Zürcher Hochschule Winterthur

Simple Network Management Protocol SNMP

Dr. Andreas Steffen

Definitions

- client/server network management application
- SNMP manager, SNMP agent
- SNMP management information base (MIB)

Message Formats

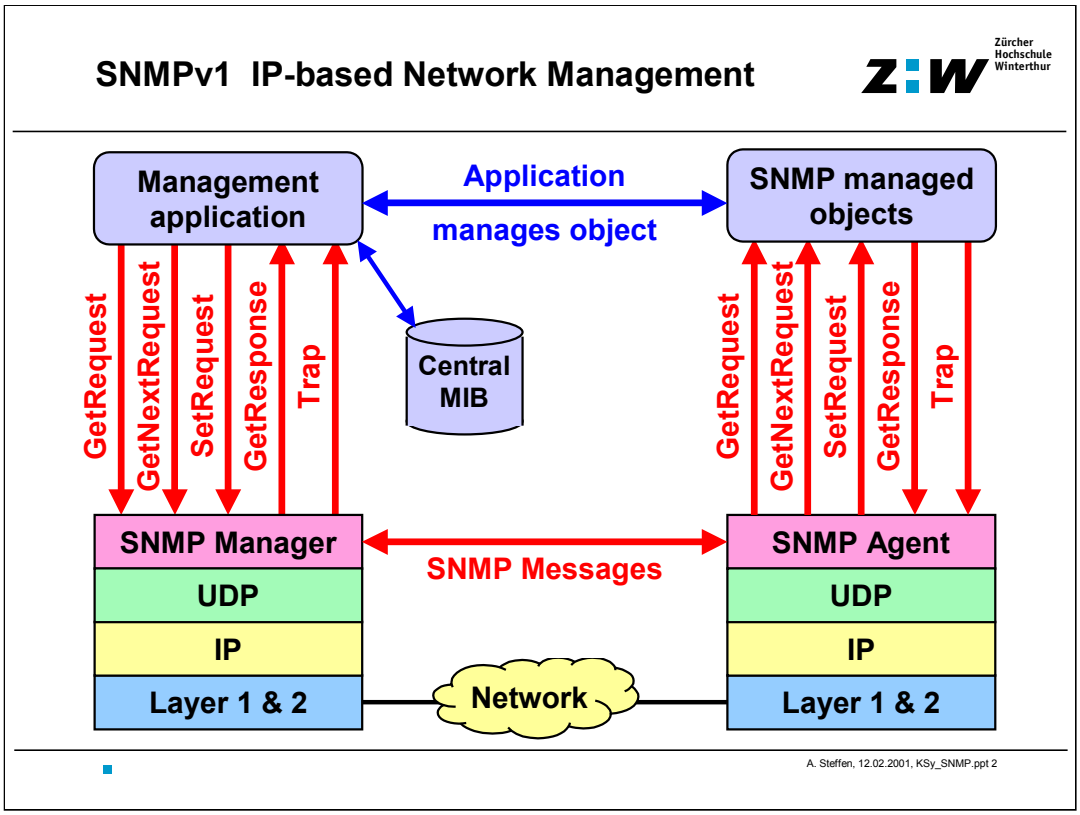
- SNMP message format
- SNMP message types – set / get / trap
- SNMP protocol data units (PDUs)
- Set- / Get-PDU format
- Variable-bindings lists
- Trap-PDU format

Object Definitions

- Predefined SNMP types
- OBJECT-TYPE macros
- Table definitions
- Instance identifiers

Standard MIB-II

- Overview



Simple Network Management Protocol (SNMP)

- SNMP is a communication protocol that has gained widespread acceptance since 1993 as a method of managing IP-based networks, including individual network devices. SNMP was developed by the IETF (Internet Engineering Task Force), and is applicable to any IP network, as well as other types of networks.
- SNMP defines a **client/server** relationship. The client program (called the **SNMP manager**) makes connections to a server program (called the **SNMP agent**) which resides on a remote network device, and serves information to the network manager regarding the device's status. On an abstract level, SNMP can be seen as a service, a **management application** makes use of, to manage **distributed objects**.
- The SNMP manager maintains a central database (called the SNMP **Management Information Base** or **MIB**) that is fed by means of queries to the SNMP agents distributed throughout the network. A MIB consists of a standard set of statistical and control values defined by various IETF RFCs and can be extended with values specific to a particular agent through the use of private or „vendor" MIBs.

