Gestione delle Reti: sinossi

Danilo Giannico

June 2022

Contents

1 Introduzione

1.1 Sistema di gestione

Perché?

- Monitorare in maniera automatica
- Analizzare guasti e cause
- Attivare operazioni automatiche
- Gestione **proattiva**(anticipa problemi) ma anche **reattiva**

SNMP

- Simple Network Management Protocol
- Alternativa: OSIMIS (OSI Management Information Service)
- 1988
- Economicità
- Semplicità
- $->$ Successo commerciale
- SNMPv2 (1993): maggior efficienza e funzionalità
- SNMPv3 (1998): funzionalità di sicurezza $->$ pesante, non usato
- Software di gestione utilizzato in laboratorio: HP OpenView

RMON

- Remote Newtork Monitoring
- Estensione di SNMP
- Monitorare da remoto intere LAN
- LAN come oggetti

1.2 Requisiti di un Sistema di Gestione

$->$ 5 aree funzionali:

- Fault Management
- Accounting management
- Configuration Mangement
- Performance Management
- Security Management

Servono **protocollo** di gestione (SNMP) e **software** di gestione

1.2.1 Fault Management

$->$ Gestione dei guasti

Guasto

- + Monitoring
- Determinare dove si è verificato
- Isolare resto rete/sistema dal guasto
- Riconfigurare rete/sistema per minimizzare impatto (*workaround*)
- Riparare e riportare rete/sistema a stato iniziale

$->$ Minimizzare *tempo di mancanza di servizio* (magari in maniera automatizzata) $->$ impatta su QoS (*Quality of Service*)

$->$ Segnalazione da parte degli utenti finali è già un fallimento (bisogna accorgersene prima)

1.2.2 Accounting Management

- + Monitoring
- Contabilizzazione dei costi o dell'uso delle risorse del sistema
- Evitare uso inefficiente o abuso delle risorse da parte degli utenti

- Di solito utenti interni ma anche esterni (es. servizi con profili a pagamento)
- Protocollo ha un ruolo limitato qui (non può essere standard, fatto da applicativo)

1.2.3 Configuration and Name Management

- + Control
- Configurazione componenti e sottosistemi
- Set variabili di configurazione
- Inizializzazione e shutdown di una rete/sistema/componente
- Spesso archivio con diverse configurazioni di rete e possibilità di passare facilmente da una all'altra
- Esempi: l'indirizzo IP ad una interfaccia, impostare il nome di dominio di una macchina, ecc.

1.2.4 Performance Management

- + Monitoring
- Efficacia applicazioni spesso si basa sulle prestazioni di componenti del sistema
- Identificazione di **indici di prestazione** $->$ Standard o nuovi
- Analisi delle prestazioni
- Previsione delle prestazioni $->$ si basa su statistiche sui dati che devono essere raccolte con regolarità

1.2.5 Security Management

- + Control
- Protezione della informazione
- Controllo degli accessi
- Guarderemo solo sicurezza del sistema di gestione e non del sistema gestito

1.3 Architettura di un sistema di gestione

Network Management System (NMS):

- Collezione di strumenti hardware e software
- Singola interfaccia semplice ma funzionale
- Visione completa e unificata dell'intera rete gestita

Architettura:

- Applicazione: **NMA** $->$ *Network Management Applicazion* $->$ Manager
- Entità: **NME** $->$ *Network Management Entity* $->$ Agent

$->$ NMA su macchina ad alte prestazioni (*Network Control Host*)

$->$ Agent spesso incluso nel sistema operativo (vanno solo configurate):

- maggior efficienza;
- gestire meglio risorse dispositivo

$->$ Anche sulla macchina del manager è presente un agent per essere gestita

1.3.1 Attività della NME e della NMA

NME:

- Raccogliere e mantenere informazioni sulle attività del dispositivo e su risorse fisiche e logiche
- Immagazzinare queste informazioni localmente $->$ MIB
- Rispondere al manager che gli può chiedere di:
  - Trasmettere informazioni raccolte
  - Modificare parametri
  - Fornire informazioni di stato
  - Generare traffico artificiale per fare test
- Inviare in modo autonomo messaggi di avviso (**trap**) quando necessario, esempio:

  - warning
  - avvisi di guasti
  - superamento soglie

- $->$ In modo da risparmiare messaggi scambiati e quindi banda
- $->$ Velocizzare il rilevamento dei guasti

NMA:

- Implementa la logica applicativa della gestione $->$ sviluppare 5 aree funzionali descritte prima
- Almeno due host $->$ ridondanza

1.3.2 Architettura di una NMA

Componenti:

- Software per la presentazione dei dati all'utente $->$ *Unified User Interface* $->$ interfaccia omogenea:
  - Facilità di utilizzo
  - Ridurre ambiguità e errori
  - Ridurre tempo di formazione del manager umano
- Network Management software $->$ Application Engine
- Software di comunicazione e gestione database

$->$ Occorre fornire strumenti adeguati per organizzare, riassumere e semplificare l'enorme quantità di informazione a disposizione.

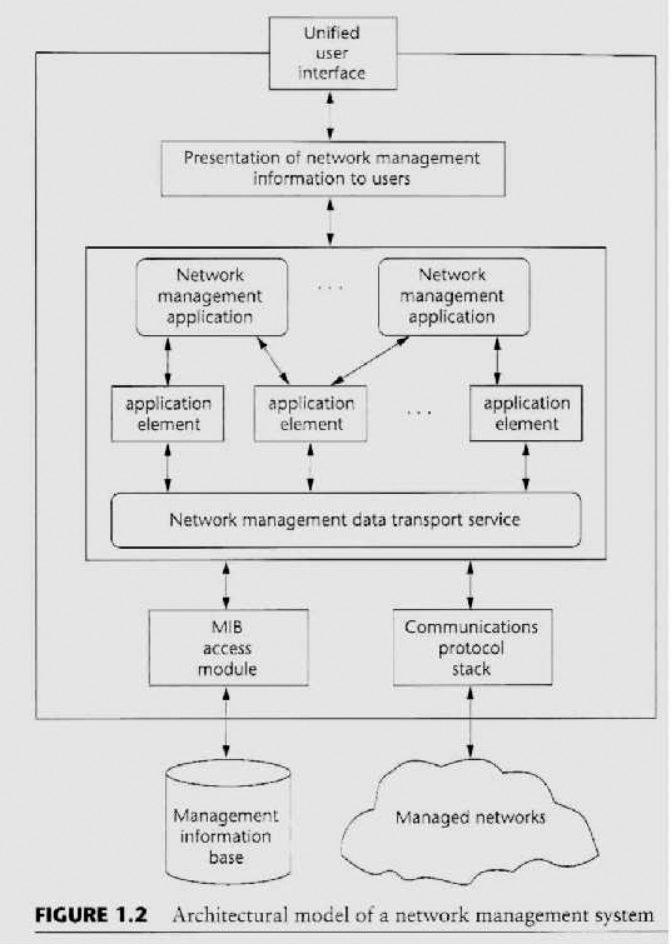


FIGURE 1.2 Architectural model of a network management system

Figure 1: Architettura di un NMS

- *Network management application*: realizzano la logica applicativa e quindi le 5 aree funzionali
- *Application Element*: componenti di più basso livello che forniscono funzionalità (es. matematiche, grafiche, statistiche) necessarie $->$ librerie
- *Servizio di trasporto dei dati di management*: mettere la NMA in comunicazione con le informazioni che le sono necessarie
- *Stack di protocolli di comunicazione*: per comunicare con agent $->$ Dati istantanei

- *Modulo di accesso alla MIB*: permette accesso ad un database che contiene informazione di gestione $->$ *Management Information Base* $->$ Dati storici

1.4 La Management Information Base (MIB)

- Collezione di informazioni di gestione organizzata gerarchicamente
- Struttura ad albero:
  - Foglie rappresentano oggetti gestiti
  - Nodi intermedi rappresentano gruppi logici (es. TCP, interfaces, ecc.)
- Obiettivo $->$ Standardizzare lo *schema dei dati* degli Agent, in modo che dispositivi di diversa natura, o di vendor diversi, siano gestibili nello stesso modo.
- Standard (RFC) chiamato *Structure of Management Information* (**SMI**), sottoinsieme del linguaggio astratto **ASN.1** (*Abstract Syntax Notation One*) $->$ definisce linguaggio e modalità per esprimere lo schema delle MIB dei dispositivi, quali tipi di dato si possono utilizzare e i formati dei pacchetti SNMP.

1.5 Gestione Distribuita

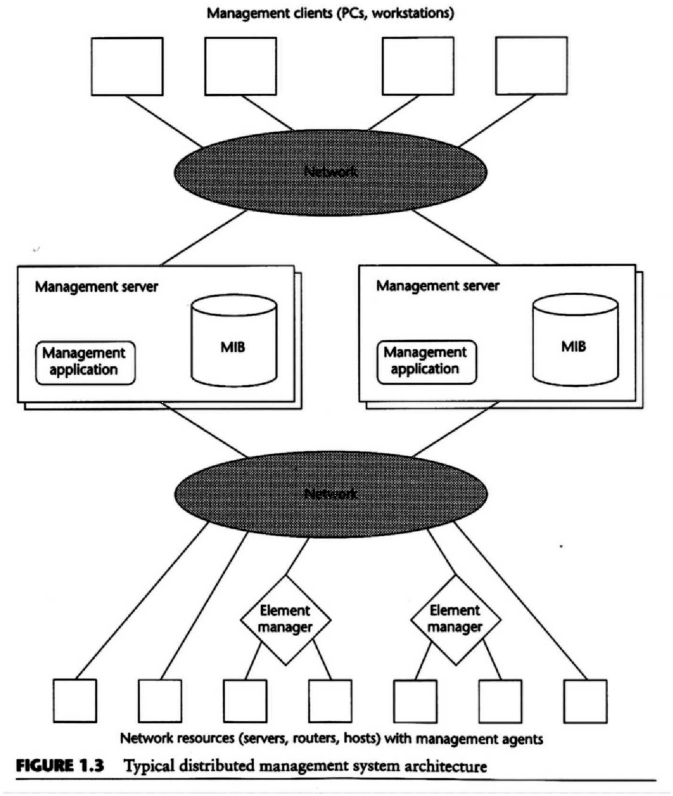


Figure 2: Tipica architettura di un sistema di gestione distribuito

- Almeno due host per manager per ragioni di affidabilità e di continuità del servizio (Altrimenti *Single Point of Failure*)
- *Element Manager*: intermediario $->$ **Proxy**. Quando:
  - Dispositivo non può ospitare agent (es. risorse limitate)
  - Dispositivo ospita agent che usa un altro protocollo proprietario

  $->$ Solitamente schema a RPC

2 Network Monitoring

− > Osservare e analizzare stato e comportamento di:

- Applicazioni
- Sistemi
- Sotto-reti

− > Vari aspetti coinvolti:

- *Accesso alla informazione da monitorare*: informazione deve essere definita e strutturata − > MIB
- *Meccanismi di monitoring*: devono essere **efficienti** per non compromettere il sistema gestito
- *Utilizzo della informazione di monitoring*: finalizzato a realizzare le 5 aree funzionali di gestione. Noi vedremo le principali: *Performance*, *Fault*, *Accounting*

2.1 Architettura di monitoring

2.1.1 Monitoring Information

- **Statica**: caratterizza configurazione sistema. Cambia molto raramente. Es: topologia rete, applicazioni installate
- **Dinamica**: caratterizza eventi che avvengono nel sistema. Cambia molto frequentemente. Es: cambio di stato, login di un utente
- **Statistica**: caratterizza evoluzione della informazione dinamica. Rappresenta una "sintesi" di grandi quantità di informazioni. Es. numero di pacchetti inviati nell'unità di tempo, media e varianza ecc.

