

Network Working Group
Request for Comments: 2074
Category: Standards Track

A. Bierman
Cisco Systems
R. Iddon
AXON Networks, Inc.
January 1997

Remote Network Monitoring MIB Protocol Identifiers

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Table of Contents

1 Introduction	3
2 The SNMP Network Management Framework	3
2.1 Object Definitions	3
3 Overview	3
3.1 Terms	4
3.2 Relationship to the Remote Network Monitoring MIB	6
3.3 Relationship to the Other MIBs	6
4 Protocol Identifier Encoding	7
4.1 ProtocolDirTable INDEX Format Examples	9
4.2 Protocol Identifier Macro Format	10
4.2.1 Mapping of the Protocol Name	12
4.2.2 Mapping of the VARIANT-OF Clause	13
4.2.3 Mapping of the PARAMETERS Clause	13
4.2.3.1 Mapping of the 'countsFragments(0)' BIT	14
4.2.3.2 Mapping of the 'tracksSessions(1)' BIT	15
4.2.4 Mapping of the ATTRIBUTES Clause	15
4.2.5 Mapping of the DESCRIPTION Clause	15
4.2.6 Mapping of the CHILDREN Clause	16
4.2.7 Mapping of the ADDRESS-FORMAT Clause	16
4.2.8 Mapping of the DECODING Clause	16
4.2.9 Mapping of the REFERENCE Clause	17
4.2.10 Evaluating a Protocol-Identifier INDEX	17
5 Protocol Identifier Macros	18
5.1 Base Identifier Encoding	18
5.1.1 Protocol Identifier Functions	19
5.1.1.1 Function 0: No-op	19
5.1.1.2 Function 1: Protocol Wildcard Function	19
5.2 Base Layer Protocol Identifiers	20
5.2.1 Ether2 Encapsulation	21

5.2.2 LLC Encapsulation	22
5.2.3 SNAP over LLC (OUI=000) Encapsulation	23
5.2.4 SNAP over LLC (OUI != 000) Encapsulation	24
5.2.5 IANA Assigned Protocols	25
5.2.5.1 IANA Assigned Protocol Identifiers	27
5.3 L3: Children of Base Protocol Identifiers	27
5.3.1 IP	28
5.3.2 IPX	29
5.3.3 ARP	30
5.3.4 IDP	30
5.3.5 AppleTalk ARP	31
5.3.6 AppleTalk	31
5.4 L4: Children of L3 Protocols	32
5.4.1 ICMP	32
5.4.2 TCP	32
5.4.3 UDP	33
5.5 L5: Application Layer Protocols	33
5.5.1 FTP	33
5.5.1.1 FTP-DATA	33
5.5.1.2 FTP Control	34
5.5.2 Telnet	34
5.5.3 SMTP	34
5.5.4 DNS	35
5.5.5 BOOTP	35
5.5.5.1 Bootstrap Server Protocol	35
5.5.5.2 Bootstrap Client Protocol	35
5.5.6 TFTP	36
5.5.7 HTTP	36
5.5.8 POP3	36
5.5.9 SUNRPC	37
5.5.10 NFS	38
5.5.11 SNMP	38
5.5.11.1 SNMP Request/Response	38
5.5.11.2 SNMP Trap	39
6 Acknowledgements	39
7 References	40
8 Security Considerations	43
9 Authors' Addresses	43

1. Introduction

This memo defines an experimental portion of the Management Information Base (MIB) for use with network management protocols in the Internet community. In particular, it describes the algorithms required to identify different protocol encapsulations managed with the Remote Network Monitoring MIB Version 2 [RMON2]. Although related to the original Remote Network Monitoring MIB [RFC1757], this document refers only to objects found in the RMON-2 MIB.

2. The SNMP Network Management Framework

The SNMP Network Management Framework presently consists of three major components. They are:

- o the SMI, described in RFC 1902 [RFC1902], - the mechanisms used for describing and naming objects for the purpose of management.
- o the MIB-II, STD 17, RFC 1213 [RFC1213], - the core set of managed objects for the Internet suite of protocols.
- o the protocol, STD 15, RFC 1157 [RFC1157] and/or RFC 1905 [RFC1905], - the protocol for accessing managed information.

Textual conventions are defined in RFC 1903 [RFC1903], and conformance statements are defined in RFC 1904 [RFC1904].

The Framework permits new objects to be defined for the purpose of experimentation and evaluation.

2.1. Object Definitions

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. Objects in the MIB are defined using the subset of Abstract Syntax Notation One (ASN.1) defined in the SMI. In particular, each object type is named by an OBJECT IDENTIFIER, an administratively assigned name. The object type together with an object instance serves to uniquely identify a specific instantiation of the object. For human convenience, we often use a textual string, termed the descriptor, to refer to the object type.

3. Overview

The RMON-2 MIB [RMON2] uses hierarchically formatted OCTET STRINGS to globally identify individual protocol encapsulations in the protocolDirTable.

This guide contains algorithms and examples of protocol identifier encapsulations for use as INDEX values in the protocolDirTable.

This document is not intended to be an authoritative reference on the protocols described herein. Refer to the Official Internet Standards document [RFC1800], the Assigned Numbers document [RFC1700], or other appropriate RFCs, IEEE documents, etc. for complete and authoritative protocol information.

3.1. Terms

Several terms are used throughout this document, as well as in the RMON-2 MIB [RMON2], that should be introduced:

layer-identifier:

An octet string fragment representing a particular protocol encapsulation layer. A string fragment identifying a particular protocol encapsulation layer. This string is exactly four octets, (except for the 'vsnap' base-layer identifier, which is exactly eight octets) encoded in network byte order. A particular protocol encapsulation can be identified by starting with a base layer encapsulation (see the 'Base Protocol Identifiers' section for more detail), and following the encoding rules specified in the CHILDREN clause and assignment section for that layer. Then repeat for each identified layer in the encapsulation. (See section 4.2.10 'Evaluating a Protocol-Identifier INDEX' for more detail.)

protocol:

A particular protocol layer, as specified by encoding rules in this document. Usually refers to a single layer in a given encapsulation. Note that this term is sometimes used in the RMON-2 MIB [RMON2] to name a fully-specified protocol-identifier string. In such a case, the protocol-identifier string is named for its upper-most layer. A named protocol may also refer to any encapsulation of that protocol.

protocol-identifier string:

An octet string representing a particular protocol encapsulation, as specified by encoding rules in this document. This string is identified in the RMON-2 MIB [RMON2] as the protocolDirID object. A protocol-identifier string is composed of one or more layer-identifiers.

protocol-identifier macro:

A group of formatted text describing a particular protocol layer, as used within the RMON-2 MIB [RMON2]. The macro serves several purposes:

- Name the protocol for use within the RMON-2 MIB [RMON2].
- Describe how the protocol is encoded into an octet string.
- Describe how child protocols are identified (if applicable), and encoded into an octet string.
- Describe which protocolDirParameters are allowed for the protocol.
- Describe how the associated protocolDirType object is encoded for the protocol.
- Provide reference(s) to authoritative documentation for the protocol.

protocol-variant-identifier macro:

A group of formatted text describing a particular protocol layer, as used within the RMON-2 MIB [RMON2]. This protocol is a variant of a well known encapsulation that may be present in the protocolDirTable. This macro is used to document the IANA assigned protocols, which are needed to identify protocols which cannot be practically identified by examination of 'appropriate network traffic' (e.g. the packets which carry them). All other protocols (which can be identified by examination of appropriate network traffic) should be documented using the protocol-identifier macro. A protocol-variant-identifier is documented using the protocol-variant version of the protocol-identifier macro.

protocol-parameter:

A single octet, corresponding to a specific layer-identifier in the protocol-identifier. This octet is a bit-mask indicating special functions or capabilities that this agent is providing for the corresponding protocol.

protocol-parameters string:

An octet string, which contains one protocol-parameter for each layer-identifier in the protocol-identifier. See the section 'Mapping of the PARAMETERS Clause' for more detail. This string is identified in the RMON-2 MIB [RMON2] as the protocolDirParameters object.

protocolDirTable INDEX:

A protocol-identifier and protocol-parameters octet string pair that have been converted to an INDEX value, according to the encoding rules in in section 7.7 of RFC 1902 [RFC1902].

pseudo-protocol:

A convention or algorithm used only within this document for the purpose of encoding protocol-identifier strings.

