Unit 2

Internet based Network Management
In the Internet world Network management refers to ‘management of networks, devices, and hosts’.

Achieved through 'Simple Network Management Protocol’ which specifies:
- Management operations (minimal)
- Structure of Management Information (SMI)
- Pre-defined managed objects (MIB)
SNMP

• Virtually all major vendors of host computers, workstations, bridges, routers, and hubs offer basic SNMP

• SNMP has been extended with the specification of RMON (Remote Monitoring) which provides the ability to monitor subnetworks as a whole rather than just individual devices

• SNMP v2 and v3 have been developed to address the shortcomings of SNMP when applied to larger, more sophisticated networks
Basic Concepts

- Model of Network Management for TCP/IP networks include the following elements
  - Management Station
  - Management agent
  - Management Information Base
  - Network Management Protocol
Management Station

Typically stand alone device (but capability may be implemented on a shared system)

• Contains:
  1. Set of management applications for analysis, fault recovery, etc
  2. Interface by which Network Manager may monitor & control the network
  3. Capability of translating network manager's requirements into actual monitoring and control of remote elements in the network
  4. A database of information extracted from the MIBs of all the managed entities in the network

Note: Only 3 & 4 are covered by the SNMP standard
Management Agent

- Computing platforms (hosts), bridges, routers, switches, hubs are equipped with SNMP agents
- Agents respond to requests for information and actions from the management station
- May provide asynchronous unsolicited information (via traps)
- Most processing done in management station - agent tends to have minimal functionality
Management Information Base

• Resources managed by representing them as Objects
• SNMP object more like ISO attributes (i.e. are not complex & do not exhibit inheritance properties)
• Each object is essentially a data variable that represents one aspect of the managed agent
• The collection of these objects is called the Management Information Base
• Specifies a Common set of objects used to manage each particular types of equipment e.g. bridge, router etc.
Managers are linked to agents via a SNMP protocol which has the following capabilities:

- **get**: enables the management station to retrieve the value of objects at the agent
- **set**: enables the management station to set the value of objects at the agent
- **trap**: enables an agent to notify the management station of significant events
- **getbulk**: allows efficient retrieval of large amount of data (many table rows) \([v2+]\)
- **inform**: trap + acknowledgement from management station \([v2+]\)
Network Management Protocol Architecture

Manager Process
- SNMP
- UDP
- IP
- Network dependent protocols

Agent Process
- User Processes
  - SNMP
  - FTP etc..
  - UDP
  - TCP
  - IP
  - Network dependent protocols

User Processes
- TCP
- FTP etc..

Central MIB

Internetwork
Management Station - Agent Interaction

Management Application

SNMP Manager

GetRequest
GetNextRequest
SetRequest
GetResponse
Trap

UDP
IP
Network dependent protocols

Managed Resources

SNMP Managed Objects

GetRequest
GetNextRequest
SetRequest
GetResponse
Trap

UDP
IP
Network dependent protocols

network
Trap Directed Polling

• Excessive polling of SNMP agents is discouraged as networks not intended to carry excessive management information and agents not designed to respond to frequent requests
• Trap directed Polling Strategy:
  – Management station polls agents for Management information (e.g. performance stats) at initialization time or at infrequent intervals
  – Management station then refrains from further polling
  – Agent responsible for notifying Management station of unusual event(s)
  – Management Station then does directed poll to agent originating event report or neighbouring agents to determine problem
# Proxies

<table>
<thead>
<tr>
<th>Management Station</th>
<th>Proxy Agent</th>
<th>Proxied Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manager Process</td>
<td>Agent Process</td>
<td>Management Process</td>
</tr>
<tr>
<td>SNMP</td>
<td>SNMP</td>
<td>Protocol architecture used by proxied device</td>
</tr>
<tr>
<td>UDP</td>
<td>UDP</td>
<td>Protocol architecture used by proxied device</td>
</tr>
<tr>
<td>IP</td>
<td>IP</td>
<td>Network dependent protocols</td>
</tr>
<tr>
<td>Network dependent protocols</td>
<td>Network dependent protocols</td>
<td>Network dependent protocols</td>
</tr>
</tbody>
</table>

**Mapping function**

- Protocol architecture used by proxied device
- Network dependent protocols
SNMP Management Information

