Controlling and Monitoring Audio Systems with Simple Network Management Protocol (SNMP)

Kevin P. Gross and Tom Holtzen, Peak Audio, Inc., Boulder, CO, USA

Introduction

Having developed a means in CobraNet for delivering high quality real time-audio over a standard Ethernet network, we have recently turned our attention to the implementation of an accompanying control protocol for CobraNet devices. CobraNet is a combination of hardware, software and protocol for transporting isochronous, asynchronous and clock data over an Ethernet computer network. Isochronous refers to timing sensitive services such as live audio, video and real-time control. A CobraNet device consists of a 100Mbit Ethernet controller and digital signal processor with associated memory and audio I/O. A CobraNet device may be managed through a parallel data connection by an optional local microcontroller. We sought to expand this management capability to allow remote management through the Ethernet connection. This paper details our experiences in design/selection and implementation of a control and monitoring protocol and details the workings of the protocol we selected for this application – Simple Network Management Protocol (SNMP).

Protocol Selection

Our foremost concern in producing or selecting a control and monitoring protocol was to achieve maximal functionality using the simplest possible protocol implementation. A simple protocol conserves memory and development time. Furthermore we believe a simple system is less likely to contain hidden flaws. We envisioned a client server approach with each CobraNet device containing a server element that could respond to requests from a client – typically some sort of workstation or control surface. We saw the possibility, even likelihood of having multiple clients in a system. We considered it a requirement that whatever protocol we adopted or designed offer support of multiple clients and, of course, multiple servers.

We believe CobraNet derives significant benefit from its use of the standard Ethernet family of networking technologies. Wherever possible, we have attempted to adopt standard network methodologies. We believe that any proprietary protocol for control and monitoring has to fully justify itself. To this end, we have put substantial effort into surveying existing Ethernet control and monitoring protocols.

Just as the ubiquity of Ethernet drove us to adopt Ethernet as the physical media for CobraNet, the ubiquity of Internet technology has driven us to favor the Internet Protocol (IP) as a transport mechanism. The transport mechanism serves as the addressing and delivery framework for all control protocol messages. It initially appeared as though an IP implementation would be a complex undertaking. This was indicated by examination the size some of the commercial and public domain C code implementing IP. The lightest of these implementations weighed in at 72K bytes of object code. Closer examination of IP reveals it a suite of independent and interdependent protocols. The bulk of the

complexity in the Internet protocol suite is in the Transmission Control Protocol (TCP). TCP allows for reliable communications over an unreliable physical communications link. It handles retransmission of lost packets, arrival of packets out of order and duplicated packets. While a TCP level of service is required for many networking applications, SNMP does not desire nor require the TCP services. SNMP is conventionally carried on a truly simple User Datagram Protocol (UDP) transport. In addition to UDP and the IP transport itself, we determined that a couple other protocols from the IP suite would be required. The Address Resolution Protocol (ARP) translates IP addresses to Ethernet Media Access Control (MAC) addresses. The Internet Control Message Protocol (ICMP) provides basic diagnostic services for a network. We were confident that a minimally functional UDP/IP stack could be composed with reasonable effort. We went ahead and completed the IP transport portion of the project before knowing what protocol we'd be running over it. Our experience concurred with and contributed to the finding that IP networking is considerably scalable.

Our initial inclination was towards development of a proprietary protocol. We believed our requirement for a small footprint and the desire to combine the local and network monitoring functions were unique. We had gone so far as to publish a proposal for a simple control and monitoring protocol called Management Interface (MI). MI would be easy to implement because it was designed with our application and the capabilities of our processing platform in mind. MI began to lose favor when portions of the protocol supporting network management were deemed, by some, as overly complex for local management. They were right. As much as we desired to kill two birds with one stone, local communications is a different problem than network communications. A local communications protocol operates in a homogenous environment where data and link integrity issues are not a concern. As we went through design and implementation of a much simpler protocol for local communications dubbed Host Management Interface (HMI), we became convinced that different protocols would be required for the two different applications. We proceeded to tackle the network protocol separately.