3.2. Relationship to the Remote Network Monitoring MIB

This document is intended to identify possible string values for the OCTET STRING objects protocolDirID and protocolDirParameters. Tables in the new Protocol Distribution, Host, and Matrix groups use a local INTEGER INDEX, in order to remain unaffected by changes in this document. Only the protocolDirTable uses the strings (protocolDirID and protocolDirParameters) described in this document.

This document is not intended to limit the protocols that may be identified for counting in the RMON-2 MIB. Many protocol encapsulations, not explicitly identified in this document, may be present in an actual implementation of the protocolDirTable. Also, implementations of the protocolDirTable may not include all the protocols identified in the example section below.

This document is intentionally separated from the MIB objects to allow frequent updates to this document without any republication of MIB objects. Protocol Identifier macros submitted from the RMON working group and community at large (to the RMONMIB WG mailing list at 'rmonmib@cisco.com') will be collected and added to this document.

Macros submissions will be collected in the IANA's MIB files under the directory "ftp://ftp.isi.edu/mib/rmonmib/rmon2_pi_macros/" and in the RMONMIB working group mailing list message archive file "ftp://ftp.cisco.com/ftp/rmonmib/rmonmib".

This document does not discuss auto-discovery and auto-population of the protocolDirTable. This functionality is not explicitly defined by the RMON standard. An agent should populate the directory with 'interesting' protocols--depending on the intended applications.

3.3. Relationship to the Other MIBs

The RMON Protocol Identifiers document is intended for use with the protocolDirTable within the RMON MIB. It is not relevant to any other MIB, or intended for use with any other MIB.

4. Protocol Identifier Encoding

The protocolDirTable is indexed by two OCTET STRINGS, protocolDirID and protocolDirParameters. To encode the table index, each variable-length string is converted to an OBJECT IDENTIFIER fragment, according to the encoding rules in section 7.7 of RFC 1902 [RFC1902]. Then the index fragments are simply concatenated. (Refer to figures 1a - 1d below for more detail.)

The first OCTET STRING (protocolDirID) is composed of one or more 4-octet "layer-identifiers". The entire string uniquely identifies a particular protocol encapsulation tree. The second OCTET STRING, (protocolDirParameters) which contains a corresponding number of 1-octet protocol-specific parameters, one for each 4-octet layer-identifier in the first string.

A protocol layer is normally identified by a single 32-bit value. Each layer-identifier is encoded in the ProtocolDirID OCTET STRING INDEX as four sub-components [a.b.c.d], where 'a' - 'd' represent each byte of the 32-bit value in network byte order. If a particular protocol layer cannot be encoded into 32 bits, (except for the 'vsnap' base layer) then it must be defined as a 'ianaAssigned' protocol (see below for details on IANA assigned protocols).

The following figures show the differences between the OBJECT IDENTIFIER and OCTET STRING encoding of the protocol identifier string.

Fig. 1a
protocolDirTable INDEX Format

c !	c !
n ! protocolDirID	n ! protocolDir
t !	t ! Parameters

Fig. 1b
protocolDirTable OCTET STRING Format

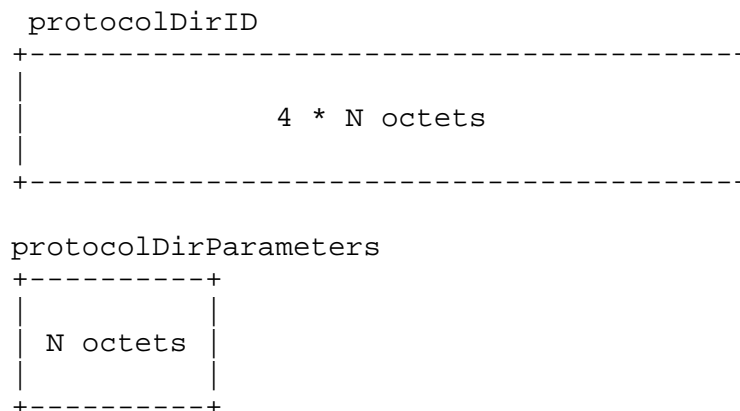


Fig. 1c
protocolDirTable INDEX Format Example

protocolDirID					protocolDirParameters					
c	proto	proto	proto	proto	c	par	par	par	par	
n	base	L3	L4	L5	n	ba-	L3	L4	L5	
t	(+flags)				t	se				
1	4 or 8	4	4	4	1	1/2	1	1	1	subOID count

where N is the number of protocol-layer-identifiers required for the entire encapsulation of the named protocol. Note that the 'vsnap' base layer identifier is encoded into 8 sub-identifiers, All other protocol layers are either encoded into 4 sub-identifiers or encoded as a 'ianaAssigned' protocol.

Fig. 1d
protocolDirTable OCTET STRING Format Example

protocolDirID

+	-----+	+	-----+	+	-----+	+	-----+	
	proto		proto		proto		proto	
	base		L3		L4		L5	
+	-----+	+	-----+	+	-----+	+	-----+	
	4 or 8		4		4		4	
								octet
								count

protocolDirParameters

+	-----+	+	-----+	+	-----+	+	-----+	
	par		par		par		par	
	ba-		L3		L4		L5	
	se							
+	-----+	+	-----+	+	-----+	+	-----+	
	1/2		1		1		1	
								octet
								count

where N is the number of protocol-layer-identifiers required for the entire encapsulation of the named protocol. Note that the 'vsnap' base layer identifier is encoded into 8 protocolDirID sub-identifiers and 2 protocolDirParameters sub-identifiers.

Although this example indicates four encapsulated protocols, in practice, any non-zero number of layer-identifiers may be present, theoretically limited only by OBJECT IDENTIFIER length restrictions, as specified in section 3.5 of RFC 1902 [RFC1902].

Note that these two strings would not be concatenated together if ever returned in a GetResponse PDU, since they are different MIB objects. However, protocolDirID and protocolDirParameters are not currently readable MIB objects.

4.1. ProtocolDirTable INDEX Format Examples

```
-- HTTP; fragments counted from IP and above
ether2.ip.tcp.www-http =
    16.0.0.0.1.0.0.8.0.0.0.0.6.0.0.0.80.4.0.1.0.0

-- SNMP over UDP/IP over SNAP
snap.ip.udp.snmp =
    16.0.0.0.3.0.0.8.0.0.0.0.17.0.0.0.161.4.0.0.0.0
```

```
-- SNMP over IPX over SNAP
snap.ipx.snmp =
    12.0.0.0.3.0.0.129.55.0.0.144.15.3.0.0.0

-- SNMP over IPX over raw8023
-- ianaAssigned(ipxOverRaw8023(1)).snmp =
    12.0.0.0.5.0.0.0.1.0.0.155.15.3.0.0.0

-- IPX over LLC
llc.ipx =
    8.0.0.0.2.0.224.224.3.2.0.0

-- SNMP over UDP/IP over any link layer
-- wildcard-ether2.ip.udp.snmp
    16.1.0.0.1.0.0.8.0.0.0.0.17.0.0.0.161.4.0.0.0.0

-- IP over any link layer; base encoding is IP over ether2
-- wildcard-ether2.ip
    8.1.0.0.1.0.0.8.0.2.0.0

-- AppleTalk Phase 2 over ether2
-- ether2.atalk
    8.0.0.0.1.0.0.128.155.2.0.0

-- AppleTalk Phase 2 over vsnap
-- vsnap(apple).atalk
    12.0.0.0.4.0.8.0.7.0.0.128.155.3.0.0.0
```

4.2. Protocol Identifier Macro Format

The following example is meant to introduce the protocol-identifier macro. (The syntax is not quite ASN.1.) This macro is used to represent both protocols and protocol-variants.

