

# Administration à distance

## Protocoles de gestion répartie

### Principe

- Des agents qui surveillent l'état local — peuvent
  - être interrogés
  - envoyer des alarmes
  - accomplir des actions
  - agir comme proxy pour récupérer de l'information sur des matériels hétérogènes

Localisation : intégrés au matériel réseau, et matériels spécifiques (e.g. sondes)

- Une ou plusieurs stations d'administration (*managers*) qui interagissent avec les agents :
  - collectent des informations
  - génèrent des rapports, des traces
  - configurent
  - détectent les matériels présents et leurs caractéristiques
  - détectent des situations anormales et les signalent ou les réparent
  - peuvent agir comme agents pour d'autres stations

### Applications

- Surveillance réseau
  - Mesure sur les interfaces
  - Alarmes seuils
- Surveillance stations, serveurs
  - disponibilité, charge UC, disques
  - file d'attente d'impression

- Configuration à distance
- Gestion à distance, programmation ...

## **Matériel concerné**

- Stations de travail, serveur de terminaux
- Routeur, autocommutateur
- Un hub, un pont, une alimentation
- Matériels spécialisés de surveillance (sondes)

## **Standards existants (NMS Network management standard)**

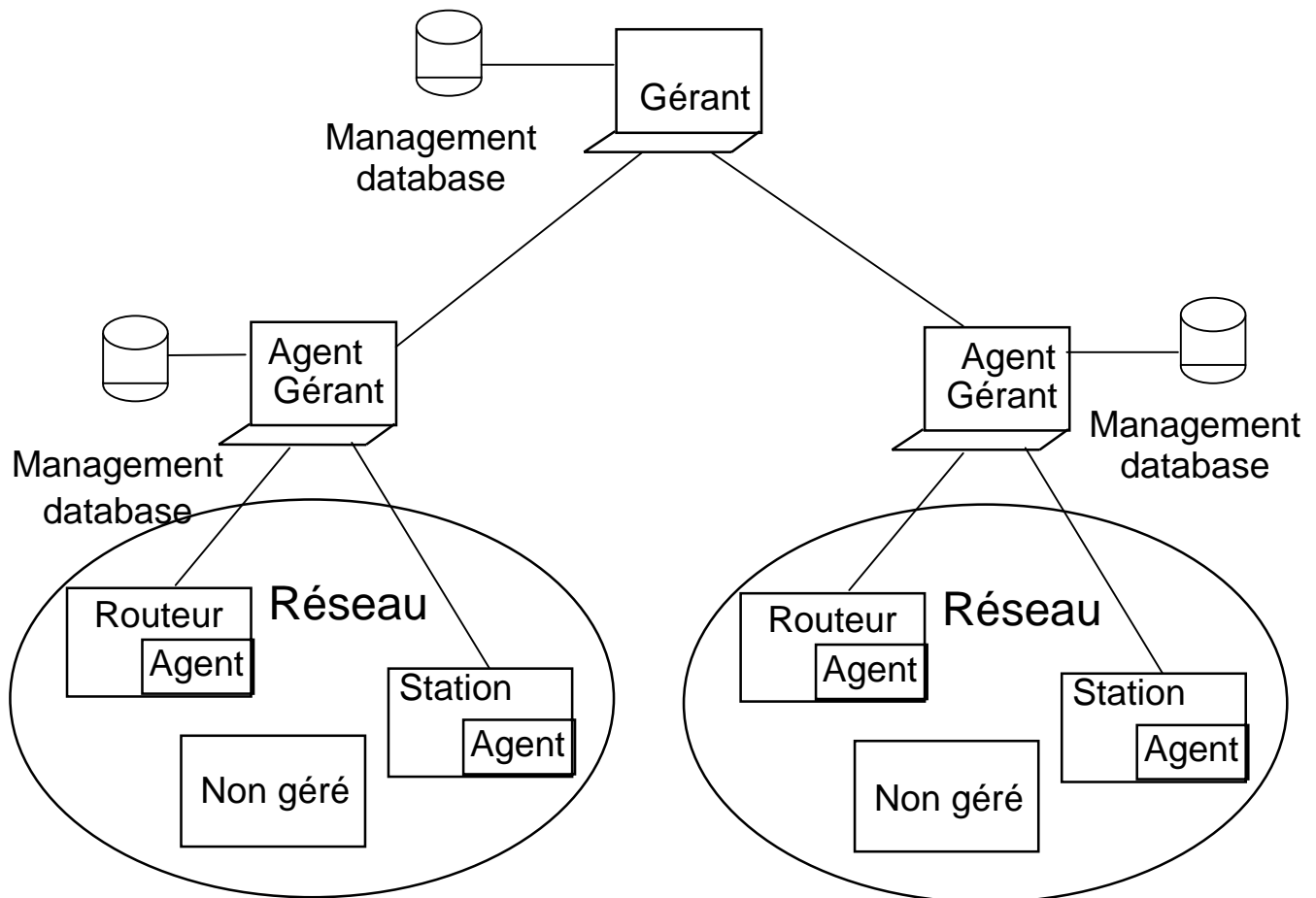
- OSI/CMIP (Common Management Information Protocol)
  - Très puissant, structuré et en couche, orienté objet, pour réseaux (LAN/WAN), gère les 7 couches
  - Très coûteux en ressources
- IETF/SNMP (Simple Network Management Protocol)
  - Simple (trop ?), surtout SNMPv1 : peu de sécurité
  - Peu coûteux pour les agents, très répandu
  - Plusieurs révisions : SNMP v1, v2, v3
- ITU-T/TMN (Telecommunication Management network)
  - Pour réseaux téléphoniques, basé sur OSI/CMIP
  - Modèle très évolué (facturation, business)
- IEEE (basé sur OSI/CMIP)
- Modèles de programmation : OSF/DME, Corba, D/Com
- Modèles basés WEB
  - WBEM WEB-based Enterprise management
  - JMX (Java Management Extensions) (applets)

# Modèle d'administration

Donne les principes du NMS

L'OSI définit 4 composantes

- **Organisation model** décomposition en composants et leur fonctionnalité, quels agents, quels stations, découpages en couche & zones, interactions entre managers



- **Information model** définit la structure des informations, les objets manipulés et leur accès
- **Communication model** protocole et ses fonctionnalités
- **Functional model** comment utiliser le NMS pour configurer, gérer, traiter fautes, performance, sécurité, comptabilité

# Modèle d'Information

Un NMS manipule des objets, avec des attributs. Ces objets forment la "management Information base" (MIB) - en fait il existe des vues: MIB de l'agent, MIB du gérant (union des MIBs des agents gérés).

SMI (Structure of management information) décrit la syntaxe et la sémantique des objets de la MIB. Elle décrit quels sont les éléments et leurs attributs.

## Exemple SNMP

- Identificateur: sysUpTime -- ou 1.3.6.1.2.1.1.3
- Syntaxe: TimeTicks
- Accès : read-only
- Statut : mandatory
- Description : temps en 1/100 s depuis dernier reboot

## Exemple CMIP

- Object class : packetCounter
- Attributes: single-valued
- Operations : get, set
- Behavior : retrieves or reset values
- Notification : generate notifications on new value

Noms : noms uniques (commençant par une minuscule, sauf les types) ou OID : identifiant numérique unique, suite de nombres (1.3.6.1.2.1.1.3) espace de noms universel (commun a CMIP et SNMP)

# Modèle de communication

Modèle client serveur : le gérant envoie des requêtes et l'agent lui renvoie des réponses

Modèle asynchrone: l'agent génère des notifications/traps

Transport : selon le NMS, ou divers

Sécurité : dépend fortement du NMS et du modèle d'organisation

Codage/syntaxe : Utilise ASN.1 (Abstract Syntax Notation One), défini par l'ISO et l'ITU-T

Fournit un langage universel et un format d'échange d'information indépendant des machines

## Exemples

PageNumber ::= INTEGER

Trade-Message ::= SEQUENCE

```
{invoice-no  INTEGER,
 name      GraphicString,
 details   SEQUENCE OF
           SEQUENCE
           {part-no INTEGER, quantity  INTEGER},
 charge    REAL }
```

→ valeur { 10, "achat HUB", { {2, 1}, {3, 4}}, 100.1 }

Il existe des types de base (INTEGER, BOOLEAN, BIT STRING, OCTET STRING, NULL, OBJECT IDENTIFIER, ...) et des types construits (CHOICE, SET, SEQUENCE, SEQUENCE OF, ENUMERATED, ...)

