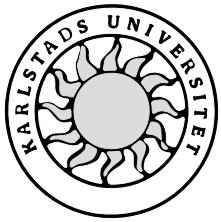


Datavetenskap

Johan Alexanderson Per-Ola Gustafsson

Design och implementation av en SNMP-baserad övervakningsserver

---

Examensarbete, C-nivå

Pubnum, Arkiv | Egenskaper | Eget

Design och implementation av en SNMP-baserad övervakningsserver

Johan Alexanderson Per-Ola Gustafsson

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Johan Alexanderson

Per-Ola Gustafsson

Godkänd, Date, Arkiv | Egenskaper | Eget

Per Strömgren: Advisor, Arkiv | Egenskaper | Eget

Stefan Lindskog: Examiner, Arkiv | Egenskaper | Eget

Sammanfattning

I takt med att Internet ökar i storlek har datornätverk bland företag blivit allt vanligare. Företagen kräver att deras nätverk skall fungera säkert och robust. Då det uppstår problem i nätverket har administratören till uppgift att åtgärda problemet. Men det är inte alltid som administratören finns i närheten då fel inträffar. Under sådana omständigheter vore det önskvärt att ha en programvara vilket kan förvarna en administratör redan *innan* ett fel inträffar. Med hjälp av protokollet SNMP är detta möjligt. Program som använder SNMP kan nämligen hämta in information om de enheter som är kopplade till nätverket och sedan behandla den erhållna informationen. Programmet kan exempelvis få fram data om hur mycket minne som finns kvar i en viss enhet eller hur tät nättrafiken är.

Denna rapport beskriver designen och implementationen av ett serverprogram som använder SNMP för att övervaka enheter i ett nätverk. Rapporten innehåller även en ingående beskrivning av vad SNMP är och hur det fungerar.

Resultatet av projektet blev en fungerande övervakningsserver som följer kravspecifikationen, men med den begränsningen att endast två av de specificerade nätverksenheterna kan övervakas.

Design and Implementation of an SNMP based monitoring server application

Abstract

Because of the growth of the Internet computer networks among companies have become more common. Companies demand robustness and functionality in their network. Problems in the network must be resolved by an administrator, but the administrator might not always be present at the time. During these circumstances it would be desirable to have an application capable of informing the administrator of problems even before they occur. The SNMP protocol makes this possible. An application that uses SNMP can retrieve local information from a network entity and manage the information. The local information can be of various type, for instance the amount of memory left in a certain entity, or the amount of network traffic.

This report describes the design and implementation of a monitoring server application based on SNMP. The report also includes a description of the SNMP protocol.

The outcome of this project is a working monitoring server application, according to the requirement specification, except that the only two of the specified network entities are monitored.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

Den här rapporten beskriver design och implementation av en övervakningsserver baserad på SNMP (Simple Network Management Protocol), ett standardprotokoll som verkar i applikationslagret och vars huvudsakliga uppgift är att kontrollera de fysiska enheterna i ett nätverk. Övervakningsservern kommer att kunna hämta in information om enheter i ett nätverk via SNMP och sedan spara denna information i en databas. Informationen i databasen kommer sedan att presenteras via ett grafiskt gränssnitt för en administratör. Det grafiska gränssnittet är utvecklat av Ataco.

För att få en bättre förståelse om vad SNMP är handlar de första kapitlen om SNMP:s uppbyggnad och funktionalitet. Därefter följer en detaljerad beskrivning om hur programmet är designat och implementerat. I de sista kapitlen presenteras projektets resultat och slutsatser.

1.1 Mål

Målet med projektet var att skapa en fungerande SNMP-baserad övervakningsserver för ett nätverk. Programmet skall kunna hämta in information om nätverksenheterna i nätverket och spara informationen i en MySQL-databas.

1.2 Kapitelöversikt

Kapitel 1: Inledning

Här förklaras vad målet med projektet är. Denna kapitelöversikt är en del av detta kapitel.

Kapitel 2: Bakgrund

I det här kapitlet beskrivs bakgrunden till projektet, d.v.s. anledningen till att arbetet kom att handla om just design och implementation av en SNMP-baserad övervakningsserver.

Kapitel 3: Förväntningar och krav

Kapitlet handlar om vilka krav som ställs på programmet.

Kapitel 4: Introduktion till SNMP

Kapitlet börjar med en kort översikt om vad SNMP är, följt av lite historik. Därefter kommer en mer ingående beskrivning av SNMP. Det innefattar beskrivning av de tre SNMP-versioner som finns, förklaring till vad managers och agenter är, hur SNMP:s struktur ser ut samt beskrivning av MIB:en och OID:er. Sedan behandlas hur inhämtning av data från agenter går till via SNMP och även vilka typer av operationer (PDU:er) som finns. Avslutningsvis tas fördelar och nackdelar med SNMP upp.

Kapitel 5: Design

Här beskrivs designen av varje modul i programmet.

Kapitel 6: Utvecklingsmiljö

Kapitlet innehåller en kortfattad introduktion till de verktyg och utvecklingsmiljöer som använts i projektet.

Kapitel 7: Implementation

Här beskrivs de konstruktionsval som gjordes under implementationen av övervakningsserverns moduler.

Kapitel 8: Testning

Här visas hur testningen av applikationen gick till och vilka testresultaten blev.

Kapitel 9: Resultat

Kapitlet presenterar projektets resultat.

Kapitel 10: Slutsatser

I det sista kapitlet tas slutsatser som kommit fram vid avslutande av projektet upp.

2 Bakgrund

Internets kolossala ökning i storlek har lett till att Internet allt oftare får problem med bland annat hastighet och säkerhet. Företag som tillhandahåller service till sina kunder via Internet kan inte tolerera att sådant inträffar. Ataco i Karlstad är ett företag som tillhandahåller Internet-tjänster och support till sina kunder. De är i behov av ett övervakningssystem för övervakning av enheterna i deras nätverk. Ataco har övervägt lösningar från kommersiella företag, t.ex. HP Openview och Tivoli, men dessa har visat sig vara alltför dyra. Det finns även flera kostnadsfria alternativ, så kallade "open source system", att tillgå, men dessa har visat sig sakna den funktionalitet som Ataco behöver. En tredje lösning är då att designa och implementera ett eget övervakningssystem med hjälp av SNMP. Denna rapport handlar om hur man löser en sådan uppgift.

3 Förväntningar och krav

3.1 Inledning

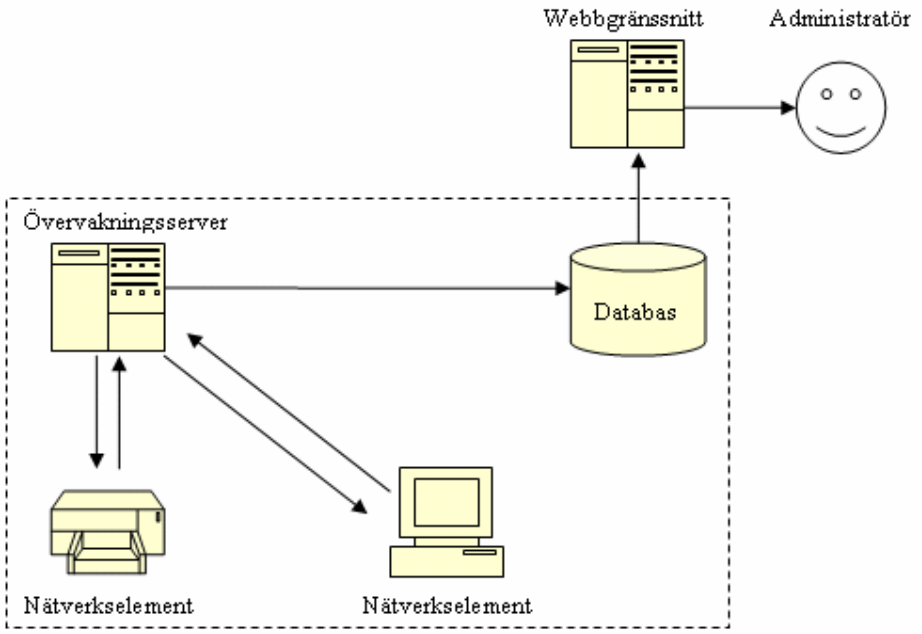
SNMP används till att ta fram information om enheter i ett nätverk, till exempel ta fram en viss enhets resterande hårddiskutrymme eller hur mycket minne som den utnyttjar för tillfället. Denna information om en viss enhets status kan man utnyttja för att förebygga att fel inträffar i nätverkets enheter. Uppgiften som företaget Ataco ställt är att göra ett övervakningsprogram som med hjälp av SNMP kan övervaka enheter i ett nätverk och ta fram information om deras status. Informationen skall sedan sparas i en databas, vilket kommer att presenteras via ett grafiskt gränssnitt för en administratör. Det grafiska gränssnittet är utvecklat av Ataco.

3.2 Översikt över systemet

Nätverket som skall övervakas med hjälp av SNMP består av tre viktiga komponenter: nätverksenheter, agenter och manager.

Nätverksenheter är noder i nätverket, vilka samlar in och lagrar information om sin egen status. I det här projektet är nätverksenheter servrar, klienter, skrivare och switchar. Nätverksenheterna server och klient behandlas på samma sätt och den enda anledningen till att de åtskiljs är för eventuella framtida förändringar av systemet. Agenter är mjukvara som finns installerad i nätverksenheten. Det är via agenten som managern kan hämta information om nätverksenheten. Agenter ingår i de flesta nätverksenheter, men kräver oftast en installation.

Managern är en applikation som finns installerad i den enhet, i figur 1 kallad övervakningsservern, som har till uppgift att övervaka nätverket. En manager hämtar information från nätverksenheterna och lagrar den i en databas. Databasen i det här projektet har Ataco designat. Via ett grafiskt gränssnitt, även det konstruerat av Ataco, kan den lagrade nätverksinformationen presenteras för administratören. Att designa och implementera en manager är vad det här projektet går ut på, i figur 1 angiven med streckad ruta.



Figur 1 Systemöversikt.

3.3 Krav

Övervakningssystemet skall uppfylla följande krav:

- Övervakningssystemet skall bestå av ett program som använder SNMP för att hämta in information om nätverksenheter.
- Programmet skall kunna ta fram data om nätverksenheterna klienter, servrar, skrivare och switchar.
- All information som hämtas in från nätverksenheterna skall lagras i en databas.
- Det skall inte finnas någon begränsning av hur många nätverksenheter som kan övervakas.
- Programmet skall helst vara implementerat i C/C++.
- Koden skall vara mycket väl dokumenterad. Detta för att andra skall kunna redigera koden i framtiden utan problem.
- När övervakningssystemet är klart skall det förpackas i deb-filer, för enkel installation under Debian.

4 Introduktion till SNMP

Med hjälp av Internet-protokollet SNMP går det att förhindra att problem i nätverksstrukturen inträffar. SNMP kan nämligen avläsa status på resurser i ett nätverkselement, innan problemen inträffar, och på så sätt förvarna administratören, som då kan vidtaga åtgärder. De flesta nätverkselement har stöd för SNMP. Resurserna som kan avläsas från ett nätverkselement kan vara allt ifrån IP-adress till hårddiskstorlek.

SNMP är server/klient-baserat vilket betyder att servern och klienten skickar SNMP-meddelanden mellan varandra. Ett SNMP-meddelande definierar ett kommando som skall utföras på en klient. Ett sådant kommando kallas för PDU (Protocol Data Unit). Serverapplikationen heter i SNMP-sammanhang manager och det kan finnas en eller flera manager i en övervakad nätverksstruktur. De övervakade nätverkselementen innehåller en mjukvara för hantering av SNMP-meddelanden, vilken kallas för agent. En agent använder en intern databas, en MIB (Management Information Base), i vilken namnen på de övervakade resurserna lagras.