If the 'VariantOfPart' component of the macro is present, then the macro represents a protocol-variant instead of a protocol. A protocol-variant-identifier is used only for IANA assigned protocols, enumerated under the 'ianaAssigned' base-layer.

```

RMON-PROTOCOL-IDENTIFIER MACRO ::=
BEGIN
    PIMacroName "PROTOCOL-IDENTIFIER"
        VariantOfPart
            "PARAMETERS"    ParamPart
            "ATTRIBUTES"    AttrPart
            "DESCRIPTION"   Text
            ChildDescrPart
            AddrDescrPart
            DecodeDescrPart
            ReferPart
    "::=" "{ EncapsPart }"

    PIMacroName ::=
        identifier

    VariantOfPart ::=
        "VARIANT-OF" identifier | empty

    ParamPart ::=
        "{ ParamList }"

    ParamList ::=
        Params | empty

    Params ::=
        Param | Params "," Param

    Param ::=
        identifier "(" nonNegativeNumber ")"

    AttrPart ::=
        "{ AttrList }"

    AttrList ::=
        Attrs | empty

    Attrs ::=
        Attr | Attrs "," Attr

    Attr ::=
        identifier "(" nonNegativeNumber ")"

    ChildDescrPart ::=
        "CHILDREN" Text | empty

    AddrDescrPart ::=
        "ADDRESS-FORMAT" Text | empty

```

```

DecodeDescrPart ::=
    "DECODING" Text | empty

ReferPart ::=
    "REFERENCE" Text | empty

EncapsPart ::=
    "{" Encaps "}"

Encaps ::=
    Encap | Encaps "," Encap

Encap ::=
    BaseEncap | NormalEncap | VsnapEncap | IanaEncap

BaseEncap ::=
    nonNegativeNumber

NormalEncap ::=
    identifier nonNegativeNumber

VsnapEncap ::=
    identifier "(" nonNegativeNumber ")" nonNegativeNumber

IanaEncap ::=
    "ianaAssigned" nonNegativeNumber
    | "ianaAssigned" identifier
    | "ianaAssigned" identifier "(" nonNegativeNumber ")"

Text ::=
    "" string ""

END

```

4.2.1. Mapping of the Protocol Name

The 'PIMacroName' value should be a lower-case ASCII string, and contain the name or acronym identifying the protocol. NMS applications may treat protocol names as case-insensitive strings, and agent implementations must make sure the protocolDirTable does not contain any instances of the protocolDirDescr object which differ only in the case of one or more letters (if the identifiers are intended to represent different protocols).

It is possible that different encapsulations of the same protocol (which are represented by different entries in the protocolDirTable) will be assigned the same protocol name.

A protocol name should match the "most well-known" name or acronym for the indicated protocol. For example, the document indicated by the URL:

`ftp://ftp.isi.edu/in-notes/iana/assignments/protocol-numbers`

defines IP Protocol field values, so protocol-identifier macros for children of IP should be given names consistent with the protocol names found in this authoritative document.

4.2.2. Mapping of the VARIANT-OF Clause

This clause is present for IANA assigned protocols only. It identifies the protocol-identifier macro that most closely represents this particular protocol, and is known as the "reference protocol". (A protocol-identifier macro must exist for the reference protocol.) When this clause is present in a protocol-identifier macro, the macro is called a 'protocol-variant-identifier'.

Any clause (e.g. CHILDREN, ADDRESS-FORMAT) in the reference protocol-identifier macro should not be duplicated in the protocol-variant-identifier macro, if the 'variant' protocols' semantics are identical for a given clause.

Since the PARAMETERS and ATTRIBUTES clauses must be present in a protocol-identifier, an empty 'ParamPart' and 'AttrPart' (i.e. "PARAMETERS {}") must be present in a protocol-variant-identifier macro, and the 'ParamPart' and 'AttrPart' found in the reference protocol-identifier macro examined instead.

Note that if a 'ianaAssigned' protocol is defined that is not a variant of any other documented protocol, then the protocol-identifier macro should be used instead of the protocol-variant-identifier version of the macro.

4.2.3. Mapping of the PARAMETERS Clause

The protocolDirParameters object provides an NMS the ability to turn on and off expensive probe resources. An agent may support a given parameter all the time, not at all, or subject to current resource load.

The PARAMETERS clause is a list of bit definitions which can be directly encoded into the associated ProtocolDirParameters octet in network byte order. Zero or more bit definitions may be present. Only bits 0-7 are valid encoding values. This clause defines the entire BIT set allowed for a given protocol. A conforming agent may choose to implement a subset of zero or more of these PARAMETERS.

By convention, the following common bit definitions are used by different protocols. These bit positions must not be used for other parameters. They should be reserved if not used by a given protocol. Bits are encoded in network-byte order.

Table 3.1 Reserved PARAMETERS Bits

Bit Name	Description

0	countsFragments higher-layer protocols encapsulated within this protocol will be counted correctly even if this protocol fragments the upper layers into multiple packets.
1	tracksSessions correctly attributes all packets of a protocol which starts sessions on well known ports or sockets and then transfers them to dynamically assigned ports or sockets thereafter (e.g. TFTP).

The PARAMETERS clause must be present in all protocol-identifier macro declarations, but may be equal to zero (empty). Note that an NMS must determine if a given PARAMETER bit is supported by attempting to create the desired protocolDirEntry. The associated ATTRIBUTE bits for 'countsFragments' and 'tracksSessions' do not exist.

4.2.3.1. Mapping of the 'countsFragments(0)' BIT

This bit indicates whether the probe is correctly attributing all fragmented packets of the specified protocol, even if individual frames carrying this protocol cannot be identified as such. Note that the probe is not required to actually present any re-assembled datagrams (for address-analysis, filtering, or any other purpose) to the NMS.

This bit may only be set in a protocolDirParameters octet which corresponds to a protocol that supports fragmentation and reassembly in some form. Note that TCP packets are not considered 'fragmented-streams' and so TCP is not eligible.

This bit may be set in at most one protocolDirParameters octet within a protocolDirTable INDEX.

4.2.3.2. Mapping of the 'tracksSessions(1)' BIT

The 'tracksSessions(1)' bit indicates whether frames which are part of remapped-sessions (e.g. TFTP download sessions) are correctly counted by the probe. For such a protocol, the probe must usually analyze all packets received on the indicated interface, and maintain some state information, (e.g. the remapped UDP port number for TFTP).

The semantics of the 'tracksSessions' parameter are independent of the other protocolDirParameters definitions, so this parameter may be combined with any other legal parameter configurations.

4.2.4. Mapping of the ATTRIBUTES Clause

The protocolDirType object provides an NMS with an indication of a probe's capabilities for decoding a given protocol, or the general attributes of the particular protocol.

The ATTRIBUTES clause is a list of bit definitions which are encoded into the associated instance of ProtocolDirType. The BIT definitions are specified in the SYNTAX clause of the protocolDirType MIB object.

Table 3.2 Reserved ATTRIBUTES Bits

Bit Name	Description

0 hasChildren	indicates that there may be children of this protocol defined in the protocolDirTable (by either the agent or the manager).
1 addressRecognitionCapable	indicates that this protocol can be used to generate host and matrix table entries.

The ATTRIBUTES clause must be present in all protocol-identifier macro declarations, but may be empty.

4.2.5. Mapping of the DESCRIPTION Clause

The DESCRIPTION clause provides a textual description of the protocol identified by this macro. Notice that it should not contain details about items covered by the CHILDREN, ADDRESS-FORMAT, DECODING and REFERENCE clauses.

The DESCRIPTION clause must be present in all protocol-identifier macro declarations.