Our survey of control and monitoring protocols included Echelon LonTalk, CEBus Home Plug and Play (HPnP) draft versions of the AES24 control and monitoring protocol, SNMPv1 and SNMPv3, Common Management Information Protocol (CMIP), network Remote Procedure Call (RPC) and Microsoft's Common Object Model (COM). It was obvious that the RPC protocols to a significant extent and COM to a great extent were not designed to operate in an embedded context. We viewed the traditional network management protocols, SNMP and CMIP as largely mutually exclusive. The same can be said of the stand-alone control and monitoring protocols, LonTalk and HPnP. In each of these categories we choose the most applicable protocol for more in-depth study. We gave special consideration to AES24 because it targets audio applications. This process of elimination left us with SNMP, LonTalk, AES24 and our MI protocol for further consideration.

SNMP, LonTalk and AES24 take divergent approaches to control and monitoring of network connected devices. Control and monitoring via SNMP and LonTalk is accomplished almost exclusively through inspection and alteration of management

variables. This exclusive use of variables for control and monitoring is known as a descriptive approach. The advantage of a descriptive approach is in reduction of protocol primitives. The descriptive approach typically requires but two primitives – one to inspect values and another to alter them. AES24 takes a more functional approach in that control and monitoring is accomplished through the remote invocation of methods upon managed objects. The advantage of the functional approach is that users are comfortable with the way these systems operate. The functional "load preset 3" represents an arguably more natural approach than an alternative descriptive interface which might be implemented with "target preset number" and "current preset number" variables. With regards to functional versus descriptive, we sided with the designers of SNMP who in addition to recognizing the simplifying benefits of a descriptive approach, identified the potential for an ever increasing command set including commands with arbitrarily complex semantics in a functional approach.

SNMP requires a variable be polled in order to observe changes. LonTalk does the work of monitoring network variables for changes thus freeing applications from the overhead of doing so. This magic comes at the cost of increased complexity, however. LonTalk was designed to be carried by a proprietary transport protocol over a proprietary physical media known as LonWorks. LonTalk has since been released from LonWorks and may be carried over Ethernet, though LonWorks remains its primary target. Since SNMP was designed with UDP/IP and Ethernet in mind from the outset, it fit much more nicely into our requirements.

SNMP was developed under the auspices of the Internet Engineering Task Force (IETF). This is the same organization that standardized the IP protocol suite as well as many of the other protocols present on the Internet today. The charter for development of SNMP was to produce a truly simple management protocol to encourage rapid adoption and deployment of network management for the Internet. The resulting SNMP is conceptually very simple and was enthusiastically deployed almost immediately upon introduction in 1988. SNMP defines a total of five protocol primitives and a uniform addressing scheme for all management variables. SNMP variables as well as the protocol itself are described using an academic syntax called Abstract Syntax Notation One (ASN.1). SNMP uses variable length addressing and the Basic Encoding Rules (BER) packet construction associated with ASN.1.

Having narrowed in on SNMP as a potential control and monitoring protocol for CobraNet, we still had the feeling that our MI protocol was simpler and more functionally appropriate. Our doubts about using SNMP were allayed when we looked at what it would be giving us. An SNMP managed device can be managed using off-the-shelf network management packages such as HP OpenView, Tivoli NetView, Cabletron SPECTRUM or Computer Associates Unicenter TNG. Given these choices, custom or single-sourced software is not necessary to monitor and manage your network. If a custom interface is desired, SNMP is well supported at the manager side with tools and Application Programmers Interface (API) layers to get even higher level Visual Basic and database applications talking SNMP. SNMP is also well supported at the agent side. Several vendors offer SNMP agent implementations for embedded systems. Major

workstation operating systems typically include an SNMP agent. Commercially available simulators and test suites can assist in development of an SNMP agent. And all of this support would not be available if SNMP was not the well-deployed and dominant network management standard that it is.

SNMP Architecture

SNMP operates according to a client server model. Under SNMP, servers go by the name "agent" and clients are called "managers". Agents exist within managed devices and host management variables. Management variables may be inspected and altered by managers using SNMP commands. A manager is typically an application running on a workstation. There is no limit to the number of agents or managers on a network.