- In SNMP, the Management Information Base (MIB) is a structured collection of objects representing a managed resource.
- As with CMIP (managed objects), the following must be supported:
  - Managed objects used to represent a particular resource must be the same at each system.
  - There must exist a common scheme for representation of managed objects (in order to support interoperability across different equipment vendors).
Structure of Management Information

• ‘Structure of Management Information' (SMI) defines general framework within which MIBs can be defined and constructed (RFC 1155)
• SMI identifies data types usable in a MIB and specifies how resources within a MIB are represented & named
  SNMP ‘philosophy’ is to be simple & easy to implement
    => SMI only supports simple data types
    e.g. scalars and two-dimensional arrays (tables)
• SMI provides:
  – standardised technique for definition structure of a particular MIB
  – standardised technique for definition individual objects, (including object syntax & values)
  – standardised technique for encoding object values
MIB Structure

• Managed Objects arranged in hierarchical structure to allow object type identification

• Leaf objects of the tree are the actual managed objects, each of which represent some resource, activity or related information to be managed

• The tree structure itself defines a grouping of objects into logically related sets

• Each object type has an Object Identifier to uniquely identify object

• Because of the value associated with the type (i.e. object identifier), the naming convention also serves to identify the structure of object types
MIB Hierarchy for Object Identification
MIB Object Syntax

• Each object defined in formal way (using ASN.1)
• Definition of object syntax includes the specification of the data type of the object, its allowable forms and value ranges, & its relationship to other objects within the MIB
• Data types supported in MIBs are:
  – Integer
  – Octet string
  – NULL
  – Object identifier
  – Sequence, sequence of
  – Application-wide data types
Application-wide data types

- **network address**: currently only one specified, namely ipaddress (a 32 bit address)
- **counter**: (or rollover counter) is a non negative integer that may be incremented but not decremented. Max values of $2^{32-1}$. When counter reaches its maximum, it wraps around & starts increasing again from zero
- **gauge**: a non negative integer that can increase & decrease, with maximum value of $2^{32-1}$. If max reached, gauge remains latched at that value until reset
- **timeticks**: a non negative integer counts the time in hundredths of a second since some epoch. When object defined in MIB, epoch is identified e.g. start-up or re-initialization. Is NOT supported by time sync. Protocol.
- **opaque**: supports the ability to pass arbitrary data. Data is encoded as octet string for transmission. Data format itself may be defined by another syntax type.
Defining Objects in a MIB

- Each managed object has an object type (defined in ASN.1)
- Key components of an Object Type:
  - **Syntax**: must resolve to an instance of one of the object Syntaxes mentioned earlier
  - **Access**: defines the way in which an instance of an object may be accessed (via SNMP). Specifies the minimum level of support required for that object type. Options are *read-only*, *read-write*, *write-only* & *not-accessible*
  - **Status**: Indicates implementation support for object (mandatory or optional or deprecated). Deprecated is one which must be supported but which will most likely be removed from the next version of the MIB
  - **Description**: a textual description of the semantics of the object type (optional)
More Key components of an Object Type

Example Object Type definition:

tcpMaxConn OBJECT-TYPE
SYNTAX INTEGER
ACCESS read-only
STATUS mandatory
DESCRIPTION
“The limit on the total number of TCP connections the entity can support. In entities where the maximum number of connections is dynamic, this object should contain the value -1”
::= {tcp 4}
Other elements of Object Type definition

- **ReferPart**: a textual cross reference to an object defined in some other MIB module (optional)
- **IndexPart**: used in defining tables. This clause may be present only if the object type corresponds to a conceptual row.
- **DefValPart**: defines acceptable default value that may be used when an object instance is created (at the discretion of the agent). (optional)
- **Value Notation**: indicates the name used to access the object via SNMP
Defining Tables

• SMI supports only one form of structured data: a simple two-dimensional table

• Definition of tables involves using sequence and sequence of ASN.1 types and IndexPart of the Object-Type definition

Example:

• Consider the TcpConnectionEntry managed object which contains information about a particular current TCP connection and contains tcp connection state, tcp connection local address, tcp connection local port, tcp connection remote address, tcp connection remote port

• A management station maintains a table of such TCP connection entries. Thus each entry in this table (tcpConnTable) represents the state information stored in the managed station for one connection.
Example cont.