4.2.6. Mapping of the CHILDREN Clause

The CHILDREN clause provides a description of child protocols for protocols which support them. It has three sub-sections:

- Details on the field(s)/value(s) used to select the child protocol, and how that selection process is performed
- Details on how the value(s) are encoded in the protocol identifier octet string
- Details on how child protocols are named with respect to their parent protocol label(s)

The CHILDREN clause must be present in all protocol-identifier macro declarations in which the 'hasChildren(0)' BIT is set in the ATTRIBUTES clause.

4.2.7. Mapping of the ADDRESS-FORMAT Clause

The ADDRESS-FORMAT clause provides a description of the OCTET-STRING format(s) used when encoding addresses.

This clause must be present in all protocol-identifier macro declarations in which the 'addressRecognitionCapable(1)' BIT is set in the ATTRIBUTES clause.

4.2.8. Mapping of the DECODING Clause

The DECODING clause provides a description of the decoding procedure for the specified protocol. It contains useful decoding hints for the implementor, but should not over-replicate information in documents cited in the REFERENCE clause. It might contain a complete description of any decoding information required.

For 'extensible' protocols ('hasChildren(0)' BIT set) this includes offset and type information for the field(s) used for child selection as well as information on determining the start of the child protocol.

For 'addressRecognitionCapable' protocols this includes offset and type information for the field(s) used to generate addresses.

The DECODING clause is optional, and may be omitted if the REFERENCE clause contains pointers to decoding information for the specified protocol.

4.2.9. Mapping of the REFERENCE Clause

If a publicly available reference document exists for this protocol it should be listed here. Typically this will be a URL if possible; if not then it will be the name and address of the controlling body.

The CHILDREN, ADDRESS-FORMAT, and DECODING clauses should limit the amount of information which may currently be obtained from an 'authoritative' document, such as the Assigned Numbers document [RFC1700]. Any duplication or paraphrasing of information should be brief and consistent with the authoritative document.

The REFERENCE clause is optional, but should be implemented if an authoritative reference exists for the protocol (especially for standard protocols).

4.2.10. Evaluating a Protocol-Identifier INDEX

The following evaluation is done after protocolDirTable INDEX value has been converted into two OCTET STRINGS according to the INDEX encoding rules specified in the SMI [RFC1902].

Protocol-identifiers are evaluated left to right, starting with the protocolDirID, which length should be evenly divisible by four. The protocolDirParameters length should be exactly one quarter of the protocolDirID string length.

Protocol-identifier parsing starts with the base layer identifier, which must be present, and continues for one or more upper layer identifiers, until all OCTETs of the protocolDirID have been used. Layers may not be skipped, so identifiers such as 'SNMP over IP' or 'TCP over anylink' can not exist.

The base-layer-identifier also contains a 'special function identifier' which may apply to the rest of the protocol identifier.

Wild-carding at the base layer within a protocol encapsulation is the only supported special function at this time. Refer to the 'Base Protocol Identifiers' section for wildcard encoding rules.

After the protocol-tree identified in protocolDirID has been parsed, each parameter bit-mask (one octet for each 4-octet layer-identifier) is evaluated, and applied to the corresponding protocol layer.

A protocol-identifier label may map to more than one value. For instance, 'ip' maps to 5 distinct values, one for each supported encapsulation. (see the 'IP' section under 'L3 Protocol Identifiers'),

It is important to note that these macros are conceptually expanded at implementation time, not at run time.

If all the macros are expanded completely by substituting all possible values of each label for each child protocol, a list of all possible protocol-identifiers is produced. So 'ip' would result in 5 distinct protocol-identifiers. Likewise each child of 'ip' would map to at least 5 protocol-identifiers, one for each encapsulation (e.g. ip over ether2, ip over LLC, etc.).

5. Protocol Identifier Macros

The following PROTOCOL IDENTIFIER macros can be used to construct protocolDirID and protocolDirParameters strings.

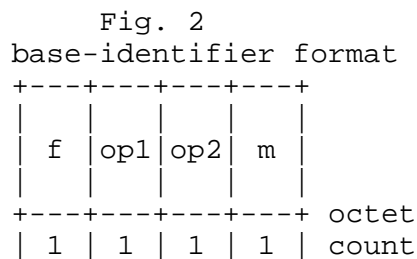
The sections defining protocol examples are intended to grow over subsequent releases. Minimal protocol support is included at this time. (Refer to section 3.2 for details on the protocol macro update procedure.)

An identifier is encoded by constructing the base-identifier, then adding one layer-identifier for each encapsulated protocol.

5.1. Base Identifier Encoding

The first layer encapsulation is called the base identifier and it contains optional protocol-function information and the base layer (e.g. MAC layer) enumeration value used in this protocol identifier.

The base identifier is encoded as four octets as shown in figure 2.



The first octet ('f') is the special function code, found in table 4.1. The next two octets ('op1' and 'op2') are operands for the indicated function. If not used, an operand must be set to zero. The last octet, 'm', is the enumerated value for a particular base layer encapsulation, found in table 4.2. All four octets are encoded in network-byte-order.

5.1.1.1. Protocol Identifier Functions

The base layer identifier contains information about any special functions to perform during collections of this protocol, as well as the base layer encapsulation identifier.

The first three octets of the identifier contain the function code and two optional operands. The fourth octet contains the particular base layer encapsulation used in this protocol (fig. 2).

Table 4.1 Assigned Protocol Identifier Functions

Function	ID	Param1	Param2
none	0	not used (0)	not used (0)
wildcard	1	not used (0)	not used (0)

5.1.1.1.1. Function 0: No-op

If the function ID field (1st octet) is equal to zero, the the 'op1' and 'op2' fields (2nd and 3rd octets) must also be equal to zero. This special value indicates that no functions are applied to the protocol identifier encoded in the remaining octets. The identifier represents a normal protocol encapsulation.

5.1.1.1.2. Function 1: Protocol Wildcard Function

The wildcard function (function-ID = 1), is used to aggregate counters, by using a single protocol value to indicate potentially many base layer encapsulations of a particular network layer protocol. A protocolDirEntry of this type will match any base-layer encapsulation of the same protocol.

The 'op1' field (2nd octet) is not used and must be set to zero.

The 'op2' field (3rd octet) is not used and must be set to zero.

Each wildcard protocol identifier must be defined in terms of a 'base encapsulation'. This should be as 'standard' as possible for interoperability purposes. If an encapsulation over 'ether2' is permitted, than this should be used as the base encapsulation.

The agent may also be requested to count some or all of the individual encapsulations for the same protocols, in addition to wildcard counting. Note that the RMON-2 MIB [RMON2] does not require that agents maintain counters for multiple encapsulations of the same protocol. It is an implementation-specific matter as to how an agent determines which protocol combinations to allow in the protocolDirTable at any given time.

5.2. Base Layer Protocol Identifiers

The base layer is mandatory, and defines the base encapsulation of the packet and any special functions for this identifier.

There are no suggested protocolDirParameters bits for the base layer.

The suggested ProtocolDirDescr field for the base layer is given by the corresponding "Name" field in the table 4.1 below. However, implementations are only required to use the appropriate integer identifier values.

For most base layer protocols, the protocolDirType field should contain bits set for the 'hasChildren(0)' and 'addressRecognitionCapable(1)' attributes. However, the special 'ianaAssigned' base layer should have no parameter or attribute bits set.

By design, only 255 different base layer encapsulations are supported. There are five base encapsulation values defined at this time. New base encapsulations (e.g. for new media types) are expected to be added over time.

Table 4.2 Base Layer Encoding Values

Name	ID
ether2	1
llc	2
snap	3
vsnap	4
ianaAssigned	5

5.2.1. Ether2 Encapsulation

ether2 PROTOCOL-IDENTIFIER

```

PARAMETERS { }
ATTRIBUTES {
    hasChildren(0),
    addressRecognitionCapable(1)
}

```

DESCRIPTION

"DIX Ethernet, also called Ethernet-II."

CHILDREN

"The Ethernet-II type field is used to select child protocols. This is a 16-bit field. Child protocols are deemed to start at the first octet after this type field."