Almost all monitoring and control functions are accomplished through inspection and alteration of management variables instantiated in the SNMP agent. Management variables in an SNMP agent are organized in a tree structure called the Management Information Base (MIB). Management variables are located at the nodes and leafs of the MIB though many nodes simply serve as place holders and do not host a variable. Each branch from a node is given a number. Branches are also given names. Variables are uniquely addressed using an Object Identifier (OID). The OID is a data structure that enumerates the path from the root of the MIB to the variable. Almost any type of data can be represented in a MIB. Standard variable types include text strings and integers. Additional data types may be derived from primitive types and conventions have been established for organizing variables into multidimensional arrays.

A device's MIB defines the set of management variables hosted by an SNMP agent. MIBs are commonly documented in a language called Abstract Syntax Notation One (ASN.1). This format enumerates and describes each variable, indicates its data type, access rights and expected value range. Figure 1 shows a portion of Peak Audio's MIB for CobraNet devices. MIB compilers read ASN.1 to produce reports, and configuration settings for network management systems. MIB compilers are also used as a tool to ensure that a MIB is syntactically correct and to simplify implementation at both manager and agent sides.

To encourage interoperability, MIBs are not constructed arbitrarily. The process of defining a MIB begins with a standard MIB template called MIB-II. MIB-II defines the basic variables and structure of the MIB. The basic structure defined by MIB-II is shown in Figure 2. Supplementary MIB structures are defined for standard types of network devices such as workstations, printers, routers, hubs, etc. Wherever applicable, agents follow these basic templates to assure maximal interoperability. Device-specific extensions to a standard MIB structure are rooted in the "enterprises" section of the management hierarchy. Devices with little resemblance to standard network components may feature many variables in this area. Any organization may obtain enterprise branch assignment by contacting the Internet Assigned Numbers Authority (IANA).

SNMP Commands

For the most part SNMP operates in a question-answer mode. A typical transaction sees a manager sending a request packet to an agent. The agent performs the requested operation then returns a response packet to the manager. There is no connection between manager and agent between transactions. If either the request or response packet is lost on the network, it is the manager's responsibility to detect this and retransmit the request if desired. If the retransmission is the result of a lost response packet, the retransmission will have the agent fielding the same request twice. Because of this possibility, SNMP variables must be designed to be idempotent – multiple applications of a request have the same effect as one.

SNMP version 1 (SNMPv1) defines five packet types. **GetRequest** allows a manager to inspect variables on an agent. **SetRequest** allows a manger to alter variables on an agent. A manager can use **GetNextRequest** to discover through traversal all or part of an agent's MIB. An agent uses the **GetRespose** to respond to all manager requests. The agent may also transmit a **Trap** as the result of a local triggering event such as a power up.

SNMP Packet Structure

The structure of the SNMP request/response packet is shown in
Figure 3. This structure is used for all but the Trap packet type. The first two sections of the packet, the **IP** and **UDP** headers relate to the UDP/IP transport and are not actually part of the SNMP protocol. The SNMP header begins with the **Version** field indicating the SNMP protocol version. There are currently three deployed SNMP versions. Each new version is backward compatible with previous versions. The **Community** text field is a password than can be used to control access to devices. A default community, "public," is most commonly used. The **Protocol Data Unit (PDU) Type** distinguishes the five types of SNMP packets. Packet types are enumerated in Figure 4. The **Request ID** is used to pair requests and responses. The manager generates a unique integer for each request. An agent's response to the request will contain the same request ID value. The **Error Status** is used in response packets to indicate the overall result of a request as detailed in Figure 5. In the event of a non-zero error status, **Error Index** indicates the OID/value pair that caused the error. The remainder of the packet consists of OID/value pairs. Even when the value is unused, as it the case in the GetRequest packet, it is given a null value and included as a placeholder.