Chaque type ou champ est "taggé" implicitement ("universal") ou explicitement ("application" ou "context-specific") pour distinguer les informations.

```
ApplicationSyntax ::= CHOICE {  
    address      NetworkAddress,  
    counter      Counter,  
    gauge        Gauge,  
    ticks        TimeTicks,  
    arbitrary    Opaque
```

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

```
Gauge ::= [APPLICATION 2] IMPLICIT INTEGER (0..4294967295)
```

```
TimeTicks ::= [APPLICATION 3] IMPLICIT INTEGER (0..4294967295)
```

Ces tags permettent de toujours connaître le type d'un champ dans une valeur. Le codage externe précise toujours les tags et la longueur des champs

Ex : TimeTicks 132320314 (15 jours 7:33:22) est codé

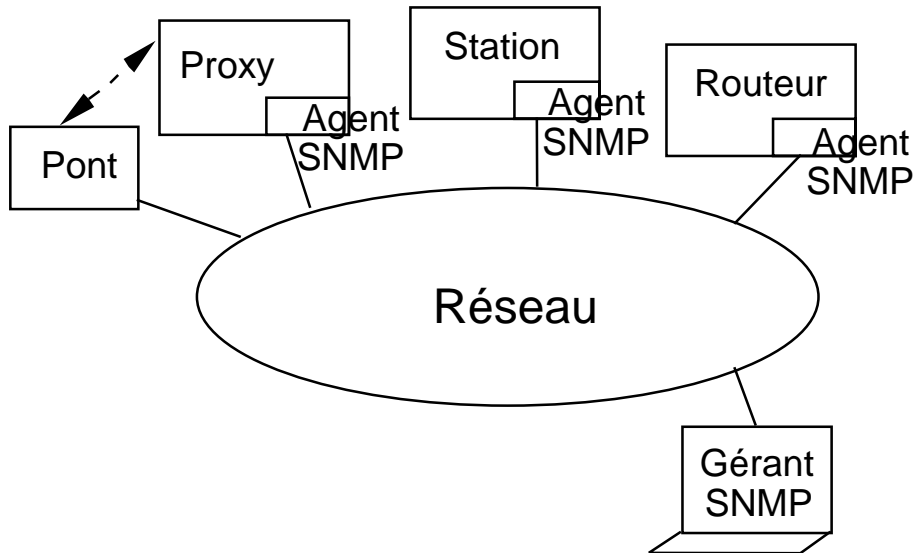
0x43 (2bits 01: application, 1bit: type simple, +3 du tag)

0x4 longueur

0x07 0xe3 0x0c 0x3a la valeur

# SNMP V1

SNMP : Simple Network Management Protocol



## Choix faits pour l'agent

- Etre simple : ne pas surcharger le nœud
- Fournir les mêmes fonctionnalités sur tous les matériels
- "proxy" pour les matériels spécialisés/non réseau
- informations accessibles limitées (valeurs simples)

## Choix faits pour le modèle d'organisation

- Pas de hiérarchie : une station face aux agents, si plusieurs stations chacune gère ses agents séparément
  - Protocole client serveur en mode utilisateur : le gérant envoie des demandes et reçoit des réponses, il doit gérer les demandes en cours (match demandes – réponses, "retries", erreurs)
- ➔ protocole UDP, pas de gestion d'erreur dans agent

## Primitives du protocole

- Un agent est défini par des *variables* simples qu'on peut lire et écrire à distance
- Opérations
  - get : le gérant demande une ou plusieurs variables, l'agent répond une liste (nom, valeur)
  - getnext : comme get, mais on récupère les variables "après" celle demandée (parcours de tables)
  - set : le gérant envoie une ou plusieurs variables avec valeur, l'agent répond la liste (nom, valeur nouvelle); sert aussi au contrôle (set variable statut)
  - trap — message asynchrone généré par l'agent sur condition *exceptionnelle* envoi message (type, valeurs) (e.g. arrêt d'une interface d'un routeur)

le protocole privilégie le "polling" par la station d'administration  
→ pas de notifications.

Pas de commandes : pour changer un élément d'état, modifier une variable qui est son statut (interface on/off/en test, reboot prévu dans xx secondes ...)

# Structure de l'information Management Information base (MIB)

## Définie par

- RFC 1155 — SMI : Structure of Management Information  
Définit le schéma de structure des MIB et les types (string, integer, ...)
- RFC 1212 — CMI : Concise MIB definition  
Définit les règles générales d'écritures des MIB

## Propriétés des MIB

La MIB décrit l'agent complet (== une machine sur le réseau)

Propriétés associées aux nœuds (définies par la MIB)

- Identité : forme numérique et forme symbolique
- mode d'accès (Read Only, RW, Write Only, Not accessible)
- structure, syntaxe, type

## Espace des noms

Notation: Dewey (liste pointée de nombres) – Utilisation de noms symboliques (un nom doit être unique dans une MIB)

Les adresses (ou clés) s'appellent des oid ("object identifier")

internet **OBJET IDENTIFIER** ::= { iso org(3) dod(6) 1 }

mgmt **OBJET IDENTIFIER** ::= { internet 2 }

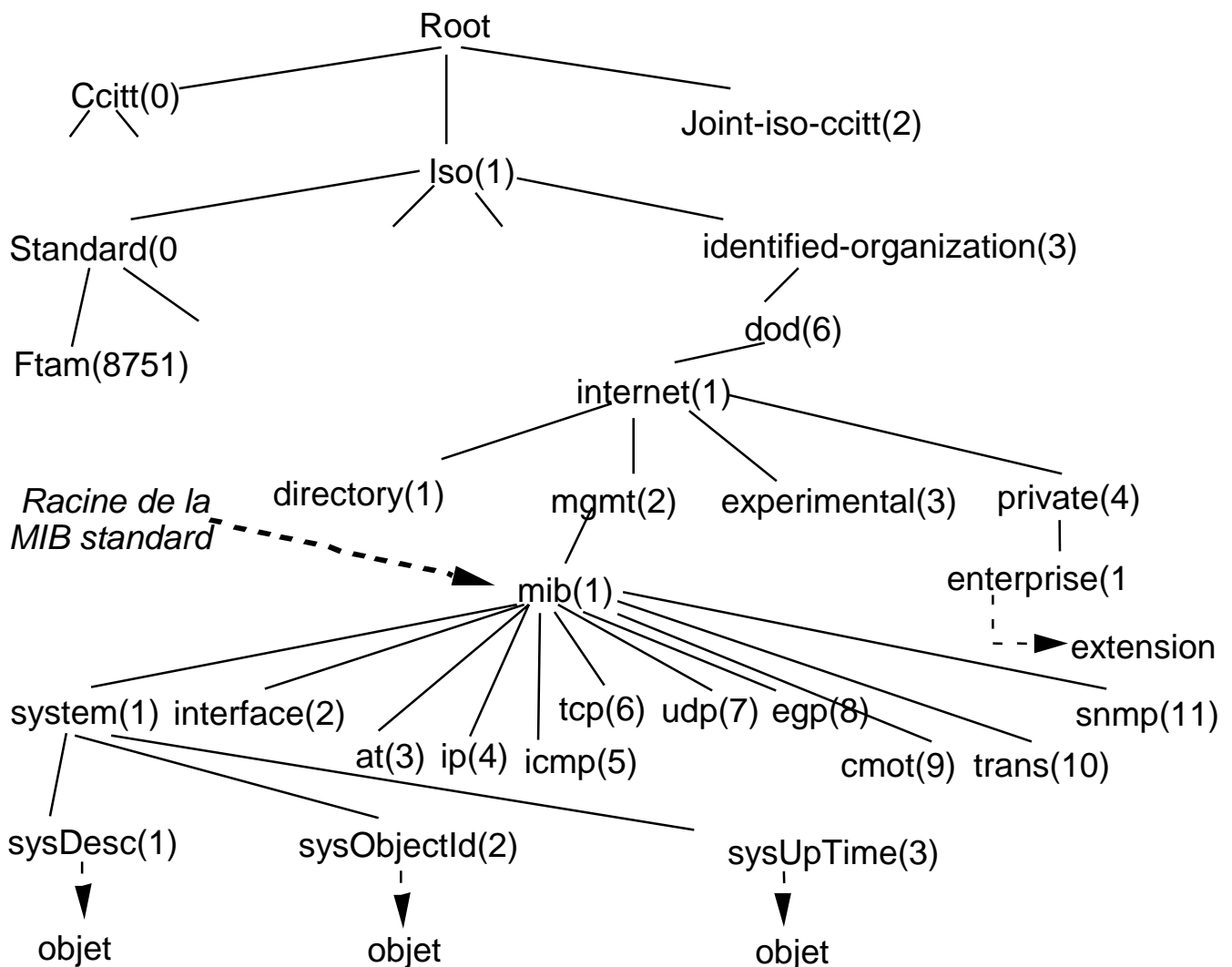
Possibilité d'ajouter un sous arbre à la MIB de base:

