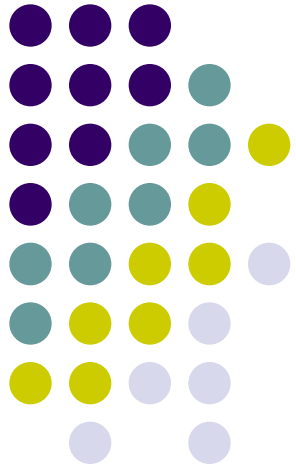
Simple Network Management Protocol



Lecture 6

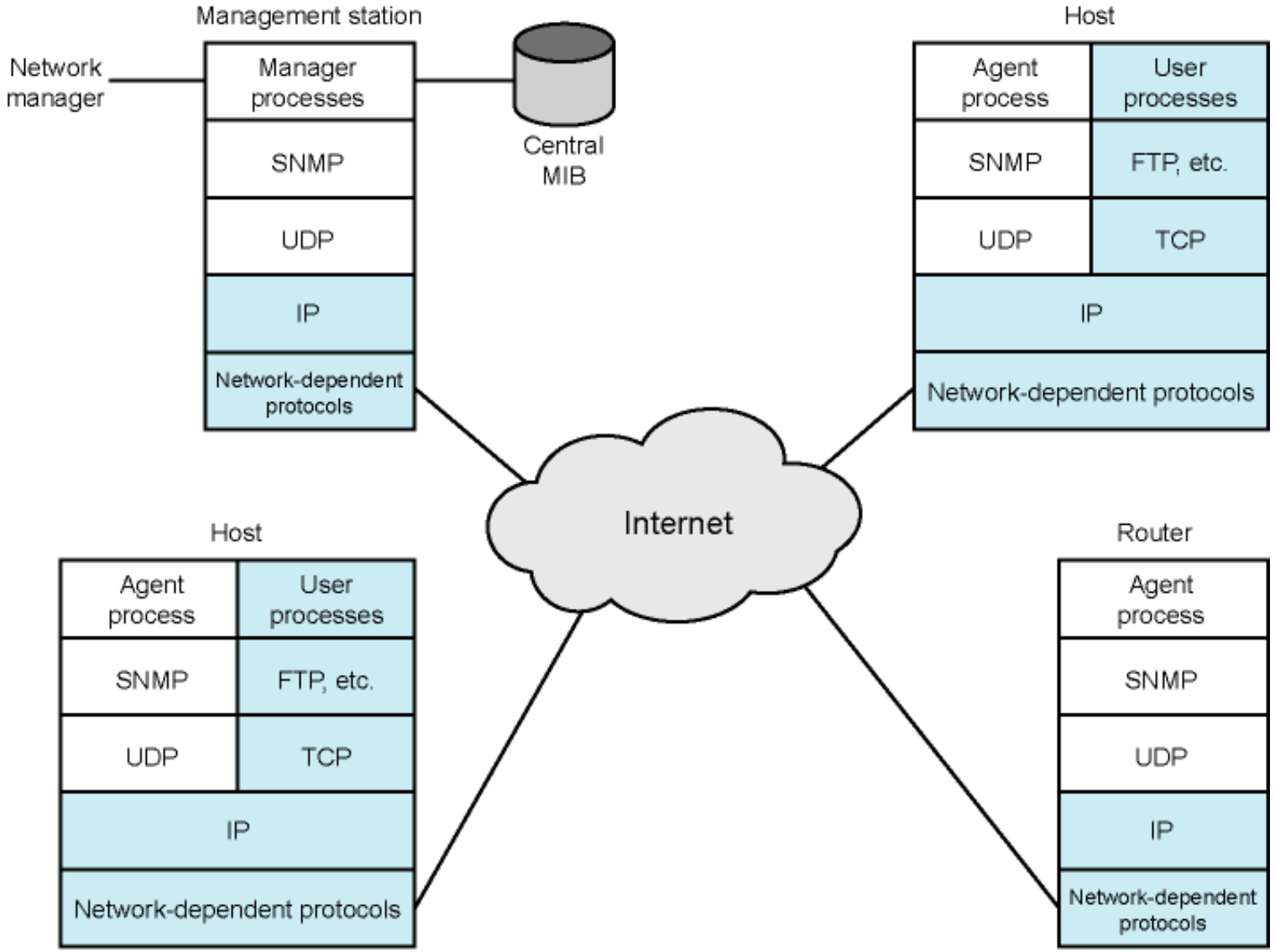
SNMPv1 & v2

SNMPv3 (RFC3410-3418)

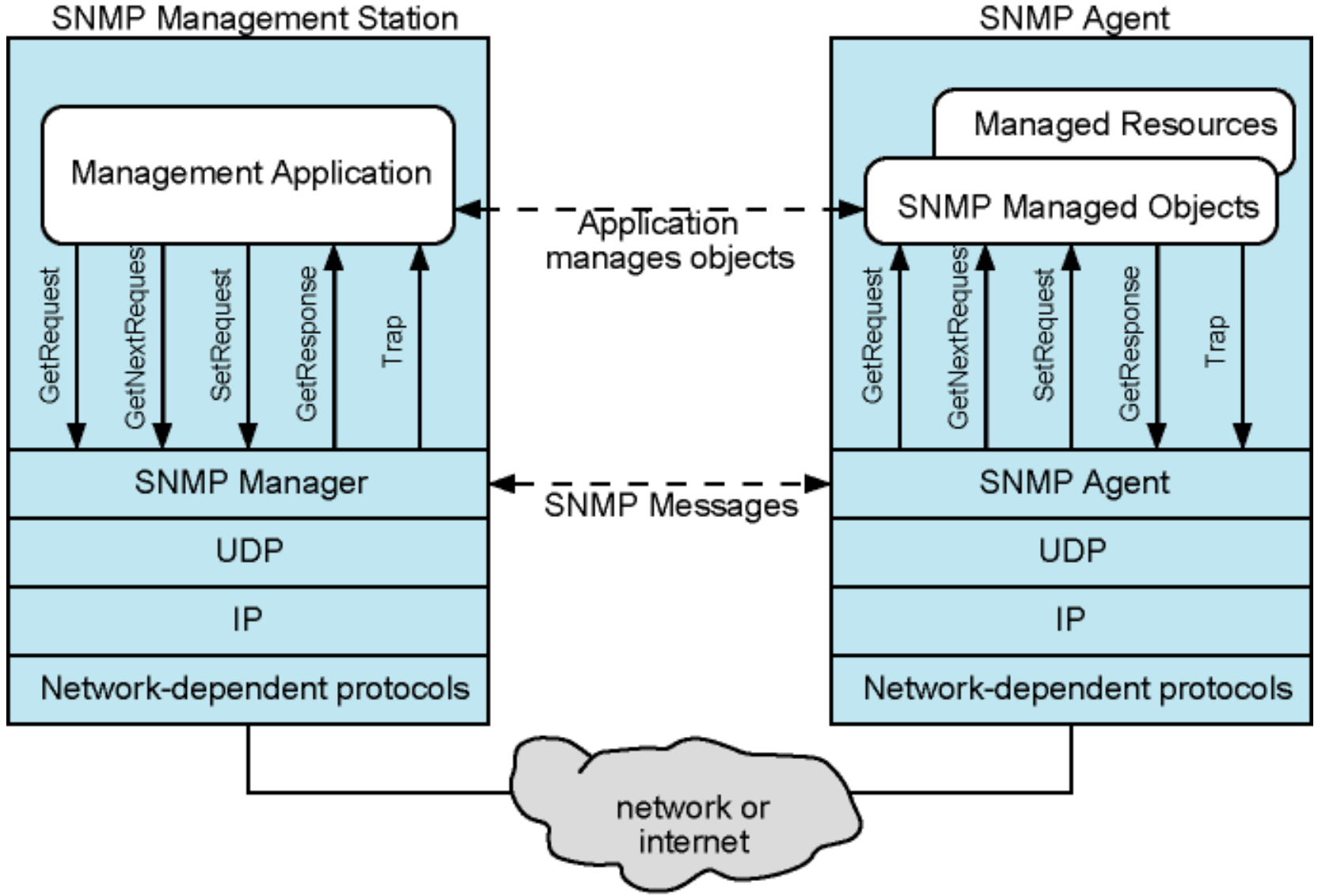
Note: Both SNMPv1 (RFC 1157) and SNMPv2 (RFC1901-1908) are duplicate

(supplementary: Chapter 4 of tutorial slides)

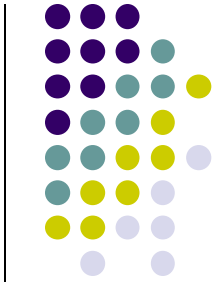
SNMPv1 Configuration



The Role of SNMPv1

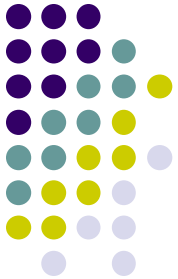


SNMPv1 Protocol



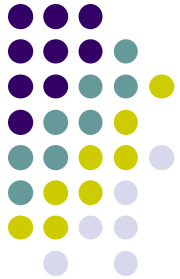
- Supported operations
  - get, set, trap
- Simplicity vs. limitations
  - Not possible to change the structure of MIB by adding or deleting object instances
  - Access is provided only to leaf objects
    - Not possible to access entire table or row in single action

SNMPv1 Operations Support



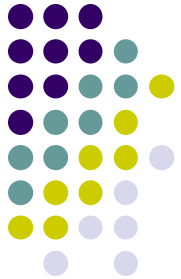
- Three general-purpose operations may be performed on scalar objects：
  - **Get**：A management station retrieves a scalar object value from a managed station.
  - **Set**：A management station updates a scalar object value in a managed station.
  - **Trap**：A managed station sends an unsolicited scalar object value to management station

SNMP Protocol – security concern



- In management environment
  - The management station and managed agent
    - One-to-many relationship
    - One station may manage all or a subset of target
  - The managed system and management station
    - One-to-may relationship
    - Each managed agent controls its local MIB and must be able to control the use of that MIB
    - Three aspects
      - Authentication service
      - Access policy
      - Proxy service

One-to-many Relationship



- A managed station controls its own local MIB and must be able to control the MIB access by a number of management stations
  - **Authentication service**: The managed station (agent) may wish to limit access to the MIB to authorized management stations.
  - **Access policy**: The managed station (agent) may wish to give different access privileges to different management stations.
  - **Proxy service**: A managed station may act as a proxy to other managed stations.

