles oppgaver som å lese konfigurasjonsfilen, håndtere diverse timere, holde styr på objekter og laste inn plugin-moduler som utfører den faktiske jobben. Plugin-modulene deles i to typer; en som representerer fysiske ting (devicer), og en som behandler events. Figur 9 gir en oversikt.

Devicer

For devicer har vi implementert et arvehierarki, med en klasse Device i bunnen (vi regner med at alle events vil være knyttet til en device i device-tabellen). Denne abstrakte klassen har generelle metoder som up() og down() (devicen er gått ned eller kommet opp), og så arver man den for å få inn mer spesialiserte ting. F.eks NetBox vil være en Device som er koblet opp et sted på nettet og dermed kan stå bak en port, Nettel vil igjen arve NetBox og få med metoder som linkDown(), linkUp(), warmStart() osv. Server vil også arve NetBox og få med serviceDown(), serviceUp().

For å organisere alle disse objektene er det en felles klasse (som er en del av event engine) DeviceDB som inneholder referanser til alle Device-objekter og har metoder for å få tilgang til dem.

Å få opprettet objektene fra data i tabellene er litt komplisert da alt dette er plugin-moduler som event-engine i utgangspunktet ikke vet noe om. Device får en statisk metode updateFromDB() (kan naturligvis overstyres i sub-klasser) som leser inn data fra nødvendige tabeller og oppdaterer DeviceDB. Ulempen er at man gjør flere SQL-kall enn nødvendig, har man f.eks klasser GW, SW og Server vil hver av disse gjøre sine egne databasekall, men siden disse i utgangspunktet kan være interessert i helt forskjellige data fra ulike tabeller må det bli slik.



Figur 9: Event Engine klassehierarki

Events

Så var det den andre typen plugin-moduler, de som faktisk behandler events. Når event engine leser ut en event fra eventq sjekker den om den har en event handler for denne typen event, og gir den så videre til den. Event-handleren må så sjekke hva eventen sier, f.eks kan det være en boxState-event med verdi ”Down”, i så tilfellet ber den DeviceDB om en referanse til den aktuelle boksen og kaller down() på den. Nå er det opp til implementasjonen av down() (som kan være forskjellig alt etter om dette er en GW, SW, server (eller alt hva man har skrevet klasser for) hva som skal skje.

F.eks kan en timer bli startet, og etter 30 sekunder sjekkes det hvilke bokser som det fortsatt er kontakt med (det kan jo være flere som går ned, og vi gir litt tid for å få inn rapporten fra alle), og så sendes det videre nede- og skyggemeldinger (ved at det postes på alertq og alerthistory). Viser det seg at boksen er kommet opp igjen vil vi f.eks sende en boxDownTransient event videre; uansett er dette opp til implementasjonen i plugin-modulen.

3.6.3 BoxState skyggeforhold algoritmen

En av de viktigste oppgavene når det gjelder boxState alerts er å si om en boks er i skygge eller ikke. Selve algoritmen er ikke så stor, bare 20-30 linjer, nå som alt rundt den er kommet på plass. Algoritmen er slik:

1) Spør

- For en boks som er varslet nede (av pingeren via eventkøen):
  - Gå gjennom alle boksens uplinker *på vlanet boksen står*:
    - For hver uplink spør boksen over:

Er nettet er nåbart via deg?

Boksen over vil på samme vis gå gjennom sine uplinker, dette er altså rekursivt.

2) Svar

- Rekurseringen stopper dersom man kommer til en ruter; den returnerer da enten 'ja' eller 'nei' avhengig av om den er oppe eller nede.[11]
- Det stopper også hvis man kommer til en boks som er nede, den sier da 'nei' (nettet er ikke nåbart via meg).
- Dersom en boks ikke kan avgjøre om nettet er nåbart via seg, så returnerer den 'vet ikke'; dette vil skje dersom en boks ikke har gyldige uplinker, altså ved manglende topologi.