- The overall table consists of a sequence of (zero or more) TcpConnEntry.
- Each row consists of a sequence that includes five scalar elements (Integer, Ipaddress, Integer, Ipaddress, Integer).
- The Index component determines which object value(s) will be used to distinguish each row in the table.
- Note: Nesting of tables (i.e. an entry in a table is itself a table) is NOT allowed in SNMP.
Caveats and Limitations of SNMP MIBs

- Evidence has shown that although vendors claim to support SNMP agents (and managed objects), their accuracy of measurements can be inconsistent.
- There are limitations in using SNMP MIBs e.g. sophistication & representative power for expressing managed objects.
- MIB-II (RFC 1212) provides a ‘improved’ and more concise set of MIB definition from the original MIB definitions (RFC 1155)
Simple Network Management Protocol (SNMP) v1

- SNMP only supports the alteration and inspection of variables (objects)
- **Get**: a Management station retrieves a scalar object value from a managed station (i.e. the agent)
- **Set**: a Management station updates a scalar object value in a managed station (i.e. the agent)
- **Trap**: a managed station (i.e. the agent) sends an unsolicited scalar object value to a management station
- Not supported:
  - actions
SNMP v1 Agents and Security

• Many Management stations may access/manage an agent
  => each agent must control the use of that MIB by one (or more) management stations
  – Agent control consists of the following security issues:
    • Authentication service: agent can limit access to MIB to authorised management stations
    • Access Policy: agent controls different access privileges to different management stations
    • Proxy service: agent may act as proxy to other managed stations (i.e. may implement authentication service and access policy for other managed systems on proxy system)
Authentication Service

- Ensures the recipient that the received message is from the source from which it claims to be
- SNMP v1 authentication is trivial!
- Every (Get or Set) request from a management station includes a community name
- Message is assumed authentic if the sender knows the appropriate community name (for management station/agent pairing)

=> Community Name used as a password between management station and agent!
SNMP, Security and Communities

• SNMP only provides limited security based on concept of community
• **SNMP Community**: relationship between SNMP agent and set of SNMP managers that defines authentication, access control and proxy characteristics
• Each community has a unique name (within the agent) termed ‘community name’
• Agent may establish a number of communities with overlapping management station memberships
• Same name may be used by different agents
=> management station must keep track of community name(s) associated with each agent
SNMP V2

• SNMP V1 only provides limited security based on concept of community
• SNMP V2 went through several revisions to provide new security and access control models
  – SNMP V2: Changes to PDU formats, and other extensive protocol changes. This was largely rejected by the community
  – SNMP V2c: Community model – lighter-weight revision of SNMPv1
  – SNMP V2u: User model – user-oriented model, adapted and adopted
• SNMP v3: Authentication, Privacy, Access Control
Access Policy

• Agent typically limits access to its MIB to a selected set of management stations

• Agent can provide different categories of MIB access to different management stations by:
  – limiting the management station(s) view of the MIB to a specific subset of objects
  – limiting the access mode for the community (either read-only or read-write) and is applied to all objects in a communities MIB view

• Combination of MIB view and access mode is referred to as SNMP ‘profile’
## Relationship between MIB access category and SNMP access Mode

<table>
<thead>
<tr>
<th>MIB Access Category</th>
<th>SNMP Access Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-only</td>
<td>READ-ONLY</td>
</tr>
<tr>
<td>Read-write</td>
<td>Available for get and trap operations</td>
</tr>
<tr>
<td>Write-only</td>
<td>Available for get &amp; trap operations, but the value is implementation specific</td>
</tr>
<tr>
<td>Not accessible</td>
<td>Unavailable</td>
</tr>
<tr>
<td></td>
<td>AVAILABLE FOR GET, SET &amp; TRAP OPERATIONS, BUT THE VALUE IS IMPLEMENTATION SPECIFIC FOR GET AND TRAP OPERATIONS</td>
</tr>
</tbody>
</table>
Proxy Service

• SNMP can act as a proxy for network devices which either do not support SNMP or TCP/IP protocol or for devices which need proxy service to minimise interaction between device and management station

• For each device that the proxy system represents, it maintains an SNMP access policy

=> proxy knows which MIB objects can be used to manage the proxied system (MIB view) and their access mode
Identifying Object Instances

• Can identify an Object Type by using the object identifier (defined by position of the object in the MIB tree),

• How do we identify object instances?