MIB Access and SNMP Access Mode

TABLE 7.1 Relationship Between MIB ACCESS Category and SNMP Access Mode

MIB ACCESS Category	SNMP Access Mode	
	READ-ONLY	READ-WRITE
read-only	Available for get and trap operations	
read-write	Available for get and trap operations	Available for get, set, and trap operations
write-only	Available for get and trap operations, but the value is implementation-specific	Available for get, set, and trap operations, but the value is implementation-specific for get and trap operations
not accessible	Unavailable	

SNMP Access Policy

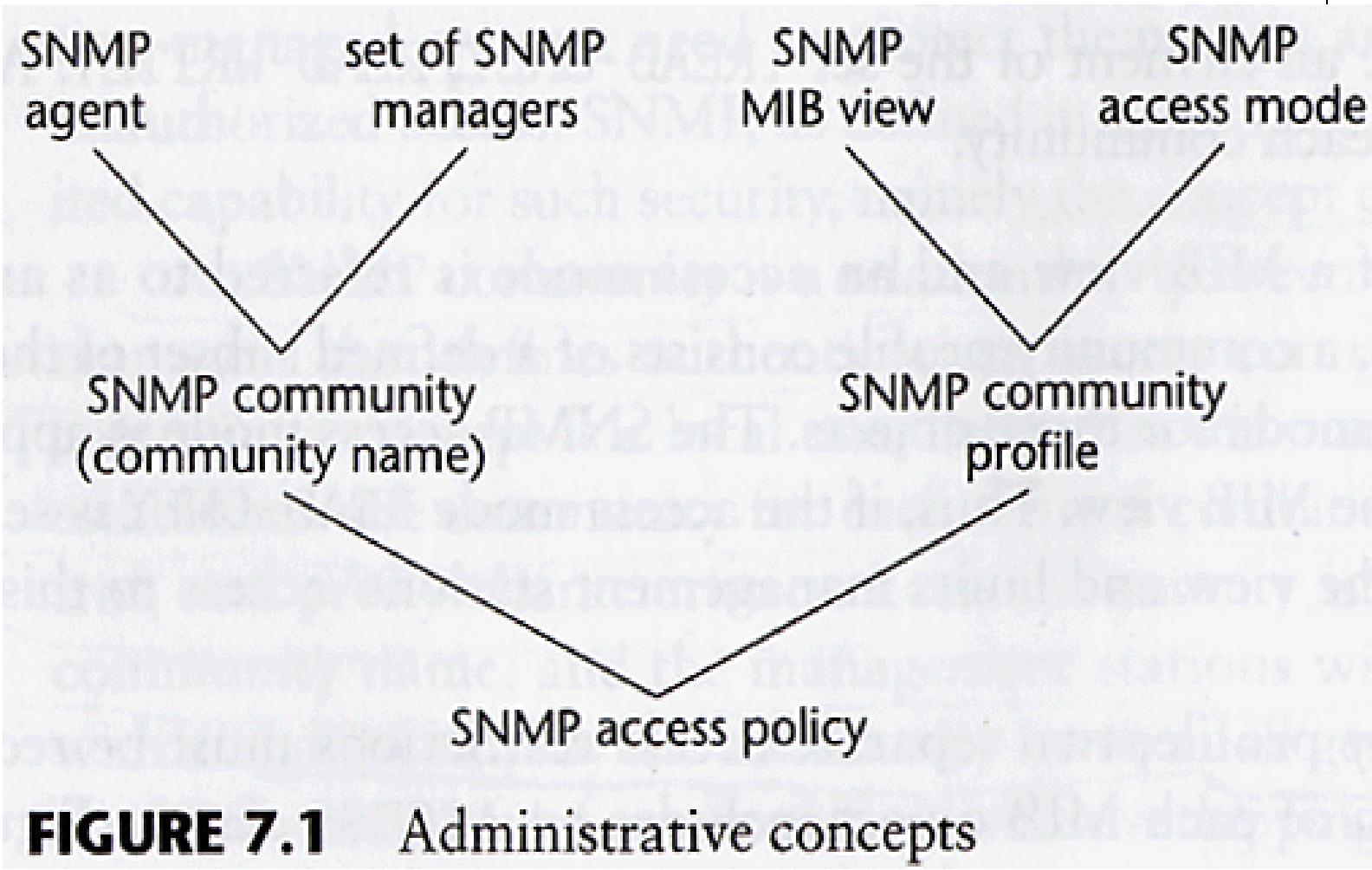
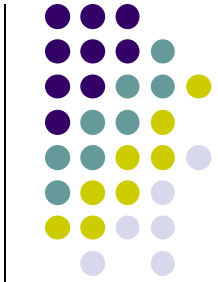


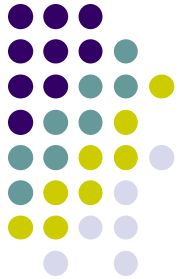
FIGURE 7.1 Administrative concepts

Concept of Community



- An SNMP community is
  - A relationship between an SNMP agent and a set of SNMP managers that defines
    - Authentication, access control and proxy
  - The managed system establishes one community for each combination of authentication, access control and proxy
  - Each community has a unique “community name”
  - Management station use certain community name in all get and set operations

Authentication and Access Using Community



- Authentication
  - The community name (password)
- Access policy
  - Community profile
    - SNMP MIB view
      - A subset of MIB objects
    - SNMP access mode
      - READ-ONLY, READ-WRITE

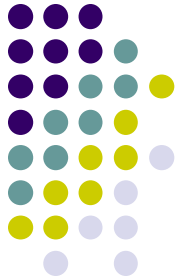
SNMPv1

- SNMP Message
  - Version Identifier
  - Community Name
  - Protocol Data Unit

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

- The length of SNMP messages should not exceed 484 octets.

SNMP Authentication



- Community
  - Relationship between an Agent and Managers.
- Community Name
  - Used to validate the SNMP messages.
  - SNMP Password.
  - Default ‘Get’ community name: “public”.
- Authentication Failure
  - Agent sends “*Authentication Failure Trap*” to Manager.

SNMPv1 PDU

Five SNMP PDUs:

GetRquest : [0] PDU
GetNextRequest : [1] PDU
GetResponse : [2] PDU
SetRequest : [3] PDU
Trap : [4] Trap-PDU

```
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
                    SEQUENCE OF {
                        SEQUENCE {
                        name  ObjectName,
                        value  ObjectSyntax
                        }
                    }
    }
```

PDU: Protocol Data Unit

SNMPv1 PDU (*cont.*)

GetRequest, GetNextRequest, SetRequest

PDU type	request-id	0	0	variable-bindings

GetResponse

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

variable-bindings

name	value	name	value	. . .	name	value

Note: PDU type is not defined in the SNMP or SNMPv2-PDU specifications, but it can be embedded using the BER encoding whenever an operation in encountered.

Trap-PDU

Enterprise:
- Type of Object generating trap.

Agent Address:
- Address of object generating trap.

Generic Trap:
- Generic trap type.

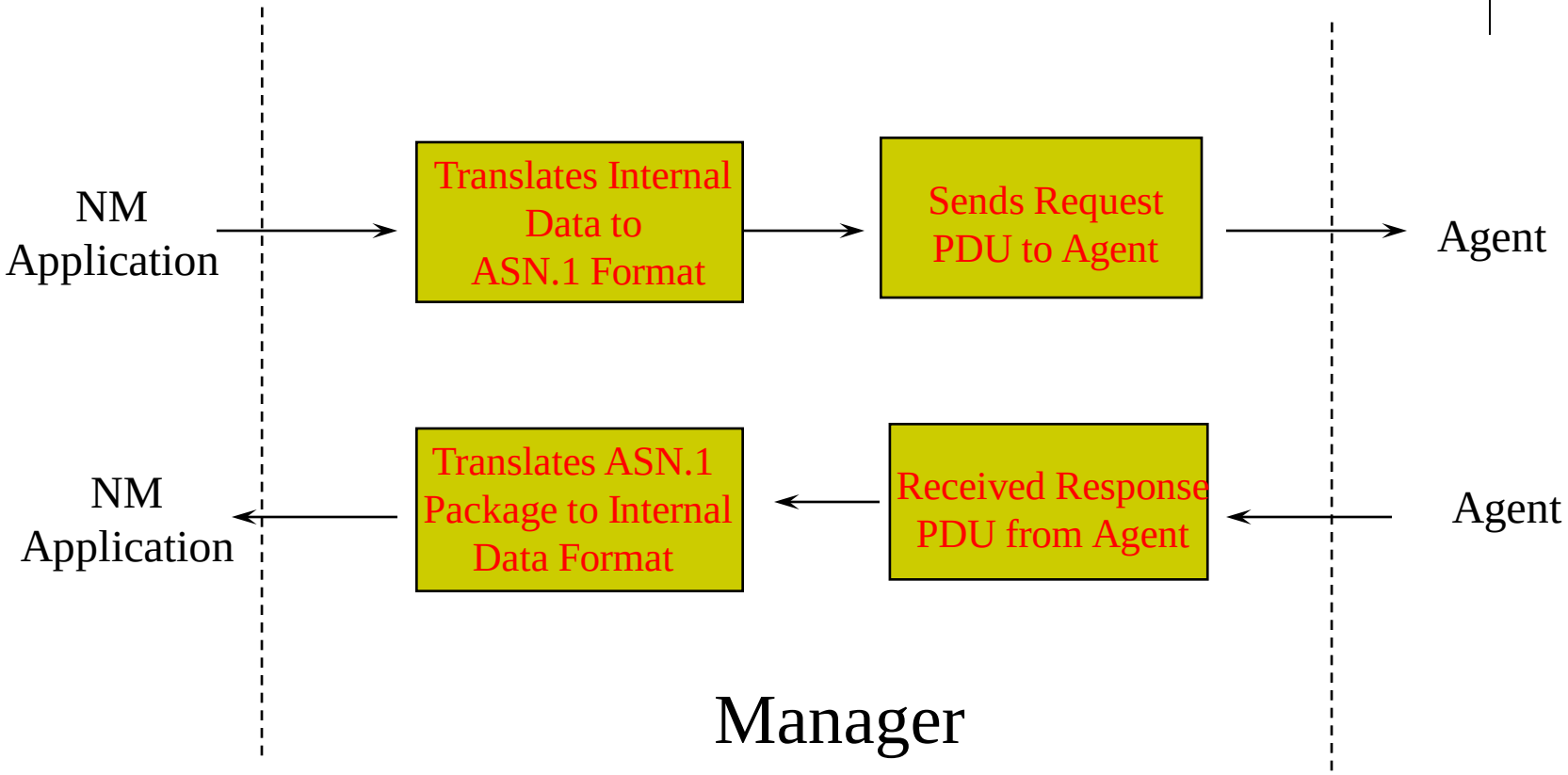
Specific Trap:
- Enterprise specific trap.