− > Natura della informazione implica dove può essere *generata* o *raccolta* e su dove può essere *conservata*:

- Statica: generata dall'elemento che essa descrive.
- Dinamica: raccolta dall'elemento che essa descrive. Evento potrebbe essere osservato dall'esterno, quindi anche da elemento diverso − > *Remote Monitor*
- Statistica: generata da qualunque elemento abbia accesso all'informazione dinamica.

2.1.2 Configurazioni di monitoring

Un sistema di monitoring può essere scomposto in 4/5 componenti fondamentali:

- *Applicazione di monitoring*: include le funzioni visibili all'utente del sistema di monitoring
- *Funzione di Manager*: il modulo che fornisce le funzioni di base di raccolta delle informazioni di monitoring dagli altri elementi del sistema
- *Funzione di Agent*: il modulo che registra e fornisce le informazioni di monoitoring da uno o più elementi del sistema e le comunica al manager
- *Managed objects*: l'informazione di management che rappresenta le risorse e le loro attività
- *Monitoring Agent*: modulo addizionale per evitare che grandi quantità di informazioni vengano trasferite sulla rete. Genera sommari e analisi statistici.

2.1.3 Polling e Event Reporting

Due modi di fornire informazioni al manager:

Polling :

- Interazione di tipo *richiesta-risposta*
- Manager chiede informazioni all'agent e agent risponde
- **GET - SET**
- Informazioni richieste possono essere di varia natura ma ovviamente devono essere nella MIB dell'agent. Agent deve essere in grado di eseguire una ricerca all'interno della sua MIB. Oppure informazioni riguardo la MIB stessa contenuta nell'agent
- Vantaggio: decidere cosa e quando richiedere informazioni
- Svantaggio: se ci sono tanti dispositivi si rischia di saturare la rete

Event reporting :

- Agent invia autonomamente informazioni al manager
- Informazioni di tipo diverso.
- Agent può generare un report periodico
- Informazione fornita e periodo di generazione del report predefinite dell'agent oppure configurate dal manager (con il polling).
- **TRAP**
- Es. quando avviene un evento significativo o inusuale come un guasto
- Vantaggi: risparmiare messaggi scambiati e velocizzare rilevamento guasti
- Svantaggio: se l'evento ha compromesso il dispositivo, l'agent potrebbe non essere in grado di inviare la trap

Scelta su tecnica da utilizzare dipende da:

- Traffico di rete generato
- Robustezza in situazioni critiche
- Costi di elaborazione richiesti
- Applicazioni di gestione supportate
- Natura della informazione:
  - Prettamente statica: event reporting
  - Prettamente dinamica: polling

2.2 Performance Monitoring

$->$ Scegliere **indicatori di prestazione**:

- *Orientati al servizio*: misurano la qualità del servizio offerto. Esempi: *disponibilità*, *tempo di risposta*, *accuratezza* $->$ mantenere alta la soddisfazione dell'utente
- *Orientati all'efficienza*: sono più generici e misurano le prestazioni del sistema che sono di interesse per chi realizza il servizio. Esempi: *throughput*, *coefficiente di utilizzazione* $->$ diminuire costi

Disponibilità

- Percentuale di tempo che il sistema è *disponibile* per i suoi utilizzatori
- *Disponibilità binaria*: sistema è ON / sistema è OFF
- *Disponibilità funzionale*: quando la disponibilità può essere solo parziale, in modo da tenere conto delle funzionalità limitate che il sistema non completamente funzionante ha comunque offerto ai suoi utenti
- Si basa su *affidabilità* dei componenti del sistema = *probabilità* che il componente si comporti bene
- La disponibilità è spesso espressa in termini di tempo medio fra due fallimenti (*Mean Time Between Failures*, MTBF), e di tempo medio per riparare (*Mean Time To Repair*, MTTR) il fallimento, nel seguente modo:
$$A = \frac{MTBF}{(MTBF + MTTR)}$$
- Esempio: se abbiamo due modem collegati in serie da una linea di comunicazione e indichiamo con M e L la loro disponibilità, la disponibilità del sistema complessivo è:
$$A = M * L * M$$

Tempo di risposta

- Tempo impiegato dal sistema per (iniziare a) fornire la risposta all'input da parte dell'utente
- Per un *sistema server* è in generale il tempo che impiega a rispondere alla richiesta che gli arriva dal client
- Se il client è esterno al sistema, non si considerano le latenze della rete che connette il client al server perché non dipendono dal sistema in sé $->$ la prestazione della rete di comunicazione va comunque monitorata per segnalare eventuali problemi al fornitore (Service Provider)
- Parametro prestazionale molto critico perché un tempo di risposta inadeguato può rendere il sistema totalmente inadatto a certi usi
- La diminuizione del tempo di risposta aumenta la produttività dell'utente

- In un sistema client/server in cui ci interessa il tempo di risposta all'utente interattivo abbiamo almeno tre componenti: client-rete-server.
- Nel *monitorare* e *analizzare* il tempo di risposta di un sistema è essenziale conoscere la topologia della rete gestita e quali sono i componenti da cui transita la richiesta/risposta.

Accuratezza

- La percentuale dei risultati corretti sui risultati totali
- Risultati corretti = risultati che rispettano le specifiche
- Si vorrebbe che fosse tendenzialmente pari a 1
- Misura alternativa complementare: *percentuale di errori*
- Nei protocolli di comunicazione: *integrità dei dati ricevuti*
- Monitorare i file di log per misurare situazioni di errore

Throughput $->$ Portata

- Misura di qualcosa *per un periodo sostanziale di tempo*, esempi:
  - Numero di transazioni al secondo
  - Numero di sessioni utente al minuto
  - Numero di byte trasferiti al secondo
- Misura *media* e non *istantanea*
- Misura *interna* al sistema, non visibile all'utilizzatore
- Si vuole misurare la capacità del sistema di sopportare un carico alto
- Per farlo bisogna sovraccaricare il sistema con un numero elevato di richieste per un periodo sufficientemente lungo e misurare qui il tempo medio di risposta e il tempo necessario a soddisfare tutte le richieste

Utilizzazione

- Percentuale di tempo in cui una risorsa è in uso in un certo periodo di tempo
- Oppure percentuale di utilizzo di una risorsa in un determinato istante di tempo. Esempio: utilizzazione banda
- Misura di prestazione interna al sistema
- Importante perché spesso le prestazioni del sistema degradano in modo esponenziale con il crescere dell'utilizzazione
- Monitorare che l'utilizzazione delle varie risorse del sistema sia "equilibrato", cioè non vi siano risorse che sono molto più utilizzate di altre

2.2.1 Performance Monitoring

$->$ 3 componenti:

- *Misura delle prestazioni*: raccolta di statistiche sul traffico e dei tempi di risposta
- *Analisi delle prestazioni*: interpretare, ridurre e presentare i dati raccolti $->$ *pianificare la crescita* del sistema (*capacity planning*)
- *Generazione di traffico sintetico*: permette di osservare il comportamento del sistema sotto carico controllato.

2.2.2 Misure esaustive vs statistiche

- *Misura esaustiva*: può essere inaccettabile e irrealizzabile quando il carico del sistema è elevato e i dati da raccogliere sono estremamente numerosi
- *Misura statistica*: considera ogni grandezza da misurare come una *variabile aleatoria* le cui caratteristiche devono essere valutate mediante un appropriato campionamento

2.3 Fault Monitoring

Obiettivi:

- Identificare il guasto il più presto possibile

- Identificare la causa del guasto, per poter prendere azioni correttive

$->$ Indispensabile per realizzare un'alta disponibilità del sistema e garantire QoS agli utenti

2.3.1 Problemi nel fault monitoring

$->$ Complessità del sistema comporta:

- *Fault non osservabili*: per loro natura (es. deadlock) o per carenza di strumentazione (es. dispositivo non segnala il guasto)
- *Fault parzialmente osservabili*: osservabile ma non è sufficiente per individuarne la causa. Esempio: un componente può comportarsi in modo scorretto a causa del malfunzionamento di un altro componente a lui collegato $->$ Diagnosi più difficile
- *Incertezza nella osservazione*: l'osservazione del guasto può non essere significativa, per esempio per la complessità della rete

$->$ Problemi nell'individuazione della causa:

- *Molte cause potenziali*: quando sono coinvolte diverse tecnologie, lo stesso malfunzionamento può essere causato da cause di natura molto diversa $->$ Quando un servizio cade il guasto che ne causa l'assenza deve essere ricercato in tutti i possibili punti di fallimento (alcuni potrebbero essere però non gestibili, es. problema su una dorsale)
- *Più cause*: concomitanza di più eventi che ha causato il malfunzionamento
- *Assenza di strumenti automatici di test*: strumenti per individuare e isolare i guasti. Utili ma difficili e costosi da amministrare

2.3.2 Funzioni di fault monitoring

Un sistema di fault monitoring deve rispondere a numerosi requisiti:

- *Individuare e riportare al manager* i guasti $->$ implica la capacità di riconoscerli
- *Mantenere log* di eventi significativi e di potenziali errori. I manager controllerà periodicamente questi log attraverso il **polling**

- *Anticipare i fault* (gestione proattiva): necessario che l'agent permetta di impostare soglie di allarme e sia in grado di riportare l'evento di superamento delle soglie (**trap**)

$->$ Essenziale: efficace interfaccia utente

2.4 Accounting Monitoring

- Controllo della contabilizzazione
- Tenere traccia dell'uso che gli utenti fanno delle risorse
- Requisiti variano a seconda degli ambienti e delle finalità. Esempio: se c'è una fornitura di servizio, occorre raccogliere dati che permettano di *fatturare* il costo dell'uso delle risorse al singolo utente del sistema
- Esempi di risorse:
  - Servizi di comunicazione
  - Hardware
  - Software e sistemi
  - Servizi
- Esempi di dati da raccogliere:
  - Identificazione dell'utente
  - Risorsa utilizzata
  - Numero di pacchetti
  - Livello di sicurezza richiesto
  - Timestamp

3 Network Control

- *Monitoring*: monitorare, tenere sotto esame
- *Control*: determinare e dettare il comportamento

$->$ Essere in grado di modificare (opportunamente) i parametri del sistema gestito e di far eseguire azioni specifiche

3.1 Configuration Control

- Inizializzare, mantenere in esercizio e spegnere sia componenti individuali che sottosistemi
- Inizializzazione: occorre *identificare* e *specificare* le caratteristiche dei componenti e delle risorse che costituiscono il sistema da gestire e in che modo questi interagiscono
- Include le seguenti funzioni:
  - Definire l'informazione di configurazione
  - Settare e modificare valori di attributi
  - Definire e modificare relazioni
  - Inizializzare e terminare operazioni all'interno del sistema
  - Distribuire il software
  - Esaminare valori e relazioni (Monitoring)
  - Riportare lo stato / il cambiamento di stato della configurazione (Monitoring)

3.1.1 Definire l'informazione di configurazione

- Descrive la natura e lo stato delle risorse
- Quali sono le risorse, fisiche e logiche
- Quali loro attributi interessa specificare
- Raramente il sistema di gestione è in grado di modificare quali informazioni siano disponibili (troppo complesso da gestire e organizzare). Quali informazioni siano disponibili dipende dall'architettura di gestione presente $->$ in SNMP sono le MIB che determinano questo confine