• Two kinds of object instances: columnar objects (i.e. objects that occur in a table) and scalar objects

• For columnar objects, two techniques for identifying instances
  – serial-access - based on lexicographic ordering of instances in MIB
  – random access - identification of a specific columnar object in a table
Random Access technique

- When a table is defined, one or more objects which make up the elements of a row are specified as INDEX objects
- these INDEX objects are used to distinguish one row from another
- Combination of object identifier for a columnar object and one set of values of the INDEX objects specifies a particular scalar object in a particular row of a table
- SNMP concatenates the scalar object identifiers with the values of the INDEX objects, listed in the order in which the INDEX objects appear in the table definition
Example of Random Access technique

Remember the tcpConnTable which had identifier 1.3.6.1.2.1.6.13
Then to identify a row in the table
x.i.(tcpConnState).(tcpConnLocalAddress).(tcpConnLocalPort).
   (tcpConnRemAddress).(tcdConnRemPort)

Where x = 1.3.6.1.2.1.6.13.1 == tcpConnEntry
i = the last subidentifier in the object identifier for
   a columnar object (i.e. its position within the table)

   _basics.html

e.g. x.i.(listen(2)).(10.0.0.99).(21).(9.1.2.3).(0)
i = 1/2/3/4/5
Converting Object instance values into subidentifiers

For each INDEX object instance:

• integer-valued: A single subidentifier takes the integer value (valid only for nonnegative integers)
• string-valued, fixed length: each octet of the string is encoded as a separate subidentifier, for a total of n subidentifiers for a string of length n octets
• string-valued, variable length: for a string of length n octets, the first subidentifier is n; this is followed by each octet of the string encoded as a separate subidentifier, for a total of n+1 subidentifiers
• object-identifier-valued: for an object identifier with n subidentifiers, the first subidentifier is n, this is followed by the value of each subidentifier in order, for a total of n+1 subidentifiers
• Ipaddress-valued: these are four subidentifiers, in the familiar a.b.c.d notation
Ambiguities in row identification

- Theoretically ambiguities in identifying a row should not occur as INDEX object values should unambiguously distinguish a conceptual row.
- However in MIB I this is not always the case.
- However in MIB II any new table objects conform to the definition of INDEX and any existing tables which suffer from this ambiguity are deprecated.
Table and Row Identifiers

- For table and row objects no instance identifier is defined because they are not leaf objects and therefore are not accessible by SNMP.
- Such non leaf object are defined as not accessible.
Scalar Objects

• There is no ambiguity between an object type and an instance of that type since there is only one object instance for each scalar type

• however, for consistency, (& to distinguish between object type and object instance) an instance identifier of a nontabular scalar object consists of its object identifier concatenated with 0
Lexicographic Ordering

- Ordering of the components of an object identifier can be generated by traversing the tree of object identifiers in the MIB.
- This ordering extends to object instance identifiers since an object instance identifier is also a sequence of integers.
- This ordering can be used by a management station which does not know the exact makeup of the MIB view supported by an agent.
- In this case the management station needs a way of searching for and accessing objects without specifying their name.
Lexicographic Ordering (cont.)

• By using lexicographic ordering, the management station can supply an object or object instance identifier and then ask for the object instance that occurs next in the ordering.

i.e. given a particular position in the MIB tree the management station can then traverse the MIB with a sequence of get-next retrieval operations.
SNMP Protocol Specification

- Each SNMP message includes a version number (version of SNMP protocol), a community name, a SNMP Protocol Data Unit

- SNMP PDU are:
  - GetRequest PDU
  - GetNextRequest PDU
  - SetRequest PDU
  - GetResponse PDU
  - Trap PDU
### SNMP Format [v1]

#### SNMP Message

<table>
<thead>
<tr>
<th>Version</th>
<th>Community</th>
<th>SNMP PDU</th>
</tr>
</thead>
</table>

#### GetRequest, GetNextRequest & SetRequest PDU

<table>
<thead>
<tr>
<th>PDU Type</th>
<th>Request id</th>
<th>Variable bindings</th>
</tr>
</thead>
</table>

#### GetResponse PDU

<table>
<thead>
<tr>
<th>PDU Type</th>
<th>Request id</th>
<th>Error-status</th>
<th>Error-index</th>
<th>Variable bindings</th>
</tr>
</thead>
</table>