Time Stamp:
- Time elapsed between the last initialization of the network entity and the generation of the trap.

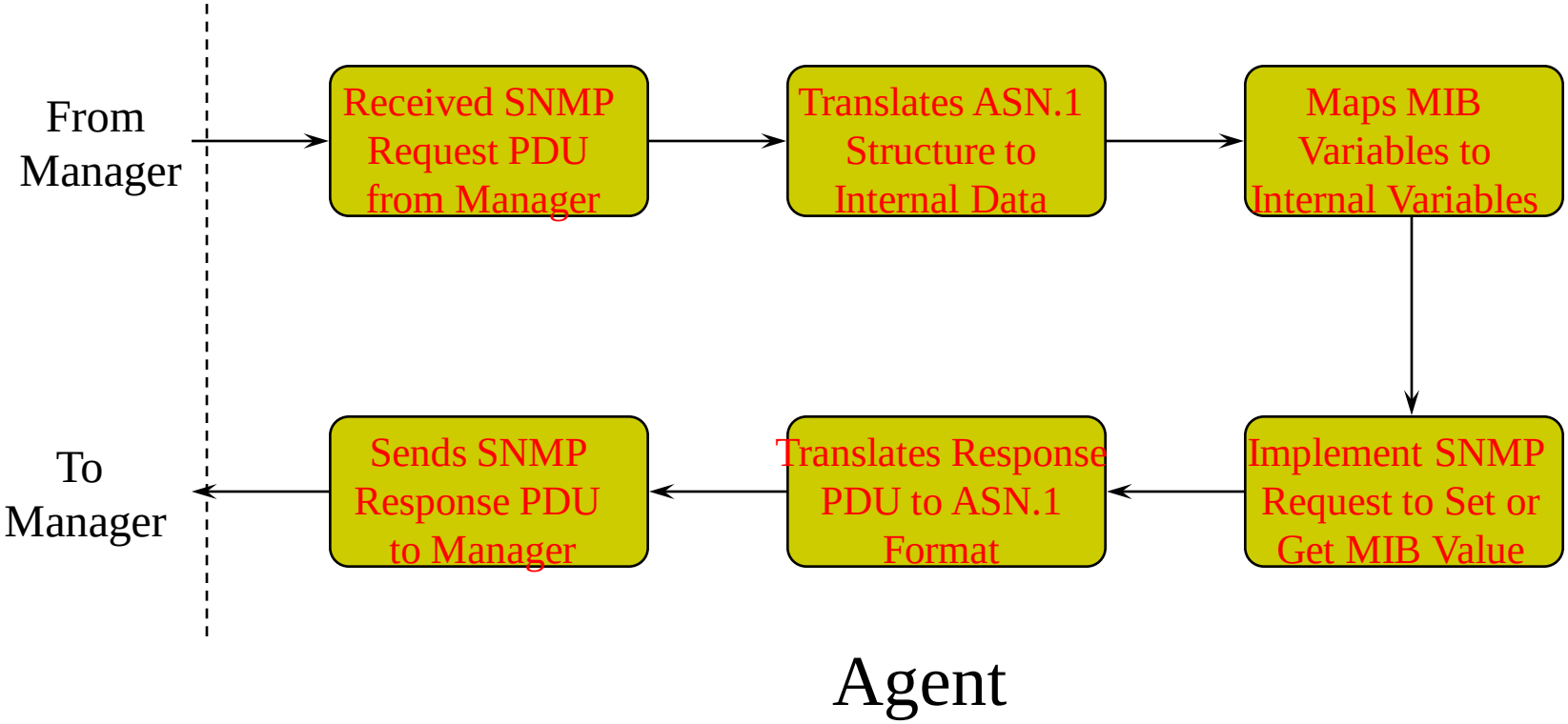
Variable Bindings
- "Interesting" information

```
Trap-PDU ::= [4]
    IMPLICIT 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
          }
```

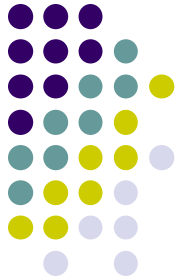
How does a Manager do?



How does an Agent do?

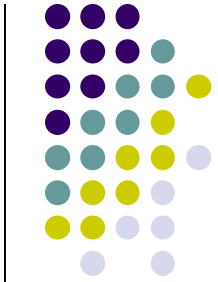


Main Loop of Agent (1/2)



- Agent waits for an incoming datagram in Port 161
- Reads the datagram from UDP and notes the transport address of the sending entity.
- Increments the QUANTUM to keep track of the logical request-id being processed by agent
- De-serializes the datagram into an ASN.1 structure. If error occurs, log error and discard packet.
- The ASN.1 structure is translated into SNMP message. If error occurs, log error and discard packet.
- Check on VERSION-NUMBER field. If error occurs, log error and discard packet.

Main Loop of Agent (2/2)

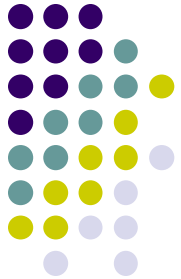


- Community name is looked up.
  - If community is unknown to agent, agent send AUTHENTICATION trap to Manager station in Port 162; log error and discard packet.
- Agent loops through list of variables in the request.
  - If no prototype is found, return a GET-RESPONE with error noSuchName and discard package.
  - Once prototype is fund, operation is checked against community profile. If mismatch occurs, return get-respone with error noSuchName or readOnly and discard package.
  - Otherwise, agent invokes access routine to perform the desired operation.

Transmission of an SNMP Message

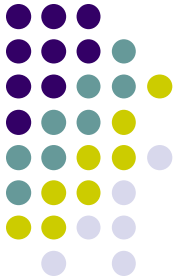
- Actions taken by a Transmitter:
  - The PDU is constructed, using the ASN.1 structure defined in RFC 1157.
  - This PDU is then passed to an authentication service. The authentication service then performs any required transformation for this exchange.
  - The protocol entity then constructs a message, consisting of a version field, the community name, and the result from step 2.
  - This new ASN.1 object is then encoded using the basic encoding rules and passed to the transport service.

Receipt of an SNMP Message



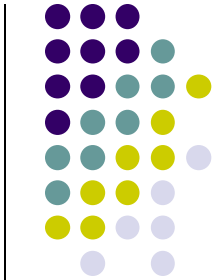
- Actions taken by a Receiver:
  - It does a basic syntax-check of the message and discards the message if it fails to parse.
  - It verifies the version number and discards the message if there is a mismatch.
  - The protocol entity then passes the user name , the PDU portion of the message and the source and destination transport addresses to an authentication service.
    - If authentication fails, the authentication service signals the SNMP protocol entity, witch generates a trap and discards the message.
    - If authentication succeeds, the authentication service returns a PDU in the form of an ASN.1 object .
  - The protocol entity does a basic syntax-check of the PDU and using the named community, the appropriate SNMP access policy is selected and the PDU is processed.

What's New in SNMPv2 ?



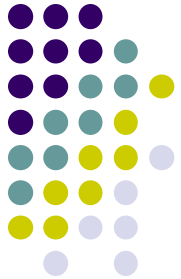
- No more Trap PDU, but 3 New PDUs:
  - GetBulkRequest, InformRequest, SNMPv2-Trap
- Added Security
- 18 Error Status Values (versus 6 in SNMPv1)
- SNMPv2 SMI / SNMPv2 MIB
- M-to-M Communications
- Enhanced Table Operations
- ...

SNMPv2 macro for object definition

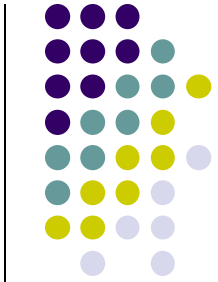


```
OBJECT-TYPE MACRO ::=
BEGIN
   TYPE NOTATION ::=
              "SYNTAX" Syntax
              UnitsPart
              "MAX-ACCESS" Access
              "STATUS" Status
              "DESCRIPTION" Text
              ReferPart
              IndexPart
              DefValPart
   VALUE NOTATION ::=
              value(VALUE ObjectName)

   Syntax ::=   -- Must be one of the following:
                -- a base type (or its refinement),
                -- a textual convention (or its refinement),
        or
                -- a BITS pseudo-type
                type
            | "BITS" "{" NamedBits "}"

   NamedBits ::= NamedBit
            | NamedBits "," NamedBit

   NamedBit ::=  identifier "(" number ")" -- number is
        nonnegative

   UnitsPart ::=
              "UNITS" Text
            | empty

   Access ::=
              "not-accessible"
            | "accessible-for-notify"
            | "read-only"
            | "read-write"
            | "read-create"

   Status ::=
              "current"
            | "deprecated"
            | "obsolete"

   ReferPart ::=
              "REFERENCE" Text
            | empty

   IndexPart ::=
              "INDEX"    "{" IndexTypes "}"
            | "AUGMENTS" "{" Entry      "}"
            | empty
   IndexTypes ::=
              IndexType
            | IndexTypes "," IndexType
   IndexType ::=
              "IMPLIED" Index
            | Index
   Index ::=
                -- use the SYNTAX value of the
                -- correspondent OBJECT-TYPE invocation
              value(ObjectName)
   Entry ::=
                -- use the INDEX value of the
                -- correspondent OBJECT-TYPE invocation
              value(ObjectName)

   DefValPart ::= "DEFVAL" "{" Defvalue "}"
            | empty

   Defvalue ::=  -- must be valid for the type specified in
              -- SYNTAX clause of same OBJECT-TYPE macro
              value(ObjectSyntax)
            | "{" BitsValue "}"

   BitsValue ::= BitNames
            | empty

   BitNames ::=  BitName
            | BitNames "," BitName

   BitName ::= identifier

   -- a character string as defined in section 3.1.1
   Text ::= value(IA5String)
END
```