4.1 Historik

Under senare delen av 80-talet ökade Internet drastiskt i storlek då fler och fler företag började utnyttja det. Detta medförde att det blev mer instabilt och att prestandan kunde sjunka till en oacceptabel nivå. Det blev även mer arbetsamt för de administratörer som underhöll trafiken över nätet då de i många fall var tvungna att manuellt utföra kontroller. För att motverka detta togs ett beslut av IAB (Internet Architecture Board) att införa ett protokoll med vars hjälp det är möjligt att övervaka nätverk automatiskt. 1988 sattes tre förslag upp: HEMS, SNMP och CMIP. HEMS och SNMP var protokoll som används i TCP/IP-baserade nätverk. HEMS var ett mer kapabelt protokoll än SNMP, men eftersom man förutspådde att TCP/IP snart skulle vara historia till förmån för OSI ville man inte gärna lägga ner så mycket arbete på att utveckla ett avancerat protokoll för TCP/IP. I stället satsade man på SNMP som gick snabbt att utveckla. CMIP, som var ett OSI-baserat protokoll, skulle sedan användas då TCP/IP dött ut. TCP/IP dog aldrig ut och det visade sig att SNMP blev så pass populärt att det än idag är det gällande protokollet för övervakning i de flesta av dagens nätverk.

4.2 Olika versioner av SNMP

Strukturen på de meddelanden som utgör Internettrafiken bestäms av protokoll. I detta sammanhang är ett protokoll en standard framtagen av IETF (Internet Engineering Task Force). IETF är en öppen internationell organisation, vilket innebär att de som så önskar kan ta del i arbetet med att utveckla och ta fram nya protokoll. Protokollen dokumenteras och är tillgängliga för användare i dokument som kallas för RFC:er (Request for Comments). Ett RFC-dokument har en dokumenterad status vilken kan vara av typen *förslag, utkast* eller *standard*. Väldigt få av de RFC:er som existerar idag har status av typen standard, men de flesta företag och utvecklare följer de specifikationer av ett protokoll som beskrivs i en RFC.

SNMP är ett av de protokoll som beskrivs i RFC:er. SNMP har sedan slutet av 80-talet utvecklats och anpassats till de krav som nya produkter och tillämpningar ställer på ett övervakningsprotokoll. Detta har resulterat i att det i dagsläget finns tre versioner av SNMP.

4.2.1 SNMPv1

SNMP Version 1 (SNMPv1) är den äldsta versionen av SNMP, och är den enda av de olika SNMP-versionerna som räknas som en regelrätt standard. SNMPv1 beskrivs i RFC 1157.

Ett av problemen med SNMPv1 är dess bristande säkerhet. Säkerheten i SNMPv1 är baserad på ett klartextlösenord vilket benämns "community". En applikation som känner till lösenordet tillåts åtkomst av resurser hos ett övervakat nätverkselement. Standarden anger att det skall finnas tre olika lösenord; ett då en applikation skall avläsa status för en resurs, ett annat när en applikation skall ändra status för en resurs, och ett tredje när ett nätverkselement skickar en notis om en statusförändring till en övervakningsserver. Dessa lösenord kallas på engelska för read-only, read-write och trap.

4.2.2 SNMPv2 och SNMPv2c

I början av 1990-talet påbörjades arbetet med att förbättra SNMPv1 där ett av målen med arbetet var att höja säkerheten och förhindra avlyssning och intrång. I mars 1993 publicerade IETF ett förslag för SNMP version 2 (SNMPv2). Under 1996, då arbetsgruppen hade tre års erfarenhet av SNMPv2 omarbetades RFC:en och i samband med detta uteslöts säkerhetsförbättringarna i SNMPv2, varför den nuvarande versionen av SNMPv2 kallas för "community

based SNMPv2", eller SNMPv2c. Anledningen till att alla säkerhetsförbättringar som fanns med i 1993 års RFC togs bort ur 1996 års version är att det fanns lite intresse från leverantörer och användare för de förslagna förändringarna. När omarbetningen av RFC:en påbörjades 1996 var det med förhoppningen att det skulle räcka med några mindre ändringar i specifikationen och att det inte skulle ta lång tid att införa förändringarna. Men när arbetet nästan var färdigt påvisade en av projektmedlemmarna att säkerhetslösningen var felaktigt designad, och det hela slutade med att säkerhetsförändringarna i SNMPv2 uteslöts ur RFC:en. Trots detta innehåller SNMPv2c nyheter och de mest betydelsefulla är två nya SNMP-meddelanden, GetBulkRequest och InformRequest. Med hjälp av meddelandet GetBulkRequest kan en övervakningsserver ställa en fråga till ett nätverkselement med begäran om status för mer än en resurs, och nätverkselementet kan svara med ett gemensamt svar för samtliga efterfrågade resurser. I SNMPv1 måste en övervakningsserver skicka ett SNMP-meddelande till nätverkselementet för varje resurs. GetBulkRequest leder således till minskad nätverkstrafik. InformRequest används mellan övervakningsservrar för att utbyta information.

Arbetet med SNMPv2 är dokumenterat i RFC 1901-1908.

4.2.3 SNMPv3

För att råda bot på säkerhetsproblemen i SNMPv2 bildades flera oberoende grupper vilka arbetade för att hitta lösningar på säkerhetsproblemet, och åtminstone två nya versioner av SNMPv2 presenterades av dessa grupper. De oberoende gruppernas arbete resulterade i att IETF återupptog sitt arbete, och efter nästan ett års arbete publicerade IETF fem nya RFC:er, RFC 2271-2275, vilka beskriver SNMPv3. Dessa RFC:er beskriver en övergripande arkitektur och säkerhetsförändringar men definierar inga nya PDU:er. En SNMPv3-implementation består således av den arkitektur och de säkerhetsförändringar som beskrivs i RFC 2271-2275, samt de SNMP-meddelanden och funktionalitet som beskrivs i SNMPv2, dvs. RFC 1901-1908.

4.3 Manager och Agenter

En nätverksstruktur som övervakas eller administreras med hjälp av SNMP består av ett eller flera nätverkselement. Ett av nätverkselementet är speciellt då det innehåller en programvara för övervakning av systemet, kallad manager. Ett nätverkselement är utrustning i en nätverksstruktur som använder en TCP/IP stack vid kommunikation t.ex. värddatorer, routrar, switchar, skrivare etc. Nätverkselementen innehåller mjukvara avsedd för hantering av SNMP-meddelanden. Denna mjukvara brukar kallas agent. En agent har till uppgift att samla

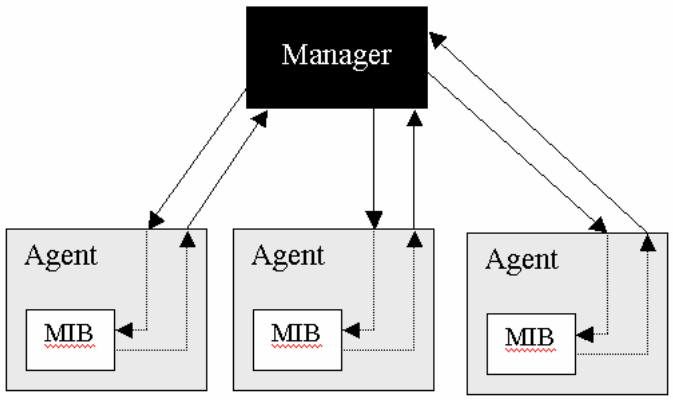
statistik om nätverkselementet t.ex. nätverksrelaterade aktiviteter och lagra den insamlade statistiken lokalt i nätverkselementet. Managerns uppgift är att tillhandahålla ett gränssnitt mellan användare och den övervakade nätverksstrukturen. Detta består av att presentera den lagrade informationen insamlad från agenter och ge användaren möjlighet att ändra variabelvärden hos agenter.

Kommunikationen mellan en manager och en agent sker med hjälp av utbyte av SNMP-meddelanden, vilka delas upp i tre olika typer:

- *get* tillåter en manager att avläsa ett variablevärde hos en agent.
- *set* tillåter en manager att ändra ett variabelvärde hos en agent.
- *trap* tillåter en agent att meddela en manager om signifikanta värdeändringar i dess variabler.

Resurser i ett nätverkselement har samtliga ett namn och ett värde associerat till namnet. Variablerna är strukturerade i en MIB, vilken utgör accesspunkter mellan en manager och en agent.

I figur 2 visas hur en manager kommunicerar med agenter i nätverket och hur en agent använder MIB:en för att verifiera att den efterfrågade variabeln existerar och har rätt format.



Figur 2 Kommunikation mellan en manager och agenter.

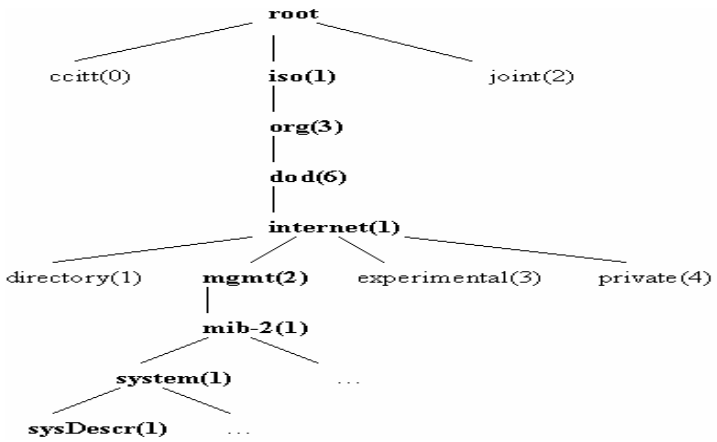
4.4 Informationsstruktur

Samtliga nätverkselement vilka ingår i en SNMP-övervakad nätverksstruktur upprätthåller en databas över de resurser en manager kan övervaka eller administrera. Resurser representeras internt inom ett nätverkselement av variabler och är strukturerade och lagrade i en MIB.

För att en manager skall kunna hantera variabler oberoende av tillverkare och typ av nätverkselementen, behöver nätverkselementens MIB:ar följa ett fördefinierat regelverk. Regelverket för variablerna kallas SMI (Structure of Management Information). SMI definierar en variabels syntax, d.v.s. vilken datatyp en variabel har, likaså definieras hur variablerna namnges och representeras (eller kodas som är den term som används i detta dokument). En av drivkrafterna bakom SMI är att MIB:ar skall vara enkla och möjliga att utöka. Detta medför att en MIB endast kan hantera enkla datatyper: skalärer och vektorer av skalärer. Mer komplexa datastrukturer kan inte hanteras av MIB:arna och således inte heller av SNMP.

4.4.1 MIB-struktur

Variablerna i ett nätverkselement är ordnade i en trädstruktur. Löven i trädet utgör de övervakade variabelvärden, vilka representerar de resurser som kan övervakas i nätverkselementet. Variablerna i trädet är i sin tur grupperade i logiskt relaterade grupper. Varje variabel i trädet kan unikt identifieras med hjälp av en s.k. OID (Object Identifier) som är en sekvens av heltal separerade med punkter, där varje heltal identifierar de noder i trädet som ingår i en väg räknat från trädets första nivå ner till aktuell variabel. Vägen i trädet motsvaras av att läsa en OID från vänster till höger, se Figur 3.