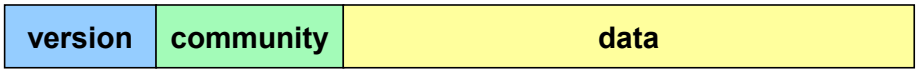
SNMP Messages

- SNMP messages are transported using the unreliable UDP protocol. This has the advantage that control connections will not hang indefinitely when an SNMP agent becomes temporarily unavailable or goes off-line altogether. On the negative side messages can get lost, especially in the critical case when networks become congested and management information is most needed.

Source: „An Introductory Overview of SNMP", diversified data resources, inc., Novato CA, http://www.ddri.com

SNMPv1 Protocol Data Units I
General Message Format (RFC 1157)

Zürcher Hochschule Winterthur




```
Message ::= SEQUENCE {
   version   INTEGER {version-1(0)},
   community OCTET STRING,
   data      PDUs
}

PDUs::= CHOICE {
   get-request      [0] IMPLICIT PDU,
   get-next-request [1] IMPLICIT PDU,
   get-response     [2] IMPLICIT PDU,
   set-request      [3] IMPLICIT PDU,
   trap             [4] IMPLICIT Trap-PDU
}
```


SNMP Message Format

- The SNMP message format is defined using the abstract syntax notation #1 (ASN.1) and encoded for transmission over UDP using the Basic Encoding Rules (BER).

SNMP Message Fields

- **version** – SNMP exists in three versions: SNMPv1 and SNMPv2, both of which are concurrently used by SNMP managers depending on the capabilities of a particular SNMP agent. SNMPv3 introduced a certain number of security concepts (secure authentication / encryption) that had been neglected in the earlier versions. Due to its complexity and experimental status SNMPv3 is still not widely used.
- **community** – The community name designates a management zone controlled by a SNMP management application. It is used primarily as a password, offering a rudimentary protection against unauthorized read/write access to SNMP information. The default values are „public“ for get requests and „private“ for set requests.
- **data** – The data part consists of a **Protocol Data Unit** (**PDU**) whose format depends on the particular SNMP message to be sent.

SNMP Message Types

- **get-request** – requests the current values of one or several objects
- **get-next-request** – fetches the next object in lexigraphical order and is used to traverse MIB tree structures and/or MIB tables.
- **get-bulk-request** (v2 only) – allows to fetch a MIB subtree or a MIB table with a single request message.
- **set-request** – used to set the values of one or several objects. Rarely used due to security reasons.
- **get-response** – contains either the result of a get-request or is used as an acknowledgement for a set-request.
- **trap** – is sent by an SNMP agent to one or several pre-configured SNMP manager when exceptional events like e.g. alarms or failures occur.

SNMPv1 Protocol Data Units II
PDU Format (RFC 1157)

Zürcher Hochschule Winterthur

request-id	error-status	error-index	variable-bindings

```
PDU ::= SEQUENCE {
  request-id        INTEGER,
  error-status      INTEGER {
                       noError    (0),
                       tooBig     (1),
                       noSuchName(2),
                       badValue   (3),
                       readOnly   (4),
                       genErr     (5)
                    },
  error-index       INTEGER,
  variable-bindings VarBindList
}
```


PDU Format

- **request-id** – used by an SNMP manager to correlate incoming responses with outstanding requests. In cases where an unreliable datagram service is being used, the request-id also provides a simple means of identifying messages duplicated by the network.
- **error-status** – a non-zero instance of error-status in a get-response is used to indicate that an exception occurred while processing a request.
- **error-index** – in case of an non-zero error-status, the error-index variable may provide additional information by indicating which variable in the variable-bindings list caused the exception.
- **variable-bindings** - a variable binding, or **VarBind**, refers to the pairing of the name of a MIB variable to the variable's value. A **VarBindList** is a simple list of variable names and corresponding values. Some PDUs are concerned only with the name of a variable and not its value (e.g., the GetRequest-PDU). In this case, the value portion of the binding is ignored by the protocol entity. However, the value portion must still have valid ASN.1 syntax and encoding. It is recommended that the ASN.1 value NULL be used for the value portion of such bindings.