Definitions associated with SNMPv2 macro for object definition

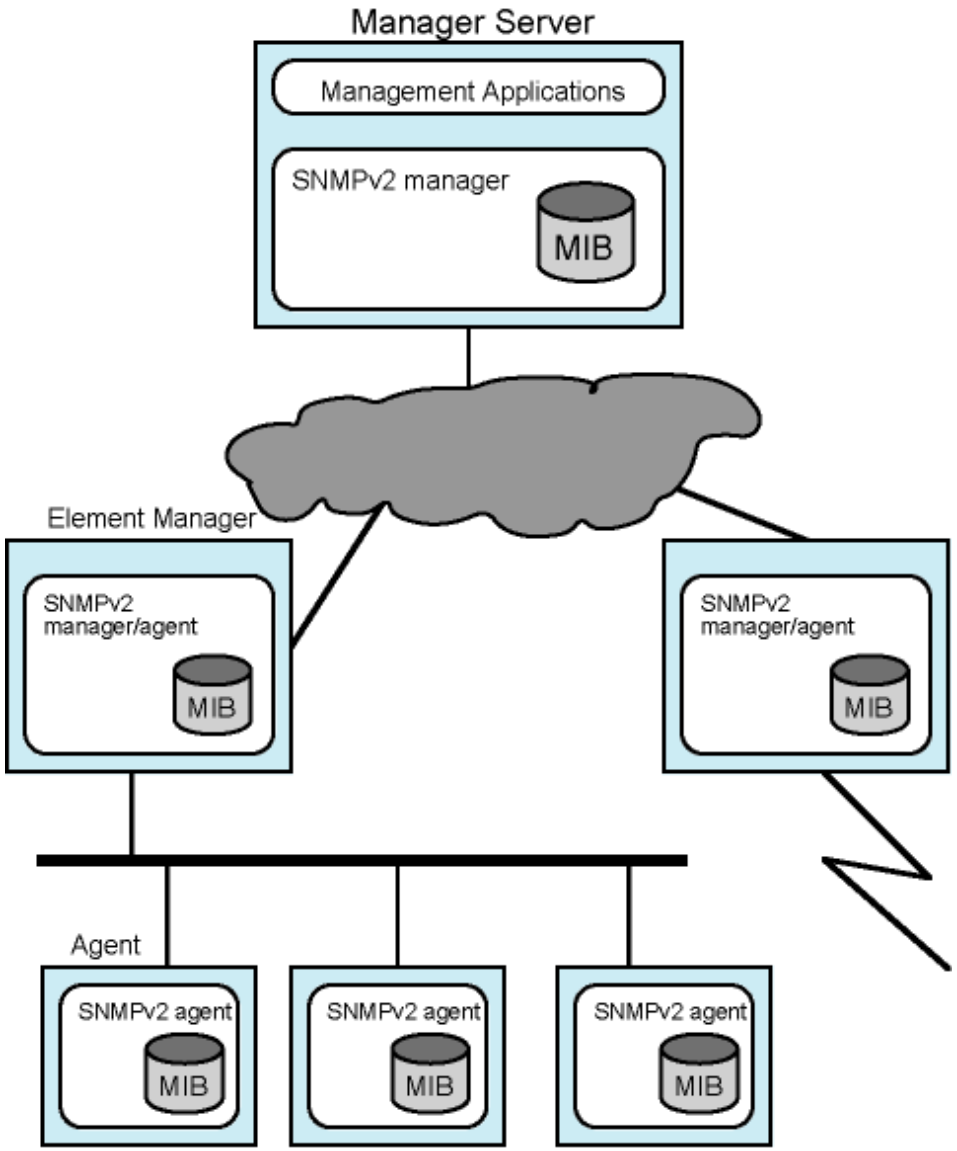


```
        ObjectName ::=
 OBJECT IDENTIFIER
NotificationName ::=
   OBJECT IDENTIFIER
-- syntax of objects
-- the "base types" defined here are:
--   3 built-in ASN.1 types: INTEGER, OCTET STRING, OBJECT IDENTIFIER
--   8 application-defined types: Integer32, IpAddress, Counter32,
--            Gauge32, Unsigned32, TimeTicks, Opaque, and Counter64
ObjectSyntax ::=
   CHOICE {
      simple
         SimpleSyntax,
        -- note that SEQUENCEs for conceptual tables and
        -- rows are not mentioned here...
      application-wide
         ApplicationSyntax
   }

-- built-in ASN.1 types
SimpleSyntax ::=
   CHOICE {
      -- INTEGERs with a more restrictive range
      -- may also be used
      integer-value              -- includes Integer32
         INTEGER (-2147483648..2147483647),
      -- OCTET STRINGs with a more restrictive size
      -- may also be used
      string-value
         OCTET STRING (SIZE (0..65535)),
      objectID-value
         OBJECT IDENTIFIER
   }

-- indistinguishable from INTEGER, but never needs more than
-- 32-bits for a two's complement representation
Integer32 ::=
      INTEGER (-2147483648..2147483647)

-- application-wide types

ApplicationSyntax ::=
   CHOICE {
      ipAddress-value
         IpAddress,

      counter-value
         Counter32,

      timeticks-value
         TimeTicks,

      arbitrary-value
         Opaque,

      big-counter-value
         Counter64,

      unsigned-integer-value  -- includes Gauge32
         Unsigned32
   }

-- in network-byte order
-- (this is a tagged type for historical reasons)
IpAddress ::=
   [APPLICATION 0]
      IMPLICIT OCTET STRING (SIZE (4))

-- this wraps
Counter32 ::=
   [APPLICATION 1]
      IMPLICIT INTEGER (0..4294967295)

-- this doesn't wrap
Gauge32 ::=
   [APPLICATION 2]
      IMPLICIT INTEGER (0..4294967295)

-- an unsigned 32-bit quantity
-- indistinguishable from Gauge32
Unsigned32 ::=
   [APPLICATION 2]
      IMPLICIT INTEGER (0..4294967295)
```

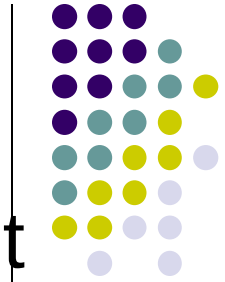


```
-- hundredths of seconds since an epoch
TimeTicks ::=
    [APPLICATION 3]
        IMPLICIT INTEGER (0..4294967295)

-- for backward-compatibility only
Opaque ::=
    [APPLICATION 4]
        IMPLICIT OCTET STRING

-- for counters that wrap in less than one hour
with only 32 bits
Counter64 ::=
    [APPLICATION 6]
        IMPLICIT INTEGER
(0..18446744073709551615)
```

SNMPv2 Managed Configuration

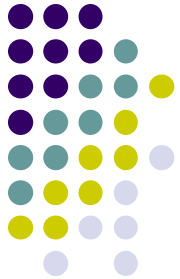


SNMPv2 --- Introduction (1/2)



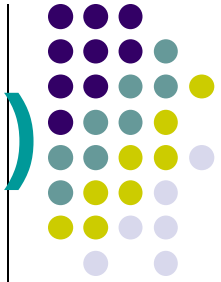
- Framework on which network management applications can be built
  - e.g fault management, performance monitoring, accounting
- Protocol used to exchange management information
- Each player maintains local MIB
  - Structure defined in standard
- At least one system responsible for management
  - Houses management applications

SNMPv2 --- Introduction (2/2)



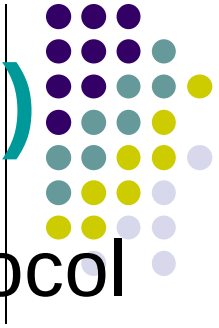
- Support central or distributed management
- In distributes system, some elements operate as manager and agent
- Exchanges use SNMP v2 protocol
  - Simple request/response protocol
  - Typically uses UDP
    - Ongoing reliable connection not required
    - Reduces management overhead

The Development of SNMPv2 (1/3)



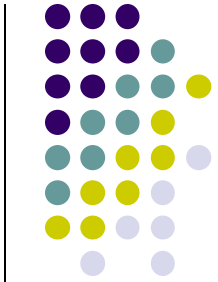
- An inadequate facility SNMP is not available.
- One major flaw that has inhibited the use of SNMP is that it provides no security facilities. Many vendors have chosen not to implement the *Set* command.
- Secure SNMP was issued in July 1992.
- SMP(Simple Management Protocol) was adopted by many enterprises, which was proposed by:
  - Jeffrey Case, SNMP Research, Inc.
  - Keith McCloghrie, Cisco Systems
  - Marshall Rose, Dover Beach Consulting, Inc.
  - Steven Waldbusser, International Network Services

The Development of SNMPv2 (2/3)



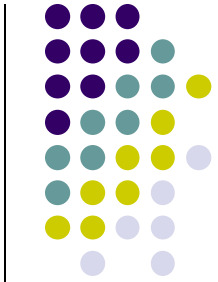
- The extensions defined in the SMP protocol fall into four categories:
  - *Scope*: SMP is designed to facilitate management of arbitrary resources, not just “network” resources.
  - *Size, speed, and efficiency:* The major change is the development of a bulk transfer capability for management information.
  - *Security and privacy:* SMP incorporates the enhancement found in secure SNMP.
  - *Deployment and compatibility* : SMP is designed to run or top of TCP/IP, OSI.

The Development of SNMPv2 (3/3)



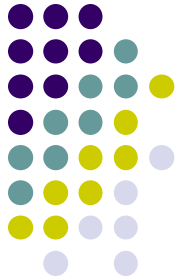
- SMP was accepted as a baseline for developing SNMPv2.
- Two working group were formed in October 1992, included function working group and security working group.
- SNMPv2 was completed in March 1993.
- SNMPv2 was revisited in 1996, and security aspects of SNMPv2.
- SNMPv2 is termed as community-based SNMPv2, or SNMPv2C

SNMPv2 Enhancements



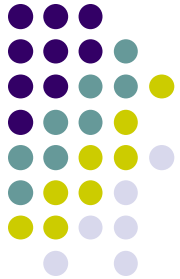
- SNMPv2 can support either a highly centralized network management strategy or a distributed one.
- In distributed case, superior and intermediate managers are constructed.
- Three key enhancements to SNMP are:
  - **SMI**
  - **Manager-to-manager capability**
  - **Protocol operations**
- The new function of creating and deleting conceptual rows in a table.
- SNMPv2 MIB contains basic traffic information about the operation of the SNMPv2 protocol.
- Two new PDUs: GetBulkRequest and InformRequest, one refinement: SNMPv2-Trap

SNMPv2 SMI



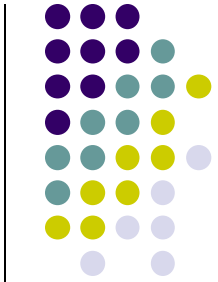
- SNMPv2 SMI is a superset of SNMP SMI
- The SNMPv2 SMI introduces four key concepts:
  - Object definitions
  - Conceptual tables
  - Notification
  - Information modules

Row Creation and Deletion



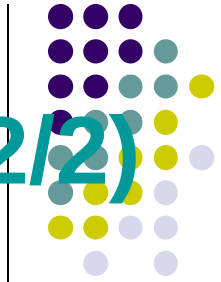
- Of all the issues addressed in evolving from SNMPv1 to SNMPv2, the two general strategies were considered :
  - Defined two new protocol data units, Create and Delete.
  - Embed the semantics for row creation and deletion into the MIB.