3) Tolk

- En boks går gjennom svarene fra sine uplinker og returnerer selv en verdi basert på dette:

Svar fra uplinkene	Returner selv
Minst en 'ja'	'ja'
Ingen 'ja', minst en 'nei'	'nei'
Bare 'vet ikke'	'vet ikke'

- Dersom den opprinnelige boksen returnerer 'nei' står den i skygge.
- Dersom den returnerer 'ja' eller 'vet ikke' står den *ikke* i skygge.

Denne algoritmen takler korrekt mange "vanskelige" tilfeller, f.eks der et vlan går gjennom en boks som har sin IP på et annet vlan, som godt kan

[11] Algoritmen takler ikke at rutere er i skygge enda, denne problemstillingen er utsatt.

være oppe, mens en boks under den boksen (på det andre vlanet) likevel kan stå i skygge.

Det er også verdt å merke seg at vi ikke gjør noen "if box = gw, if box = sw, if box = srv" i algoritmen, dette løses i steden med arving, noe som gjør at hvis vi skal endre algoritmen for rutere så trenger vi ikke å røre koden for svitsjer i det hele tatt.

3.7 Alert køen

Event engine poster kvalifiserte alarmer på alertkøen, dvs alertq (jfr fig 8). Alertq er tilsvarende i oppbygning som eventq (den har ikke target feltet, men er ellers lik). Også analogt med eventkøen finnes en alertqvar. Her stopper imidlertid likheten. Alertqvar har følgende felter:

- msgtype: sms, email (evt flere i fremtiden)
- language: no(rsk), en(glish)
- msg: selve meldingen som skal sendes ut, ferdigformattert.

Et eksempel på en ferdigformattert alertq.msg er (sms-melding på norsk):

Tjeneste imap på boks gekko.stud.ntnu.no er nede siden 2002-12-06 01:59:13.

Vi har altså valgt å la event engine ferdigformattere meldinger. Alternativt kunne alert engine gjort dette, men da måtte alert engine ha detaljkjennskap til hendelsen, noe event engine uansett har. Vi har ønsket å gjøre alert engine ”dum” i så måte. Fokus for alert engine er å sende ut alarmer. Primær utfordring for alert engine er å vite hvem som skal ha hvilken alarm, på hvilket språk, på hvilken form (sms, epost) og til hvilken tid (med en gang, oppsamlet 1 gang i døgnet eller…).

For at en NAV-installasjon lett skal kunne endre formatteringen av meldinger har vi laget en konfigurasjonsfil, alertmsg.conf, som event engine bruker. Et utdrag fra alertmsg.conf er vist:

```
boxState {
  boxDown {
    email {
      no {
        Subject: Boks $sysname er nede
        Dette er en automatisk generert melding fra NAV:

        Boks $sysname er nede siden $time.
      }
      en {
        Subject: Box $sysname is down
        This is an automatically generated message from NAV:

        Box $sysname is down since $time.
      }
    }
```

```
    sms {
      no: Boks $sysname er nede siden $time.
      en: Box $sysname is down since $time.
    }
  }
  boxUp {
  ...
    sms {
      no: Boks $sysname er oppe siden $time.
      en: Box $sysname is up since $time.
    }
}
serviceState {
...
}
```

Event engine vet ingenting om hvilke brukere (om noen) som skal ha en gitt melding, heller ikke på hvilket språk, eller i hvilken form (sms, epost). Av den grunn må event engine alltid lage n*m alertqvar-poster for hver alertq-post, der n er antall meldingstyper (msgtype) og m er antall språk (language). Alternativt måtte alert engine formatert meldinger og da typisk i hvert enkelt brukertilfelle, det er heller ikke optimalt. Vi har diskutert en del pro/contra her før vi landet.

3.8 Alert engine

Alert engine er implementert i perl og kjører som en demon. Den poller jevnlig alertkøen for nye alarmer. Dernest sjekker den alle brukerprofilene for å se om noen skal ha aktuelle alarm, og i tilfelle på hvilket format og språk, og til hvilken tid (oppsamling av alarmer støttes, en bruker kan for eksempel velge å få døgnbaserte rapporter).

Når alle interesserte brukere har fått alarmen vil alert engine slette alertq-posten og alle tilhørende alertqvar-poster. Algoritmen alert engine jobber etter er illustrert på figur 10.