#### Trap PDU

<table>
<thead>
<tr>
<th>PDU Type</th>
<th>enterprise</th>
<th>Agent address</th>
<th>Generic trap</th>
<th>specific trap</th>
<th>Time stamp</th>
<th>Variable bindings</th>
</tr>
</thead>
</table>

#### Variable bindings

<table>
<thead>
<tr>
<th>name1</th>
<th>vale1</th>
<th>name2</th>
<th>value2</th>
<th>.....</th>
<th>Name n</th>
<th>Value n</th>
</tr>
</thead>
</table>
Steps in transmission of SNMP Message

1. Construct PDU (using ASN.1 definitions)
2. Pass PDU to authentication service with source and destination transport addresses and a community name.
   => Authentication service performs required transformations for exchange e.g. encryption or inclusion of an authentication code and returns result
3. Construct a message consisting of a version field, community name and result from step (2)
4. New ASN.1 object is encoded (using basic encoding rules) and passed to transport service
Steps in reception of SNMP Message

1. Perform basic syntax check of message & discard message if it fails to parse
2. Verify version number & discard if mismatch
3. Pass user name, PDU portion of message and source & destination transport addresses to authentication service
   - If authentication fails, generate a trap & discard message
   - If authentication succeeds, return PDU in form of an ASN.1 object back from authentication service
4. Do basic syntax check of PDU & discard if fails to parts. If parses, then select named community and appropriate SNMP access policy and process PDU accordingly

Note: In practice, authentication service merely used to verify that community name authorises receipt of messages from the source SNMP entity
Variable Bindings

• All SNMP operations involve access to (scalar) object instance
• It is possible to group a number of SNMP operation of the same type (get, set, trap) into single SNMP message to reduce network burden
• To allow this, all SNMP PDUs include variable bindings i.e. a sequence of references to object instances together with the value of those instances
• If PDU only concerned with name of object instance then value entry(s) are NULL
SNMP PDU Interactions
GetRequest PDU

Contains

- PDU Type: indicates that it is a GetRequest PDU
- request-id: number assigned by sending entity to correlate outstanding requests with responses to the same agent. Also enables sending entity to cope with duplicate responses
- variable bindings: lists of object instances whose values are requested
• GetRequest is atomic => if any one of the values requested cannot be retrieved then no values are sent to requesting entity
• Error conditions:
  1. object named does not match any object identifier in relevant MIB or named object does not have instance value (non leaf object) => ‘noSuchName’ error returned
  2. Size of object value list exceed local (agent) limitations => ‘tooBig’ error returned
  3. Not able to supply a value for at least one of the objects for some other reason => ‘genErr’ error returned

In cases 1 & 3 errorIndex is set to the index of the value in the variable bindings which caused the problem
GetRequest PDU (cont 2)

- Rules for responding to GetRequest place burden on the network management station to be clever in the use of this operation.
- If network management station requires numerous values, then it's desirable to ask for a large number of values in a single PDU.
- If a response is not possible (for even one of the objects) or if the response to all objects is too big for a single GetResponse PDU => no information returned!
- **Note:** SNMP protocol does not state that request-ids from a management station should be monotonically increasing - this is left up to implementation choices.
Set Request PDU

- Similar to getRequest (same exchange pattern & format)
- Used to write an object values
- Variable bindings in setRequest contain the object identifier(s) for the object(s) being written to as well as the new value(s) being written
- If even one of the object values cannot be written (for some reason) then no object values are written
Example of Set Request

If manager issues the request `SetRequest (ipRouteMetric1.9.1.2.3 = 9)`
Agent will return (if successful) `GetResponse (ipRouteMetric1.9.1.2.3 = 9)`

<table>
<thead>
<tr>
<th>ipRouteDest</th>
<th>ipRouteMetric1</th>
<th>ipRouteNextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1.2.3</td>
<td>3</td>
<td>99.0.0.3</td>
</tr>
<tr>
<td>10.0.0.51</td>
<td>5</td>
<td>89.1.1.42</td>
</tr>
<tr>
<td>10.0.0.99</td>
<td>5</td>
<td>89.1.1.42</td>
</tr>
</tbody>
</table>