The use of the same packet structure for all requests and responses simplifies agent implementation. The agent can directly use the request packet as the basis for a response. A response to a SetRequest merely requires that the agent change the PDU Type field before sending the original request packet back to the manager.

The trap packet type follows the slightly different format shown in Figure 6. Trap contains an additional header information prior to the OID/value pairs. The **Enterprise** field contains the OID of the sub-tree assigned to the manufacturer of the device

transmitting the trap. The **Agent Address** is the IP address of the agent transmitting the trap. **Trap Types** are described in Figure 7. Six specific traps are defined, with a seventh one allowing vendors to implement an enterprise specific trap. The enterprise specific trap is indicated in the **Enterprise Specific Trap** field. The enterprise specific trap is zero for standard trap types.

Traps are sent asynchronously by agents to managers. Traps are useful for focusing the attention of a network manager. On a large network, using SNMP's request/response protocol to sequentially poll all network elements, it may take a considerable time for a manager to discover a malfunctioning piece of equipment. This continuous polling of all network components would also produce a considerable amount of traffic on the network. Through a judicious use of traps, a manager can be alerted to exceptional events throughout the network and use request/response transactions to obtain additional detail. The details of how traps are set up and used under SNMP are largely implementation specific.

BER Encoding

We have described SNMP packets as consisting of various fields without giving detail as to how these fields are constructed. SNMP packets are built according to the variable-length Basic Encoding Rules (BER) associated with ASN.1. This encoding scheme insures that all data is encoded uniformly when sent over the network regardless of how it is stored on the agent or manager. BER eliminates big-endian/little-endian, 16/32-bit integer and EBCDIC/ASCII clashes. Unfortunately, producing and decoding BER's variable length byte oriented packets on today's 16, 24, 32 and 64-bit processors is difficult to accomplish efficiently.

The basic structure of a BER encoded field is shown in Figure 8. **Type** identifies the data type of value the encoded. Examples of identifier types are integer, octet (binary) string, and display (text) string. **Length** indicates the size, in bytes of the Value field. The length field is itself variable length so that data values larger than 255 bytes may be encoded. The length field may consist of one octet for short form (up to 128 values), or a long form of up to 128 octets. **Value** contains zero or more bytes and conveys the value of the data. Figure 9 shows and example BER encoding for an integer.

The scope of BER encoding actually extends beyond the binary representation of the individual fields in a SNMP packet. Under BER the entire packet is considered a single data field which hosts BER encoded sub-fields. Many of those sub-fields may well host their own BER encoded sub-fields.

CobraNet SNMP Implementation

Management of a CobraNet device is accomplished through inspection and alteration of a set of management variables. The variables themselves are nothing more than scattered data storage within our embedded application. These management variables are simultaneously accessible to a local microprocessor through the HMI and to any SNMP

manager through the SNMP agent. Variables as seen through HMI are arranged in a sparsely populated 24-bit (16Mword) address space. Variables as seen through SNMP are arranged in a tree structure according to MIB guidelines. There exists a mapping structure associated with each of these interfaces to perform mappings to the shared variables. It was imperative to keep the memory overhead in these mappings to a minimum. This was accomplished on the HMI side by taking a block based approach. By arranging data in contiguous blocks both in physical memory and in the 24-bit HMI address space, a single mapping entry serves a handful of variables. Combined with a concise mapping structure, we were able to keep memory consumed by the mapping mechanism well below the size of the actual data it serves. We devised SNMP mapping a scheme for representing a tree structure which requires only one memory word per node. The SNMP mapping data is therefore approximately the same size as the data it supports.

We took special care to insure that control and monitoring functions did not interfere with the main task at hand – delivery of uninterrupted audio over the network. A preemptive task scheduler assures that audio functionality always takes precedence over the fielding of management requests. We envision scenarios, such as the presence of multiple active managers on a network, were management requests arrive at the agent faster than they can be processed. Under these circumstances we are forced to arbitrarily ignore request packets. Fortunately the unreliable UDP/IP transport has taught SNMP managers to expect this sort of behavior if not from the agents then from the network itself. A manager cannot distinguish between packets are dropped in transit and ones occasionally ignored once reaching an agent.