- MIB expérimentales (FDDI, CLNS, Appletalk ...)
- MIB "d'entreprise" privées (Cisco, HP ...)

Valeurs : Uniquement dans les feuilles (les autres nœuds sont "non-accessibles")

Numéros : à partir de 1 dans la partie SNMP

On ne peut que rajouter des fils à droite ou déclarer un nœud obsolète : pas de réutilisation autorisée



RFC1155-SMI DEFINITIONS ::= BEGIN

**EXPORTS** -- EVERYTHING

internet, directory, mgmt,  
experimental, private, enterprises,  
OBJECT-TYPE, ObjectName, ObjectSyntax, SimpleSyntax,  
ApplicationSyntax, NetworkAddress, IpAddress,  
Counter, Gauge, TimeTicks, Opaque;

-- the path to the root

internet       **OBJECT IDENTIFIER** ::= { iso org(3) dod(6) 1 }  
directory      **OBJECT IDENTIFIER** ::= { internet 1 }  
mgmt           **OBJECT IDENTIFIER** ::= { internet 2 }  
experimental   **OBJECT IDENTIFIER** ::= { internet 3 }  
private        **OBJECT IDENTIFIER** ::= { internet 4 }  
enterprises    **OBJECT IDENTIFIER** ::= { private 1 }

-- definition of object types

**OBJECT-TYPE** MACRO ::= **BEGIN**

**TYPE NOTATION** ::= "SYNTAX" type (**TYPE** ObjectSyntax)  
                  "ACCESS" Access  
                  "STATUS" Status

**VALUE NOTATION** ::= value (**VALUE** ObjectName)

Access ::= "read-only"  
          | "read-write"  
          | "write-only"  
          | "not-accessible"  
Status ::= "mandatory"  
          | "optional"  
          | "obsolete"

**END**

-- names of objects in the MIB

ObjectName ::= **OBJECT IDENTIFIER**

-- syntax of objects in the MIB

ObjectSyntax ::= **CHOICE** {  
    simple  
    SimpleSyntax,

-- note that simple SEQUENCES are not directly mentioned here to keep things simple (i.e.,  
-- prevent misuse). However, application-wide types which are **IMPLICITLY** encoded simple  
-- SEQUENCES may appear in the following **CHOICE**

    application-wide  
    ApplicationSyntax  
}

SimpleSyntax ::= **CHOICE** {

```

    number    INTEGER,
    string    OCTET STRING,
    object    OBJECT IDENTIFIER,
    empty     NULL
}

ApplicationSyntax ::=
CHOICE {
    address    NetworkAddress,
    counter    Counter,
    gauge      Gauge,
    ticks      TimeTicks,
    arbitrary  Opaque
-- other application-wide types, as they are
-- defined, will be added here
}

-- application-wide types

NetworkAddress ::=
CHOICE {
    internet    IPAddress
}

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

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

Gauge ::=
[APPLICATION 2]
IMPLICIT INTEGER (0..4294967295)

TimeTicks ::=
[APPLICATION 3]
IMPLICIT INTEGER (0..4294967295)

Opaque ::=
[APPLICATION 4]    -- arbitrary ASN.1 value,
IMPLICIT OCTET STRING -- "double-wrapped"

END

```

# Exemple de MIB

## MIB II (RFC 1213)

Commune aux différents agents (== "entité réseau IP")

Mise à jour de la version MIB I (RFC1156)

- Groupe "Système"
  - Informations génériques de configuration
- Groupe "Interfaces"
  - Informations concernant les interfaces :
  - type, adresse, nb octets in/out, statut,...
- Groupe "ARP"
  - Liaison Adresse Physique – Adresse logique
  - obligatoire par compatibilité MIB-I, mais déprécié : utiliser les tables/protocole
- Groupe "IP"
  - Informations sur le niveau IP: TTL, forwarding?, tables de routage, nombre de paquets, d'octets, de "forward"...
- Groupe "ICMP"
  - Informations statistiques sur les messages ICMP:
  - Nombre de messages, de messages Echo, ...
- Groupe "TCP"
  - Informations sur les connexions TCP : connexions en cours, nombre de messages, de circuits ouverts, TTL, ...
- Groupe "UDP", "EGP", "SNMP",

## RFC1213-MIB

**DEFINITIONS ::= BEGIN**

### **IMPORTS**

mgmt, NetworkAddress, IpAddress, Counter, Gauge,  
TimeTicks  
FROM RFC1155-SMI

### **OBJECT-TYPE**

FROM RFC-1212;

-- MIB-II (same prefix as MIB-I)

mib-2 **OBJECT IDENTIFIER** ::= { mgmt 1 }

-- textual conventions

DisplayString ::=

### **OCTET STRING**

-- This data type is used to model textual information taken  
-- from the NVT ASCII character set. By convention, objects  
-- with this syntax are declared as having

-- SIZE (0..255)

PhysAddress ::=

### **OCTET STRING**

-- This data type is used to model media addresses. For many  
-- types of media, this will be in a binary representation.  
-- For example, an ethernet address would be represented as  
-- a string of 6 octets.

-- groups in MIB-II

system **OBJECT IDENTIFIER** ::= { mib-2 1 }

interfaces **OBJECT IDENTIFIER** ::= { mib-2 2 }

at **OBJECT IDENTIFIER** ::= { mib-2 3 }

ip **OBJECT IDENTIFIER** ::= { mib-2 4 }

icmp **OBJECT IDENTIFIER** ::= { mib-2 5 }

tcp **OBJECT IDENTIFIER** ::= { mib-2 6 }

udp **OBJECT IDENTIFIER** ::= { mib-2 7 }

egp **OBJECT IDENTIFIER** ::= { mib-2 8 }

-- historical (some say hysterical)

-- cmot **OBJECT IDENTIFIER** ::= { mib-2 9 }

transmission **OBJECT IDENTIFIER** ::= { mib-2 10 }

snmp **OBJECT IDENTIFIER** ::= { mib-2 11 }

-- the System group

-- Implementation of the System group is mandatory for all  
-- systems. If an agent is not configured to have a value  
-- for any of these variables, a string of length 0 is  
-- returned.

sysDescr **OBJECT-TYPE**

SYNTAX DisplayString (SIZE (0..255))

ACCESS read-only

STATUS mandatory

DESCRIPTION

"A textual description of the entity. This value  
should include the full name and version  
identification of the system's hardware type,  
software operating-system, and networking

software. It is mandatory that this only contain printable ASCII characters."

