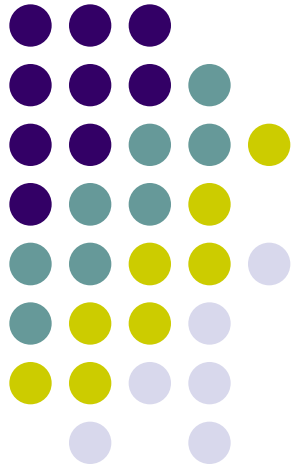
Remote Monitoring

Lecture 5

RMON1 (RFC 1757)

RMON2 (RFC 1997,2021)

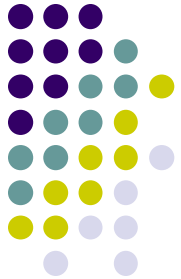
(supplementary: Chapter 5 of tutorial slides)



Scope of RMON1 & 2

APPLICATION		↑						
Presentation		|					
Session	RMON 2	|	RMON 2					
Transport		|					
Network		↓						
Data Link (MAC)	RMON 1	↑ |	Ethernet	Token Ring	FDDI	Frame Relay, HDLC, PPD, SDLL, X.25, CIRP		
Physical		| ↓				V-series	T1	E1 G703

RMON Goals

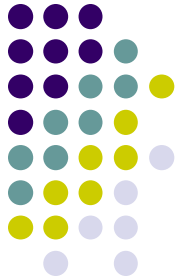


- Off-line operation
- Proactive monitoring
- Problem detection and reporting
- Value-added-data
- Multiple managers

In short,

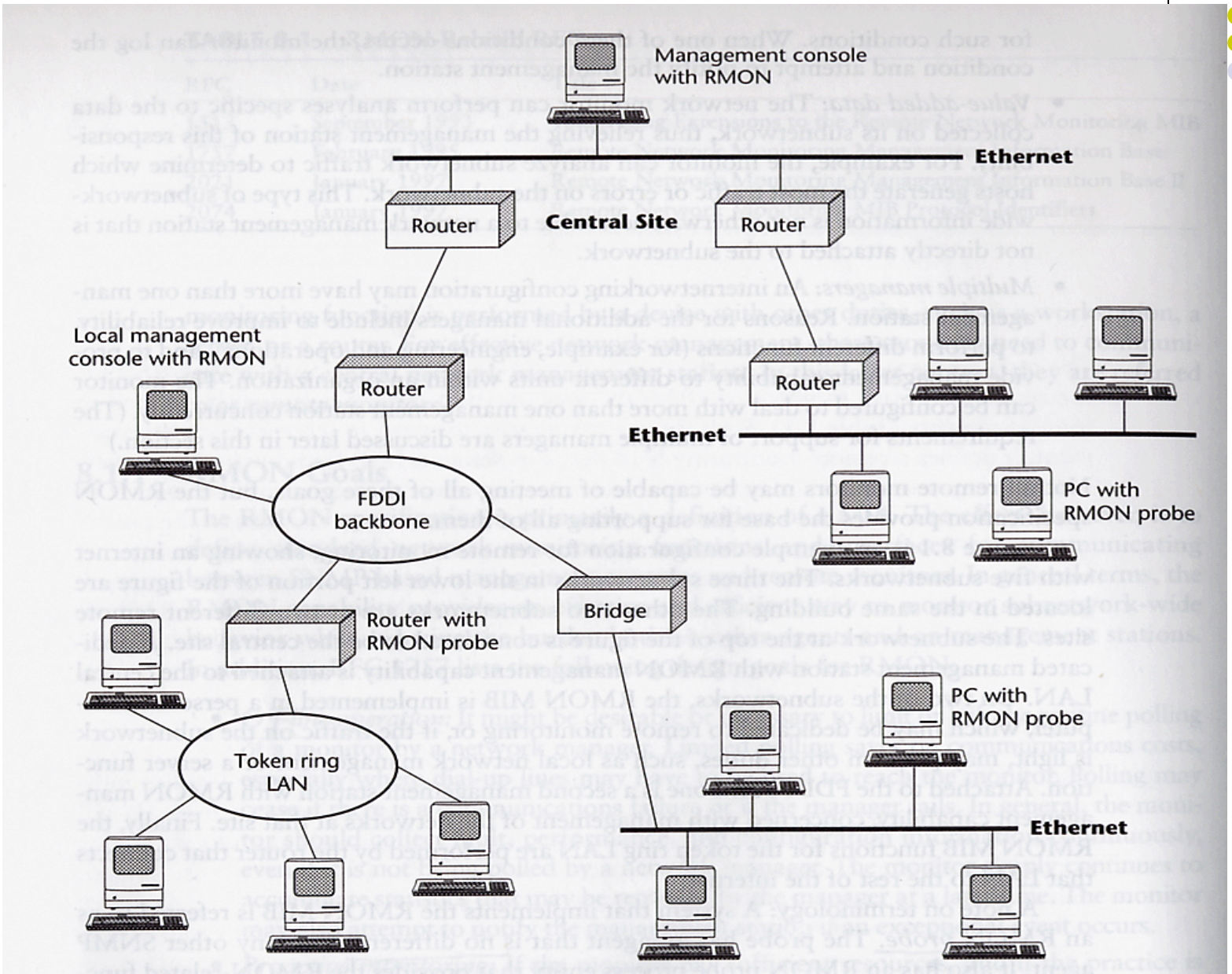
An SNMP manager can learn of the amount of traffic into and out of each device with MIB-II, but can't easily learn about the traffic on the LAN as a whole.

Control of Remote Monitors

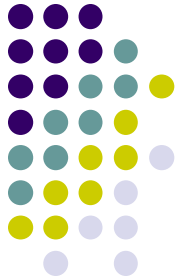


- RMON MIB contains features that support extensive control from the management station, which falls into two categories:
  - Configuration
    - Configuration dictates the type and form of data to be collected.
    - The MIB is organized into a number of function groups. Each group contains one or more control tables and one or more associated data tables.
  - Action invocation
    - SNMP *Set* operation is used to issue a command, a process called ***action invocation***.
    - An object can be used to represent a command, so that a specific action is taken if the object is set to a specific value. This is possible using RMON MIB objects.

Example Configuration using RMON



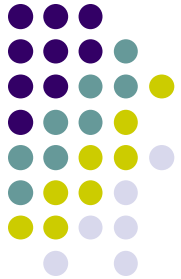
Multiple Managers



- The problem of concurrent access:
  - A manager may capture and hold monitor resources for a long period of time.
  - Resources could be assigned to manager that crashes without releasing the resource.
- Associated with each control table is a columnar object that identifies the owner of a particular row of the table and of the associated function.
- Ownership label could be:

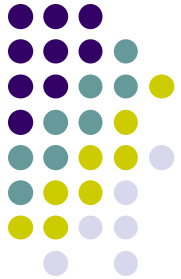
IP address, management station name, network manager's name, location, or phone number

Multiple Managers (Cont.)



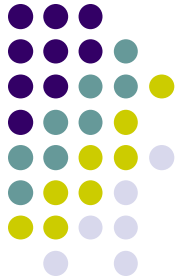
- The ownership label can be used in the following ways:
  - A manager may recognize resources it owns and no longer needs.
  - A network operator can identify the manager that owns a particular resource or function.
  - A network operator may have authority unilaterally to free resources which another network operation has reserved.
  - If a manager experiences a re-initialization, it can recognize resources it has reserved.