Figur 3 Exempel av en MIB med OID:en för sysDescr.1

Samtliga noder i trädet har också ett namn som kan användas istället för OID:en. Variabeln sysDescr.1, se figur 3, vilken används för att en systemägare skall kunna ge en valfri beskrivning av ett nätverkselement har OID 1.3.6.1.2.1.1.5.1, vilket motsvaras av namnet iso.org.dod.internet.mgmt.mib-2.system.sysDescr.1. Detta betyder att om man följer vägen från roten i trädet så finns det en nod som heter iso vilken har ett subträd vid namn org som

har ett subträd vid namn iso etc. Detta upprepas till dess ett löv i trädet påträffas, som i figur 2 är sysDescr.1. Namnet på noden iso motsvarar första heltalet i strängen 1.3.6.1.2.1.1.5.1 dvs. 1. Likaså motsvarar namnet org det andra heltalet i samma sträng osv. Det subträd som normalt är av intresse för SNMP är mib-2, vilket medför att de variabler som normalt används alltid har en OID som börjar med 1.3.6.1.2.1. Eftersom OID:er är sekvenser av heltal kan de ordnas lexikografiskt, vilket medför att en manager kan traversera ett träd genom att fråga efter nästkommande variabel i lexikalisk ordning. Som exempel på den lexikaliska ordningen kan vi studera noden system, .iso.org.dod.internet.mgmt.mib-2.system, vilken har sju barn. Barnen är ordnade lexikografiskt vilket visas i tabell 1 nedan. En manager kan be en agent utföra ett kommando på samtliga variabler eller barn till noden system genom att be agenten operera på nästkommande variabel i lexikalisk ordning, med start i noden system.

OID	**Namn**
1.3.6.1.2.1.1.**1**	sysDescr
1.3.6.1.2.1.1.**2**	sysObjectID
1.3.6.1.2.1.1.**3**	sysUpTime
1.3.6.1.2.1.1.**4**	sysContact
1.3.6.1.2.1.1.**5**	sysName
1.3.6.1.2.1.1.**6**	sysLocation
1.3.6.1.2.1.1.**7**	sysServcices

Tabell 1 Samtliga OID:er under noden system listade i lexikalisk ordning

4.4.2 Syntax

Varje variabel i en MIB har en formell definition vilken definierar variabelns datatyp, tillåtna former och värdeintervall, samt relationer till andra variabler inom MIB:en. Den formella definitionen uttrycks med hjälp av ASN.1-notation (Abstract Syntax Notation One). Denna används också för att definiera hela MIB:en. Som tidigare nämnts är en av drivkrafterna bakom SMI att MIB:ar skall vara enkla varför endast en delmängd av ASN.1 används för att definiera MIB:ar. En variabel kan definieras som en av typerna listade i tabell 2.

Typ	**Beskrivning**
integer	Heltal.
octetstring	Sträng.
null	Null.
object identifier	OID identifierar en positionen för ett löv eller nod i en MIB.
sequence	Tabell konstruerad av en av de fyra ovan beskrivna typerna.
sequence-of	Samma som sequence.
networkaddress	Tillåter val av adress format av en ur flera protokollfamiljer. I dagsläget tillåts endast protkollfamiljen ipaddress.
ipaddress	32-bitar heltal som identifierar en IP-adress.
Counter	Räknare: Icke negativt heltal som kan ökas men inte minskas. Maxvärde 2^32 - 1. När räknaren når maxvärdet slår den om och börjar från noll igen.
Gauge	Icke negativt heltal som kan ökas och minskas. Samma maxvärde som counter. Då maxvärdet nås behålls värdet till dess det sätts till ett nytt.
timeticks	Icke negativt heltal som räknas antalet hundradelar av en sekund från en angiven tidpunkt.
Opaque	Används för egendefinierade datatyper.

Tabell 2 ASN.1 Datatyper

4.4.3 Kodning

Värden av en ASN.1-typ kodas enlig basic encoding rule (BER), vilket innebär att alla värden kan representeras som en oktettsträng.

4.5 UDP

Kommunikationen mellan en manager och en agent sker genom utbyte av UDP-meddelanden. UDP (User Datagram Protocol) är ett förbindelselöst och opålitligt protokoll för överföring av enskilda datagram (datapaket) över IP. UDP upprättar ingen förbindelse mellan sändare och mottagare av datagram och protokollet ger inga garantier att datagram som skickas mellan sändare och mottagare kommer fram till mottagaren i samma ordning som de skickades.

På grund av UDP:s karaktär måste managerapplikationen avgöra om ett datagram som skickas når sin mottagare eller ej. Om ett datagram förloras kan sändaren välja att sända datagrammet igen, eller avstå från att sända ett nytt datagram. Denna kontroll sköts vanligtvis med hjälp av en timer i applikationen som sänder datagrammet. Antalet gånger en managerapplikation skall skicka om ett datagram vid uteblivet svar bestäms av en räknare i applikationen.

En managerapplikation använder UDP-port 61 för att skicka och ta emot förfrågningar och UDP-port 62 för att ta emot statusmeddelanden från agenter.

4.6 SNMP-Protokollet

4.6.1 SNMP-Format

Informationen som skickas mellan en manager och en klient är strukturerad efter ett speciellt format, vilket kallas SNMP-format.

Nedan visas strukturen för ett SNMP-meddelande.

Version	Lösenord	PDU

Figur 4 SNMP-meddelande

Varje meddelande som skickas innehåller ett versionsnummer, vars uppgift är att identifiera vilken version av SNMP som används, ett lösenord, och en PDU. PDU är det meddelandeformat som används mellan en manager och agenter. Det finns en PDU definierad för varje operation som kan utföras hos en manager eller agent.

I SNMPv2 definieras operationerna: *get, get-next, get-bulk, set, get-response, trap, notification, inform* och *report*.

De två sistnämnda *inform* och *report* används vid kommunikation mellan olika manager. Då syftet med denna introduktion är att beskriva kommunikationen mellan en manager och agenter, så beskrivs inte dessa i detalj. PDU:en *report* har aldrig implementerats utan beskrivs i en experimentversion av SNMPv2.

Värt att notera är att PDU-begreppet i SNMP utgör en del av SNMP-meddelandet, och inte hela meddelandet, vilket är fallet i UDP-sammanhang. Ett något olyckligt namnval, då dessa båda begrepp är lätta att förväxla

4.6.2 Sändning av SNMP-meddelanden

För att sända ett SNMP-meddelande oavsett typ av meddelande genomgår den sändande enheten följande steg:

1. Konstruera PDU:en
2. Överlåt PDU:en tillsammans med käll- och destinationsadressen och ett lösenord till autenticeringsenheten. Autenticeringsenheten utför ytterligare handlingar som t.ex. kryptering.
3. Konstruera SNMP-meddelandet, d.v.s. lägger till SNMP-version, lösenord och resultatet från steg 2.
4. Koda SNMP-meddelandet och överför det till transport servicen.

4.6.3 Mottagning av SNMP-meddelanden

Vid mottagande av ett SNMP-meddelande utförs följande steg:

1. Syntaxkontrollera meddelandet. Vid syntaxfel kasseras SNMP-meddelandet utan att meddela sändaren.
2. Verifiera SNMP-version. Då den mottagande enheten och den sändande enheten inte har samma SNMP-version kasseras meddelandet.
3. Autenticera meddelandet. Om autenticeringen misslyckas kasseras meddelandet och ett trap genereras. Då autenticeringen lyckas tas PDU-delen av SNMP-meddelandet om hand.
4. Syntaxkontrollera PDU-delen. Vid syntaxfel kasseras SNMP-meddelandet.
5. Utför kommandot. Kommandot kommer dock inte utföras om rättigheterna på variabeln inte tillåter det, vilket är beroende på lösenordet som användes vid skapandet av

SNMP-meddelandet. Ett felmeddelande konstrueras och skickas som svar på kommandot till sändaren.

4.7 Trap och Pollning

En agent kan generera asynkrona meddelanden, vilka benämns trap, för att meddela managern om signifikanta värdeförändringar i enheten. Den kan generera två typer av trap, fördefinierade och leverantörsspecifika. Antalet fördefinierade trap i SNMPv1 och SNMPv2 är inte många, och ett leverantörsspecifikt trap kan kanske inte tolkas och förstås av en manager. Därför är det vanligare att en manager hämtar statusinformation om ett nätverkselement genom att polla det, d.v.s. skicka GetRequest-PDU:er med jämnt tidsintervall till en agent.

För att en användare av ett SNMP-övervakat nätverk skall kunna lita på att den information som managern presenterar för användaren är korrekt behövs en policy för hur ofta en manager skall polla varje nätverkselement. Policyn för pollningen är beroende av hur många nätverkselement det finns i det nätverk som övervakas. Om vi antar att en manager pollar ett nätverkselement åt gången, kan antalet nätverkselement som är möjliga att pollas beräknas enligt;

$N \leq T/X$, där N är antalet nätverkselement, T är det önskade poll-intervallet, och X är ett medelvärde på hur lång tid det tar att genomföra en pollning. Värdet av X är beroende av en mängd faktorer:

- Tiden det tar att generera en PDU.
- Fördröjningar i kommunikationen mellan en manager och en agent.
- Tiden för en agent att behandla en förfrågan och skicka ett svar.
- Antalet gånger förfrågan måste skickas om för att managern skall få ett svar.

Beräkningar gjorda av Ben-Artzim Chandna och Warrier i "Network Management of TCP/IP Networks: Present and Future IEEE Networks Magazine, July" visar att om önskat poll-intervall är 15 minuter och en behandlingstid på 50 ms och en fördröjning i nätverket på 1 ms så blir $N \leq 15x60/0.202 \approx 4.500$

Om man tar med i beräkningen att nätverket består av flera subnät så minskas denna siffra dramatiskt, p.g.a. fördröjningar i routrar.

Det finns således tre parametrar som påverkar vilka polliningstider som kan användas i ett SNMP-övervakat nätverk: tiden att behandla förfrågningar och svar, fördröjningar i nätverket, och antalet agenter.

4.8 SNMP-operationer

4.8.1 GetRequest-PDU

En manager genererar GetRequest-PDU:er för att begära att ett *get*-kommando skall utföras hos en agent, d.v.s. begära att värdet av en variabel i en agent skall skickas till managern.

Nedan visas strukturen för en GetRequest-PDU.

PDU-typ	Id	Fel-kod	Fel-index	Variabelbindningar

Figur 5 GetRequest-PDU

Fält	**Beskrivning**
PDU-typ	Indikerar att det är ett GetRequest-PDU
Id	Heltalsvärde som identifierar PDU:en, vilken används både av managern och agenten. Agenten använder värdet för att identifiera duplicerade GetRequest-PDU:er och managern använder värdet för att korrelera frågor med svar. Det är den sändande enheten som sätter heltalsvärdet för Id. Standarden anger inga regler för hur Id:et skapas.
Fel-kod	Används inte.
Fel-index	Används inte.
Variabelbindning	En lista över de variabler som managern vill avläsa. I en GetRequest kan denna lista endast innehålla ett element. Se Figur 7

Tabell 3 Beskrivningav fälten i GetRequest-PDU

En agent svarar på ett inkommande GetRequest-PDU med ett GetResponse-PDU som innehåller samma värde i fältet "Id" som managern skickade med i GetRequest-PDU:en. Ett GetRequest-kommando är atomärt vilket innebär att agenten antingen skickar ett värde för variabeln till managern eller, då ett fel uppstår, inte skickar något värde alls. Om ett fel har uppstått svarar agenten med att skicka en GetResponse-PDU, vilken använder fälten "Fel-kod" och "Fel-index" för att meddela sändaren av GetRequest-PDU:en vad som orsakade felet.