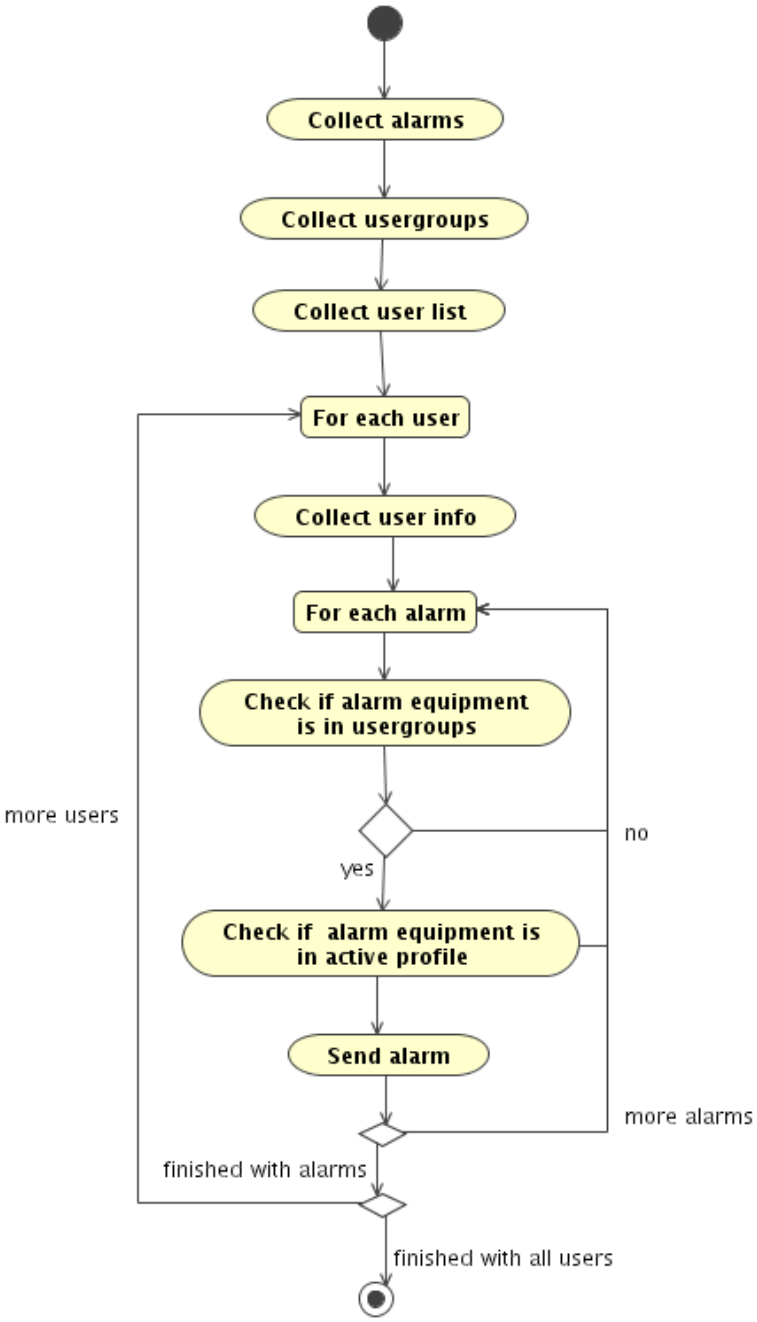
Alert engine er diskutert og spesifisert som en aktivitet innenfor NAVMore-prosjektet i tett samarbeid med UNINETT. Selve implementeringen er utført av UNINETT og er å betrake som et ”samarbeidende resultat”.

3.9 Brukerprofiler for varsling

UNINETT har også implementert en ny løsning for brukerprofiler (ved hjelp av php). Den nye løsningen er mer generell en NAV v2 sin implementasjon. En svakhet i NAV v2 løsningen er at justering av varslingsprofilen er avgrenset til å ligge på boksnivå. Dersom en server kjører flere tjenester, og en drifter kun er interessert i *en* av disse tjenestene, ikke de øvrige, så er det *ikke* mulig i NAV v2 å velge kun den *ene* i varslingsprofilen. Tilsvarende kan det være andre spesielle behov/filtre en

bruker vil ønske seg. NAV v2 har klare begrensninger her (som dog er tilfredsstillende for den typen alarmer vi har i drift i dag).