::= { system 1 }

**sysObjectID OBJECT-TYPE**

**SYNTAX OBJECT IDENTIFIER**

**ACCESS** read-only

**STATUS** mandatory

**DESCRIPTION**

"The vendor's authoritative identification of the network management subsystem contained in the entity. This value is allocated within the SMI enterprises subtree (1.3.6.1.4.1) and provides an easy and unambiguous means for determining `what kind of box' is being managed. For example, if vendor `Flintstones, Inc.' was assigned the subtree 1.3.6.1.4.1.4242, it could assign the identifier 1.3.6.1.4.1.4242.1.1 to its `Fred Router'."

::= { system 2 }

**sysUpTime OBJECT-TYPE**

**SYNTAX** TimeTicks

**ACCESS** read-only

**STATUS** mandatory

**DESCRIPTION**

"The time (in hundredths of a second) since the network management portion of the system was last re-initialized."

::= { system 3 }

**sysContact OBJECT-TYPE**

**SYNTAX** DisplayString (SIZE (0..255))

**ACCESS** read-write

**STATUS** mandatory

**DESCRIPTION**

"The textual identification of the contact person for this managed node, together with information on how to contact this person."

::= { system 4 }

**sysName OBJECT-TYPE**

**SYNTAX** DisplayString (SIZE (0..255))

.....

-- the Interfaces group

-- Implementation of the Interfaces group is mandatory for

-- all systems.

**ifNumber OBJECT-TYPE**

**SYNTAX** INTEGER

**ACCESS** read-only

**STATUS** mandatory

**DESCRIPTION**

"The number of network interfaces (regardless of their current state) present on this system."

```
::= { interfaces 1 }
```

```
-- the Interfaces table  
-- The Interfaces table contains information on the entity's  
-- interfaces. Each interface is thought of as being  
-- attached to a `subnetwork'. Note that this term should  
-- not be confused with `subnet' which refers to an  
-- addressing partitioning scheme used in the Internet suite  
-- of protocols.
```

```
ifTable OBJECT-TYPE
```

```
SYNTAX SEQUENCE OF IfEntry
```

```
ACCESS not-accessible
```

```
STATUS mandatory
```

```
DESCRIPTION
```

```
"A list of interface entries. The number of  
entries is given by the value of ifNumber."
```

```
::= { interfaces 2 }
```

```
ifEntry OBJECT-TYPE
```

```
SYNTAX IfEntry
```

```
ACCESS not-accessible
```

```
STATUS mandatory
```

```
DESCRIPTION
```

```
"An interface entry containing objects at the  
subnetwork layer and below for a particular  
interface."
```

```
INDEX { ifIndex }
```

```
::= { ifTable 1 }
```

```
IfEntry ::=
```

```
SEQUENCE {
```

```
    ifIndex          INTEGER,  
    ifDescr          DisplayString,  
    ifType           INTEGER,  
    ifMtu            INTEGER,  
    ifSpeed          Gauge,  
    ifPhysAddress    PhysAddress,  
    ifAdminStatus    INTEGER,  
    ifOperStatus     INTEGER,  
    ifLastChange     TimeTicks,  
    ifInOctets       Counter,  
    ifInUcastPkts    Counter,  
    ifInNUcastPkts  Counter,  
    ifInDiscards     Counter,  
    ifInErrors       Counter,  
    ifInUnknownProtos Counter,  
    ifOutOctets      Counter,  
    ifOutUcastPkts  Counter,  
    ifOutNUcastPkts Counter,  
    ifOutDiscards   Counter,  
    ifOutErrors      Counter,  
    ifOutQLen        Gauge,  
    ifSpecific       OBJECT IDENTIFIER
```

```
}
```

```
.....
```

# Identification des instances

La MIB décrit des objets génériques (nom système, compteur d'octets reçus sur une interface)

Dans l'agent ces objets sont instanciés et les valeurs sont associées aux instances.

Deux cas possibles

- Instance unique (e.g. nom système, ifNumber)

→ l'oid de l'instance est <id-feuille>.0

e.g. sysDescr ::= 1.3.6.1.2.1.1.1

→ instance sysDescr.0:

get(sysDescr.0) = get (1.3.6.1.2.1.1.1.0)

→ {1.3.6.1.2.1.1.1.0, String SunOs4.1}

getnext(sysDescr) = getnext (1.3.6.1.2.1.1.1)

→ {1.3.6.1.2.1.1.1.0, String SunOs4.1}

mais

getnext(sysDescr.0) → {sysObjectID.0, OID: enterprise;Sun.2.1.1}

- Instance répétée (e.g. compteur d'octets reçus sur une interface, car il y a un nombre variable d'interfaces)

Alors les différentes instances sont distinguées par un "index" et l'OID de l'instance est nom.index

e.g. ifInOctets.2 → Integer 102338

Les différentes instances sont regroupées dans des "tableaux"

e.g. tableau des interfaces IP

- structuré en colonnes qui décrivent les différentes variables d'une même instance.
  - Elles ont toutes la même structure (fixée dans la MIB)
  - Ce sont des sous-arbres d'un nœud de type "sequence".
  - La structure doit être régulière (toutes les lignes de même format, avec le même nombre de valeurs)
- Les "lignes" représentent la même information pour toutes les instances (e.g. adresse pour chaque interface).
  - L'ensemble des lignes forment un "sequence of"
  - La structure doit être régulière (toutes les lignes de même format, avec le même nombre de valeurs)
- La définition de la "sequence of" précise un champ index, formé d'un ou plusieurs noms de colonnes
- L'index instancié est la concaténation des transformations en chaîne de labels des valeurs des différents champs index

## Exemples

INDEX {ifIndex} et ifIndex INTEGER ::=3 → .3

INDEX { ipAdEntAddr } et ipAdEntAddr IpAddress::=127.0.0.1  
→ .127.0.0.1

INDEX { name } et name OCTET STRING (0..20)::="Ip2"  
→ .3.108.112.50

INDEX { udpLocalAddress, udpLocalPort } et  
udpLocalAddress IpAddress::=127.0.0.1,  
udpLocalAddress INTEGER (0..65535) = 2000  
→ .127.0.0.1.2000

Accès par get si index connu, ou par getnext. Getnext répété permet de parcourir tout le tableau

## Exemple

MIB :

```
ipAddrTable OBJECT-TYPE
    SYNTAX SEQUENCE OF IpAddrEntry ::= { ip 20 }
ipAddrEntry OBJECT-TYPE
    SYNTAX IpAddrEntry INDEX { ipAdEntAddr } ::= { ipAddrTable 1 }
IpAddrEntry ::=
    SEQUENCE {
        ipAdEntAddr      IpAddress,
        ipAdEntIfIndex   INTEGER,
        ipAdEntNetMask   IpAddress,
        ipAdEntBcastAddr INTEGER,
        ipAdEntReasmMaxSize  INTEGER (0..65535)
    }
ipAdEntAddr      OBJECT-TYPE SYNTAX IpAddress ::= { ipAddrEntry 1 }
ipAdEntIfIndex   OBJECT-TYPE SYNTAX INTEGER ::= { ipAddrEntry 2 }
ipAdEntNetMask   OBJECT-TYPE SYNTAX IpAddress ::= { ipAddrEntry 3 }
ipAdEntBcastAddr OBJECT-TYPE SYNTAX INTEGER ::= { ipAddrEntry 4 }
ipAdEntReasmMaxSize  OBJECT-TYPE SYNTAX INTEGER (0..65535) ::=
    { ipAddrEntry 5 }
```