Children of this protocol are encoded as [0.0.0.1], the protocol identifier for 'ether2' followed by [0.0.a.b] where 'a' and 'b' are the network byte order encodings of the MSB and LSB of the Ethernet-II type value.

For example, a protocolDirID-fragment value of:
0.0.0.1.0.0.8.0 defines IP encapsulated in ether2.

Children of are named as 'ether2' followed by the type field value in hexadecimal. The above example would be declared as:
ether2 0x0800"

ADDRESS-FORMAT

"Ethernet addresses are 6 octets in network order."

DECODING

"Only type values greater than or equal to 1500 decimal indicate Ethernet-II frames; lower values indicate 802.3 encapsulation (see below)."

REFERENCE

"A Standard for the Transmission of IP Datagrams over Ethernet Networks; RFC 894 [RFC894]."

The authoritative list of Ether Type values is identified by the URL:

```

ftp://ftp.isi.edu/in-notes/iana/assignments/ethernet-numbers"
::= { 1 }

```

5.2.2. LLC Encapsulation

llc PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES {

hasChildren(0),

addressRecognitionCapable(1)

}

DESCRIPTION

"The LLC (802.2) protocol."

CHILDREN

"The LLC SSAP and DSAP (Source/Dest Service Access Points) are used to select child protocols. Each of these is one octet long, although the least significant bit is a control bit and should be masked out in most situations. Typically SSAP and DSAP (once masked) are the same for a given protocol - each end implicitly knows whether it is the server or client in a client/server protocol. This is only a convention, however, and it is possible for them to be different. The SSAP is matched against child protocols first. If none is found then the DSAP is matched instead. The child protocol is deemed to start at the first octet after the LLC control field(s).

Children of 'llc' are encoded as [0.0.0.2], the protocol identifier component for LLC followed by [0.0.0.a] where 'a' is the SAP value which maps to the child protocol. For example, a protocolDirID-fragment value of:

0.0.0.2.0.0.0.240

defines NetBios over LLC.

Children are named as 'llc' followed by the SAP value in hexadecimal. So the above example would have been named:

llc 0xf0"

ADDRESS-FORMAT

"The address consists of 6 octets of MAC address in network order. Source routing bits should be stripped out of the address if present."

DECODING

"Notice that LLC has a variable length protocol header; there are always three octets (DSAP, SSAP, control). Depending on the value of the control bits in the DSAP, SSAP and control fields there may be an additional octet of control information.

LLC can be present on several different media. For 802.3 and 802.5 its presence is mandated (but see ether2 and raw802.3 encapsulations). For 802.5 there is no other link layer protocol.

Notice also that the raw802.3 link layer protocol may take precedence over this one in a protocol specific manner such that it may not be possible to utilize all LSAP values if raw802.3 is also present."

REFERENCE

"The authoritative list of LLC LSAP values is controlled by the IEEE Registration Authority:

IEEE Registration Authority
 c/o Iris Ringel
 IEEE Standards Dept
 445 Hoes Lane, P.O. Box 1331
 Piscataway, NJ 08855-1331
 Phone +1 908 562 3813
 Fax: +1 908 562 1571"

::= { 2 }

5.2.3. SNAP over LLC (OUI=000) Encapsulation

snap PROTOCOL-IDENTIFIER

```
PARAMETERS { }
ATTRIBUTES {
    hasChildren(0),
    addressRecognitionCapable(1)
}
```

DESCRIPTION

"The Sub-Network Access Protocol (SNAP) is layered on top of LLC protocol, allowing Ethernet-II protocols to be run over a media restricted to LLC."

CHILDREN

"Children of 'snap' are identified by Ethernet-II type values; the SNAP PID (Protocol Identifier) field is used to select the appropriate child. The entire SNAP protocol header is consumed; the child protocol is assumed to start at the next octet after the PID.

Children of 'snap' are encoded as [0.0.0.3], the protocol identifier for 'snap', followed by [0.0.a.b] where 'a' and 'b' are the MSB and LSB of the Ethernet-II type value. For example, a protocolDirID-fragment value of:

0.0.0.3.0.0.8.0

defines the IP/SNAP protocol.

Children of this protocol are named 'snap' followed by the Ethernet-II type value in hexadecimal. The above example would be named:

snap 0x0800"

ADDRESS-FORMAT

"The address format for SNAP is the same as that for LLC"

DECODING

"SNAP is only present over LLC. Both SSAP and DSAP will be 0xAA and a single control octet will be present. There are then three octets of OUI and two octets of PID. For this encapsulation the OUI must be 0x000000 (see 'vsnap' below for non-zero OUIs)."

REFERENCE

"SNAP Identifier values are assigned by the IEEE Standards Office. The address is:

IEEE Registration Authority
c/o Iris Ringel
IEEE Standards Dept
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331
Phone +1 908 562 3813
Fax: +1 908 562 1571"

::= { 3 }

5.2.4. SNAP over LLC (OUI != 000) Encapsulation

vsnap PROTOCOL-IDENTIFIER

```
PARAMETERS { }
ATTRIBUTES {
    hasChildren(0),
    addressRecognitionCapable(1)
}
```

DESCRIPTION

"This pseudo-protocol handles all SNAP packets which do not have a zero OUI. See 'snap' above for details of those that do."

CHILDREN

"Children of 'vsnap' are selected by the 3 octet OUI; the PID is not parsed; child protocols are deemed to start with the first octet of the SNAP PID field, and continue to the end of the packet.

Children of 'vsnap' are encoded as [0.0.0.4], the protocol identifier for 'vsnap', followed by [0.a.b.c.0.0.d.e] where 'a', 'b' and 'c' are the 3 octets of the OUI field in network byte order. This is in turn followed by the 16-bit EtherType value, where the 'd' and 'e' represent the MSB and LSB of the EtherType, respectively.

For example, a protocolDirID-fragment value of:

0.0.0.4.0.8.0.7.0.0.128.155

defines the AppleTalk Phase 2 protocol over vsnap.

Note that two protocolDirParameters octets must be present in protocolDirTable INDEX values for 'vsnap' protocols. The first protocolDirParameters octet defines the actual parameters. The second protocolDirParameters octet is not used and must be set to zero.

Children are named as 'vsnap(<OUI>) <ethertype>', where the '<OUI>' field is represented as 3 octets in hexadecimal notation or the ASCII string associated with the OUI value. The '<ethertype>' field is represented by the 2 byte EtherType value in hexadecimal notation. So the above example would be named:

'vsnap(0x080007) 0x809b' or 'vsnap(apple) 0x809b'

ADDRESS-FORMAT

"The LLC address format is inherited by 'vsnap'. See the 'llc' protocol identifier for more details."

DECODING

"Same as for 'snap' except the OUI is non-zero."

REFERENCE

"SNAP Identifier values are assigned by the IEEE Standards Office. The address is:

IEEE Registration Authority
c/o Iris Ringel
IEEE Standards Dept
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331
Phone +1 908 562 3813
Fax: +1 908 562 1571"

::= { 4 }

5.2.5. IANA Assigned Protocols

ianaAssigned PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES { }

DESCRIPTION

"This branch contains protocols which do not conform easily to the hierarchical format utilized in the other link layer branches. Usually, such a protocol 'almost' conforms to a particular 'well-known' identifier format, but additional criteria are used (e.g. configuration-based), making protocol identification difficult or impossible by examination of appropriate network traffic. preventing the any 'well-known' protocol-identifier macro from being used.

Sometimes well-known protocols are simply remapped to a different port number by one or more vendors (e.g. SNMP). These protocols can be identified with the 'user-extensibility' feature of the protocolDirTable, and do not need special IANA assignments.

A centrally located list of these enumerated protocols must be maintained to insure interoperability.
(See section 3.2 for details on the document update procedure.)
Support for new link-layers will be added explicitly, and only protocols which cannot possibly be represented in a better way will be considered as 'ianaEnumerated' protocols.