UNINETT har laget en ny brukerprofildatabase. Dokumentasjon, herunder databasediagram, kan sees på http://metanav.ntnu.no/NAVMore/navuser/. Frontenden kan sees fra v3-piloten.



Figur 10: Toppnivå algoritme Alert Engine

I korte trekk så definerer man et sett med filtre i bunn. Dette settet kan lett utvides. Man kan ha filtre som går på eventtype, severity, kategori utstyr, IP-adresse, systemnavn, tjeneste eller andre kriterier. For hvert filter tilordnes en gitt variabel en gitt verdi. Disse filtermatchene settes i neste instans sammen til utstyrsfilter, som igjen settes sammen til utstyrsgrupper, som så kan settes sammen av brukeren med ønsket varslingsprofil. Vi går ikke i detalj her, men konstaterer at løsningen er mer fleksibel (og mer kompleks) enn NAV v2 sin løsning. Vi har betatestet løsningen og vil fortsatt gjøre det i oppfølgende aktivitet til neste år.

3.10 Alert history

Et viktig prinsipp som vi tar med oss fra NAV v2 (og andre systemer) er inndelingen i tilstandsfulle og tilstandsløse hendelser. Tilstandsfulle hendelser opptrer parvis, med en ”sykmelding” og en oppfølgende ”friskmelding”. Eksempler er at en boks går ned for senere å komme opp igjen. Tilsvarende for tjenester, tilsvarende også for lastterskler som går over og under en gitt terskel. Tilstandsløse hendelser, derimot, er å betrakte som enkeltstående episoder.

Alle hendelsene defineres i eventtype-tabellen, her angir vi også om hendelsen har tilstand.

For en hendelse som har tilstand er det fem parametre event engine bruker for å lete etter en oppfølgende ”friskmelding” etter den initielle ”sykmeldingen”. De er:

1. deviceid
2. netboxid,
3. subid
4. eventtypeid
5. state

De 4 første identifiserer eventen, mens state angir om det er start eller slutt (eller stateless).

For å strukturert knytte sykmeldingen sammen med friskmeldingen har vi innført tabellen alerthist (med tilhørende alerthistvar). Denne er lik alertq, bortsett fra at den har to tidsfelter: start og slutt. En tilstandsfull hendelse har altså *en* post i alerthist med angivelse av når hendelsen inntraff, og når den opphørte (ble frisk). Det er knyttet formatterte meldinger til både sykmeldingen og friskmeldingen. Her vurderer vi å utvide med en web-formattering til nytte for web status rapporten (se kap 3.11).

Alerthist blir ikke slettet av alert engine. Alert engine ser på dataene her, for å se syk og friskmelding i sammenheng, men den sletter bare fra alertq.

Ideen med alerthist er at den skal gi en komplett og strukturert oversikt over alle hendelser som har inntruffet.[12]

En mulig utvidelse er å støtte enkel sakshåndtering, en slags primitiv trouble-ticket analogt med UNINETT sin Zino. Alerthist bør da utvides med:

- userstate: Verdier: "waiting", "working", "done", muligens flere. Altså operatørsatt tilstand. Ideen er at alle hendelser, også de som har blitt friskmeldt av event engine, skal kvitteres ut av operatør/nett-/driftsvakt.
- userid: NAV-brukeren/operatøren som har tatt saken og løst den.
- comment: Kommentar lagt inn av operatør. Sier hva som skjedde, hvorfor, hvordan det ble løst, eller annet passende. Kan være blankt.