Deux interfaces, 127.0.0.1 et 129.88.34.1 ==>

feuilles dans l'ordre de parcours de l'arbre

```
...ip.ipAddrTable.ipAddrEntry.ipAdEntAddr.127.0.0.1 -> IpAddress 127.0.0.1
...ip.ipAddrTable.ipAddrEntry.ipAdEntAddr.129.88.34.1 -> IpAddress 129.88.34.1
...ip.ipAddrTable.ipAddrEntry.ipAdEntIfIndex.127.0.0.1 -> Integer 1
...ip.ipAddrTable.ipAddrEntry.ipAdEntIfIndex.129.88.34.1 -> Integer 2
...ip.ipAddrTable.ipAddrEntry.ipAdEntNetMask.127.0.0.1-> Ipaddress 255.0.0.0
...ip.ipAddrTable.ipAddrEntry.ipAdEntNetMask.129.88.34.1-> Ipaddress
                                                                255.255.255.0
...ip.ipAddrTable.ipAddrEntry.ipAdEntBcastAddr.127.0.0.1 -> Integer 1
...ip.ipAddrTable.ipAddrEntry.ipAdEntBcastAddr.129.88.34.1 -> Integer 1
...ip.ipAddrTable.ipAddrEntry. ipAdEntReasmMaxSize.127.0.0.1 -> Integer 1536
...ip.ipAddrTable.ipAddrEntry. ipAdEntReasmMaxSize.129.88.34.1->
                                                                Integer 1500
```

## Parcours de la table :

-- Partir du père (nœud interne), getnext donne le 1er élément

getnext(ipAdEntAddr, ipAdEntNetMask) ->

```
{    {ipAdEntAddr.127.0.0.1, IpAddress 127.0.0.1}
    {ipAdEntNetMask.127.0.0.1, Ipaddress 255.0.0.0}
}
```

-- Partir d'un élément, getnext donne l'élément suivant

getnext(ipAdEntAddr.127.0.0.1, ipAdEntNetMask.127.0.0.1) ->

```
{    {ipAdEntAddr. 129.88.34.1, IpAddress 129.88.34.1}
    {ipAdEntNetMask. 129.88.34.1, Ipaddress 255.255.255.0}
}
```

-- On finit : on n'est plus dans le sous-arbre

getnext(ipAdEntAddr.129.88.34.1, ipAdEntNetMask.129.88.34.1) ->

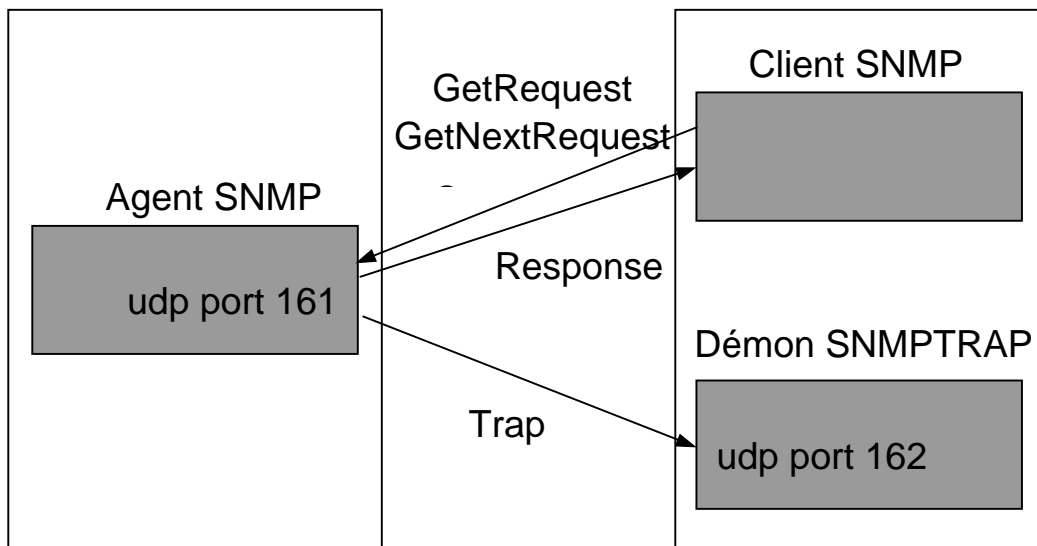
```
{    { ipAdEntIfIndex.127.0.0.1, Integer 1}
    { ipAdEntBcastAddr.127.0.0.1, Integer 1}
}
```

Fin : le parcours est terminé quand on a l'erreur EndOfMib,  
*ou si* les réponses changent de préfixe

# Le protocole SNMP V1

## Messages

- 3 opérations serveur : lire (“get”), lire suivant (“get-next”), écrire (“set”) — sur un ensemble de variables/valeurs
- 2 opérations agent : réponse, trap
- Pas de notion de commande (e.g. activer un interface, redémarrer) : on positionne des variables ad-hoc.



Conçu pour que les agents soient simples :

- Un agent par adresse IP, port figé
- Mode non connecté, pas de “retry”, transport par UDP
- Protocole asynchrone : le serveur envoie une requête (get/get-next/set). L’agent renvoie plus tard une “réponse”
- Le “set” de plusieurs variables est atomique pour permettre des mises à jour complexes

Pourquoi : doit marcher si le réseau est déficient, et ne pas encombrer le réseau

→ le client doit tout gérer

# Message SNMP V1

type, id, communauté, code d'erreur, (variable, valeur)\*  
(type = get, get-next, set, get-response)

```
Message ::= {
  version version-1(0),
  community "public",          <--- pour la sécurité
  data {
    getResponse {
      request-id 17,           <-- associe requête/réponse
      error-status noError(0), <-- présent dans requête
      error-index 0,           avec valeur 0
      variable-binding {      <-- liste de couples
        {
          name 1.3.6.1.2.1.1.1.0,
          value {              <-- pour get, valeur vide
            simple {          <-- valeur typée
              string "unix"
            }
          }
        }
      }
    }
  }
}
```

```
% smpget horus public system.sysUpTime.0
system.sysUpTime.0 = TimeTicks: 132320314 (15 jours 7:33:22)
brahma.imag.fr -> horus.imag.fr UDP D=161 S=35874 LEN=57
  0: 0090 9278 c400 0800 2086 15d0 0800 4500  ...x....  ....E.
 16: 004d f49d 4000 ff11 4046 8158 1e0a 8158  .Mô.@...@F.X...X
 32: 2601 8c22 00a1 0039 8ab9 3082 002d 0201  &..."...9..0..-..
 48: 0004 0670 7562 6c69 63a0 2002 0471 f39f  ...public. ..q..
 64: d602 0100 0201 0030 8200 1030 8200 0c06  .....0...0....
 80: 082b 0601 0201 0103 0005 00          .+.....
horus.imag.fr -> brahma.imag.fr UDP D=35874 S=161 LEN=55
  0: 0800 2086 15d0 0090 9278 c400 0800 4500  ..  ....x....E.
 16: 004b a114 4000 fe11 94d1 8158 2601 8158  .K..@.....X&..X
 32: 1e0a 00a1 8c22 0037 ced9 302d 0201 0004  .....".7..0-....
 48: 0670 7562 6c69 63a2 2002 0471 f39f d602  .public. ..q....
 64: 0100 0201 0030 1230 1006 082b 0601 0201  .....0.0...+....
 80: 0103 0043 0407 e31e ffffffff d3      ...C.....
```

## Codage

### Protocole ASN1

SNMP : choix de types restreint pour simplifier les agents :

- Types standards : entier, chaîne d'octets, compteur, "Jauges"
- Types construits : séquence, articles
- Types définis : e.g. IpAddress = chaîne de 4 octets

### Opérations

– get : récupérer une (ou plusieurs) information en la nommant

Le client envoie un Get avec les champs valeurs à 'nul'