3.1.2 Settare e modificare valori di attributi

- Manager deve essere autorizzato a modificare da remoto i valori degli attributi negli agent
- Attributi che riflettono la natura della risorsa non devono essere modificabili (potrebbero avere gravi conseguenze sul sistema gestito). Esempio: numero e natura schede di rete di un router
- Vari tipi di modifica:
  - *Modifica nel solo database*: modifica non ha effetti sul comportamento della risorsa. Esempio: modifica del nome del responsabile del server
  - *Modifica nel database e modifica nella risorsa*: la risorsa viene modificata perché l'attributo determina almeno in parte il comportamento della risorsa. Esempio: disabilitare scheda di un router
  - *Modifica nel database e azione*: ordinare l'esecuzione di azioni. In SNMP questo non è disponibile ma le azioni possono essere intraprese dall'agent in risposta alla modifica di attributi della risorsa

3.1.3 Definire e modificare relazioni

Una relazione fra componenti o risorse può descrivere:

- Una associazione
- Una connessione
- Una condizione

3.1.4 Inizializzare e terminare operazioni

- Possibilità di far partire e arrestare operazioni o componenti
- Inizializzazione: deve verificare che tutte le inizializzazioni dei parametri siano state effettuate e le relazioni definite appropriatamente
- Terminazione: deve permettere la richiesta del salvataggio o meno di configurazioni o informazioni statistiche

3.1.5 Distribuire Software

- Dipende dal sistema di gestione
- Oltre a eseguibili dovrebbero essere distribuite anche informazioni accessorie che guidano l'esecuzione del software. Esempio: distribuzione tabelle di routing

3.2 Security Control

Funzionalità che devono essere garantite:

- *Segretezza / Confidenzialità*
- *Integrità*
- *Disponibilità*

Attacchi:

- *Interruzione*: contro *disponibilità*. Esempi: *ARP poisoning, Denial of Service*
- *Intercettazione*: contro *segretezza*. Esempio: *Eavesdropping*
- *Modifica*: contro *integrità* (intercetto e modifico). Esempio: *Man in the middle*
- *Contraffazione:* contro *integrità* (invio messaggio contraffatto in modo autonomo). Esempio: *Fabrication*

4 I Concetti di Gestione di SNMP

Simple Network Management Protocol $->$ Protocollo ma indica anche tutta l'infrastruttura di gestione:

- Concetti
- Modalità
- Strutture dati

4.1 Retroscena

- All'inizio poca attenzione al problema della gestione di Internet $->$ solo con ICMP
- SNMP: 1988, RFC 1067 (attuale: 1157)
- Standard disponibile, semplice e aperto $->$ Successo e diffusione come standard di gestione di reti e sistemi distribuiti
- RMON: estensione che permette di monitorare da remoto una sottorete come un'unica entità

4.1.1 Standard relativi a SNMP

- *Structure of Management Information* (SMI) $->$ Descrive come devono essere definiti i *managed object* che sono contenuti nella MIB
- *Management Information Base* (MIB-II) $->$ Descrive i managed object *minimi* che devono essere contenuti nel database di gestione
- *Simple Network Management Protocol* (SNMP) $->$ Definisce il protocollo che deve essere usato per operare sui managed object

4.2 Concetti fondamentali

4.2.1 Architettura di Gestione

Gli standard definiscono un modello che contiene 4 elementi chiave:

- *Stazione di gestione* $->$ **Manager**
- *Agente di gestione* $->$ **Agent**

- *Base informativa di gestione* $->$ **MIB**
- *Protocollo di gestione* $->$ **SNMP**

- Monitoring $->$ Leggendo i valori dei managed object
- Control $->$ Settando il valore dei managed object
- Azioni di gestione $->$ settando valori di particolari managed object, perché in SNMP non si possono invocare esplicitamente azioni sull'agente

4.2.2 Architettura del Protocollo di Gestione

- Protocollo applicativo
- Usa UDP
- Nel *Protocol Data Unit* (pacchetto) vengono scambiati oggetti che rappresentano risorse gestite

Tipi di messaggi:

- *Manager*:
  - *GetRequest*
  - *GetNextRequest*
  - *SetRequest*
- *Agent*:
  - *GetResponse* $->$ Riscontro: OID + valore
  - *Trap* $->$ È un oggetto che ha un OID e può contenere altri oggetti

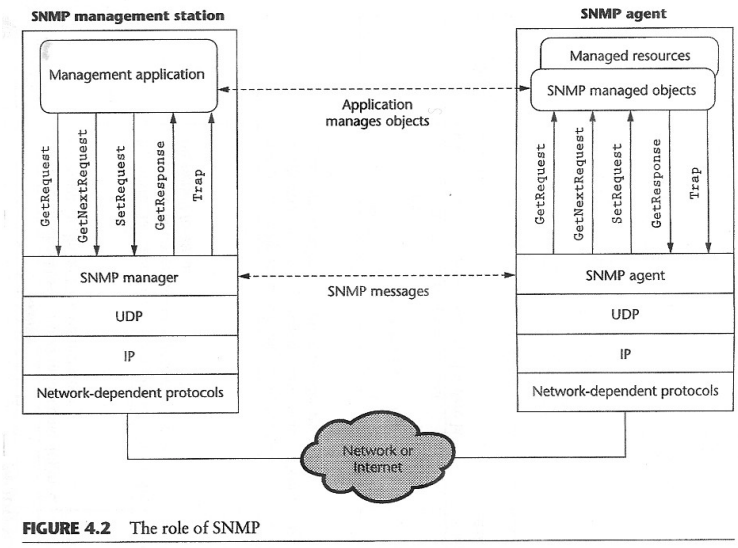


Figure 3: Funzionamento di SNMP

4.2.3 Polling guidato da Trap

Quando il numero di agenti è troppo elevato, non è accettabile richiedere periodicamente il valore di tutti gli oggetti di tutti gli agent:

- Tutte queste informazioni non sono in realtà necessarie
- Troppo traffico e troppi costi di gestione
- Elevato carico delle CPU del manager e degli agent

Strategia suggerita (polling guidato da trap):

- Polling completo: molto raramente (es. una volta al giorno). Solo informazioni essenziali richieste a tutti gli agent
- Polling periodico: poche informazioni critiche
- Polling mirato: solo a seguito di una trap che segnala un eventuale problema

$->$ Riduzione della capacità della rete utilizzata e della potenza necessaria sugli agent

Richiede analisi preliminare:

- Dei dati delle MIB che sono rilevanti per gli obiettivi di gestione del sistema
- Dei danni provocati da ciascun eventuale malfunzionamento
- Delle informazioni che ci permettono di rilevare il malfunzionamento
- Delle informazioni che ci permettono di accertare il malfunzionamento
- Delle informazioni che ci permettono di effettuare la diagnosi, ossia di individuare le cause che devono essere rimosse

4.2.4 Proxy

SNMP richiede che agent supporti tutto lo stack SNMP/UDP/IP $->$ requisito può essere difficile o troppo costoso perché:

1. Dispositivi di piccola potenza di calcolo e memoria che non sono in grado di ospitare tutto il software necessario
2. Dispositivo supporta TCP/IP ma non ha agente SNMP per questioni di costi
3. Dispositivo gestibile con protocolli diversi

$->$ Proxy agent $->$ Funzione di **mapping**:

- A livello di messaggi (richieste, risposte, segnalazioni)
- Tra oggetti gestiti

5 L'informazione di Gestione in SNMP

- Ogni risorsa da gestire è rappresentata da un oggetto, e la collezione di questi oggetti costituisce la MIB
- Struttura gerarchica ad albero, in cui le foglie sono gli oggetti (e quindi le risorse) e i nodi intermedi sono gruppi logici
- *Monitoring*: Lettura valori degli oggetti della MIB
- *Controlling*: Modifica valori degli oggetti della MIB

$->$ Perché una MIB sia funzionale alla gestione deve soddisfare alcuni requisiti:

1. Gli oggetti usati per rappresentare una certa risorsa devono essere gli stessi in ogni sistema dello stesso tipo. Esempio: per TCP, numero di connessioni attive e numero di passive $->$ Schema della MIB (MIB standard) per un dato sistema (definito attraverso SMI)
2. Si deve usare uno schema comune per rappresentare le informazioni $->$ **SMI**

5.1 Structure of Management Information (SMI) - Introduzione

$->$ Standard di Internet definito nell'RFC 1155. Definisce linguaggio e modalità per esprimere lo schema delle MIB dei dispositivi, quali tipi di dato si possono utilizzare e i formati dei pacchetti SNMP.

La SMI specifica numerosi aspetti delle MIB:

- Tipi di dato che possono essere usati
- Come le risorse descritte possono essere rappresentate
- Come si può dare un nome alle risorse di una MIB

Tipi di dati ammessi nella definizione delle sintassi dei tipi di oggetti:

- *Scalari*: tipi di dati semplici, senza struttura
- *Tabelle*: bidimensionali di scalari $->$ Semplcità alla base per:

- Semplificare l'implementazione degli agenti $->$ ridurre costi
- Aumentare l'interoperabilità

$->$ Occorre:

1. Tecnica standardizzata per definire la *struttura* di ogni MIB [1] $->$ Schema della MIB
2. Tecnica standardizzata per definire i singoli *oggetti*, incluso il tipo del loro valore $->$ Schema dei dati
3. Tecnica standardizzata per definire la *codifica* dei valori degli oggetti, in modo che possano essere scambiati tra entità $->$ Formato dei pacchetti

Per fare uno standard multivendor, e ottenere quindi (1) e (2), non bisogna mai far riferimento a:

- Architettura hardware
- Sistema operativo
- Linguaggio di programmazione
- Scelte implementative da lasciare all'implementatore

1. e 2. $->$ *Linguaggio di specifica ASN.1*

3. $->$ *Regole di codifica BER*

5.2 Abstract Syntax Notation One (ASN.1)

- Linguaggio formale standardizzato
- Serve per definire la *sintassi astratta* dei dati utilizzati da applicazioni distribuite
- Serve per definire la *struttura delle PDU* delle applicazioni

$->$ Definire struttura della Management Information sia in SNMP che in OSIMIS

[1] Su un agent si trova sempre la MIB-II; poi potrebbero trovarsi anche MIB standard per quel dispositivo oppure MIB vendor

5.2.1 Sintassi astratta

$->$ La sintassi astratta è la descrizione della *struttura dei dati* usati da un protocollo

Tipo di dato $->$ Per tipo di dato intendiamo un *insieme di valori* dotato di *nome*. Può essere:

- *Semplice*: definito specificando i valori che appartengono all'insieme
- *Strutturato*: definito in termini di altri tipi di dato con una *operazione/relazione* di *composizione* di tali tipi

Codifica $->$ Per codifica intendiamo la *sequenza di ottetti* che viene usata per rappresentare il valore appartenente ad un tipo di dato

Sintassi di trasferimento $->$ Per sintassi di trasferimento intendiamo la generica struttura con cui i dati sono *codificati in termini di ottetti* nel trasferimento fra due entità (applicative) paritarie

Regole di codifica $->$ Specificano un particolare *mapping (traduzione)* fra sintassi astratta e sintassi di trasferimento

Architettura di comunicazione:

- *Data Transfer Component*: componente che si occupa del trasferimento dei dati tra due end system (TCP / UDP e strati sottostanti) $->$ Dati come *sequenze di ottetti*
- *Application Component*: utilizza il *data transfer component* e *realizza* l'applicazione che l'utente finale utilizza (SNMP) $->$ Dati come *informazione dotata di tipo e struttura*