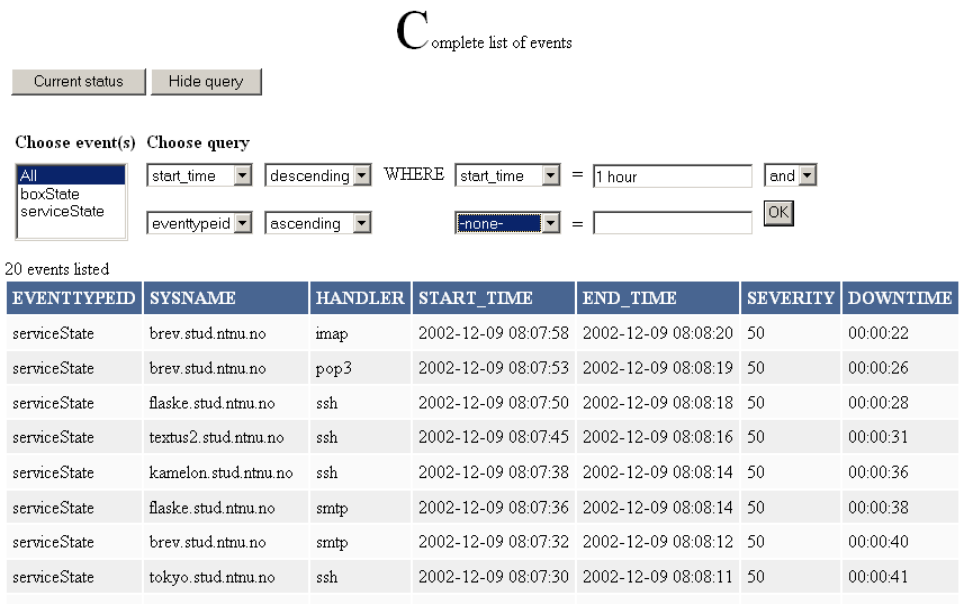
En slik utvidelse vil naturligvis kreve utvidelser også i web-frontend.

3.11 Web status rapport (deloppgave 11)

Vi har laget en web status side og en underliggende historieside, som henter data fra alerthist. Figur 11 viser en eksempelvisning fra historiesiden.

Som nevnt inngår subid-feltet som et element i den entydige beskrivelsen av en gitt hendelse. subid har ulik mening avhengig av type hendelse. For serviceState så er subid tjenesten, for thresholdState er den typisk aktuelle RRD. Event engine vet dette pr hendelse. Som nevnt ønsket vi at alert engine skulle være uvitende. Da bør heller ikke web frontend ha slik kunnskap. Løsningen vi vil gå for er en web-formattering i alertmsg.conf, som trolig vil avgrense seg til å formattere inn tolkning for subid-feltet. Pr dato er ikke dette implementert, vi har hardkodet tolkning for serviceState så langt.

[12] Det trengs enn cronjobb som sletter gamle meldinger. Dette bør være konfigurerbart, da det kan være ulike behov her. Vi kan trolig adoptere navlog sitt sletteopplegg for formålet.



Figur 11: Web-oversikt over hendelser

Kapittel 4: Videre arbeid

4.1 Arbeid som utgikk fra NAVMore

Som nevnt i kapittel 2, har vi funnet det fornuftig å la noe arbeid utgå fra NAVMore. Dette er deloppgave 4, 7, 9, 12 og 13. Vi omtaler oppgavene i korte trekk her og påpeker hva vi ønsker å videreføre.

4.1.1 Vlanutbredelse i et vindu (deloppgave 4)

Dette har vært et ønske lenge, men vi har ikke hatt tid i NAVMore. Vi tror en visualisering der man får oversikten over et helt vlan, evt en hel fysisk topologi/trunkstruktur med visning pr vlan, ville være svært nyttig. NAV databasen har denne informasjonen, og vi visualiserer den ”stegvis” i dagens nettkart og de samme dataene finnes i rapporter i rapportgeneratoren. Men vi ser altså rom for forbedring.

Vi har videreutviklet en del andre aspekter av nettkartet, så som å håndtere en sikker del med mulighet for et annet view enn usikker del. Vi mangler en konfigurasjonsfil, slik at man ved hver NAV installasjon selv kan justere hvilken popup som skal tillates på åpen og lukket del.