Följande fel kan uppstå:

1. Variabelnamnet angivet i variabelbindningslistan existerar inte i agentens MIB.
2. Agenten svarar med en GetResponse-PDU med fältet "Fel-kod" satt till *noSuchName*. Värdet i fältet "Fel-index" identifierar variabelnamnet i variabelbindningslistan.
3. Agenten kan leverera värdet för variabeln angiven i variabelbindningslistan, men storleken på GetResponse-PDU:en överstiger maxvärdet för PDU:er. Fältet ""Fel-kod"" i GetResponse sätts till *tooBig*.
4. Agenten kan inte leverera värdet för en variabel angiven i variabelbindningslistan. Fältet "Fel-kod" i GetResponse sätts till *genErr* och värdet i fältet ""Fel-index"" identifierar variabelnamnet i variabelbindningslistan.

Fältet variabelbindning är en sekvens av namnvärdepar med variabler. Storleken på listan varierar från 0 till *max_bindings*, som är ett heltalsvärde med storlek 2147483647.

Namn	Värde	…	Namn	Värde

Figur 6 Variabelbindningar

4.8.2 GetNextRequest-PDU

GetNextRequest har samma format som GetRequest men en annan semantik.

Nedan visas strukturen för en GetNextRequest-PDU.

PDU-typ	Id	Fel-kod	Fel-index	Variabelbindningar

Figur 7 GetNextRequest-PDU

GetNextRequest möjliggör att samtliga noder i en MIB kan besökas i lexikografisk ordning.

Med det första anropet till en agent anger managern vid vilken nod i MIB:en som sökningen skall börja och när en manager tar emot ett svar från en agent på ett GetNextRequest sänder den en ny GetNextRequest-PDU. Detta fortgår till dess agenten skickar ett svar innehållande ett felmeddelande, vilket innebär att samtliga barn till den ursprungligen angivna noden har besökts. Anledningen till detta förfarande är att en agent inte kan returnera en hel tabell med värden utan endast en skalär åt gången, men med hjälp av ett till flera GetNextRe-

quest meddelanden kan en hel tabell eller ett subträd returneras till managern. Liksom GetRequest är GetNextRequest atomär, och har samma felhantering som GetRequest.

4.8.3 GetBulkRequest PDU

Med SNMPv2c introducerades en ny operation, GetBulkRequest, som används då en större mängd data skall hämtas. I SNMPv1 kan endast en begränsad mängd data utbytas mellan manager och agent per operation, vilket ofta leder till att flera operationer måste utföras för en dataöverföring. Detta leder i sin tur till att nätverket blir tungt belastat. Om t.ex. information hämtas från en tabell kommer, då SNMPv1 används, ett Get/Response-kommando utföras för varje rad i tabellen. Men med GetBulkRequest går det att hämta in en hel tabell på en enda överföring, och även hämta in ytterligare information från andra variabler.

Nedan visas strukturen för en GetBulkRequest-PDU.

PDU-typ	Id	Non-repeaters	Max-repetitions	Variabelbindningar

Figur 8 GetBulkRequest-PDU

GetBulkRequest har två extra fält till skillnad mot GetRequest och GetNextRequest, vilka är "Non-repeaters" och "Max-repetitions". "Non-repeaters" anger hur många av de variabler man vill ha information om som kommer att returnera endast ett värde, t.ex. sysUpTime. Om värdet är satt till mindre än noll så sätts värdet automatiskt till noll. "Max-repetitions" anger hur många rader tabellen man vill ha information om innehåller. Sätts ett värde som är mindre än noll kommer värdet noll att sättas. Variabelbindingslistan i ett GetBulkRequest kan innehålla fler än ett variabelnamn.

4.8.4 SetRequest-PDU

SetRequest-PDU används av en manager då denna vill konfigurera ett nätverkselement, d.v.s. sätta ett nytt värde på en variabel i en agent eller lägga till eller ta bort en tabell. Strukturen på en SetRequest-PDU är den samma som för GetRequest och GetNextRequest.

Nedan visas strukturen för en SetRequest-PDU.

PDU-typ	Id	Fel-kod	Fel-index	Variabelbindningar

Figur 9 SetRequest-PDU

En SetRequest-PDU är också den atomär och har samma felhantering som GetRequest och GetNextRequest, med det undantaget att den kan returnera felkoden *badValue* vilken indikerar inkonsistens mellan en variabel och det önskade värdet. Inkonsistensen kan bero på felaktig typ, eller längd hos det nya värdet, eller om det nya värdet är det samma som det aktuella värdet.

4.8.5 GetResponse-PDU och felmeddelanden

En agent skickar som tidigare nämnt en GetResponse-PDU som svar på *get*, *get-next* och *get-bulk* kommandon.

Nedan visas strukturen för en GetResponse-PDU.

PDU-typ	Id	Fel-kod	Fel-index	Variabelbindningar

Figur 10 GetResponse-PDU

Fälten "PDU-typ" och "Id" används på samma sätt som för de övriga PDU:erna, och fältet "Variabelbindningar" innehåller namnet och värdet på den variabel som efterfrågades av kommandot *get*. När agenten svarar på ett kommando av typen *get-bulk* så kan fältet "Variabelbindningar" innehålla fler än ett variabelvärde. Då ett fel har uppstått används fältet "Fel-index" för att identifiera vilken variabel det var i förfrågan som orsakade felmeddelandet.

Ett GetResponse-PDU innehåller fältet "Fel-kod" vilket är reserverat för att innehålla ett fördefinierat felmeddelande. De flesta av dessa felmeddelanden är svar på felaktiga SetRequest- PDU:er, d.v.s. skrivoperationer.

SNMPv1 definierar fem felmeddelande, vilka listas nedan. Värdet inom parentes motsvarar felmeddelandets heltalsvärde då det förekommer i ett GetResponse-PDU.

- noError(0) – Kommandot gick att utföra utan problem.
- tooBig(1) – Svaret på ett kommando är för stort att skicka med ett PDU.
- noSuchName(2) – Den efterfrågade OID:en finns inte representerad i agentens MIB.
- badValue(3) – En variabel sätts till ett inkonsistent värde.
- readOnly(4) – Används vanligtvis inte.
- genErr(5) – Övriga felmeddelanden som inte kan kategoriseras enligt ovan.

Felmeddelanden som definierades i SNMPv1 är grovhuggna och lämpar sig inte för SNMPv2 och SNMPv3, varför SNMPv2 definierar ytterligare 13 felmeddelanden, vilka listas nedan;

- noAccess(6) - En skrivoperation misslyckades p.g.a. fel rättigheter.
- wrongType(7) - Ett försök att ändra typen för en variabel som strider mot dess definition.
- wrongLength.(8) - En skrivoperation misslyckades då det nya värdet var längre än max tillåten längd.
- wrongEncoding(9) - En skrivoperation misslyckades då variabelns nya värde kodats fel.
- wrongValue(10) - En skrivoperation misslyckades p.g.a. det nya variabelvärdet är felaktigt. Uppstår t.ex. när en variabel är definierad som en enumeration och skrivoperationen försöker skriva ett värde som inte är av typen enumeration.
- noCreation(11) - Försök att skriva till en icke existerande variabel eller vid försök att skapa en variabel som inte finns definierad i agentens MIB.
- inconsistentValue(12) - En variabel är i ett inkonsistent tillstånd och tillåter inga värdeförändringar.
- resourceUnavailable(13) - Inga systemresurser finns tillgängliga för vilka det är möjligt att utföra en skriv operation.
- commitFailed(14) - Övriga fel för skrivoperationer vilka faller utanför här beskrivna felmeddelanden.
- undoFailed(15) - En skrivoperation misslyckades och agenten kan inte återskapa redan skrivna variabelvärden fram till misslyckandet.
- authorizationError(16) - Ett felaktigt lösenord användes av ett SNMP-meddelande.
- notWritable(17) - En variabel accepterar inte att dess värde förändras trots att den skall acceptera en skrivoperation.
- inconsistentName(18) - En skrivoperation misslyckades p.g.a. interna fel.

4.8.6 Trap- och notification-PDU

Trap-PDU:er genereras av agenter, och är till för att meddela en manager om signifikanta förändringar av variabelvärden. Trap är asynkrona till sin natur, och en agent förväntar sig inte att en manager bekräftar ett mottaget trap.

Strukturen för en Trap-PDU visas nedan.

PDU-typ	*Enterprice*	Agentadress	Generellt trap	Specifikt trap	Tidstämpel	"Variabelbindningar"

Figur 11 Trap- PDU

Ett trap skiljer sig strukturellt från de andra PDU:erna. Nedan listas de fält som ingår i ett PDU-trap.

Fält	**Beskrivning**
PDU-typ	Visar vilken PDU-typ, i det här fallet trap.
Enterprice(företag)	Identifierar vilket objekt som orsakade ett trap.
Agentadress	IP adressen till den agent som orsakade trap.
Generic trap	Fördefinierade värden för ett trap.
Specifik trap	Specifika trap-koder.
Tidstämpel	Tiden som förflutit mellan agentens initierande och tidpunkten för genereringen av ett *trap*.
Variabelbindning	Namn på variabeln och dess värde som managern vill avläsa.

Tabell 4 Beskrivning av fälten i en PDU för ett trap.

Ett trap kan skicka ett av sex fördefinierade värden, vilka listas nedan, men det är också möjligt för en leverantör av ett nätverkselement att definiera egna värden genom att använda fältet "specifikt trap".

- warmStart(1) - Agenten initieras igen, utan att agentens konfiguration ändras.
- linkDown(2) - Indikerar att ett nätverkskort i ett nätverkselement har slutat fungera. Fältet "Variabelbindningar" innehåller namnet på nätverkskortet.
- linkUp(3) - Indikerar att en agent börjar fungera igen och sätter felkod till linkUp och första fältet i "Variabelbindningar" innehåller namnet på nätverkskortet.

- authenticationFailure(4) - Sätts då ett SNMP-meddelande mottagits med felaktig autenticieringsinformation.
- egpNeighborLoss(5) – Används i de fall då nätverksenheten utbyter routing-tabeller med andra nätverksenheter och fel i kommunikationen uppstår mellan nätverksenheterna.
- enterpriceSpecific(6) - Identifierar ett leverantörsspecifikt trap-meddelande.

SNMPv2 definierar s.k. *notification*-PDU:er vilka till skillnad från ett SNMPv1-trap har samma struktur som *get*- och *set*-kommandona. I övrigt gäller samma regler för vilka typer av meddelande en *notification* kan skicka som för ett trap.

4.9 Fördelar och nackdelar med SNMP

SNMP är idag en defacto-standard och de flesta leverantörer av nätverkselement tillhandahåller SNMP-agenter för sina produkter. Detta till trots har SNMP både för- och nackdelar, vilka de mest framträdande listas nedan.

Fördelar med SNMP:

- Enkelt protokoll, vilket leder till att det är enkelt att implementera egna managers.
- Allmänt använt, defacto-standard
- Oberoende av operativsystem och programspråk.
- Förekomsten av en agent på ett nätverkselement påverkar inte nämnvärt dess belastning och prestanda.

Nackdelar med SNMP:

- Lämpar sig inte för övervakning av riktigt stora nätverk.
- Inte anpassat för att ta emot stora mängder data. Förfrågan efter en stor datamängd måste delas upp i flera förfrågningar.
- Trap i SNMP bekräftas inte av managern varför en agent inte vet om meddelandet mottagits eller ej. Detta kan medföra att kritiska meddelanden aldrig når managern.
- MIB-modellen i SNMP är begränsad och tillåter inte att ett nätverkselement gör sofistikerade förändringar.

- Låg säkerhet vilket beror på att standarden endast definierar enkel autentisiering. Detta medför att SNMP inte lämpar sig för konfiguration av nätverkselement utan lämpar sig bäst för övervakning.