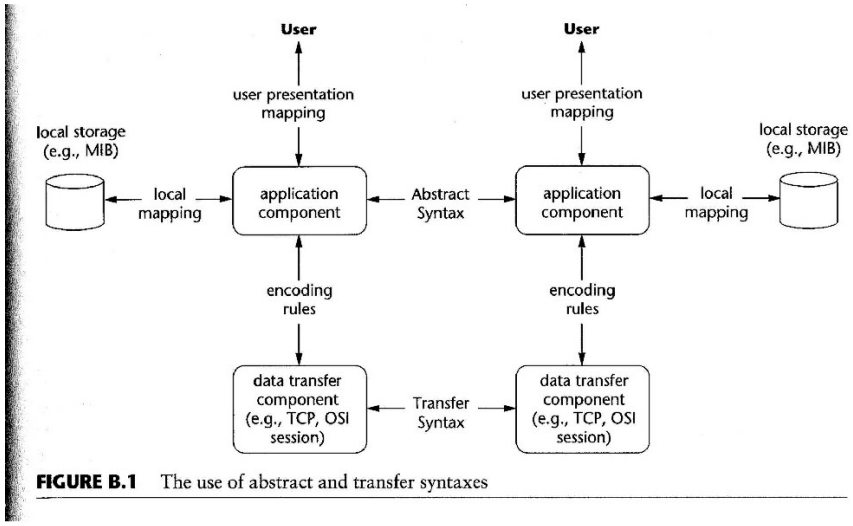


Figure 4: Sintassi astratta e di trasferimento

Ricapitolando:

- L'application component tratta un'informazione definita da una *sintassi astratta*
- Tale sintassi si chiama "astratta" perché specifica i dati indipendentemente dalla loro rappresentazione in un linguaggio di programmazione, in memoria o a livello di data transfer component
- ASN.1 è un *linguaggio formale* (un insieme di stringhe costruite sopra un alfabeto) che permette di costruire le sintassi astratte $->$ è uno strumento
- La sintassi astratta definita usando ASN.1 descrive i *dati trattati* dalle entità applicative e anche i *dati che possono essere scambiati* tra due applicazioni paritarie, perché i dati trattati sono un sovrainsieme di quelli che vengono scambiati. Quindi la sintassi astratta definisce anche le *PDU* che le applicazioni si scambiano
- Se la specifica della sintassi di un protocollo viene fatta in modo astratto, all'interno di un sistema che implementa un protocollo queste informazioni dovranno essere rappresentate e quindi *codificate* in vari modi: per presentarle all'utente, per memorizzarle in RAM, ecc. $->$ *sintassi concreta/locale*

- Mentre le *codifiche/mapping* precedenti sono una questione privata dell'implementatore, il mapping verso il data transfer component non lo è, quindi la forma di trasferimento deve essere specificata in modo *implementation independent* $->$ *sintassi di trasferimento*
- Le *regole di codifica* prescrivono come convertire in sequenze di ottetti (sintassi di trasferimento) i vari tipi di dato definiti dalla sintassi astratta. Bisogna quindi solo definire una sintassi astratta per un protocollo applicativo, poi le regole di codifica determinano automaticamente la sintassi di trasferimento
- Lo standard SMI di SNMP prescrive di utilizzare (un sottoinsieme del) le regole di codifiche chiamate *BER: Basic Encoding Rules*

5.2.2 Concetti di ASN.1

Il componente fondamentale di una specifica ASN.1 è il *modulo*. Una sintassi astratta può essere costituita da numerosi moduli che fanno riferimento l'un l'altro

Definizione di un modulo Un modulo ha questa forma:

```
<modulereference> DEFINITIONS ::=
    BEGIN
        ExportList
        ImportList
        AssignmentList
    END
```

- <*modulereference*> è il nome del modulo
- Il costrutto *ExportList* specifica la lista delle definizioni che vengono esportate dal modulo
- Il costrutto *ImportList* specifica la lista delle definizioni che si devono importare da altri moduli
- Il costrutto *AssignmentList* specifica la lista di definizioni che costituiscono il vero e proprio corpo del modulo $->$ Definizioni di:
  - *Tipi di dato* che sono necessari nella definizione di un protocollo applicativo

– *Valori* appartenenti a ben determinati tipi *Macro*

Le definizioni di tipi e valori hanno la forma:

```
<name> ::= <description>
```

Convenzioni lessicali:

- Separatore fra gli identificatori: uno o più blank
- Commenti: – all'inizio e alla fine
- Gli *identificatori lessicali* (OID): prima lettera minuscola (es. tcpConnState)
- I *tipi* e i *nomi di moduli*: prima lettera maiuscola (es. IpAddress)
- I *tipi built-int* (primitivi): solo lettere maiuscole (es. INTEGER)
- I *nomi di macro*: solo lettere maiuscole (es. OBJECT TYPE)

Tipi di Dato Astratti:

- *Semplici*: tipi i cuoi valori sono atomici in quanto non hanno componenti
- *Strutturati*: un valore di un tipo strutturato ha dei componenti e delle relazioni che "legano" fra loro questi componenti
- *Tagged*: sono sottotipi di altri, riconoscibili dai tipi "padre"
- *Metatipi*: quelli definiti con *CHOICE* o *ANY*

Ogni tipo ASN.1 (tranne metatipi) ha per definizione associato un *tag*: un tag è costituito da un *nome di classe* e un *valore intero non negativo*. Vi sono 4 possibili classi:

- *Universal*: generalmente utili, al di fuori di qualunque applicazione e non relativi ad un contesto (es. BOOLEAN)

- *Application-wide*: hanno validità solo all'interno di uno standard applicativo quindi sono definiti all'interno degli standard che definiscono il protocollo applicativo specifico (es. IpAddress)
- *Context-specific*: hanno validità in un contesto ristretto, oltre che all'interno di uno standard applicativo
- *Private*: sono come quelli Universal ma definiti da un vendor

Universal Tage Value	Type Name
1	BOOLEAN
2	INTEGER
3	BIT STRING
4	OCTET STRING
9	REAL
10	ENUMERATED
6	OBJECT IDENTIFIER
7	Object descriptor
18	NumericString
19	PrintableString
20	TeletexString
21	VideotexString
22	IA5String
25	GraphicString
26	VisibleString
27	GeneralString
5	NULL
8	EXTERNAL
23	UTCTime
24	GeneralizedTime
9-15	Reserved (ma il 9 è già usato)
28-	Reserved (dal 28 in su)
16	SEQUENCE e SEQUENCE OF
17	SET e SET OF

Figure 5: Valori dei tag universal

5.2.3 Tipi semplici

Sono tipi definiti semplicemente dall'insieme dei loro valori. Esempi: INTEGER, BOOLEAN, REAL, NULL, OBJECT IDENTIFIER, Object descriptor, ecc.

NULL È un tipo particolare che *contiene un solo valore* $->$ *null*

OBJECT IDENTIFIER (OID) È un tipo semplice i cui valori sono usati per identificare in modo univoco degli oggetti che ricorrono nei protocollo applicativi. È una sequenza di valori interi non negativi. Lo spazio degli object identifier è amministrato da ISO (International Organization for Standardization) e ITU (International Telecommunication Union) che se lo sono "spartito" in tre sottospazi (o sotto-alberi): il primo a ITU (il primo intero è 0), il secondo a ISO (il primo intero è 1) e il terzo a quei gruppi di standardizzazione in cui lavorano assieme ITU e ISO (il primo intero è 2).

Object descriptor Ha come valori delle semplici stringhe di testo che servono come annotazione da associare ad OBJECT IDENTIFIER (per aiutare l'uomo non la macchina)

5.2.4 Tipi strutturati

Sono tipi i cui valori sono costituiti da *componenti* e da una *relazione* fra di essi. Esistono 4 costruttori di tipi strutturati:

- SEQUENCE $->$ Per definire tipi i cui valori sono liste ordinate di elementi appartenenti a uno o più tipi
- SEQUENCE OF $->$ Come SEQUENCE ma elementi tutti dello stesso tipo
- SET $->$ insieme non ordinato di elementi appartenenti a uno o più tipi
- SET OF $->$ Come SET ma elementi tutti dello stesso tipo

5.2.5 Tipi tagged

Sono tipi definiti associando un nuovo tag ad un tipo già definito. Il tipo tagged è isomorfo al tipo "padre" a cui si applica il nuovo tag, ma è distinguibile da questo grazie al tag. Utile utilizzando i tag *application-wide* e *context-specific* per distinguere tipi rispetto ai tipi base.

Tagging esplicito e implicito Quando uso un tag in una definizione posso desiderare che il nuovo tag sia aggiuntivo (esplicito) a quello del tipo a cui si applica, oppure sostitutivo (implicito). ASN.1 fornisce la keyword *IMPLICIT* che permette a chi usa ASN.1 di specificare che il nuovo tag è sostitutivo di quello vecchio. Il tagging implicito produrrà delle codifiche

più compatte perché il vecchio tag non deve essere trasmesso, ma si ha una perdita di informazione

5.2.6 Metatipi

Sono tipi definiti con *CHOICE* o *ANY*: non sono veri e propri costruttori di tipo perché non hanno tag

CHOICE Permette di definire un tipo come un'alternativa fra un certo numero di tipi indicati nella definizione. Chi usa un tipo definito mediante CHOICE potrà usare un valore che appartenga a uno qualunque dei tipi elencati nella CHOICE.

ANY Viene utilizzato quando non si sa quale tipo verrà usato.
$->$ SEQUENCE è equivalente a SEQUENCE OF ANY

5.2.7 Sottotipi

Un sottotipo si definisce a partire dal tipo padre restringendo in qualche modo l'insieme dei valori originale (es. intervallo di INTEGER)
.....

5.3 Basic Endoding Rules (BER)

$->$ Regole per codificare i valori di un qualunque tipo di dato definito usando ASN.1

5.3.1 Struttura della codifica

Le BER codificano qualsiasi valore come una sequenza di ottetti (sintassi di trasferimento) che adotta una struttura chiamata:

$$TLV = Tag - Lunghezza - Valore$$

(Stallings usa il termine *Identifier* al posto di Tag). Il Valore può essere a sua volta una TLV:

- se il tag non è di tipi semplici ma di tipi strutturati
- se metto un Tag davanti a un valore perché sia distinguibile da altri (tipi tagged)

Ci sono due modi di codificare la Lughezza ma SNMP ne adotta uno solo, cioè quello detto a *lunghezza definita*:

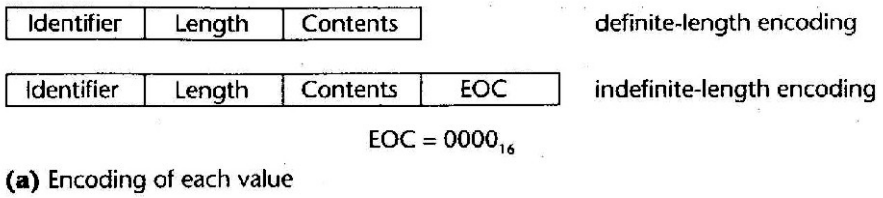


Figure 6: BER - TLV

5.3.2 Codifica del Tag/Identifier

Il primo campo della TLV ha maggiore informazione rispetto al semplice Tag di ASN.1:

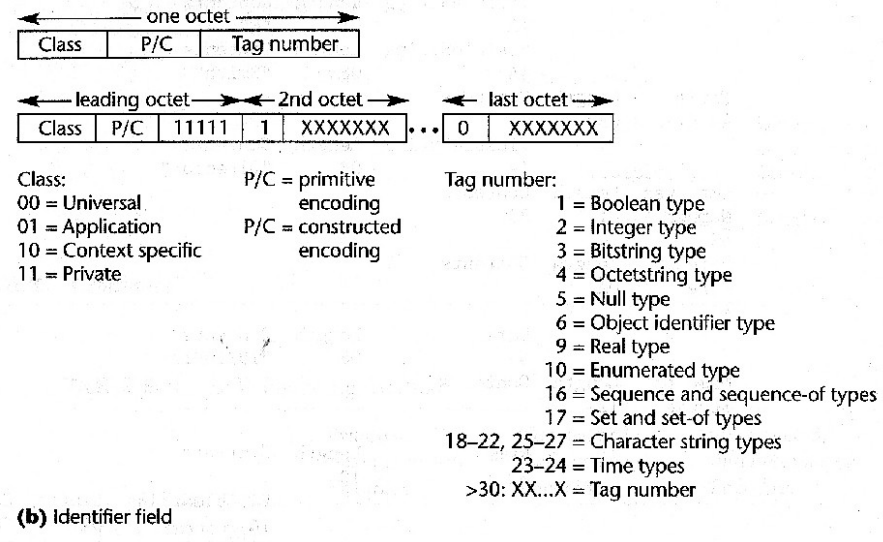


Figure 7: BER - Tag/Identifier

- *Class*: I primi 2 bit codificano la classe del tag
- *P/C*: Il terzo dice se la codifica (non la natura, può essere diversa) del valore è *Primitiva* (tipi semplici) o *Costruita* (tipi strutturati, TLV

annidati)

Tag Number Per i valori piccoli si vuole usare pochi bit e per quelli grandi se ne useranno di più. Quindi abbiamo 2 forme di rappresentazione:

1. $Valore \leq 30$: Si usano i 5 bit rimanenti dopo *Class* e *P/C*
2. $Valore > 30$: I 5 bit rimanenti si mettono a 1 e si usano gli ottetti che seguono quanto basta. Il primo bit (più significativo) di un ottetto che è seguito da un altro ottetto è messo a 1, mentre se l'ottetto è l'ultimo (o l'unico) viene messo a $0 - >$ se tag$\leq$127 si usa solo il secondo byte

5.3.3 Codifica della Lunghezza

Anche qui codifica a lunghezza variabile:

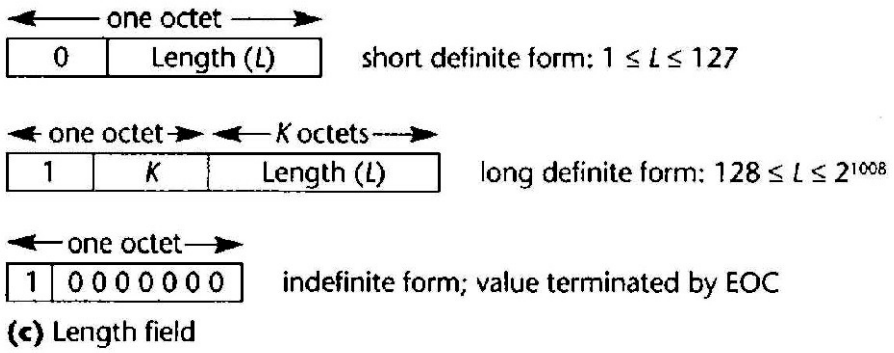


Figure 8: BER - Length

1. $Valore \leq 127$: Si usa solo il primo ottetto, il cui primo bit vale 0
2. $Valore > 127$: Il primo bit si mette a 1, i successivi 7 bit dicono di quanti ottetti è fatta la lunghezza, che quindi può occupare fino a 126 ottetti (127 è riservato per future estensioni) $- >$ può valere fino a 2^{1008}

5.3.4 Codifica dei Valori

- Valori *primitivi* codificati sempre con regola che cerca di usare meno byte

- I tipi *costruiti* di codificano in modo ricorsivo $->$ Se dal tag so che il tipo è SEQUENCE, allora il valore sarà una sequenza di triple TLV, e la lunghezza in byte di tutta la SEQUENCE è data dalla Length esterna
- *Stringhe* possono essere codificate sia in forma primitiva che costruita. In SNMP solo primitiva
- Se il tipo è *tagged*, quindi nella sintassi è stato aggiunto un tag, una tripla TLV avvolgerà la tripla del valore taggato. Per evitare ciò si può usare la parola chiave *IMPLICIT* per eliminare tag e lunghezza interni, perdendo però informazione $->$ per questo bit P/C fondamentale
- Ciascun tipo primitivo ha una sua regola di codifica del valore

Codifica dei valori INTEGER

- Sono codificati in complemento a 2 [2]
- Un ottetto può rappresentare un numero compreso tra -128 e $+127$
- Uso più ottetti per numeri maggiori di $127 ->$ so quanti sono grazie al campo *Length*

Codifica dei valori BOOLEAN Il valore FALSE viene codificato con un byte fatto tutto di zeri. Il valore TRUE con un qualunque byte diverso dal FALSE. La lunghezza è comunque 1.

Codifica del valore NULL Ha un unico valore: *null*. Quindi non è necessario trasmettere il valore, che avrà lunghezza 0. Il tutto occupa solo 2 ottetti:

$$0500_{16}$$

Codifica dei valori OCTET STRING Codifica primitiva delle stringhe in SNMP, quindi non possono essere rappresentate da concatenazione di sotto-stringhe. Le stringhe di caratteri sono sottotipi di OCTET STRING e sono codificate direttamente con la stringa che rappresenta quel carattere (codice ASCII)

[2] È il metodo più diffuso per la rappresentazione dei numeri con segno: tutti i numeri che cominciano con "1" (bit più significativo) sono numeri binari negativi, mentre tutti i numeri che cominciano con "0" sono numeri binari positivi. Un numero binario positivo si può rendere negativo invertendone i bit (complementare) e sommando 1 al valore risultante.

Codifica dei valori OBJECT IDENTIFIER Un OID è una sequenza di interi senza segno, ma in ASN.1 è un tipo di dato primitivo. I primi due interi sono amministrati, cioè possono avere solo valori molto piccoli e ben definiti (0, 1, 2). Il secondo intero, anche esso amministrato, non può superare 39. Allora le BER combinano i primi due identificatori X e Y in un unico identificatore dato dalla formula:

$$Z = (X * 40) + Y$$

- Quindi $Z \leq 119$ e si può usare un solo byte
- Ogni intero è rappresentato in base 2 da una successione di gruppi di 7 bit
- Il primo bit mi dice se segue un altro byte (bit a 1) oppure se è l'ultimo (bit a 0) di quell'intero
- Il campo *Length* mi dice quando ho finito

5.4 SMI - La struttura della MIB

Avendo visto i dettagli di ASN.1 e BER, possiamo ora vedere come SMI definisce la struttura della MIB e la sintassi degli oggetti

- Database di tipo gerarchico, strutturato ad albero
- Collezione di oggetti che rappresentano le risorse da gestire
- Gli oggetti appartengono a un tipo, cioè sono istanza di un object type
- Gli oggetti hanno un identificatore univoco (OID), ossia un valore che appartiene al tipo OBJECT IDENTIFIER
- Nella MIB vengono definiti gli object type, mentre le istanze sono presenti a run time negli agent

OID

- Un OID è una sequenza di interi senza segno che rappresenta un cammino nell'albero degli OID
- Il sottoalbero che identifica Internet è il seguente:

```
internet OBJECT IDENTIFIER ::= { iso (1) org (3) dod(6)
    1 }
```

- Questa è una definizione in ASN.1 per un valore di tipo OBJECT IDENTIFIER. Un'altra notazione è quella puntata. Il valore di *internet* è quindi spesso scritto:

  1.3.6.1

- Al di sotto di *internet* ci sono 4 nodi:

  1. *directory* $->$ Standard sul Directory
  2. *mgmt* $->$ Standard che riguardano la gestione SNMP
  3. *experimental* $->$ OID per esperimenti di Internet
  4. *private* $->$ per OID introdotti da aziende private. Serviranno per implementare agenti SNMP con MIB standard

- Gli OID che servono per le MIB sono contenuti nell'albero *mgmt*

- Sotto *mgmt* troviamo *mib-2* che è la MIB minima che qualunque agent deve implementare, la quale descrive le risorse necessarie alla gestione stessa.

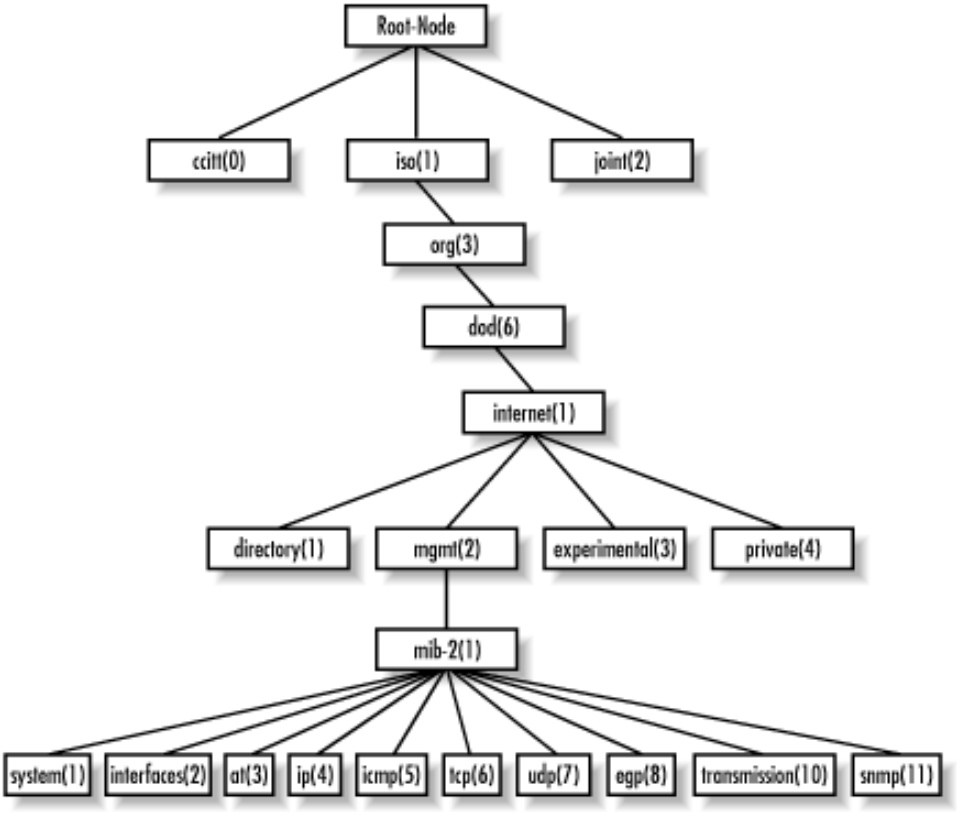


Figure 9: OID tree

5.5 SMI - La sintassi degli oggetti

Ogni oggetto usato da SNMP viene definito in modo formale, definendo in primo luogo il *tipo del valore* che contiene. Per definire il tipo del valore, SMI usa un sottoinsieme di ASN.1

5.5.1 Tipi Universali

Solo i sequenti tipi della classe UNIVERSAL possono essere usati:

- INTEGER
- OCTET STRING
- NULL
- OBJECT IDENTIFIER

- SEQUENCE, SEQUENCE OF

5.5.2 Tipi Application-wide

Il tag APPLICATION serve per definire tipi che sono rilevanti all'interno di un certo standard applicativo. In SNMP abbiamo:

- *IpAddress*: è un tipo definito in SMI come una stringa di byte di lunghezza 4:

```
IpAddress ::= [APPLICATION 0] IMPLICIT OCTET STRING (
    SIZE (4)) -- in network-byte order
```