4.1.2 Endringshåndtering, historie, offline liv (deloppgave 7)

Ideen her er å lagre informasjon om livsløpet til en fysisk enhet (svitsj, ruter, server, eller deler av dette), fra den bestilles til den ”dør”. Dette blir en sammenblanding av manuell og automatisk innlagt informasjon. Den enkelte NAV-installasjon bestemmer selv hvilket nivå den vil legge seg på. Oppgaven er i mer detalj spesifisert under NAVMore prosjektsiden, nærmere bestemt http://metanav.ntnu.no/NAVMore/todo7.html.

Arbeidet er mye diskutert og godt spesifisert. Vi har utvidet databasen, men ikke implementert løsningen. Vi har laget en enkel front-end med strekkodeleser for NTNU sin egen del, men altså ikke fått dataene inn i system i NAV databasen. Det er et sterkt ønske å få til en løsning her, og dette vil være med i et oppfølgende prosjekt.

4.1.3 Utvidelser inventardata (deloppgave 9)

Arbeidet her skjer i mindre skala hele tiden, men vi har ikke tatt noe stort løft og ryddet rundt dette. Det er ønskelig, og vil bli gjort i forhold til den ryddigere modultabellen i v3-databasen, vi vil ikke lappe så mye mer på v2.1 funksjonaliteten her.

4.1.4 Videreutvikling datainnsamling til RRD (se deloppgave 12)

Deloppgave 12 skulle se på muligheten for å skalere opp Cricket til å håndtere langt større innsamlingsvolumer. Konkret er det ønskelig å samle inn data for omlag 30000 porter (hvilket er dagens volum av kjerne- og kantsvitsjporter ved NTNU). Vi har sett nærmere på muligheter, men ikke funnet noen tilfredsstillende løsning. Vår tilnærming har vært å parallellisere snmp-pollingen ved å lage flere parallelle cricket-trær. Vi får da riktignok redusert innsamlingstiden betraktelig, men vi sitter igjen med et betydelig CPU-problem på NAV-maskinen. Vi ser ingen god løsning på dette, RRD er veldig CPU-intensiv (ved hver oppdatering skal den trykke sammen data).

Vi har også sett på alternative frontender til RRD. En interessant kandidat er Cacti. Vi ønsker i et oppfølgende prosjekt å se nærmere på mulige alternativer/supplement, ikke primært for å forkaste Cricket, men for å få øynene opp for nye, supplerende muligheter.

Cricket slik den fremstår i dag, brukes til to ting; datainnsamler og frontend viewer. Med v3 gjør vi også datainnsamling uten Cricket. Et eksempel er responstids- og pakketapsdata der vi med egen scheduling setter data direkte inn i RRD.

En nærliggende løsning er å beholde Cricket som primær innsamler, tillate innsamling utenom Cricket (selvfølgelig), beholde Cricket sitt ”viewer-tre”, men samtidig tilby en alternativ måte å hente fram RRD-dataene. Vi har noen databasetabeller på idestadiet og vi diskuterer mulige løsninger. En strukturert måte å aksessere *alle* RRD-dataene vil være nyttig for mange anvendelser. Vi nevner:

- Sorterte statistikker for *alle* RRD data.
- Enkel kobling fra rapportgenerator til statistikk.
- Generelt bedre innfallsport til dataene enn det Cricket gir (som medfører mange menyer i mange nivå).
- Mulighet for å mer fritt sette sammen ulike data i samme bilde, for å lettere kunne sammenligne data (vi har eksperimentert litt her).
- Løsning som mer generelt lager en fleksibel terskelmonitor for *alle* RRDdata, og som herunder angir sine alarmer mer presist (en del svakheter her i dag).

4.1.5 Fleksibilitet grenseverdier og terskler (deloppgave 13)

Dette har det heller ikke blitt tid til, men det er like fullt ønskelig. Det vil bli gjort i sammenheng med oppgaven nevnt over. Og det blir et ledd i en ny thresholdMon som skal rapportere til event engine.

4.2 Andre saker

Vi har nevnt en del videreføringer tidligere i rapporten. Det er mange ting vi ønsker å jobbe videre med. Vi begynner også å få en god del innspill fra andre installasjoner (hittil mest av bugfiksende karakter).