4.10 Sammanfattning

SNMP utvecklades under senare delen av 80-talet och skulle från början endast användas i en övergångsperiod till dess ett mer kompetent protokoll för övervakning utvecklats. Övergångsperioden blev i det närmaste permanent, vilket medförde att produktutvecklare implementerade SNMP till sina produkter. Tack vare detta fick protokollet stor spridning och det räknas idag som defacto-standard.

Ett SNMP-övervakat nätverk består av minst en manager och en eller flera agenter. Managern övervakar aktivitet hos en agent genom att läsa variabelvärden lokalt lagrade hos agenten. Administration av en agent sker då en manager förändrar värdet av en variabel hos en agent.

Agenternas variabler är hierarkiskt lagrade i en databas (MIB) vilken är strukturerad som ett träd. För att en manager skall kunna hantera agenter på ett enhetligt sätt så är varje MIB strukturerad efter ett fördefinierat regelverk (SMI). SMI definierar följande:

- Strukturen av en MIB.
- Syntax för variabler.
- Kodning av variabelvärden.

5 Design av övervakningsservern

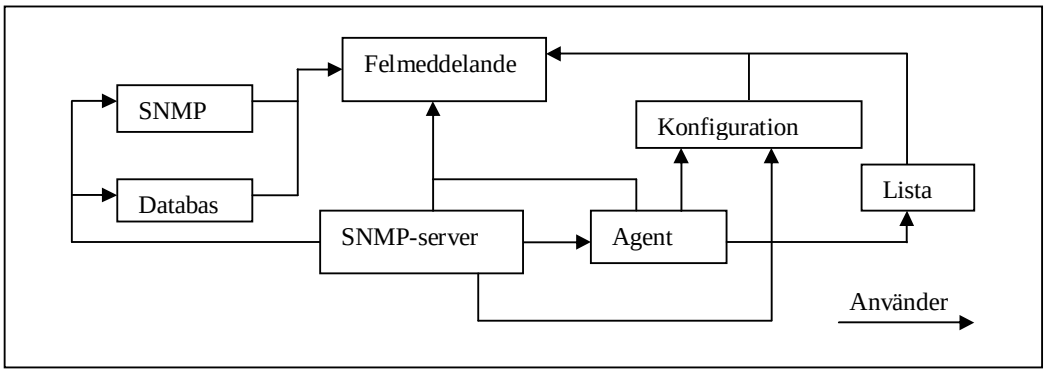
5.1 Inledning

Den SNMP-baserade övervakningsapplikationen har till uppgift att samla in variabelvärden från nätverkselement, vilket den gör genom att skicka GetRequest- och GetNextRequest-PDU:er till nätverkselementen. I nätverkselementet tar agentapplikationen hand om förfrågan och sammanställer ett svar, vilket skickas tillbaka till övervakningsapplikationen. När svaret når övervakningsapplikationen behandlas svaret med avseende på format i de fall det behövs varefter svaret lagras i en databas.

En förutsättning för att övervakningsapplikationen skall kunna utföra den ovan beskrivna uppgiften är att den har en förteckning över de nätverkselement, eller agenter, till vilka den skall sända SNMP-frågor. Övervakningsapplikationens huvuduppgifter kan sammanfattas i tre punkter;

- Hantera en förteckning över de agenter som skall ingå i övervakningen.
- Skicka och ta emot SNMP-meddelanden till och från agenter.
- Lagra erhållna variabelvärden i en databas.

Övervakningsapplikationen är uppdelad i sju moduler, vilka utnyttjar varandras tjänster, se figur 12. Modulen SNMP-server är övervakningsapplikationens huvudmodul, d.v.s. den modul vilken startar applikationen.



Figur 12 Övervakningsserverns moduler.

Övervakningsapplikationens tre huvuduppgifter utförs av modulerna Konfiguration, Agent, SNMP och Databas. Modulerna Konfiguration och Agent hanterar tillsammans listan över agenter och de attribut som tillhör en agent. Modulen Konfiguration hanterar utöver agenter även den information som är konfigurerbar vid kommunikationen med en databas, d.v.s. databasserverns IP-adress, databasens namn, och login-information. Modulen SNMP konstruerar, skickar och tar emot SNMP-meddelanden. Kommunikationen med databasen hanteras av modulen Databas. Alla moduler inklusive SNMP-servern använder en gemensam modul för hantering av felmeddelanden, benämnd Felmedddelande i figur 12, vilken har till uppgift att internt lagra felmeddelanden och att visa de lagrade felmeddelandena för en användare. Den sista modulen Lista tillhandahåller en generell lista, vilken används av de övriga modulerna då dessa behöver hantera en sekvens av element. T.ex. använder modulerna Agent och Konfiguration listan för att lagra uppgifter om agenterna som skall övervakas.

Samtliga moduler har ett väldefinierat grässnitt vilket gör det möjligt att byta ut en modul mot en ny implementation av modulen under förutsättning att gränssnittet behålls intakt, både syntaktiskt och semantiskt.

Modulerna Konfiguration, SNMP och Databas kan implementeras på ett av två sätt; antingen implementerar den själv tjänsten från grunden eller så fungerar den som en *wrapper* kring ett underliggande API vilket utför tjänsten.

5.2 Insamling av SNMP-meddelanden

Övervakningsapplikationen övervakar agenter i ett nätverk genom pollning, vilket innebär att den ställer frågor till agenter varefter den väntar en förutbestämd tid innan den upprepar frågan. Om inget annat anges pollas agenterna var 15:e minut. Agenterna som skall pollas indelas i fyra enhetstyper (eng. device type): server, klient, printer och switch. Indelningen är baserad på vilka OID:er eller variabelvärden som efterfrågas hos en agent. Varje OID utgör en fördefinierad SNMP-fråga vilken skickas till agenten. En agent kan inte tilldelas mer än en enhetstyp åt gången. Pollningen schemaläggs i modulen SNMP-server vilken ställer fördefinierade SNMP-frågor till samtliga agenter. I figur 13 visas principen för hur pollningen av agenterna går till.

```
InitAgentList();
FirstAgent();
while(NextAgent() != NULL){
        Hämta agentens attribut.
        För varje OID för agentens enhetstyp
        {
                Ställ SNMP fråga
                Formatera svaret
                Spara i Databasen
        }
}
sleep 15 minuter
```

Figur 13 Principiell beskrivning av pollning av agenter.

5.2.1 Modulen SNMP

Detaljerna hur SNMP-meddelanden konstrueras och förmedlas mellan övervakningsservern och agenter hanteras av modulen SNMP. I tabell 5 nedan visas det gränssnitt vilket implementeras av modulen snmp.

Funktionsnamn	**Beskrivning/Semantik**
initSnmp	Initierar kommunikationen mellan SNMP-servern och en agent. Argument: Agentens IP-adress. Agentens SNMP-version. Agentens Lösenord. Loginnam för användaren (Används endast i SNMPv3). Användarens lösenord (Används endast i SNMPv3). Returvärde: TRUE om anropet lyckades annars FALSE.
getSnmp	Skickar en GetRequest-PDU till en agent. Anrop till getSnmp skall föregås av ett anrop till initSnmp. Argument: OID, dvs. namnet på en variabel i agenten. Returvärde: Då anropet lyckades returneras en pekare till

	en sträng vars innehåll är värdet av efterfrågad variabel, annars returneras NULL.
getNextSnmp	Skickar en GetNext-PDU till en agent. Med hjälp av getNextSnmp kan en hel tabell av skalärer hämtas från en agent. Den första skalären som skall hämtas anges som argument till getNextSnmp, varefter anrop med argument satt till NULL returnerar nästa skalär i tabellen till dess det inte finns några skalärer kvar att returnera. Anrop till getNextSnmp skall föregås av ett anrop till initSnmp. Argument: En OID för den fösta skalären i en tabell, eller NULL om nästa skalär i en tabell skall returneras. Returvärde: Då anropet lyckades returneras en pekare till en sträng vars innehåll är värdet av den efterfrågade variabeln, annars returneras NULL.
getBulkSnmp	Skickar en GetBulk-PDU till en agent. getBulkSnmp returnerar samma resultat som getNextSnmp, men med den skillnaden att en klient kan anropa getBulkSnmp en gång istället för att upprepade gånger anropa getNextSnmp. Anrop till geBulkSnmp skall föregås av ett anrop till initSnmp. Argument: En OID för den fösta skalären i en tabell. Returvärde: TRUE om anropet lyckades annars FALSE.
getBulkResult	Returnerar de variabelvärden som en agent skickade som svar på ett getBultSnmp anrop. Variabelvärdena returneras ett och ett varför en klient som skall hantera ett svar från en agent behöver anropa get-BulkResult upprepade gånger till dess NULL returneras, vilket betyder att samtliga variabelvärden från en agent har returnerats. Argument: Inga. Returvärde:

	En pekare till en sträng vars innehåll är värdet av en variabel. Då det inte finns några fler variabelvärden att returnera så returneras NULL.
closeSnmp	Avslutar en SNMP-session med en agent. Argument: Inga. Returvärde: Inget.

Tabell 5 Gränssnitt för modulen SNMP

5.2.2 Modulen Databas

Modulen Databas hanterar detaljerna hur övervakningsservern kommunicerar med databasen. Denna definierar ett gränssnitt för att hantera upp- och nedkoppling av databasen samt hantering av frågor och svar till och från databasen. Kravet på underliggande databas är att det är en relationsdatabas, liknande MySQL.

I tabell 6 nedan visas gränssnittet för modulen Databas.

Funktionsnamn	**Beskrivning/Semantik**
OpenDatabase	Öppnar en session mot en databasmanager. Argument: IP-adress till databasserver. Namn på databasen. Användarnamn. Lösenord. Returvärden: TRUE om anropet lyckades annars FALSE.
QueryDatabase	Ställ en SQL-fråga till databasen. Argument: Sträng vilken anger SQL-frågan. Returvärde: TRUE om anropet lyckades annars FALSE.
QueryDatabaseFmt	Ställer en SQL-fråga till databasen, men använder formatering för att konstruera frågan. Första argumentet till metoden är en teckensträng

	vilken innehåller formateringstecken. Formateringstecknet %d tolkas som heltal, och %s tolkas som en sträng. Argument: Formateringssträng. Övriga argument är antingen teckensträngar eller heltal. Returvärde: TRUE om anropet lyckades annars FALSE.
PrintQueryResult	Skriver ut resultatet av en den senast ställda SQL-frågan till ett terminalfönster. Används enbart vid felsökning. Argument: Inga. Returvärde: Inget.
GetResult	Returnerar erhållet svar på den senast ställda SQL-frågan. I de fall en SQL-fråga ger en tabell tillbaka kan hela tabellen returneras genom att upprepade gånger anropa GetResult till dess GetResult returnerar NULL. När GetResult returnerar NULL betyder det att samtliga värden i ett svar har resturnerats. Argument: Inga. Returvärde: Då det finns svar att returnera, så returneras en pekare till en sträng vilken innehåller svaret i en kolumn. Då inga fler svar finns att returnera, returneras NULL.
CloseDatabase	Avslutar en databassession. Argument: Inga. Returvärde: TRUE om anropet lyckades annars FALSE.

Tabell 6 Gränssnitt för modulen databas.

