Implementation of IP-MIB Modules for IPv4 and IPv6 protocol

Mongi Abdelmoula, Neha Jha, Ashok Anand, Isabelle Astic

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Implementation of IP-MIB Modules for IPv4 and IPv6 Protocol

Mongi Abdelmoula — Neha Jha — Ashok Anand — Isabelle Astic

N° 0271

Octobre 2002

THÈME 1

ISSN 0249-0803

Implementation of IP-MIB Modules for IPv4 and IPv6 Protocol

Mongi Abdelmoula , Neha Jha , Ashok Anand , Isabelle Astic

Thème 1 — Réseaux et systèmes
Projet Resedas-Madynes

Rapport technique n° 0271 — Octobre 2002 — 199 pages

Abstract: While IPv6 is entering its operational phase, very few management architectures or tools exit to manage it. Even the SNMP (Simple Network Management Protocol) framework, which was mainly used for IPv4 networks, is unavailable for IPv6 ones, as no MIB II (Management Information Base) is fixed for these networks.

Firstly defined in 1998, the MIB II able to manage IPv6 networks had the inconvenient to separate IPv4 and IPv6 management. A second approach was defined in 2000, creating a "unified MIB". This second approach evoluated until 2002, where it seems to be fixed.

This document explains the first implementation of this "unified MIB". It describes the implementation of its new associated textual convention and of the four others drafts that defined it, into the net-snmp-5.0.3 module of the net-snmp package, on FreeBSD 4.5 system.

Key-words: SNMP, MIB, unified MIB, IPv6, draft-ietf-ipngwg-rfc2011-update00, draft-ietf-ipngwg-rfc2012-update01, draft-ietf-ipngwg-rfc2013-update01, draft-ietf-ipngwg-rfc2096-update00, RFC3291, net-snmp

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,

Implémentation de modules de MIB pour gérer les réseaux IPv4 et IPv6

Résumé : Alors que le protocole IPv6 entre dans sa phase opérationnelle, très peu d'architectures ou d'outils de supervision existent pour l'administrer. Même le cadre de travail défini par SNMP (Simple Network Management Protocol), qui est la principale architecture de supervision utilisée pour les réseaux IPv4, est indisponible pour les réseaux IPv6, car il n'existe pas encore de MIB II réellement figée, capable de modéliser ce type de réseaux.

Définie une première fois en 1998, une première évolution de la MIB II avait pour conséquence de séparer l'administration des réseaux IPv4 et IPv6. C'est pourquoi, une seconde approche a été définie en 2000, créant une MIB II "unifiée". Cette seconde approche a évolué jusqu'en 2002 où elle semble s'être stabilisée.

Ce document explique la première implémentation de cette MIB II "unifiée". Il décrit comment la nouvelle convention textuelle qui lui est associée et les quatre drafts qui la définissent ont été implémentés dans la version 5.0.3 de l'Open Source net-snmp, sous système FreeBSD 4.5.

Mots-clés : SNMP, MIB, IPv6, draft-ietf-ipngwg-rfc2011-update00, draft-ietf-ipngwg-rfc2012-update01, draft-ietf-ipngwg-rfc2013-update01, draft-ietf-ipngwg-rfc2096-update00, RFC3291, net-snmp

1 General Introduction

The new Internet Protocol IPv6, which will replace the current version IPv4, is being used in numerous academic networks and several commercial ones. But though the infrastructure and the routing protocols exist, there are very few management applications, and specially no SNMP (Simple Network Management Protocol) implementation that implements a MIB (Management Information Base) able to manage IPv6 networks.

The University of California at Davis (UCD)[1] developed a network management framework (multi Operating System) based on SNMP protocol[2]. Our work includes extensions of this distribution for use on IPv6.

This report presents our contribution which primarily involved implementing the MIB for IPv6 as, with the development of IPv6, new management information are necessary for the administration of networks supporting this protocol particularly relating to interface tables, statistical data and addresses.

1.1 Context

The administration of networks involves the deployment and co-ordination of hardware, software and human elements to ensure the correct operation of all the logical and physical components of the networks. This activity is central today, for usage, maintenance and exploitation of the networks and the services. Recognized as one of the critical components of the information systems of today and tomorrow, the network management must face many challenges such as taking into account new paradigms (mobile agents, active networks) and applications to new services and protocols.

The SNMP set of standards (among them [CFSD90], [RM91], [MR91], [McC96a], [McC96b], [McC96c], [Bak97]) provides a framework for the definition of management information and a protocol for the exchange of that information . The rapid growth of networks and the increasing diversity of the systems over the last decade presented the need for a comprehensive management of this whole infrastructure. SNMP prooves to be a good solution adopted by the IETF (Internet Engineering Task Force).

1.2 The Problem

With the emergence of the new version of the IP protocol (IPv6), it is necessary to have networks management applications to be able to instrument the entities supporting this protocol. The evolution of a management application towards IPv6 arises on two levels. The first is at the level of the communication between two entities of management. The second relates to information of management relating to the protocol.

There is currently some experimental networks. One of them is called the 6bone which is a virtual network built on top of the physical IPv4 network. The 6bone supports the

[1] www.ucdavis.edu
[2] www.net-snmp.org

routing of the IPv6 packages via «tunnel»(encapsulation of the IPv6 packages in IPv4 packages).Our research group RESEDAS/MADYNES at LORIA[3] is interested in the concepts and the software tools for telecommunications and distributed systems. Within this framework, it has an experimental platform connected to what was the G6bone (the French branch of the 6bone), and what is called now Renater 2[4].
Another IPv6 experimental network is the 6net network, mainly IPv6 native, deployed by the 6net project[5], funded by the European Committee. The RESEDAS/MADYNES team participates at the IPv6 network management Working Group of this project which work is to define the new management architecture and solutions to manage IPv6 networks.
It participates also at the VTHD++ project[6], a french government funded project, where the team has to proposed management solutions for very high rates networks.

Our work involves the extension of the MIB-II standard of new MIBs containing the new IPv6 variables. It requires the implementation of the new textual convention defined into the RFC (Request For Comments) 3291, as well as the following drafts:

- draft-ietf-ipngwg-rfc2011-update00.txt
- draft-ietf-ipngwg-rfc2012-update01.txt
- draft-ietf-ipngwg-rfc2013-update01.txt
- draft-ietf-ipngwg-rfc2096-update00.txt

1.3 Organisation of the report

Chapter 2 describes the SNMP protocol along with the general structure of the MIB.

Chapter 3 then describes the salient features of the IPv6 addressing architecture, and serves to explain the format of the addressing information extracted from the tables.

The fourth chapter describes the evolution of SNMP from IPv4 to IPv6. It explains the evolution of the MIB II from an IPv4-only point of view to the draft that we had implemented, and the new textual conventions that allowed the creation of this unified MIB II.

The fifth chapter describes brieftly the net-snmp package which was used for extending the agent and the command sets used. It explains how the agent can be extended to incorporate new mib modules, and how the relevant information is extracted.

The sixth section goes on to give a description of the code implemented, along with the different functions and the structures used in the code. It also describes the integration of those source code into the net-snmp package.

The Appendix gathers the exact version of the drafts implemented, then some extract of the code defined for the implementation, if it was not explain in detail into the core of the document, and the output of all the SNMP requests which validate this integration.

This document is the concatenation of the different reports of Mongi ABDELMOULA, Neha JHA, Ashok ANAND and Isabelle ASTIC.

[3] www.loria.fr
[4] www.renater.fr
[5] www.6net.org
[6] www.inria.fr/valorisation/actions-nationales/vthd.fr.htm

2 SNMP framework

2.1 Introduction

The SNMP protocol and the MIBs are the two key elements of the Internet Protocol Management Framework.

2.2 Simple Network Management Protocol: SNMP

It is defined into the STD 15 ([CFSD90]).

Only four operations are available in the protocol:

- *get*, which is used to retrieve specific management information and hence can be used to retrieve the value of the specific identifier.
- *get-next*, which is used to retrieve, via traversal, management information. It is used to retrieve the value of the next instance in the tree.
- *set*, which is used to manipulate management information; and
- *trap*, which is used to report extraordinary events.

It is using these operations that the manager makes a query to the agent and obtains the necessary management information.

2.3 Management Information Base: MIB

For each equipment are defined management pieces of information, like the inside temperature for a printer or the user that is currently working on a PC. This management information helps to know the equipment state and to monitor its use. This management pieces of information are called *object*. The sets of objects modelling an equipment are gathered into a *Management Information Base* (MIB). Several RFCs define those MIBs, finalized by IETF or enterprise themselves. To classify these MIBs, a MIB tree is defined (see Figure 1).
The MIB defining the information needed to manage the IP set of protocols, is called MIB II (cf [McC96a], [McC96b], [McC96c], [Bak97]), which is referred as *mib(1)* into the Figure 1. As the number of objects able to manage those protocols is important, they are split into groups like: system, interfaces, or ip. So, an agent need only to implement those groups that correspond to the functions it countains.

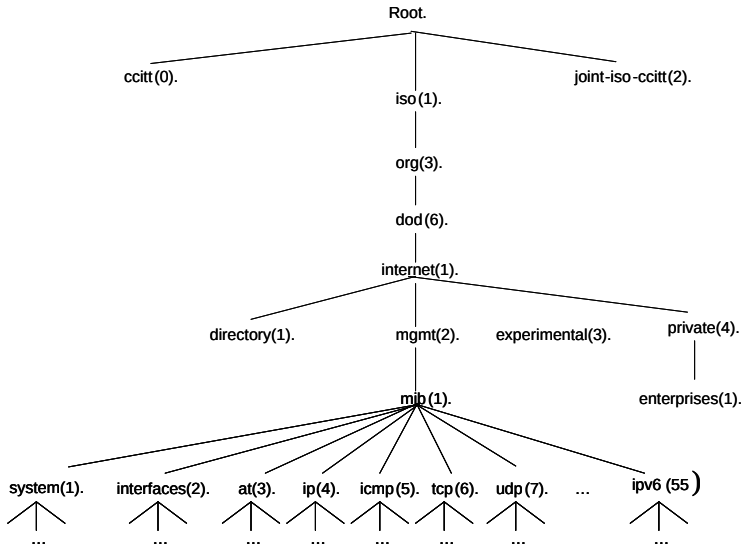

Figure 1: The MIB tree

The MIB which are not RFC are defined into the *enterprises* subtree.

Each object owns a unique *Object Identifier* (OID). This OID defines where the object is set into the MIB Tree. For example, an OID of .1.3.6.1.2.1.3.1 will represent the first object into the MIB II at group.

The model of each object is called *object type* and is defined in ASN.1 (Abstract Syntax Notation 1), as follow (cf [MPS99]):

```
object_type_name OBJECT-TYPE
SYNTAX syntax
ACCESS access-type
STATUS status
DESCRIPTION text of description
INDEX index(es)
AUGMENTS index(es)
DEFVAL default
::= { value }
```

where:

SYNTAX: refers to the ASN.1 type of the object. It could be a base type (like INTEGER, OCTET STRING or OBJECT IDENTIFIER), a BITS construct (enumeration of named bits) or textual conventions (like IpAddress, Counter32, Gauge32, TimeTicks, Opaque, Counter64, Unsigned32). It is a mandatory clause.

ACCESS: refers the way the object could be access. It could be from, least importance to the greatest one, not accessible, accessible for notify, read-write, read-only, read-create. It is a mandatory clause.

STATUS: mentions if the definition is current or historic. It could have the following value: current, obsolete, deprecated. It is a mandatory clause.

DESCRIPTION: is a text given a textual definition of the object. It is also a mandatory clause.

REFERENCE: not mandatory. It gives a cross-reference to some other document.

INDEX: mandatory only if the object is a row into a MIB table, otherwise, it must not be present. It indicates which objects index the table rows, in an unambiguous way.

AUGMENTS: alternative of the INDEX clause. Used only if there is a one-to-one correspondance between the conceptual rows of a table and an existing table.

DEFVAL: define a default value. Need to be present.

The last sentence gives its OID to the object type.

3 IPv6

3.1 Introduction

IP version 6 (IPv6) is the newest version of Internet Protocol that is the replacement for IP version 4 (IPv4). IPv4 is a fine potocol, but with the overwhelming expansion of the Internet, it is insufficient to keepup with the demand for addresses. IPv6 is the next generation protocol to alleviate the congestion that version 4 encounters. The resolution is achieved by having more addressable nodes.

3.2 Features of IPv6

Mainly defined into the RFC 2460 [DH98], IPv6 retains many of the design features that made IPv4 so successful. Like IPv4, IPv6 is connectionless - each datagram contains a destination address, and each datagram is routed independently. Like IPv4, the header in a datagram contains a maximum number of hops that the datagram can take before being discarded. However, despite retaining the basic concepts from the current version, IPv6 changes all the details.

- *Address space.* Instead of 32 bits, each IPv6 address contains 128 bits. The resulting address space allows it to :
  - Handle additional addressing levels.
  - Have more addressable nodes
  - Use easier auto-configuration of addresses
- *Multicast Routing.* IPv4 is best used for unicast addressing : a single address bit pattern corresponds to a single host. Some other types of addressing are not supported as well because of the restricted address size, and also because no provision is made for specific addressing modes. IPv6 incorporates the idea of an anycast address, in which a packet is delivered to just one of a set of nodes.

- *Header Format.* The IPv6 datagram header is completely different from the IPv4 header. Unlike IPv4, which uses a single header format for all datagrams, IPv6 encodes information into seperate headers. A datagram consists of the base IPv6 header followed by zero or more extension headers, followed by data.
- *Extensible Protocol.* Unlike IPv4, IPv6 does not specify all possible protocol features. Instead, the designers have provided a scheme that allows sender to add additional information to a datagram. This makes IPv6 more flexible and new features can be added to the design as needed.

3.3 IPv6 Addressing Architecture

IPv6 addressing architecture was defined in 1998, into the RFC 2373 [HD98].
IPv6 addresses are 128-bit identifiers for interfaces and sets of interfaces. There are three types of addresses:

- *Unicast*: An identifier for a single interface. A packet sent to a unicast address is delivered to the interface identified by that address.
- *Anycast*: An identifier for a set of interfaces (typically belonging to different nodes). A packet sent to an anycast address is delivered to one of the interfaces identified by that address (the "nearest" one, according to the routing protocols' measure of distance).
- *Multicast*: An identifier for a set of interfaces (typically belonging to different nodes). A packet sent to a multicast address is delivered to all interfaces identified by that address.

There are no broadcast addresses in IPv6, their function being superseded by multicast addresses.

3.3.1 Addressing Model

IPv6 addresses of all types are assigned to interfaces, not nodes. An IPv6 unicast address refers to a single interface. Since each interface belongs to a single node, any of that node's interfaces' unicast addresses may be used as an identifier for the node.

All interfaces are required to have at least one link-local unicast address. A single interface may also be assigned multiple IPv6 addresses of any type (unicast, anycast, and multicast) or scope (like local, site or organization). Currently IPv6 continues the IPv4 model that a subnet prefix is associated with one link. Multiple subnet prefixes may be assigned to the same link.

3.3.2 Text Representation of Address Prefixes

The preferred form is x:x:x:x:x:x:x:x, where the 'x's are the hexadecimal values of the eight 16-bit pieces of the address. Example: 1080:0:0:0:8:fe80:ba98:fedc.
In order to make writing addresses containing zero bits easier, a special syntax is available to compress the zeroes. The use of '::' indicates multiple groups of 16-bits of zeros. For example, the previous example of IPv6 address can be represented as 1080::8:fe80:ba98:fedc.
The text representation of IPv6 address prefixes is similar to the way IPv4 addresses prefixes are written in CIDR (Class-less Interdomain Routing) notation (see [FLYV93] for more explanations on CIDR). An IPv6 address prefix is represented by the notation:

```
ipv6-address/prefix-length
```

For example, the 60-bit prefix 12AB00000000CD3 (hexadecimal) may be represented as:

```
12AB::CD30:0:0:0:0\60  or,
12AB:0:0:CD30::\60
```

The leading zeroes within any 16-bit chunk of the address may be dropped, but this should only be made once.

3.3.3 Address Type Representation

The specific type of an IPv6 address is indicated by the leading bits in the address. For example:

- Aggregatable Global Unicast Addresses 001
- Link-Local Unicast Addresses 1111 1110 10
- Multicast Addresses 1111 1111

The loopback are assigned out of the 0000 0000 format prefix space. The format prefixes 001 through 111, except for Multicast Addresses (1111 1111), are all required to have 64-bit interface identifiers in EUI-64 format. Unicast addresses are distinguished from multicast addresses by the value of the high- order octet of the addresses: a value of FF (11111111) identifies an address as a multicast address; any other value identifies an address as a unicast address. Anycast addresses are taken from the unicast address space, and are not syntactically distinguishable from unicast addresses.

3.3.4 Unicast Addresses

IPv6 unicast addresses are aggregatable with contiguous bit-wise masks similar to IPv4 addresses under CIDR.

There are several forms of unicast address assignment in IPv6, including the global aggregatable global unicast address, the site-local address, the link-local address, and the IPv4-capable host address.

3.3.5 Interface Identifiers

```
|           n bits                |   128-n bits    |
-----------------------------------------------------
|        subnet prefix            | interface ID    |
```

Interface identifiers in IPv6 unicast addresses are used to identify interfaces on a link. They are required to be unique on that link.They may also be unique over a broader scope. In many cases an interface's identifier will be the same as that interface's link-layer address. The same interface identifier may be used on multiple interfaces on a single node. In a number of the format prefixes Interface IDs are required to be 64 bits long and to be constructed in IEEE EUI-64 format [HD98]. EUI-64 based Interface identifiers may have global scope when a global token is available (e.g., IEEE 48bit MAC) or may have local scope where a global token is not available (e.g., serial links, tunnel end-points, etc.). It is required that the "u" bit (universal/local bit in IEEE EUI-64 terminology) be inverted when forming the interface identifier from the EUI-64. The "u" bit is set to one (1) to indicate global scope, and it is set to zero (0) to indicate local scope.

This explains the value of ipv6InterfaceIdentifier e.g.
IP-MIB::ipv6InterfaceIdentifier.1 = STRING: **2**01:2ff:fee3:608a
IP-MIB::ipv6InterfaceIdentifier.2 = STRING: **2**01:2ff:fee3:605d.
The **2** indicates the global scope.

3.3.6 The Unspecified Address

The address 0:0:0:0:0:0:0:0 is called the unspecified address. It must never be assigned to any node as it indicates the absence of an address.

3.3.7 The Loopback Address

The unicast address 0:0:0:0:0:0:0:1 is called the loopback address and is used by a node to send an IPv6 packet to itself. It is never be assigned to any physical interface and may be thought of as being associated with a virtual interface (e.g., the loopback interface). This virtual interface has an interface index, but no physical address associated with it. The loopback address must not be used as the source address in IPv6 packets that are sent outside of a single node. An IPv6 packet with a destination address of loopback must never be sent outside of a single node and must never be forwarded by an IPv6 router.

3.3.8 Local-Use IPv6 Unicast Addresses

There are several types of local-use unicast addresses defined. The main currently used is Link Local address which is to be used on a single link. Link-Local addresses have the following format:

```
|   10     |
|  bits    |        54 bits            |          64 bits       |
----------------------------------------------------------------
|1111111010|           0               |       interface ID     |
```

that is, their prefix is always 0xFE80\64. Link-Local addresses are designed to be used for addressing on a single link for purposes such as address auto-configuration, neighbor discovery, or when no routers are present.

3.3.9 Anycast Addresses

An IPv6 anycast address is an address that is assigned to more than one interface (typically belonging to different nodes), with the property that a packet sent to an anycast address is routed to the "nearest" interface having that address, according to the routing protocols' measure of distance.

Anycast addresses are allocated from the unicast address space, using any of the defined unicast address formats. Thus, anycast addresses are syntactically indistinguishable from unicast addresses. When a unicast address is assigned to more than one interface, thus turning it into an anycast address, the nodes to which the address is assigned must be explicitly configured to know that it is an anycast address.

For any assigned anycast address, there is a longest address prefix P that identifies the topological region in which all interfaces belonging to that anycast address reside. Within the region identified by P, each member of the anycast set must be advertised as a separate entry in the routing

system (commonly referred to as a "host route"); outside the region identified by P, the anycast address may be aggregated into the routing advertisement for prefix P.

In, the worst case, the prefix P of an anycast set may be the null prefix, i.e., the members of the set may have no topological locality. In that case, the anycast address must be advertised as a separate routing entry throughout the entire internet.

3.3.10 Multicast Addresses

An IPv6 multicast address is an identifier for a group of nodes. A node may belong to any number of multicast groups. 11111111 (oxFF) at the start of the address identifies the address as being a multicast address (see Figure 2).

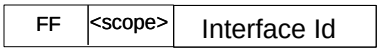

Figure 2: The structure of a multicast address

The scope part (1 byte) of the prefix is cut into 2x4 bits. The first four bits, defining the Flags field, are a set of 000T. The higher 3 flags are reserved and must be initialized to zero. T=0 indicates a permanently-assigned ("well-known") multicast address, assigned by the global internet numbering authority. T=1 indicates a non-permanently-assigned ("transcient") multicast address. The last 4 bits define the Scope field, and limit the scope of the multicast group. A multicast group could be link-local or site-local, etc...

3.3.11 A node's required address

A host is required to recognize the following addresses as identifying itself:

- its link local address for each interface,
- assigned unicast address,
- loopback address,
- all-nodes multicast address,
- solicited node multicast address for each of its assigned unicast and anycast addresses,
- multicast addresses of all other groups to which the host belongs.

4 Evolution of SNMP from IPv4 to IPv6

The SNMP standard contents the SNMP protocol and the MIBs. As the SNMP protocol was independent of the IP protocol architecture, its evolution towards IPv6 was not a real problem unlike the MIBs and specially the MIB II.
Since the specification of IPv6, in 1995, the definition of a MIB II able to manage IPv6 networks changed twice. The main problem was to define the IP address, that is its textual convention.

4.1 Evolution of the textual conventions

The evolution of the textual conventions is figured into the Figure 3.

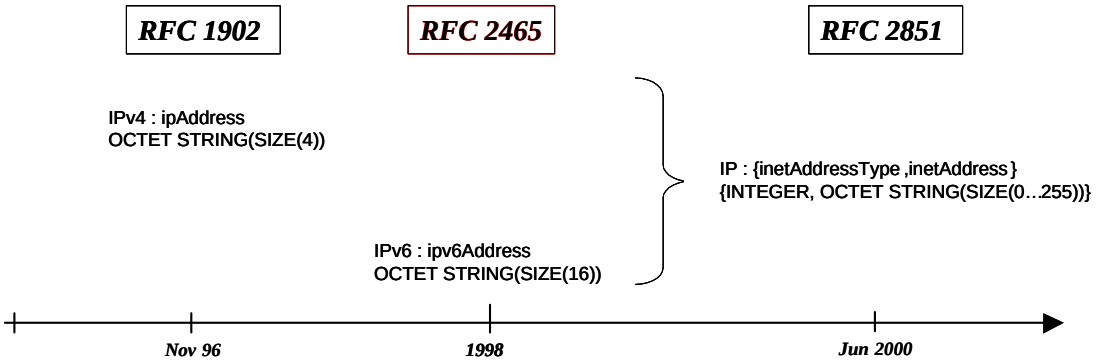

Figure 3: Evolution of the IP address textual convention to IPv6

The first textual convention defining IP address could be find into the RFC 1902 [CMRW96a]. It defined an ASN.1 type, called IpAddress, as an OCTET STRING(SIZE(4)). That means that it could be defined only with 4 bytes.
As IPv6 address is 128 bits long, 16 bytes are needed to save them. That is why the first textual convention defining IPv6 addresses was defined as an OCTET STRING(SIZE(16)), called Ipv6Address (RFC 2465 [HO98]). But this approach implies the partition of IPv4 and IPv6 management. So, IETF decided to define an unified MIB II, able to manage both IPv4 and IPv6 networks, which resulted into new textual conventions, defined in 2000, into the RFC 2851 ([DHRS00]). This RFC defines an IP address as a structure {inetAddressType, inetAddress}, where *inetAddressType* is an INTEGER which specifies if the following address is, for example, an IPv4 or IPv6 one. The *inetAddress* is defined as an OCTET STRING(SIZE(0...255)), in order to be able to save the value of an IPv4 or IPv6 address, as well as the value of a DNS name (cf [DHRS00] updated by the RFC 3291 [DHRS02]).

4.2 Evolution of the MIBs

The definition of this textual convention implies associated modifications of the MIB II.

4.2.1 The several RFC involved in this evolution

In 1996, the MIB II was completed in order to manage IPv4 networks. It was defined by the RFC 2096, RFC 2011, RFC 2012, RFC 2013. When the first textual convention for IPv6 addresses appeared in 1998, then other RFCs were defined to manage TCP and UDP over IPv6, ICMPv6 and IPv6 itself. They were RFC 2452, RFC 2454, RFC 2465, RFC 2466. (see the Figure 4).