Parsing and generating the byte-oriented SNMP messages on a word-oriented processor is a less than painless undertaking. A class of byte manipulation functions aids in these chores. Yes, you can write object oriented assembly code. Some of the details of an SNMP implementation can be quite upending if not addressed in the design process. For instance, in the event of an error, none of a request is supposed to be processed. This requires you to validate all OID/value pairs in a SetRequest before applying the first.

Conclusions

We encourage anyone considering embarking on a communications protocol design to spend the time to survey what's out there. You'll be surprised at how many you find. You may even be convinced, as we eventually were, that world is not short of communications protocols. We found SNMP to suit our particular needs and that implementing SNMP and UDP/IP on platforms with limited resources is feasible. SNMP is particularly applicable to IP and Ethernet networks

SNMP provides the basic functionality required for control and monitoring of network connected audio devices. The variable based descriptive approach keeps the communications protocol itself from expanding but does not limit expansion of management functionality. With provisions for encapsulating multiple transactions in a single packet, performance under SNMP can be quite good. Traps can be used to selectively enhance

recognition of far-flung asynchronous events, something that is typically difficult to achieve through polling.

SNMP is well deployed and well supported. The IETF continues upkeep on SNMP with backward compatible revisions and remains sensitive to the need to keep the simple in SNMP. Interoperability is achieved through the ongoing process of MIB standardization.

Bibliography

Case, J., et al., "A Simple Network Management Protocol (SNMP)", RFC 1157, Network Working Group, Internet Engineering Task Force, May 1990, http://info.internet.isi.edu/in-notes/rfc/files/rfc1157.txt.

McCloghrie. K. and Rose, M.T., eds., "Management Information Base for Network Management of TCP/IP-based internets:MIB-II", RFC 1213, Network Working Group, Internet Engineering Task Force, March, 1991, http://info.internet.isi.edu/in-notes/rfc/files/rfc1213.txt.

Kastenholtz, F. ed., "SNMP Communications Services", RFC 1270, Network Working Group, Internet Engineering Task Force, October 1991, http://info.internet.isi.edu/in-notes/rfc/files/rfc1270.txt.

Harrington, D., et al., "An Architecture for Describing SNMP Management Frameworks", RFC 2271, Network Working Group, Internet Engineering Task Force, January 1998, http://info.internet.isi.edu/in-notes/rfc/files/rfc2272.txt.

Case, J., et al., "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", RFC 2272, Network Working Group, Internet Engineering Task Force, January 1998, http://info.internet.isi.edu/in-notes/rfc/files/rfc2272.txt.

"AES standard for sound system control – Application protocol for controlling and monitoring audio devices via digital data networks – Part 1: Principles, formats, and basic procedures", Audio Engineering Society, New York, NY, May 15, 1997.

Combs, J. "Report on Downward Scalability of IP for Small Microcontrollers", Audio Engineering Society SC-10-1-h, New York, NY, November 25, 1997.

"LonTalk Protocol Specification", Echelon Corporation, Palo Alto, CA, Version 3.0, 1994, http://www.lonworks.echelon.com/Core/protocol/Default.htm.

"Home Plug and Play Specification", CEBus Industry Council, Version 0.91, July 22, 1997, http://www.cebus.org/hpnp/.

Stevens, W. Richard. *TCP/IP Illustrated Volumes I,II,III*. Reading, MA: Addison-Wesley, 1994.

Perkins, David, Evan McGinnis. *Understanding SNMP MIBs*. Upper Saddle River, NJ: Prentice Hall, 1997.

Townsend, Robert L. *SNMP Application Developer's Guide*. New York, NY: Van Nostrand Reinhold, 1995.

Rogerson, Dale. *Inside COM*. Redmond, WA: Microsoft Press, 1997.

Rosenberry, Ward and Tegue, Jim, *Distributing Applications Across DCE and Windows NT*, Sebastopol, CA: O'Reilly & Associates, Inc., 1993.