Textual Convention of Row-Status (1/2)



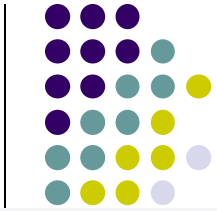
- The status including:
  - active - available
  - notInService – exists but not available for use.
  - notReady – exists but missing information necessary.
  - createAndGo – allow a management station to create a new instance, set to active.
  - createAndWait – allowing the creation but set to notInService.
  - destroy –allow deleting an existing conceptual row.

Textual Convention of Row-Status (2/2)



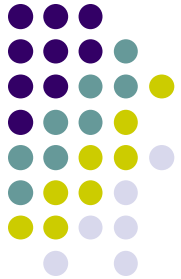
- Except ”not Ready”, all other 5 status can be specified in a management protocol set operation .
- Only three values will be returned in request to a management protocol retrieval operation: notReady, notInService, active.

RowStatus State Transitions



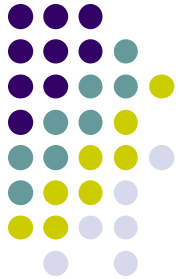
ACTION	A status column does not exist	B status column is notReady	C status column is notInService	D status column is active
set status column to createAndGo	noError → D or inconsistentValue	inconsistentValue	inconsistentValue	inconsistentValue
set status column to createAndWait	no Error see (1) or wrongValue	inconsistentValue	inconsistentValue	inconsistentValue
set status column to active	inconsistentValue	inconsistentValue or see (2) → D	noError → D	noError → D
set status column to notInService	inconsistentValue	inconsistentValue or see (3) → C	noError → C	noError → C or wrongValue
set status column to destroy	noError → A	noError → A	noError → A	noError → A
set any other column to some value	see (4)	noError see (1)	noError → C	see (5) → D

createAndWait



1. Manager instructs the agent to create a new row with a given index value.
2. If succeeds, the agent creates the row and assigns values to those objects in the row with defaultvalues.
3. If all read-create objects have default values, the row is in notInService status.
4. If some read-create objects don't have default values, the row is in notReady status.
5. The manager then issues a Get to determine the status of each read-create object. The agent responds with a value (default value), noSuchInstance (for supported but no default value), noSuchObject (for not suppoted).
6. The manager must use a Set to assign a value to all noSuchInstance objects, and can assign new value to default-value objects.
7. Once all supported objects have been created, the manager issues a Set to set the status to active.

createAndWait Example (1/2)



- Refer to page 386-389.
- pingEntry :: = SEQUENCE {

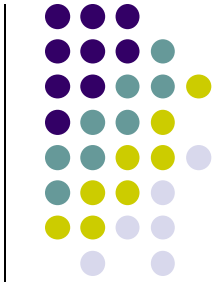
```
pingIndex   Integer32,  pingIPAddress     IpAddress,
pingDelay   Integer32,  pingsRemaining  Integer32,
pingstotal   Integer32,  pingsReceived    Integer32,
pingRTT     Integer32,  pingStatus          RowStatus,
pingSize    Integer32}
```

- Assume that the Index=1 already exists and we want to create a new row with Index=2 as in the following:

Index	IpAddress	Delay	Remaining	Total	Received	Rtt	Status
1	128.2.13.21	1000	0	10	9	3	active
2	138.2.13.99	1000	20	20	0		active

- Also, assume that pingSIze is not supported in the agent, pingReceived and pingRtt are read-only.

createAndWait Example (2/2)



- Manager issues

  SetRequest (pingStatus.2 = createAndWait)

- Agent responds with

  Response (pingStatus.2 = notReady)

- Manager issues a Get request

  GetRequest (pingIPAddress.2, pingDelay.2, pingsRemaining.2, pingsTotal.2, pingStatus.2, pingSize.2)

- Agent returns with

  Response (pingIPAddress.2 = noSuchInstance, pingDelay.2 = 1000, pingsRemaining.2 = 5, pingsTotal.2 = 5, pingStatus.2 = notReady, pingSize.2 = noSuchObject)

- Manager then sets the values

  SetRequest (pingIPAddress.2 = 128.2.13.99, pingsRemaining.2 = 20, pingsTotal.2 = 20, pingStatus.2 = active)

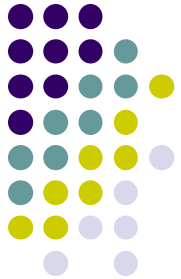
createAndGo

- createAndGo is simpler but restricted in two ways.

1. It must be limited to tables whose objects can fit into a single Set or Response PDU.
2. The manager doesn't automatically learn of default values.

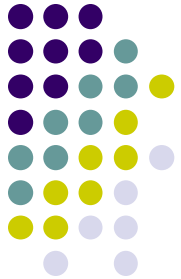
- The manger begins by selecting an object identifier. It may then issue a Get PDU to determine which read-create objects are noSuchInstance, or it may already know.
- The manager then issues a Set PDU to create a new row and assign values to that row.
- If the set operation succeeds, the row is created and put in the active state.

Protocol Operations



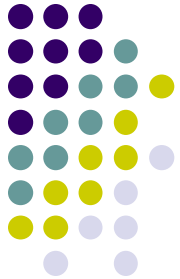
- Three types of access to management information:
  - Manager2agent request-response
  - Manager2manager request-response
  - Agent2manager unconfirmed

Transmission of an SNMPv2 Message



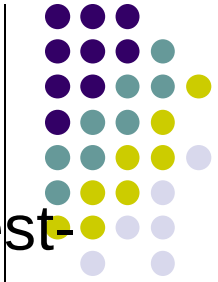
- Actions to transmit a PDU to another SNMPv2 entity:
  - The PDU is constructed, using the ASN.1 structure defined.
  - The PDU is passed to an authentication service.
  - The protocol entity constructs a message.
  - The new ASN.1 object is encoded, using the basic encoding rules (BER), and passed to the transport service.

Receipt of an SNMPv2 Message



- Actions upon reception of an SNMPv2 message:
  - syntax-check for message
  - Verifies the version number
  - Pass user name, PDU, source and destination transport addresses to an authentication.
  - Syntax-checking of PDUs and using the name community, appropriate SNMPv2 access policy is selected and PDU is processes accordingly.

SNMPv2 PDU formats



- GetRequest-PDU, GetNextRequest-PDU, SetRequest-PDU, SNMPv2-Trap-PDU, Information-PDU

PDU type	Request-id	0	0	Variable-bindings

- Response-PDU

PDU type	Request-id	Error-status	Error-index	Variable-bindings

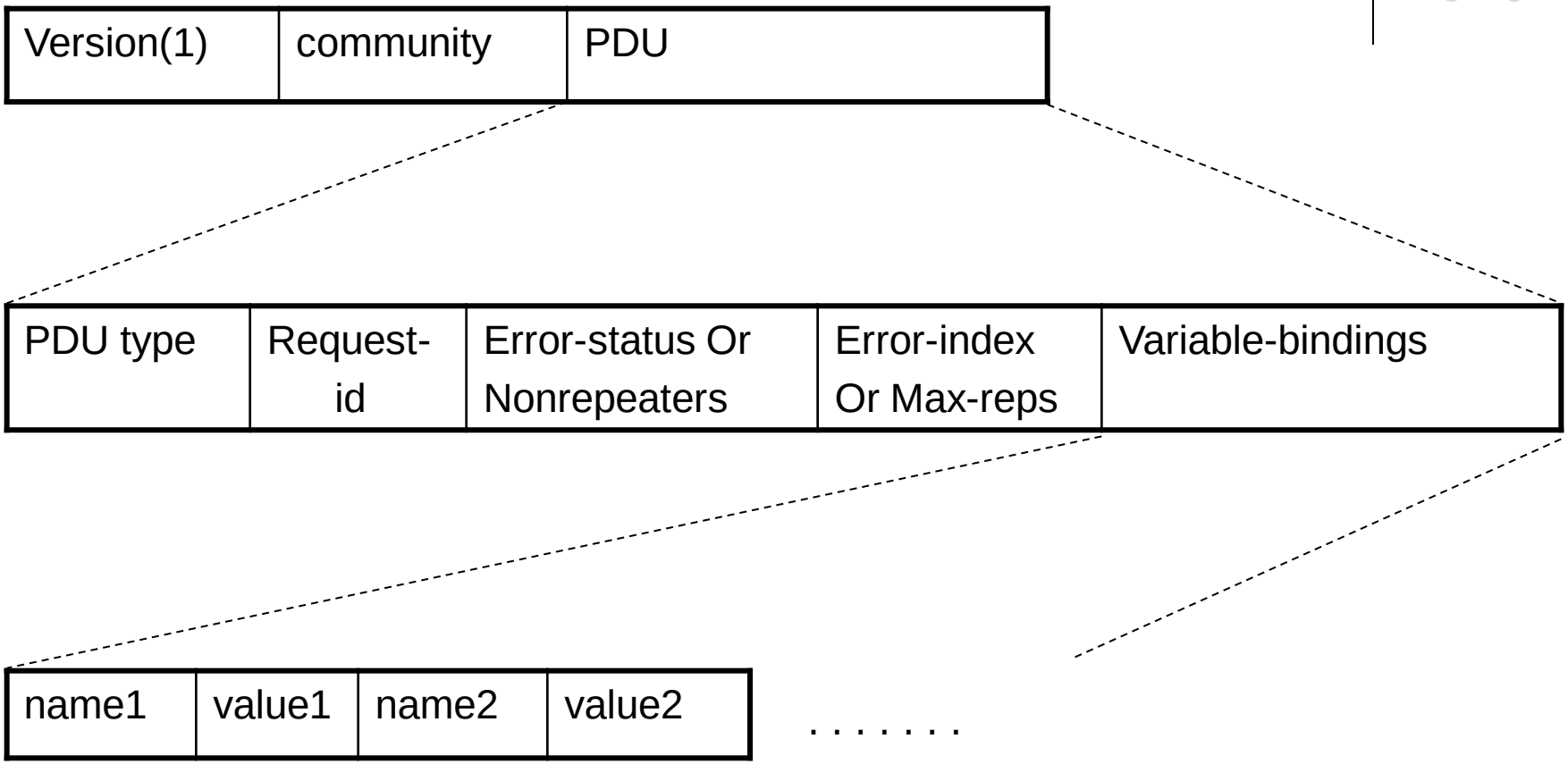
- GetBulkRequest-PDU

PDU type	Request-id	Non-repeaters	Max-repetitions	Variable-bindings

- Variable-bindings

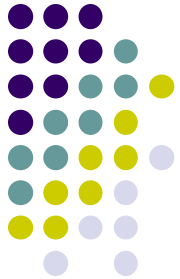
name1	value1	name2	value2	…..	namen	valuen

SNMPv2 message structure



Note: PDU type is not defined in the SNMP or SNMPv2-PDU specifications, but it can be embedded using the BER encoding whenever an operation in encountered.