Table Management



- In the SNMPv1 framework, the procedure for adding and deleting table rows are, to say the least, unclear.
- SNMPv2 provides a clearer but more complex set of procedure for adding and deleting rows, compared to that for SNMPv1.
- The RMON specification includes a set of textual conventions and procedural rules.

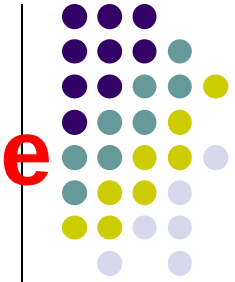
Textual Conventions



- Two data types are defined in RMON specification. In ASN.1, the definitions appear as follows:

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

Table Management --- Control Table



- rm1ControlIndex – served to identify a set of rows of rm1DataTable
- rm1ControlParamenter – applied to all data rows controlled by this control row
- rm1ControlOwner – the owner of this row
- rm1ControlStatus – the status of this row

Row Addition

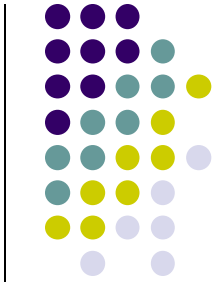
rmlControlTable

rmlControlIndex	rmlControlParameter	rmlControlOwner	rmlControlStatus
1	5	monitor	valid(1)
2	26	manager alpha	valid(1)
3	19	manager beta	valid(1)

rmlDataTable

rmlDataControlIndex	rmlDataIndex	rmlDataValue
1	1	46
2	1	96
2	2	85
2	3	77
2	4	27
2	5	92
3	1	86
3	2	26

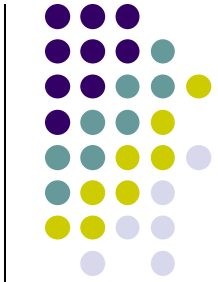
RMON Polka



- To resolve the problem of concurrent table addition attempts from multiple mgmt stations:

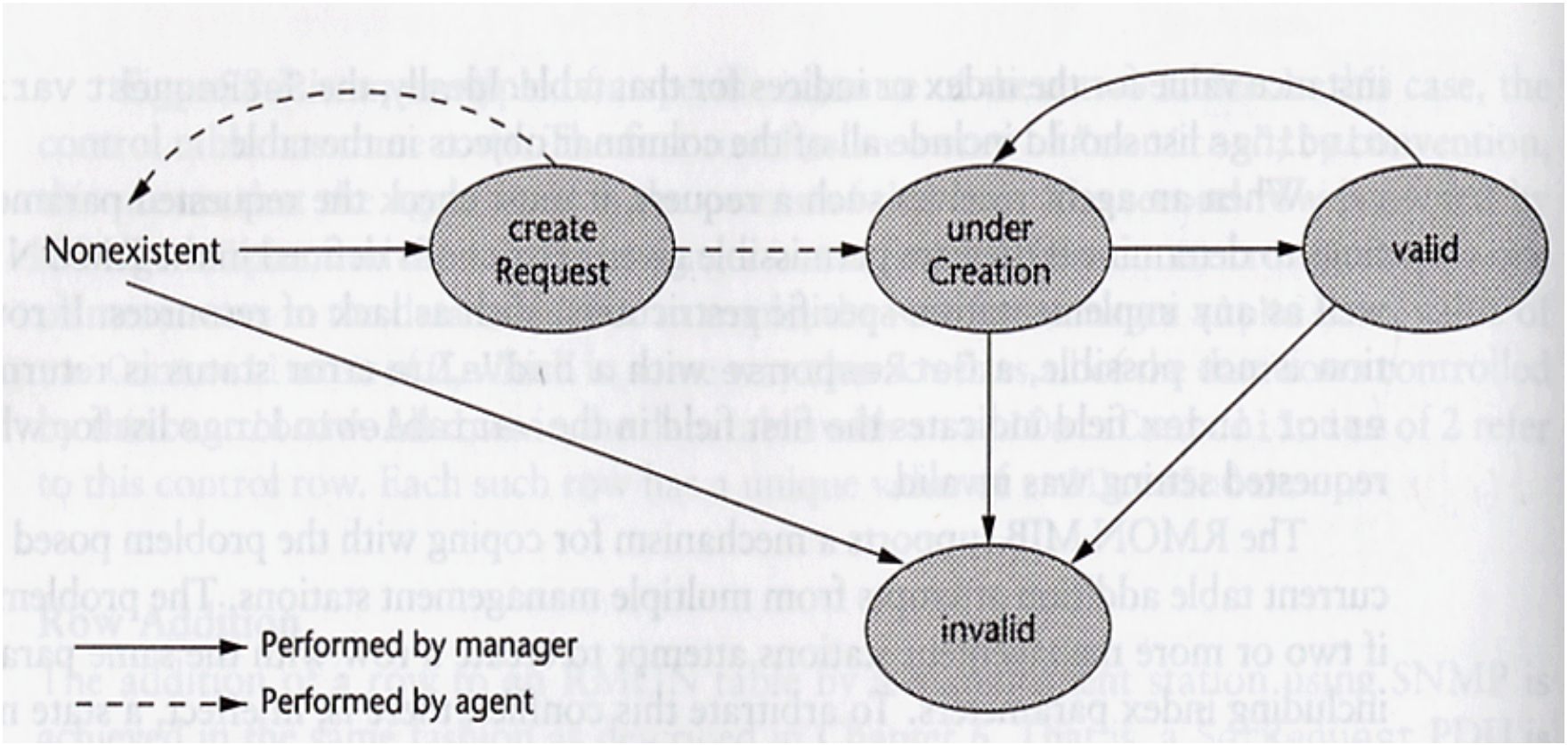
1. If a management station attempts to create a new row, and the index object value(s) do not exist, the row is created with a status object value of createRequest(2)
2. Immediate after completing the create operation, the agent sets the status object value to underCreation(3)
3. Rows shall exist in underCreation state until the management station has finished creation. At this point, the management station sets the status valid (1)
4. If an attempt is made to create a new row and the row already exists, an error will be returned.

Row Modification and Deletion

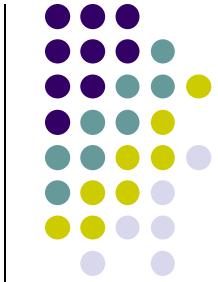


- A row is deleted by setting the status object value (by manager) for the row to Invalid.
- The owner of the row can therefore delete by issuing the appropriate SetRequest PDU.
- A row can be modified by first invalidating the row and then providing the row with new parameter values.