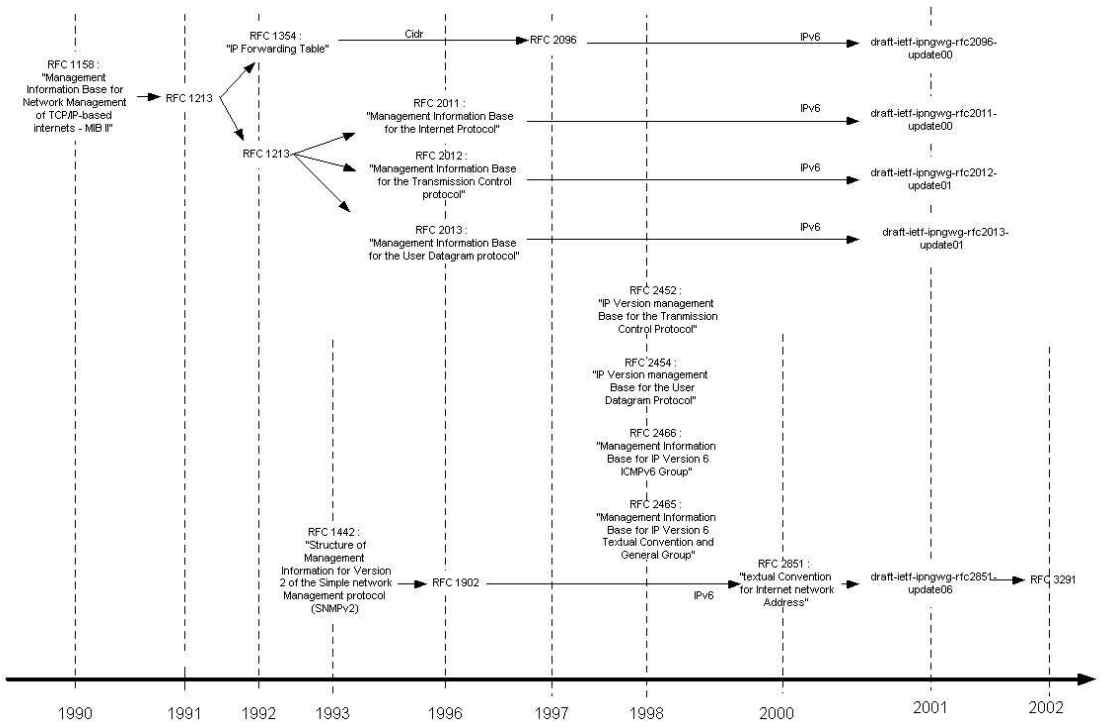

Figure 4: Evolution of the RFC to manage IPv6 networks

After the definition of the unified textual convention, the "old" RFCs were updated by the drafts that we implemented in their June 2002 version.

4.2.2 The MIBs modifications

If we take the MIB objects and tables point of view, the definition of these multiple MIBs had the following impact:

- In 1996, the RFC 2011, 2012 and 2013 defined 3 groups: ip, tcp and udp (see the Figure 5). Each group contains simple objects and tables.
- In 1998, another group was defined, called ipv6 containing simple objects and tables, sometimes saving the same information than for IPv4 networks but into an IPv6 context. For examples, the ipv6IfTable contains must of the simple objects defined into the ip(4) group, the ipv6AddrTable or the ipv6TcpConnTable had the same purpose than, respectively, the ipAddrTable or the tcpConnTable (Figure 6).

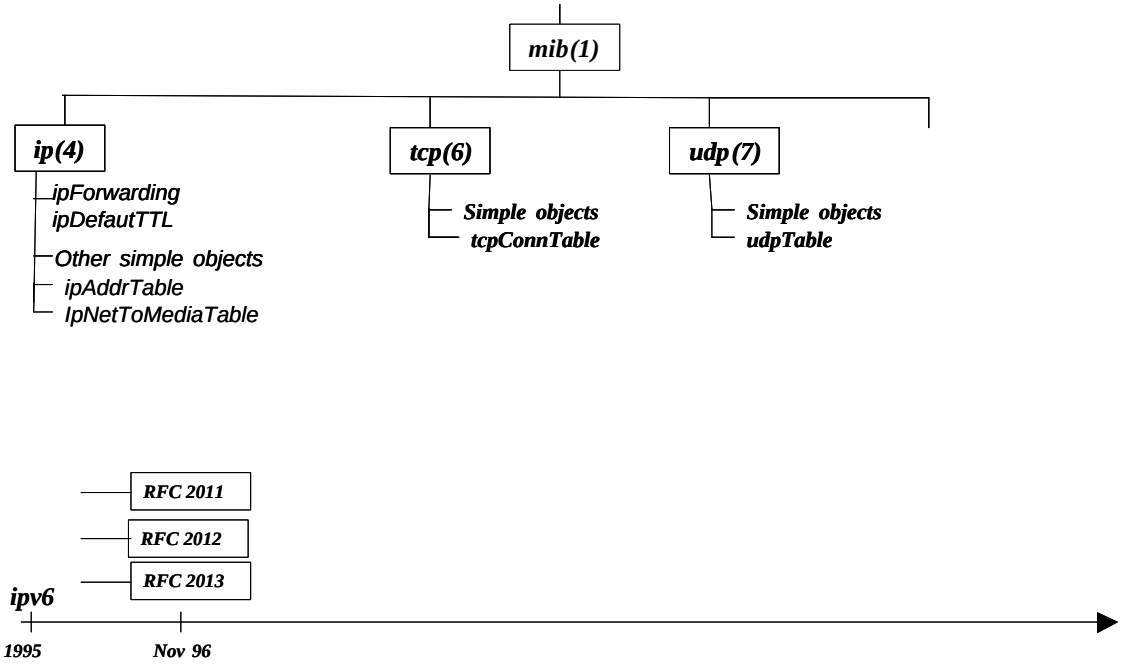

Figure 5: The MIB II in 1996

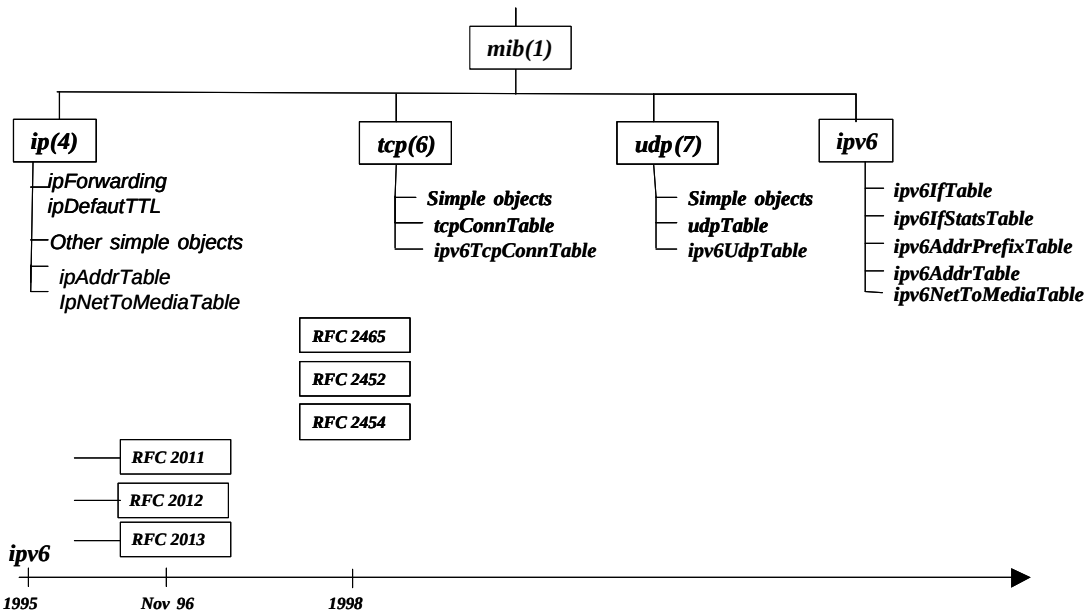

Figure 6: The definition of the separated IPv6 MIB

Further the fact that, by this approach, IPv6 was supposed to be another protocol, there was the risk to save wrong information into the table, like IPv4 data into IPv6 table and conversely.

- The unified approach, synthetized into the Figure 7, unified all the tables : ipAddrTable and ipv6AddrTable became ipAddressTable, ipNetToMediaTable and ipv6NetToMediaTable became inetNetToMediaTable, all the simple objects defined into the IPv4 MIB and the ipv6IfTable became the ipIfStatsTable. The same arrives with ipv6TcpConnTable and tcpConnTable which became tcpConnectionTable, and with udpTable and ipv6UdpTable, which became udpListenerTable. It must be noticed that, in addition to this table, issued from the IPv4 management architecture, new tables were defined like the ipv6InterfaceTable. See the drafts in Appendix for further information.

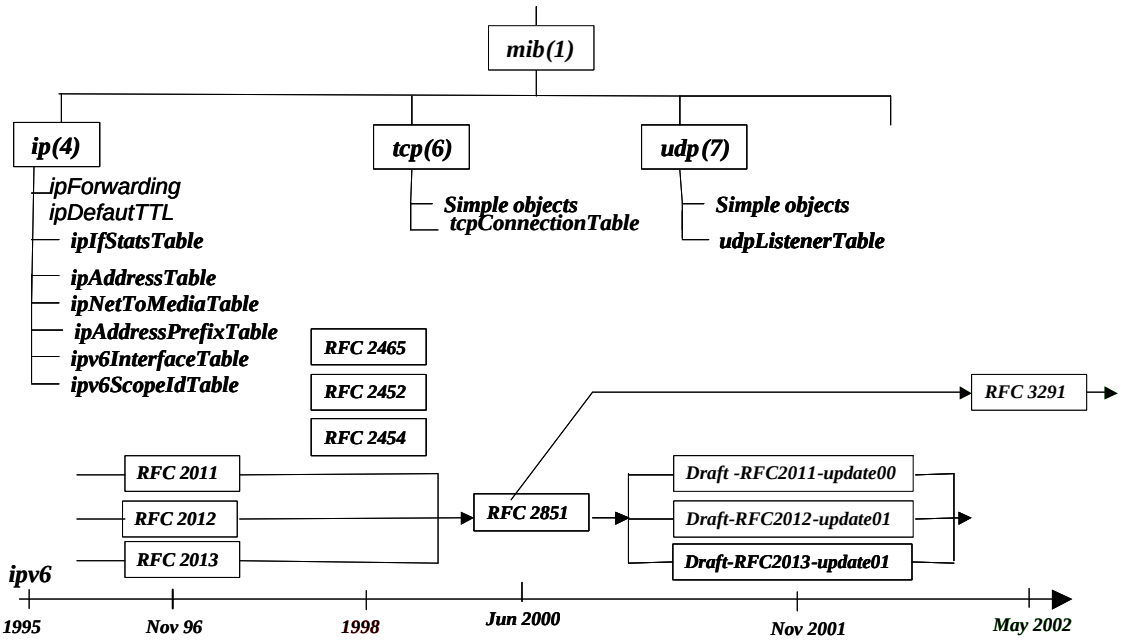

Figure 7: The unified MIB II

5 Net SNMP

As we believe that the unified MIB is the good approach, we decided to implement the June 2002 drafts into an OpenSource SNMP package. We choose the net-snmp project[7] because it was the most advanced project with IPv6. Our choice was confirmed by the announce that SNMP was ported

[7] www.net-snmp.org

upon IPv6 into the net-snmp-5.0.3 package. It was the first time that SNMP was implemented over IPv6. This announce avoid us to make ourselves this implementation, as we had planned to.

5.1 Introduction

This package was originally based on the Carnegie Mellon University[8] and University of California at Davis SNMP implementations (ucd-snmp toolkit). It has various tools relating to the Simple Network Management Protocol including:

- An extensible agent,
- An SNMP library,
- tools to request or set information from SNMP agents,
- tools to generate and handle SNMP traps,
- a version of the unix 'netstat' command using SNMP.

The package was used for extending the agent and extracting the management informations for the MIB II tables, as we will explain into the next sub-sections.

5.2 Command Set

The ucd-snmp toolkit provides a suite of command line applications that can be used to query and act on remote SNMP agents. For example:

- *snmpget* command can be used to retrieve data from a remote host given its host name, authentication information and an OID.
- *snmpgetnext* command, which is similar in usage to the *snmpget* command, is used to retrieve the next OID in the mib tree of data. Instead of returning the data requested, it returns the next OID in the tree and its value.
- The *snmpwalk* command essentially performs a whole series of getnexts automatically, and stops when it returns results that are no longer inside the range of the OID that was originally specified. This command can be used to get *all of the information* stored on a machine in the system MIB group.

5.3 Extending the Agent

There are several ways to extend the net-snmp agent. One is to implement new MIB modules.We will describe into the following sub-sections the mecanism used for their implementation.

In order to implement a new MIB module, three files are necessary:

- a MIB definition file,
- a C header file,
- a C implementation file.

[8] www.cmu.edu

5.3.1 The MIB File

It defines the MIB module to be implemented. The agent itself does not make any direct use of the MIB definitions. However, it is advisable to start with this because:

- It provides an initial specification for what is to be implemented.
- If the new MIB file is read in with the other MIB files, this lets the applications used to test the new agent, provide symbolic OIDs and values, rather than the bare numeric forms.
- The perl mib2c module uses this description to produce the two code files(see Figure 8). This is by far the easiest way to develop a new module.

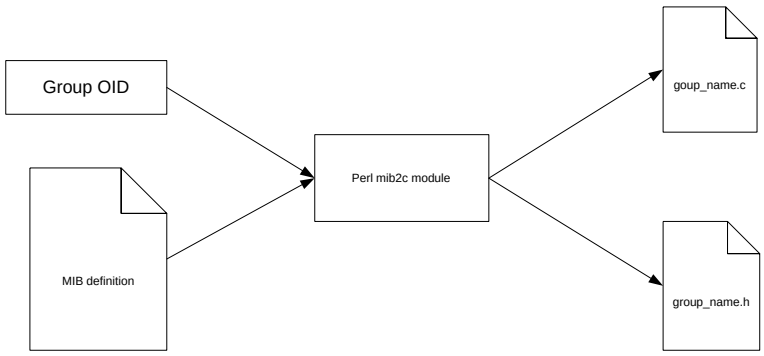

Figure 8: The use of the Perl mib2c module

5.3.2 The C Code Header file (the .h file)

It simply contains definitions of the public visible routines and the magic numbers, and can be generated completely by mib2c.

Public visible routines They are the prototypes of the functions that are used into the .c file. It could be:

- the init function:
  extern void init_ip(void);
- the function that search the implemented object instances:
  *extern u_char *var_ip(struct variable *vp, oid *name, size_t *length, int exact, size_t *var_len, WriteMethod **write_method)*

Magic Numbers Within the header file are defined a set of "magic numbers". They are used to differentiate one object to another in the same group, as it is easier to manipulate and save numbers rather than strings. Only objects that could be instanced own a magic number. Those with a Non-Accessible status should not have one.

5.3.3 Structure of the Implementation Code (the .c file)

The core work of implementing the module is done in the C code file. In this section, we present the common structure of all these files, that is their main elements.

The variables list The main element of this is a variable structure specifying the details of the objects implemented. This takes the form of an unconstrained array of type struct variableN (where N is the length of the longest suffix in the table). Thus :

```
struct variable2 example_variables[] = {
{magic_number, ASN_TYPE, ACCESS, handle_entry_routine, length, sub_OID}
        };
```

Each entry corresponds to one object in the MIB tree (or one column in the case of table entries), and these should be listed in increasing OID order. A single entry consists of six fields:

- a magic number (the #defined integer constant described above),
- a type indicator (from the values listed in <include/net-snmp/library/snmp_impl.h>),
- an access indicator (essentially RWRITE or RONLY),
- the name of the routine used to handle this entry,
- the length of the OID suffix used,
- an array of integers specifying this suffix.

Thus a typical variable list would look like:

```
struct variable4 ip_variables[] = {
....
{ IPADDRPREFIX, ASN_OCTET_STR, RONLY, var_ipAddressEntry, 3, {28,1,5}},
........
}
```

The location into the MIB tree The module also needs to declare the location within the MIB tree where it should be registered. This is done using a declaration of the form:

```
oid example_variables_oid[] = { SNMP_OID_MIB2, 4}
```

where, in this case, the group we want to implement is the group 4 under the MIB II tree (i.e. MIB II(4), i.e. ip).

The routines Several kind of routines could be defined into a code file. But they should contain at least the routine used to handle each entry. The init function should also be defined, into the unique .c file if it implements the whole group, or into one of the .c files implementing that group.

The init routine An exemple, from the ip.c file, is:

```
void init_ip(void)
{
REGISTER_MIB("mibII/ip", ip_variables, variable4, ip_variables_oid);
...
}
```

The first command (REGISTER_MIB) records the group into the MIB tree. It gives the variables list (*ip_variables*) and the place where this new group should be set (*ip_variables_oid*)

The routine handling the variable list entry This routine is mandatory. It handles a request for a particular variable instance. This is the routine that appeared in the variableN structure, so while the name is not fixed, it should be the same as was used there.

This routine has six parameters, and, four of these parameters are used for passing in information about the request :

```
struct variable *vp;
        // The entry in the variableN array from the
        //   header file, for the object under consideration.
        // Note that the name field of this structure has been
        //   completed into a fully qualified OID, by prepending
        //   the prefix common to the whole array.
oid *name;      // The OID from the request
int *length;    // The length of this OID
int exact;      // A flag to indicate whether this is an exact
                // request (GET/SET) or an 'inexact' one (GETNEXT)
```

Four of the parameters are used to return information about the answer. The function also returns a pointer to the actual data for the variable requested (or NULL if this data is not available for any reason) :

```
oid *name;      // The OID being returned
int *length;    // The length of this OID
int *var_len;   // The length of the answer being returned
WriteMethod **write_method;
                // A pointer to the SET function for this variable
```

Two of the parameters (name and length) serve a dual purpose, being used for both input and output.

This routine is defined as follow:

- first, the validation of the request, to ensure that it does indeed lie in the range implemented by this particular module,
- secondly, the association of the data received from the kernel or wherever they could be found, with their correct OID. This association is done in different way, depending if the requested object is simple or an element into a table.

- in the case of a simple object, this OID is immediatly found out because it is associated to a magic number,
- in the case of a table element, it is necessary to sort the data received from the kernel. So, we need to add this specific C code (here, in the case of an IPv4 address):

```
memcpy(current, vp->name, (int)vp->namelen * sizeof(oid));
/* current contains the required OID */
for (i = 0; i < nifs; i++) {
op = &current[10];
/* 10 is the length of the OID for an IP group table */
cp = (u_char *)&ifs[i].addr;
*op++ = *cp++;
*op++ = *cp++;
*op++ = *cp++;
*op++ = *cp++;
/* for an IPv4 address, we should add only the four bytes */
/* of the address */
if (snmp_oid_compare(current, 14, name, *length) == 0) {
memcpy(lowest, current, 14 * sizeof(oid));
lowinterface = i;
/* lowinterface sort the table item in a lexicographic order */
break;
/* no need to search further */
}
}
```

A similar C code should be add for IPv6 address (see 6.3.1 p 45).

- thirdly, the routine uses the Magic Number field from the vp parameter to determine which of the possible variables implemented is being requested. This is done using a switch statement, which should have as many cases as there are entries in the variableN array (or more precisely, as many as specify this routine as their handler), plus an additional default case to handle an erroneous call. For example:

```
switch(vp->magic){
case IPFORWARDING:
return(u_char *)&ipstat.ipForwarding;
case IPDEFAULTTTL:
return(u_char *)&ipstat.ipDefaultTTL;
...
}
```

5.4 sysctl and ioctl operations

One of the first steps employed by many programs that deal with the network interfaces on a system is to obtain from the kernel, information on all the interfaces configured on the system. The *ioctl* and *sysctl* functions are traditionally used for extracting the routing table within the kernel.
The two next sub-sections explain these two functions in the context of their used within our implemention. Fore more information about them, see [Ste98].

5.4.1 sysctl function

The PF_ROUTE domain is used for accessing the interface to the kernel's routing system in BSD. The operations supported by a raw socket in the PF_ROUTE domain, called routing sockets, are:

- a process can send a message to the kernel by writing to a routing socket. This is how routes are added and deleted,
- a process can read a message from the kernel on a routing socket,
- a process can use the sysctl function to either dump the routing table or to list all the configured interfaces.

The first two operations require super user privileges while the last one can be performed by any process.

The datalink socket address structure contains the return values of some of the messages returned on a routing socket. It is defined in net/if_dl.h

```
struct sockaddr_dl {
u_char sdl_len; /* Total length of sockaddr */
u_char sdl_family; /* AF_LINK */
u_short sdl_index; /* if != 0, system given index for interface */
u_char sdl_type; /* interface type */
u_char sdl_nlen; /* interface name length, no trailing 0 reqd. */
u_char sdl_alen; /* link level address length */
u_char sdl_slen; /* link layer selector length */
char sdl_data[12]; /* minimum work area, can be larger;
   contains both if name and ll address */
u_short sdl_rcf; /* source routing control */
u_short sdl_route[16]; /* source routing information */
};
```

An interface is identified as the connection between a network device and a network, and each interface has a unique positive index into the kernel representation. sdl_data member contains both the name and the link layer address (e.g. the 48 bit address for an Ethernet interface). The link layer address begins sdl_nlen bytes after the name.

```
#define LLADDR(s) ((caddr_t)((s)->sdl_data + (s)->sdl_nlen))
```

This header defines the macro to return the pointer to the link layer address.

After a process creates a routing socket, it can send commands to the kernel by writing to that socket, and read information from the kernel by reading from that socket. There are 12 different routing commands of which 5 can be used by the process. These are defined in net/route.h. The two message types used in our source code are:

- **RTM_IFINFO**: interface going up, down etc, the structure type being rt_msghdr,
- **RTM_NEWADDR**: address being added to interface, the structure type being ifa_msghdr.

The three different structures exchanged across a routing socket are:

- *rt_msghdr* :

```
struct rt_msghdr {
u_short rtm_msglen; /* to skip over non-understood messages */
u_char rtm_version; /* future binary compatibility */
u_char rtm_type; /* message type */
u_short rtm_index; /* index for associated ifp */
int rtm_flags; /* flags, incl. kern & message, e.g. DONE */
int rtm_addrs; /* bitmask identifying sockaddrs in msg */
pid_t rtm_pid; /* identify sender */
int rtm_seq; /* for sender to identify action */
int rtm_errno; /* why failed */
int rtm_use; /* from rtentry */
u_long rtm_inits; /* which metrics we are initializing */
struct rt_metrics rtm_rmx; /* metrics themselves */
};
```

  defined in net/route.h,

- *if_msghdr*:

```
struct if_msghdr {
u_short ifm_msglen; /* to skip over non-understood messages */
u_char ifm_version; /* future binary compatability */
u_char ifm_type; /* message type */
int ifm_addrs; /* like rtm_addrs */
int ifm_flags; /* value of if_flags */
u_short ifm_index; /* index for associated ifp */
struct if_data ifm_data;/* statistics and other data about if */
};
```

  defined in net/if.h.orig,

- and *ifa_ msghdr*:

```
struct ifa_msghdr {
u_short ifam_msglen; /* to skip over non-understood messages */
u_char ifam_version; /* future binary compatability */
u_char ifam_type; /* message type */
int ifam_addrs; /* like rtm_addrs */
int ifam_flags; /* value of ifa_flags */
u_short ifam_index; /* index for associated ifp */
int ifam_metric; /* value of ifa_metric */
};
```

  defined in net/if.h.orig.

The members rtm_addrs, ifm_addrs and ifam_addrs of those structures are bitmasks, specifying which of eight possible socket address structures follow the message. For instance, the bitmask **RTA_NETMASK** denotes a socket address structure containing the network mask, while for **RTA_IFA** the structure contains interface address.

Whereas, the creation of a routing socket requires super user priviledges, any process can examine the routing table and the interface list using sysctl function.

```
int sysctl(int *name, u_int namelen, void *oldp, size_t *oldlenp,
           void *newp, size_t newlen);
```

The *name* argument is an array of integers specifying the name, and *namelen* specifies the number of elements in the array. This element are rely by a hierarchical dependency (see the Figure 9). The first element in the array specifies which subsystem of the kernel the request is directed to (for example **CTL_NET**, for network control). The second element specifies some part of that subsystem (for example **AF_ROUTE**) and so on.

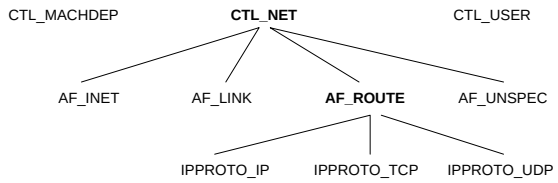

Figure 9: Hierarchical arrangement of sysctl names

To fetch a value, *oldp* points to a buffer in which the kernel stores the value. *oldlenp* is a value-result argument; when the function is called the value pointed to by oldlenp specifies the size of the oldp buffer, and on return the value contains the amount of data stored in this buffer by the kernel. As a special case, oldp can be null pointer and oldlenp a non null

pointer, and the kernel determines how much data the call would have returned and returns the size through oldlenp.

The *newp* and *newlen* fields are used to set a new value, and should be 0 if a new value is not being specified.

The first element of the name array is set to CTL_NET for the networking subsystem.

When the second element is AF_ROUTE, information on either the routing table or the interface list is returned, the third element (a protocol number) is always 0 (since there are no protocols within the AF_ROUTE family), the fourth element is an address family, and the fifth and sixth levels specify what to do.

Thus, to obtain the information on all the configured interface list the hierarchy used is (CTL_NET, AF_ROUTE, 0, AF_INET, NET_RT_IFLIST, 0).

The buffer returned by the kernel for this sysctl command is shown into the Figure 10.