Source: RFC 1157 „A Simple Network Management Protocol (SNMP)“

SNMPv1 Protocol Data Units III
readOnly error-code got lost in RFC 1157

Zürcher Hochschule Winterthur

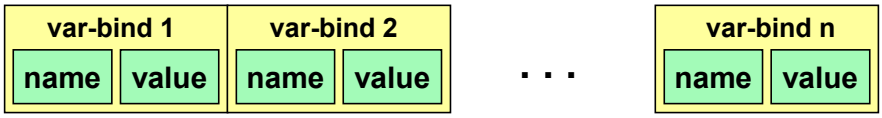
```
(1) If, for any object named in the variable-
    bindings field, the object is not available
    for set operations in the relevant MIB view,
    then the receiving entity sends to the
    originator of the received message the
    GetResponse-PDU of identical form, except
    that the value of the error-status field is
    noSuchName, and the value of the error-index
    field is the index of said object name
    component in the received message.

(2) ... badValue ...

(3) ... tooBig ...

(4) ... genErr ...

    -- SNMPv2 introduces notWritable
```


„readOnly“ error code not used due to editor‘s error in RFC 1157

- When a set-request tries to overwrite a read-only MIB variable, the resulting get-response is expected to contain a **readOnly(4)** error-code. Instead the **noSuchName(2)** error-code is usually returned by the SNMP agent. The reason for this strange behaviour is due to a mishap that occurred in the final draft for RFC 1157. The paragraph describing the use of the „readOnly“ error-code got inadvertently deleted by the editor. After acceptance of RFC 1157 this could not be corrected any more, so the error-code „noSuchName“ was selected as a work-around value.
- SNMPv2 introduced a new error-code **notWritable(17)** to cover the readOnly error case.

SNMPv1 Protocol Data Units IV
Variable-Bindings Format (RFC 1157)

Zürcher Hochschule Winterthur




```
VarBindList  ::= SEQUENCE OF VarBind

VarBind      ::= SEQUENCE {
   name  ObjectName,
   value ObjectSyntax
}

ObjectName   ::= OBJECT IDENTIFIER

ObjectSyntax ::= CHOICE {
   simple           SimpleSyntax,
   application-wide ApplicationSyntax
}
```


VarBindList

- A VarBindList is an ASN.1 sequence containing a variable number of variable-bindings.

VarBind

- A variable-binding or VarBind is an object name / object value pair. The object name is specified using an ASN.1 **object identifier** (**OID**) and the corresponding object value is either of a **simple type** or an **application-specific type**.

SNMPv1 Protocol Data Units V
Predefined SNMP Types (RFC 1155)

Zürcher Hochschule Winterthur

```
SimpleSyntax ::= CHOICE {
   number INTEGER,
   string OCTET STRING,
   object OBJECT IDENTIFIER,
   empty  NULL
}
ApplicationSyntax::= CHOICE {
   address   NetworkAddress,
   counter   Counter,
   gauge     Gauge,
   ticks     TimeTicks,
   arbitrary Opaque
}
NetworkAddress::= CHOICE {
   internet  IpAddress
}
```


Predefined SNMP Types – Simple Types

- SNMP uses a subset of the available ASN.1 simple types. Allowed are
  - INTEGER [UNIVERSAL 2]
  - OCTET STRING [UNIVERSAL 4]
  - OBJECT IDENTIFIER [UNIVERSAL 6]
  - NULL [UNIVERSAL 5]

Predefined SNMP Types – Constructed Types

- SNMP uses only the SEQUENCE and SEQUENCE OF constructed types, making parsing at the receiver much easier by banning the SET and SET OF constructs
  - SEQUENCE [OF] [UNIVERSAL 16]