Transitions of EntryStatus state

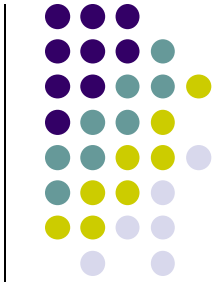


RMON1 MIB (mib-2 16)



- The RMON MIB is divided into ten groups:
- Statistics (1)
- History (2)
- Alarm (3)
- Host (4)
- HostTopN (5)
- Matrix (6)
- Filter (7)
- Capture (8)
- Event (9)
- tokenRing (10)

Five Groups for Traffic Statistics

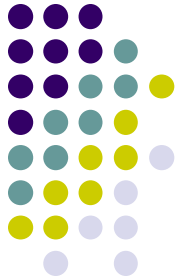


- statistics group
  - one control-data table -- etherStatsTable
- history group
  - one control table -- **historyControlTable**
  - one data table -- **etherHistoryTable**
- host group
  - one control tables -- **hostControlTable**
  - two data tables – **hostTable and hostTimeTable**
- hostTopN group
  - one control table -- **hostTopNControlTable**
  - one data table -- **hostTopNTable**
- matrix group
  - one control table -- **matrixControlTable**
  - two data table – **matrixSDTable and matrixDSTable**

Statistics Group (1/2)

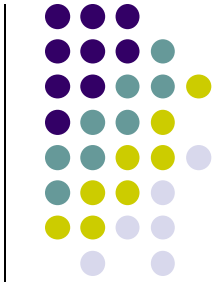
- Statistics are in the form of counters that start from zero when a valid entry is entered
- Collect a variety of counts for each attached subnetwork, including byte, packet, error, and frame size counts. It is for use with Ethernet interface.
- Indexed by **etherStatsIndex** in **etherStatsTable**
- Read-write objects: **etherStatsDataSource, etherStatsOwner**, and **etherStatsStatus**

Statistics Group



- Two noncounter objects:
  - **etherStatsIndex** – an integer row index
    etherStatsDataSource – identify the interface that is the source of the data in this row
- Provide useful information about the **load** on a subnet and the **overall health** of the subnet (CRC alignment errors, collisions, undersized or oversized packets)
- Statistics on traffic into the monitor across an interface vs total traffic into and out of an agent's interface (MIB-2)

History Group

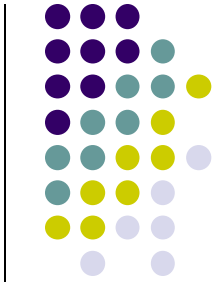


- Sample scheme is dictated by historyControlBucketGranted (50) and historyControlInterval (1800 s)
- etherHistoryUtilization =

  (etherStatsPkts*(96+64) + etherStatsOctets*8) / (historyControlInterval * 100 Mbits) * 100%, where 64-bit preamble and 96-bit inter-frame gap
- Each row of etherHistoryTable, called a bucket, holds the statistics gathered during one sampling interval.
- etherHistoryPkts = difference between etherStatsPkts at the end of that sampling interval and the beginning

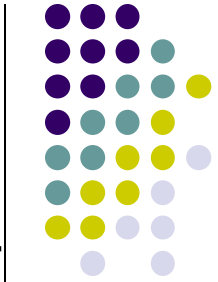
  (refer to p226,229 of the textbook for details)

Host Group



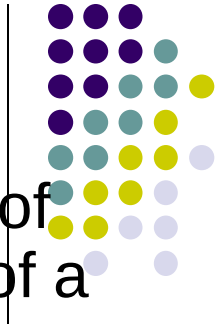
- Gather statistics about specific hosts on the LAN
- hostControlTableSize determines the number of rows in hostTable and hostTimeTable, N=number of rows in hostTable = sum of hostControlTableSize of each index
- hostTable is indexed by MAC address of the host as well as by the interface index (as in the probe)
- hostTimeTable contains the same information as hostTable but indexed by the creation order rather than the MAC address
- All the information in host group is obtainable directly from each host via MIB-II information in the interfaces group

HostTopN Group



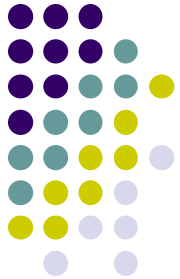
- Maintain statistics about the set of hosts on one subnetwork that top a list based on some parameter
- Data is derived from **host group**
- Manager creates a row of control table to specify a new report. The control entry instructs the monitor to measure the difference between the beginning and ending values of a particular host group variable over a specified sampling interval.
- The sampling period value is stored in both **hostTopNDuration** and **hostTopNTimeRemaining**
- If the manager wishes to generate an additional report for a new time period, it first gets the results then resets **hostTopNTimeRemaining** to the value in **hostTopNDuration,** which causes the associated data rows to be deleted and a new report to be prepared.

Matrix Group



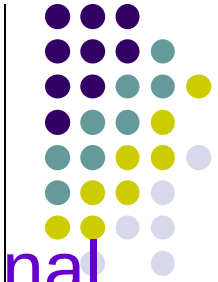
- Record information about the traffic between pairs of hosts on a subnetwork, and store info. in the form of a matrix.
- Useful in finding out which devices are making the most use of a server.
- When the monitor detects a new conversation that involves a new host pairing, it creates two new rows in both data tables. If the limit specified in **matrixControlTableSize** is reached, the least recently used entries are deleted.
- **matrixSDTable** stores statistics on the traffic from a particular source host to a number of destinations. It is indexed by **matrixSDSourceAddress, matrixSDDestinationAddress,** and **matrixSDIndex**.

Four Groups for Alarms and Filters



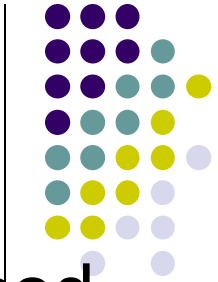
- alarm group
  - 1 control table -- alarmTable
- filter group
  - 2 control table – filterTable and channelTable
- packet capture group
  - 1 control tables -- bufferControlTable
  - 1 data table -- captureBufferTable
- event group
  - 1 control table -- eventTable
  - 1 data table -- logTable

Group Dependencies



- All of the groups in the RMON MIB are optional. However, the following three groups are dependent:
  - The alarm group requires the implementation of the event group
  - The hostTopN group requires the implementation of host group
  - The packet capture group requires the implementation of the filter group
  - The filter group requires the implementation of the event and the packet capture group

Alarm Group (1C) (1/3)



- Is used to define a set of **thresholds** for network performance. If a threshold is crossed, an alarm is generated and sent to the NMS.
- For example, an alarm could be generated if there are **more than 500 CRC errors** in any 5-minutes period.
- Object types of **alarmVariable**: INTEGER, counter, gauge, and TimeTicks.
- **alarmSampleType**: absoluteValue(1) or deltaValue(2)
- **alarmStartupAlarm**: risingAlarm(1), fallingAlarm(2), or risingOrFallingAlarm(3)

Alarm Group (1C) (2/3)

- Rules for the generation of rising-alarm events:

1. if the first sampled value obtained after the row becomes valid is less than the rising threshold, then a rising event is generated the first time that the sample value becomes greater than or equal to the rising threshold.

2. if the first sampled value obtained after the row becomes valid is greater than or equal to the rising threshold, and if the value of **alarmStartupAlarm** is 1 or 3, then a rising event is generated.

Alarm Group (1C) (3/3)

- Rules for the generation of rising-alarm events:

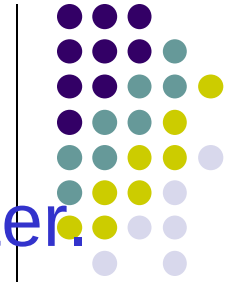
3. if the first sampled value obtained after the row becomes valid is greater than or equal to the rising threshold, and if the value of **alarmStartupAlarm** is 2, then a rising event is generated the first time that the sample value again becomes greater than or equal to the rising threshold after having falling below the rising threshold.

4. After a rising-alarm event is generated, another such event will not be generated until the sampled value has fallen below the rising threshold, reached the falling threshold, and reached the rising threshold again.

- Hysteresis Mechanism

  (refer to the diagram in tutorial slides of simpleweb)

Filter Group (2C) (1/5)



- Two kinds of filters: data filter and status filter.
- The stream of packets that pass the filtering test is referred to as a **channel.**
- A channel is defined by a set of filters.
- If the channel is in an enabled state, it can be configured to generate an event, defined in the **event** group.
- The packets passing through the channel can be captured if the mechanism is defined in the **capture** group.