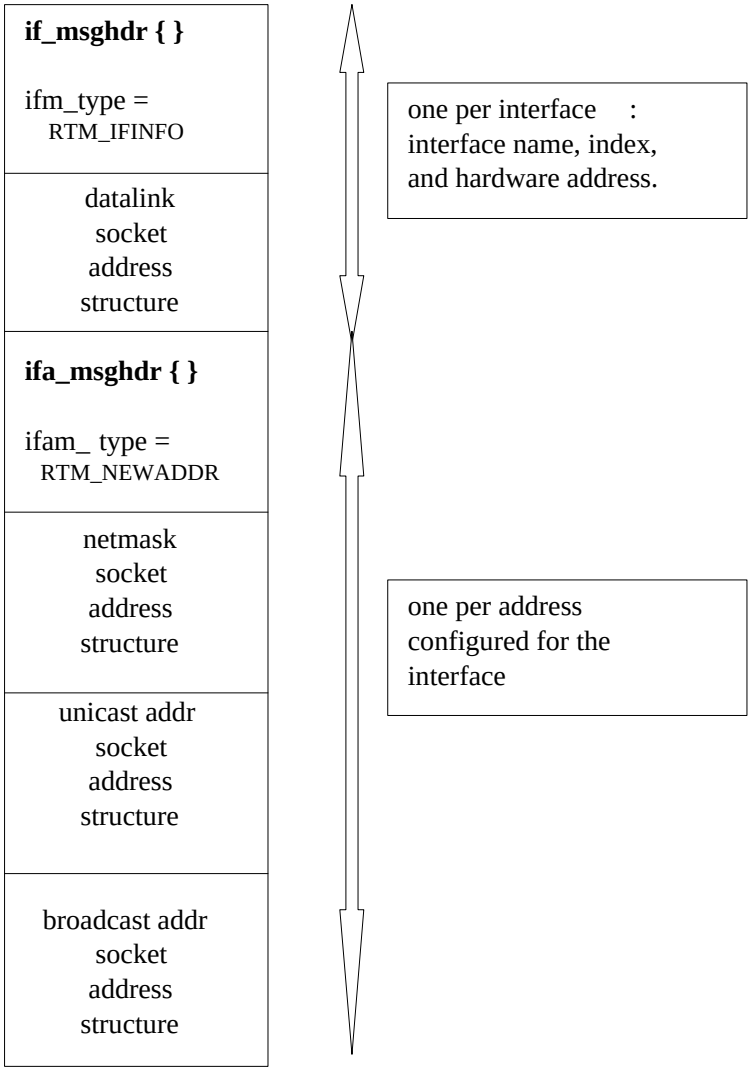

Figure 10: Structure of the buffer returned by the kernel

For each interface, a RTM_IFINFO message is returned, followed by a RTM_NEWADDR message for each address assigned to the interface. The RTM_IFINFO message is followed by one datalink socket address structure and each RTM_NEWADDR message is followed by up to three socket address structures: the interface address, the network mask, and the broadcast address.

5.4.2 ioctl function

For the *ioctl* operation, the function declaration is of the form :

```
ioctl(int fd, unsigned long request, ...);
```

This function affects an open file, referenced by the *fd* argument. The requests related to networking can be divided into six categories, socket operations, file operations, interface operations, ARP cache operations, routing table operations, stream system. The ioctl operation required for the part of the mib module implemented, relates to the interface operation, and the request is SIOCGIFAFLAG_IN6. It is used to obtain the IPv6 interface flags which are required for the ipAddressTable, and specify if the address is a unicast, an anycast or a multicast address. The structure in which the information retrieved from the kernel is stored is *in6_ifreq*.

```
struct in6_ifreq {
char ifr_name[IFNAMSIZ];
union {
struct sockaddr_in6 ifru_addr;
struct sockaddr_in6 ifru_dstaddr;
short ifru_flags;
int ifru_flags6;
int ifru_metric;
caddr_t ifru_data;
struct in6_addrlifetime ifru_lifetime;
struct in6_ifstat ifru_stat;
struct icmp6_ifstat ifru_icmp6stat;
u_int32_t ifru_scope_id[16];
} ifr_ifru;
};
```

The flags for IPv6 addresses are present in *ifru_flags6* field.

First, the *socket* function is used to create an end point for communication, and returns a descriptor. The declaration of the function has the form:

```
int  socket(int domain, int type, int protocol);
```

The domain parameter specifies a communication domain within which communication will take place; this selects the protocol family which should be used. As the ioctl function is called to obtain the IPv6 interface flags, the protocol family is AF_INET6.

The type parameter specifies the semantics of communication. The type SOCK_DGRAMsupports datagram sockets.

The protocol specifies a particular protocol to be used with the socket. Normally, only a single protocol exists to support a particular socket type within a given protocol family. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner. This field is kept 0 as only a single protocol exists for the type and domain used in our code. The descriptor thus returned by this function is used for making the ioctl function call.

6 Implementation of unified MIB II

6.1 Introduction

As we said above, the unified MIB II was mainly defined by 5 RFCs, which are:

- *RFC 3291*, for the IP addresses textual conventions ([DHRS02]),
- *Draft-ietf-ipngwg-rfc2011-update-00.txt*, for the IP Group. This module manages the IP and ICMP implementations, but not the management of IP routes,
- *Draft-ietf-ipngwg-rfc2012-update-01.txt*, for the TCP Group. This module allows to manage the TCP implementation,
- *Draft-ietf-ipngwg-rfc2013-update-01.txt*, for the UDP Group. This module allows to manage the UDP implementations,
- *Draft-ietf-ipngwg-rfc2096-update-00.txt*, for the ipForward Group. This module allows to manage the IP routes.

To implement the unified MIB II, we first had to implement the textual convention, then each group of this MIB. So, this section is defined as follow. The first sub-section describes the implementation of the textual convention, while the third sub-section describes the implementation of the several groups. To conclude, the last sub-section explains how those modifications were integrated into the net-snmp package.

6.2 Implementation of the RFC 3291

In this section, we will describe how the new textual convention for IP addresses was implemented.

The textual conventions defined into the RFC 3291 are defined with the TEXTUAL-CONVENTION macro. As the RFC 1903 ([CMRW96b]) says, this macro is used to *"convey the syntax and semantics associated with a textual convention. It should be noted that the expansion of the TEXTUAL-CONVENTION macro is something which conceptually happens during implementation and not during run-time"*.
The definition of a textual convention with this macro is more simple and do not require the definition of an ASN.1 type. The textual convention defined this way has the same ASN.1 type than its syntax. For example, the inetAddress is defined as follow into the RFC 3291:

```
InetAddress ::= TEXTUAL-CONVENTION
    STATUS      current
    DESCRIPTION
        "Denotes a generic Internet address.
         ...''
    SYNTAX      OCTET STRING (SIZE (0..255))
```

So, its associated ASN.1 type is ASN_OCTET_STRING.
As those textual convention are implemented only when they are used, and as the implementation of the drafts only require the definition of the InetAddressType, InetAddress and the InetAddressPrefixLength, for the moment, we had only implemented those three textual conventions.

For InetAddressType and InetAddressPrefixLength, as they are defined as INTEGER or Unsigned32, no specific implementations were needed.
It was not the same with the InetAddress textual convention. In the IPv4 MIB II, an IPv4 address was defined as an ASN.1 type ASN_IPADDRESS. This type was automatically displayed as a set of decimal numbers separated by a dot (“.”). In the unified MIB II, both IPv4 and IPv6 addresses are defined with an OCTET STRING(0...255), but their representation differs. The IPv4 address should be represented in the same way than for the IPv4 MIB II, while the IPv6 address should be represented as a set of hexadecimal numbers, separated by semi-colons (“:”). Also, the ipAddressAddr could contain DNS name, which are strictly octet strings. So, the solution we decided to implement was to transform the value of an IPv6 or IPv4 address into its associated string before saving this value into its ipAddressAddr object instance. This is done by the *inet_ntop()* function. The first argument of this function specifies which type of IP address it should transform. For example, this function will be called, in a case of an IPv6 address, as follow:

```
inet_ntop(AF_INET6, (void *)ifs[i].addr, buf6, sizeof(buf6))
```

where

- *ifs[i].addr6* is an *in6_addr* structure containing the binary IPv6 address (as it is returned from the kernel by the ioctl or sysctl function).
- *buf6* is a string which will contain this address as an octet string.

6.3 Implementation of the draft-ietf-ipngwg-rfc2011-update00.txt

.

This draft deeply redefines the RFC 2011. Most of the objects or tables defined into the draft are elements of the IP Group, but some of them are part of the MIB II or the Icmp Group. The following table (Table 6.3) lists the old items of the RFC 2011 and the corresponding new ones.

Under the ip Group :

Old item	New item	Description
ipForwarding	ipForwarding	Indicates if the entity is acting as an IPv4 router in respect to the forwarding of datagrams
ipDefaultTTL	ipDefaultTTL	The default value of the Time-To-Live field of the IPv4 header, whether a TTL value is not supplied by the Transport Layer protocol
ipInReceives	ipIfStatInReceives	All the following objects, until the ipAddrTable, are now defined into the ipIfStatstable
ipInHdrErrors	ipIfStatsInHdrErrors	
ipForwDatagrams	none	
ipInUnknownProtos	ipIfStatsInUnknownProtos	
ipInDiscards	ipIfStatsInDiscards	
ipInDelivers	ipIfStatsInDelivers	
ipOutRequests	ipIfStatsOutRequests	
ipOutDiscards	ipIfStatsOutDiscards	
ipOutNoRoutes	none	
ipReasmTimeout	none	
ipReasmReqds	ipIfStatsReasmReqds	
ipReasmOKs	ipIfStatsReasmOKs	
ipReasmFails	ipIfStatsReasmFails	
ipFragOKs	ipIfStatsOutFragOK	
ipFragFails	ipIfStatsOutFragFails	
ipFragCreates	ipIfStatsOutFragCreates	
ipAddrTable	ipAddressTable	The table containing all the information about inet address
ipNetToMedia	inetNetToMediaTable	The IP address translation table used for mapping from IP addresses to physical addresses

Under the ICMP Group , all the simple objects defined into the RFC 2011 were deprecated and new objects and tables were defined.

- *inetIcmpTable*, defines the generic counters,
- *inetIcmpMsgTable*, gathers the per-message ICMP counters, (per interface or system-wide),

Under the MIB and IP Groups , new tables and objects were defined:

- Under the MIB Group, the objects:
  - *ipv6Forwarding*, which defines if the entity is acting as a router for the IPv6 protocol or not,
  - *ipv6DefaultHopLimit*, which defines the default value of the Hop Limit field, whenever a TTL is not defined by the Transport Layer Protocol,
- Under the IP Group:
  - *ipv4IfTable*, which contains per-interface IPv4 specific information,
  - *ipv6InterfaceTable*, which contains per-interface IPv6 specific information,
  - *ipIfStatsTable*, which contains traffic statistics,
  - *ipAddressPrefixTable*, which is the inet prefix table,
  - *ipv6ScopeIdTable*, which describes IPv6 unicast and multicast scope zones.

To implement these new tables or objects, new files were defined:

- draftRFC2011.c (and .h), draft_icmp.h , implements the two Icmp Table : inetIcmpTable and inetIcmpMsgTable,
- ipv4If.c (and .h), implements the ipv4IfTable,
- ipAddressPrefix.c (and .h) implements the ipAddressPrefixTable,
- ipv6Interface.c (and .h) implements the ipv6InterfaceTable,
- ipIfStats.c (and .h), implements the ipIfStatsTable,
- ipAddress.c (and .h), implements the ipAddressTable,
- ipv6ScopeId.c (and .h), implements the ipv6ScopeIdTable,
- inetEntry.c (and .h), implements thet inetNetToMediaTable.

Because some objects defined into the RFC 2011 are still available into the draft, the ip.c file (and its corresponding .h) is still used. On the other hand, the icmp.c and the icmp.h files are not used.

6.3.1 Implementation of ipAddressTable

The ipAddressTable contains the details of all the addresses configured on the interfaces for an agent. It contains information for all the IP addresses for that particular agent. Thus, the table is indexed by ipAddressAddrType and ipAddressAddr. The information contained for each address is (see Table 6.3.1):

Object	Description
ipAddressAddrType	specifies whether the address is IPv6 or IPv4
ipAddressAddr	the address itself
ipAddressIfIndex	identifies the interface to which the address belongs
ipAddressType	type of the address, whether unicast , anycast etc
ipAddressPrefix	the OID of the ipAddressPrefixTable entry which contains the prefix for this particular address

ipAddressOrigin and ipAddressStatus are deprecated in the "unified" MIB II and so, no field is allotted for them, that is why they do not appear into the Table 6.3.1.
Below, we show some extracts of the C code to explain what was done. The whole code could be found into the ipAddress.c file.

```
0   struct iflist {
1
      int addrtype;
      int index;
      struct in_addr addr;
5     struct in6_addr addr6;
      oid prefix[128];
      int ipaddrtype;
    };
```

The main structure for the address entries in the ipAddressTable. The address information is extracted from the kernel using the sysctl function, and is stored in an array of type struct iflist pointed by ifs1 (a global variable of type struct iflist *). The meaning of each field of this structure is the following:

Field	Description
addrtype	denote the type of the ip address, whether IPv4 or IPv6
index	identifies the interface for which this address is configured
addr	stores the IPv4 addresses
addr6	stores the IPv6 addresses
prefix	the OID of the prefix table entry to which the prefix of the address corresponds
ipaddrtype	denotes whether it's a unicast, anycast or broadcast(for IPv4 only) address

The two structures defined to collect the IP addresses are:

```
struct in_addr {
in_addr_t s_addr;
};
```

defined in /usr/include/netinet/in.h , for the IPv4 address of 32 bits,

```
struct in6_addr {
union {
u_int8_t   __u6_addr8[16];
u_int16_t  __u6_addr16[8];
u_int32_t  __u6_addr32[4];
} __u6_addr; /* 128-bit IP6 address */
};
```

defined in /usr/include/netinet6/in6.h , for the 128 bits IPv6 addresses.

```
    static struct iflist *ifs,*ifs1,*ifs2;
10  static int nifs,naddrs1,naddrs2;

    static  char buf[INET_ADDRSTRLEN], buf6[INET6_ADDRSTRLEN];
```

ifs1 is the pointer to the array storing the details for the IPv4 addresses, and is indexed by *naddrs1*, while *ifs2* is used for the IPv6 addresses, indexed by *naddrs2*. *ifs* is the pointer for the array formed by combining the two address lists, and *nifs* is the count for the total address information available.

```
    #define ROUNDUP(a) \
15          ((a) > 0 ? (1 + (((a) - 1) | (sizeof(long) - 1))) : sizeof(long))
    #define ADVANCE(x, n) (x += ROUNDUP((n)->sa_len))

    static void
```

```
    rt_xaddrs(cp, cplim, rtinfo)
20          caddr_t cp, cplim;
            struct rt_addrinfo *rtinfo;
    {
      struct sockaddr *sa;
      int i;
25
      memset(rtinfo->rti_info, 0, sizeof(rtinfo->rti_info));

      for (i = 0; (i < RTAX_MAX) && (cp < cplim); i++) {
        if ((rtinfo->rti_addrs & (1 << i)) == 0)
30        continue;
        rtinfo->rti_info[i] = sa = (struct sockaddr *)cp;
        ADVANCE(cp, sa);
      }
    }
35
```

rt_xaddrs is the function used to loop through the socket addresses for an address. The loop in this function looks at each of the bitmask constants, as *rti_addrs* contains the bitmask identifying the socket addresses which are present, and for the bit present, the index is set to the corresponding structure. The socket address structures are of varying length, and the *sa_len* field, specifying their length is used to advance through the structures. The *ROUNDUP* function is used to round up to the next multiple of sizeof long as the addresses are aligned.
The *get_iflist* function (and *get_iflist6* function, see below) obtains the address table from the kernel, using the sysctl operation, with the following hierarchy:

```
static int mib[6] = { CTL_NET, PF_ROUTE, 0, AF_INET , NET_RT_IFLIST, 0 };
```

for IPv4 information and the following one:

```
static int mib[6] = { CTL_NET, PF_ROUTE, 0, AF_INET6 , NET_RT_IFLIST, 0 };
```

for IPv6 data (see 5.4.1).

```
    static void
    get_iflist(void)
    {
      int  bit;
40    static int mib[6] = { CTL_NET, PF_ROUTE, 0, AF_INET , NET_RT_IFLIST, 0 };
      char *cp, *ifbuf;
      size_t len;
      struct rt_msghdr *rtm;
      struct if_msghdr *ifm;
45    struct ifa_msghdr *ifam,*ifam1;
```

```
      int flags;
      int length;
      int i;
      u_char *temp;
50    struct sockaddr *sa;
      struct sockaddr_in *sin,*sin2;
      struct rt_addrinfo *info1;
      struct sockaddr_dl *sdl;

55    naddrs1 = 0;

      if (ifs1)
        free(ifs1);
      ifs1 = 0;
60    len = 0;
```

ifs1 is freed if allocated and the counters are initialised.

```
      if (sysctl(mib, 6, 0, &len, 0, 0) < 0)
        return;
```

The first sysctl call is made with the buffer set to 0 so as to obtain the length of the buffer returned by the function call into *len*.

```
      ifbuf = malloc(len);
      if (ifbuf == 0)
65      return;
      if (sysctl(mib, 6, ifbuf, &len, 0, 0) < 0) {
        syslog(LOG_WARNING, "sysctl net-route-iflist: %m");
        free(ifbuf);
        return;
70    }
```

The buffer *ifbuf* is then allocated with size *len*. The sysctl function is called again and the interface list returned by the kernel gets stored in the buffer *ifbuf*. And then a loop is executed to extract the information for each address for each of the interfaces that have been configured. The address information is indexed in the array pointed by *ifs1*. As the total number of addresses is not known before end, the first time the loop is executed, information about the addresses is not stored in the array, rather the number of addresses is computed, and at the end of the loop, the number is available in *naddrs1*. Space is then allocated for *ifs1*, and the loop is executed again, this time, storing the address information in the array. Below, we first write the loop, then we explain it, indexing each explanation with the associated line number.

```
loop:
```

```
      naddrs1 = 0;
      cp = ifbuf;
75    while (cp < &ifbuf[len]) {
        int gotaddr;

        gotaddr = 0;
        rtm = (struct rt_msghdr *)cp;
80
        if (rtm->rtm_version != RTM_VERSION || rtm->rtm_type != RTM_IFINFO){
          free(ifs);
          ifs = 0;
          free(ifbuf);
85        return;
        }
        ifm = (struct if_msghdr *)rtm;
        flags = ifm->ifm_flags;
        sdl = (struct sockaddr_dl *)(ifm + 1);
90
        cp += ifm->ifm_msglen;
        rtm = (struct rt_msghdr *)cp;

        while (cp < &ifbuf[len] && rtm->rtm_type == RTM_NEWADDR){
95        ifam = (struct ifa_msghdr *)rtm;
          ifam1 = ifam;
          cp += sizeof(*ifam);
          info1 = malloc(sizeof(*info1));
          info1->rti_addrs = ifam1->ifam_addrs;
100       rt_xaddrs((char *)(ifam1 + 1), ifam1->ifam_msglen + (char *)ifam1, info1);

          sin = (struct sockaddr_in *)info1->rti_info[RTAX_NETMASK];
          sin2 = (struct sockaddr_in *)info1->rti_info[RTAX_IFA];

105       length=calcul_prefix(&sin->sin_addr,sizeof(struct in_addr));
          temp = (u_char *)&(*sin2).sin_addr;

    /* from route.c */
    #define ROUND(a) \
110         ((a) > 0 ? (1 + (((a) - 1) | (sizeof(long) - 1))) : sizeof(long))

          for (bit = 1; bit && cp < &ifbuf[len]; bit <<= 1) {
            if (!(ifam->ifam_addrs & bit))
              continue;
115         sa = (struct sockaddr *)cp;
            cp += ROUND(sa->sa_len);

                                  /*
```

```
                                    * Netmasks are returned as bit
120                                 * strings of type AF_UNSPEC.  The
                                    * others are pretty ok.
                                    */
            if (bit == RTA_IFA) {

125   #define satosin(sa) ((struct sockaddr_in *)(sa))

              if (ifs1) {

                ifs1[naddrs1].addr  =satosin(sa)->sin_addr;
130             ifs1[naddrs1].index  = ifam->ifam_index;
                ifs1[naddrs1].addrtype = 1;
                ifs1[naddrs1].ipaddrtype = 1;
                memset(ifs1[naddrs1].prefix,0,128*sizeof(oid));

135             for(i = 13;length > 7;i++)
                  {
                    ifs1[naddrs1].prefix[i] = *temp++;
                    length -= 8;
                  }
140
                ifs1[naddrs1].prefix[i] = (0xff << (8 - length))&(*temp);
                ifs1[naddrs1].prefix[17] = calcul_prefix(&sin->sin_addr,sizeof(struct in_addr));
                ifs1[naddrs1].iporigin = 1;
              }
145           gotaddr = 1;
            }
          }

          if (gotaddr)
150         naddrs1++;

          cp = (char *)rtm + rtm->rtm_msglen;
          rtm = (struct rt_msghdr *)cp;
        }
155   }

      if (ifs1) {
        free(ifbuf);
        return;
160   }
      ifs1 = malloc(naddrs1 * sizeof(*ifs1));

      if (ifs1 == 0) {
        free(ifbuf);
```

```
165         return;
          }

          goto loop;
        }
```

At line 75, the outer while loop is used to increment the pointer through the entire interface information.
At line 79, rtm is pointing to the header at the beginning of each interface, and then a check is made for the version and type.
At line 89, *sdl* is used to extract the datalink structure present after the message header.
Line 92, rtm points to the message header present at the beginning of the first address which comes immediately after the interface information.
The while loop at line 94 is used to increment the pointer through the address information for any interface.
cp, at line 97, now points to the first socket address structure which follows the address message header.
The part of the code line 99 and line 100, is used to extract the socket address structures associated with each address and fills the array *info1* with pointers to the corresponding socket address structures. The array is indexed by the bitmasks which identify the socket addresses. Thus, the index RTA_IFA owns the pointer to the socket containing the interface address. rt_xaddrs loops through the socket address for an address.
The netmask address is stored in *sin* at line 102, and the interface address is stored in *sin2*, at line 103. The prefix length is then calculated from the netmask using the *calcul_prefix* function (line 105). This is done by counting the number of consecutive 1's from the right side in the bit representation. For e.g. a netmask of 255.128.0.0 corresponds to a prefix length of 9.
The loop beginning at line 112, is used to obtain the bitmask for the address so as to ascertain the socket addresses which are present. *ifam_addrs* has the bits for which the socket addresses are present turned "on", and so anding with the *bit* yields true value only for the socket addresses that are present.
The value 1 is assigned to *addrtype*, at line 131, as this table contains only IPv4 addresses.The value 1 is used for *ipaddrtype* as the bit corresponds to RTA_IFA and hence to unicast address. This code is executed only in the second phase of the loop (see oine 127), as during the first phase, *ifs1* has not been allocated.
The part of the code beginning at line 135 and ending at line 144, calculates the prefix address from the prefix length. The prefix address is required for the prefix OID, which is indexed by ipAddressIfIndex, IpAddressPrefixType, IpAddressPrefixprefix and, ipAddressPrefixLength.
At line 153, *rtm* now points to the next address of the interface, if present, or to the beginning of another interface.
After the number of addresses has been determined, ifs1 is allocated at line 161, and the loop is perform another time (line 168). The test defined line 157 avoid a third loop, because the second time, ifs1 was already allocated (see line 161), and so, the get_iflist function returns at line 159.
The code for extracting the IPv6 addresses is very similar, except for some modifications that are noticed.

```
170   static void
      get_iflist6(void)
      {
        int  bit;
        static int mib[6] = { CTL_NET, PF_ROUTE, 0, AF_INET6 , NET_RT_IFLIST, 0 };
175
```

The sysctl is called with another hierarchy defining IPv6 INET family.

```
        char *cp, *ifbuf;
        size_t len;
        struct rt_msghdr *rtm;
        struct if_msghdr *ifm;
180     struct ifa_msghdr *ifam, *ifam2;
        int flags;
        int length;
        int i;
        u_char *temp;
185     struct sockaddr *sa;
        struct sockaddr_in6 *sin,*sin2;
        struct        rt_addrinfo *info2;
        int s6;
        struct in6_ifreq ifr6;
190     u_int32_t flags6;
        struct sockaddr_dl *sdl;

        if (ifs2)
          free(ifs2);
195     ifs2 = 0;
        naddrs2 = 0;
        nifs = 0;
        len = 0;
        if (sysctl(mib, 6, 0, &len, 0, 0) < 0)
200       return;

        ifbuf = malloc(len);
        if (ifbuf == 0)
          return;
205     if (sysctl(mib, 6, ifbuf, &len, 0, 0) < 0) {
          syslog(LOG_WARNING, "sysctl net-route-iflist: %m");
          free(ifbuf);
          return;
        }
```

The address information is stored in array *ifs2* indexed by *naddrs2*. The datalink socket address structure, extracted after each interface, is used to obtain the interface name, which is required for making the ioctl function call.

```
210   loop:
        cp = ifbuf;

        while (cp < &ifbuf[len]) {

215       int gotaddr;

          gotaddr = 0;
          rtm = (struct rt_msghdr *)cp;

220       if (rtm->rtm_version != RTM_VERSION || rtm->rtm_type != RTM_IFINFO) {
            free(ifs2);
            ifs = 0;
            nifs = 0;
            free(ifbuf);
225         return;
          }
          ifm = (struct if_msghdr *)rtm;
          flags = ifm->ifm_flags;
          sdl = (struct sockaddr_dl *)(ifm + 1);
230
          cp += ifm->ifm_msglen;
          rtm = (struct rt_msghdr *)cp;

          while (cp < &ifbuf[len] && rtm->rtm_type == RTM_NEWADDR) {
235
            ifam = (struct ifa_msghdr *)rtm;
            ifam2 = ifam;
            cp += sizeof(*ifam);
            flags6 = 0;
240
            info2 = malloc(sizeof(*info2));
            info2->rti_addrs = ifam2->ifam_addrs;

            rt_xaddrs((char *)(ifam2 + 1), ifam2->ifam_msglen + (char *)ifam2, info2);
245
            sin = (struct sockaddr_in6 *)info2->rti_info[RTAX_NETMASK];
            sin2 = (struct sockaddr_in6 *)info2->rti_info[RTAX_IFA];
            if ((s6 = socket(AF_INET6, SOCK_DGRAM, 0)) < 0)
              return;
```