5.2.3 Modulen Konfiguration

Modulen Konfiguration har till uppgift att läsa en konfigurationsfil vilken innehåller uppgifter om vilka agenter som skall övervakas, samt de uppgifter som behövs för att övervakningsservern skall kunna kommunicera med en databas. Dessa uppgifter lagras i en gemensam konfigurationsfil som kan editeras vid behov. Konfigurationsmodulens gränssnitt är uppdelat i två olika delar, en för hantering av agenterna och en för hantering av informationen om databasen. Gemensamt för båda är dock att en klient till konfigrationsmodulen initierar modulen med ett anrop till *InitConfigLib*. Gränssnittet för *InitConfigLib* visas i tabell 7

Funktionsnamn	**Beskrivning/Semantik**
InitConfigLib	Initierar konfigurationsmodulen. Öppnar och läser konfigurationsfilen angiven som argument i ett anrop. Efter ett anrop till InitConfigLib har en lista över agenter skapats, vilken returneras vid anrop till GetAgents. Databas konfigurationen kan hämtas genom anrop till funktioner i modulen konfiguration, se nästa avsnitt. Argument: Sökväg till konfigurationsfil. Returvärde: TRUE om anropet lyckades annars FALSE.

Tabell 7 Gränssnittet för funktionen InitConfigLib i modulen Konfiguration.

5.2.3.1 Databaskonfiguration

Modulen Databas förväntar sig att SNMP-servern skickar med uppgifter om databasserverns IP-adress, databasens namn och login-information då en session mellan övervakningsserven och databasmanagern upprättas. En administratör skall kunna ändra dessa uppgifter på ett enkelt sätt, varför de lagras i en konfigurationsfil. Den läses in av modulen Konfiguration vid start av övervakningsservern. I figur 14 visas hur informationen om databasen lagras i konfigurationsfilen. Eftersom både konfigurationen för databasen och agenterna ligger i samma fil, används etiketter för att skilja konfigurationerna åt. En databaskonfiguration är omgiven av etiketterna <Database> och </Database>. Tecknet '#' i konfigurationsfilen anger början på en kommentar.

```
<Database>
   <Name> name </Name>
   <User> testuser </User>
   <Server> 120.12.21.21 </Server>
   <Password> secret </Password>
</Database>
```

Figur 14 Exempel databaskonfiguration

För att konfigurationen skall vara konsekvent anges även databasens övriga konfiguration mellan etiketter. I figur 14 visas konfigurationen för en databasserver med IP-adress 120.12.21.21 och databasnamnet *name*, samt anvädarnamnet *testuser* och lösenordet *secret*.

I tabell 8 visas den del av gränssnittet för modulen Konfiguration som en klient kan använda för att läsa konfigurationen för databasen.

Funktionsnamn	**Beskrivning/Semantik**
GetDBServer	Hämtar databasserverns IP-adress. Argument: Inget. Returvärde: En sträng innehållande IP-adressen till databasservern då anropet lyckades, annars returneras NULL.
GetDBDatabaseName	Hämtar namnet på databasen. Argument: Inget. Returvärde: En sträng innehållande databasens namn då anropet lyckades, annars returneras NULL.
GetDBUserName	Hämtar användarnamnet för databasen. Argument: Inget. Returvärde: En sträng innehållande användarnamnet då anropet lyckades, annars returneras NULL.
GetDBPassword	Hämtar lösenordet för databasen.

	Argument: Inget. Returvärde: En sträng innehållande lösenordet då anropet lyckades, annars returneras NULL.

Tabell 8 Gränssnittet för databaskonfiguration i modulen Konfiguration.

5.2.3.2 Agentkonfiguration

Tillsammans med konfigurationen för databasen lagras informationen om vilka agenter som övervakningsservern skall övervaka. Vid anrop av funktionen *InitConfigLib* konstrueras en lista över de agenter som är definierade i konfigurationsfilen. Modulen Konfiguration tillhandahåller en funktion GetAgents vilken returnerar listan över agenterna. I tabell 9 visas Konfigurationsmodulen gränssnittet för GetAgents.

Funktionsnamn	**Beskrivning/Semantik**
GetAgents	Returnerar en lista över de agenter som listats i konfigurationen. Argument: Inga. Returvärde: Returnerar en lista med agenter då anropet lyckade, annars returneras NULL:

Tabell 9 Gränssnittet för funktionen GetAgents i modulen Konfiguration.

När övervakningsservern skall anropa en agent behöver den uppgifter om vilken IP-adress agenten har, vilken version av SNMP som agenten kan hantera, och SNMP-lösenordet för agenten. Notera att SNMPv3 inte är ett alternativ för en agent, vilket beror på att de flesta nätverksenheter inte stöder SNMPv3 utan använder SNMPv1 eller SNMPv2. (Windows 2000 använder SNMPv2c och HP LaserJet använder SNMPv1).

Uppgiften om agenterna och deras attribut lagras i samma konfigurationsfil som databas-konfigurationen, och en agent konfigureras på liknande sätt som databasen, d.v.s. med etiketter. I figur 15 visas hur en agent definieras i konfigurationsfilen. Tecknet '#' i konfigurationsfilen anger början på en kommentar.

```
<Agent>
  <IP> 10.151.1.243 </IP>
 <Hostname> WindowsClient1.domain.se </Hostname>
 <DeviceType> Client </DeviceType>
 <SNMPVersion> v1 </SNMPVersion>
 <SNMPCommnity> public </SNMPCommunity>
</Agent>
```

Figur 15 Exempel på konfiguration av en agent.

För varje agent skall fem attribut konfigureras:

- IP-adress - Anges mellan etiketterna <IP> och </IP>.
- Hostnamn - Anges mellan etiketterna <Hostname> och </Hostname>.
- Enhetstyp – Anges mellan etiketterna <DeviceType> och </DeviceType>.
- SNMP-version – Anges mellan etiketterna <SNMPVersion> och </SNMPVersion>.
- SNMP-lösenord – Anges mellan etiketterna </SNMPCommunity> och </SNMPCommunity>.

De två första attributen används för att identifiera vilken agent som övervakningsservern skall polla. En agent tillhör en av fyra fördefinierade enhetstyper; klient, server, skrivare och switch. Dessa enhetstyper definierar vilka OID:er, eller variabelvärden, som övervakningsservern skall avläsa hos en agent. En enhetstyp i konfigurationsfilen bestäms av strängarna; Client, Server, Printer eller Switch.

Attributen SNMP-version och SNMP-lösenord definierar vilken version av SNMP som agenten använder och det lösenord som skall användas vid läsning av variabelvärden i agenten. SNMP-versionen anges med en av följande textsträng: v1 eller v2.

En agents SNMP-attribut används av SNMP-modulen när denna konstruerar SNMP-frågorna som skickas till agenten.

5.3 Modulen Agent

Modulen Agent tillhandahåller en lista över agenterna som övervakas och funktioner för åtkomst av deras attribut. Listan skapas via anrop till funktionen *InitAgentList*. Med hjälp av funktionerna *FirstAgent* och *NextAgent* kan elementen i listan stegas genom. I tabell 10 nedan visas agentmodulens gränssnitt. Samtliga access-funktioner börjar med ordet Get.

Funktionsnamn	**Beskrivning/Semantik**
InitAgentList	Initierar listan över agenterna i agentmodulen. Argument: Inga. Returvärde: TRUE om anropet lyckades annars FALSE.
NextAgent	Initierar samtliga Get-operationer i agentmodulen att arbeta mot nästa element i listan över agenter. Argument: Inga. Returvärde: TRUE om anropet lyckades annars FALSE.
FirstAgent	Initierar samtliga Get-operationer i agentmodulen att arbeta mot det första elementet i listan över agenter. Argument: Inga. Returvärde: TRUE om anropet lyckades annars FALSE.
GetAgentIP	Hämtar IP-adressen för en agent. Argument: Inga. Returvärde: En sträng innehållande agentens IP-adressen då anropet lyckades, annars returneras NULL.
GetAgentNodeName	Hämtar nodnamnet för en agent. Argument: Inga. Returvärde: En sträng innehållande agentens nodnamn då anropet lyckades, annars returneras NULL.
GetAgentDeviceType	Hämtar enhetstypen för en agent. Argument: Inga.

	Returvärde: En sträng innehållande agentens enhetstyp då anropet lyckades, annars returneras NULL.
GetAgentSNMPVersion	Hämtar agentens SNMP-version. Argument: Inga. Returvärde: En sträng innehållande agentens SNMP version då anropet lyckades, annars returneras NULL.
GetAgentSNMPCommunity	Hämtar SNMP-lösenordet för en agent. Argument: Inga. Returvärde: En sträng innehållande agentens SNMP lösenord Då anropet lyckades, annars returneras NULL.

Tabell 10 Gränssnitt för modulen Agent.

5.4 Modulen Lista

Modulen Lista tillhandahåller en lista vilken kan användas av de övriga modulerna. Listan är generell, vilket innebär att den kan innehålla element av godtycklig typ. I tabell 11 visas gränssnittet för modulen Lista.

Funktionsnamn	**Beskrivning/Semantik**
NewList	Skapar en ny lista. Argument: Inga. Returvärde: Då anropet lyckades returneras en referens till den nya listan, annars returneras NULL.
Add	Lägg till ett nytt element till en lista. Argument: Referens till listan vilket elementet skall ingå .

	Referens till det nya elementet. Returvärde: Då anropet lyckades returneras en referens till en lista med det nya elementet, annars returneras NULL.
Delete	Raderar ett element ur listan baserat på elementets värde. Om ett element påträffas som har samma värde som argument två i anropet till Delete raderas detta och en referens till en ny lista utan det gamla elementet returneras. Argument: En referens till listan från vilket ett element skall raderas. Returvärde: En referens till den nya listan.
DeleteAll	Raderar samtliga element i en lista. Argument: Referens till listan. Returvärde: Då anropet lyckades returneras NULL, annars returneras en referens till listan vilken anges som argument.
Get	Returnerar en referens till det första element i listan som har samma värde som skickas med som argument nummer två i anropet till get. Argument: Referens till en lista. Ett datavärde vilket representerar värde hos det element som skall raderas. Returvärde: Då anropet lyckades returneras en referens till det sökta elementet. I annat fall returneras NULL.
GetNext	Returnerar en referens till nästa element i en lista. En referens till det element som är före det önskade elementet anges som argument till GetNext. Argument: Referens till ett element i listan. Returvärde:

	Referens till det element i listan som följer elementet angivet som argument. Om det sista elementet i en lista anges som argument returneras NULL.
isLast	Predikat för att avgöra om ett listelement är det sista i en lista. Argument: Referens till ett element i en lista. Returvärde: TRUE om anropet lyckades annars FALSE.

Tabell 11 Gränssnittet för modulen Lista.

5.5 Modulen Felmeddelanden

Samtliga moduler använder samma modul för hantering av felmeddelanden. Med hantering av felmeddelande menas att lagra och rapportera eventuella felmeddelanden. Ett felmeddelande lagras internt i felmeddelandemodulen till dess en rapportering begärs.

En rapportering sker genom att samtliga lagrade felmeddelanden skrivs ut till ett terminalfönster. Modulen för felmeddelande definierar ett grässnitt med två metoder, Error och ReportError. Då metoden Error anropas lagras ett felmeddelande, vilket anges som parameter till metoden Error. ReportError-funktionen skriver ut samtliga lagrade felmeddelanden samtidigt som de lagrade felmeddelanden i modulen Felmeddelande raderas. Eftersom samtliga moduler och även modulen SNMP-server använder modulen Felmeddelande så kan returvärdet från funktionsanrop som SNMP-server gör till de övriga modulerna kontrolleras och vid eventuella fel anropar modulen SNMP-server funktionen ReportError. ReportError presenterar samtliga felmeddelanden i samma ordning som de rapporterades, vilket medför att det första rapporterade och troligtvis mest relevanta felmeddelandet listas först. Eventuella följdfel presenteras sedan i fallande ordning.

I tabell 12 nedan beskrivs gränssnittet för modulen Felmeddelande.