Predefined SNMP Types – Application-Wide Types

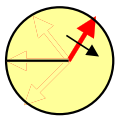
- RFC 1155 defines the following application-wide types:
  - IpAddress [APPLICATION 0]
  - Counter [APPLICATION 1]
  - Gauge [APPLICATION 2]
  - TimeTicks [APPLICATION 3]
  - Opaque [APPLICATION 4]

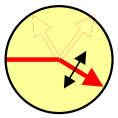
SNMPv1 Protocol Data Units VI

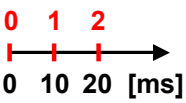
Application-wide SNMP Types (RFC 1155)

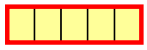
Zürcher Hochschule Winterthur

```
IpAddress ::= [APPLICATION 0] IMPLICIT
                  OCTET STRING (SIZE (4))
```
160.85.128.1
A0 55 80 01

```
Counter   ::= [APPLICATION 1] IMPLICIT
                  INTEGER (0..4294967295)
```
0  2^{31}

```
Gauge     ::= [APPLICATION 2] IMPLICIT
                  INTEGER (0..4294967295)
```
0  2^{31}

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



```
Opaque    ::= [APPLICATION 4] IMPLICIT
                  OCTET STRING
```


SNMPv1 Protocol Data Units VII
Trap-PDU Format (RFC 1157)

Zürcher Hochschule Winterthur

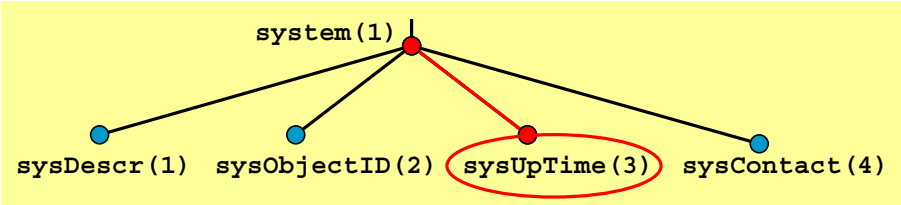
```
Trap-PDU ::= SEQUENCE {
    enterprise          OBJECT IDENTIFIER,
    agent-addr          NetworkAddress,
    generic-trap        INTEGER {
                            coldStart             (0),
                            warmStart             (1),
                            linkDown              (2),
                            linkUp                (3),
                            authenticationFailure (4),
                            egpNeighborLoss       (5),
                            enterpriseSpecific    (6)
                        },
    specific-trap       INTEGER,
    time-stamp          TimeTicks,
    variable-bindings   VarBindList
}   -- enterprise OID equals sysObjectID
```


Trap-PDU Format

- **enterprise** – contains value of the standard MIB object **sysObjectID** uniquely identifying the manufacturer and model of the managed network entity that generated the trap.
- **agent-addr** – Network address of the network entity that generated the trap.
- **generic-trap** – any of the standard trap causes listed above.
- **specific-trap** – details the trap cause in case of an **enterpriseSpecific(6)** trap.
- **time-stamp** - time elapsed between the last (re)initialization of the network entity and the generation of the trap.
- **variable-bindings** - The significance of the variable-bindings component of the Trap-PDU is implementation-specific.

SNMPv2 Defining Objects
The OBJECT-TYPE Macro (RFC 1212)

Zürcher Hochschule Winterthur




```
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 }
```