The *socket* function is called for extracting the descriptor *s6* for IPv6 datagrams. s6 is then used by ioctl for communicating with the kernel and extracting the address flag information for each address configured for a particular interface.

```
250         strcpy(ifr6.ifr_name,sdl->sdl_data);
```

```
        ifr6.ifr_addr = *sin2;
```

ifr6 is of type *struct in6_ ifreq* and is used to store the information retrieved from ioctl. The interface name, and the interface address is first copied on to the corresponding fields of this structure.

```
        if (ioctl(s6, SIOCGIFAFLAG_IN6, &ifr6) < 0)
          close(s6);

255     flags6 = ifr6.ifr_ifru.ifru_flags6;
        close(s6);
```

Then the ioctl function is called, making a request for the flag associated with this particular address, and the information is then stored in *flag6*.

```
        length=calcul_prefix(&sin->sin6_addr,sizeof(struct in6_addr));
        temp = (u_char *)&(*sin2).sin6_addr;

260     if(*temp == 0xfe && *(temp+1)== 0x80)
          *(temp+3)=0x00;
```

The prefix length is calculated from the netmask address, and the interface address is stored in *temp*. The next part of the code is a correction made for the link local addresses in FreeBSD. The addresses beginning with 0xfe80 are reserved for link local addresses in IPv6. The prefix of all link local addresses are of the form fe80:0:0:0::\64 followed by the 64 bits of the interface identifier. However, FreeBSD gives a specific representation for link local addresses by giving the first 64 bits as fe80:0n:0:0::\64, where n is the index of the interface to which the link local address belongs. So the correction defined lines 260 and 261, is required.

```
    /* from route.c */
    #define ROUND(a) \
            ((a) > 0 ? (1 + (((a) - 1) | (sizeof(long) - 1))) : sizeof(long))
265
          for (bit = 1; bit && cp < &ifbuf[len]; bit <<= 1) {
            if (!(ifam->ifam_addrs & bit))
              continue;
            sa = (struct sockaddr *)cp;
270         cp += ROUND(sa->sa_len);

                                    /*
                                     * Netmasks are returned as bit
                                     * strings of type AF_UNSPEC.  The
275                                  * others are pretty ok.
                                     */
            if (bit == RTA_IFA) {
```

```
    #define satosin6(sa) ((struct sockaddr_in6 *)(sa))
280
              if (ifs2) {

                                    ifs2[naddrs2].addr6  = satosin6(sa)->sin6_addr;
                                    ifs2[naddrs2].index  = ifam->ifam_index;
285                                 ifs2[naddrs2].addrtype = 2;

                                    if ((flags6 & IN6_IFF_ANYCAST) != 0 )
                                      ifs2[naddrs2].ipaddrtype = 2;
                                    else
290                                   ifs2[naddrs2].ipaddrtype = 1;
```

Within the loop

```
for (bit = 1; bit && cp < &ifbuf[len]; bit <<= 1) {
```

defined in line 266, when the values are stored in the *ifs2* array (line 281 and further), a check is made for the type of the IPv6 address, using the *flags6* variable (line 287). As the unicast and anycast addresses in IPv6 cannot be distinguished from the value of the address itself, this check is necessary to get the relevant information.

```
                                    ifs2[naddrs2].iporigin = 1;

                                    memset(ifs2[naddrs2].prefix,0,128*sizeof(oid));
                                    for(i = 13;length > 7;i++)
295                                   {
                                        ifs2[naddrs2].prefix[i] = *temp++;
                                        length -= 8;
                                      }

300                                 ifs2[naddrs2].prefix[i] = (0xff << (8 - length))&(*temp);
                                    ifs2[naddrs2].prefix[29] = calcul_prefix(&sin-
    >sin6_addr,sizeof(struct in6_addr));

              }
305           gotaddr = 1;
            }

          }

310       if (gotaddr)
            naddrs2++;

          cp = (char *)rtm + rtm->rtm_msglen;
          rtm = (struct rt_msghdr *)cp;
```

```
315       }
        }
        if (ifs2) {
          nifs = naddrs2 + naddrs1;
          free(ifbuf);
320       return;
        }
```

At the end of the second execution of the loop of the get_iflist6 function when the information has been stored in the array, the total number of addresses of both IPv4 and IPv6 are calculated and stored in *nifs*. This value is used later for the concatenation of the two arrays.

```
        ifs2 = malloc(naddrs2 * sizeof(*ifs2));
        if (ifs2 == 0) {
          free(ifbuf);
325       return;
        }
        naddrs2 = 0;
        goto loop;
      }
```

The following functions are useful miscellaneous functions:

```
330   static int
      calcul_prefix(val, size)
              void *val;
              int size;
      {
335     register u_char *name = (u_char *)val;
        register int byte, bit, plen = 0;

        for (byte = 0; byte < size; byte++, plen += 8)
          if (name[byte] != 0xff)
340         break;
        if (byte == size)
          return (plen);
        for (bit = 7; bit != 0; bit--, plen++)
          if (!(name[byte] & (1 << bit)))
345         break;

        for (; bit != 0; bit--)
          if (name[byte] & (1 << bit))
            return(0);
350           byte++;
              for (; byte < size; byte++)
                if (name[byte])
                  return(0);
```

```
            return (plen);
355 }


    static void free_buf(char *buf)
    {
360   int i;


      for (i = 0; i<INET_ADDRSTRLEN; i++)
        buf[i] = 'g';
365
    }

    static void free_buf6(char *buf6)
    {
370   int i;

      for (i = 0; i < INET6_ADDRSTRLEN; i++)
        buf6[i] = 'g';

375 }

    static int longueur_buf(char *buf)
    {
      int i;
380
      for (i = 0; i<INET_ADDRSTRLEN; i++){
        if (buf[i]=='g')
          return (i-1);
      }
385
    }


    static int longueur_buf6(char *buf6)
390 {
      int i;

      for (i = 0; i < INET6_ADDRSTRLEN; i++){
        if (buf6[i]=='g')
395       return (i-1);
      }
    }
```

var_ ipAddressEntry is the main function which is used by the agent to supply details for the ipAddressTable. It obtains the address table from the kernel by calling the functions

get_ iflist, for IPv4, and *get_ iflist6*, for IPv6, populating respectively, the two arrays *ifs1* and *ifs2*. Those arrays are concatenated into the array *ifs* (line 430 to 435).

```
   u_char *
   var_ipAddressEntry(struct variable *vp,
400                oid *name,
                   size_t *length,
                   int exact,
                   size_t *var_len,
                   WriteMethod **write_method)
405 {
          /*
           * object identifier is of form:
           * 1.3.6.1.2.1.4.20.1.?.A.B.C.D,  where A.B.C.D is IP address.
           * IPADDR starts at offset 10.
410        */
      oid lowest[28];
      oid current[28], *op;
      u_char *cp;
      int lowinterface = -1;
415   int i;
      char vide[]="0.0.0.0";
      int oid_compare_length;

          /* fill in object part of name for current (less sizeof instance part) */
420   memcpy(current, vp->name, (int)vp->namelen * sizeof(oid));


          /*
           * Get interface table from kernel.
425        */
      get_iflist();
      get_iflist6();
      ifs = malloc(nifs * sizeof(*ifs));

430   ifs = ifs1;
      for (i=0 ; i <naddrs1 ; i++)
        ifs[i] = ifs1[i];

      for (i=naddrs1; i < nifs; i++)
435     ifs[i]=ifs2[i-naddrs1];

      for (i = 0; i < nifs; i++) {

        op = &current[10];
440     if (i < naddrs1){
          cp = (u_char *)&ifs[i].addr;
```

```
            oid_compare_length=16;

            *op++ = 1;
445         *op++ = 32;

            *op++ = *cp++;
            *op++ = *cp++;
            *op++ = *cp++;
450         *op++ = *cp++;

          }
```

The first 10 fields of the object identifier are used to identify the ipAddressEntry in the MIB architecture. As the entries are indexed by the address type and the address itself, the next field of the OID, gives the address type (1 for IPv4 and 2 for IPv6), the next field then gives the length of the address in bits (32 bits for IPv4 and 128 for IPv6), and this is followed by the address itself (4 fields for representing an IPv4 address, and 16 for an IPv6 address, each field owning one byte of the address). This makes the total length of OID 16 for IPv4 and 28 for IPv6. *oid_ compare_ length* is used for distinguishing between the two different lengths.
The OID field is then filled by inserting the address type, address length and the address information, for each entry in *ifs*, starting from the first entry, until the constructed OID is then matched with the requested OID (for get request) or till the constructed OID is the appropriate OID (for the get-next request) (line 483 to 503). This filling is made first for IPv4 addresses (line 440 through 450), then for IPv6 addresses (line 454 through 480).

```
          else {
            cp = (u_char *)&ifs[i].addr6;
455         oid_compare_length=28;

            *op++ = 2;
            *op++ = 128;

460         *op++ = *cp++;
            *op++ = *cp++;
            *op++ = *cp++;
            *op++ = *cp++;

465         *op++ = *cp++;
            *op++ = *cp++;
            *op++ = *cp++;
            *op++ = *cp++;

470         *op++ = *cp++;
            *op++ = *cp++;
            *op++ = *cp++;
```

```
          *op++ = *cp++;

475       *op++ = *cp++;
          *op++ = *cp++;
          *op++ = *cp++;
          *op++ = *cp++;

480     }


        if (exact) {
          if (snmp_oid_compare(current, oid_compare_length, name, *length) == 0) {
485         memcpy(lowest, current, oid_compare_length * sizeof(oid));
            lowinterface = i;
            break;  /* no need to search further */
          }
        } else {
490       if ((snmp_oid_compare(current,oid_compare_length, name, *length) > 0) &&
    (lowinterface < 0 ||
             (snmp_oid_compare(current,oid_compare_length, lowest,oid_compare_length)
    < 0))) {
                /*
495              * if new one is greater than input
                 * and closer to input than previous
                 * lowest, save this one as the "next"
                 * one.
                 */
500         lowinterface = i;
            memcpy(lowest, current,oid_compare_length * sizeof(oid));
          }
        }

505   } /* end of the for */

      if (lowinterface < 0)
        return NULL;

510   i = lowinterface;

      if(ifs[i].addrtype==1)
        oid_compare_length=16;
      else
515     oid_compare_length=28;

      memcpy(name, lowest,oid_compare_length * sizeof(oid));
      *length = oid_compare_length;
```

```
520     *write_method = 0;

        *var_len = sizeof(long_return);

        switch (vp->magic) {
```

And then according to the magic number the information required is retrieved and returned.

```
525     case IPADDRADDRTYPE:
          long_return = ifs[i].addrtype;
          return (u_char *)&long_return;

        case IPADDRADDR:
530       switch (ifs[i].addrtype){
          case 1:
            free_buf(buf);

            if (inet_ntop(AF_INET,(void *)&ifs[i].addr, buf, sizeof(buf)) != NULL)
535           *var_len = longueur_buf(buf);

            long_return = buf;
            break;

540       case 2:
            free_buf6(buf6);
            if (inet_ntop(AF_INET6,(void *)&ifs[i].addr6, buf6, sizeof(buf6)) != NULL)
              *var_len = longueur_buf6(buf6);

545         long_return = buf6;
            break;

          case 0:
            long_return = vide;
550         break;

          }

          return (u_char *)long_return;
555
        case IPADDRIFINDEX:

       long_return = ifs[i].index;
       return (u_char *)&long_return;
560
        case IPADDRPREFIX:
```

```
        memcpy(ifs[i].prefix, name,7 * sizeof(oid));

565     ifs[i].prefix[7] = 27;
        ifs[i].prefix[8] = 1;
        ifs[i].prefix[9] = 3;
        ifs[i].prefix[10] = ifs[i].index;
        ifs[i].prefix[11] =ifs[i].addrtype;
570
        if(ifs[i].addrtype == 1)
          {
            ifs[i].prefix[12] = 32;
            *var_len=18*sizeof(oid);
575
          }
        else
          {
            ifs[i].prefix[12] = 128;
580         *var_len=30*sizeof(oid);

          }

        long_return = ifs[i].prefix;
585     return (u_char *)long_return;
```

The part of the code between line 565 to line 580 implements the ipAddrPrefix object of the ipAddrTable. This object is the OID of the ipAddressPrefixTable entry which contains the prefix of that particular address. So, here, we should calculates the OID of that entry in the ipAddressPrefixTable. The ipAddressPrefixTable location into the "unified" MIB II is {ip 27}. So the ipAddressPrefixPrefix corresponds to ip.27.1.3. Next part of the OID contains the IfIndex, followed by the PrefixType and the length of the prefix address in bits. The PrefixPrefix and the PrefixLength information is already present in the *prefix* array.

```
      case IPADDRTYPE:
        long_return = ifs[i].ipaddrtype;
        return (u_char *)&long_return;

590   default:
        DEBUGMSGTL(("snmpd", "unknown sub-id %d in var_ipAddressEntry\n", vp-
    >magic));
      }
      return NULL;
595 }
```

The other structures used in the code are

```
struct rt_addrinfo {
```

```
int rti_addrs;
struct sockaddr *rti_info[RTAX_MAX];
};
```

defined in net/route.h, and is used for storing the socket address structures corresponding to each address, depending on the bitmask present.

```
struct sockaddr {
u_char sa_len; /* total length */
sa_family_t sa_family; /* address family */
char sa_data[14]; /* actually longer; address value */
};
```

defined in sys/socket.h.

```
struct sockaddr_in6 {
u_int8_t sin6_len; /* length of this struct(sa_family_t)*/
u_int8_t sin6_family; /* AF_INET6 (sa_family_t) */
u_int16_t sin6_port; /* Transport layer port # (in_port_t)*/
u_int32_t sin6_flowinfo; /* IP6 flow information */
struct in6_addr sin6_addr; /* IP6 address */
u_int32_t sin6_scope_id; /* scope zone index */
};
```

defined in netinet6/in6.h.

6.3.2 Integration of ipAddressPrefixTable

The ipAddressPrefixTable is indexed by ipAddressPrefixIfIndex, ipAddressPrefixType, ipAddressPrefixPrefix, ipAddressPrefixLength. The fields of the table that were implemented are (see Table 6.3.2):

Object	Description
ipAddressPrefixIfIndex	identifies the interface
ipAddressPrefixType	identifies the address family
ipAddressPrefixPrefix	gives the prefix of the addresses configured for the interface
ipAddressPrefixLength	gives the prefix length
ipAddressPrefixAutonomousFlag	identifies whether the prefix can be used for autonomous address configuration
ipAddressPrefixPreferredLifetime	the time of the prefix until deprecation
ipAddressPrefixValidLifetime	the time of the prefix until invalidation

The last 3 fields are applicable only for IPv6 addresses.

The implementation for ipAddressPrefixTable is similar to that for ipAddressTable. For this table also, the interface list is obtained from the kernel using the sysctl function. The required fields for the table are then extracted from the interface list. The ipAddressPrefixTable is implemented into the ipAddressPrefix.c and ipAddressPrefix.h files. Below, we are going to explain the difference between the C code defined to implement the ipAddressTable and the one defined to implement the ipAddressPrefixTable.

The main structure for the address entries in the ipAddressPrefixTable is:

```
struct iflist {

  int addrtype;
  int index;
  struct in_addr addr;
  struct in6_addr addr6;
  int prefix_length;
};
```

and is defined in net-snmp-5.0.1/agent/mibgroup/mibII/ipAddressPrefix.c. The fields of this structure are the same than for the ipAddressTable, except the *prefix_length* field which is the length of the prefix returned.
The beginning of the file is the same than the previous one: same global variables, same ROUNDUP and rt_xaddrs functions. The *get_iflist* and *get_iflist6* functions get the interface list for, respectively, IPv4 and IPv6 addresses. They are defined as the previous get_iflist and get_iflist6 functions.

```
0   static void
1   get_iflist(void)
    {
      int  bit;
      static int mib[6] = { CTL_NET, PF_ROUTE, 0, AF_INET , NET_RT_IFLIST, 0 };
5
      char *cp, *ifbuf;
      size_t len;
      struct rt_msghdr *rtm;
      struct if_msghdr *ifm;
10    struct ifa_msghdr *ifam,*ifam1;
      int flags;
      int length;
      int i;
      u_char *temp1,*temp2;
15    struct sockaddr *sa;
      struct sockaddr_in *sin,*sin2;
      struct rt_addrinfo *info1;

      naddrs1 = 0;
20
      if (ifs1)
```

```
        free(ifs1);
      ifs1 = 0;
      len = 0;
25
      if (sysctl(mib, 6, 0, &len, 0, 0) < 0)
        return;

      ifbuf = malloc(len);
30    if (ifbuf == 0)
        return;
      if (sysctl(mib, 6, ifbuf, &len, 0, 0) < 0) {
        syslog(LOG_WARNING, "sysctl net-route-iflist: %m");
        free(ifbuf);
35      return;
      }

    loop:

40    naddrs1 = 0;
      cp = ifbuf;
      while (cp < &ifbuf[len]) {

        int gotaddr;
45
        gotaddr = 0;
        rtm = (struct rt_msghdr *)cp;
        if (rtm->rtm_version != RTM_VERSION || rtm->rtm_type != RTM_IFINFO) {
          free(ifs);
50        ifs = 0;
          free(ifbuf);
          return;
        }
        ifm = (struct if_msghdr *)rtm;
55      flags = ifm->ifm_flags;

        cp += ifm->ifm_msglen;
        rtm = (struct rt_msghdr *)cp;
        while (cp < &ifbuf[len] && rtm->rtm_type == RTM_NEWADDR) {
60
          ifam = (struct ifa_msghdr *)rtm;
          ifam1 = ifam;
          cp += sizeof(*ifam);

65        info1 = malloc(sizeof(*info1));
          info1->rti_addrs = ifam1->ifam_addrs;
          rt_xaddrs((char *)(ifam1 + 1), ifam1->ifam_msglen + (char *)ifam1, info1);
```

```
        sin = (struct sockaddr_in *)info1->rti_info[RTAX_NETMASK];
70      sin2 = (struct sockaddr_in *)info1->rti_info[RTAX_IFA];

        length=calcul_prefix(&sin->sin_addr,sizeof(struct in_addr));
        temp2 = (u_char *)&(*sin2).sin_addr;

75
    /* from route.c */
    #define ROUND(a) \
            ((a) > 0 ? (1 + (((a) - 1) | (sizeof(long) - 1))) : sizeof(long))

80      for (bit = 1; bit && cp < &ifbuf[len]; bit <<= 1) {
          if (!(ifam->ifam_addrs & bit))
            continue;
          sa = (struct sockaddr *)cp;
          cp += ROUND(sa->sa_len);
85
                                  /*
                                   * Netmasks are returned as bit
                                   * strings of type AF_UNSPEC.  The
                                   * others are pretty ok.
90                                 */
          if (bit == RTA_IFA) {


    #define satosin(sa) ((struct sockaddr_in *)(sa))
95
            if (ifs1) {
              memset((void *)&ifs1[naddrs1].addr, 0x00, sizeof(sin->sin_addr));
              for (temp1 = (u_char *)&ifs1[naddrs1].addr; length > 7; length -= 8)
                *temp1++ = *temp2++;
100
              *temp1 = (0xff << (8 - length))&(*temp2);

              ifs1[naddrs1].index  = ifam->ifam_index;
              ifs1[naddrs1].addrtype = 1;
105           ifs1[naddrs1].prefix_length =  calcul_prefix(&sin->sin_addr,sizeof(struct
    in_addr));

            }
            gotaddr = 1;                                    }
110
      }

      if (gotaddr)
```

```
            naddrs1++;
115
          cp = (char *)rtm + rtm->rtm_msglen;
          rtm = (struct rt_msghdr *)cp;
        }
      }
120   if (ifs1) {

        free(ifbuf);
        return;
      }
125   ifs1 = malloc(naddrs1 * sizeof(*ifs1));
      if (ifs1 == 0) {
        free(ifbuf);
        return;
      }
130
      goto loop;

    }

135
    static void
    get_iflist6(void)
    {
      int  bit;
140   static int mib[6] = { CTL_NET, PF_ROUTE, 0, AF_INET6 , NET_RT_IFLIST, 0 };

      char *cp, *ifbuf;
      size_t len;
      struct rt_msghdr *rtm;
145   struct if_msghdr *ifm;
      struct ifa_msghdr *ifam, *ifam2;
      int flags;
      int length;
      int i;
150   u_char *temp1,*temp2;
      struct sockaddr *sa;
      struct sockaddr_in6 *sin,*sin2;
      struct rt_addrinfo *info2;

155   if (ifs2)
        free(ifs2);
      ifs2 = 0;
      naddrs2 = 0;
      nifs = 0;
```

```
160     len = 0;
        if (sysctl(mib, 6, 0, &len, 0, 0) < 0)
          return;

        ifbuf = malloc(len);
165     if (ifbuf == 0)
          return;
        if (sysctl(mib, 6, ifbuf, &len, 0, 0) < 0) {
          syslog(LOG_WARNING, "sysctl net-route-iflist: %m");
          free(ifbuf);
170       return;
        }

      loop:
        cp = ifbuf;
175
        while (cp < &ifbuf[len]) {

          int gotaddr;

180       gotaddr = 0;
          rtm = (struct rt_msghdr *)cp;
          if (rtm->rtm_version != RTM_VERSION || rtm->rtm_type != RTM_IFINFO) {
            free(ifs2);
            ifs = 0;
185         nifs = 0;
            free(ifbuf);
            return;
          }
          ifm = (struct if_msghdr *)rtm;
190       cp += ifm->ifm_msglen;
          rtm = (struct rt_msghdr *)cp;

          while (cp < &ifbuf[len] && rtm->rtm_type == RTM_NEWADDR) {

195         ifam = (struct ifa_msghdr *)rtm;
            ifam2 = ifam;
            cp += sizeof(*ifam);

            info2 = malloc(sizeof(*info2));
200         info2->rti_addrs = ifam2->ifam_addrs;

            rt_xaddrs((char *)(ifam2 + 1), ifam2->ifam_msglen + (char *)ifam2, info2);

            sin = (struct sockaddr_in6 *)info2->rti_info[RTAX_NETMASK];
205         sin2 = (struct sockaddr_in6 *)info2->rti_info[RTAX_IFA];
```

```
        length=calcul_prefix(&sin->sin6_addr,sizeof(struct in6_addr));
        temp2 = (u_char *)&(*sin2).sin6_addr;

/* from route.c */
#define ROUND(a) \
        ((a) > 0 ? (1 + (((a) - 1) | (sizeof(long) - 1))) : sizeof(long))

        for (bit = 1; bit && cp < &ifbuf[len]; bit <<= 1) {
          if (!(ifam->ifam_addrs & bit))
            continue;
          sa = (struct sockaddr *)cp;
          cp += ROUND(sa->sa_len);

                                  /*
                                   * Netmasks are returned as bit
                                   * strings of type AF_UNSPEC.  The
                                   * others are pretty ok.
                                   */
          if (bit == RTA_IFA) {


#define satosin6(sa) ((struct sockaddr_in6 *)(sa))
            if (ifs2) {
              memset((void *)&ifs2[naddrs2].addr6, 0x00, sizeof(sin->sin6_addr));

              for (temp1 = (u_char *)&ifs2[naddrs2].addr6; length > 7; length -= 8)
                *temp1++ = *temp2++;

              *temp1 = (0xff << (8 - length))&(*temp2);

              temp1 = (u_char *)&ifs2[naddrs2].addr6;
              if(*temp1 == 0xfe && *(temp1+1)== 0x80)
                *(temp1+3)=0x00;

              ifs2[naddrs2].index  = ifam->ifam_index;
              ifs2[naddrs2].addrtype = 2;
              ifs2[naddrs2].prefix_length =  calcul_prefix(&sin->sin6_addr,sizeof(struct
in6_addr));

            }
            gotaddr = 1;
          }

        }
```

```
            if (gotaddr)
              naddrs2++;

255         cp = (char *)rtm + rtm->rtm_msglen;
            rtm = (struct rt_msghdr *)cp;
          }
        }
        if (ifs2) {
260       nifs = naddrs2 + naddrs1;
          free(ifbuf);
          return;
        }
        ifs2 = malloc(naddrs2 * sizeof(*ifs2));
265     if (ifs2 == 0) {
          free(ifbuf);
          return;
        }
        naddrs2 = 0;
270     goto loop;

      }
```

The calcul_prefix(), free_buf(), free_buf6(), longueur_buf(), longueur_buf6() functions into the ipAddressPrefix.c file are the same than the one defined into the ipAddress.c file. The main differences occured into the var_ip function, called here var_ipAddressPrefixEntry() , when the magic number is tested (line 405 and upper).

```
      u_char *
      var_ipAddressPrefixEntry(struct variable *vp,
275                   oid *name,
                      size_t *length,
                      int exact,
                      size_t *var_len,
                      WriteMethod **write_method)
280   {
            /*
             * object identifier is of form:
             * 1.3.6.1.2.1.4.20.1.?.A.B.C.D,  where A.B.C.D is IP address.
             * IPADDR starts at offset 10.
285          */
        oid lowest[30];
        oid current[30], *op;
        u_char *cp;
        int lowinterface = -1;
290     int i;
        char vide[]="0.0.0.0";
        int oid_compare_length;
```

```
int mib2[4]={ CTL_NET,AF_INET6,IPPROTO_IPV6,IPV6CTL_TEMPPLTIME};
```