Filter Group (1/5 Supplement)

- Channel can be configured to generate an event, defined in the event group, when a packet passes through the channel and the channel is in an enabled state, channelDataControl is on.
  - Depending upon the channelAcceptType (acceptedMatched or acceptFailed), channelEventIndex, and channelEventStatus (eventReady, eventFired, eventAlwaysready)
  - In event group, eventIndex = channelEventIndex (of channeltable), and eventType could be none, log, snmp-trap, or log-and-trap.
- The packets passing through a channel can be captured if the mechanism is defined in the capture group.
  - In packet capture group, bufferControlChannelIndex = channelIndex of channelTable

Filter Group (2C) (2/5)

- Filter Logic

  input = the incoming packets

  filterPktData = bit pattern to be tested for

  filterPktDataMask = relevant bits to be tested for (1 for relevant)

  filterPktDataNotMask = used to test for a match or a mismatch, where

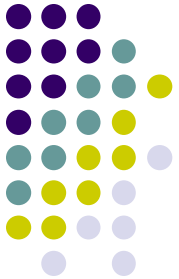
  0-bits indicate exact match is required between the relevant bits of input and filterPktData, while 1-bits for mismatch is required

- Screen for packets with a specific source address
- Screen for packets that didn't have the server as a source
- Test for packets with any multicast destination address

  Note: class D (224.0.0.0 – 239.255.255.255) for multicasting
- Screen for packets that had exact match on the DA field and mismatch on the SA field

Note: =^ stands for bitwise exclusive-or

Filter Group (2C) (3/5)



- Screen for packets with a specific source address

  if ((input =^ filterPktData) == 0) filterResult = match;

- Screen for packets that didn't have the server as a source

  if ((input =^ filterPktData) != 0) filterResult = mismatch;

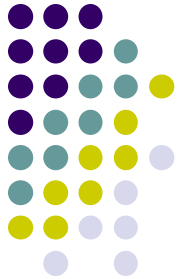
- Test for packets with any multicast destination address

```
if (( input =^ filterPktData) & filterPktDataMask) == 0)
        filterResult = match_on_relevant_bits;
else
        filterResult = mismatch_on_relevant_bits;
```

- Screen for packets that had exact match on the DA field and mismatch on the SA field

```
relevant_bits_different= (input =^ filterPktData) & filterPktDataMask
if ((relevant_bits_different &~ filterPktDataNotMask) == 0)
   filterResult = successful_match;
if (((relevant_bits_different & filterPktDataNotMask) != 0) |
   (filterPktDataNotMask = 0))
  filterResult = successful_mismatch;
```

Filter Group (2C) (4/5)



(Logic for the filter test is summarized in Figure 9.4, P260)

- Three specific tests are performed:

(1) packet must be long enough (at least as many bits as in filterPktData)

(2) Each bit set to 0 in filterPktDataNotMask indicates a bit position in which the relevant bits of the packet portion should match filterPktData

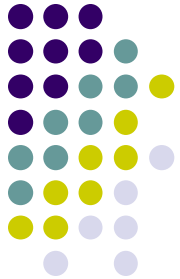
(3) Each bit set to 1 in filterPktDataNotMask indicates a bit position in which the relevant bits of the packet portion should not match filterPktData

- Example of using the filtering test

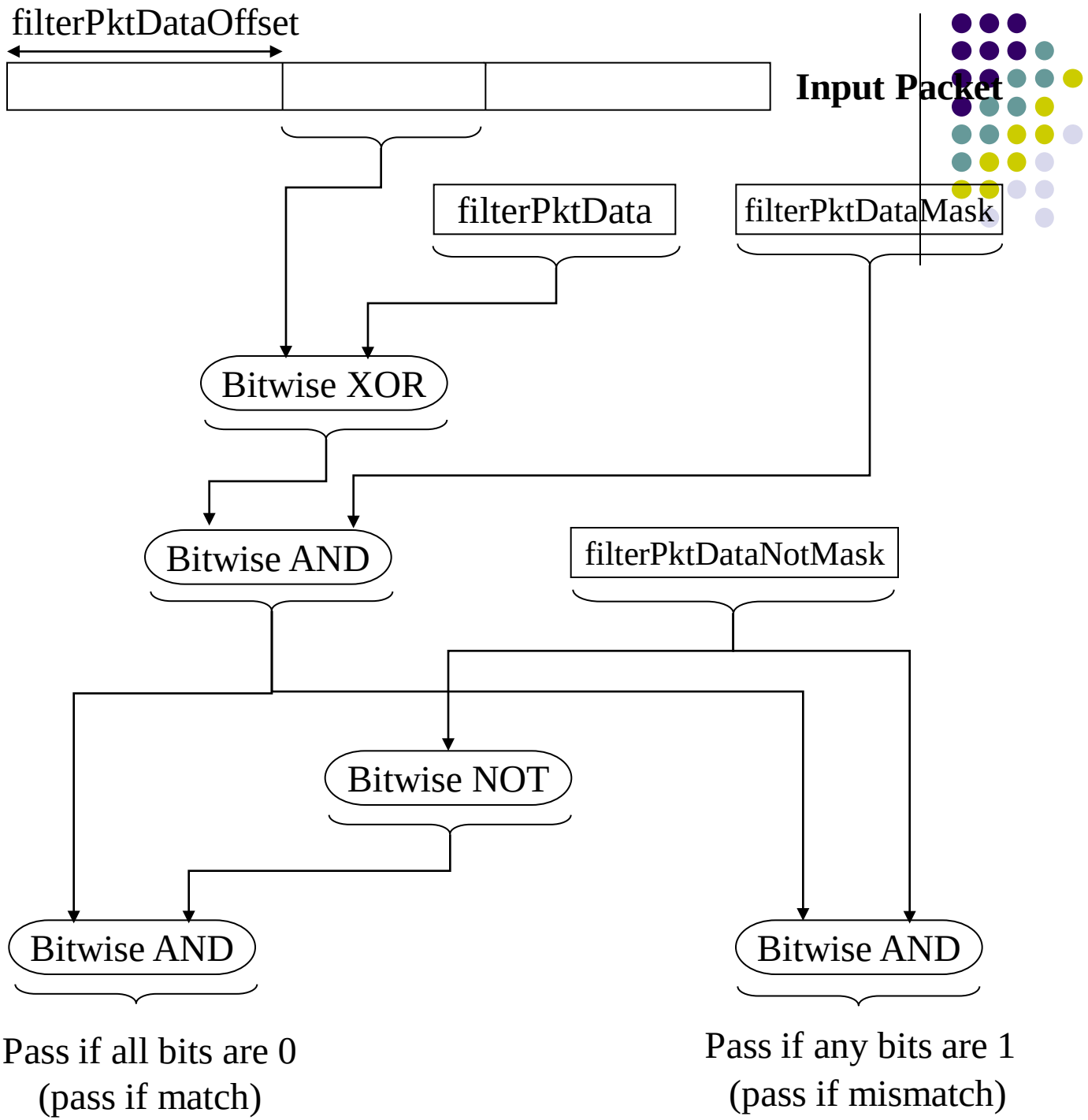
we wish to accept all Ethernet packets that have a destination address of 0xA5 and that do not have a source address of 0xBB.