ObjectSyntax

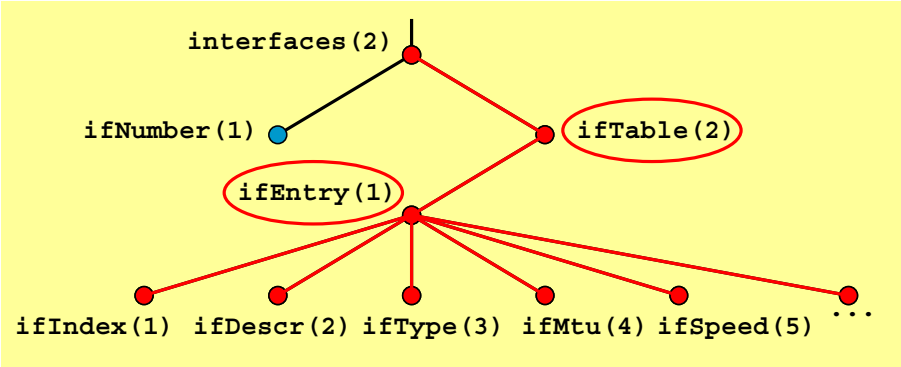
ObjectName

ASN.1 Macros

- In order to facilitate the use of tools for processing the definition of the MIB, the OBJECT-TYPE macro is used for specifying MIB objects. This macro permits the key aspects of an object type to be represented in a formal way.

SNMPv2 Defining Tables I
Table of Network Interfaces (RFC 1213)

Zürcher Hochschule Winterthur



- `ifTable` — head of table
- `ifEntry` — rows of table
- `ifIndex, ifDescr, etc` — columns of table
- `ifNumber` — number of rows

MIB Table Objects

- Two-dimensional table objects must be used if several instances of an object must be managed. Since ASN.1 object identifiers are organized as nodes in a hierarchical tree. **Table rows** representing the **object instances** are the leaves attached below a node defining the **table column** standing for a MIB **object type**.
- A typical example for a MIB table object are the parameters of a variable number of network interfaces.

SNMPv2 Defining Tables II
Definition of objects `ifTable` and `ifEntry`

Zürcher Hochschule Winterthur

```
ifTable OBJECT-TYPE
      SYNTAX  SEQUENCE OF IfEntry
      ACCESS  not-accessible
      STATUS  mandatory
      DESCRIPTION
        "A list of interface entries"
::= { interfaces 2 }

ifEntry OBJECT-TYPE
      SYNTAX  IfEntry
      ACCESS  not-accessible
      STATUS  mandatory
      DESCRIPTION
        "An interface entry containing objects"
      INDEX { ifIndex }
::= { ifTable 1 }
```


Elements of a Table

- **ifTable** – is the **root** node of the table.
  It cannot be directly accessed.
- **ifEntry** – contains the **column definition** of the table and defines how the **index** is formed to access a **specific row** or object instance.
  It cannot be directly acces

SNMPv2 Defining Tables III
Definition of type `IfEntry`

Zürcher Hochschule Winterthur

```
IfEntry ::= SEQUENCE {
        ifIndex   INTEGER,
        ifDescr   DisplayString,
        ifType    INTEGER,
        ifMtu     INTEGER,
        ifSpeed   Gauge,
        ...
        ...
}
```


Elements of a Table – Column Definition

- The variables contained in the sequence defined by the **IfEntry type** define the names and types of the **table columns**.

SNMPv2 Defining Tables IV
Definition of column objects

Zürcher Hochschule Winterthur

```
ifIndex OBJECT-TYPE
   SYNTAX  INTEGER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
      "A unique value for each interface, ranging
       between 1 and the value of ifNumber."
::= { ifEntry 1 }

ifDescr OBJECT-TYPE
   SYNTAX  DisplayString (SIZE (0..255))
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
     "A textual string containing information
     about the interface."
::= { ifEntry 2 }
```