L'agent renvoie un message Response avec les mêmes champs, mais avec les valeurs

Erreur si réponse trop longue

– getnext : récupérer l'information 'suivante'

La demande a le même format que pour get

L'agent renvoie un message Response où chaque couple

(nom, nul) est remplacé par le couple

(suivant(nom), valeur(suivant(nom)))

suivant : au sens du parcours de l'arbre MIB

Il existe une erreur : 'Endof MIB'

Lecture de tableau : faire 'getnext' jusqu'à ce que le nom retourné ne soit plus dans la table, ou jusqu'à EndOfMib

– set : sert à modifier

Le client envoie un Set avec des couples (nom, valeur)

L'agent renvoie un message Response avec les couples  
(nom, valeur réelle)

Modification atomique : toutes les variables ou aucune

Note : Protocole trop simple, revu en SNMPV2

– trap : signale un changement d'état — valeurs prédéfinies :  
coldStart, linkDown, LinkUp, ..., enterpriseSpecific +  
arguments

Asynchrone, envoyé à une adresse configurée dans l'agent

# Sécurité SNMP V1

La 'communauté' sert d'identification ( $\cong$  mot de passe)

A une communauté est associé des droits d'accès sur un sous-ensemble des variables de l'agent : "*Read-Only*" ou "*Read-Write*"

Chaque variable de l'agent a l'un des caractéristiques : "*Read-Only*", "*Read-Write*", "*Write-Only*", "*Not-accessible*"

Opérations autorisées :

Mode d'accès	Objet accédé suivant la MIB			
	Read-Only	Read-Write	Write-Only	Not-accessible
Read-Only	3	3	1	1
Read-Write	3	2	4	1

Classe	1	2	3	4
Opérations possibles	Rien	Get, Get-Next, Set, Trap	Get, Get-Next, Trap	Get, Get-Next, Set, Trap



Pas de valeur lisible

Remarques :

Sécurité faible (on peut 'écouter' et trouver la communauté) →

- soit 'set' non disponible
- soit filtrage supplémentaire selon l'adresse source IP

# Example de MIB: Repeater (RFC 1516)

Un répéteur connecte des segments de types ethernet, avec un domaine de collision et une sortie répliquée sur tous les segments. Les répéteurs sont souvent modulaires → les "ports" sont regroupés en "groupes".

```
snmpDot3RptrMgt OBJECT IDENTIFIER ::= { mib-2 22 }
```

La MIB définit une convention textuelle

```
MacAddress ::= OCTET STRING (SIZE (6)) -- a 6 octet address
```

et 3 groupes,

- rptrBasicPackage group (obligatoire)  
caractéristiques, statut, contrôle par : répéteur, groupe (table), port (table à 2 indexes)

```
rptrOperStatus OBJECT-TYPE
```

```
SYNTAX INTEGER { other(1),           -- undefined or unknown status
                 ok(2),               -- no known failures
                 rptrFailure(3),      -- repeater-related failure
                 groupFailure(4),     -- group-related failure
                 portFailure(5),      -- port-related failure
                 generalFailure(6) }  -- failure, unspecified type
```

```
...
```

```
REFERENCE      "Reference IEEE 802.3 Rptr Mgt, 19.2.3.2, aRepeaterHealthState."
::= { rptrRptrInfo 2 }
```

```
rptrReset OBJECT-TYPE
```

```
SYNTAX INTEGER { noReset(1), reset(2) }
ACCESS read-write STATUS mandatory
DESCRIPTION
```

```
"Setting to reset(2) causes a transition to the START state of Fig 9-2 in section 9 [IEEE 802.3 Std]."
```

```
rptrPortAdminStatus OBJECT-TYPE
```

```
SYNTAX INTEGER { enabled(1), disabled(2) }
ACCESS read-write STATUS mandatory
DESCRIPTION      "Setting this object to disabled(2) disables the port. ..."
```

```
rptrPortAutoPartitionState OBJECT-TYPE
```

```
SYNTAX INTEGER { notAutoPartitioned(1), autoPartitioned(2) }
ACCESS read-only STATUS mandatory
DESCRIPTION      "The autoPartitionState flag indicates whether the port is
                 currently partitioned by the repeater's auto-partition protection ..."
```

- rptrMonitorPackage group (optionnel : tout ou rien)  
compteurs par : répéteur, groupe, port  
Chaque info est au niveau selon son type :
  - nombre collisions: global et port,
  - octets in: port,
  - octet out: groupe

*!! compte des paquets physiques et ceux que le répéteur "voit" : ce n'est ni la MIB-2 ni une sonde.*
- rptrAddrTrackPackage group (optionnel)  
table par port : adresse Mac source du dernier paquet reçu
- Traps  
Indique le changement global de l'état du répéteur ou d'un groupe (mais pas d'un port)

rptrHealth TRAP-TYPE

ENTERPRISE snmpDot3RptrMgt

VARIABLES { rptrOperStatus }

DESCRIPTION

"The rptrHealth trap conveys information related to the operational status of the repeater. This trap is sent either when the value of rptrOperStatus changes, or upon completion of a non-disruptive test. The rptrHealth trap must contain the rptrOperStatus object. The agent may optionally include the rptrHealthText object in the varBind list. See the rptrOperStatus and rptrHealthText objects for descriptions of the information that is sent. The agent must throttle the generation of consecutive rptrHealth traps so that there is at least a five-second gap between traps of this type. When traps are throttled, they are dropped, not queued for sending at a future time. (Note that 'generating' a trap means sending to all configured recipients.)"

REFERENCE

"Reference IEEE 802.3 Rptr Mgt, 19.2.3.4, hubHealth notification."

::= 1

# MIB RMON (RFC 1271) Remote monitoring

- Groupe “Statistique”  
Statistiques par interface :
  - Nb d’octets (distribution en fonction de la taille)
  - Nb paquets : tous types, broadcast, multicast
  - Nb paquets erronés (trop court, trop long, err CRC)
  - Nb collisions
- Groupe “Historique”  
Mémorisation des informations statistiques (période programmable par la station de gestion)
- Groupe “Alarme”  
Positionnement d’intervalles et de seuils pour générer des traps en fonction des informations statistiques
- Groupe “Host”  
Statistiques par machine sur un interface (octets, paquets, ...) - à programmer par groupe “HostControl”
- Groupe “Matrice”  
Matrice de trafic entre toutes les paires de machines d'un interface (octets, paquets, ...) - à programmer
- Groupe “HostTopN”  
Liste des N entités les plus actives dans un délai programmable (utilise groupe “Host”)
- Groupes “Filtre”, “Capture”, “Events”  
Permet de capturer des paquets et/ou de générer des événements

## Exemple de programmation

Chaque table Host a un numéro (index) et est décrite par une ligne de la table HostControl, avec un HostControlIndex et un HostControlStatus de valeurs EntryStatus

```
hostControlEntry OBJECT-TYPE ... INDEX { hostControlIndex }
```

```
HostControlEntry ::= SEQUENCE {  
    hostControlIndex      INTEGER (1..65535),      -- read only  
    hostControlDataSource OBJECT IDENTIFIER,    -- read-write, ifIndex de capture  
    hostControlTableSize  INTEGER,              -- read-only  
    hostControlLastDeleteTime TimeTicks,        -- read-only  
    hostControlOwner      OwnerString,          -- read-write  
    hostControlStatus     INTEGER }              -- read-write
```

```
EntryStatus ::= INTEGER {  
    valid(1),  
    createRequest(2),  
    underCreation(3),  
    invalid(4) }
```

```
hostEntry OBJECT-TYPE ... INDEX { hostIndex, hostAddress } ...
```

- Créer une entrée :  
choisir un index et exécuter  
    set (HostControlStatus.index, createRequest)  
L'index ne doit pas encore exister. L'agent initialise la ligne, et met le HostControlStatus à "underCreation".
- Tant que l'entrée est "underCreation" la station peut la configurer; l'entrée - et la table Host - n'est pas active.
- En fin de configuration, activation par  
    set (HostControlStatus.index, valid)
- Destruction par  
    set (HostControlStatus.index, invalid)
- Après un set(invalid) une entrée peut continuer à être vue ou disparaître → vérifier le statut des lignes vues par getNext
- L'agent peut d'office supprimer des lignes "underCreation" sont dans cet état depuis "trop longtemps".