- **filterPktDataOffset = 0;**
- **filterPktdata = 0x0000000000A50000000000BB**
- **filterPktDataMask = 0xFFFFFFFFFFFFFFFFFFFFFFFF**
- **filterPktDataNotMask = 0x000000000000FFFFFFFFFFFF**

Filter Group (2C) (5/5)



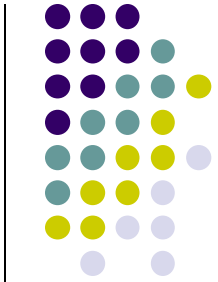
- channelAcceptType is either acceptMatched(1) or acceptFailed(2)
- If the value is 1, packets will be accepted for this channel if they pass both the data and status matches of at least one of the associated filters. (ref. fig 9.5, p262)
- If the value is 2, packets will be accepted for this channel only if they fail either the data or status match of every associated filter.
- channelEventStatus could be eventReady(1), eventFired(2), or eventAlwaysReady(3).
- If channelDataControl is on, then an event will be generated if (1) channelEventStatus has the value eventReady or eventAlwaysReady and (2) an event is defined for this channel in channelEventIndex


filterPktDataOffset
Input Packet
filterPktData
filterPktDataMask
Bitwise XOR
Bitwise AND
filterPktDataNotMask
Bitwise NOT
Bitwise AND
Bitwise AND
Pass if all bits are 0
(pass if match)
Pass if any bits are 1
(pass if mismatch)

Packet Capture Group (1C1D)

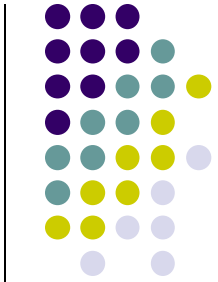
- Used to set up a buffering scheme for capturing packets from one of the channels in the filter group, bufferControlTable and captureBufferTable.
- bufferControlTable defines one buffer to capture and store packets from one channel.
- Several buffer control parameters determine how much of a packet is stored in the buffer and how much is available for delivery to a management station in one SNMP **get** or **getNext** request: ~CaptureSliceSize (CS), ~DownloadSliceSize (DS), ~DownloadOffset (DO), and captureBufferPacketData (PD), *PacketLength (PL), …
- Length of PD, PDL = MIN (PL, CS)
- PD = the octets 0..min(actualStoredData-1, 99), if DO=0 and DS=100 (default value in capture buffer)

Event Group (1C1D)



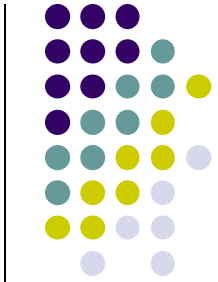
- This group supports the definition of events.
- An event is triggered by a condition located elsewhere in the MIB.
- An event can trigger an action defined elsewhere in the MIB, may cause information to be logged, and may cause an SNMP trap message to be issued.
- evenType could be one of none(1), log(2), snmp-trap(3), log-and-trap(4)
- The key use of the event group is in conjunction with the alarm group. Also, the filter group can reference an event that will occur when a packet is captured.
  - alarmRisingEventIndex or alarmFallingEventIndex matches with the eventIndex of eventTable for the conjunction

The RMON2 MIB (mib-2 16)



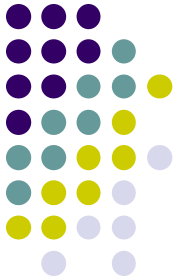
- protocolDir (11)
  - protocolDirLastChange (1)
  - protocolDirTable (2)
- protocolDist (12)
  - protocolDistControlTable (1)
  - protocolDistStatsTable (2)
- addressMap (13)
  - addressMapInserts (1), addressMapDeletes (2)
  - addressMapMaxDesireEntries (3)
  - addressMapControlTable (4)
  - addressMapTable (5)

The RMON2 MIB (mib-2 16)



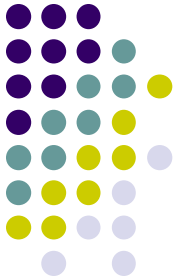
- nlHost (14)
  - nlHostControlTable (1), nlHostTable (2)
- nlMatrix (15)
  - nlMatrixControlTable (1)
  - nlMatrixSDTable (2), nlMatrixDSTable (2)
  - nlMatrixTopNControlTable (4), nlMatrixTopNTable(5)
- alHost (16)
  - alHostTable (1)
- alMatrix (17)
- userHistory (18)
  - usrHistoryControlTable (1), userHistoryObjectTable (2)
  - usrHistoryTable (3)
- probeConfig (19)
  - 10 scalars and 4 tables

RMON 2 MIB (mib-2 16)



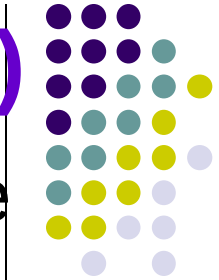
Group	OID	Function	Tables
Protocol Directory	rmon 11	Inventory of protocols	protocolDirTable
Protocol Distribution	rmon 12	Relative statistics on octets and packets	protocolDistControlTable protocolDistStatsTable
Address Map	rmon 13	Mac address to network address on the interfaces	addressMapControlTable addressMapTable
Network Layer Host	rmon 14	Traffic data from and to each host	n1HostControlTable n1HostTable
Network Layer Matrix	rmon 15	Traffic data from each pair of hosts	n1MatrixControlTable n1MatrixSDTable n1MatrixDSTable n1MatrixTopNControlTable n1MatrixTopNTable

RMON 2 MIB (mib-2 16)



Application Layer Host	rmon 16	Traffic data by protocol from and to each host	a1HostTable
Application Layer Matrix	rmon 17	Traffic data by protocol between pairs of hosts	a1MatrixSDTable a1MatrixDSTable a1MatrixTopNControlTable a1MatrixTopNTable
User History Collection	rmon 18	User-specified historical data on alarms and statistics	usrHistoryControlTable usrHistoryObjectTable usrHistoryTable
Probe Configuration	rmon 19	Configuration of probe parameters	serialConfigTable netConfigTable trapDestTable serialConnectionTable
RMON Conformance	rmon 20	RMON2 MIB Compliances and Compliance Groups	See Section 8.4.2

Network & Application Visibility (1/2)



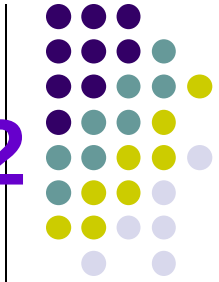
- RMON2 provides the probe capability above the MAC level
  - Based on network-layer protocol and IP-address
  - Based on an application-level traffic
- The capability of seeing above MAC layer can answer questions, such as:
  - If a router is overloaded because of high amount of outgoing traffic, what local hosts are responsible for, and to what destination?
  - If there is a high load of pass-through traffic, arriving via one router and departing via another router, what networks or hosts are responsible for the bulk of this traffic?