Funktionsnamn	**Beskrivning**
Error	Lagrar ett felmeddelande internt till dess det skrivs ut av ReportError. Argument: Textsträng innehållande ett felmeddelande. Returvärde: Inget.

ReportError	Skriver ut samtliga lagrade felmeddelanden till skärm varefter samtliga lagrade felmeddelanden raderas. Argument: Inga. Returvärde: Inget.

Tabell 12 Gränssnittet för modulen Felmeddelande.

5.6 Konfiguration av övervakningsserverns moduler

En administratör eller utvecklare kan konfigurera övervakningsservern med avseende på vilka implementationer för modulerna SNMP, Databas och Konfiguration som skall användas av övervakningsservern. Detta görs i en separat konfigurationsfil, vilken läses in av övervakningsservern innan den använder någon av de övriga modulerna. Modulerna skall kunna bytas ut efter behag, varför de antas vara implementerade som mjukvarumoduler (DLL-bibliotek i Windows eller dynamiskt länkade bibliotek i UNIX). Modulerna benämns med mjukvarumodulens filnamn, som i UNIX skall vara en absolut sökväg. Mjukvarumodulens filnamn skrivs mellan två etiketter. I figur 16 nedan listas samtliga definierade etiketter.

```
<ConfigLib>    Namnet på konfigurationsmodulen </ConfigLib>
<DataBaseLib>  Namnet på databasmodulen        </DataBasLib>
<SNMPLib>      Namnet på SNMPmodulen           </SNMPLib>
```

Figur 16 De definierade etiketterna för konfiguration av övervakningsserverns moduler.

Den individuella ordningen mellan modulernas definition är inte signifikant. Tecknet '#' anger början på en kommentar och från det att tecknet påträffas till slutet av raden.

Notera att konfigurationsfilen **måste** finnas och att sökvägarna eller namnen på modulerna måste vara valida annars fungerar inte övervakningsservrarna som den ska.

5.7 Databasen

De avlästa variabelvärdena lagras i en relationsdatabas, varpå en applikation kan läsa och presentera de avlästa variabelvärdena för en användare. Relationsdatabasen har designats av Ataco, och en beskrivning av databasens tabeller och fält ges i Appendix C.

6 Utvecklingsmiljö

6.1 Utvecklingsverktyg och maskinpark

Övervakningsservern utvecklades i programspråket C på ett UNIX-system. Utöver UNIX-systemets C-kompilator användes verktygen Lex och Yacc för konstruktion av parsrar och lexikaliska analysatorer, vilka används för hantering av övervakningsserverns konfigurationsfiler. Maskinparken, vilken användes både vid utveckling och testning av övervakningsservern, tillhandahölls av Ataco och bestod av två PC-datorer och en dator med en installation av MySQL. Operativsystemen på PC-datorerna var Windows 2000 och Debian Linux. Linux-distributionen ersattes till förmån för FreeBSD 5.0, då brister i Linux-systemet påträffades i inledningsskedet av projektet. Under testningen av övervakningsservern användes en evalueringsversion av programvaran VMware Workstation 4, se Terminologi.

6.2 UCD-SNMP

Tjänstemodulen SNMP är implementerad som en wrapper kring ett öppet programbibliotek för SNMP, vilket implementerar ett API skrivet i programspråket C. Programbiblioteket ingår i en SNMP-distribution som heter UCD-SNMP version 4.2.6, där användare har tillgång till källkoden för hela distributionen. Distributionen innehåller förutom programbiblioteket enkla program med vilka det är möjligt att läsa och skriva variabelvärden hos en agent. En grafisk MIB-navigator ingår också i distributionen samt en implementation av en agent. UCD-SNMP är från början baserad på en implementation av SNMP, skapad vid Carnegie Mellon University och heter CMU 2.1.2.1. UCD-SNMP har dock omarbetats så mycket att de båda distributionerna skiljer sig avsevärt från varandra.

I november 2000 bytte UCD-SNMP namn till NET-SNMP, i samband med en omstrukturering av utvecklingsprojektet, men mycket lite av designen skiljer mellan UCD-SNMP och NET-SNMP. Med början på version 5.0 av NET-SNMP planeras en omstrukturering av källkoden för att göra den mer lättläst och därmed lättare att utveckla och underhålla.

Distributionen för UCD-SNMP är tillgänglig från NET-SNMP projektets hemsida [6], vilken tillhandahåller binärformat av distributionen för både UNIX-baserade system och Windows-system.

6.3 MySQL

Databasen, vilken är designad av Ataco, använder databasservern MySQL version 4.0. Fördelarna med MySQL är flera. Bland annat är distributionen kostnadsfri och användare har tillgång till källkoden för distributionen. Databasservern kan även användas på en mängd olika plattformar. Det finns dock en del begränsningar med MySQL. Saknaden av kontroll på om en programmerare inför inkonsistens i SQL-koden kan vara en stor nackdel. T.ex. om det finns en tabell "Person" och en tabell "Personnummer", så måste det för varje värde i "Person" finnas ett värde i "Personnummer". Men i MySQL finns det ingen kontroll för att det verkligen är det, utan det är fullt möjligt att det finns ett värde i "Person" utan att det finns något värde för motsvarande rad i "Personnummer". Det finns heller ingen support för nästlade SELECT-satser, för att ställa mer invecklade frågor.

7 Implementation av övervakningsservern

7.1 Inledning

I Kapitel 5 beskrivs de sju modulerna vilka tillsammans utgör en implementation av en övervakningsserver. I detta kapitel beskrivs de konstruktionsval och tekniker som använts vid implementation av modulerna. Modulerna är implementerade i programspråket C, och komplett källkodslistning för modulerna finns på den medföljande disketten.

7.2 Modulen SNMP-server

Från början var OID:erna för de olika enhetstyper som övervakas definierade i en separat fil. Men eftersom OID:erna alltid kommer att vara desamma, drogs slutsatsen att det var bäst att deklarera dem statiskt, d.v.s. som globala konstanter, i programkoden. Tack vare det finns det ingen risk att fel OID anges.

I programmet används som tidigare nämnt dynamiska bibliotek. För att kunna använda de funktioner som finns i biblioteken deklareras globala pekare. Detta medför modularitet.

Efter att all initiering av systemet utförts går programmet in i en evighetsloop varifrån agenterna pollas med ett visst tidsintervall.

7.3 Modulerna Konfiguration och Agent

I modulen Konfiguration används verktygen Lex och Yacc. Med dessa verktyg kan en parser genereras med vars hjälp syntaxen i konfigurationsfilerna kan kontrolleras. Syntaxen i konfigurationsfilerna måste vara helt korrekt skriven eftersom ett fel där betraktas som allvarligt. Om konfigurationsfilen innehåller ett fel kommer övervakningsprogrammet av säkerhetsskäl att avslutas.

Etiketterna i konfigurationsfilerna behöver inte skrivas i en specifik ordning, utan modulen är implementerad så att vilken ordning som helst godtas. Detta för att underlätta för användaren så att han/hon inte skriver fel.

Med hjälp av en symboltabell kan modulen garantera att alla IP-adresser för de agenter som övervakas är unika. Om det visar sig att en adress redan finns i tabellen kommer den inte att läggas till.

Modulen Agent har en lista över agenter som övervakas. När den sökta agenten påträffats i listan kan information hämtas från den via de funktioner som ingår i agentmodulen.

7.4 Modulen Databas

De variabelvärden som övervakningsservern läser från de övervakade agenterna lagras i en MySQL-databas. Kommunikationen mellan övervakningsservern och databasen implementeras i modulen Databas. Enligt designen kan denna modul implementeras på ett av två olika sätt, antingen används ett fördefinierat API för ett bibliotek vilket hanterar detaljerna vid kommunikationen med databasservern, eller så implementeras funktioner i modulen med motsvarande funktionalitet. I implementationen av Databas används ett fördefinierat API, vilket innebär att funktionerna i Databas är enkla och förhållandevis korta.

Eftersom svaret på en SQL-fråga kan innehålla fler än ett fält, och databasmodulen bara kan returnera ett fält åt gången, så lagras resultatet internt i databasmodulen.

7.5 Modulen Lista

Gemensamt för flertalet av modulerna i övervakningsservern är att de har behov av att lagra element i en lista. Typen av element varierar dock från modul till modul. För att slippa implementera en lista för varje typ av element tillhandahålls en generell list-implementation av modulen Lista.

Implementationen är uppdelad i två delar, en del som implementerar gränssnittsfunktionerna och en annan del som hanterar detaljerna kring skapandet och hantering av elementen i listan. Uppdelningen av implementationen gör att den interna representationen av listan kan ändras utan att de moduler som använder listor inte skall påverkas vid en förändring.

En målsättning med implementationen av modulen är att den skall vara enkel varför listan internt representeras som en länkad lista. Elementen i listan representeras av pekare till void, vilket medför att en klient kan bestämma typen på elementen i listan.

7.6 Modulen Felmeddelande

Då en modul påträffar ett fel som skall rapporteras används funktionen *Error* i modulen Felmeddelanden. Modulen Felmeddelanden upprätthåller en intern lista med inrapporterade meddelanden. När funktionen *ReportError* anropas skrivs listans innehåll ut på skärm varefter listan töms på sitt innehåll. Listan skriver ut samtliga felmeddelanden i den ordning vilket de rapporterades. Detta får till följd att den troligaste orsaken till ett fel rapporteras först, varefter eventuella följdfel rappoteras. Modulen Felmeddelande lägger inte på någon annan information, t.ex. datum och tid, då felmeddelandet lagrades.

7.7 Inläsning av dynamiska bibliotek

Modulen SNMP-server använder ett gränssnitt för systemfunktionerna *dlopen*, *dlsym* och *dlclos,* när denna läser in de dynamiska biblioteken för modulerna SNMP, Konfiguration och Databas. Systemfunktionerna *dlopen* och *dlsym* används av processer för att öppna och använda funktioner definierade i dynamiska bibliotek.

8 Testning

Övervakningsserverns modulära design har gjort det möjligt att felsöka och provköra modulerna som isolerade enheter. Under utvecklingen av en modul har små testprogram konstruerats för att testa en enskild modul samt dess interaktion med andra moduler. Slutligen testades hela övervakningsserverns funktionalitet genom att polla agenter vilka installerades både på ett UNIX-system och på ett system med Windows 2000. En utvärderingsversion av programvaran VMware Workstation 4 installerades på Windows-maskinen för att emulera ytterligare ett UNIX-system. Det emulerade UNIX-systemet möjliggjorde testning av maskiner med multipla Ethernet- och IP-adresser på ett enkelt sätt, utan att behöva installera ny hårdvara i befintlig utvecklingsmiljö. Sammanlagt testades således tre nätverksenheter tillsammans med övervakningsservern.

Övervakningsserverns interaktion med databasen testades genom att jämföra innehållet i databasen för och efter en pollning mot agenterna med det förväntade resultatet av pollningen. Mellan pollningarna infördes förändringar i agenterna, IP-adresser och MAC-adresser byttes och diskutnyttjandet ändrades genom att ta bort eller lägga till stora filer på ett filsystem.
Hanteringen av konfigurationsfilerna testades genom att införa syntaxfel t.ex.

- Felaktiga etiketter.
- Definitioner utan etikett som markerar slutet av definitionen.
- IP-adresser med fel format.
- Definitioner av agenter med samma IP-adress.

I samtliga fallen hanterade Lex och Yacc parsern tillsammans med symboltabellen samtliga fel vilket resulterade i att övervakningsservern avslutades, vilket den förväntas göra vid fel i konfigurationsfilerna.