The AF_INET6 domain is used to get or set variables affecting the Internet Protocols. The next level specifies the protocol using one of the IPPROTO_xxx constants.

```
295     int val;
        size_t len;

              /* fill in object part of name for current (less sizeof instance part) */
        memcpy(current, vp->name, (int)vp->namelen * sizeof(oid));
300

              /*
               * Get interface table from kernel.
               */
305     get_iflist();
        get_iflist6();

        ifs = malloc(nifs * sizeof(*ifs));

310     ifs = ifs1;
        for (i=0 ; i <naddrs1 ; i++)
          ifs[i] = ifs1[i];

        for (i=naddrs1; i < nifs; i++)
315       ifs[i]=ifs2[i-naddrs1];

        for (i = 0; i < nifs; i++) {

          op = &current[10];
320       *op++ = ifs[i].index;
          *op++ = ifs[i].addrtype;

          if(i<naddrs1){
            cp = (u_char *)&ifs[i].addr;
325         *op++ = 32;

            *op++ = *cp++;
            *op++ = *cp++;
            *op++ = *cp++;
330         *op++ = *cp++;

            oid_compare_length=18;

          }
335       else{
```

```
          cp = (u_char *)&ifs[i].addr6;
          *op++ = 128;

          *op++ = *cp++;
          *op++ = *cp++;
          *op++ = *cp++;
          *op++ = *cp++;

          *op++ = *cp++;
          *op++ = *cp++;
          *op++ = *cp++;
          *op++ = *cp++;

          *op++ = *cp++;
          *op++ = *cp++;
          *op++ = *cp++;
          *op++ = *cp++;

          *op++ = *cp++;
          *op++ = *cp++;
          *op++ = *cp++;
          *op++ = *cp++;

          oid_compare_length=30;
        }

        *op++ = ifs[i].prefix_length;

        if (exact) {
          if (snmp_oid_compare(current,oid_compare_length, name, *length) == 0) {
            memcpy(lowest, current,oid_compare_length * sizeof(oid));
            lowinterface = i;
            break;  /* no need to search further */
          }
        } else {
          if ((snmp_oid_compare(current,oid_compare_length, name, *length) > 0) &&
      (lowinterface < 0 ||

      (snmp_oid_compare(current,oid_compare_length, lowest, 30) < 0))){
                 /*
                  * if new one is greater than input
                  * and closer to input than previous
                  * lowest, save this one as the "next"
                  * one.
                  */
            lowinterface = i;
```

```
            memcpy(lowest, current,oid_compare_length * sizeof(oid));
          }
        }
385
      }

      if (lowinterface < 0)
        return NULL;
390
      i = lowinterface;

      if(ifs[i].addrtype==1)
        oid_compare_length=18;
395   else
        oid_compare_length=30;

      memcpy(name, lowest,oid_compare_length * sizeof(oid));
      *length = oid_compare_length;
400
      *write_method = 0;

      *var_len = sizeof(long_return);

405   switch (vp->magic) {

      case IPADDRPREFIXIFINDEX:
        long_return = ifs[i].index;
        return (u_char *)&long_return;
410
      case IPADDRPREFIXTYPE:
        long_return = ifs[i].addrtype;
        return (u_char *)&long_return;

415   case IPADDRPREFIXPREFIX:

        switch (ifs[i].addrtype){
        case 1:
          free_buf(buf);
420
          if (inet_ntop(AF_INET,(void *)&ifs[i].addr, buf, sizeof(buf)) != NULL)
            *var_len = longueur_buf(buf);

          long_return = buf;
425       break;

        case 2:
```

```
          free_buf6(buf6);

430       if (inet_ntop(AF_INET6,(void *)&ifs[i].addr6, buf6, sizeof(buf6)) != NULL)
            *var_len = longueur_buf6(buf6);

          long_return = buf6;
          break;
435
        case 0:
          long_return = vide;
          break;

440     }

        return (u_char *)long_return;

      case IPADDRPREFIXLENGTH:
445     long_return = ifs[i].prefix_length;
        return (u_char *)&long_return;

      case IPADDRPREFIXORIGIN:
        return NULL;
450
      case IPADDRPREFIXONLINKFLAG:
        return NULL;

      case IPADDRPREFIXAUTONOMOUSFLAG:
455     mib2[3]=IPV6CTL_AUTO_LINKLOCAL;
        len=sizeof(val);

        if(sysctl(mib2,4,&val,&len,NULL,0)<0)
          return NULL;
460
        if(ifs[i].addrtype==1)
          long_return = 2;
        else
          long_return=val;
465
        return (u_char *)&long_return;

      case IPADDRPREFIXADVPREFEREDLIFETIME:

470     mib2[3]=IPV6CTL_TEMPPLTIME;
        len=sizeof(val);

        if(sysctl(mib2,4,&val,&len,NULL,0)<0)
```

```
          return NULL;
475
        if(ifs[i].addrtype==1)
          long_return = 4294967295;
        else
          long_return=val;
480
        return (u_char *)&long_return;

      case IPADDRPREFIXADVVALIDLIFETIME:

485     mib2[3]=IPV6CTL_TEMPVLTIME;

        len=sizeof(val);

        if(sysctl(mib2,4,&val,&len,NULL,0)<0)
490       return NULL;

        if(ifs[i].addrtype==1)
          long_return = 4294967295;
        else
495       long_return=val;

        return (u_char *)&long_return;
```

The value of the autonomous flag, the preferred lifetime and the valid lifetime for the IPv6 address prefix is obtained using the sysctl function. As the values are not defined for IPv4 addresses, the autonomous flag is set to false and the lifetimes set to a value of 4294967295, which represents infinity.

```
      default:
        DEBUGMSGTL(("snmpd", "unknown sub-id %d in var_ipAddressPrefixEntry\n",
500 vp->magic));
      }
      return NULL;
    }
```

You can see into the Appendix C.1.2 section the output of this implementation.

6.3.3 Implementation of the IpIfStatsTable

The ipIfStatsTable is indexed by ipIfStatsAFType and ipIfStatsIfIndex. Unlike the previous two tables, it maintains management information for the entire interface and not for each particular address of that interface. However, as of now, FreeBSD supports only a few of the fields of this table. The fields of the IpIfStatsEntry which could be implemented are (see Table 6.3.3):

Object	Description
ipIfStatsAFType	gives the address family
ipIfStatsIfIndex	identifies the interface
ipIfStatsInReceives	the total number of input IP datagrams received by the interface, including those received in error
ipIfStatsInHdrErrors	number of input IP datagrams discarded due to errors in their IP headers
ipIfStatsInUnknownProtos	The number of locally-addressed IP datagrams received successfully but discarded because of an unknown or unsupported protocol.
ipIfStatsInOctets	The total number of octets in input IP datagrams received by the interface, including those received in error
ipIfStatsOutOctets	The total number of octets in output IP datagrams delivered to the lower layer on this interface
ipIfStatsInMcast	The number of IP multicast packets received by the interface
ipIfStatsOutMcast	The number of IP multicast packets transmitted by the interface

The code is implemented into the ipIfStat.c and ipIfStat.h files. It is similiar to that for the previous two tables, however the information is required only for each interface, and not for every address on that interface, and so the individual addresses are not traversed. The files included are the same as for ipAddress.c
The main structure used into the .c file to save the information read from the kernel is :

```
struct iflist {
        int addrtype;
        int index;
        u_long in_recieve;
        u_long in_errors;
        u_long  in_noproto;
        u_long in_octets;
        u_long  out_octets;
        u_long in_mcast;
        u_long  out_mcast;
};
```

The global variables defined at the beginning of the file are the same than for the previous tables. The get_iflist() and get_iflist6() functions are defined as follow :

```
0   static void
1   get_iflist(void)
    {
      int  bit;
      static int mib[6] = { CTL_NET, PF_ROUTE, 0, AF_INET , NET_RT_IFLIST, 0 };
5
      char *cp, *ifbuf;
      size_t len;
      struct rt_msghdr *rtm;
      struct if_msghdr *ifm;
10    struct ifa_msghdr *ifam,*ifam1;

      nindex1 = 0;

      if (ifs1)
15      free(ifs1);
      ifs1 = 0;
      len = 0;

      if (sysctl(mib, 6, 0, &len, 0, 0) < 0)
20      return;

      ifbuf = malloc(len);
      if (ifbuf == 0)
        return;
25    if (sysctl(mib, 6, ifbuf, &len, 0, 0) < 0){
        syslog(LOG_WARNING, "sysctl net-route-iflist: %m");
        free(ifbuf);
        return;
      }
30
    loop:
      nindex1 = 0;
      cp = ifbuf;
      while (cp < &ifbuf[len]) {
35      int gotindex;

        gotindex = 0;
        rtm = (struct rt_msghdr *)cp;
        if (rtm->rtm_version != RTM_VERSION || rtm->rtm_type != RTM_IFINFO) {
40        free(ifs);
          ifs = 0;
          free(ifbuf);
          return;
        }
45
```

```
        ifm = (struct if_msghdr *)rtm;
        cp += ifm->ifm_msglen;
        rtm = (struct rt_msghdr *)cp;

50      while (cp < &ifbuf[len] && rtm->rtm_type == RTM_NEWADDR) {

          if (ifs1 && !gotindex) {

            ifs1[nindex1].in_recieve=ifm->ifm_data.ifi_ipackets;
55          ifs1[nindex1].in_errors=ifm->ifm_data.ifi_ierrors;
            ifs1[nindex1].in_noproto=ifm->ifm_data.ifi_noproto;
            ifs1[nindex1].in_octets=ifm->ifm_data.ifi_ibytes;
            ifs1[nindex1].out_octets=ifm->ifm_data.ifi_obytes;
            ifs1[nindex1].in_mcast=ifm->ifm_data.ifi_imcasts;
60          ifs1[nindex1].out_mcast=ifm->ifm_data.ifi_omcasts;
            ifs1[nindex1].index  = ifm->ifm_index;
            ifs1[nindex1].addrtype = 1;

          }
65
          gotindex = 1;

          cp = (char *)rtm + rtm->rtm_msglen;
          rtm = (struct rt_msghdr *)cp;
70      }
```

After the interface information is extracted, the remaining addresses in the interface are skipped over, so as to traverse the outer while loop and extract the details for the next interface. Information for each address is not stored. However, it is necessary to traverse the header and reach the first address on each interface, so that details of only those interfaces which have proper configured addresses are stored.

```
        if (gotindex)
          nindex1++;

      }
75    if (ifs1) {
        free(ifbuf);
        return;
      }
      ifs1 = malloc(nindex1 * sizeof(*ifs1));
80    if (ifs1 == 0) {
        free(ifbuf);
        return;
      }

85    goto loop;
```

```
}
```

The interface information for IPv6 addresses is obtained similarly. The main function var_ipIfStatsEntry() is based on the same principle than the previous var_xxxx functions. See the Appendix B.1 for further details.

6.3.4 Implementation of the ipv6InterfaceTable

The ipv6InterfaceTable is indexed by ipv6InterfaceIfIndex, and the information is available for the entire interface and not every particular address on that interface. Information for ipv6InterfaceReasmMaxSize was not available, hence the fields which were implemented are defined into the Table 6.3.4:

Object	Description
ipv6InterfaceIfIndex	identifies the interface
ipv6InterfaceEffectiveMtu	the size of the largest IPv6 packet which can be sent/received on the interface, specified in octets.
ipv6InterfaceIdentifier	the Interface Identifier for this interface that is (at least) unique on the link this interface is attached to.
ipv6InterfaceIdentifierLength	the length of the Interface Identifier in bits
ipv6InterfacePhysicalAddress	the interface's physical address

The fields in this table store information for the entire interface, however the InterfaceIdentifier field is obtained by extracting the last 64 bits of the link local address of that interface, hence it is necessary to traverse all the addresses belonging to that interface in order to reach the link local address. The files included are the same as for ipAddress.c.

The main structure defined to save the information received from the kernel is :

```
struct iflist {
        int index;
        u_long effect_mtu;
        char PhysAddr[6];
        u_char Interface_Id[8];
        int prefixlength;
};
```

Here is only defined the get_iflist6 functions as follow:

```
0
1   static void
    get_iflist6(void)
```

```
{
  int bit;
  static int mib[6] = { CTL_NET, PF_ROUTE, 0, AF_INET6 , NET_RT_IFLIST, 0 };

  char PhysAddr[6];
  char *cp, *ifbuf;
  u_char *temp;
  size_t len;
  struct rt_msghdr *rtm;
  struct if_msghdr *ifm;
  struct ifa_msghdr *ifam, *ifam2;
  struct sockaddr_dl *sdl;
  struct sockaddr *sa;
  struct sockaddr_in6 *sin,*sin2;
  struct rt_addrinfo *info2;
  int length;

  if (ifs)
    free(ifs);
  ifs = 0;
  nifs = 0;
  len = 0;
  if (sysctl(mib, 6, 0, &len, 0, 0) < 0)
    return;

  ifbuf = malloc(len);
  if (ifbuf == 0)
    return;
  if (sysctl(mib, 6, ifbuf, &len, 0, 0) < 0) {
    syslog(LOG_WARNING, "sysctl net-route-iflist: %m");
    free(ifbuf);
    return;
  }

loop:
  nifs = 0;
  cp = ifbuf;

  while (cp < &ifbuf[len]) {
    int gotaddr;

    gotaddr = 0;
    rtm = (struct rt_msghdr *)cp;

    if (rtm->rtm_version != RTM_VERSION || rtm->rtm_type != RTM_IFINFO) {
      free(ifs);
```

```
          free(ifbuf);
          return;
        }

        ifm = (struct if_msghdr *)rtm;
        sdl = (struct sockaddr_dl *)(ifm + 1);

        memset(PhysAddr,0,sizeof(PhysAddr));
        memcpy(PhysAddr, (char *) LLADDR(sdl), sdl->sdl_alen);
```

The datalink socket address following each interface header contains both the name and physical address for the link, and the physical address is extracted from this structure.

```
        cp += ifm->ifm_msglen;
        rtm = (struct rt_msghdr *)cp;

        while (cp < &ifbuf[len] && rtm->rtm_type == RTM_NEWADDR) {

          ifam = (struct ifa_msghdr *)rtm;
          ifam2 = ifam;
          cp += sizeof(*ifam);

          info2 = malloc(sizeof(*info2));
          info2->rti_addrs = ifam2->ifam_addrs;

          rt_xaddrs((char *)(ifam2 + 1), ifam2->ifam_msglen + (char *)ifam2, info2);

          sin = (struct sockaddr_in6 *)info2->rti_info[RTAX_NETMASK];
          sin2 = (struct sockaddr_in6 *)info2->rti_info[RTAX_IFA];


    /* from route.c */
    #define ROUND(a) \
            ((a) > 0 ? (1 + (((a) - 1) | (sizeof(long) - 1))) : sizeof(long))

          for (bit = 1; bit && cp < &ifbuf[len]; bit <<= 1) {
            if (!(ifam->ifam_addrs & bit))
              continue;
            sa = (struct sockaddr *)cp;
            cp += ROUND(sa->sa_len);

                                    /*
                                     * Netmasks are returned as bit
                                     * strings of type AF_UNSPEC.  The
                                     * others are pretty ok.
```

```
                                    */
            if (bit == RTA_IFA) {

95
     #define satosin6(sa) ((struct sockaddr_in6 *)(sa))

              temp = (u_char*)&sin2->sin6_addr;
              length = calcul_prefix(&sin->sin6_addr,sizeof(struct in6_addr));
100
              if(*temp == 0xfe && *(temp+1) == 0x80 || length ==128){

                if (ifs && !gotaddr) {

105               memcpy(ifs[nifs].PhysAddr, PhysAddr, sizeof(PhysAddr));
                  ifs[nifs].effect_mtu=ifm->ifm_data.ifi_mtu;
                  ifs[nifs].index  = ifam->ifam_index;
                  ifs[nifs].prefixlength =  length;
                  memcpy(ifs[nifs].Interface_Id,(temp+8),8 * sizeof(u_char));
110               }

                  gotaddr = 1;

                }
115
```

For each interface address a check is made whether it is the link local address of the interface, as all link local addresses begin with 0xfe80, or whether it is the loopback address of the interface, as all IPv6 loopback addresses have a prefix length of 128. A check is made for loopback addresses, because the loopback interface is a virtual one, and so no link local address is configured for this interface .The InterfaceIdentifier is then obtained by extracting the last 64 bits of the link local address or the loopback address.

```
              }

            }
120
            cp = (char *)rtm + rtm->rtm_msglen;
            rtm = (struct rt_msghdr *)cp;
          }
          if (gotaddr)
125
            nifs++;

        }
        if (ifs) {
130       free(ifbuf);
```

```
      return;
    }
    ifs = malloc(nifs * sizeof(*ifs));
    if (ifs == 0) {
      free(ifbuf);
      return;
    }
    goto loop;

}
```

The main function var_ipv6interfaceEntry() is of the same type than the previous var_xxx function. But, care should be taken about the loopback address :

```
u_char *
var_ipv6InterfaceEntry(struct variable *vp,
                oid *name,
                size_t *length,
                int exact,
                size_t *var_len,
                WriteMethod **write_method)
{

  oid lowest[11];
  oid current[11], *op;
  u_char *cp;
  int lowinterface = -1;
  int i;
  int start;
  char vide[]="0.0.0.0";


       /* fill in object part of name for current (less sizeof instance part) */
  memcpy(current, vp->name, (int)vp->namelen * sizeof(oid));


  /*
   * Get interface table from kernel.
   */
  nifs=0;

  get_iflist6();

  for (i = 0; i < nifs; i++) {

    op = &current[10];
    *op++=ifs[i].index;
```

```
175
        if (exact) {
          if (snmp_oid_compare(current,11, name, *length) == 0) {
            memcpy(lowest, current,11 * sizeof(oid));
            lowinterface = i;
180         break;  /* no need to search further */
          }
        } else {
                if ((snmp_oid_compare(current,11, name, *length) > 0) &&
                    (lowinterface < 0
185                  || (snmp_oid_compare(current,11, lowest,11) < 0))) {
                  /*
                   * if new one is greater than input
                   * and closer to input than previous
                   * lowest, save this one as the "next"
190                * one.
                   */
                  lowinterface = i;
                  memcpy(lowest, current,11 * sizeof(oid));
                }
195     }

      }

            if (lowinterface < 0)
200           return NULL;

            i = lowinterface;

            memcpy(name, lowest,11 * sizeof(oid));
205         *length = 11;

            *write_method = 0;
            *var_len = sizeof(long_return);

210         switch (vp->magic) {

            case IPV6IFINDEX :
              long_return = ifs[i].index;
              return (u_char *)&long_return;
215
            case IPV6EFFECTIVEMTU :
              long_return = ifs[i].effect_mtu;
              return (u_char *)&long_return;
```

```
220             case IPV6REASMMAXSIZE :
                  return NULL;

                case IPV6IDENTIFIER :
                  long_return = ifs[i].Interface_Id;
225
                  *var_len = 8 * sizeof(u_char);
                  return (u_char *)long_return;


230             case IPV6IDENTIFIERLENGTH :
                  long_return =64;
                  return (u_char *)&long_return;

                case IPV6PHYSICALADDRESS :
235               *var_len = sizeof(ifs[i].PhysAddr);

                  if(ifs[i].prefixlength==128)
                    *var_len=0;
```

The interface for the loop back addresses is a virtual one, and so no physical address is associated with it.

```
                  return (u_char *) (ifs[i].PhysAddr);
240

                default:
                  DEBUGMSGTL(("snmpd", "unknown sub-id %d in var_ipAddressEntry\n",
      vp->magic));
245             }
                return NULL;
      }
```

6.3.5 Implementation of ipv4IfTable

This implementation is made into the file ipv4If.c and ipv4If.h. This table is indexed by ipv4IfIndex, and the entries are per interface. However, information for ipv4IfReasmMaxSize could not be obtained for FreeBSD, and as such this table only serves to provide the interface index information. Consequently, the information received from the sysctl call are memorized into a structure of iflist type, limited at:

```
struct iflist {
        int index;
};
```

As for the previous two tables, ipIfstats and ipv6Interface tables, the management information is maintained per interface. The include files remain the same.
The parameters of the sysctl call are:

```
static int mib[6] = { CTL_NET, PF_ROUTE, 0, AF_INET , NET_RT_IFLIST, 0 }
```

The index is defined into the ifa_msghdr structure of a RTM_NEWADDR type. See code of the appropriate function in Appendix B.2.

6.3.6 Implementation of the inetNetToMediaTable

For implementing table, structures used are:

```
struct iflist {
  int ifphysflag;
  int type;
  int addrtype;
  int index;
  struct in_addr addr;
  struct in6_addr addr6;
  char     PhysAddr[6];
};
```

In the structure, if *ifphysflag* is not set, it denotes the loopback address. It is important to have this information because the representation defined into the MIB for this virtual address is a String of zero length.
The *type* field is for inetNetToMediaType, to precise if this media is IPv4 or IPv6 enable.
The *addrtype* field of this structure, denotes type of address, IPv4 or IPv6.
The *index* field of this structure shows the interface index.

The *addr* field is an IPv4 address, its type is struct in_addr declared in /usr/include/netinet/in.h.
The *addr6* field is for IPv6 address, its type is struct in6_addr (/usr/include/netinet6/in6.h).
The *PhysAddr* field is for Physical address of 48 bits.

The main function of this implementation is the var_inetEntry function, where the interface table from kernel is obtained. For IPv4 , it is obtained by calling function get_iflist and the table is stored in array of structures of type *struct iflist* pointed by ifs1 (a global variable declared as struct iflist *ifs1). For IPv6, get_iflist6 is called and the table is stored in array of structures of type *struct iflist* pointed by ifs2 (a global variable declared as struct iflist *ifs2).
In get_iflist() function, sysctl function is used for getting information from kernel. The call is:

```
0
1   static int mib[6] = { CTL_NET, PF_ROUTE, 0, AF_INET , NET_RT_IFLIST, 0 };
```

The first argument, CTL_NET, denotes the networking subsystem. PF_ROUTEasks the kernel to return information on either the routing table or the interface list. The third element is a protocol number and it is 0, because there are no protocols within PF_ROUTE family as they are in AF_INETfamily. The fourth one, AF_INET, denotes the address family IPv4 and the fifth one, NET_RT_IFLIST, is for information of all configured interface. The

sixth one is zero, indicating that information will be given on all configured interfaces. If it were nonzero, it will indicate an interface number and only information on that interface would be returned.

```
        if (sysctl(mib, 6, 0, &len, 0, 0) < 0)
                return;
        ifbuf = malloc(len);
```

This sysctl is called to know the size of buffer returned by kernel. It allows to allocate a buffer large enough to save the information we need and then sysctl function is again called, and the information is stored in ifbuf.

```
        if (sysctl(mib, 6, ifbuf, &len, 0, 0) < 0) {
                syslog(LOG_WARNING, "sysctl net-route-iflist: %m");
                free(ifbuf);
                ifbuf=0;
                return;
        }
loop:
        naddrs1 = 0;
        cp = ifbuf;
```

cp points to the beginning of the buffer. We are now going to search all the configured addresses of all the interfaces of the station. The loop:

```
while (cp < &ifbuf[len] && rtm->rtm_type == RTM_NEWADDR)
```

means that we are looking for the ifam_addrs structure for all addresses configured on that interfaces, while the loop:

```
  while (cp < &ifbuf[len])
```

means that we are looking for all the interfaces.

```
        while (cp < &ifbuf[len]) {
                int gotaddr;

                gotaddr = 0;
                rtm = (struct rt_msghdr *)cp;
                if (rtm->rtm_version != RTM_VERSION
                    || rtm->rtm_type != RTM_IFINFO) {
                        free(ifs);
                        ifs = 0;
                        free(ifbuf);
                        return;
                }
                ifm = (struct if_msghdr *)rtm;
                flags = ifm->ifm_flags;
```

The structure if_msghdr is extracted from the block of data which type is RTM_VERSION. Then, the datalink socket is pointed by sdl:

```
                        sdl = (struct sockaddr_dl *)(ifm + 1);
                    memcpy(PhysAddr, (char *) LLADDR(sdl), sdl->sdl_alen);
                cp += ifm->ifm_msglen;
35              rtm = (struct rt_msghdr *)cp;
                while (cp < &ifbuf[len] && rtm->rtm_type == RTM_NEWADDR) {
```

The sdl variable is defined as a struct sockaddr_dl declared in /usr/include/net/if_dl.h (see the 5.4.1 section). The sdl_data member contains both the name and link layer address (e.g. the 48-bit MAC address for an Ethernet interface). The name begins at sdl_data[0] and is not null terminated. The link-layer address begins sdl_nlen bytes after the name. The LLADDR macro returns the pointer on the link layer address (see 5.4.1). So from this datalink socket address structure, we are able to get physical or link-layer address corresponding to the interface:

```
memcpy(PhysAddr, (char *) LLADDR(sdl), sdl->sdl_alen);
```

After this header, the buffer ifbuf contains struct ifa_msghdr followed up to three socket address structures: the interface address, the network mask, and the broadcast address. So, first, struct ifa_maghdr is extracted and then cp is made to point to start of these three socket address structures.

```
                        ifam = (struct ifa_msghdr *)rtm;
                        cp += sizeof(*ifam);
```

In the structure ifa_msghdr, there is a field ifam_addrs which is basically bitmask identifying socket address structures. Since we are concerned with the IPv4 address related to the interface, we get this socket address structure and confirm it by seeing whether bit RTA_IFA is set or not (line 53):

```
    /* from route.c */
40  #define ROUND(a) \
        ((a) > 0 ? (1 + (((a) - 1) | (sizeof(long) - 1))) : sizeof(long))
                        for (bit = 1; bit && cp < &ifbuf[len]; bit <<= 1) {
                                if (!(ifam->ifam_addrs & bit))
                                        continue;
45                              sa = (struct sockaddr *)cp;
                                cp += ROUND(sa->sa_len);

                                /*
                                 * Netmasks are returned as bit
50                               * strings of type AF_UNSPEC.  The
                                 * others are pretty ok.
                                 */
                                if (bit == RTA_IFA) {
    #define satosin(sa) ((struct sockaddr_in *)(sa))
```

```
55                                      if (ifs1) {
                                           memcpy(ifs1[naddrs1].PhysAddr, PhysAddr, sizeof(PhysAddr));
                                           if(sdl->sdl_alen != 0)
                                             ifs1[naddrs1].ifphysflag=1;
                                           else
60                                            ifs1[naddrs1].ifphysflag=0;
                                           ifs1[naddrs1].type=1;
                                          ifs1[naddrs1].addr  =satosin(sa)->sin_addr;
                                          ifs1[naddrs1].index  = ifam->ifam_index;
                                          ifs1[naddrs1].addrtype = 1;
65                                      }
                                        gotaddr = 1;
                                      }
                              }

70                            if (gotaddr)
                                naddrs1++;
                              cp = (char *)rtm + rtm->rtm_msglen;
                              rtm = (struct rt_msghdr *)cp;
                      }
75            }
              if (ifs1) {
                free(ifbuf);
                return;
              }
80            ifs1 = malloc(naddrs1 * sizeof(*ifs1));
              if (ifs1 == 0) {
                      free(ifbuf);
                      return;
              }
85
              goto loop;
```

The algorithm works as follows, in the first pass, total number of interface-IPv4 address combinations is known, and memory is allocated for that much size to ifs1 (see line 80). Then, in the second phase, the test "if(ifs1)", line 55, is valid and each field of structure pointed by ifs1[k] is filled, where k varies from zero to naddrs1 (here it is total number of interface-IPv4 address combinations). Thus from get_iflist() function call, we get a list of information for each interface and each address configured for that interface for IPv4.
The different structures used into this function are:

- *struct rt_ msghdr*, defined in /usr/include/net/route.h,
- *struct if_ msghdr*, defined in /usr/include/net/if.h,
- *struct ifa_ msghdr*, defined in /usr/include/net/if.h.