- *NetworkAddress*: è un tipo per descrivere qualunque indirizzo di livello rete, anche non di Internet, quindi viene definito con una *CHOICE*; al momento però l'unica alternativa è costituita dall'indirizzo IP:

```
NetworkAddress ::= CHOICE { internet IpAddress }
```

- *Counter*: è un numero intero senza segno a 32 bit che *può solo aumentare* il suo valore nel tempo, e *quando dovrebbe superare il massimo ritorna invece a zero* (es. contare eventi come pacchetti ricevuti, byte spediti, errori, ecc. $->$ *ifOutOctets*):

```
Counter ::= [APPLICATION 1] IMPLICIT INTEGER (0 ..
    4294967295)
```

- *Gauge*: è un numero intero senza segno a 32 bit che *può aumentare o diminuire* il suo valore, *senza mai superare il suo valore massimo* (es. indicare il valore corrente di qualche entità come il numero di pacchetti presenti in una coda) $->$ *ifSpeed*
- *TimeTicks*: è un numero intero senza segno a 32-bit che misura su un apparato, in centesimi di secondo, il tempo trascorso dal boot (uptime) $->$ *sysUpTime*
- Le soglie non sono definite in SMI ma saranno poi introdotte nella *Remot Monitoring* MIB (RMON)

5.5.3 La definizione degli Oggetti

- Gli oggetti hanno un nome (OID), un tipo e un valore
- Un *object type* è la definizione di un particolare genere di oggetti, ossia una descrizione dotata di sintassi.
- Un *object instance* è un'istanza di quell'*object type*. Sono gli agent che creano le istanze degli object type.
- Per definire gli object type si usa la macro presente nell'RFC 1212

Macro OBJECT-TYPE La macro prende alcuni parametri, fra i quali i più significativi:

- SYNTAX: è la sintassi astratta a cui appartiene il valore contenuto nelle istanze dell'object type. Può essere
  - *semplice* (*SimpleSyntax*), ossia usare i tipi universali visti nel paragrafo 5.5.1
  - *application-wide* (*ApplicationSyntax*), ossia usare i tipi visti nel paragrafo 5.5.2
- ACCESS: specifica il modo con cui si può accedere al valore dell'istanza dell'object type. Può essere:
  - *read-only*
  - *read-write*
  - *write-only*
  - *not-accessible* $->$ Viene usato per i nodi intermedi dell'albero che rappresentano gruppi logici (es. *tcp*)
- STATUS: è il parametro mediante il quale si specifica quanto l'implementazione deve supportare l'object type. Può essere:
  - *mandatory*
  - *optional*
  - *deprecated*
  - *obsolete*
- DESCRIPTION: contiene la descrizione in linguaggio naturale della semantica dell'object type. È essenziale per il manager umano

- INDEX: è necessario nella definizione delle tabelle. Viene aggiunto al prefisso dell'OID per determinare la riga di una tabella

5.5.4 La definizione delle tabelle

- In SMI: Tabelle bidimensionali i cui elementi siano costituiti da tipi semplici
- La generica riga (entry) di una tabella ha una struttura che viene definita mediante il costruttore SEQUENCE
- La tabella è quindi definita mediante il costruttore SEQUENCE OF, che indica la ripetizione di un un numero arbitrario di righe tutte con la stessa struttura definita precedentemente
- Tabella e riga dichiarati *not-accessible*. Si potrà accedere solo ai singoli elementi delle righe
- L'ordine delle colonne dipende all'ordine con cui appaiono nella definizione della SEQUENCE, e sono quindi identificate da un intero positivo che parte da 1 per la prima colonna, e viene aggiunto come suffisso all'OID del padre, in questo caso l'oggetto che rappresenta la riga. In base a questo, possiamo utilizzare l'OID opportuno per identificare la colonna di cui vogliamo un valore
- Per identificare poi anche la riga, bisogna utilizzare un opportuno *indice* da aggiungere anch'esso come suffisso al tutto. Nella maggior parte dei casi, una parte dei dati contenuti nella riga costituisce l'indice da utilizzare.

Esempio: tcpConnTable Usiamo come esempio il caso della tabella delle connessioni TCP. Definiamo un object type *tcpConnTable* che avrà come OID:

1.3.6.1.2.1.6.13

Per ogni connessione vogliamo che siano presenti queste informazioni (quindi colonne):

- tcpConnState $->$ TCP definisce 11 stati; ne viene aggiunto un 12esimo, *deleteTCB*: quando il manager setta questo stato, l'entità TCP deve distruggere la connessione ($abort$) $->$ azione eseguita mediante settaggio di un valore

- tcpConnLocalAddress
- tcpConnLocalPort
- tcpConnRemAddress
- tcpConnRemPort

La definizione della tabella nella MIB *mib-2* è la seguente:

```
tcpConnTable OBJECT-TYPE
    SYNTAX SEQUENCE OF TcpConnEntry
    ACCESS not accessible
    STATUS mandatory
    DESCRIPTION
        "A table containing TCP connection-specific information
            ."
    ::= { tcp 13 }


TcpConnEntry ::= SEQUENCE { tcpConnState     INTEGER,
                            tcpConnLocalAddress      IpAddress,
                            tcpConnLocalPort INTEGER (0..65535),
                            tcpConnRemAddress    IpAddress,
                            tcpConnRemPort  INTEGER (0..65535)
}


tcpConnEntry OBJECT-TYPE
    SYNTAX TcpConnEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
        " Information about a particular current TCP
          connection.  An object of this type is transient,
          in that it ceases to exist when (or soon after)
          the connection makes the transition to the CLOSED
          state. "
    INDEX { tcpConnLocalAddress, tcpConnLocalPort,
            tcpConnRemAddress, tcpConnRemPort }
    ::= { tcpConnTable 1}
```

Viene quindi definito:

- Innanzitutto l'object type *tcpConnTable* che rappresenta la tabella. Essa è una SEQUENCE OF di *TcpConnEntry* ossia le righe, e il suo OID sarà 13 posto come suffisso all'OID del gruppo *tcp*

- Viene poi definito il tipo *TcpConnEntry* usato in precedenza, come SEQUENCE delle varie informazioni (quindi le colonne) ancora da definire che abbiamo descritto prima
- Viene quindi definito l'object type di questa generica riga della tabella, che avrà come tipo *TcpConnEntry* appena definito. Viene assegnato anche qui un nuovo OID, direttamente "sotto" a *tcpConnTable*. Da notare che qui viene specificato l'INDEX contenente la quadrupla che identifica una connessione TCP. Questo è quindi l'indice che dovrà essere usato (posto come suffisso all'OID della colonna) ogni qualvolta siamo interessati ad individuare una precisa riga della tabella (elementi della stessa colonna hanno lo stesso prefisso).

Bisogna ora definire i 5 object type che compongono la riga. Per brevità, vediamo solo *tcpConnState*:

```
tcpConnState OBJECT-TYPE
    SYNTAX INTEGER { closed(1), listen(2), synSent(3),
        synReceived(4),
                     established(5), finWait1(6), finWait2(7),
                         closeWait(8),
                     lastAck(9), closing(10), timeWait(11),
                         deleteTCB(12) }
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
        "The state of this TCP connection..."
    ::= { tcpConnEntry 1 }
```

Object type di questo tipo possono essere istanziati varie volte (tante quante sono le connessioni aperte) $->$ Vengono detti *colonnari*, per distinguerli dagli *scalari*, ossia quelli per cui esiste una sola istanza.

INDEX

- L'INDEX specifica quali sono gli elementi che compongono la chiave per accedere ad un preciso elemento della tabella.
- Quando vi sono più elementi, la parte finale dell'OID sarà fatta di tante sottoparti quanti sono gli elementi della chiave.
- Se un elemento non è un intero non negativo ma ha un altro tipo, la SMI definisce una regola di *mapping* per generare sequenze di interi

non negativi in base al valore di questo tipo.

- Nel caso in cui abbiamo un OCTET STRING o un altro OBJECT IDENTIFIER di lunghezza variabile, prima degli interi non negativi generati dal contenuto degli ottetti, si inserisce la lunghezza della stringa/OID.
- Se un *object type* non è un colonnare ma uno scalare, si aggiunge 0 all'OID per ottenere il nome della sua istanza.

6 MIB Standard

Perché una MIB sia funzionale alla gestione, gli oggetti usati per rappresentare una certa risorsa devono essere gli stessi in ogni sistema dello stesso tipo. Gli organismi di standardizzazione di Internet hanno sviluppato numerose MIB *standard*, perché molti sono i sistemi che il mercato richiede che siano gestiti in modo indipendente dal venditore.

6.1 MIB-II

Viene definita nel 1991 nel RFC 1213. Lo scopo di questa MIB è di permettere la gestione delle risorse che devono essere presenti su un agente SNMP, indipendentemente dal sistema. È quindi la MIB di base obbligatoria per ogni genere di agente SNMP.

6.1.1 Criteri di Progetto della MIB-II

I criteri adottati, esplicitamnete citati in RFC 1213, sono i seguenti:

- *Un oggetto deve essere necessario per fault o per configuration management*: MIB minimale, quindi obiettivi molto limitati
- *Sono permessi solo oggetti di controllo "deboli"*: Per deboli si intende oggetti che non possano causare gravi danni se mal gestiti. Ciò è dovuto alla debolezza di SNMP dal punto di vista della sicurezza
- *Ci devono essere prove certe di utilità e di uso corrente*
- *Nessun limite sul numero di oggetti*: Quando fu sviluppata la MIB-I c'erano solo circa 100 oggetti
- *Niente oggetti ridondanti*: Nessun oggetto incluso deve poter essere ricavabile da altri
- *Oggetti che sono specifici solo di alcune implementazioni non devono essere inclusi*
- *Occorre evitare di strumentare eccessivamente sezioni critiche di codice*: Strumentare il codice significa inserire al suo interno elementi di intervento in lettura o in scrittura, o aggiungere software di misura (contatori o gauge), La strumentazione del codice rallenta il codice perché è necessario garantire mutua esclusione

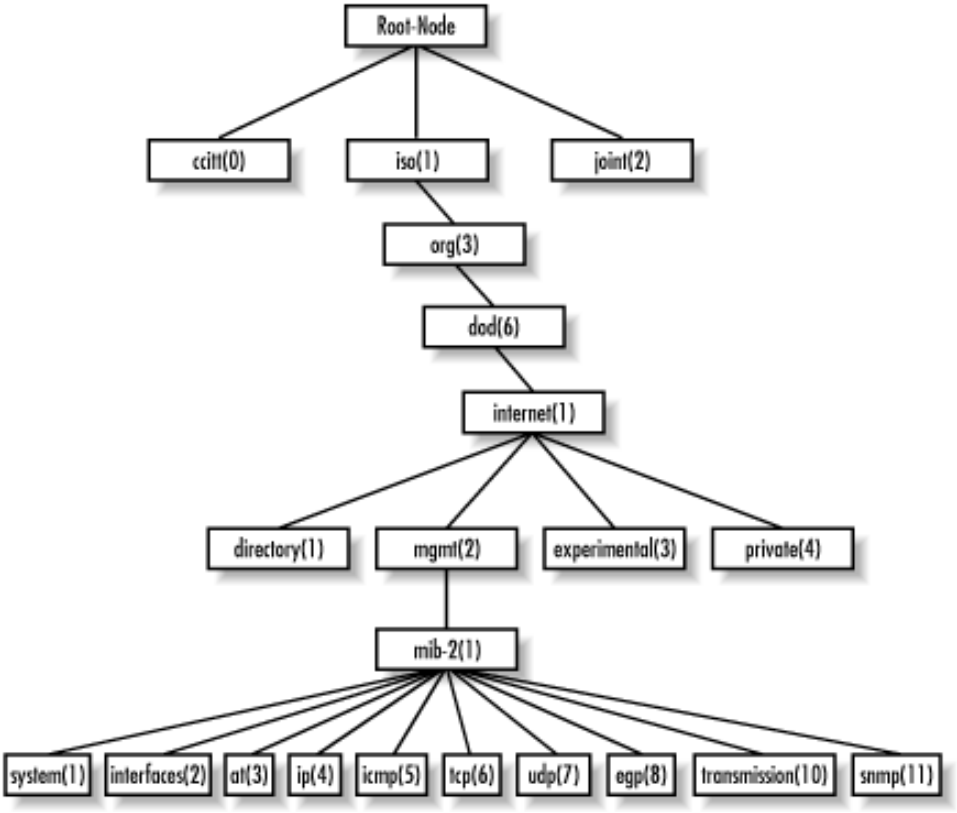


Figure 10: OID tree

Gruppi presenti nella MIB-II Le altre MIB relative a risorse più specifiche di quelle minimali saranno introdotto come *estensione* di MIB-II, quindi all'interno del sottoalbero *iso.org.dod.internet.mgmt.mib2*, come quindi fratelli dei nodi che descriviamo nel seguito.