Systemet har idag inga kända fel, däremot har inte systemets funktionalitet kunnat testas fullt ut p.g.a. tidsbrist, se kapitel 9.2. Systemet har bara kunnat testats med max tre agenter och pollning av agenter under en längre tidsperiod har inte kunnat utföras inom tidsramen för projektet.

9 Resultat

Det resultat som åstadkommits under projektets gång är en fungerande serverapplikation, skrivet i programspråket C, för övervakning av ett nätverk. Applikationen kan, med hjälp av SNMP, hämta in information från nätverkskomponenter i ett nätverk och lagra denna information i en MySQL-databas. Nätverkskomponenterna kan vara av obegränsat antal, så länge de är kopplade i samma nätverk. Men en begräsning i applikationen är att den inte kan hämta in data från skrivare eller switchar. Detta på grund av att det tog längre tid att färdigställa projektet än beräknat. Via en konfigurationsfil i form av ett textdokument går det att i viss mån konfigurera hur programmet skall exekveras. Skulle programmeraren exempelvis välja att använda en annan databasserver än MySQL för databashanteringen, kan han/hon ange detta i konfigurationsfilen. I ytterligare en konfigurationsfil anges information om databasen samt vilka agenter som används. Koden för hela programmet har dokumenterats väl för att underlätta för andra programmerare i framtiden.

9.1 Kunskaper

Projektet har krävt en intensiv instudering av SNMP, då kunskapen om ämnet innan projektets början var mycket knapp. Som tur var så finns det en uppsjö med information om SNMP på Internet och även en mängd bra litteratur att läsa om detta ämne. Informationssökningen har varit en stor del av projektet eftersom det varit nödvändigt att förstå hur SNMP fungerar för att kunna skapa programmet. En hel del inläsning krävdes också för att veta hur SNMP-biblioteket skall användas. Som exempel finns det flera varianter att programmera en viss funktionalitet i SNMP.

För att skapa funktionaliteten i "error.c" behövdes kunskaper om hur man kan skapa en egen funktion med ett okänt antal inparametrar, i stil med printf-funktionen. Det fanns det bra exempel på i litteraturen.

Användandet av biblioteksfiler i programmet, då någon av tjänstemodulerna skulle behövas bytas ut, har också varit ett delvis nytt område.

9.2 Problem

Implementeringen har tagit upp den största tiden. När man tänker på vad programmets uppgift är, att hämta in data från agenter och lagra den i en databas, så låter det kanske inte så avancerat. Men till detta tillkom en lång rad funktioner för såväl databashanteringen, SNMP-delen och diverse andra funktioner som behövs för att få hela programmet att fungera. Den planerade tiden att genomföra programmet på stämde inte med den faktiska tiden det tog. Detta var dels på grund av att SNMP var ett nytt ämne som tog tid att sätta sig in i, samt att arbetsinsatsen vid projektets start var lägre än den var mot mitten och slutet. Eftersom det tog så lång tid att programmera har det medfört att projektet inte kunde slutföras fullständigt, vilket innebar att inhämtning av data från några agenttyper inte kunde implementeras.

Ett annat problem var att det ursprungliga operativsystemet (Debian) som användes inte kunde hantera *dlopen*-anrop, varför detta byttes mot ett nytt operativsystem, FreeBSD.

Problem uppstod inte bara vid implementeringen utan även vid informationssökning. När det gäller informationssökning har Internet en oändlig mängd sidor innehållande information om SNMP. Problemet är att en hel del av sidorna är skrivna av personer som inte med säkerhet vet om det de skriver är korrekt, vilket medför att uppgifterna på webbsidorna inte alltid överrensstämmer. På så sätt kan det vara svårt att veta vad som är rätt information och vad som är fel. Dock finns RFC:er, som oftast ger en exakt, om än i vissa fall svårläst, definition av vad något betyder.

Ytterligare svårigheter har varit att finna OID:er för de olika nätverksenheterna. De flesta större företag har specifika OID:er som ligger under trädgrenen "private" i MIB:ar. Men då gäller det att veta vilket företag som tillverkat enheten samt hitta en sida med dess OID:er, vilket inte varit det lättaste. Det finns dock OID:er som fungerar för alla enheter under trädgrenen "mgmt", såsom sysName.

Eftersom de flesta böcker och webbsidor på Internet varit skrivna på engelska har det under själva rapportskrivningen varit svårigheter med att veta översättningen för en viss term, t.ex. trap, eller veta om den engelska termen skall användas eller den svenska.

9.3 Förslag på vidareutveckling

Det som inte kunde implementeras inom projektets tidsramar var lagring av data i databasen för skrivare och switchar. Ett första steg för att slutföra projektet är att ta fram de OID:er för skrivare och switchar som programmet skall kunna hantera. Det kan antingen utföras ge-

nom att leta reda på OID:erna via webbsidor på Internet, eller genom att koppla till enheten till en dator som har UCD-SNMP installerat och lista alla OID:er enheten har med programmet snmpwalk. Nästa steg är att implementera funktionalitet i modulen SNMP-server för inhämtning av data från skrivare och switch och för lagring av den inhämtade data i databasen. Det kan implementeras på ungefär samma sätt som för implementationen av inhämtning och lagring av data från klient och server.

10 Slutsatser

Efter att ha slutfört projektet blev resultatet en testad och fungerande SNMP-baserad server. Programmet är väl dokumenterat och är därför enkelt att sätta sig in i, så förhoppningsvis kan det komma att vara till användning i framtiden. Eftersom SNMP var ett nytt område gav det många nya kunskaper om inte bara SNMP utan även programmering och nätverkshantering i sig. Denna kunskap är säkerligen något som kommer att vara användbart för framtida projekt, då Internet används mer och mer och då SNMP säkerligen kommer att vara det gällande nätverksprotokollet för de närmaste åren. Man kan då hoppas att SNMPv3 kommer att användas på fler plattformar än det gör nu då version ett har så dåliga säkerhetskontroller.

Projektet kunde dock inte slutföras inom den utsatta tidsramen. Detta var mycket p.g.a. att det uppskattades vara mindre problem med implementationen än vad det faktiskt var. Lärdomen man kan dra av detta är att det ofta är svårt att uppskatta hur lång tid en implementation tar av ett program.

Litteraturförteckning

[1] Kernighan Brian W., Ritchie Dennis M., The C programming Language, Prentice Hall International 1989

[2] Brown Doug, Levin John R., Mason Tony, lex & yacc, O'reilly 1995

[3] Mauro Douglas R, Schmidt Kevin J., Essential SNMP, O'Reilly 2001

[4] McCloghrie Keith, Rose Marshall T., How To Manage Your Network Using SNMP, P T R Prenticehall 1995

[5] Net-SNMP hemsida http://net-snmp.sourceforge.net/, 2003-12-12

[6] MySQL hemsida http://www.mysql.com, 2003-12-12

[7] Stallings, William, SNMP, SNMPv2, SNMPv3 and RMON1 and 2 Third Edition, Addison-Wesley 1999

[8] Svenska datatermgruppen http://www.nada.kth.se/dataterm/, 2003-12-12

[9] Svenska datatermer http://www.tnc.se/html/2datafrag.html, 2003-12-12

A Terminologi

Agent

En applikation i en enhet som rapporterar statusen för en enhet vid begäran av en manager.

ASN.1 (Abstract Syntax Notation One)

Ett formellt språk som används för att definiera en MIB. Användandet av ASN.1 medför att konflikter mellan olika datorers representation av data inte uppstår.

BER (Basic Encoding Rules)

Definierar hur objekt är kodade och avkodade så att de kan skickas över nätet.

CMIP (Common Management Information Protocol)

Ett protokoll över OSI som förutsågs ersätta SNMP, men har inte kommit att användas.

Community string

Lösenord i SNMP-meddelanden som används vid autenticering av agenter då en manager skall skriva eller läsa ett variabelvärde.

HEMS (High-Level Entity Management System)

Ett protkoll som var en variant av det troligtvis första övervankningsprotkollet HMP, (Host Monitoring Protocol).

IAB (Internet Architechture Board)

Ett tekniskt rådgivande organ inom Internet society, grundat 1983.

Manager

En applikation som används vid övervakning av nätverksenheter.

MIB (Management Information Base)

En databas som specificerar de variabler som agenten hanterar. Variablerna är hierarkiskt ordande i en trädstruktur och varje variabel kan identifieras med hjälp av en s.k. OID.

OID (Object Identifier)
Ett sätt att unikt identifiera de objekt som finns i en MIB. Skrivs som en serie av nummer separerade med punkter, eller som ett namn.

PDU (Protocol Data Unit)
Ett datagram som är den minsta enheten som används vid nätverksöverföring.

Polling
Då managern frågar en agent om information.

Skalära objekt
Objekttyp i SNMP som endast innehåller en instans av ett visst objekt. T. ex. antal papper i en skrivare.

SMI (Structure of Management Information)
En standard som specificerar formatet för de objekt och enheter som SNMP läser och skriver. Den specificerar ASN.1 datatyper, SMI datatyper och MIB-tabeller.

SNMP (Simple Network Management Protocol)
Ett applikationslagerprotokoll som kontrollerar de fysiska enheterna i ett nätverk genom att utbyta information med dem.

TCP/IP (Transmission Control Protocol/Internet Protocol)
Ett standardprotokoll för att överföring av data på ett nätverk.

Trap
Det som agenten skickar asynkront för att informera mangern om att något viktigt inträffat i agenten.

UDP (User Datagram Protocol)
Ett snabbt protokoll inkluderat i TCP/IP som inte kräver uppkoppling.

WRAPPER

En wrapper är en funktion som kapslar in andra funktioner och därmed erbjuder ett enklare gränssnitt

VMware Workstation 4

En mjukvara från VMware Inc. med vilket det är möjligt att simulerar PC-datorer. Med VMware Workstation 4 kan en användare installera och använda fler än ett operativsystem på en och samma dator.

B Dokumentation av databasen

Appendix C är en samling av den dokumentation som sammanställts för databasen vilken används i implementationen av övervakningsservern.

Tabell namn **syss**
Beskrivning System information

Key	Name	Type	Length	Description
PK	sys_id	int	10	Systemets unika id
	sys_sysName	varchar	30	Systemets unika namn i nätet
	sys_sysDescr	varchar	200	Beskrivning av systemet
	sys_sysContact	varchar	50	Kontaktperson för systemet
	sys_sysLocation	varchar	30	Systemets fysiska position
	sys_sysUpTime	int	10	Systemets uptime
	sys_lastUpdate_op	int	11	Tidpunkt senast updaterad av systemansvarig <utime>
	sys_lastUpdate_sys	int	11	Tidpunkt senast updaterad av systemet självt <utime>
FK	sys_type_nr	int	10	Refererar till sys_types:sys_type_id

Tabell namn **sys_types**
Beskrivning Systemets enhets typ

Key	Name	Type	Length	Description
PK	sys_type_id	int	10	Systemtypens unika id
	sys_type_descr	varchar	20	Anger typ av system

Tabell namn **sys_nics**
Beskrivning Systemets Ethernet adresser

Key	Name	Type	Length	Description
PK	sys_nic_id	int	10	NICets unika id
FK	sys_nr	int	10	Refererar till syss:sys_id
	sys_nic_PhysAddress	char	17	NICets MAC-adress

Tabell namn **sys_ips**
Beskrivning Systemets IP-adresser

Key	Name	Type	Length	Description
PK	sys_ip_id	int	10	IP-Adressens unika id
FK	sys_nic_nr	int	10	Refererar till sys_nics:sys_nic_id
	sys_ip_NetAddress	varchar	15	En av IP-Adresserna som tillhör NICet

Tabell namn **prn_es**