If manager issues the request `SetRequest (ipRouteMetric1.9.1.2.3 = 9)`
Agent will return (if successful) `GetResponse (ipRouteMetric1.9.1.2.3 = 9)`

<table>
<thead>
<tr>
<th>ipRouteDest</th>
<th>ipRouteMetric1</th>
<th>ipRouteNextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1.2.3</td>
<td>9</td>
<td>99.0.0.3</td>
</tr>
<tr>
<td>10.0.0.51</td>
<td>5</td>
<td>89.1.1.42</td>
</tr>
<tr>
<td>10.0.0.99</td>
<td>5</td>
<td>89.1.1.42</td>
</tr>
</tbody>
</table>
Example 2: Set Request

- Suppose management wishes to add a new row to the table (ipRoutDest1 = 11.3.3.12, ipRouteMetric1 = 9 and ipRouteNextHop = 91.0.0.5)

SetRequest (ipRoutDest. 11.3.3.12 = 11.3.3.12, ipRouteMetric1.11.3.3.12 = 9 and ipRouteNextHop . 11.3.3.12 = 91.0.0.5)

- Agent could:
  - reject the operation (noSuchName)
  - attempt to create the new object instances (but perhaps finds one of the assigned values is inappropriate due to syntax or value and returns error ‘badValue’)
  - accept operation and create a new row (table now has four rows)
Row Deletion

- Set command can also be used to delete a row of a table (logically) by setting a value in that row e.g.
  - SetRequest (ipRouteType.7.3.5.3 = invalid)
  - The effect is to (logically) eliminate the row of the table indexed by an ipRouteDest value 7.3.5.3 (whether row is actually deleted from agent MIB is an implementation specific issue)
Performing an Action

- No specific support for issuing an action on an agent
- However, can implement agent so that it will carry out a specific operation if an object is set to a specific value
TRAP PDU

- Provides management station(s) with an asynchronous notification of some significant error
- PDU type: indicating that this is a Trap PDU
- agent-addr: Ip address of the object generating the trap
- generic-trap: one of the predefined trap types
- specific-trap: indicates more specifically the nature of the trap
- time-stamp: indicates last (re)initialisation of network entity that issued the trap
- variablebindings: additional information relating to the trap
Trap PDU (cont)

• Generic trap types
  – coldStart (0)
  – warmStart(1)
  – linkDown(2)
  – linkUp(3)
  – authenticationFailure(4)
  – egpNeighborLoss(5)
SNMP and the transport layer

- SNMP does not assume connection oriented or connectionless transport service.
- However, most implementations use UDP (connectionless transport service).
- SNMP uses two port numbers.
  - Agents listen for incoming GetRequest, GetNextRequest, and SetRequest commands on port 161.
  - Management stations listen for incoming Traps on port 162.
- SNMP has no provision to guarantee delivery => burden to cope with lost PDU is with application that is using SNMP.
- Applications can use:
  - Time out mechanisms
  - Re-issuing of request (operations)
How frequently to Poll?

- SNMP management station cannot rely purely on information derived from traps
- => must Poll relatively frequently
  .. But how frequently and what are the determining influences on this frequency?
Polling frequency Equation

Simplification

– Management station can only poll one agent at time
– management station polls a particular agent & does not do other work until it has finished with that agent
– a poll may involve one or more subsequent get/response transactions between the manager & agent

\[ N = \frac{\text{T}}{\Delta} \]

where

- \( N \) = number of agent
- \( \text{T} \) = desired polling interval (i.e. desired elapsed time between successive polls of the same agent)
- \( \Delta \) = average time required to perform a single poll
Average time required to perform a single poll

$\Delta$ depends on

- processing time to generate a request at a management station
- network delay from manager to agent
- processing time at the agent to interpret message
- processing time at the agent to generate response
- network delay from agent to manager
- processing time at manager to receive and interpret response
- number of request/response exchanges to obtain all desired information from an agent
Limitations of SNMPv1

- Not suitable for management of very large networks (principally because of its reliance on polling)
- Not suited for retrieval of large volumes of data
- SNMP traps are not acknowledged
- Authentication is trivial
- SNMP does not directly support imperative commands (actions)
- SNMP MIB is limited and does not readily support applications
- SNMP does not support manager to manager communications
- Many of above deficiencies are addressed in SNMPv3