I gruppi di MIB-II sono:

1. *system*: informazioni generali
2. *interfaces*: descrive le interfacce di rete presenti
3. *at*: (address translation) deprecata
4. *ip*: include la routing table
5. *icmp*
6. *tcp*: include tcp connection table

7. *udp*
8. *egp*: Exterior Gateway Protocol (BGP). Statistiche e neighbor table
9. *transmission*: descrive gli schemi di trasmissioni e i protocolli di accesso di ciascuna interfaccia. Di base non ci sono oggetti, ma altre MIB specifiche sono state definite usando questo sottoalbero
10. *snmp*

6.1.2 Il gruppo system

- Tutti i nomi degli *object type* iniziano con il prefisso *sys*
- Tutti scalari
- *sysServices* si interpreta come una maschera a 7 bit, in cui ogni bit settato indica che il corrispondente livello dello stack ISO-OSI è implementato sulla macchina. Il valore numerico rappresentato nella istanza di questo object type è:

$$sysServices = \sum 2^{L-1}, L \in S$$

  Quindi un dispositivo di rete avrà valore 4 (2^{3-1}), mentre un host offre i livelli 4 e 7 per cui *sysServices* vale 72 ($2^{4-1} + 2^{7-1}$)

- *sysUpTime* è di tipo *TimeTicks* contiene il tempo passato dal boot dell'agent. Permette al manager di rilevare che l'agent ha resettato e quindi tutti gli oggetti che hanno come sintassi *Counter* sono altresì stati resettati
- *sysObjectID* contiene un OID che identifica il costruttore e il modello del sistema. Permette al manager di capire la natura del sistema gestito

6.1.3 Il gruppo interfaces

- Contiene informazioni generiche sulle interfacce fisiche di rete del sistema, incluse informazioni di configurazione e statistiche sul traffico.
- Il gruppo è costituito da uno scalare *ifNumber* che contiene il numero di interfacce, e da una tabella *ifTable*

- In *ifTable* vi è una riga per ogni interfaccia. L'indice della tabella è *ifIndex* che è un intero, progressivo fra 1 e *ifNumber* inclusi
- Per associare ogni interfaccia al suo *ifIndex* possiamo usare gli indirizzi fisici delle interfacce (*ifPhysAddress* oppure gli indirizzi di rete usando l'object type *ip.ipAddrTable.ipAddrEntry.ipAdEntIfIndex*)
- *ifAdminStatus* è *read-write* e specifica lo stato dell'interfaccia che il manager ha richiesto settando il valore di questo oggetto (*up:1 ; down:2*)
- *ifOperStatus* indica invece lo stato in cui si trova l'interfaccia $->$ Se il manager setta *ifAdminStatus* a *up* e invece l'interfaccia va in *ifOperStatus* uguale a *down*, allora vuol dire che l'interfaccia è guasta. Quindi è sufficiente confrontare questi due valori per diagnosticare un guasto dell'interfaccia
- *ifLastChange* riporta il valore che *sysUpTime* aveva quando è avvenuto l'ultimo cambiamento di stato dell'interfaccia
- *ifSpeed* è un *gauge* che stima la velocità dell'interfaccia in bit/s

6.1.4 Il gruppo ip

- *ipAddrTable* è la tabella che descrive il mapping fra indirizzi logici e indirizzi fisici assegnati alla scheda
- *ipRouteTable* contiene la tabella di routing

6.1.5 Il gruppo tcp

Importanti:

- La specifica dell'algoritmo e dei parametri di ritrasmissione adottati
- Statistiche sulle connessioni e sugli errori
- La tabella delle connessioni

7 SNMP: Simple Newtork Management Protocol

Protocollo definito nell'RFC 1157 del 1990

7.1 Concetti fondamentali

- Le operazioni del protocollo
- Il concetto di community
- Come si identificano le istanze degli object type
- Ordinamento imposto dalla convenzione per identificare gli oggetti

7.1.1 Operazioni supportate

- *get*: *GetRequest*, *GetNextRequest* (agent risponde con *GetResponse*)
- *set*: *SetRequest*
- *trap*: *Trap*

7.1.2 Community e Community Name

Il sistema di gestione di applicazioni distribuite è a sua volta una applicazione distribuita, le cui entità sono i manager e gli agent. Bisogna definire le relazioni fra agent e uno o più manager in termini di autenticazione e controllo degli accessi. Gli agent devono controllare l'uso che i manager possono fare della MIB locale, mentre i manager devono controllare quali agent possono provocare il loro intervento inviando delle trap.

- *Servizio di autenticazione*: l'agent deve poter limitare l'accesso solo a certi manager ben identificati
- *Politica di accesso*: l'agent deve poter dare differenti privilegi di accesso a diversi manager autenticati
- *Proxy*: l'agent deve poter dare accesso alle funzionalità di proxy solo a certi manager autenticati

Community SNMPv1 fornisce agli agenti lo strumento delle community. Una *community* è una *relazione* fra un agent SNMP e un *insieme di manager* SNMP che definisce *autenticazione*, *controllo degli accessi* e *caratteristiche di proxy*. Si può definire più di una community e ognuna di esse è identificata da un *community name*, univoco nell'ambito di ogni singolo agent (concetto locale all'agent).

Servizio di Autenticazione Un manager deve *autenticarsi*. La forma di autenticazione è la conoscenza del *community name*. Il vero problema è che i messaggi del protocollo *viaggiano in chiaro* sulla rete e quindi il *community name* è facilmente *sniffabile*. Inoltre, usando UDP, non c'è protezione contro il *replay attack*, cioè la ritrasmissione di una PDU catturata in precedenza.

Politica di accesso L'agent può limitare l'accesso alla sua MIB ad un particolare insieme di manager. Ci sono 2 aspetti che definiscono una politica di accesso:

- *SNMP MIB view*: Si può specificare solo una parte della MIB che può essere gestita, e gli oggetti visti possono non appartenere allo stesso sottoalbero
- *SNMP MIB access mode*: Definita la MIB view, gli access mode sono solo 2:
  - *READ-ONLY*
  - *READ-WRITE*

La combinazione di MIB view e di access mode è detta *community profile*. La combinazione di una community e di una particolare community profile definisce una *SNMP access policy*:

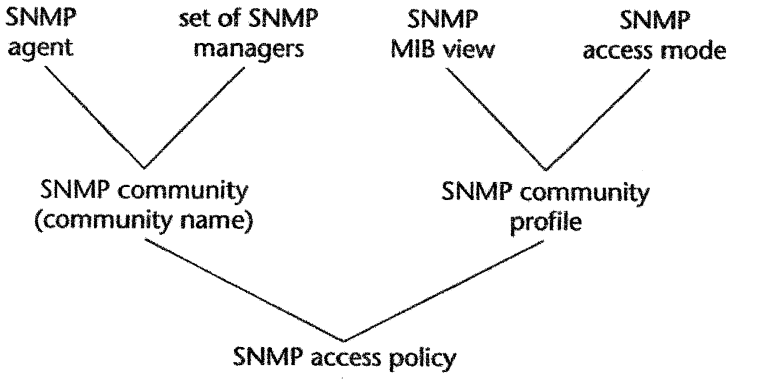


FIGURE 7.1 Administrative concepts

Figure 11: SNMP access policy

7.1.3 Identificazione delle istanze

- Ogni oggetto è identificato da un OID univoco
- Una *istanza* di un object type ha un OID che come *prefisso* ha l'OID dell'object type; quindi le istanze sono le foglie del sottoalbero.
- Nelle operazioni di SNMP si può accedere solo a istanze di oggetti
- Inoltre si può accedere solo a object type la cui sintassi sia di tipo semplice dell'ASN.1, quindi non direttamente alle tabelle (introdotte nella MIB solo per leggibilità) ma solo ai valori contenuti in esse, ossia agli oggetti colonnari definiti nella definizione della tabella
- Gli oggetti colonnari possono avere un numero di istanze maggiore di 1, quindi serve una regola per generare gli OID di ciascuna istanza
- Gli oggetti scalari invece hanno una sola istanza. Per ottenere il suo OID basta aggiungere 0 all'OID dell'object type

Oggetti colonnari L'OID delle istanze dei colonnari si ottiene aggiungendo all'OID dell'object type della colonna una successione di interi non negativi che viene derivata dal valore dei campi che agiscono come *INDEX* delle righe. Se riprendiamo l'esemio della tabella delle connesioni TCP, ogni istanza di ciascuna colonna, ad esempio della colonna *tcpConnState*, ha la forma:

```
x.i.(tcpConnLocalAddress).(tcpConnLocalPort).(tcpConnRemAddress)
    .(tcpConnRemPort)
```

Dove:

- x = 1.3.6.1.2.1.6.13.1, cioè l'OID di tcpConnEntry
- i = indice della colonna nella tabella, ad es. per tcpConnState vale 1

7.1.4 Ordinamento lessicografico

Gli OID sono una sequenza di interi non negativi, che riflette la struttura ad albero dello spazio degli OID. Un OID *precede* un altro se ne è un prefisso oppure se esiste un valore k per cui le prime (k-1) etichette sono uguali, la k-esima è diversa e determina l'ordinamento. Su questa nozione di ordinamento è basata l'operazione *get-next*

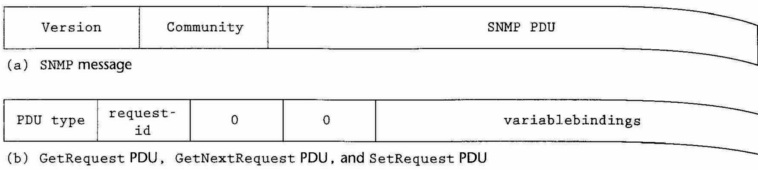
7.2 Specifica del Protocollo

7.2.1 Formati SNMPv1

Ogni messaggio è composto da:

- *Version*
- *Community*
- SNMP PDU $->$ chiamata così solo questa parte invece che l'intero pacchetto per lasciarla aperta ad eventuali estensioni a livello di sicurezza (due parti iniziali avrebbero potuto fornire elementi per decifrare la terza parte)

Le PDU di *GetRequest*, *GetNextRequest* e *SetRequest* hanno tutte lo stesso formato e si contraddistinguono solo dal *PDUtype*