Similarly for IPv6, we get the list pointed by ifs2 when get_iflist6() function is called. Everything is same in get_iflist6() function except for the sysctl function parameters,

```
static int mib[6] = { CTL_NET, PF_ROUTE, 0, AF_INET6 , NET_RT_IFLIST, 0 };
```

where AF_INET6 denotes the IPv6 address families,

For link local unicast IPv6 address, kernel doesn't give right address, in the prefix part. Instead of giving fe80::\64, it gives fe80::k\64, where k is the interface index, so this is checked in the function get_iflist6(),

```
0
1       temp1 = (u_char *)&ifs2[naddrs2].addr6;

                if((int)(*temp1) == -2 && (int)(*(temp1+1)) == -128)
                                        *(temp1+3)=0;
```

(signed integral value of '0xfe' is -2 and of '0x80' is -128)

Now in the main function var_inetEntry, we concatane the two lists,

```
0           ifs = malloc(nifs * sizeof(*ifs));
1
            for (i=0 ; i <naddrs1 ; i++)
              ifs[i] = ifs1[i];

5           for (i=naddrs1; i < nifs; i++)
              ifs[i]=ifs2[i-naddrs1];
```

and we construct OID for each element of new concataned list. We go through the whole list until the constructed OID matches with the requested OID (from get request) or constructed OID is appropriate OID (for the get-next request).

```
0           for (i = 0; i < nifs; i++) {
1            // oid is constructed
                      op = &current[10];
              if (i < naddrs1)
                cp = (u_char *)&ifs[i].addr;
5             else {
                cp = (u_char *)&ifs[i].addr6;
                }

              if(i<naddrs1)
10              {
                oid_compare_length=17;
                cp = (u_char *)&ifs[i].addr;
                *op++=ifs[i].index;
                *op++=1;
15              *op++=32;
                *op++ = *cp++;
```

```
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                }
              else
                {
                oid_compare_length=29;
                cp = (u_char *)&ifs[i].addr6;
                *op++=ifs[i].index;
                *op++ = 2;
                *op++ =128;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                }

              if (exact) {
                if (snmp_oid_compare(current, oid_compare_length, name, *length) == 0) {
                  memcpy(lowest, current,oid_compare_length  * sizeof(oid));
                  lowinterface = i;
                  break;    /* no need to search further */
                }
              } else {
                if ((snmp_oid_compare(current, oid_compare_length, name, *length) > 0) &&
                    (lowinterface < 0
                     || (snmp_oid_compare(current,oid_compare_length ,
                                  lowest,oid_compare_length ) < 0))) {
                  /*
                   * if new one is greater than input
                   * and closer to input than previous
                   * lowest, save this one as the "next"
                   * one.
                   */
```

```
                lowinterface = i;
                memcpy(lowest, current, oid_compare_length * sizeof(oid));
65            }
            }

        }

70      if (lowinterface < 0)
                return NULL;

        i = lowinterface;

75      if(ifs[i].addrtype==1)
          oid_compare_length=17;
        else
          oid_compare_length=29;
        memcpy(name, lowest, oid_compare_length * sizeof(oid));
80      *length =oid_compare_length ;

        *write_method = 0;

        *var_len = sizeof(long_return);
```

According to vp->magic, appropriate action is taken.

```
 0      switch (vp->magic) {
 1
        case INETNETTOMEDIANETADDRESSTYPE:
                   long_return = ifs[i].addrtype;
                    return (u_char *)&long_return;
 5
        case INETNETTOMEDIANETADDRESS:
          switch (ifs[i].addrtype){
          case 1:
                free_buf(buf);
10              if (inet_ntop(AF_INET,(void *)&ifs[i].addr, buf, sizeof(buf)) != NULL)
                        *var_len = longueur_buf(buf);
                long_return = buf;
                break;

15        case 2:
                free_buf6(buf6);
                if (inet_ntop(AF_INET6,(void *)&ifs[i].addr6, buf6, sizeof(buf6)) != NULL)
                        *var_len = longueur_buf6(buf6);
                long_return = buf6;
20              break;
```

```
            case 0:
                  long_return = vide;
                  break;
          }
        return (u_char *)long_return;

        case INETNETTOMEDIAPHYSADDRESS:
          if(ifs[i].ifphysflag==0)
            *var_len=0;
          else
            *var_len = sizeof(ifs[i].PhysAddr);
          return (u_char *) (ifs[i].PhysAddr);

        case INETNETTOMEDIATYPE:
             *var_len = sizeof long_return;
             long_return = ifs[i].type;
             return (u_char *) & long_return;


        default:
                DEBUGMSGTL(("snmpd", "unknown sub-id %d in var_ipAddressEntry\n", vp->magic));
        }
        return NULL;
```

For INETNETTOMEDIANETADDRESStype, the RFC 3291 chose to represent IP addresses in the form of character strings. Before, for a variable of the type addresses IPv4, we can assign the type ASN_IPADDRESS to it which will take care of the formatting of the address on the level of the station of management (addition of the character '.' between each 8 bits). But, now that the RFC 3291 decided to have the same convention for IPv4 and IPv6 addresses, the representation of these addresses by character strings is ensured by a call to the function inet_ntop:

```
inet_ntop(AF_INET6,(void *)&ifs[i].addr6, buf6, sizeof(buf6));
```

where buf6 is array of chracters.
Similarly for IPv4, inet_ntop function is called:

```
inet_ntop(AF_INET,(void *)&ifs[i].addr, buf, sizeof(buf));
```

In the case INETNETTOMEDIAPHYSADDRESS, *ifphysflag* denotes whether it is loop back address or not, for which a string of zero length is returned.

If no case matches, NULL is returned.

6.3.7 Implementation of the ipv6ScopeIdTable

As only link-local addresses were defined, we were only concerned to return the scope Identifier for the link-local address, which is the interface index of the interface in the FreeBSD implementation. All other Scope Id's are returned with the default value, i.e. zero.

The associated code could be found in Appendix B.3.

6.3.8 Implementation of the ICMP Group

The Internet Protocol [IP] is not designed to be absolutely reliable. So was defined the Internet Control Message Protocol (ICMP) which purpose is to provide feedback about problems in the communication environment, not to make IP reliable. There are still no guarantees that a datagram will be delivered or a control message will be returned. Some datagrams may still be undelivered without any report of their loss. The higher level protocols that use IP (the transport layer, if TCP, or the application, if UDP) must implement their own reliability procedures if reliable communication is required.

The ICMP messages typically report errors in the processing of datagrams. To avoid the infinite regress of messages about messages etc., no ICMP messages are sent about ICMP messages.

The information defined into the ICMP Group of the previous MIB II are now split into 2 tables, the inetIcmpTable, and the inetIcmpMsgTable.

Implementation of the inetIcmpTable The inetIcmpTable contains the generic counters, that is the total number of ICMP messages received with or without errors in them, and the total number of ICMP messages received without errors or not sent because they were owning an error. For each interface, it was not possible to get the information about ICMP messages for the current version of FreeBSD, (operation was not supported), so we used value of inetIcmpIfIndex zero indicating system wide values.

To gather the information from the kernel, we used two structures struct icmpstat for IPv4, and struct icmp6stat, for IPv6. They are defined into /usr/include/netinet/icmp_var.h for IPv4, and into /usr/include/netinet/icmp6.h for IPv6.

For IPv4, the parameters of the sysctl function were as defined into the sysctl-man page:

```
len = sizeof(struct icmpstat);
sname[0]=CTL_NET;
sname[1]=AF_INET;
sname[2]=IPPROTO_ICMP;
sname[3]=ICMPCTL_STATS;
ret_value = sysctl(sname, 4,&icmpstat, &len, 0, 0);
```

Our interest is in the networking subsystem, so the first element of array sname is CTL_NET (defined in /usr/include/sys/sysctl.h). We have to get information about struct icmpstat, that is we are getting information related to the Internet Protocols, so second variable is AF_INET and the third element(a protocol number) is IPPROTO_ICMP (defined in /usr/include/netinet/in.h). The fourth one indicating that information has to be stored in struct icmpstat (defined in /usr/include/netinet/icmp_var.h) which is of type struct icmpstat. If the sysctl function is successful, all the informations required is stored in structure icmpstat.

For IPv6, for calling sysctl function, the parameters are

```
    sname[0]=CTL_NET; /* defined in /usr/include/sys/sysctl.h */
    sname[1]=AF_INET6; /* defined in /usr/include/sys/socket.h */
    sname[2]=IPPROTO_ICMPV6; /* defined in /usr/include/netinet/in.h */
    sname[3]=ICMPV6CTL_STATS; /* defined in /usr/include/netinet/icmp6.h */
    len = sizeof(struct icmp6stat);
    ret_value = sysctl(sname, 4, &icmp6stat, &len, 0, 0);
```

The elements of array sname are different from that in IPv4 in the respect that, 2nd field is indicating Internet protocol version 6, and third element indicates ICMP6 protocol, fourth element show the structure struct icmp6stat, which is of type struct icmp6stat. Similarly for this, if the sysctl function is successful, all the informations required are stored in structure icmp6stat.

Depending of the OID required, a magic number is defined into the variable list. For the IPv4-only MIB II, the variable list associated with the objects that will be defined into the new inetIcmpTable, is defined into the icmp.c file as follow:

```
struct variable1 icmp_variables[] = {
    {ICMPINMSGS, ASN_COUNTER, RONLY, var_icmp, 1, {1}},
    {ICMPINERRORS, ASN_COUNTER, RONLY, var_icmp, 1, {2}},
...
    {ICMPOUTMSGS, ASN_COUNTER, RONLY, var_icmp, 1, {14}},
    {ICMPOUTERRORS, ASN_COUNTER, RONLY, var_icmp, 1, {15}},
                ...
};
```

For the "unified" MIB II that we want to implement, the variable list defined for the inetIcmpTable, into the draft_RFC2011.c file, is:

```
struct variable4 icmp_variables[] = {
    {ICMPINMSGS, ASN_COUNTER, RONLY, var_icmp, 3, {27,1,3}},
    {ICMPINERRORS, ASN_COUNTER, RONLY, var_icmp, 3, {27,1,4}},
    {ICMPOUTMSGS, ASN_COUNTER, RONLY, var_icmp, 3, {27,1,5}},
    {ICMPOUTERRORS, ASN_COUNTER, RONLY, var_icmp, 3, {27,1,6}},
};
```

For each magic number (defined into the first column of the list), the source code is redefined, to return the appropriate value to the manager. For example, for the icmpInMsg object of the inetIcmpTable:

```
       case ICMPINMSGS:
      if(j==0)   // that is, this case holds for ipv4
{
        long_return = icmpstat.icps_badcode +
             icmpstat.icps_tooshort +
            icmpstat.icps_checksum + icmpstat.icps_badlen;
         for (i = 0; i <= ICMP_MAXTYPE; i++)
            long_return += icmpstat.icps_inhist[i];
}
```

```
      else
{ // otherwise, for ipv6
long_return = icmp6stat.icp6s_badcode +
        icmp6stat.icp6s_tooshort +
         icmp6stat.icp6s_checksum + icmp6stat.icp6s_badlen;
for (i = 0; i <= 255; i++)
long_return += icmp6stat.icp6s_inhist[i];
}
      return (u_char *) & long_return;
```

Implementation of the inetIcmpMsgTable The Internet Control Message Protocol has many messages that are identified by a "type" field, defined into the RFC 2463 ([CD98]). Many of the types of ICMP message are now obsolete and are no longer seen in the Internet. Some important ones which are widely used include: Echo Reply (0), Echo Request (8), Redirect (5), Destination Unreachable (3), Traceroute (30), Time Exceeded (11). The full list is shown below:

Type	Name	Reference
1	Destination Unreachable	RFC 2463 [CD98]
2	Packet Too Big	RFC 2463
3	Time Exceeded	RFC 2463
4	Parameter Problem	RFC 2463
128	Echo Request	RFC 2463
129	Echo Reply	RFC 2463
130	Multicast Listener Query	MC-LIST [VCF+02]
131	Multicast Listener Report	MC-LIST
132	Multicast Listener Done	MC-LIST
133	Router Solicitation	RFC 2461 [NNS98]
134	Router Advertisement	RFC 2461
135	Neighbor Solicitation	RFC 2461
136	Neighbor Advertisement	RFC 2461
137	Redirect Message	RFC 2461
138	Router Renumbering	Crawford [Cra02]
139	ICMP Node Information Query	Crawford
140	ICMP Node Information Response	Crawford
141	Inverse Neighbor Discovery Solicitation Message	RFC 3122 [Con01]
142	Inverse Neighbor Discovery Advertisement Message	RFC 3122

Many of these ICMP types have a "code" field that we list here:

Type	Code	Name	Reference
1		Destination Unreachable	RFC 2463 [CD98]
	0	no route to destination	
	1	communication with destination administratively prohibited	
	2	(not assigned)	
	3	address unreachable	
	4	port unreachable	
2		Packet Too Big	RFC 2463
	0		
3		Time Exceeded	RFC 2463
	0	hop limit exceeded in transit	
	1	fragment reassembly time exceeded	
4		Parameter Problem	RFC 2463
	0	erroneous header field encountered	
	1	unrecognized Next Header type encountered	
	2	unrecognized IPv6 option encountered	
128		Echo Request	RFC 2463
	0		
129		Echo Reply	RFC 2463
	0		
130		Multicast Listener Query	MC-LIST [VCF+02]
	0		
131		Multicast Listener Report	MC-LIST
	0		
132		Multicast Listener Done	MC-LIST
	0		
133		Router Solicitation	RFC 2461 [NNS98]
	0		
134		Router Advertisement	RFC 2461
	0		
135		Neighbor Solicitation	RFC 2461
	0		
136		Neighbor Advertisement	RFC 2461
	0		
137		Redirect Message	RFC 2461
	0		
138		Router Renumbering	Crawford [Cra02]
	0	Router Renumbering Command	
	1	Router Renumbering Result	
	255	Sequence Number Reset	
141		Inverse Neighbor Discovery Solicitation Message	RFC 3122 [Con01]
	0		
142		Inverse Neighbor Discovery Advertisement Message	RFC 3122
	0		

This inetIcmpMsgTable contains the statistics of each type of the above message that were

sent by each interface or by the wide system.
The structures used and the parameters used in sysctl function call for implementing this table are same as discussed for the inetIcmpTable.

For this table also, the value of inetIcmpMsgIfIndex is chosen as zero indicating system wide values.

The values of type and code, which we are implementing is shown below. If no code is specified for a type, then the value 256 is used for that type as written in MIB.

Type	Code
unreachable & no route	administratively prohibited beyond scope address unreachable port unreachable
packet too big	
time exceed	time exceed transit time exceed reassembly
parameter problem	erroneous header field unrecognised next header unrecognised option
multicast listener report	
multicast listener done	
router advertisement	
neighbour solicitations	
neighbour advertisement	
redirect	

Whatever information could be taken by the system, we have used that for implementing the table. For example, for IPv6, it was possible to get information about each code for type 0, but for IPv4, we could get information for the type only (not specified any code), as struct icmp6stat contains those variables required, but struct icmpstat doesn't contain required variables. In such cases, we have used value of code 256 as specified in MIB. For example, for inetIcmpInMsgs:

```
// for ipv4
```

```
if(j==0 && name[12]==1 && name[13]==256)
return (u_char *) &  icmpstat.icps_inhist[ICMP_UNREACH];
if(j==0 && name[12]==3 && name[13]==256)
return (u_char *) & icmpstat.icps_inhist[ICMP_TIMXCEED];

if(j==0 && name[12]==4 && name[13]==256)
return (u_char *) &  icmpstat.icps_inhist[ICMP_PARAMPROB];
if(j==0 && name[12]==134 && name[13]==0)
return (u_char *) &  icmpstat.icps_inhist[ICMP_ROUTERADVERT];
if(j==0 && name[12]==137 && name[13]==0)
return (u_char *) & icmpstat.icps_inhist[ICMP_REDIRECT];


// for ipv6

if(j==1 && name[12]==1 && name[13]==256)
return (u_char *) &  icmp6stat.icp6s_inhist[ICMP6_DST_UNREACH];
if(j==1 && name[12]==2 && name[13]==0)
return (u_char *) &  icmp6stat.icp6s_inhist[ICMP6_PACKET_TOO_BIG];

if(j==1 && name[12]==3 && name[13]==256)
return (u_char *) &  icmp6stat.icp6s_inhist[ICMP6_TIME_EXCEEDED];
if(j==1 && name[12]==4 && name[13]==256)
return (u_char *) &  icmp6stat.icp6s_inhist[ICMP6_PARAM_PROB];

if(j==1 && name[12]==131 && name[13]==0)
return (u_char *) &  icmp6stat.icp6s_inhist[MLD6_LISTENER_REPORT];
if(j==1 && name[12]==132 && name[13]==0)
return (u_char *) &  icmp6stat.icp6s_inhist[MLD6_LISTENER_DONE];

if(j==1 && name[12]==134 && name[13]==0)
return (u_char *) &  icmp6stat.icp6s_inhist[ND_ROUTER_ADVERT];
if(j==1 && name[12]==135 && name[13]==0)
return (u_char *) &  icmp6stat.icp6s_inhist[ND_NEIGHBOR_SOLICIT];

if(j==1 && name[12]==136 && name[13]==0)
return (u_char *) &  icmp6stat.icp6s_inhist[ND_NEIGHBOR_ADVERT];
if(j==1 && name[12]==137 && name[13]==0)
return (u_char *) &  icmp6stat.icp6s_inhist[ND_REDIRECT];
```

6.4 Implementation of the draft-ietf-ipngwg-rfc2012-update01.txt

For implementing the draft-ietf-ipngwg-rfc2012-update01.txt for both IPv4 and IPv6, we need to get information for both IPv4 and IPv6 from the kernel. But the current version of FreeBSD doesn't allow to obtain information about TCP for IPv6. This has been, and

could be, verified by using the sysctl function, the netstat function or the ifconfig function. So, we could not fully implement this MIB. At most, we were able to change the format of output for IPv4 for this new TCP-MIB.

6.5 Implementation of the draft-ietf-ipngwg-rfc2013-update01.txt

The problem was similar here as with the previous draft. So, we could not fully implement this MIB. For this also, we were only able to change the format of output for IPv4 for the new UDP-MIB.

6.6 Implementation of the draft-ietf-ipngwg-rfc2096-update01.txt

This implementation is defined into the draft_RFC2096.c file and its corresponding .h file. The main structure used was:

```
struct snmprt {
TAILQ_ENTRY(snmprt) link;
struct rt_msghdr *hdr;
struct in_addr  dest ;
struct in_addr gateway;
struct in_addr netmask;
int index;
struct in_addr ifa;
int  type;
struct in6_addr dest6;
struct in6_addr gateway6;
struct in6_addr netmask6;
struct in6_addr ifa6;
};
```

The *dest* field is of type *struct in_addr* for IPv4 address, same is for fields *gateway*, *netmask* and *ifa*. For IPv6, similar structure is *struct in6_addr*. *dest6* (destination IPv6 address, *gateway6*, *netmask6* and *ifa6* are of this structure type).
The *index* field is for the interface index.
The *type* field is set to 1 if this structure contains information for IPv4, 2, if it contains information for IPv6.
The *link* field is for linked list of this structure.

The main function is *var_ipForwardEntry*, where all the information of the routing table is obtained from the kernel, and parsed into a linked list *rthead*, of structure type *struct snmprt*. This is done by the function *suck_rt*.

```
0   u_char         *
1   var_ipForwardEntry(struct variable * vp,
                       oid * name,
                       size_t * length,
                       int exact, size_t * var_len, WriteMethod ** write_method)
```

```
{
    /*
     * object identifier is of form:
     * 1.3.6.1.2.1.4.21.1.1.A.B.C.D,  where A.B.C.D is IP address.
     * IPADDR starts at offset 10.
     */
    int             Save_Valid, result;
    u_char         *cp;
    oid            *op;
    struct snmprt  *rt;
    static struct snmprt *savert;
    static int      saveNameLen, saveExact;
    static oid      saveName[25], Current[49],lowest[49];
    int type=1;
    int oid_compare_length;
    int lowinterface;
    int nooid;

    lowinterface=-1;


        /*
         * fill in object part of name for current
         * (less sizeof instance part)
         */

        memcpy(Current, vp->name, (int) (vp->namelen) * sizeof(oid));

        suck_krt(0);
```

After that call, the list is scanned for each element of the linked list, OID is constructed.

```
        for (rt = rthead.tqh_first; rt; rt = rt->link.tqe_next) {
            op = Current + 11;
            if(rt->type==1)
              oid_compare_length=25;
            else
              oid_compare_length=49;
            if(rt->type==1)
              {
            cp = (u_char *) & rt->dest;
            *op++=1;
            *op++=1;
            *op++=32;
            *op++ = *cp++;
            *op++ = *cp++;
            *op++ = *cp++;
```

```
                *op++ = *cp++;
50              *op++=calcul_prefix(&rt->netmask,sizeof(struct in_addr));
                *op++=1;
                 if (rt->gateway.s_addr == 0 && rt->ifa.s_addr == 0)
                   {
                     *op++=32;
                      *op++ = 0;
55                    *op++ = 0;
                      *op++ = 0;
                      *op++ = 0;
                   }
60               else
                   {
                     if (rt->gateway.s_addr == 0)
                       cp=(u_char *)& rt->ifa;
                     else
65                     cp = (u_char *) & rt->gateway;
                     *op++=32;
                     *op++ = *cp++;
                     *op++ = *cp++;
                     *op++ = *cp++;
70                   *op++ = *cp++;
                   }
                  }
                else
                  {
75              cp = (u_char *) & rt->dest6;
                *op++=1;
                *op++=2;
                *op++=128;
                *op++ = *cp++;
80              *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
85              *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
90              *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
```

```
95              *op++=calcul_prefix(&rt->netmask6,sizeof(struct in6_addr));
                *op++=2;
                 if (check_all_zero(&rt->gateway6,sizeof(struct in6_addr)) == 1
                    &&
                    check_all_zero(&rt->ifa6,sizeof(struct in6_addr)) == 1
100                 )
                   {
                     *op++=128;
                      *op++ = 0;
                      *op++ = 0;
105                   *op++ = 0;
                      *op++ = 0;
                      *op++ = 0;
                      *op++ = 0;
                      *op++ = 0;
110                   *op++ = 0;
                      *op++ = 0;
                      *op++ = 0;
                      *op++ = 0;
                      *op++ = 0;
115                   *op++ = 0;
                      *op++ = 0;
                      *op++ = 0;
                   }
120                else
                   {
                     if (check_all_zero(&rt->gateway6,sizeof(struct in6_addr)) == 1)
                       cp=(u_char *)& rt->ifa6;
                     else
125                    cp = (u_char *) & rt->gateway6;
                     *op++=128;
                     *op++ = *cp++;
                     *op++ = *cp++;
                     *op++ = *cp++;
130                  *op++ = *cp++;
                     *op++ = *cp++;
                     *op++ = *cp++;
                     *op++ = *cp++;
                     *op++ = *cp++;
135                  *op++ = *cp++;
                     *op++ = *cp++;
                     *op++ = *cp++;
                     *op++ = *cp++;
                     *op++ = *cp++;
140                  *op++ = *cp++;
```

```
                    *op++ = *cp++;
                    *op++ = *cp++;
                  }
                 }
145               if(rt->type==1)
                   oid_compare_length=25;
                 else
                   oid_compare_length=49;

150              result = snmp_oid_compare(Current,oid_compare_length,  name,*length);

                   if ((exact && (result == 0))
                     || (!exact && (result > 0)))
                     break;
155          }
             if (rt == NULL)
                 return NULL;

             if(rt->type==1)
160                oid_compare_length=25;
             else
                   oid_compare_length=49;
             memcpy(name, Current, oid_compare_length * sizeof(oid));
             *length = oid_compare_length;
165
```

If this OID matches the requested OID (snmpget) or its appropriate next OID (snmpget-next), then this element contains all the required information. The appropriate value is returned for the requested *magic* field in the structure pointed by *vp*, one of the input parameter of the var_ipForwardEntry):

```
   *write_method = write_rte;
       *var_len = sizeof long_return;
       switch (vp->magic) {

170    case INETCIDROUTEINSTANCE: long_return = 1;
          return (u_char *) & long_return;

       case INETCIDROUTEDESTTYPE: long_return = rt->type;
          return (u_char *) & long_return;
175
       case INETCIDROUTEDEST:
          switch (rt->type){
             case 1:
                             free_buf(buf);
180
                             if (inet_ntop(AF_INET,(void *)&rt->dest, buf, sizeof(buf)) != NULL)
```

```
                                *var_len = (size_t)longueur_buf(buf);

185                             long_return = (long int)buf;

                                 break;

              case 2:
190                            free_buf6(buf6);
                               if (inet_ntop(AF_INET6,(void *)&rt->dest6, buf6, sizeof(buf6)) != NULL)

                               *var_len = (size_t)longueur_buf6(buf6);

195                             long_return = (long int)buf6;

                                break;
              }

200           return (u_char *)long_return;

          case INETCIDROUTEIFINDEX:
              long_return = rt->index;
              return (u_char *) & long_return;
205
          case INETCIDROUTEMETRIC1:
                long_return = rt->hdr->rtm_rmx.rmx_hopcount + 1;
                return (u_char *) & long_return;

210       case INETCIDROUTEMETRIC2:
            long_return=-1;
              return (u_char *) & long_return;
```

The whole source file could be read into the Appendix B.4.