Figures

```
PEAKAUDIO-MIB DEFINITIONS ::= BEGIN

     -- Title:  CobraNet MIB version 1.0
     -- Date:  20 October 1997
     -- By:   Tom Holtzen

     IMPORTS
         enterprises
             FROM RFC1155-SMI
         OBJECT-TYPE
             FROM RFC-1212
         DisplayString
             FROM RFC-1213;

     PeakAudio    OBJECT IDENTIFIER ::= { enterprises 2680 }

     cobraNet      OBJECT IDENTIFIER ::= { PeakAudio 1 }

     audio       OBJECT IDENTIFIER ::= { cobraNet 2 }

     -- audio ************************************************************

     audioTable OBJECT-TYPE
       SYNTAX SEQUENCE OF AudioEntry
       ACCESS not-accessible
       STATUS mandatory
       DESCRIPTION
           "Audio metering table."
       ::= { audio 1 }

     audioEntry OBJECT-TYPE
       SYNTAX AudioEntry
       ACCESS not-accessible
       STATUS mandatory
       DESCRIPTION
           "Row."
       INDEX  { audioIndex }
       ::= { audioTable 1 }

     AudioEntry ::=
       SEQUENCE {
         audioIndex
           INTEGER,
         audioMeterChannel
           INTEGER,
         audioPeakLevel
           INTEGER,
         audioCurrLevel
           INTEGER
       }
```

```
audioIndex OBJECT-TYPE
  SYNTAX INTEGER (0..32)
  ACCESS read-only
  STATUS mandatory
  DESCRIPTION
      "Table Index"
  ::= { audioEntry 1 }

audioMeterChannel OBJECT-TYPE
  SYNTAX INTEGER (0..'FF'h)
  ACCESS read-write
  STATUS mandatory
  DESCRIPTION
      "Channel assignment"
  ::= { audioEntry 2 }

audioCurrLevel OBJECT-TYPE
  SYNTAX INTEGER (0..'800000'h)
  ACCESS read-write
  STATUS mandatory
  DESCRIPTION
      "Audio current level"
  ::= { audioEntry 4 }

audioLoopTable OBJECT-TYPE
  SYNTAX SEQUENCE OF AudioLoopEntry
  ACCESS not-accessible
  STATUS mandatory
  DESCRIPTION
      "Audio loop back table."
  ::= { audio 2 }

audioLoopEntry OBJECT-TYPE
  SYNTAX AudioEntry
  ACCESS not-accessible
  STATUS mandatory
  DESCRIPTION
      "Row."
  INDEX  { audioLoopIndex }
  ::= { audioLoopTable 1 }

AudioLoopEntry ::=
  SEQUENCE {
    audioLoopIndex
      INTEGER,
    audioSource
      INTEGER,
    audioDest
      INTEGER
  }

audioLoopIndex OBJECT-TYPE
  SYNTAX INTEGER (0..32)
  ACCESS read-only
  STATUS mandatory
  DESCRIPTION
      "Table Index"
  ::= { audioLoopEntry 1 }

audioSource OBJECT-TYPE
  SYNTAX INTEGER (0..'FF'h)
  ACCESS read-write
  STATUS mandatory
  DESCRIPTION
      "Source assignment. (0 indicates silence source)
      Default value = 0"
  ::= { audioLoopEntry 2 }

audioDest OBJECT-TYPE
  SYNTAX INTEGER (0..'FF'h)
  ACCESS read-write
```

```
        STATUS mandatory
        DESCRIPTION
            "Destination assignment (0 indicates no destination).
            Default value = 0"
        ::= { audioLoopEntry 3 }
END
```