Network & Application Visibility (2/2)

- In Network-Layer Visibility, RMON2 probe can not only monitor the total traffic into and out of routers(RMON1 does), but also to determine the ultimate source of incoming traffic arriving via the router or the ultimate destination of outgoing traffic leaving via the router.
- Generate charts and graphs depicting traffic percentage
  - By protocols
  - By applications

    (any protocol above the network layer is considered as “application level”)

New Functional Features in RMON2

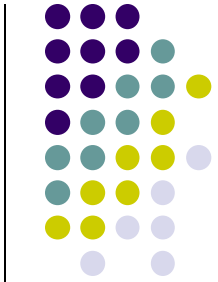


- Two new features of table indexing:
  - Indexing with external objects

    SMI for SNMPv2 defines a possible usage for an object not part of a particular table as an index for that table. RMNO2 adopts this definition.
  - Time filter indexing

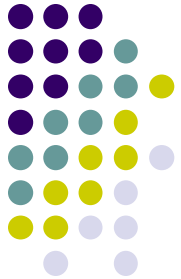
    return values only for those values have changed since last poll.
  - Status objects are specified as having syntax RowStatus (textual convention defined in SNMPv2, p358) rather than EntryStatus.

EntryStatus and RowStatus



- EntryStatus :: INTEGER{valid (1),
  createRequest (2),
  underCreation (3),
  invalid (4)}
- RowStatus :: TEXTUAL-CONVENTION
  STATUS current
  DESCRIPTION …
  SYNTAX INTEGER { active (1),
  notInService (2),
  notReady (3),
  createAndGo (4),
  createAndWait (5),
  destroy (6)}

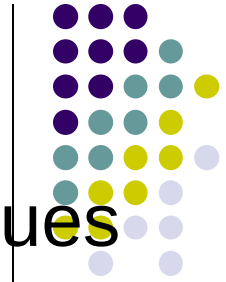
Indexing with External Objects



- Refer to the examples in p217 and p282

- Note that the data table has one few object in the table definition for the table indexed by an external object.

Time Filter Indexing



- The probe return values of objects whose values have changed since the last poll.
- The mechanism used in RMON2 relies on a new textual convention, defined as follows:

TimeFilter ::= TEXTUAL-CONVENTION

STATUS CURRENT

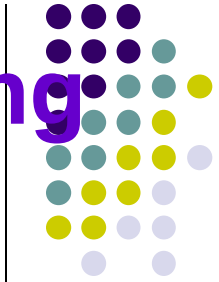
DESCRIPTTION

“. . .”

SYNTAX TimeTicks

- TimeFilter is used exclusively as an index to a table.The purpose of this index is to enable a manager to download from a probe’s table only those rows that have changed since a specified time.

Table def. of using Time Filter Indexing



```
fooTable OJECT-TYPE
   SYNTAX      SEQUENCE OF FooEntry
   ACCESS      not-accessible
   STATUS      current
DECRIPTION “a control table”
::= {ex 1}

footEntry OBJECT-TYPE                    FooEntry :: = SEQUENCE {
   SYNTAX      FooEntry                   fooTimeMark     TimeFilter,
   ACCESS      not-accessible             fooIndex        INTEGER,
   STATUS      current                    fooCounts       counter32}
DESCRIPTION “one row in footable”
INDEX    {fooTimemark, fooIndex}

fooTimeMark OBJECT-TYPE
   SYNTAX      TimeFilter
   ……
```

Implementation of TimeFilter indexing

fooTimeMark (fooTable.1.1)	fooIndex (fooTable.1.2)	fooCounts (fooTable.1.3)
0	1	5
0	2	9
1	1	5
1	2	9
2	1	5
2	2	9
3	1	5
3	2	9
4	1	5
4	2	9
5	1	5
5	2	9
6	1	5
6	2	9
7	2	9
8	2	9

Conceptual view of fooTable at sysUpTime ≥8

Example of using TimeFilter indexing

TimeStamp	fooIndex (fooTable.1.2)	fooCounts (fooTable.1.3)
6	1	5
8	2	9

Possible implementation view of fooTable at probe

Application scenario (1) -- Time Filtering

- Suppose that the current value of the counter associated with fooIndex = 1 is 5 and the counter was most recently updated at time 6.

- Suppose that the current value of the counter associated with fooIndex = 2 is 9 and the counter was most recently updated at time 8.

- At time 10, the manager issues the following request:

  GetRequest (fooCounts.7.1, fooCounts.7.2), which those values have been updated since time 7.

  the response is: Response (fooCounts.7.2 = 9)

Changes in Agent's Implementation view

TimeStamp	fooIndex	fooCounts
0	1	0
0	2	0

Time=0

TimeStamp	fooIndex	fooCounts
900	1	2
1100	2	1

Time=1100

TimeStamp	fooIndex	fooCounts
500	1	1
0	2	0

Time=500

TimeStamp	fooIndex	fooCounts
900	1	2
1400	2	2

Time=1400

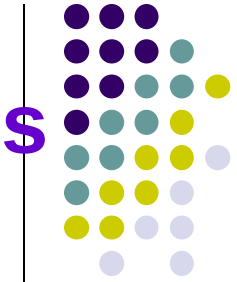
TimeStamp	fooIndex	fooCounts
900	1	2
0	2	0

Time=900

TimeStamp	fooIndex	fooCounts
2300	1	3
1400	2	2

Time=2300

The Basic Row was updated as follows



- Row1 (fooIndex = 1):

sysUpTime	fooCounts.*.1
500	1
900	2
2300	3

- Row2 (fooIndex = 2):

sysUpTime	fooCounts.*.2
1100	1
1400	2

Application scenario (2) -- Time Filtering

- Assume that (1) the manager polls the probe every 15 seconds. (2) The time difference between the manager and agent is 400 ms.

  (3) The manager keeps a clock, **nms**, that records time in ms.

- At nms = 1000, the manager does a baseline poll to get everything since the last agent restarted (TimeFilter = 0)

  GetRequest (sysUpTime.0, fooCounts.0.1, fooCounts.0.2)

  Response (sysUptime.0=600, fooCounts.0.1=1, fooCounts.0.2=0)

Application scenario (3) -- Time Filtering

- At nms = 2500, the manager gets an update on all changes since last report (agent time 600):

  GetRequest (sysUpTime. 0, fooCounts.600.1, fooCounts.600.2)

  Response (sysUptime.0=2100, fooCounts.600.1=2, fooCounts.600.2=2)

- At nms = 4000, the manager gets an update on all changes since last report (agent time 3600):

  GetRequest (sysUpTime.0, fooCounts.2100.1, fooCounts.2100.2)

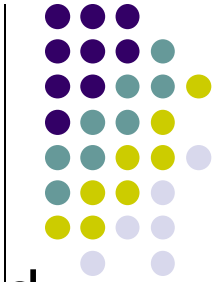
  Response (sysUptime.0=3600, fooCounts.2100.1=3)

- At nms = 5500, the manager gets an update on all changes since last report (agent time 5100): :

  GetRequest (sysUpTime.0, fooCounts.3600.1, fooCounts.3600.2)

  Response (sysUptime.0=5100)

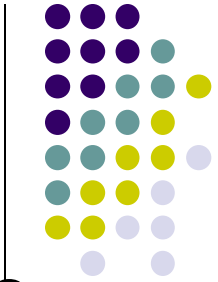
Protocol Directory Group



- One scalar object, protocolDirLastChange, and one table, protocolDirTable, for an RMON2 manager to learn which protocols a probe interprets.

- The group includes:
  - protocolDirTable covers MAC-, Network-, and higher-layer protocols and is indexed by protocolDirID and protocolDirParameters.
  - protocolDirLastChange contains the time of the last table update.