SNMPv2 Operations

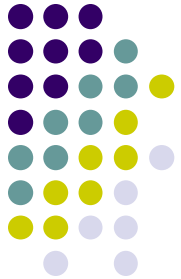


- SimpleWeb materials for snmpv2 (and snmpv3)
- Supplementary materials for (待補充資料)

1. getBulkRequest p378-382
2. Row creation and table augmentation

   P348-354 and P385-392

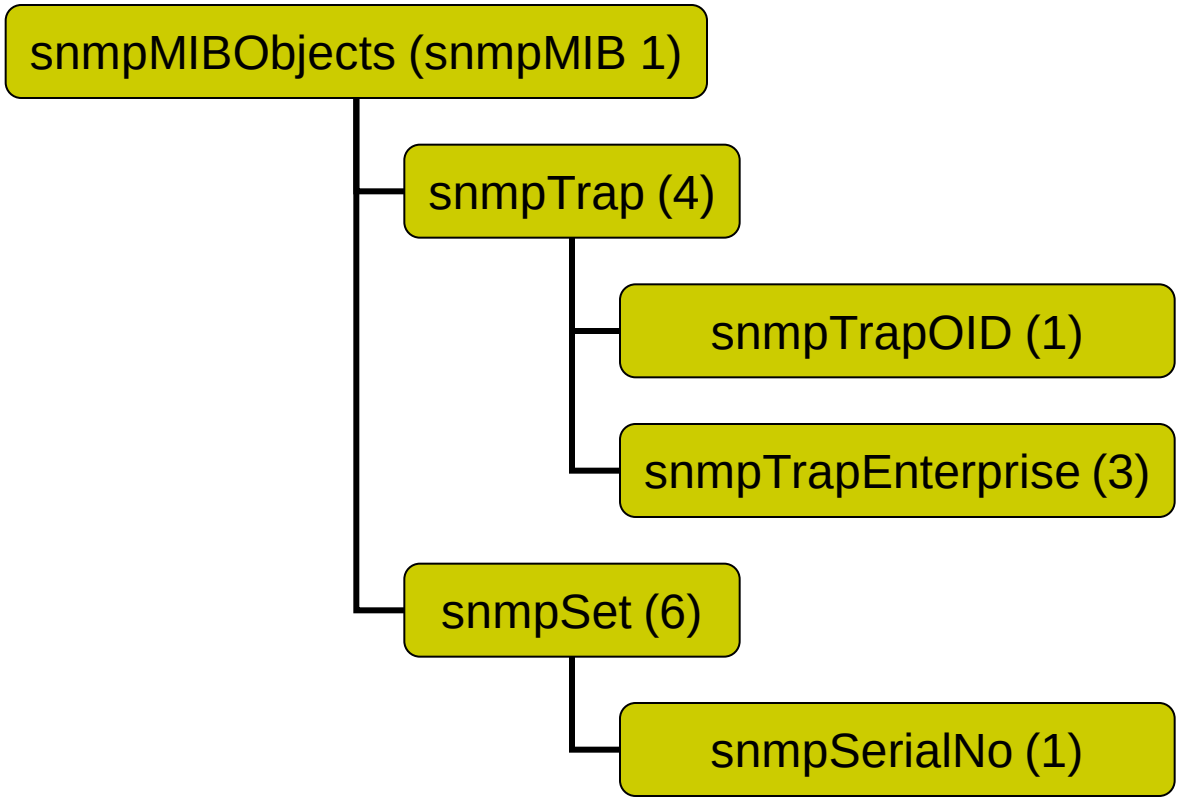
SNMPv2 MIB



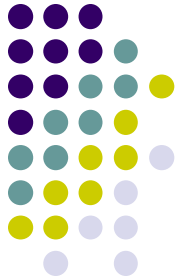
- **SNMPv2 MIB defines three groups:**
  - System group: expansion to MIB system group
  - SNMP group: refinement to MIB SNMP group
  - MIB objects group: deal with SNMPv2-trap and managers for set operations.
- **System group extenstions:**
  - sysORLastChanges
  - sysORTable
- **SNMP group refinement:**
  - Delete (7)~(29) objects of original snmp group
  - Add two counter objects: snmpSilentDrops and snmpProxyDrops for PDU drops

SNMP MIB Objects Group

Under 1.3.6.1.6(snmpV2).3(snmpModules).1(snmpMIB)

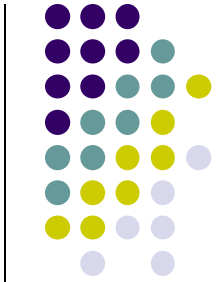


Object Group



- snmpTrap
  - snmpTrapOID
  - snmpTrapEnterprise
- snmpSet
  - snmpSerialNo
    - Multiple sets in sequence
    - Multiple managers concurrent issue set operation

Conformance Statements

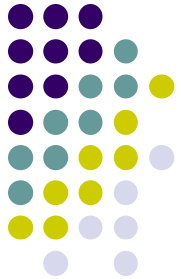


- OBJECT-GROUP
  - to specify a group of related managed objects
- NOTIFICATION-GROUP
  - to define a collection of notifications for conformance purpose
- MODULE-COMPLIANCE
  - to specify a min set of requirements wrt the implementation of MIB modules
- AGENT-CAPABILITIES
  - to specify the capabilities present in an SNMPv2 protocol entity acting in an agent role

Interfaces group improvements

- RFC 1573 introduces 4 tables to interfaces group
  - Extensions table (ifXTable)
    - addition info to ifTable
  - Stack table (ifStackTable)
    - shows the relationship among rows in ifTable
  - Test table (ifTestTable)
    - enable manager to instruct an agent to test an if
  - Receive address table (ifRcvAddressTable)

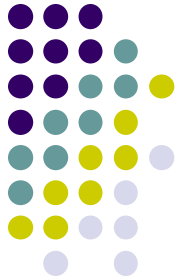
What's New in SNMPv3 ?



- SNMPv3 = SNMPv2 +Security + Administration
- Address security issues of SNMP:

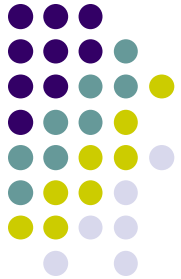
  RFC 2271-2275,2570,2575,2586,3411-3418
- Defines overall architecture and several textual conventions
- To be used together with SNMP v1 & v2
- Both managers and agents are called SNMP entities.

Goals of SNMPv3



- Rely heavily on SNMPv2u and SNMPv2*
- Address the need for secure Set request messages over the real world.
  - USM and CACM, Against principal threats
- Design a modular architecture for wide range of operational environments and accommodate alternative security models.
  - Self-contained documents
- Keep SNMP as simple as possible.
  - Controlled complexity

Principal Threats



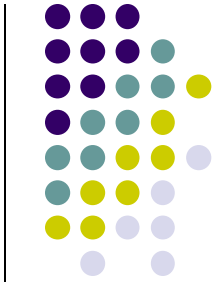
- Modification of information
  - Altering an in-transit message, encryption
- Masquerade
  - Access by unauthorized entity, authentication
- Message stream modification.
  - Reordered, delayed, or replayed, authentication
- Disclosure
  - Illegally observe on-going exchanges, encryption

- Not including DoS and traffic analysis

SNMP document roadmap (RFC 2271)

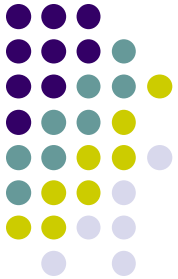
- Document roadmap, applicability statement, and coexistence and transition
- Message Handling
  - Transport mappings, message processing and dispatcher, security
- PDU Handling
  - Protocol operations, applications, access control
- Information Model
  - SMI, textual conventions, conformance stmts
- MIBs
  - Rfc 1157, 1212, 1442-1444, 1902-1904

SNMPv3 Entities



- An SNMP entity consists of applications and SNMP engine (identified by snmpEngineID).
- The SNMP engine is composed of:
  - Dispatcher – sends and receives messages
  - Message processing – prepares messages to be sent and extracts the data from the received messages
  - Security – provides authentication and privacy services
  - Access control – is responsible for controlling access to MIB objects
- User based authentication uses MD5 or SHA algorithms to authenticate users without sending a password in the clear.
- The privacy service uses DES to encrypt messages
- Access control subsystem
  - Configure agents to provide a number of levels of access to MIB
  - Control what objects a user can access and what operations she is allowed to perform on those objects

SNMPv3 Applications

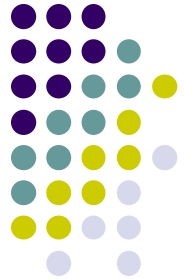


- Command generator
- Command responder
- Notification originator
- Notification receiver
- Proxy forwarder

- An agent contains: proxy forwarder, command responder, and notification originator
- A manager contains: command generator, notification originator, and notification receiver

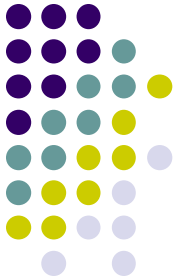
(Refer to p456, 458, and 460 for the detailed figures)

Traditional Manager and Agent



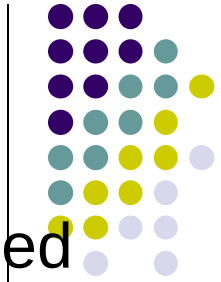
- For manager, the SNMP engine contains:
  - Dispatcher
  - Message processing subsystem (v1,v2c, v3)
  - Security subsystem
- For agent, the SNMP engine contains:
  - Dispatcher
  - Message processing subsystem (v1,v2c, v3)
  - Security subsystem
  - Access control subsystem
- Note that security-related functions are organized into two separate subsystems: security (privacy) and access control (authentication)

SNMPv3 Terminology



- Associated with each SNMP entity is snmpEngineID
- Each SNMP entity manages a number of contexts of management information, each has a contextName
- There is a single manager of contexts within an entity, each entity has a unique contextEngineID
- Access control is governed by the specific context and the identity of the user requesting access (called **principal,** an individual(s) or application(s))
- **snmpMessageProcessingModel** – v1, v2c, v3
- **snmpSecurityModel** – SNMPv1, v2c, USM
- **snmpSecurityLevel** – noAuthnoPriv, authNoPriv, authPriv

Abstract Service Interfaces



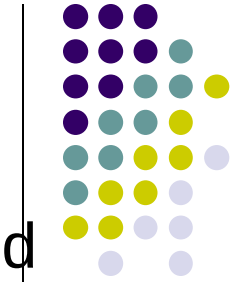
- The services between modules in an entity are defined in terms of primitives and parameters.
- A primitive specifies the function to be performed.
- The parameters are used to pass and control information.