The *suck_rt* function is used to obtain information about the routing table. Sysctl function is called twice, first to estimate the size of the buffer and secondly to store the information in buffer:

```
0   static int suck_krt(int force)
1   {
        time_t          now,now1;
        struct snmprt  *rt, *next,*temp;
        size_t          len;
5       static int      name[6] =
            { CTL_NET, PF_ROUTE, 0, AF_INET, NET_RT_DUMP, 0 };
        char           *cp;
        struct rt_msghdr *rtm;
```

```
10      time(&now);
        if (now < (lasttime + CACHE_TIME) && !force)
            return 0;
            lasttime = now;

15      for (rt = rthead.tqh_first; rt; rt = next) {
            next = rt->link.tqe_next;
            free(rt);
        }
        TAILQ_INIT(&rthead);
20
        if (sysctl(name, 6, 0, &len, 0, 0) < 0) {
            syslog(LOG_WARNING, "sysctl net-route-dump: %m");
            return -1;
        }
25
        free(rtbuf);
        rtbuf=malloc(len);

        if (sysctl(name, 6, rtbuf, &len, 0, 0) < 0) {
30          syslog(LOG_WARNING, "sysctl net-route-dump: %m");
            return -1;
            }
```

The 5th argument (NET_RT_DUMP), indicates that the Routing Table is asked and is required for all the interfaces. The buffer returned contains a variable number of *rt_msghdr* structures of *rtm_type* RTM_GET with each rt_msghdr structure followed by up to four socket address structures: the destination, the gateway, network mask, and cloning mask of the routing table entry. Each rt_msghdr structure followed by up to four socket address structure is parsed. From this, a variable of type struct snmprt is constructed and inserted in the linked list pointed by rthead. This is done by calling the rtmsg() function.
For IPv6, instead of AF_INET in the fourth element of the array name, AF_INET6 is used, specifying IPv6 address family. And in a similar way, sysctl function is called and a variable of type struct snmprt is constructed and inserted into the linked list by the call of the rtmsg6() function:

```
        cp = rtbuf;
        while (cp < rtbuf + len) {
35          rtm = (struct rt_msghdr *) cp;
            /*
             * NB:
             * You might want to exclude routes with RTF_WASCLONED
             * set.  This keeps the cloned host routes (and thus also
40           * ARP entries) out of the routing table.  Thus, it also
             * presents management stations with an incomplete view.
             * I believe that it should be possible for a management
             * station to examine (and perhaps delete) such routes.
             */
```

```
45          if (rtm->rtm_version == RTM_VERSION && rtm->rtm_type == RTM_GET)
                rtmsg(rtm);
            cp += rtm->rtm_msglen;
        }

50      force=0;
           time(&now1);
        if (now1 < (lasttime2 + CACHE_TIME) && !force)
            return 0;
            lasttime2 = now1;
55


          for (temp = rthead.tqh_first; temp; temp = next) {
            next = temp->link.tqe_next;
            }
60
        name[3]=AF_INET6;

        if (sysctl(name, 6, 0, &len, 0, 0) < 0) {
            syslog(LOG_WARNING, "sysctl net-route-dump: %m");
65          return -1;
        }

        free(rtbuf2);
        rtbuf2=malloc(len);
70
        if (sysctl(name, 6, rtbuf2, &len, 0, 0) < 0) {
            syslog(LOG_WARNING, "sysctl net-route-dump: %m");
            return -1;
        }
75
        cp = rtbuf2;
        while (cp < rtbuf2 + len) {
            rtm = (struct rt_msghdr *) cp;

80          if (rtm->rtm_version == RTM_VERSION && rtm->rtm_type == RTM_GET)
                rtmsg6(rtm);
            cp += rtm->rtm_msglen;
        }

85         for (temp = rthead.tqh_first; temp; temp = next) {
            next = temp->link.tqe_next;
        }
        return 0;
    }
```

The *rtmsg()* function needs one parameter which is a pointer to a *rt_ msghdr* structure. This structure's field *rtm_ addrs* is a bitmask defining which of the eight possible socket address structures follow the rt_msghdr structure. The meaning of each bitmask is the following:

Bitmask Constant	Value	Socket address structure
RTA_DST	0x01	destination address
RTA_GATEWAY	0x02	gateway address
RTA_NETMASK	0x04	network mask
RTA_GENMASK	0x08	cloning mask
RTA_IFP	0x10	interface name
RTA_IFA	0x20	interface address
RTA_AUTHOR	0x40	author of redirect
RTA_BRD	0x80	broadcast or point to point destination address

These constants are defined in /usr/include/net/route.h.
This structure's field *rtm_ msglen* is the size of struct rt_msghdr plus the size of following structures. All the structures following the structure rt_msghdr are extracted in this function and from them a variable of type *struct snmprt* is constructed which is inserted into the linked list pointed by *rthead*.

```
0   static void rtmsg(struct rt_msghdr *rtm)
1   {
        struct snmprt  *rt;
        struct sockaddr *sa;

5       int  bit, gotdest, gotmask;

        rt = malloc(sizeof *rt);
        if (rt == 0)
            return;
10      rt->hdr = rtm;
        rt->ifa.s_addr = 0;
        rt->dest = rt->gateway = rt->netmask = rt->ifa;
        rt->index = rtm->rtm_index;
        rt->type=1;
15      gotdest = gotmask = 0;
        sa = (struct sockaddr *)(rtm + 1);

        for (bit = 1; ((char *) sa < (char *) rtm + rtm->rtm_msglen) && bit;
             bit <<= 1) {
20          if ((rtm->rtm_addrs & bit) == 0)
                continue;
```

```
            switch (bit) {
            case RTA_DST:
#define satosin(sa) ((struct sockaddr_in *)(sa))
                rt->dest = satosin(sa)->sin_addr;
                gotdest = 1;
                break;
            case RTA_GATEWAY:
                if (sa->sa_family == AF_INET)
                    rt->gateway = satosin(sa)->sin_addr;
                break;
            case RTA_NETMASK:
                if (sa->sa_len >= offsetof(struct sockaddr_in, sin_addr))
                                rt->netmask = satosin(sa)->sin_addr;
                gotmask = 1;
                break;
            case RTA_IFA:
                if (sa->sa_family == AF_INET)
                    rt->ifa = satosin(sa)->sin_addr;
                break;
            }

#define ROUNDUP(a) \
            ((a) > 0 ? (1 + (((a) - 1) | (sizeof(long) - 1))) : sizeof(long))
            sa = (struct sockaddr *) ((char *) sa + ROUNDUP(sa->sa_len));
        }
        if (!gotdest) {
            /*
             * XXX can't happen if code above is correct
             */
            snmp_log(LOG_ERR, "route no dest?\n");
            free(rt);
        } else {
            /*
             * If no mask provided, it was a host route.
             */
            if (!gotmask)
                rt->netmask.s_addr = ~0;

                    //printf("rt->type for 4 for rtmsg function: %d, %X \n",rt->type,rt);
            TAILQ_INSERT_TAIL(&rthead, rt, link);
        }
    }
```

A function called rtmsg6() is defined for IPv6. It is similar as rtmsg().

6.7 Integration into the net-snmp-5.0.3 module

Now that all the source code files were implemented and tested, we have to integrate those modifications into the net-snmp current package. At the time of that work, the current one was net-snmp-5.0.3. To include a new MIB, the simplest way is to used the "with-mib-modules" option. This suppose that each module is implemented with a .c file named as the module. So, to include the DRAFT_RFC2011 module, the main file should be named: draft_RFC2011.c and its .h file : draft_RFC2011.h.

To include the draft modifying the RFC2011, we had to take care about the ICMP module. In the implementation of the IPv4-only MIB II, RFC2011 is implemented with the ip.c and ip.h files for the IP Group, and with the icmp.c and icmp.h files, for the ICMP Group. In the "unified" MIB II, the ICMP Group is defined into the draft_2011.c and draft_2011.h files, and not into the icmp.c and .h files. As the icmp.c file is linked by default with the agent into the net-snmp package, we must specify during the configuration phase, that, when the draft_RFC2011 is required, those files should not be linked. That is why we had to modify the configuration file *configure.in*. We added those instructions:

```
dnl
dnl Check if the draft-RFC2011 module is wanted
dnl

case $with_mib_modules in
  *draft_RFC2011*)
     AC_MSG_RESULT(DRAFT_RFC2011 is wanted so ICMP is omitted)
with_out_mib_modules="mibII/icmp $with_out_mib_modules";;
esac
```

The DRAFT-RFC2011.txt was implemented using multiple files : draft_RFC2011.c, ipAddress.c, ipAddressPrefix.c, ipv4If.c, inetEntry.c, ipv6Interface.c, ipIfStats.c, ip.c, ipv6ScopeId.c. To oblige the configuration file to load all these files when this MIB module is required, we had to add into the draft_RFC2011.h file, the following command:

```
config_require(mibII/ipv4If mibII/ipAddressPrefix mibII/ipv6Interface mibII/ipIfStats
mibII/ipAddress mibII/ipv6ScopeId mibII/inetEntry)
```

A Text of the implemented implemented

These drafts, as they were "on-work" files, could not be found yet on the IETF site. That is why we have included them here.