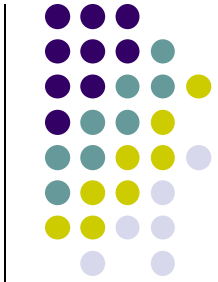
Protocol identifier (1/2)



- The protocolDirID object contains a unique octet string for a specific protocol.
- The root of the tree is the identifier of a MAC-level protocol.
- Each protocol level is identifiers by one or more 32-bit values, and each such value is encoded as four subidentifiers, [a.b.c.d].
- Several well-known assigned numbers:

ether2 = 1 [0.0.0.1] llc = 2 [0.0.0.2]
snap = 3 [0.0.0.3] vsnap = 4 [0.0.0.4]
ianaAssigned = 5 [0.0.0.5]

Protocol Identifier (2/2)

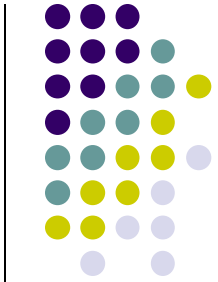


- General format

  cnt protocolDirID cnt protocolDirParameters
- IP over Ethernet MAC type is 0x0080, UDP over IP with protocol value 17,and SNMP uses UDP port number 161. protocolDirID has velue:

  ether2.ip.udp.snmp (16.0.0.0.1.0.0.8.0.0.0.0.17.0.0.0.161)
- protocolDirParameters is structured as a one-octet count field followed by a set of N-octets parameters, one for each protocol.
- Protocol directory table also contains LocalIndex, Descr, Owener, Status, and 3 configuration enumerated values: DirHostConfig, HostConfig, and MatrixConfig

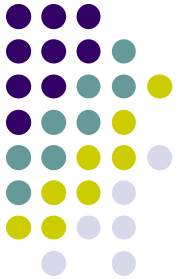
Protocol Distribution Group



- Summarize how many octets and packets have been sent from each of the protocol supported.
- protocolDistControlTable controls collection pf basic statistics for all supported protocols, and protocolDistStatsTable records the data.
- protocolDistStatsTable includes one row for each protocol in protocolDirTable for which at one packet has been seen. It is indexed by protocolDistControlIndex and protocolDirLocalIndex.
- protocolDistStatsTable contains two objects:

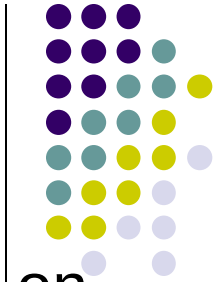
  protocolDistStatPkts and protocolDistStatsOctets

Address Map Group (1/2)



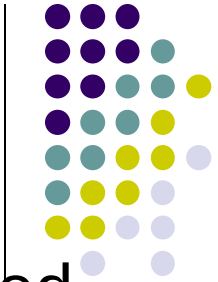
- Match each network address to a specific MAC-address and to a specific port on the network interface.
- Helpful in node discovery and network topology applications.
- Contain 3 scalars objects, one control table addressMapControlTable, and one data table addressMapTable.
- There is a single central data table containing entries that provide the mapping between IP and MAC addresses.
- Data table is not indexed by a row of the control table.

Address Map Group (2/2)



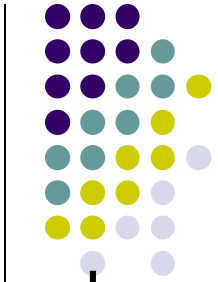
- addressMapTable collects address mappings based on source MAC and network addresses.
- An entry is created for all protocols in the protocol directory table whose value of protocolDirAddressMapConfig is equal to supportedOn(3).
- addressMapTable is indexed by 4 objects: ~TimeMark, protocolDirLocalIndex, ~NetworkAddress, ~Source.
- Given a network address for a particular protocol observed on a particular interface within a particular amount of time, the MAC address for that network address can be read.
- This group is useful in detecting duplicate IP addresses.

Host Groups



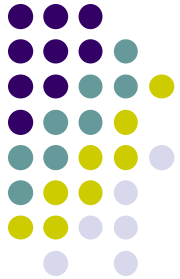
- nlHost group allows users to decode packets based on their network-layer addresses.
- alHost group creates entries for all application layer protocols in the protocol directory table whose value of ~AlHostConfig is equal to supportedOn(3).
- nlHostTable is indexed by 4 objects: ~ControlIndex, ~TimeMark, ~Address, and protocolDirLocalIndex.
- alHostTable is indexed by 5 objects: ~TimeMark, nlControlIndex, nlHostAddress, protocolDirLocalIndex, protocolDirLocalIndex (for network and application protocols)
- alHostTable allows users to trace traffic in/out of a host on the basis of application protocol.

Matrix Groups (1/2)



- nlMatrix and alMatrix groups gather statistics based on network layer address and application layer protocol
- nlMatrix contains 2 control tables and 3 data tables (2 for matrix statistics, 1 for topN statistics).
- HostTopN ranks individual hosts on one subnetwork, while RMON2 TopN statistics ranks the traffic between pairs of hosts.
- alMatrix contains 1 control table and 3 data tables.
- alMatrixDSTable is indexed by 6 objects from 3 tables.

Matrix Groups (2/2)



- An sample object instance of alMatrixSDTable : **alMatrixSDPkts.1.783459.18.4.128.2.6.6.4.128.2.6.7.34**

  [1] the first row of nlMatrixControlTable

  [783497] time in time ticks for this row

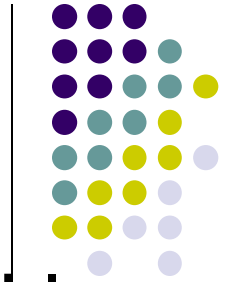
  [4.128.2.6.6] and [4.128.2.6.7] specify the source and destination hosts

  [18] specify the network layer protocol defined by row 18 of protocolDirTable

  [34] specify the application layer protocol defined by row 34 of protocolDirTable

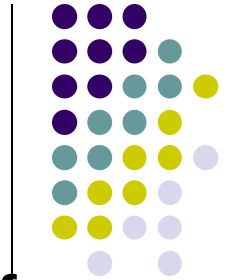
- Both nlMatrixTopNRateBase and alMatrixTopNRateBase specify two objects for table sorting: ~TopNPkts and ~TopNOctets

User History Collection Group



- Periodically polls particular statistics/variables and then logs that data based on user-defined parameters
- Allows the network manager configures history studies of any counter in the system
- Three-level hierarchy of tables: userHistoryControlTable (for sampling details), userHistoryObjectTable, and userHistoryTable.

Probe Configuration Group



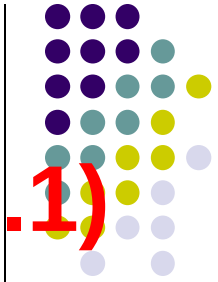
- Defines a set of configuration parameters for probes to enhance interoperability among probes and managers.

- Contains 9 scalar objects and 4 tables: serialConfigTable, netConfigTable, trapDestTable, and serialConnectionTable

Extensions to RMON1 for RMON2 Devices

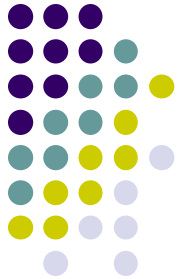
- A createTime object is added to all control table
- A droppedFrames object is added to a number of tables, included as a filter object in all filter definitions
- The object filterProtocolDirLocalIndex is added to filterTable

APPENDIX:
Abstract Syntax Notation One (ASN.1)



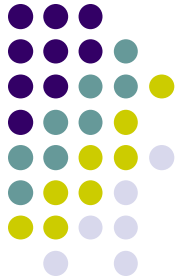
- ASN.1:
  - **ISO/ITU-T Standards: ISO 8824/ITU-T X.208**
- Abstract Syntax:
  - **Use a syntax to define data/data structure independent of machine-oriented structures and restrictions.**
- Use in SNMP
  - Define SNMP PDU format
  - Define management information (MIB)

What are defined using ASN.1



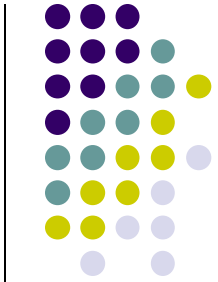
- Types:
  - data structures
  - e.g. Counter, Gauge, IpAddess, ...
- Values:
  - instances (variables) of a type
  - e.g. sysContact, ifTable, ifSpeed, ...
- Macros:
  - used to change the actual grammar of ASN.1
  - e.g. OBJECT-TYPE, ACCESS, ...