- The SNMPv3 documents distinguish among 3 kinds of values: input values, output values, and primitive values.

  for example,

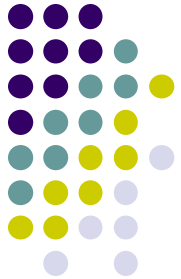
  funtioncode – requestservice (IN w IN x OUT y OUT z)

Dispatcher primitives



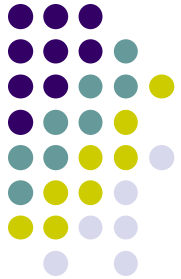
- Define the interface between SNMP applications and Dispatcher.
- Primitives are
  - sendPdu (command generator or notification originator)
  - processResponse Pdu
  - ProcessPdu (command responder)
  - returnResponsePdu
  - registerContextEngineID
  - unregisterContextEngineID
- Parameters are such as
  - TransportDomain
  - securityModel
  - PDU
  - pduType, etc

MIBs for SNMPv3 Applications



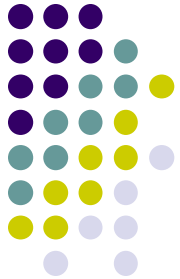
- Management target MIB
- Notification MIB
- Proxy MIB

SNMPv3 PDU



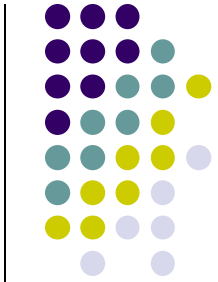
- Page 490 and 500

MIBs for SNMPv3 Applications



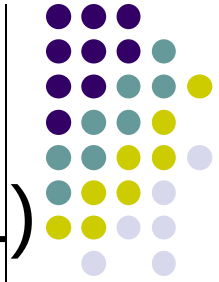
- the management target MIB
- the notification MIB
- the proxy MIB

SNMPv3 Textual Conventions



- snmpEngineID
- snmpSecurityModel
- snmpMessageProcessingModel
- snmpSecurityLevel
- snmpAdminString
- snmpTagValue
- snmpTagList
- keyChange

USM --- Introduction

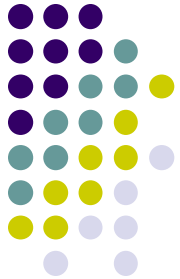


- SNMPv3 message format (refer to fig. 16.1)
- Command generator and responder message processing (refer to fig.16.3, 16.4)
- USM specification encompasses:
  - Authentication (HMAC)
  - Timeliness
  - Privacy (DES)
  - Message format (refer to fig 16.6)
  - Discovery
  - Key management

USM --- SNMPv3 Message Format

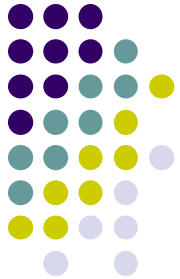
- SNMPv3 message format (refer to fig. 16.1)
  - **msgGlobalData (HeaderData)**
    - ✓ msgVersion
    - ✓ msgID
    - ✓ msgMaxSize
    - ✓ msgFlags
    - ✓ msgSecurityModel
  - **msgSecurityParameters**
  - **msgData (ScopedPdu, scope of encription)**
    - ✓ contextEngineID
    - ✓ contextName
    - ✓ PDU

msgFlags & msgSecurityModel



- **msgFlags**
  - reportableFlag – report PDU sent to sender if applied
  - privFlag – encryption was applied
  - authFlag – authentication was applied
- **msgSecurityModel**
  - 1 for SNMPv1
  - 2 for SNMPv2c
  - 3 for USM

msgSecurityParameters



- msgAuthoritativeEngineID
- msgAuthoritativeEngineBoots
- msgAuthoritativeEngineTime
- msgUserName
- msgAuthenticationParameters
- msgPrivacyParameters

Command Generator Message Processing

- refer to Fig.16.3
- **Preparing an outgoing request message**
  1. **sendPdu**(securityLevel,TAddress, CNamd, PDU) fromCommand generator to dispatcher (dispatcher assigns a unique sendPDUHandle)
  2. **prepareOutgoingMsg(**TAddress, CName, pduHandle, PDU) from dispatcher to MPS (Message Processor assigns a unique msgID to this message)
  3. MP prepares a **scopedPDU & msgGlobalData**
  4. **generateRequestMessage** from MP to Security Model
  5. MP caches **cache**(msgID, pduhandle, CName)
  6. **prepareOutgoingMessage** is back from MP to D
  7. **sendPDU** is back from D to CG with sendPDUHandle
  8. D also passes the message to UDP transport layer for transmission, using the TAddress

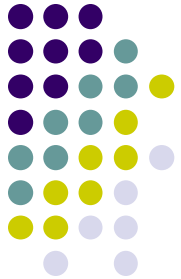
Command Generator Message Processing

- refer to fig.16.3
- **Receiving an Incoming Response Message**
  1. Transport layer passes the received message to Dispatcher
  2. **prepareDataElements(TAddr, msg)** is issued from D to MP
  3. MP extracts the related fields and passed the received message to Security Model, using processIncomingMdssage
  4. SM processes **msgSecurityParameters** and returns the **scopePdu**(CName, PDU) and statusInformation
  5. MP extracts the CName from scopePdu and msgID is used to check the match of securityModel, securityName, securityLevel, contextEngineID, and contextEngineName
  6. **prepareDataElements** (PDU, CName, sendPduHandle) is retrieved from the cache and sent from MP to Dispatcher
  7. **processResponsePdu(**sendPduHandle, CName, PDU) is dispatched to application from Dispatcher

User-Based Security Model (USM)

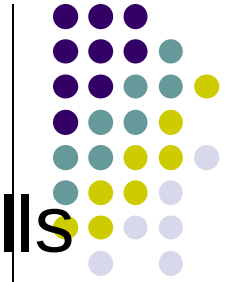
- USM specification encompasses:
  - Authentication (HMAC) – data integrity and authentication
  - Timeliness – protect against data delay and replay
  - Privacy (DES) – protect against disclosure of msg payload
  - Message format – define msgSecurityParameters
  - Discovery – for one SNMP engine finding out another engine
  - Key management – key generation, update, and use

USM -- Authoritative SNMP Engine



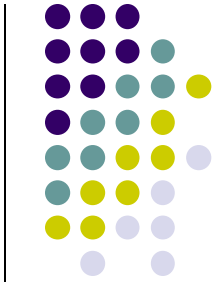
- Authoritative SNMP Engine
  - When an SNMP message contains a payload which expect a response (ex. Get, GetNext, GetBulk, Inform PDU), then the **receiver** is authoritative
  - When an SNMP message contains a payload which does not expect a response (ex. SNMPv2-Trap, Response, Report PDU), then the **sender** is authoritative
  - For message sent on behalf of a **Command Generator** and **Inform message from a Notification Originator**, the **receiver** is authoritative
  - For message sent on behalf of a **Command Responder** or for **trap message from a Notification Originator**, the sender is authoritative

USM – UsmSecurityParameters



- For outgoing, from MP to USM, the USM fills the parameters
- For Incoming, from MP to USM, the USM processes the values
  - msgAuthoritativeEngineID
  - msgAuthoritativeEngineBoots
  - msgAuthoritativeEngineTime
  - msgUserName
  - msgAuthenticationParameters (HMAC)
  - msgPrivacyParameters (DES CBC)

USM Timeliness Mechanism (1/5)



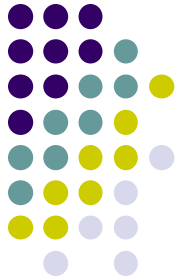
- **Management of authoritative clocks ,** each authoritative engine must maintain 2 objects**:**

  snmpEngineBoots and **snmpEngineTime**

  - When snmpEngineTime reaches 2^{31} - 1, snmpEngineBoots is incremented
  - If an authoritative engine is unable to determine its latest snmpEngineBoots value, it sets the value to the maximum value
  - This will cause the **notInWindow** authentication failure, which needs to manually configure the system

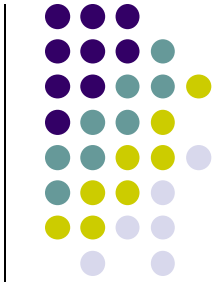
- **Synchronization**
- **Timeliness checking**

USM Timeliness Mechanism (2/5)



- **Management of authoritative clocks**
- **Synchronization**
  - Nonauthoritative engine keeps three objects for each authoritative engine: **snmpEngineBoots, snmpEngineTime,** and **latestReceivedEngineTime**
  - The latestReceivedEngineTime is updated when a larger **msgAuthoritativeEngineTime** is received.
  - The purpose of this message is to protect against a **replay** message attack.

USM Timeliness Mechanism (3/5)

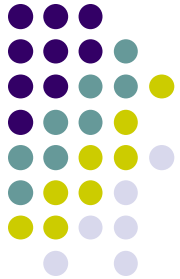


- **Management of authoritative clocks**
- **Synchronization**
  - If the message is authentic, then the received nonauthoritative engine updates those 3 variables according to the rules:

    (1) an update is called if at least one of the following conditions holds: msgAuthoritativeEngineBoots > snmpEngineBoots OR [(msgAuthoritativeEngineBoots = snmpEngineBoots) AND (msgAuthoritativeEngineTime > latestReceivedEngineTime)]

    (2) If an update is called, the following changes are made:
    - snmpEngineBoots = msgAuthoritativeEngineBoots
    - snmpEngineTime = msgAuthoritativeEngineTime
    - latestReceivedEngineTime = msgAuthoritativeEngineTime

USM Timeliness Mechanism (4/5)



- **Timeliness Checking by Authoritative Receiver**
  - Message must be received within a reasonable time window to avoid delay and replay attacks
  - If the message is authenticated and whose MAEI is the same as the value of SEI for this engine, then engine compares the values of **MAEB** and **MAET** from the incoming message with values of **SEB** and **SET** that the engine maintains for itself.
  - The incoming message is considered outside the window if

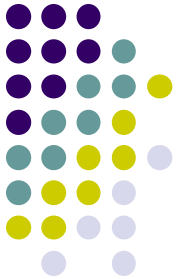
    (1) SEB has reached its maximum, OR

    (2) MAEB =! SEB, OR

    (3) MAET differs from SET by more than 150 seconds.