Noen hovedsaker nevnes her:

- Lage en meldingstjeneste for å kunne legge inn driftmeldinger på statussiden. Funksjonalitet for å knytte meldinger til feilsituasjoner, vise meldinger i gitt tidsvindu m.m. Ambisjonen er at NAV sin statusside blir en samleside som gir en total IT-driftsmessig status for høgskolen/ universitetet.
- Jobbe videre med event engine med å støtte flere hendelser.
- Implementere thresholdMon og moduleMon, samt trap parser, evt også epostmottak av alarmer.
- Styrke alert engine med mulighet for køing av meldinger (muligens), samt støtte for nye filtermatcher. Tilsvarende støtte i web frontend for varslingsprofiler.
- Støtte flere tjenester i tjenesteovervåkeren. Høste mer erfaring rundt løsningen. Se nærmere på RRD datainnsamling.
- Få struktur på RRD responstids- og pakketapsdata.
- Ferdigstille frontend for registrering av data i kildefiler.
- Jobbe mer med systemdrift management, videre veivalg må først diskuteres.
- Styrke feillogging i hele NAV. Gjøre logging mer konsistent, ta i bruk NAVlog i større grad.
- Bedre støtte for trådløse basestasjoner.
- Bedre rapporter rundt maskinsporing / svitsjeportstatus.
- Støtte for nye utstyrstyper.
- Rapportgeneratoren:
  - Mulighet for mer logikk/scripting i rapportgeneratorspråket.
  - Mulighet for å lage rapporter mot flere databaser (i dag kun manage)
  - Mulighet for å automatisk lage en indeksside over alle rapportene.

Kapittel 5: Konklusjon

Prosjekt NAVMore har jobbet ut i fra en ambisiøs prosjektplan med ønske om å implementere et vidt sett med nye løsninger. Brorparten av oppgavene er løst, mer er påbegynt, mens en del er til gode og er nevnt i kapitlet om videre arbeid. Alt i alt er vi fornøyd med fremdrift og utførelse.

Vi skulle implementere løsninger innen to hovedområder:

1. NAV v2.1 funksjonalitet som skulle gjøres tilgjengelig med NAVRun sin installasjonspakke. Dette er gjennomført og funksjonaliteten er oppe og kjører på 10 NAV-installasjoner (pr dato). Vi oppsummerer:
   - Maskinsporing på svitsjeportnivå
   - Strukturert håndtering av Cisco syslogmeldinger
   - Strukturert system for NAV feilmeldinger
   - Styrket topologiavledning, lag2 og lag3
   - Utbedringer rundt datainnsamling
   - Grafisk visualisering av adresseromsforbruk

2. NAV v3 funksjonalitet skulle utvikles frem mot en testversjon for NTNU. Dette har vi også gjennomført, skjønt vi er ikke helt i mål med alle aspekter. Viktige biter er imidlertid på plass. Spesielt trekker vi frem:
   - Sorterte RRD-data
   - Cricket statistikk for servere
   - Tjenesteovervåker
   - Ny og parallell pinger (statusmonitor)
   - Nytt hendelsessystem, event engine, med grensesnitt mot nytt varslingssystem, alert engine.
   - UNINETT har selv implementert en alert engine pilot, samt et nytt system for NAV-brukeres varslingsprofiler.

Med NAVMore utvider NAV seg for første gang i retning av system-overvåkning. NAV vil alltid ha et nettfokus i bunn, men gitt at vi beholder grunnfilosofien med et modulært system, så er utvikling i bredden bare av det gode. Prosjekt NAVMore har initiert et nytt og spennende utviklings-samarbeid mellom ITEAs nett- og systemdriftgrupper, og ikke minst, mot UNINETT selv. Dette er allianser vi ønsker å bygge videre på.

Konkret avslutter NAVMore med en pilot med mye ny funksjonalitet, samt en lang liste med forslag til videre arbeid. Vi er ustoppelig, og ser frem til et nytt NAV-år i 2003 med et videre godt samarbeid internt på NTNU, med UNINETT og med evt andre interesserte universitet/høgskoler!

The NAV must go on.
q.e.d.