# SNMP v2

## Limites de SNMP v1

- Modèle de sécurité très faible
- Modèle administratif faible : un niveau gérant - agents
- Transport simple → collecte de tables : coûteux

## → SNMP v2

- Modèle d'architecture modifié
  - nouveaux PDU :
    - inform-request (demande d'information de gérant à gérant)
    - get-bulk-request ( $\cong$  getnext avec répétition)  
getbulk(non-repeater, max-repetition, var1, ... varn) → PDU response contenant des (var, val) :
      - ◆ en tête, comme pour getnext pour les variables varX, X de 1 à 'non-repeater',
      - ◆ puis, si réponse pas trop longue, comme pour getnext pour les variables varX, X de 'non-repeater'+1 à n,
      - ◆ puis, si réponse pas trop longue, au plus 'max-repetition' -1 fois, comme pour getnext appliqué aux n-'non-repeater' dernières variables mises dans la réponse.

### Exemple (cf. exemple getnext plus haut)

```
getbulk(1, 3, sysUpTime, ipAdEntAddr, ipAdEntNetMask) →  
  { {sysUpTime.0, TimeTicks 132320314}  
    {ipAdEntAddr.127.0.0.1, IpAddress 127.0.0.1}  
    {ipAdEntNetMask.127.0.0.1, Ipaddress 255.0.0.0}  
    {ipAdEntAddr.129.88.34.1, IpAddress 129.88.34.1}  
    {ipAdEntNetMask.129.88.34.1, Ipaddress 255.255.255.0}  
    { ipAdEntIfIndex.127.0.0.1, Integer 1}  
    { ipAdEntBcastAddr.127.0.0.1, Integer 1}      }
```

- Format du trap modifié pour se rapprocher des autres

- PDU (appelé désormais "notification")
- Transport possible autres que UDP (e.g. ISO)
- Modifications SMI et MIB
  - Notion de déclaration de "conformance" (gère les notions d'obligatoire, optionnel, extension)
  - Définition de 'textual conventions'
  - Gestion des tables standardisée
    - Possibilité d'ajouter des colonnes à une table en déclarant une table 'augmentée' ailleurs dans la MIB
    - Ajout/suppression de lignes : standardisation de la méthode de l'exemple HostControl de RMON
  - Modification du groupe SNMP pour gérer les info SNMP gestion des dialogues 'manager to manager' contextes de sécurité ...
- Sécurité : modèle complet, transport avec authentification et chiffrement possible
- Compatibilité SNMP V1 & V2 : pas de compatibilité ascendante → gérants bilingues ou utilisation de proxy

Mais SNMPv2 non accepté (à cause du modèle de sécurité)

→ standardisation de la partie admise début 1996 (SNMPv2C, C pour communauté) et relance de la réflexion pour un nouveau modèle SNMPv3 (1998)

# SNMP v3

Cherche à être un modèle général, consensuel, incluant les autres versions.

Nombreux documents (RFC) → modèle d'architecture pour la documentation

## **Modèle d'architecture**

Chaque entité est identifiée par un 'snmpEngineID' (système, adresse réseau + autres paramètres)

Chaque entité est formée de composants qui interagissent.

## **Modèle de sécurité**

'User based' : les droits sont liés à un nom d'utilisateur. Les échanges peuvent être signés (HMAC), chiffrés (CBC-DES), datés pour protéger contre les différentes attaques possibles (sauf 'denial of service' et 'analyse de trafic').

Chaque agent possède une base locale qui définit les droits d'accès, selon le modèle (VACM - View Based Access Control Model). Le VACM définit pour un utilisateur son niveau de sécurité, le contexte d'exécution, les 'vues' possibles sur la MIB et les politiques d'accès autorisées (généralisation du modèle de sécurité de SNMPv1)

# Les produits

## Agents

- intégré au produit, y compris les MIB spécifiques –avec lecture noyau pour récupérer les informations

La plupart des produits réseaux – routeurs, ponts, ...– sont fournis avec un agent intégré – en standard si matériel important (routeur, commutateur programmable) ou en option (hub, petit commutateur)

→ support complet avec toutes les MIB applicables

- démon, avec protocole d'échange d'information pour récupérer l'information réparti dans le noyau et dans différents programmes (gated, lpd, ...)
  - disponible en standard sur la plupart des systèmes
  - Mais pas de protocole général (SunNet Manager utilise RPC) et nombreuses applications non interfacées → souvent vue partielle
- Quelques logiciels 'génériques' CMU, MIT, Isode - très sensible à la version système (problème de collectes de statistiques système)

## Logiciels et stations de gestion

- Des logiciels pour accéder aux variables SNMP un à une : package perl, awkperl, logiciels CMU, MIT, Isode
- Implémentations de gérants complets — souvent intégrés, avec des bases de configuration du système, l'utilisation d'autres protocoles (RPC), d'enquêtes de configuration automatiques.
  - Surveillance et analyse : monitoring de variables

- permet de tracer de courbes de charge, l'état du réseau, ...
- Encore peu utilisé pour les interventions (sécurité faible de SNMP-V1).
- Possibilité de programmation, d'ajout de programmes spécifiques (e.g. programmation d'un hub)

## **Exemples de produits**

- NetView (IBM-Tivoli),
- OpenView (HP),
- SunNet Manager (Sun) et environnement Solstice
- SNMPc
- Pas de logiciel libre proposant une station complète, mais il existe des logiciels de monitoring basé sur SNMP, e.g. MTRG (Multi router graphic router) qui collecte des statistiques de trafic de routeur sur de longues périodes

## Autres solutions

- CMIP

Protocole OSI, vise à l'exhaustivité.

Modèle objet avec héritage, programmation par envoi d'action.

Utilisé dans plusieurs gros systèmes de gestion (e.g. TMN des télécoms)

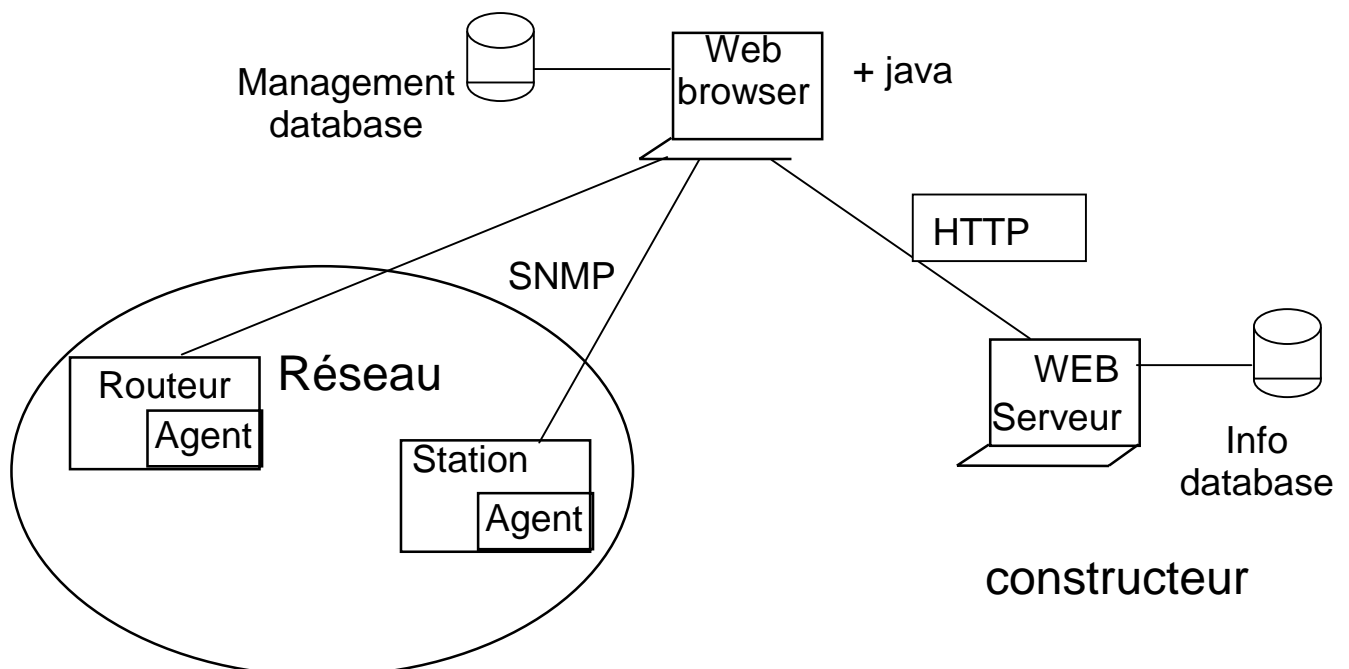
Lourd à implémenter dans les petit matériels