  - MAET represents the remote engine's actual timer value for nonauthoritative receiver, and represents the remote engine's notion of the receiver's actual timer value for authoritative receiver.

USM Timeliness Mechanism (5/5)



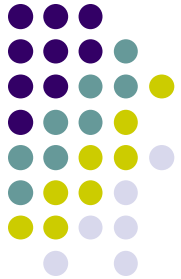
- **Timeliness Checking by Nonauthoritative Receiver**
  - If each incoming message is authenticated and whose MAEI is **not the same as (??)** the value of SEI for this engine, then engine compares the values of **MAEB** and **MAET** from the incoming message with values of **SEB** and **SET** that the engine maintains for itself.
  - The incoming message is considered outside the window if

    (1) SEB has reached its maximum, OR

    (2) MAEB < SEB, OR

    (3) MAEB = SEB and MAET =< SET – 150 seconds.

usmUser Group

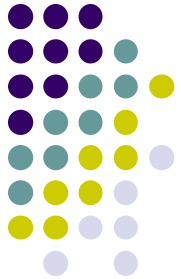


- **Each SNMP entity maintains a usmUser MIB**
  - A single scalar object, usmUserSpinLock

    which enables cooperating Command generator applications to coordinate their use of facilities to change keys using the usmUserTable
  - A Table usmUserTable

    which contains 13 columnar objects indexed by usmUserEngineID and usmUserName

Cryptographic functions

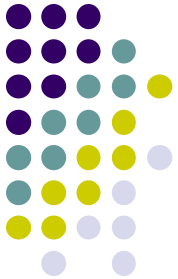


- **Athentication -- HMAC-MD5-96 or HMAC-SHA-96**
  - 16 octets (128 bits) authkey using MD5
  - 20 octets (160 bits) authkey using SHA-1
  - Both are truncated to 12 octets

- **Encryption**
  - **CBC mode of DES is used**
  - **16 octets of privkey with the first 8 octets for DES key**
  - **The last 8 octets are used as the pre-IV**

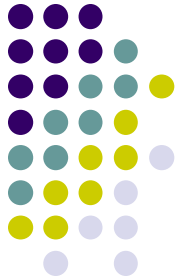
- **Two values: privKey and authKey (are not accessible via SNMP)**

USM Message Processing (1/4)



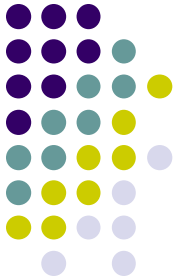
- **Command Generator Message Processing**
  - Message Processor provides the following input parameters: messageProcessingModel, globalData, maxMessageSize, securityModel, securityName, securityLevel, and scopedPDU.
- **Steps upon receipt of generateRequestMsg from MP**
  1. USM determines if there is an entry in usmUserTable for the detination securityEngineID and the source securityName
  2. USM determines if the required securityLevel is supportedif
  3. if the securityLevel requires privacy, then the scopedPdu is encrypted using CBC-DES and privkey. If privacy is not required, then msgPrivacyParameters is set to NULL string.

USM Message Processing (2/4)



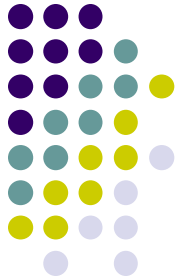
- **Steps upon receipt of generateRequestMsg from MP**
  4. snmpEngineID is stored in msgAuthoritativeEngineID
  5. securityName is stored in msgUserName
  6. If securityLevel requests authentication, the current value of snmpEngineBoots and snmpEngineTime corresponding to the snmpEngineID are stored in msgAuthoritativeEngineBoots, and ~Time respectively. The message authentication code is calculated over the entire message using HMAC and authkey and stored in msgAuthenticationParameters.
  7. The completed message with its length is returned to the calling Message Processing Module.

USM Message Processing (3/4)



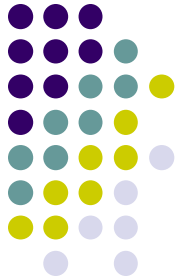
- **USM receives a response**
  - Message Processor provides the following input parameters: messageProcessingModel, maxMessageSize, securityParameters, securityModel, securityLevel, wholeMsg, wholeMsgLength.
- **Steps upon receipt of processIncomingMsg from MP**
  1. The security parameter values are extracted from securityParameters
  2. If the value of msgAuthoritativeEngineID in securityParameters is unknown and if this SNMP engine performs discovery, it may optionally creates a new entry in its usmUserGroup MIB.
  3. USM determines if there is an entry in usmUserTable for the remote authoritative securityEngineID and the local securityName.

USM Message Processing (4/4)



- **Steps upon receipt of processIncomingMsg from MP**
  4. USM determines if the requested securityLevel is supported
  5. If authentication is specified, then HMAC and authkey are applied and the results are compared to msgAuthenticationParameters.
  6. USM performs time synchronization.
  7. USM performs timeliness checking to see if the message is within the window.
  8. If the securityLevel indicates that encryption was performed, then the scopedPdu is decrypted using CBC-DES and privkey for this user.
  9. The scopedPdu is returned to the calling MP module.

View-Based Access Control Model (VACM)

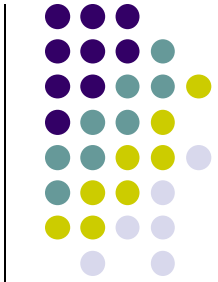


- Two important characteristics:

1. Determines whether access to a managed object in a local MIB by a remote principal should be allowed
2. Makes use of a MIB that defines the access control policy for this agent and makes it possible for remote configuration to be used

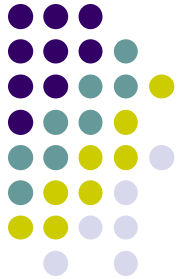
- Five VACM elements:
  - Groups
  - Security level
  - Contexts
  - MIB views
  - Access policy

Groups



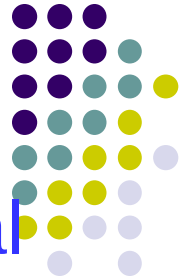
- A group is defined as a set of 0 or more <securityModel, securityName> tuples.
- A secuirtName refers to a principal, and access rights for all principals in a given group are identical.
- It is useful for categorizing managers w.r.t. access rights.
- Any combination of securityModel and securityName belong to at most one group

Contexts (1/2)



- The context is a concept that relates to access control.
- A MIB context is a named subset of the object instances in the local MIB.
- Contexts provide a useful way of aggregating objects into collections with different access policies.
- Contexts have the following characteristics:
  - An SNMP entity, identified by contextEngineID may contain more than one context.
  - An object or object instance may appear in more than one context
  - When more than one contexts exist, to identify an individual, its contextName and contextEngineID must be identified in addition to its object type and instance.

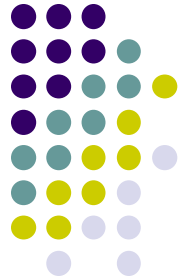
Contexts (2/2)



- When a mgmt station interacts with an agent, then the interaction is between a mgmt principal and the agent's SNMP engine, and the access control privileges are expressed in a MIB view that applies to this principal and this context.
- For example,
  - The managed object type ifDescr provides textual information about an interface.

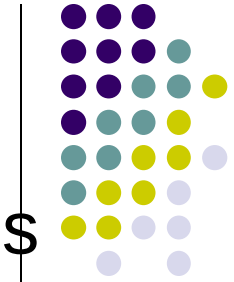
    To identify the description of device-x's 1st network interface, 4 pieces of information are needed:
    - contextEngineID,
    - contextName (device-x),
    - the managed object type (ifDescr),
    - and the instance ("1").

MIB Views



- To restrict the access of a particular group to a subset of the managed objects at an agent.
- Access to context is by means of a MIB view, which defines a specific set of managed objects.
- Based upon view subtrees and view families. Or say, the MIB view is defined in terms of a collection (families) of subtrees, with each subtree being included in or excluded from the view.
- A subtree is simply a node in the MIB's naming hierarchy plus all its subordinate elements.
- Associated with each entry in vacmAccessTable are 3 MIB views, one each for read, write, and notify access.

Access Policy



VACM enables an SNMP engine to be configured to enforce a particular set of access rights. Access determination depends on the following factors:

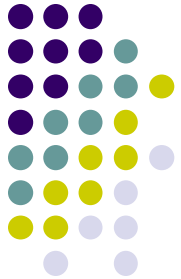
- He principal making the access request.
- The security level by which the request was communicated in an SNMP message.
- The security model used for processing the request message.
- The MIB context for the request.
- The specific object instance for which access is requested.
- The type of access requested (read, write, notify).

Motivation of VACM

SNMPv1 and SNMPv2C use community string to represent the following security-related info.:

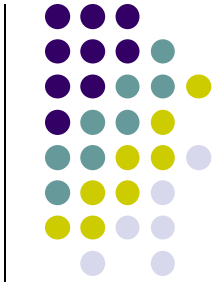
- The identity of requesting entity (management station)
- The identity of performing entity (agent acting for itself or for proxied entity)
- The identity of location of management information to be access (agent or proxied entity)
- Authentication information
- Access control information (authentication to perform required operation)
- MIB view information

Substantial improvement of VACM



- requesting entity – msgUserName and msgSecurityModel, mapped into groupName
- performing entity – contextEngineID
- location of mgmt information -- contextName
- Authentication information – msgAuthenticationParameters by USM, not directed used by VACM
- Access control information – entry in vacmAccessTable, which depends on groupName, contextName, securityModel, and securityLevel (derived from msgFlags in incoming request message)
- MIB view information – view objects in vacmViewTreeFamilyTable, which is determined by operation type.

VACM logic & VACM MIB



- Refer to fig 17.2 and p533 for detailed descriptions

- VACM MIB contains following information:
  - Local contexts
  - groups
  - Access rights
  - MIB views

  All are defined in vacmMIBObjects (p536)