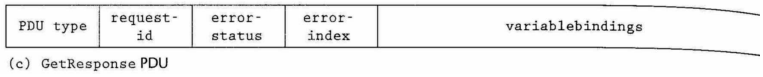


Figure 12: SNMP PDU

La *GetResponse* PDU porta con sè non solo i varbind richiesti nel caso di Get ma anche i varbind dopo la Set, che sia andata bene o male

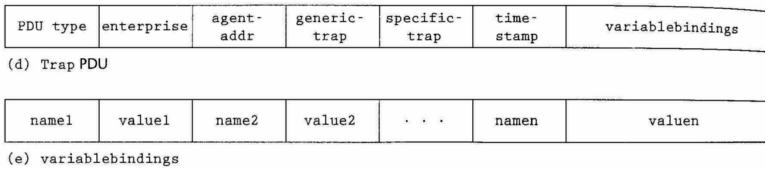


Figure 13: Trap PDU e formato varbind

Vediamo meglio i vari campi:

- *version*: la versione 1 è indicata da 0
- *community*: il nome della community
- *PDU type*: sono di tipo *context-specific* e un tag permette di distinguerle $->$ *GetRequest* è 0, ecc.
- *request-id*: un INTEGER usato per distinguere una invocazione dalle altre; nella risposta deve essere uguale alla richiesta, altrimenti la risposta viene scartata (viene scartata anche dopo un timeout settato dal manager)
- *error-status*: un INTEGER che codifica i vari tipi di errore (*noError* è 0)
- *error-index*: se *error-status* non è zero, specifica la posizione del VarBind che ha generato l'errore
- *variablebindings*: una lista di *VarBind*; è quindi una SEQUENCE OF SEQUENCE $\big(\langle name\rangle, \langle value\rangle\big)$ $->$ In una *GetRequest* si conoscono solo gli OID e non il loro valore. Quindi nella coppia ci sarà *null* come *value*
- *enterprise*: un OID che identifica il tipo di apparato, o di un suo componente, che ha generato la trap. Dato che l'apparato è certamente

fatto da un vendor, questo OID sta nell'albero *enterprises* ed è lo stesso memorizzato in *sysObjectID*

- *agent-addr*: l'indirizzo IP dell'agent dell'apparato che ha generato la trap
- *generic-trap*: un INTEGER che indica il tipo di trap (7 tipi)
- *specific-trap*: un INTEGER che indica la trap specifica all'interno del tipo generico
- *time-stamp*: è un *TimeTicks* che indica il momento in cui la trap si è verificata

7.2.2 GetRequest

- I valori associati agli OID sono settati a *null*
- L'agent risponde con una *GetResponse* che contiene lo stesso *request-id* corrispondente
- In SNMPv1 la *GetRequest* è atomica, ossia l'agent risponde con tutti i valori richiesti oppure con nessuno
- Poche condizioni di errore:
  - L'OID non esiste oppure non è semplice: *noSuchName* e *error-index* indica la posizione del primo VarBind che ha causato l'errore
  - Le risorse necessarie a generare la risposta eccedono i limiti locali (memoria o tempo di CPU) dell'agent: *tooBig*
  - Altro: *genErr*

7.2.3 GetNextRequest

- Stessi parametri della *GetRequest*
- Per ogni OID viene restituito il valore dell'OID della *prima foglia successiva* all'OID richiesto
- Se non conosco il valore dei campi di un *INDEX* posso usare la *GetNextRequest* per "scoprire" la prima foglia del sotto-albero di un object-type colonnare. $->$ Nel VarBind restituito, trovo il valore dei campi indice codificato all'interno dell'OID

- Nel caso in cui tutti i campi tranne uno siano indice della tabella, è sufficiente scandire una sola colonna per ricavare il valore di tutti i campi! $->$ Come nella tabella delle connessioni TCP, usando come colonnare *tcpConnState*

7.2.4 SetRequest

- Questa volta il campo *value* dei VarBind è significativo
- Agent risponde con *GetResponse* con lo stesso *request-id* della richiesta
- In caso di fallimento, non viene restituito alcun VarBind e viene segnalato l'errore con i campi opportuni
- L'operazione è atomica: se fallisce un assegnamento falliscono tutti
- Oltre agli errori della *GetRequest* anche *badValue* quando un VarBind è incosistente

Aggiunta di una riga ad una tabella Bisogna assegnare *coerentemente* i valori degli indici, usando OID che contengono già i valori degli indici che stiamo inserendo. Esempio:

```
SetRequest(
    (ipRouteDest.11.3.3.12=11.3.3.12),
    (ipRouteMetric.11.3.3.12=9),
    (ipRouteNextHop.11.3.3.12=91.0.0.5)
)
```

Aggiunge alla *ipRouteTable* una nuova riga con destinazione (indice) 11.3.3.12, metrica 9 e next hop 91.0.0.5

Cancellazione di una riga Solo se nella tabella è presente un opportuno campo il cui valore indichi che la corrispondente riga sia invalida. Ad esempio nella *ipRouteTable* c'è il campo *ipRouteType* che può essere settato a *invalid*. L'agent può decidere se effettuare anche la cancellazione fisica oppure solo quella logica.

Eseguire una azione SNMP non prevedere un comando specifico per richiedere l'esecuzione di azioni. Si può usare la *SetRequest* su particolari

oggetti per chiedere all'agent di eseguire un'azione. Ad esempio l'oggetto *ifAdminStatus* per settare lo stato di un'interfaccia

7.2.5 Trap

Notifica asincrona di un evento significativo insieme al valore di alcuni oggetti significativi

Il campo *generic-trap* ha solo 7 varianti:

- *cold-start*: l'agent si sta reinizializzando e lo stato è alterato/resettato
- *warm-start*: reinizializzazione a caldo senza perdita di stato
- *linkDown*: fallimento di un'interfaccia di rete. Il primo elemento dei *variablebindigs* è costituito dal nome e dal valore della istanza di *ifIndex* dell'interfaccia in causa. Quindi identifica la riga della tabella che descrive l'interfaccia
- *linkUp*
- *authenticationFailure*
- *egpNeighborLoss*
- *enterpriseSPecific*: evento non standard, specifico del vendor

A differenza degli altri comandi, la *Trap* non richiede alcuna risposta. Questo perché è stata pensata per situazioni in cui non si ha comunque tempo per aspettare la risposta e ripetere l'invio in caso contrario

7.3 Supporto a livello di Transport

7.3.1 Servizio di Trasporto connection-less

- SNMP è stato progettato contando su un servizio di trasporto connection-less: UDP
- Lo Standard prevede che agent ascolti sulla porta 161 e l'agent sulla porta 162
- UDP non garantisce l'affidabilità, quindi deve essere implementata dall'applicazione

- Lo Standard non prescrive molto ma le scelte sono lasciate agli implementatori
- Il *request-id* permette di risolvere il problema delle risposte duplicate
- Nel caso di una mancata risposta ad una *Set* si dovrebbe verificare con una *Get* se la modifica sia stata eseguita o meno
- Le *Trap* non sono riscontrate; il manager deve effettuare del polling con frequenza opportuna per verificare di non aver perso una trap (*snmpOutTraps*)

7.3.2 Servizio di Trasporto connection-oriented

Non viene utilizzata perché una connessione:

- Consuma risorse
- Consuma banda per apertura e chiusura
- Consuma banda per gli ack

7.4 Il gruppo SNMP di MIB-II

- Permette una gestione, molto limitata, della implementazione di SNMP ("gestire il sistema di gestione")
- Non ha object type specifici per il sistema gestito ma solo quelli necessari per gestire SNMP
- Tutti gli object type sono *read-only* tranne *snmpEnableAuthenTraps* che abilita o disabilita l'invio di trap di autenticazione al manager
- MIB-II è quindi una MIB orientata al *monitoring* piuttosto che al *control*

7.5 Aspetti pratici

7.5.1 Oggetti non supportati

Potrebbe capitare che non venga implementata la *semantica* di alcuni oggetti, per ridurre i costi o perché il sistema gestito non è in grado. L'RFC richiede che se si implementa un gruppo di una MIB occorre implementarlo completamente; oppure può essere omesso

7.5.2 Frequenza di Polling

La politica di polling da adottare dipende da numerosi fattori:

- Il *ritardo* tollerato con cui ci vogliamo accorgere di problemi nel caso che la trap venga persa
- La *dimensione della rete*
- La *capacità della rete*
- La *potenza* della stazione di gestione

Indichiamo con *Delta* il tempo medio per completare il poll di un agent e con T l'intervallo di polling desiderato. Considerando di fare polling di un solo agent alla volta, il numero massimo di agent controllabili nell'intervallo è:

$$N \leq T/Delta$$

Il tempo medio *Delta* necessario per completare il poll dipende da numerosi fattori:

- *Ritardi di rete in entrambi i sensi*
- *Tempo di elaborazione della richiesta*
- *Tempo di generazione della risposta sull'agent*
- *Tempo di generazione della risposta sul manager*
- *Numero di oggetti da reperire sull'agent* $->$ Se eccessivo, potrebbe determinare *frammentazione* del pacchetto: tutti i ritardi di rete almeno raddoppiano e aumenta il tasso di perdita dei datagrammi UDP.

Alcune considerazioni:

- Un sistema di gestione viene valutato in primo luogo sul *tempo massimo* che impiega, nel *caso peggiore*, per risolvere il guasto. Il *periodo di polling* quindi deve essere sensibilmente minore del tempo massimo che assegnano all'amministratore del sistema, per risolvere il guasto
- Viene valutato anche il *tempo medio di risoluzione dei guasti*: il tempo medio di rilevazione basato sulla attività di polling è pari a metà del periodo di polling, supponendo che il processo stocastico di generazione dei guasti sia poissoniano

- In linea generale, il traffico di gestione non dovrebbe superare il 10% della banda disponibile
- Bisogna capire in primis quali sono le risorse più critiche necessarie al funzionamento di tutto il sistema. Questa analisi porta ad una riduzione del numero di agent su cui fare polling
- Bisogna limitare poi il polling alle informazioni veramente necessarie in modo da diminuire la dimensione delle risposte e quindi l'esigenza di banda di comunicazione.

Il ruolo delle trap Le trap non sono garantite arrivare però arrivano con una certa probabilità *tr*. Quando una trap arriva a destinazione possiamo assumere che il tempo di individuazione del guasto sia nullo. Ipotizziamo una frequenza di polling di 15 minuti. Nel *caso peggiore* (trap persa e guasto subito dopo il polling) avremmo:

$$Tp = 15 * 60$$

Se calcoliamo invece il *tempo medio*, avendo a disposizione le trap, abbiamo:

$$Tm = 0 * tr + 15 * 60 * (1 - tr)/2$$

Senza trap avremmo:

$$Tm = 15 * 60/2$$

Se *tr* è ragionevolmente vicino a 1 (cioè lo stato della rete è ragionevole), ottengo un tempo medio molto minore. Il ruolo delle trap è quindi quello di abbassare il valore medio (statistico) del tempo di rilevazione

7.5.3 Le limitazioni di SNMPv1

- Non è adatto per reperire grandi quantità di dati come ad esempio una tabella di routing, perché deve fare una richiesta per ogni riga (in SNMPv2 viene introdotta per questo la *GetBulk*)
- Le trap non sono riscontrate (in SNMPv2 viene introdotta la *Inform*)
- Gravi problemi di sicurezza: è adatto solo per il monitoring
- Non supporta direttamente "comandi imperativi"
- Non supporta comunicazioni manager-to-manager, che invece sarebbe necessaria per trattare in modo gerarchico sistemi distribuiti di grandi dimensioni