- DME (distributed management environnement)

Modèle de programmation distribué — permet de développer des systèmes de gestion. Devait faire partie de l'offre OSF, il existe encore des environnements commerciaux.

- Autres environnements distribués + bibliothèque d'accès SNMP : Corba, D/Com

- WEB - Java



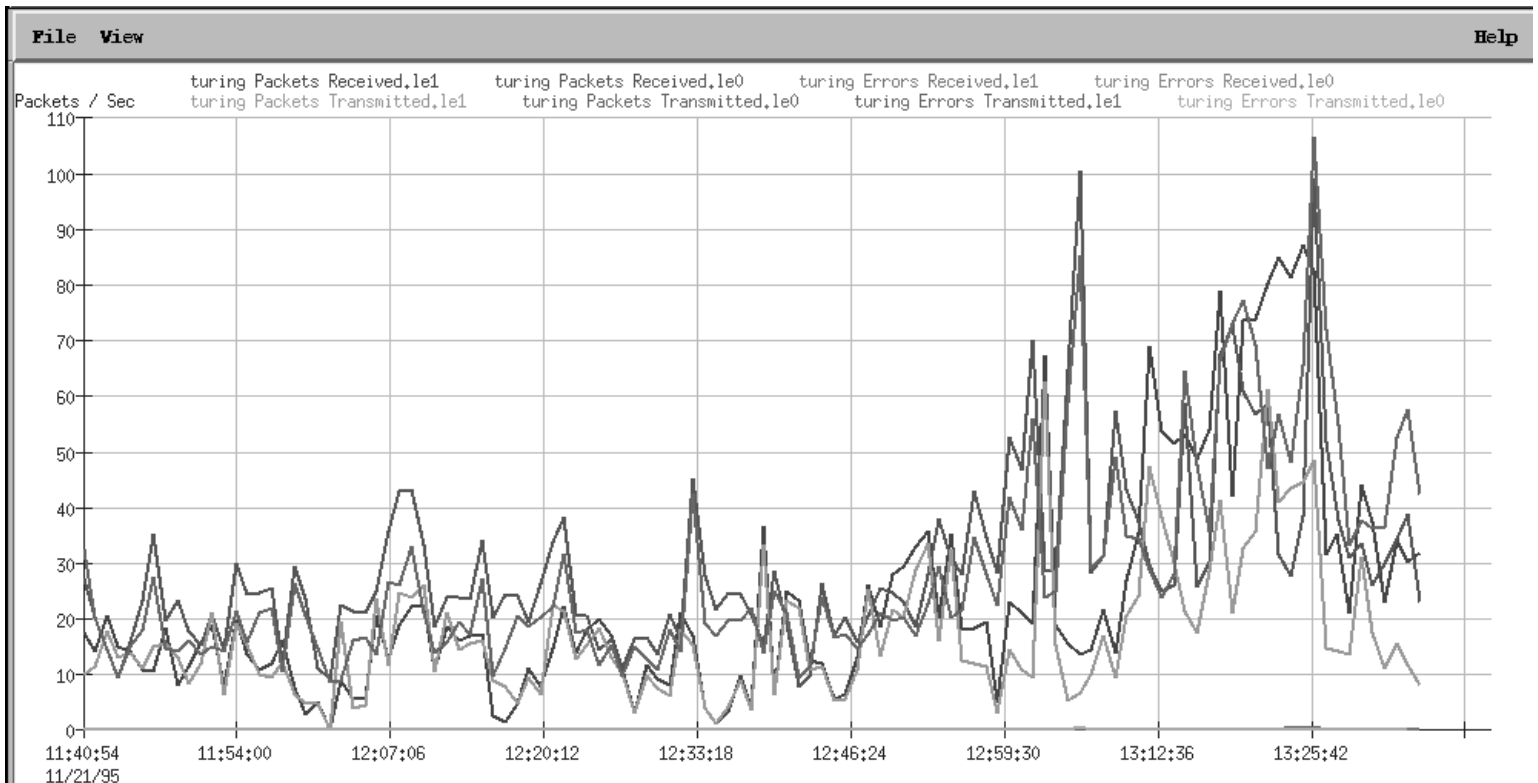
# Conclusion

- Pour la surveillance réseau : SNMP est la solution, disponible facilement.
- Reste à définir ce qu'il faut observer, les alarmes, ...
- Pour la gestion, les problèmes de sécurité gêne le déploiement. SNMPv2/3 n'est pas encore courant
- Au delà du réseau : problèmes de généralité du modèle, hétérogénéité des systèmes (e.g. qu'est-ce qu'une file d'attente d'impression) → beaucoup d'outils pour des produits, mais peu de solutions homogènes

# Le MIB browser de NV6000

Name or IP Address		Community Name	
<input type="text" value="turing.imag.fr"/>		<input type="text" value=""/>	
MIB Object ID			
<input type="text" value=".iso.org.dod.internet.mgmt.mib-2.system"/>			
<ul style="list-style-type: none"><li>sysDescr</li><li>sysObjectID</li><li>sysUpTime</li><li>sysContact</li><li>sysName</li><li>sysLocation</li><li>sysServices</li></ul>		<input type="button" value="Up Tree"/> <input type="button" value="Down Tree"/> <input type="button" value="Describe"/> <input type="button" value="Start Query"/> <input type="button" value="Stop Query"/> <input type="button" value="Graph"/>	
MIB Instance		SNMP Set Value	
<input type="text" value=""/>		<input type="text" value=""/>	
<input type="button" value="Get"/>			
MIB Values			
<p>0 : Sun SNMP Agent, SPARCStation 1+, Company Property Number 123456</p>			
Messages			
<p>Note: using community "public" for node turing.imag.fr</p>			
<input type="button" value="Close"/>		<input type="button" value="Reselect"/>	
<input type="button" value="Save As ..."/>		<input type="button" value="Help"/>	

# Suivi du trafic réseau par interface avec NV6000



## Statistiques du graphe

Line	Minimum	Average	Maximum	Last Value
turing Packets Received.le1	0.15	23.78	87.09	25.53
turing Packets Received.le0	7.97	30.67	100.54	25.59
turing Errors Received.le1	0.00	0.00	0.00	0.00
turing Errors Received.le0	0.00	0.00	0.00	0.00
turing Packets Transmitted.le1	0.15	16.44	62.58	8.27
turing Packets Transmitted.le0	8.58	30.37	106.54	42.61
turing Errors Transmitted.le1	0.00	0.08	0.63	0.00
turing Errors Transmitted.le0	0.00	0.05	0.72	0.00

Close Save Help

# Cartographie de réseau par Net View 6000