IANA assigned protocols are identified by the base-layer-selector value [0.0.0.5], followed by the four octets [a.b.c.d] of the integer value corresponding to the particular IANA protocol.

Do not create children of this protocol unless you are sure that they cannot be handled by the more conventional link layers above."

CHILDREN

"Children of this protocol are identified by implementation-specific means, described (as best as possible) in the 'DECODING' clause within the protocol-variant-identifier macro for each enumerated protocol.

For example, a protocolDirID-fragment value of:
0.0.0.5.0.0.0.1

defines the IPX protocol encapsulated directly in 802.3

Children are named 'ianaAssigned' followed by the name or numeric of the particular IANA assigned protocol. The above example would be named:

'ianaAssigned 1' or 'ianaAssigned ipxOverRaw8023'"

DECODING

"The 'ianaAssigned' base layer is a pseudo-protocol and is not decoded."

REFERENCE

"Refer to individual PROTOCOL-IDENTIFIER macros for information on each child of the IANA assigned protocol."

::= { 5 }

5.2.5.1. IANA Assigned Protocol Identifiers

The following protocol-variant-identifier macro declarations are used to identify the RMONMIB IANA assigned protocols in a proprietary way, by simple enumeration. Note that an additional four-octet layer identifier may be used for some enumerations (as with the 'vsnap' base-layer identifier). Refer to the 'CHILDREN' clause in the protocol-identifier macro for a particular protocol to determine the number of octets in the 'ianaAssigned' layer-identifier.

ipxOverRaw8023 PROTOCOL-IDENTIFIER

VARIANT-OF "ipx"

PARAMETERS { }

ATTRIBUTES { }

DESCRIPTION

"This pseudo-protocol describes an encapsulation of IPX over 802.3, without a type field.

Refer to the macro for IPX for additional information about this protocol."

DECODING

"Whenever the 802.3 header indicates LLC a set of protocol specific tests needs to be applied to determine whether this is a 'raw8023' packet or a true 802.2 packet. The nature of these tests depends on the active child protocols for 'raw8023' and is beyond the scope of this document."

::= { ianaAssigned 1 }

5.3. L3: Children of Base Protocol Identifiers

Network layer protocol identifier macros contain additional information about the network layer, and is found immediately following a base layer-identifier in a protocol identifier.

The ProtocolDirParameters supported at the network layer are 'countsFragments(0)', and 'tracksSessions(1)'. An agent may choose to implement a subset of these parameters.

The protocol-name should be used for the ProtocolDirDescr field. The ProtocolDirType ATTRIBUTES used at the network layer are 'hasChildren(0)' and 'addressRecognitionCapable(1)'. Agents may choose to implement a subset of these attributes for each protocol, and therefore limit which tables the indicated protocol can be present (e.g. protocol distribution, host, and matrix tables)..

The following protocol-identifier macro declarations are given for example purposes only. They are not intended to constitute an exhaustive list or an authoritative source for any of the protocol

information given. However, any protocol that can encapsulate other protocols must be documented here in order to encode the children identifiers into protocolDirID strings. Leaf protocols should be documented as well, but an implementation can identify a leaf protocol even if it isn't listed here (as long as the parent is documented).

5.3.1. IP

ip PROTOCOL-IDENTIFIER

```
PARAMETERS {
    countsFragments(0)  -- This parameter applies to all child
                        -- protocols.
}
```

```
ATTRIBUTES {
    hasChildren(0),
    addressRecognitionCapable(1)
}
```

DESCRIPTION

"The protocol identifiers for the Internet Protocol (IP). Note that IP may be encapsulated within itself, so more than one of the following identifiers may be present in a particular protocolDirID string."

CHILDREN

"Children of 'ip' are selected by the value in the Protocol field (one octet), as defined in the PROTOCOL NUMBERS table within the Assigned Numbers Document.

The value of the Protocol field is encoded in an octet string as [0.0.0.a], where 'a' is the protocol field .

Children of 'ip' are encoded as [0.0.0.a], and named as 'ip a' where 'a' is the protocol field value. For example, a protocolDirID-fragment value of:

```
0.0.0.1.0.0.8.0.0.0.0.1
```

defines an encapsulation of ICMP (ether2.ip.icmp)"

ADDRESS-FORMAT

"4 octets of the IP address, in network byte order. Each ip packet contains two addresses, the source address and the destination address."

DECODING

"Note: ether2/ip/ipmap4/udp is a different protocolDirID than ether2/ip/udp, as identified in the protocolDirTable. As such, two different local protocol index values will be assigned by the agent. E.g. (full INDEX values shown):

```
ether2/ip/ipmap4/udp 16.0.0.0.1.0.0.8.0.0.0.0.4.0.0.0.17.4.0.0.0.0
ether2/ip/udp        12.0.0.0.1.0.0.8.0.0.0.0.0.17.3.0.0.0 "
```

REFERENCE

"RFC 791 [RFC791] defines the Internet Protocol; The following URL defines the authoritative repository for the PROTOCOL NUMBERS Table:

```
ftp://ftp.isi.edu/in-notes/iana/assignments/protocol-numbers"
::= {
    ether2 0x0800,
    llc 0x06,
    snap 0x0800,
    ip 4,
    ip 94
}
```

5.3.2. IPX

ipx PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES {

hasChildren(0),

addressRecognitionCapable(1)

}

DESCRIPTION

"Novell IPX"

CHILDREN

"Children of IPX are defined by the 16 bit value of the Destination Socket field. The value is encoded into an octet string as [0.0.a.b], where 'a' and 'b' are the network byte order encodings of the MSB and LSB of the destination socket field."

ADDRESS-FORMAT

"4 bytes of Network number followed by the 6 bytes Host address each in network byte order".

REFERENCE

"The IPX protocol is defined by the Novell Corporation

A complete description of IPX may be secured at the following address:

Novell, Inc.
 122 East 1700 South
 P. O. Box 5900
 Provo, Utah 84601 USA
 800 526 5463
 Novell Part # 883-000780-001"

```
 ::= {
    ether2      0x8137,          -- 0.0.129.55
    llc         0xe0e003,        -- 0.224.224.3
    snap        0x8137,          -- 0.0.129.55
    ianaAssigned 0x1             -- 0.0.0.1   (ipxOverRaw8023)
  }
```

5.3.3. ARP

arp PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES { }

DESCRIPTION

"An Address Resolution Protocol message (request or response).
 This protocol does not include Reverse ARP (RARP) packets, which
 are counted separately."

REFERENCE

"RFC 826 [RFC826] defines the Address Resolution Protocol."

```
 ::= {
    ether2 0x806,    -- [ 0.0.8.6 ]
    snap 0x806
  }
```

5.3.4. IDP

idp PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES {

hasChildren(0),

addressRecognitionCapable(1)

}

DESCRIPTION

"Xerox IDP"

CHILDREN

"Children of IDP are defined by the 8 bit value of the Packet
 type field. The value is encoded into an octet string as [
 0.0.0.a], where 'a' is the value of the packet type field in
 network byte order."

ADDRESS-FORMAT

"4 bytes of Network number followed by the 6 bytes Host address each in network byte order".

REFERENCE

"Xerox Corporation, Document XNSS 028112, 1981"

```
::= {
    ether2  0x600,      -- [ 0.0.6.0 ]
    snap    0x600
}
```

5.3.5. AppleTalk ARP

atalkarp PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES { }

DESCRIPTION

"AppleTalk Address Resolution Protocol."

REFERENCE

"AppleTalk Phase 2 Protocol Specification, document ADPA #C0144LL/A."

```
::= {
    ether2 0x80f3,  -- [ 0.0.128.243 ]
    vsnap(0x080007) 0x80f3
}
```

5.3.6. AppleTalk

atalk PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES {

hasChildren(0),

addressRecognitionCapable(1)

}

DESCRIPTION

"AppleTalk Protocol."

CHILDREN

"Children of ATALK are defined by the 8 bit value of the DDP type field. The value is encoded into an octet string as [0.0.0.a], where 'a' is the value of the DDP type field in network byte order."