Figure 1: Portion of CobraNet MIB in ASN.1

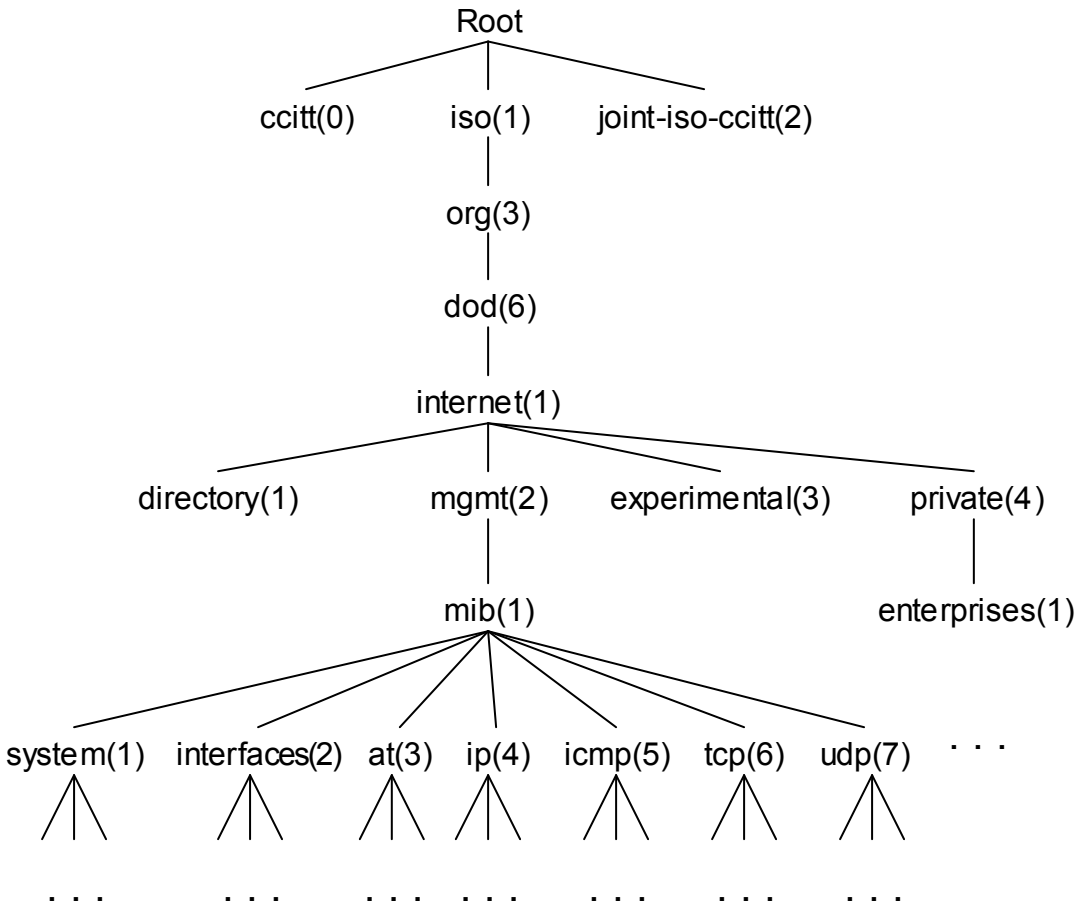


Figure 2: Base SNMP variable hierarchy (MIB-II)

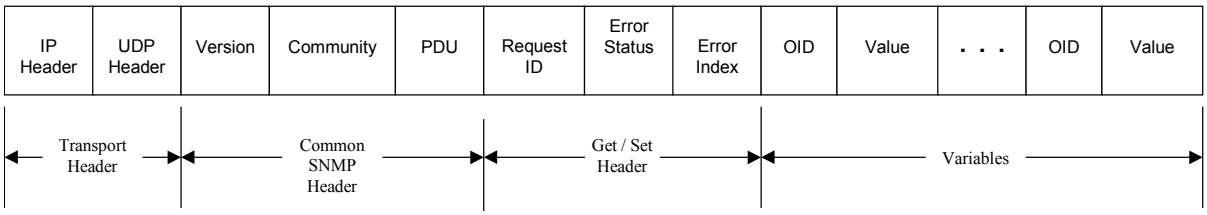


Figure 3: SNMP request/response packet

PDU Type	Name	Direction	Usage
0	GetRequest	Manager to agent	Requests enumerated variables be read.
1	GetNextRequest	Manager to agent	Returns OID and values for lexicographical successors to specified variables.
2	GetResponse	Agent to manger	Agent response to GetRequest, GetNextRequest and SetRequest. Contains current values of enumerated variables.
3	SetRequest	Manager to agent	Requests enumerated variables be written with supplied values.
4	Trap	Agent to manager	Sent asynchronously upon detection of a triggering event at the agent.