SNMPv2 Defining Tables V
Instance Identifiers for `ifTable` cells

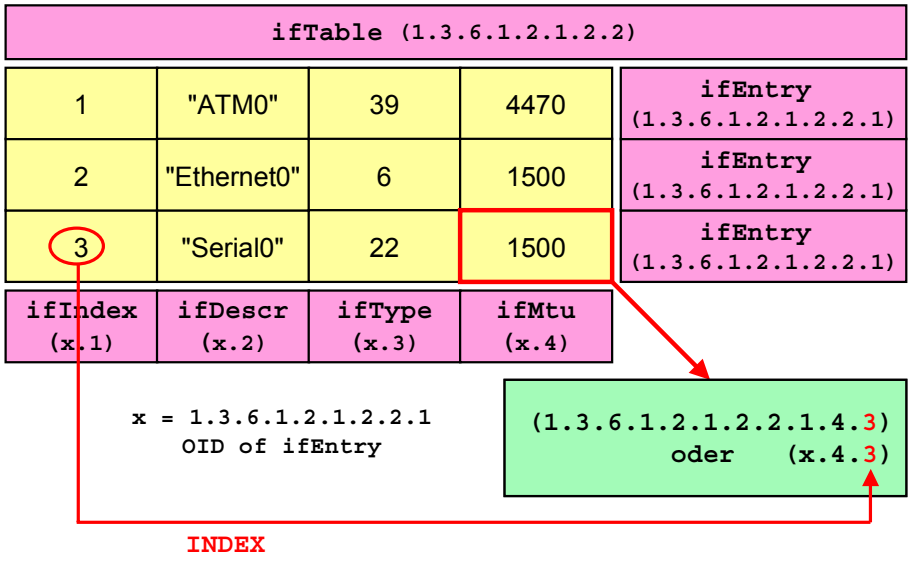


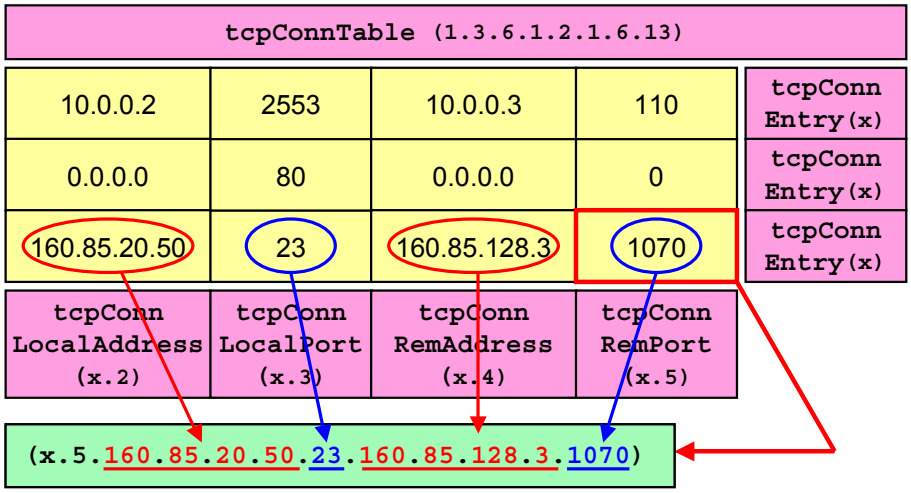
Table Access using Instance Identifiers

- The generic table addressing structure in order to access a particular instance of a column object is

<OID of table root>.<table column definition>.<relative column OID>.<row index>

`1.3.6.1.2.1.2.2 . 1 . 4 . 3`

SNMPv2 Defining Tables VI
Instance Identifiers for `tcpConnTable` cells



x = 1.3.6.1.2.1.6.13.1
OID of tcpConnEntry

Composite Instance Identifiers

- Instead of using a dedicated index column object acting as a „primary key“, the index identifying a particular instance can also be formed by a **concatenation** of the instance values of several column objects.

SNMPv2 Defining Tables VII
Definition of object `tcpConnEntry`

Zürcher Hochschule Winterthur

```
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 }
```


Composite Index Definitions

- This example of a table listing all current TCP/IP connections uses the local and remote network addresses and ports to uniquely identify a particular connection instance.

SNMP v2 Standard MIB-II (RFC 1213)

Zürcher Hochschule Winterthur

- system (1) : overall information about the system
- interfaces (2) : information about each system interface
- at (3) : description of address-translation table (deprecated)
- ip (4) : information related to IP on this system
- icmp (5) : information related to ICMP on this system
- tcp (6) : information related to TCP on this system
- udp (7) : information related to UDP on this system
- egp (8) : information related to EGP on this system
- transmission (10) : information about transmission schemes and access protocols at each system interface
- snmp (11) : information related to SNMP on this system