ADDRESS-FORMAT

"2 bytes of Network number followed by 1 byte of node id each in network byte order".

REFERENCE

"AppleTalk Phase 2 Protocol Specification, document ADPA
#C0144LL/A."

```
 ::= {
    ether2 0x809b,    -- [ 0.0.128.155 ]
    vsnap(0x080007) 0x809b
 }
```

5.4. L4: Children of L3 Protocols

5.4.1. ICMP

icmp PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES { }

DESCRIPTION

"Internet Message Control Protocol."

REFERENCE

"RFC 792 [RFC792] defines the Internet Control Message Protocol."

```
 ::= { ip 1 }
```

5.4.2. TCP

tcp PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES {

hasChildren(0)

}

DESCRIPTION

"Transmission Control Protocol."

CHILDREN

"Children of TCP are identified by the 16 bit Destination Port value as specified in RFC 793. They are encoded as [0.0.a.b], where 'a' is the MSB and 'b' is the LSB of the Destination Port value. Both bytes are encoded in network byte order. For example, a protocolDirId-fragment of:

```
0.0.0.1.0.0.8.0.0.0.0.6.0.0.0.23
```

identifies an encapsulation of the telnet protocol
(ether2.ip.tcp.telnet)"

REFERENCE

"RFC 793 [RFC793] defines the Transmission Control Protocol.

The following URL defines the authoritative repository for reserved and registered TCP port values:

```
ftp://ftp.isi.edu/in-notes/iana/assignments/port-numbers"
 ::= { ip 6 }
```


5.4.3. UDP

```

udp  PROTOCOL-IDENTIFIER
    PARAMETERS { }
    ATTRIBUTES {
        hasChildren(0)
    }
    DESCRIPTION
        "User Datagram Protocol."
    CHILDREN
        "Children of UDP are identified by the 16 bit Destination Port
        value as specified in RFC 768. They are encoded as [ 0.0.a.b ],
        where 'a' is the MSB and 'b' is the LSB of the Destination Port
        value. Both bytes are encoded in network byte order.  For
        example, a protocolDirId-fragment of:
            0.0.0.1.0.0.8.0.0.0.0.17.0.0.0.161

        identifies an encapsulation of SNMP (ether2.ip.udp.snmp)"
    REFERENCE
        "RFC 768 [RFC768] defines the User Datagram Protocol.

        The following URL defines the authoritative repository for
        reserved and registered UDP port values:

            ftp://ftp.isi.edu/in-notes/iana/assignments/port-numbers"
    ::= { ip 17 }

```

5.5. L5: Application Layer Protocols

5.5.1. FTP

5.5.1.1. FTP-DATA

```

ftp-data PROTOCOL-IDENTIFIER
    PARAMETERS { }
    ATTRIBUTES { }
    DESCRIPTION
        "The File Transfer Protocol Data Port; the FTP Server process
        default data-connection port. "
    REFERENCE
        "RFC 959 [RFC959] defines the File Transfer Protocol.  Refer to
        section 3.2 of [RFC959] for details on FTP data connections."
    ::= { tcp 20 }

```

5.5.1.2. FTP Control

ftp PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES { }

DESCRIPTION

"The File Transfer Protocol Control Port; An FTP client initiates an FTP control connection by sending FTP commands from user port (U) to this port."

REFERENCE

"RFC 959 [RFC959] defines the File Transfer Protocol."

::= { tcp 21 }

5.5.2. Telnet

telnet PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES { }

DESCRIPTION

"The Telnet Protocol; The purpose of the TELNET Protocol is to provide a fairly general, bi-directional, eight-bit byte oriented communications facility. Its primary goal is to allow a standard method of interfacing terminal devices and terminal-oriented processes to each other. "

REFERENCE

"RFC 854 [RFC854] defines the basic Telnet Protocol."

::= { tcp 23 }

5.5.3. SMTP

smtp PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES { }

DESCRIPTION

"The Simple Mail Transfer Protocol; SMTP control and data messages are sent on this port."

REFERENCE

"RFC 821 [RFC821] defines the basic Simple Mail Transfer Protocol."

::= { tcp 25 }

5.5.4. DNS

domain PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES { }

DESCRIPTION

"Domain Name Service Protocol; DNS may be transported by either UDP [RFC768] or TCP [RFC793]. If the transport is UDP, DNS requests restricted to 512 bytes in length may be sent to this port."

REFERENCE

"RFC 1035 [RFC1035] defines the Bootstrap Protocol."

::= { udp 53,
 tcp 53 }

5.5.5. BOOTP

5.5.5.1. Bootstrap Server Protocol

bootps PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES { }

DESCRIPTION

"Bootstrap Protocol Server Protocol; BOOTP Clients send requests (usually broadcast) to the bootps port."

REFERENCE

"RFC 951 [RFC951] defines the Bootstrap Protocol."

::= { udp 67 }

5.5.5.2. Bootstrap Client Protocol

bootpc PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES { }

DESCRIPTION

"Bootstrap Protocol Client Protocol; BOOTP Server replies are sent to the BOOTP Client using this destination port."

REFERENCE

"RFC 951 [RFC951] defines the Bootstrap Protocol."

::= { udp 68 }

5.5.6. TFTP

tftp PROTOCOL-IDENTIFIER

```
PARAMETERS {  
    tracksSessions(1)  
}
```

```
ATTRIBUTES { }
```

DESCRIPTION

"Trivial File Transfer Protocol; Only the first packet of each TFTP transaction will be sent to port 69. If the tracksSessions attribute is set, then packets for each TFTP transaction will be attributed to tftp, instead of the unregistered port numbers that will be encoded in subsequent packets."

REFERENCE

"RFC 1350 [RFC1350] defines the TFTP Protocol (revision 2); RFC 1782 [RFC1782] defines TFTP Option Extensions; RFC 1783 [RFC1783] defines the TFTP Blocksize Option; RFC 1784 [RFC1784] defines TFTP Timeout Interval and Transfer Size Options."

```
::= { udp 69 }
```

5.5.7. HTTP

www-http PROTOCOL-IDENTIFIER

```
PARAMETERS { }
```

```
ATTRIBUTES { }
```

DESCRIPTION

"Hypertext Transfer Protocol; "

REFERENCE

"RFC 1945 [RFC1945] defines the Hypertext Transfer Protocol (HTTP/1.0)."

```
::= { tcp 80 }
```

5.5.8. POP3

pop3 PROTOCOL-IDENTIFIER

```
PARAMETERS { }
```

```
ATTRIBUTES { }
```

DESCRIPTION

"Post Office Protocol -- Version 3. Clients establish connections with POP3 servers by using this destination port number."

REFERENCE

"RFC 1725 [RFC1725] defines Version 3 of the Post Office Protocol."

```
::= { tcp 110 }
```

5.5.9. SUNRPC

sunrpc PROTOCOL-IDENTIFIER

PARAMETERS { }

 ATTRIBUTES {
 hasChildren(0) -- port mapper function numbers
 }

DESCRIPTION

"SUN Remote Procedure Call Protocol. Port mapper function requests are sent to this destination port."

CHILDREN

Specific RPC functions are represented as children of the sunrpc protocol. Each 'RPC function protocol' is identified by its function number assignment. RPC function number assignments are defined by different naming authorities, depending of the function identifier value.

From [RFC1831]:

Program numbers are given out in groups of hexadecimal 20000000 (decimal 536870912) according to the following chart:

0	- 1fffffff	defined by rpc@sun.com
20000000	- 3fffffff	defined by user
40000000	- 5fffffff	transient
60000000	- 7fffffff	reserved
80000000	- 9fffffff	reserved
a0000000	- bfffffff	reserved
c0000000	- dfffffff	reserved
e0000000	- ffffffff	reserved

Children of 'sunrpc' are encoded as [0.0.0.111], the protocol identifier component for 'sunrpc', followed by [a.b.c.d], where a.b.c.d is the 32 bit binary RPC program number encoded in network byte order. For example, a protocolDirID-fragment value of:

0.0.0.111.0.1.134.163

defines the NFS function (and protocol).