Modules



- Module: A collection of ASN.1 descriptions
- Module Structure
  - <module name> DEFINITION ::= BEGIN
  - <module body>
  - END
- Example
  - EmptyModule DEFINITION ::= BEGIN
    END

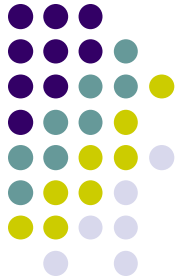
Tags and Types (1)



- Tags
  - Every type defined in ASN.1 is assigned a tag
  - Tag = Class + Number
    - Class: (Bit 8,7 in BER tag)
      - Universal 0 0
      - Application 0 1
      - Context-specific 1 0
      - Private 1 1
    - Number: non-negative Integer

BER: Basic Encoding Rules

Tags and Types (2)

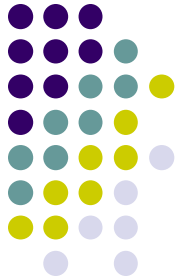


Universal Tag	ASN.1 Type
1	BOOLEAN
2	INTEGER
3	BIT STRING
4	OCTET STRING
5	NULL
6	OBJECT IDENTIFIER
7	ObjectDescriptor
8	EXTERNAL
9	REAL
10	ENUMERATED
12-15	Reserved
16	SEQUENCE, SEQUENCE OF
17	SET, SET OF

-

Universal Tag	ASN.1 Type
18	NumericString
19	PrintableString
20	TeletexString
21	VediotextString
22	IA5String
23	UTCTime
24	GeneralizeTime
25	GraphicString
26	VisssibleString
27	GeneralString
28	CharacterString
29-...	Reserved

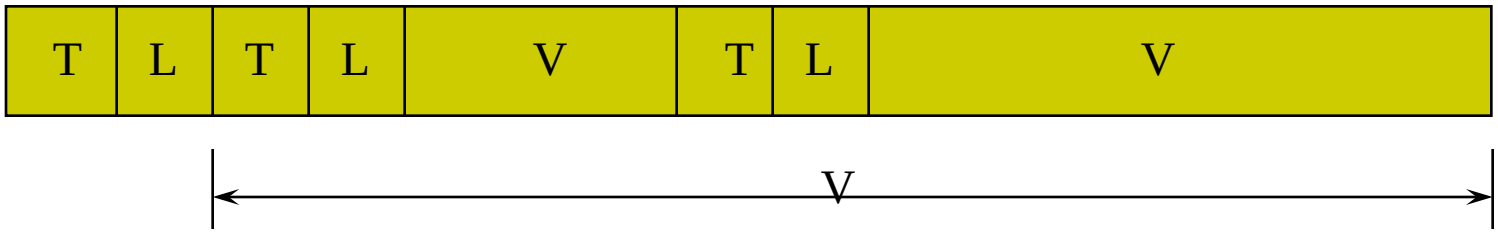
Values in ASN.1



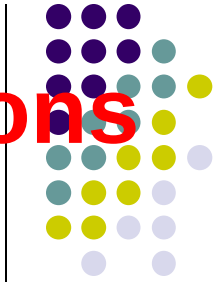
- General format of a value assignment
  - <valuereference> <type> ::= <value>
- Examples:
  - BOOLEAN
    - Married ::= BOOLEAN
    - currentStatus Married ::= FALSE
  - INTEGER
    - Color ::= INTEGER{red (0), blue (1), yellow (2)}
    - defaultColor Color ::= 1
    - defaultColor Color ::= blue

Basic Encode Rules

- BER
  - A ***transfer syntax notation***
  - ISO/ITU-T Standards: ISO 8825/ITU-T X.209
  - Values from any abstract syntax defined using ASN.1 can be encoded with BER
  - BER uses Tag, Length, Value (TLV) encoding
    - **Tag**: "identifier", **Length**: length of content, **Value**: "contents"
  - Each value may itself be made up of one or more TLV-encoded values



BER Numeric and Tag Representations



- Integer Representations
  - 2's complement for positive & negative
    - The first 9 bits can't be 0- or 1- filled.
  - Unsigned presentation for non-negative integers
    - The first 9 bits can't be 0- or 1- filled.

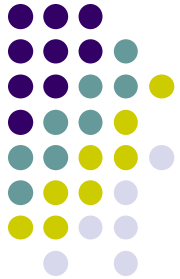
  - There are four classes of tags in ASN.1:
    - universal tags, application tags, context-specific tags, and private-use tags
    - If an ASN.1 type is primitive, it may be sent either in primitive form or in constructed form.
    - If the ASN.1 type is constructed, then it is always sent in constructed form.

  - Tag is encoded as: class(2-bit), flag(1-bit), number(5-bit)

BER Tag and Length Representations

- Tag Encoding
  - 2 bits for class field
    - (0,0) for universal, (0,1) for application-wide, (1,0) for context-specific, (1,1) for private-use
  - 1 bit for primitive or constructed type
    - 0 for primitive and 1 for constructed
  - 5 bits for non-negative tag number
    - If the number is less than 31, then it is simply encoded
    - If it is larger than 31, then the first 5 bits are set to all ones, 7 bits are used in the rest octets. The most significant bit 0 indicates the last octet.
- Length Encoding
  - If the length is less than 128, then one octet is used.
  - If the length is longer, then more than one octet is used,and the first octet has the high-order bit set to 1.
  - Two-pass length determination
    - The first to calculate the length, and the second to do the actual encoding

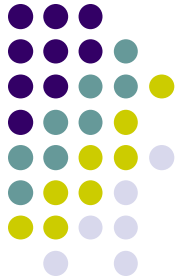
BER Value Representations



- Simple Types
  - INTEGER
    - In primitive form using 2's complement
  - BIT STRING
    - The first octet indicates how many bits are unused in the final content octet.
    - If no bits are presented in the BIT STRING value, then the single octet is encoded with value 0.
  - OCTET STRING
  - NULL
    - A zero contents octet is encoded.
  - OBJECT IDENTIFIER
    - 40*x + y is the first sub-identifier, for example, 1.0.8571.5.1 is encoded with 40, 8571, 5, and 1. Except the first two ids, all others are treated as unsigned int.

- Constructed Types
  - SEQUENCE
    - 00-1-16 for the initial tag octet
  - SEQUENCE OF
    - 00-1-16 for the initial tag octet
  - Tagged Types
    - For example, SomeType :: = [APPLICATION 7] OtherType is encoded as 01-1-7 in the first tag octet

Basic Encode Rules --- Example



- BER deficiencies
  - Problem of processing efficiency
  - Increase the load of managed node

(Refer the simple book, p.310)