Figure 4: SNMP packet PDU Type field values and meanings

Error Status	Name	Description
0	NoError	All is ok
1	TooBig	Indicates that the response to the request would be too large to send back in a single packet. Agents are only required to support packets larger than 484 bytes. Ethernet may carry packets up to 1500 bytes. Support for these larger packets is encouraged.
2	NoSuchName	Indicates that a request referenced an OID which is not accessible.
3	BadValue	A set operation specified an invalid value or syntax.
4	ReadOnly	A set operation attempted to modify a read-only variable.
5	GenErr	A catch all error, which may be returned when no other recourse is available

Figure 5: Error Status values

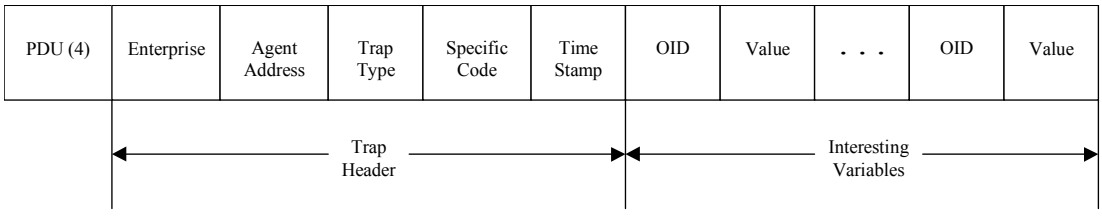


Figure 6: Trap packet format

Trap Type	Name	Description
0	ColdStart	Agent is initializing itself.
1	WarmStart	Agent is reinitializing itself.
2	LinkDown	An interface has changed from the up to the down state. The first variable in the message identifies the interface
3	LinkUp	An interface has changed from the down to the up state. The first variable in the message identifies the interface
4	AuthenticationFai-lure	A message was received from an SNMP manager with an invalid community.
5	EgpNeighborLoss	An EGP peer has changed to the down state. The first variable in the message contains the IP address of the peer.
6	EnterpriseSpecific	Look in the specific code field for the information on the trap.

Figure 7: Trap types

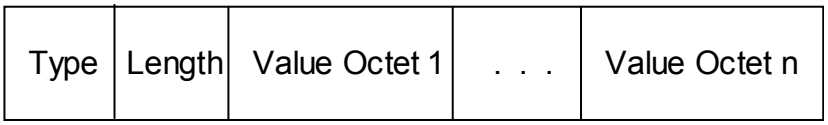
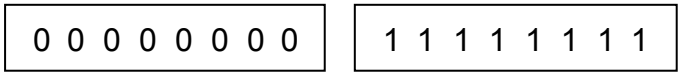


Figure 8: BER encoded value

Integer values are represented in two's-complement. Sign extending the value so that the value is expressed as a multiple of 8 bits forms the basis for the encoding for the integer value 255. Note that expressing 255 as a two's compliment value requires 9 bits, the 9th being the sign bit.



Add the type and length fields to arrive at the final 4 byte encoding.

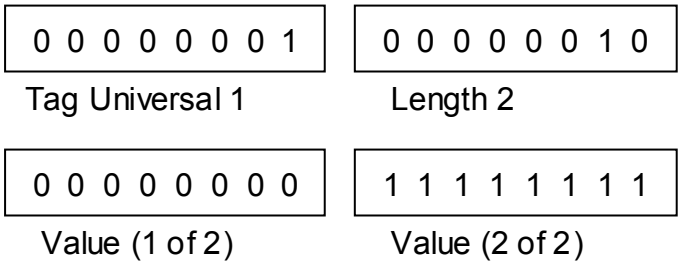


Figure 9: Integer BER example