Children are named as 'sunrpc' followed by the RPC function number in base 10 format. For example, NFS would be named:

'sunrpc 100003'.

REFERENCE

"RFC 1831 [RFC1831] defines the Remote Procedure Call Protocol Version 2. The authoritative list of RPC Functions is identified by the URL:

ftp://ftp.isi.edu/in-notes/iana/assignments/sun-rpc-numbers"
 ::= { udp 111 }

5.5.10. NFS

nfs PROTOCOL-IDENTIFIER

```
PARAMETERS {
    countsFragments(0)
}
```

ATTRIBUTES { }

DESCRIPTION

"Sun Network File System (NFS);"

DECODING

"The first packet in an NFS transaction is sent to the portmapper, and therefore decoded statically by monitoring RFC portmap requests [RFC1831]. Any subsequent NFS fragments must be decoded and correctly identified by 'remembering' the port assignments used in each RPC function call (as identified according to the procedures in the RPC Specification Version 2 [RFC1831])."

The 'countsFragments(0)' PARAMETER bit is used to indicate whether the probe can (and should) monitor portmapper activity to correctly attribute all NFS packets."

REFERENCE

"The NFS Version 3 Protocol Specification is defined in RFC 1813 [RFC1813]."

```
::= {
    sunrpc 100003          -- [0.1.134.163]
}
```

5.5.11. SNMP

5.5.11.1. SNMP Request/Response

snmp PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES { }

DESCRIPTION

"Simple Network Management Protocol. Includes SNMPv1 and SNMPv2 protocol versions. Does not include SNMP trap packets."

REFERENCE

"The SNMP SMI is defined in RFC 1902 [RFC1902]. The SNMP protocol is defined in RFC 1905 [RFC1905]. Transport mappings are defined in RFC 1906 [RFC1906]; RFC 1420 (SNMP over IPX) [RFC1420]; RFC 1419 (SNMP over AppleTalk) [RFC1419]."

```
::= {
    udp 161,
    ipx 0x900f,  -- [ 0.0.144.15 ]
    atalk 8
}
```

5.5.11.2. SNMP Trap

snmptrap PROTOCOL-IDENTIFIER

PARAMETERS { }

ATTRIBUTES { }

DESCRIPTION

"Simple Network Management Protocol Trap Port."

REFERENCE

"The SNMP SMI is defined in RFC 1902 [RFC1902]. The SNMP protocol is defined in RFC 1905 [RFC1905]. Transport mappings are defined in RFC 1906 [RFC1906]; RFC 1420 (SNMP over IPX) [RFC1420]; RFC 1419 (SNMP over AppleTalk) [RFC1419]."

```
::= {  
    udp 162,  
    ipx 0x9010,  
    atalk 9  
}
```

6. Acknowledgements

This document was produced by the IETF RMONMIB Working Group.

The authors wish to thank the following people for their contributions to this document:

Anil Singhal
Frontier Software Development, Inc.

Jeanne Haney
Bay Networks

Dan Hansen
Network General Corp.

7. References

[RFC768]

Postel, J., "User Datagram Protocol", STD 6, RFC 768, USC/Information Sciences Institute, August 1980.

[RFC791]

Postel, J., ed., "Internet Protocol - DARPA Internet Program Protocol Specification", STD 5, RFC 791, USC/Information Sciences Institute, September 1981.

[RFC792]

Postel, J., "Internet Control Message Protocol - DARPA Internet Program Protocol Specification", STD 5, RFC 792, USC/Information Sciences Institute, September 1981.

[RFC793]

Postel, J., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", STD 5, RFC 793, USC/Information Sciences Institute, September 1981.

[RFC821]

Postel, J., "Simple Mail Transfer Protocol", STD 10, RFC 821, USC/Information Sciences Institute, August 1982.

[RFC826]

Plummer, D., "An Ethernet Address Resolution Protocol or "Converting Network Protocol Addresses to 48-bit Ethernet Addresses for Transmission on Ethernet Hardware", STD 37, RFC 826, MIT-LCS, November 1982.

[RFC854]

Postel, J. and J. Reynolds, "Telnet Protocol Specification", STD 8, RFC 854, ISI, May 1983.

[RFC894]

Hornig, C., "A Standard for the Transmission of IP Datagrams over Ethernet Networks", RFC 894, Symbolics, April 1984.

[RFC951]

Croft, B., and J. Gilmore, "BOOTSTRAP Protocol (BOOTP)", RFC 951, Stanford and SUN Microsystems, September 1985.

[RFC959]

Postel, J., and J. Reynolds, "File Transfer Protocol", STD 8, RFC 959, USC/Information Sciences Institute, October 1985.

[RFC1035]

Mockapetris, P., "Domain Names - Implementation and Specification", STD 13, RFC 1035, USC/Information Sciences Institute, November 1987.

[RFC1157]

Case, J., M. Fedor, M. Schoffstall, J. Davin, "Simple Network Management Protocol", STD 15, RFC 1157, SNMP Research, Performance Systems International, MIT Laboratory for Computer Science, May 1990.

[RFC1213]

McCloghrie, K., and M. Rose, Editors, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", STD 17, RFC 1213, Hughes LAN Systems, Performance Systems International, March 1991.

[RFC1350]

Sollins, K., "TFTP Protocol (revision 2)", RFC 1350, MIT, July 1992.

[RFC1419]

Minshall, G., and M. Ritter, "SNMP over AppleTalk", RFC 1419, Novell, Inc., Apple Computer, Inc., March 1993.

[RFC1420]

Bostock, S., "SNMP over IPX", RFC 1420, Novell, Inc., March 1993.

[RFC1700]

Reynolds, J., and J. Postel, "Assigned Numbers", STD 2, RFC 1700, USC/Information Sciences Institute, October 1994.

[RFC1725]

Myers, J., and M. Rose, "Post Office Protocol - Version 3", RFC 1725, Carnegie Mellon, Dover Beach Consulting, November 1994.

[RFC1757]

S. Waldbusser, "Remote Network Monitoring MIB", RFC 1757, Carnegie Mellon University, February 1995.

[RFC1782]

Malkin, G., and A. Harkin, T "TFTP Option Extension", RFC 1782, Xylogics, Inc., Hewlett Packard Co., March 1995.

[RFC1783]

Malkin, G., and A. Harkin, T "TFTP BlockOption Option", RFC 1783, Xylogics, Inc., Hewlett Packard Co., March 1995.

[RFC1784]

Malkin, G., and A. Harkin, "TFTP Timeout Interval and Transfer Size Options", RFC 1784, Xylogics, Inc., Hewlett Packard Co., March 1995.

[RFC1800]

Postel, J., Editor, "Internet Official Protocol Standards", STD 1, RFC 1920, IAB, March 1996.

[RFC1831]

Srinivasan, R., "Remote Procedure Call Protocol Version 2", RFC 1831, Sun Microsystems, Inc., August 1995.

[RFC1902]

SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1902, January 1996.

[RFC1903]

SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Textual Conventions for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1903, January 1996.

[RFC1904]

SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Conformance Statements for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1904, January 1996.

[RFC1905]

SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, January 1996.

[RFC1906]

SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1906, January 1996.

[RFC1945]

Berners-Lee, T., and R. Fielding, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, MIT/UC-Irvine, November 1995.

[RMON2]

S. Waldbusser, "Remote Network Monitoring MIB (RMON-2)", draft-ietf-rmonmib-rmon2-03.txt, International Network Services, January 1996.

8. Security Considerations

Security issues are not discussed in this memo.

9. Authors' Addresses

Andy Bierman
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134

Phone: 408-527-3711
EMail: abierman@cisco.com

Robin Iddon
3Com/AXON
40/50 Blackfrias Street
Edinburgh, UK

Phone: +44 131.558.3888
EMail: robin_iddon@3mail.3com.com