A.1 draft-ietf-ipngwg-RFC2011-update-00

```
IP-MIB DEFINITIONS ::= BEGIN

IMPORTS

    MODULE-IDENTITY, OBJECT-TYPE,
    Integer32, Counter32, IpAddress, mib-2, Unsigned32, Counter64
                                        FROM SNMPv2-SMI
    PhysAddress, TruthValue, TimeStamp, RowPointer,
    TEXTUAL-CONVENTION -- XXX
                                        FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP     FROM SNMPv2-CONF
    InetAddress, InetAddressType,
    InetAddressPrefixLength             FROM INET-ADDRESS-MIB
    InterfaceIndex,
    InterfaceIndexOrZero, ifIndex       FROM IF-MIB;


ipMIB MODULE-IDENTITY
    LAST-UPDATED "200107130000Z"
    ORGANIZATION "IETF IPv6 MIB Revision Team"
    CONTACT-INFO
           "Editor:
            Bill Fenner
            AT&T Labs - Research
            75 Willow Rd
            Menlo Park, CA

            Phone: +1 650 330-7893
            Email: <fenner@research.att.com>"
    DESCRIPTION
           "The MIB module for managing IP and ICMP implementations, but
            excluding their management of IP routes."
    REVISION      "200107130000Z"
    DESCRIPTION
           "IP version neutral revision, published as RFC XXXX."
    REVISION      "9411010000Z"
    DESCRIPTION
           "Published seperately as RFC 2011."
    REVISION      "9103310000Z"
    DESCRIPTION
           "The initial revision of this MIB module was part of MIB-II."
    ::= { mib-2 48}

-- the IP general group

ip       OBJECT IDENTIFIER ::= { mib-2 4 }

ipForwarding OBJECT-TYPE
    SYNTAX     INTEGER {
                   forwarding(1),    -- acting as a router
                   notForwarding(2)  -- NOT acting as a router
               }

    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
           "The indication of whether this entity is acting as an IPv4
            router in respect to the forwarding of datagrams received
            by, but not addressed to, this entity.  IPv4 routers forward
            datagrams.  IPv4 hosts do not (except those source-routed
            via the host)."
    ::= { ip 1 }

ipDefaultTTL OBJECT-TYPE
    SYNTAX     INTEGER (1..255)
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
           "The default value inserted into the Time-To-Live field of
            the IPv4 header of datagrams originated at this entity,
            whenever a TTL value is not supplied by the transport layer
```

```
            protocol."
    ::= { ip 2 }

ipReasmTimeout OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The maximum number of seconds which received fragments are
            held while they are awaiting reassembly at this entity."
    ::= { ip 13 }

-- the IPv6 general group

ipv6MIB OBJECT IDENTIFIER    ::= { mib-2 55 }

ipv6MIBObjects OBJECT IDENTIFIER   ::= { ipv6MIB 1 }

ipv6Forwarding OBJECT-TYPE
    SYNTAX     INTEGER {
                    forwarding(1),    -- acting as a router
                    notForwarding(2)  -- NOT acting as a router
               }
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
           "The indication of whether this entity is acting as an IPv6
            router in respect to the forwarding of datagrams received
            by, but not addressed to, this entity.  IPv6 routers forward
            datagrams.  IPv6 hosts do not (except those source-routed
            via the host)."
    ::= { ipv6MIBObjects 1 }

ipv6DefaultHopLimit OBJECT-TYPE
    SYNTAX     INTEGER (0..255)
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
           "The default value inserted into the Hop Limit field of the
            IPv6 header of datagrams originated at this entity, whenever
            a Hop Limit value is not supplied by the transport layer
            protocol."
    ::= { ipv6MIBObjects 2 }

--
-- IPv4 Interface Table
--

ipv4IfTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF Ipv4IfEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The table containing per-interface IPv4-specific
            information."
    ::= { ip 25 }

ipv4IfEntry OBJECT-TYPE
    SYNTAX     Ipv4IfEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "An entry containing IPv4-specific information for a specific
            interface."
    INDEX { ipv4IfIndex }
    ::= { ipv4IfTable 1 }

Ipv4IfEntry ::= SEQUENCE {
       ipv4IfIndex         InterfaceIndex,
       ipv4IfReasmMaxSize  Integer32
    }

ipv4IfIndex OBJECT-TYPE
    SYNTAX     InterfaceIndex
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The interface to which these values apply."

    ::= { ipv4IfEntry 1 }

ipv4IfReasmMaxSize OBJECT-TYPE
    SYNTAX     Integer32 (0..65535)
    MAX-ACCESS read-only
```

```
    STATUS     current
    DESCRIPTION
           "The size of the largest IPv4 datagram which this entity can
            re-assemble from incoming IPv4 fragmented datagrams received
            on this interface."
    ::= { ipv4IfEntry 2 }



--
-- v6 interface table
-- XXX I suspect that most of these objects can go away.
--
-- Open Issues:
--  ipv6InterfaceAdminStatus: does it make sense to enable/disable
--   IPv6 on its own on the interface?
--  ipv6InterfaceOperStatus: other than the above, noIfIdentifier(3)
--   is this one's only useful state, which can be determined from
--   the Address table if DAD failed or there is no v6 address on
--   this interface. [not efficiently, though]

ipv6InterfaceTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF Ipv6InterfaceEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The table containing per-interface IPv6-specific
            information."
    ::= { ip 31 }

ipv6InterfaceEntry OBJECT-TYPE
    SYNTAX     Ipv6InterfaceEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "An entry containing IPv6-specific information for a given
            interface."
    INDEX { ipv6InterfaceIfIndex }
    ::= { ipv6InterfaceTable 1 }

Ipv6InterfaceEntry ::= SEQUENCE {
        ipv6InterfaceIfIndex           InterfaceIndex,
        ipv6InterfaceEffectiveMtu      Unsigned32,
        ipv6InterfaceReasmMaxSize      Unsigned32,
        ipv6InterfaceIdentifier        Ipv6AddressIfIdentifier,
        ipv6InterfaceIdentifierLength  INTEGER,
        ipv6InterfacePhysicalAddress   PhysAddress
    }

ipv6InterfaceIfIndex OBJECT-TYPE
    SYNTAX     InterfaceIndex
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The interface for which this row contains IPv6-specific
            information."
    ::= { ipv6InterfaceEntry 1 }

ipv6InterfaceEffectiveMtu OBJECT-TYPE
    SYNTAX     Unsigned32
    UNITS       "octets"
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The size of the largest IPv6 packet which can be
            sent/received on the interface, specified in octets.

            XXX - why isn't this ifMtu - sizeof(ipv6 header)?"
    ::= { ipv6InterfaceEntry 2 }

ipv6InterfaceReasmMaxSize OBJECT-TYPE
    SYNTAX     Unsigned32 (0..65535)
    UNITS       "octets"
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The size of the largest IPv6 datagram which this entity can
            re-assemble from incoming IPv6 fragmented datagrams received
            on this interface."
    ::= { ipv6InterfaceEntry 3 }

-- XXX ugh: I want to get rid of this, which is why it's in the middle
-- of nowhere
Ipv6AddressIfIdentifier ::= TEXTUAL-CONVENTION
```

```
     DISPLAY-HINT "2x:"
     STATUS       current
     DESCRIPTION
       "This data type is used to model IPv6 address
       interface identifiers. This is a binary string
        of up to 8 octets in network byte-order."
     SYNTAX      OCTET STRING (SIZE (0..8))


ipv6InterfaceIdentifier OBJECT-TYPE
    SYNTAX     Ipv6AddressIfIdentifier
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
           "The Interface Identifier for this interface that is (at
            least) unique on the link this interface is attached to. The
            Interface Identifier is combined with an address prefix to
            form an interface address.

            By default, the Interface Identifier is autoconfigured
            according to the rules of the link type this interface is
            attached to.

            XXX - is this an EUI64 that belongs more in the IF-MIB?"
    ::= { ipv6InterfaceEntry 4 }

ipv6InterfaceIdentifierLength OBJECT-TYPE
    SYNTAX     INTEGER (0..64)
    UNITS       "bits"
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
           "The length of the Interface Identifier in bits."
    ::= { ipv6InterfaceEntry 5 }

ipv6InterfacePhysicalAddress OBJECT-TYPE
    SYNTAX     PhysAddress
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The interface's physical address. For example, for an IPv6
            interface attached to an 802.x link, this object normally
            contains a MAC address. Note that in some cases this address
            may differ from the address of the interface's protocol sub-
            layer.  The interface's media-specific MIB must define the
            bit and byte ordering and the format of the value of this
            object. For interfaces which do not have such an address
            (e.g., a serial line), this object should contain an octet
            string of zero length.

            XXX When can this be different from the address of the
            interface's protocol sub-layer, and why?"
    ::= { ipv6InterfaceEntry 6 }


--
-- Per-Interface or System-Wide IP statistics.
--
-- Open issues:
--  Add octet counters similar to ifTable and ifXTable?

ipIfStatsTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF IpIfStatsEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The table containing traffic statistics.  These statistics
            may be kept per-interface and/or system-wide."
    ::= { ip 26 }

ipIfStatsEntry OBJECT-TYPE
    SYNTAX     IpIfStatsEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "An interface statistics entry containing objects for a
            particular interface, or system-wide.

            A row with an ipIfStatsIfIndex value of zero indicates a
            system-wide value; a row with a non-zero ipIfStatsIfIndex
            indicates an interface-specific value.  A system may provide
            both system-wide and interface-specific values, in which
            case it is important to note that the system-wide value may
            not be equal to the sum of the interface-specific values
```

```
            across all interfaces due to e.g. dynamic interface
            creation/deletion."
    INDEX { ipIfStatsAFType, ipIfStatsIfIndex }
    ::= { ipIfStatsTable 1 }

IpIfStatsEntry ::= SEQUENCE {
        ipIfStatsAFType             InetAddressType,
        ipIfStatsIfIndex            InterfaceIndexOrZero,
        ipIfStatsInReceives         Counter32,
        ipIfStatsInHdrErrors        Counter32,
        ipIfStatsInTooBigErrors     Counter32,
        ipIfStatsInNoRoutes         Counter32,
        ipIfStatsInAddrErrors       Counter32,
        ipIfStatsInUnknownProtos    Counter32,
        ipIfStatsInTruncatedPkts    Counter32,
        ipIfStatsInDiscards         Counter32,
        ipIfStatsInDelivers         Counter32,
        ipIfStatsOutForwDatagrams   Counter32,
        ipIfStatsOutRequests        Counter32,
        ipIfStatsOutDiscards        Counter32,
        ipIfStatsOutFragOKs         Counter32,
        ipIfStatsOutFragFails       Counter32,
        ipIfStatsOutFragCreates     Counter32,
        ipIfStatsReasmReqds         Counter32,
        ipIfStatsReasmOKs           Counter32,
        ipIfStatsReasmFails         Counter32,
        ipIfStatsInMcastPkts        Counter32,
        ipIfStatsOutMcastPkts       Counter32,
        ipIfStatsInOctets           Counter32,
        ipIfStatsOutOctets          Counter32,
        ipIfStatsInBcastPkts        Counter32,
        ipIfStatsOutBcastPkts       Counter32,
        ipIfStatsHCInOctets         Counter64,
        ipIfStatsHCInUcastPkts      Counter64,
        ipIfStatsHCInMcastPkts      Counter64,
        ipIfStatsHCInBcastPkts      Counter64,
        ipIfStatsHCOutOctets        Counter64,
        ipIfStatsHCOutUcastPkts     Counter64,
        ipIfStatsHCOutMcastPkts     Counter64,
        ipIfStatsHCOutBcastPkts     Counter64,
        ipIfStatsInMcastOctets      Counter32,
        ipIfStatsOutMcastOctets     Counter32,
        ipIfStatsHCInMcastOctets    Counter64,
        ipIfStatsHCOutMcastOctets   Counter64,
        ipIfStatsDiscontinuityTime  TimeStamp
    }

ipIfStatsAFType OBJECT-TYPE
    SYNTAX      InetAddressType
    MAX-ACCESS not-accessible
    STATUS      current
    DESCRIPTION
           "The address family for this row.  May only be IPv4 or IPv6."
    ::= { ipIfStatsEntry 1 }

ipIfStatsIfIndex OBJECT-TYPE
    SYNTAX      InterfaceIndexOrZero
    MAX-ACCESS not-accessible
    STATUS      current
    DESCRIPTION
           "The interface index, or zero for system-wide counters."
    ::= { ipIfStatsEntry 2 }

ipIfStatsInReceives OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
           "The total number of input IP datagrams received by the
            interface, including those received in error."
    ::= { ipIfStatsEntry 3 }

ipIfStatsInHdrErrors OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
           "The number of input IP datagrams discarded due to errors in
            their IP headers, including version number mismatch, other
            format errors, hop count exceeded, errors discovered in
            processing their IP options, etc.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
```

```
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 4 }

ipIfStatsInTooBigErrors OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of input IP datagrams that could not be forwarded
            because their size exceeded the link MTU of the outgoing
            interface.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 5 }

ipIfStatsInNoRoutes OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of input IP datagrams discarded because no route
            could be found to transmit them to their destination.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."

    ::= { ipIfStatsEntry 6 }

ipIfStatsInAddrErrors OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of input IP datagrams discarded because the IP
            address in their IP header's destination field was not a
            valid address to be received at this entity.  This count
            includes invalid addresses (e.g., ::0) and unsupported
            addresses (e.g., addresses with unallocated prefixes).  For
            entities which are not IP routers and therefore do not
            forward datagrams, this counter includes datagrams discarded
            because the destination address was not a local address.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 7 }

ipIfStatsInUnknownProtos OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of locally-addressed IP datagrams received
            successfully but discarded because of an unknown or
            unsupported protocol. This counter is incremented at the
            interface to which these datagrams were addressed which
            might not be necessarily the input interface for some of the
            datagrams.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 8 }

ipIfStatsInTruncatedPkts OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of input IP datagrams discarded because datagram
            frame didn't carry enough data.
            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 9 }
```

```
ipIfStatsInDiscards OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of input IP datagrams for which no problems were
            encountered to prevent their continued processing, but which
            were discarded (e.g., for lack of buffer space).  Note that
            this counter does not include any datagrams discarded while
            awaiting re-assembly.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 10 }

ipIfStatsInDelivers OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of datagrams successfully delivered to IP
            user-protocols (including ICMP).  This counter is
            incremented at the interface to which these datagrams were
            addressed which might not be necessarily the input interface
            for some of the datagrams.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 11 }

ipIfStatsOutForwDatagrams OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of output datagrams which this entity received
            and forwarded to their final destinations.  In entities
            which do not act as IP routers, this counter will include
            only those packets which were Source-Routed via this entity,
            and the Source-Route processing was successful.  Note that
            for a successfully forwarded datagram the counter of the
            outgoing interface is incremented.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 12 }

ipIfStatsOutRequests OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of IP datagrams which local IP user-
            protocols (including ICMP) supplied to IP in requests for
            transmission.  Note that this counter does not include any
            datagrams counted in ipIfStatsOutForwDatagrams.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 13 }

ipIfStatsOutDiscards OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of output IP datagrams for which no problem was
            encountered to prevent their transmission to their
            destination, but which were discarded (e.g., for lack of
            buffer space).  Note that this counter would include
            datagrams counted in ipIfStatsOutForwDatagrams if any such
            packets met this (discretionary) discard criterion.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
```

```
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 14 }

ipIfStatsOutFragOKs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of IP datagrams that have been successfully
            fragmented at this output interface.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 15 }

ipIfStatsOutFragFails OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of IP datagrams that have been discarded because
            they needed to be fragmented at this output interface but
            could not be.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 16 }

ipIfStatsOutFragCreates OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of output datagram fragments that have been
            generated as a result of IP fragmentation at this output
            interface.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 17 }

ipIfStatsReasmReqds OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of IP fragments received which needed to be
            reassembled at this interface.  Note that this counter is
            incremented at the interface to which these fragments were
            addressed which might not be necessarily the input interface
            for some of the fragments.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 18 }

ipIfStatsReasmOKs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of IP datagrams successfully reassembled.  Note
            that this counter is incremented at the interface to which
            these datagrams were addressed which might not be
            necessarily the input interface for some of the fragments.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 19 }

ipIfStatsReasmFails OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
```

```
    STATUS     current
    DESCRIPTION
           "The number of failures detected by the IP re-assembly
            algorithm (for whatever reason: timed out, errors, etc.).
            Note that this is not necessarily a count of discarded IP
            fragments since some algorithms (notably the algorithm in
            RFC 815) can lose track of the number of fragments by
            combining them as they are received.  This counter is
            incremented at the interface to which these fragments were
            addressed which might not be necessarily the input interface
            for some of the fragments.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 20 }

ipIfStatsInMcastPkts OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of IP multicast packets received by the interface

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 21 }

ipIfStatsOutMcastPkts OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of IP multicast packets transmitted by the
            interface

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 22 }

ipIfStatsInOctets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of octets in input IP datagrams received by
            the interface, including those received in error.

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 23 }

ipIfStatsOutOctets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of octets in output IP datagrams delivered
            to the lower layer on this interface.
            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 24 }

ipIfStatsInBcastPkts OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 25 }
```

```
ipIfStatsOutBcastPkts OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 26 }

ipIfStatsHCInOctets OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 27 }

ipIfStatsHCInUcastPkts OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 28 }

ipIfStatsHCInMcastPkts OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 29 }

ipIfStatsHCInBcastPkts OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 30 }

ipIfStatsHCOutOctets OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "

            Discontinuities in the value of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 31 }


ipIfStatsHCOutUcastPkts OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
```

```
    DESCRIPTION
            "

             Discontinuities in the value of this counter can occur at
             re-initialization of the management system, and at other
             times as indicated by the value of
             ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 32 }

ipIfStatsHCOutMcastPkts OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "

             Discontinuities in the value of this counter can occur at
             re-initialization of the management system, and at other
             times as indicated by the value of
             ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 33 }

ipIfStatsHCOutBcastPkts OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "

             Discontinuities in the value of this counter can occur at
             re-initialization of the management system, and at other
             times as indicated by the value of
             ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 34 }

ipIfStatsInMcastOctets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "

             Discontinuities in the value of this counter can occur at
             re-initialization of the management system, and at other
             times as indicated by the value of
             ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 35 }

ipIfStatsOutMcastOctets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "

             Discontinuities in the value of this counter can occur at
             re-initialization of the management system, and at other
             times as indicated by the value of
             ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 36 }

ipIfStatsHCInMcastOctets OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "

             Discontinuities in the value of this counter can occur at
             re-initialization of the management system, and at other
             times as indicated by the value of
             ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 37 }

ipIfStatsHCOutMcastOctets OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "

             Discontinuities in the value of this counter can occur at
             re-initialization of the management system, and at other
             times as indicated by the value of
```

```
            ipIfStatsDiscontinuityTime."
    ::= { ipIfStatsEntry 38 }

ipIfStatsDiscontinuityTime OBJECT-TYPE
    SYNTAX     TimeStamp
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The value of sysUpTime on the most recent occasion at which
            any one or more of this entry's counters suffered a
            discontinuity.

            If no such discontinuities have occurred since the last
            reinitialization of the local management subsystem, then
            this object contains a zero value."
    ::= { ipIfStatsEntry 39 }




--
-- Internet Address Prefix table
--
-- Open Issues:
--  What's OnLinkFlag for IPv4?
--  What's AutonomousFlag for IPv4?
--  What are PreferredLifetime and ValidLifetime for IPv4?
--   Is there a better SMI data type for *Lifetime objects?

ipAddressPrefixTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF IpAddressPrefixEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "inet prefix table"
    ::= { ip 27 }

ipAddressPrefixEntry OBJECT-TYPE
    SYNTAX     IpAddressPrefixEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "inet prefix entry"
    INDEX    { ipAddressPrefixIfIndex, ipAddressPrefixType,
                                          ipAddressPrefixPrefix, ipAddressPrefixLength }
    ::= { ipAddressPrefixTable 1 }

IpAddressPrefixEntry ::= SEQUENCE {
        ipAddressPrefixIfIndex               InterfaceIndex,
        ipAddressPrefixType                  InetAddressType,
        ipAddressPrefixPrefix                InetAddress,
        ipAddressPrefixLength                InetAddressPrefixLength,
        ipAddressPrefixOrigin                INTEGER,
        ipAddressPrefixOnLinkFlag            TruthValue,
        ipAddressPrefixAutonomousFlag        TruthValue,
        ipAddressPrefixAdvPreferredLifetime  Unsigned32,
        ipAddressPrefixAdvValidLifetime      Unsigned32
    }

ipAddressPrefixIfIndex OBJECT-TYPE
    SYNTAX     InterfaceIndex
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The interface on which this prefix is configured."
    ::= { ipAddressPrefixEntry 1 }

ipAddressPrefixType OBJECT-TYPE
    SYNTAX     InetAddressType
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The address type of ipAddressPrefix.  Only IPv4 and IPv6
            addresses are expected."
    ::= { ipAddressPrefixEntry 2 }

ipAddressPrefixPrefix OBJECT-TYPE
    SYNTAX     InetAddress (SIZE(0..36))
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The address prefix.  Bits after ipAddressPrefixLength must
            be zero."
    ::= { ipAddressPrefixEntry 3 }
```

```
ipAddressPrefixLength OBJECT-TYPE
    SYNTAX     InetAddressPrefixLength
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
            "The prefix length associated with this prefix."
    ::= { ipAddressPrefixEntry 4 }

ipAddressPrefixOrigin OBJECT-TYPE
    SYNTAX     INTEGER {
        other(1),
                                    manual(2),
                                    wellknown(3),
                                    dhcp(4),
                                    routeradv(5)
    }
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The origin of this prefix.  manual(2) indicates a prefix
             that was manually configured.  wellknown(3) indicates a
             well-known prefix, e.g. 169.254/16 for IPv4
             autoconfiguration or fe80::/10 for IPv6 link-local
             addresses.  dhcp(4) indicates a prefix that was assigned by
             a DHCP server.  routeradv(5) indicates a prefix learned from
             a router advertisement."
    ::= { ipAddressPrefixEntry 5 }

ipAddressPrefixOnLinkFlag OBJECT-TYPE
    SYNTAX     TruthValue
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "This object has the value 'true(1)', if this prefix can be
             used  for on-link determination and the value 'false(2)'
             otherwise."
    ::= { ipAddressPrefixEntry 6 }

ipAddressPrefixAutonomousFlag OBJECT-TYPE
    SYNTAX     TruthValue
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "Autonomous address configuration flag. When true(1),
             indicates that this prefix can be used for autonomous
             address configuration (i.e. can be used to form a local
             interface address).  If false(2), it is not used to
             autoconfigure a local interface address."
    ::= { ipAddressPrefixEntry 7 }

ipAddressPrefixAdvPreferredLifetime OBJECT-TYPE
    SYNTAX     Unsigned32
    UNITS       "seconds"
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The length of time in seconds that this prefix will remain
             preferred, i.e. time until deprecation.  A value of
             4,294,967,295 represents infinity.

             The address generated from a deprecated prefix should no
             longer be used as a source address in new communications,
             but packets received on such an interface are processed as
             expected."
    ::= { ipAddressPrefixEntry 8 }

ipAddressPrefixAdvValidLifetime OBJECT-TYPE
    SYNTAX     Unsigned32
    UNITS       "seconds"
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The length of time in seconds that this prefix will remain
             valid, i.e. time until invalidation.  A value of
             4,294,967,295 represents infinity.

             The address generated from an invalidated prefix should not
             appear as the destination or source address of a packet."
    ::= { ipAddressPrefixEntry 9 }
```

```
--
-- Internet Address Table
--
-- Open Issues:
--  should ipAddressv4BcastAddr go somewhere else?
--  meeting notes said: dave: pointer to prefix table.  What's that mean?

ipAddressTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF IpAddressEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "inet addr table"
    ::= { ip 28 }

ipAddressEntry OBJECT-TYPE
    SYNTAX     IpAddressEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "inet addr entry"
    INDEX { ipAddressAddrType, ipAddressAddr }
    ::= { ipAddressTable 1 }

IpAddressEntry ::= SEQUENCE {
       ipAddressAddrType  InetAddressType,
       ipAddressAddr      InetAddress,
       ipAddressIfIndex   InterfaceIndex,
       ipAddressType      INTEGER,
       ipAddressPrefix    RowPointer,
       ipAddressOrigin    INTEGER,
       ipAddressStatus    INTEGER
    }

ipAddressAddrType OBJECT-TYPE
    SYNTAX     InetAddressType
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The address type of ipAddressAddr."
    ::= { ipAddressEntry 1 }

ipAddressAddr OBJECT-TYPE
    SYNTAX     InetAddress (SIZE(0..36))
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The IP address to which this entry's addressing information
            pertains."
    ::= { ipAddressEntry 2 }

ipAddressIfIndex OBJECT-TYPE
    SYNTAX     InterfaceIndex
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The index value which uniquely identifies the interface to
            which this entry is applicable.  The interface identified by
            a particular value of this index is the same interface as
            identified by the same value of the IF-MIB's ifIndex."
    ::= { ipAddressEntry 3 }

ipAddressType OBJECT-TYPE
    SYNTAX     INTEGER {
                 unicast(1),
                 anycast(2),
                 broadcast(3)
    }
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The type of address.  broadcast(3) is not a valid value for
            IPv6 addresses [draft-ietf-ipngwg-addr-arch-v3-05.txt]."
    ::= { ipAddressEntry 4 }

ipAddressPrefix OBJECT-TYPE
    SYNTAX     RowPointer
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "A pointer to the row in the prefix table to which this
            address belongs.  May be { 0 0 } if there is no such row."
    ::= { ipAddressEntry 5 }
```

```
ipAddressOrigin OBJECT-TYPE
    SYNTAX     INTEGER {
                 other(1),
                                          manual(2),
                                          wellknown(3),
                                          dhcp(4),-- XXX or assignedbyserver ?
                                          linklayer(5),
                                          random(6)
    }
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The origin of the address.  manual(2) indicates that the
            address was manually configured.  wellknown(3) indicates an
            address constructed from a well-known value, e.g. an IANA-
            assigned anycast address.  dhcp(4) indicates an address that
            was assigned to this system by a DHCP server.  linklayer(5)
            indicates an address created by IPv6 stateless
            autoconfiguration.  random(6) indicates an address chosen by
            random, e.g. an IPv4 address within 169.254/16, or an RFC
            3041 privacy address."
    ::= { ipAddressEntry 6 }

ipAddressStatus OBJECT-TYPE
    SYNTAX     INTEGER {
                    preferred(1),

                    deprecated(2),

                    invalid(3),

                    inaccessible(4),

                    unknown(5),  -- status can not be determined
                                 -- for some reason.
                                          tentative(6),

                                          duplicate(7)
                   }
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "Address status.  The preferred(1) state indicates that this
            is a valid address that can appear as the destination or
            source address of a packet.  The deprecated(2) state
            indicates that this is a valid but deprecated address that
            should no longer be used as a source address in new
            communications, but packets addressed to such an address are
            processed as expected. The invalid(3) state indicates that
            this is not valid address which should not appear as the
            destination or source address of a packet. The
            inaccessible(4) state indicates that the address is not
            accessible because the interface to which this address is
            assigned is not operational.  The tentative(6) state
            indicates the uniqueness of the address on the link is being
            verified.  The duplicate(7) state indicates the address has
            been determined to be non-unique on the link and so must not
            be used.

            In the absence of other information, an IPv4 address is
            always preferred(1)."
    ::= { ipAddressEntry 7 }


-- the Internet Address Translation table

inetNetToMediaTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF InetNetToMediaEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The IP Address Translation table used for mapping from IP
            addresses to physical addresses.

            The Address Translation tables contain the IP address to
            'physical' address equivalences.  Some interfaces do not use
            translation tables for determining address equivalences
            (e.g., DDN-X.25 has an algorithmic method); if all
            interfaces are of this type, then the Address Translation
            table is empty, i.e., has zero entries."
    ::= { ip 29 }

inetNetToMediaEntry OBJECT-TYPE
```

```
    SYNTAX     InetNetToMediaEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "Each entry contains one IP address to 'physical' address
            equivalence."
    INDEX      { ifIndex,
                 inetNetToMediaNetAddressType,
                 inetNetToMediaNetAddress }
    ::= { inetNetToMediaTable 1 }

InetNetToMediaEntry ::= SEQUENCE {
        inetNetToMediaNetAddressType  InetAddressType,
        inetNetToMediaNetAddress      InetAddress,
        inetNetToMediaPhysAddress     PhysAddress,
        inetNetToMediaLastUpdated     TimeStamp,
        inetNetToMediaType            INTEGER,
        inetNetToMediaState           INTEGER
    }

inetNetToMediaNetAddressType OBJECT-TYPE
    SYNTAX     InetAddressType
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The type of inetNetToMediaNetAddress."
    ::= { inetNetToMediaEntry 1 }

inetNetToMediaNetAddress OBJECT-TYPE
    SYNTAX     InetAddress (SIZE(0..36))
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The IP Address corresponding to the media-dependent
            'physical' address."
    ::= { inetNetToMediaEntry 2 }

inetNetToMediaPhysAddress OBJECT-TYPE
    SYNTAX     PhysAddress
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
           "The media-dependent 'physical' address."
    ::= { inetNetToMediaEntry 3 }

inetNetToMediaLastUpdated OBJECT-TYPE
    SYNTAX     TimeStamp
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The value of sysUpTime at the time this entry was last
            updated.  If this entry was updated prior to the last re-
            initialization of the local network management subsystem,
            then this object contains a zero value."
    ::= { inetNetToMediaEntry 4 }


inetNetToMediaType OBJECT-TYPE
    SYNTAX     INTEGER {
                 other(1),        -- none of the following
                 invalid(2),      -- an invalidated mapping
                 dynamic(3),
                 static(4),
                 local(5)         -- local interface
               }
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
           "The type of mapping.

            Setting this object to the value invalid(2) has the effect
            of invalidating the corresponding entry in the
            inetNetToMediaTable.  That is, it effectively disassociates
            the interface identified with said entry from the mapping
            identified with said entry.  It is an implementation-
            specific matter as to whether the agent removes an
            invalidated entry from the table.  Accordingly, management
            stations must be prepared to receive tabular information
            from agents that corresponds to entries not currently in
            use.  Proper interpretation of such entries requires
            examination of the relevant inetNetToMediaType object.

            The 'dynamic(3)' type indicates that the IP address to
            physical addresses mapping has been dynamically resolved
```

```
            using e.g. IPv4 ARP or the IPv6 Neighbor Discovery protocol.
            The 'static(4)' type indicates that the mapping has been
            statically configured.  The 'local(5)' type indicates that
            the mapping is provided for an entity's own interface
            address."
    ::= { inetNetToMediaEntry 5 }

inetNetToMediaState OBJECT-TYPE
    SYNTAX     INTEGER {
                   reachable(1), -- confirmed reachability

                   stale(2),     -- unconfirmed reachability

                   delay(3),     -- waiting for reachability
                                 -- confirmation before entering
                                 -- the probe state

                   probe(4),     -- actively probing

                   invalid(5),   -- an invalidated mapping

                   unknown(6),   -- state can not be determined
                                 -- for some reason.
                              incomplete(7) -- address resolution is being performed.
                  }
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The Neighbor Unreachability Detection [3] state for the
            interface when the address mapping in this entry is used.
            If Neighbor Unreachability Detection is not in use (e.g. for
            IPv4), this object is always unknown(6)."
    REFERENCE "RFC2461"
    ::= { inetNetToMediaEntry 6 }



--
-- The IPv6 Scope Identifier Table.
--
-- Open Issues:
--   Should there be associated objects to provide a scope description,
--   similar to ipMRouteScopeNameString?

-- XXX ScopeIdentifier TC should move to INET-ADDRESS-MIB
ScopeIdentifier ::= TEXTUAL-CONVENTION
    STATUS     current
    DESCRIPTION
           "A Scope Identifier identifies an instance of a specific
            scope.

            The scope identifier MUST disambiguate identical address
            values. For link-local addresses, the scope identifier will
            typically be the interface index (ifIndex as defined in the
            IF-MIB) of the interface on which the address is configured.

            The scope identifier may contain the special value 0 which
            refers to the default scope. The default scope may be used
            in cases where the valid scope identifier is not known
            (e.g., a management application needs to write a site-local
            InetAddressIPv6 address without knowing the site identifier
            value). The default scope SHOULD NOT be used as an easy way
            out in cases where the scope identifier for a non-global
            IPv6 address is known."
    SYNTAX     Unsigned32

ipv6ScopeIdTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF Ipv6ScopeIdEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The table used to describe IPv6 unicast and multicast scope
            zones."
    ::= { ip 30 }

ipv6ScopeIdEntry OBJECT-TYPE
    SYNTAX     Ipv6ScopeIdEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "Each entry contains the list of scope identifiers on a given
            interface."
    INDEX { ipv6ScopeIdIfIndex }
    ::= { ipv6ScopeIdTable 1 }
```

```
Ipv6ScopeIdEntry ::= SEQUENCE {
        ipv6ScopeIdIfIndex            InterfaceIndex,
        ipv6ScopeIdLinkLocal          ScopeIdentifier,
        ipv6ScopeIdSubnetLocal        ScopeIdentifier,
        ipv6ScopeIdAdminLocal         ScopeIdentifier,
        ipv6ScopeIdSiteLocal          ScopeIdentifier,
        ipv6ScopeId6                  ScopeIdentifier,
        ipv6ScopeId7                  ScopeIdentifier,
        ipv6ScopeIdOrganizationLocal  ScopeIdentifier,
        ipv6ScopeId9                  ScopeIdentifier,
        ipv6ScopeIdA                  ScopeIdentifier,
        ipv6ScopeIdB                  ScopeIdentifier,
        ipv6ScopeIdC                  ScopeIdentifier,
        ipv6ScopeIdD                  ScopeIdentifier
    }

ipv6ScopeIdIfIndex OBJECT-TYPE
    SYNTAX     InterfaceIndex
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The interface to which these scopes belong."
    ::= { ipv6ScopeIdEntry 1 }

ipv6ScopeIdLinkLocal OBJECT-TYPE
    SYNTAX     ScopeIdentifier
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The Scope Identifier for the link-local scope on this
            interface."
    ::= { ipv6ScopeIdEntry 2 }

ipv6ScopeIdSubnetLocal OBJECT-TYPE
    SYNTAX     ScopeIdentifier
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The Scope Identifier for the subnet-local scope on this
            interface."
    ::= { ipv6ScopeIdEntry 3 }

ipv6ScopeIdAdminLocal OBJECT-TYPE
    SYNTAX     ScopeIdentifier
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The Scope Identifier for the admin-local scope on this
            interface."
    ::= { ipv6ScopeIdEntry 4 }

ipv6ScopeIdSiteLocal OBJECT-TYPE
    SYNTAX     ScopeIdentifier
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The Scope Identifier for the site-local scope on this
            interface."
    ::= { ipv6ScopeIdEntry 5 }

ipv6ScopeId6 OBJECT-TYPE
    SYNTAX     ScopeIdentifier
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The Scope Identifier for scope 6 on this interface."
    ::= { ipv6ScopeIdEntry 6 }

ipv6ScopeId7 OBJECT-TYPE
    SYNTAX     ScopeIdentifier
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The Scope Identifier for scope 7 on this interface."
    ::= { ipv6ScopeIdEntry 7 }

ipv6ScopeIdOrganizationLocal OBJECT-TYPE
    SYNTAX     ScopeIdentifier
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The Scope Identifier for the orgainzation-local scope on
            this interface."
```

```
    ::= { ipv6ScopeIdEntry 8 }

ipv6ScopeId9 OBJECT-TYPE
    SYNTAX     ScopeIdentifier
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The Scope Identifier for scope 9 on this interface."
    ::= { ipv6ScopeIdEntry 9 }

ipv6ScopeIdA OBJECT-TYPE
    SYNTAX     ScopeIdentifier
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The Scope Identifier for scope A on this interface."
    ::= { ipv6ScopeIdEntry 10 }

ipv6ScopeIdB OBJECT-TYPE
    SYNTAX     ScopeIdentifier
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The Scope Identifier for scope B on this interface."
    ::= { ipv6ScopeIdEntry 11 }

ipv6ScopeIdC OBJECT-TYPE
    SYNTAX     ScopeIdentifier
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The Scope Identifier for scope C on this interface."
    ::= { ipv6ScopeIdEntry 12 }

ipv6ScopeIdD OBJECT-TYPE
    SYNTAX     ScopeIdentifier
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The Scope Identifier for scope D on this interface."
    ::= { ipv6ScopeIdEntry 13 }



icmp     OBJECT IDENTIFIER ::= { mib-2 5 }

--
-- ICMP non-message-specific counters
--
--
-- To do:
--  expand table DESCRIPTION to describe index
--   (including whether an agent MUST support system-wide, per-if,
--    both, or neither, to be compliant to this MIB.
--    Also, it might be useful to remind readers that the
--    system-wide value is not the sum of the per-if counters.)
-- ************************************************************

inetIcmpTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF InetIcmpEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The table of generic ICMP counters.  These counters may be
            kept per-interface and/or system-wide."
    ::= { icmp 27 }

inetIcmpEntry OBJECT-TYPE
    SYNTAX     InetIcmpEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "A conceptual row in the inetIcmpTable.

            A row with an inetIcmpIfIndex value of zero indicates a
            system-wide value; a row with a non-zero inetIcmpIfIndex
            indicates an interface-specific value.  A system may provide
            both system-wide and interface-specific values, in which
            case it is important to note that the system-wide value may
            not be equal to the sum of the interface-specific values
            across all interfaces due to e.g. dynamic interface
            creation/deletion."
    INDEX    { inetIcmpAFType, inetIcmpIfIndex }
    ::= { inetIcmpTable 1 }
```

```
InetIcmpEntry ::= SEQUENCE {
        inetIcmpAFType     InetAddressType,
        inetIcmpIfIndex    InterfaceIndexOrZero,
        inetIcmpInMsgs     Counter32,
        inetIcmpInErrors   Counter32,
        inetIcmpOutMsgs    Counter32,
        inetIcmpOutErrors  Counter32
    }


inetIcmpAFType OBJECT-TYPE
    SYNTAX     InetAddressType
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The IP address family of the statistics."
    ::= { inetIcmpEntry 1 }

inetIcmpIfIndex OBJECT-TYPE
    SYNTAX     InterfaceIndexOrZero
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The ifindex of the interface, or zero for system-wide
            stats."
    ::= { inetIcmpEntry 2 }

inetIcmpInMsgs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of ICMP messages which the entity received.
            Note that this counter includes all those counted by
            inetIcmpInErrors."
    ::= { inetIcmpEntry 3 }

inetIcmpInErrors OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of ICMP messages which the entity received but
            determined as having ICMP-specific errors (bad ICMP
            checksums, bad length, etc.)."
    ::= { inetIcmpEntry 4 }

inetIcmpOutMsgs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of ICMP messages which the entity received.
            Note that this counter includes all those counted by
            inetIcmpOutErrors."
    ::= { inetIcmpEntry 5 }

inetIcmpOutErrors OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of ICMP messages which this entity did not send
            due to problems discovered within ICMP such as a lack of
            buffers.  This value should not include errors discovered
            outside the ICMP layer such as the inability of IP to route
            the resultant datagram.  In some implementations there may
            be no types of error which contribute to this counter's
            value."
    ::= { inetIcmpEntry 6 }



--
-- per-AF, per-interface(optionally), per-msg type and code ICMP counters
--


inetIcmpMsgTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF InetIcmpMsgEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
```

```
          "The table of per-message ICMP counters.  These counters may
           be kept per-interface and/or system-wide."
    ::= { icmp 28 }

inetIcmpMsgEntry OBJECT-TYPE
    SYNTAX     InetIcmpMsgEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
          "A conceptual row in the inetIcmpMsgTable.

           A row with an inetIcmpMsgIfIndex value of zero indicates a
           system-wide value; a row with a non-zero inetIcmpMsgIfIndex
           indicates an interface-specific value.  A system may provide
           both system-wide and interface-specific values, in which
           case it is important to note that the system-wide value may
           not be equal to the sum of the interface-specific values
           across all interfaces due to e.g. dynamic interface
           creation/deletion.

           XXX How to phrase this if?  If the system keeps track of
           individual ICMP code values (e.g.  destination unreachable,
           code administratively prohibited), it creates several rows
           for each inetIcmpMsgType, each with an appropriate value of
           inetIcmpMsgCode.  A row with the special value of
           inetIcmpMsgCode, 256, counts all packets with type
           inetIcmpMsgType that aren't counted in rows with a value of
           inetIcmpMsgCode other than 256."
    INDEX { inetIcmpMsgAFType, inetIcmpMsgIfIndex, inetIcmpMsgType,
           inetIcmpMsgCode }
    ::= { inetIcmpMsgTable 1 }

InetIcmpMsgEntry ::= SEQUENCE {
       inetIcmpMsgAFType   InetAddressType,
       inetIcmpMsgIfIndex  InterfaceIndexOrZero,
       inetIcmpMsgType     Integer32,
       inetIcmpMsgCode     Integer32,
       inetIcmpMsgInPkts   Counter32,
       inetIcmpMsgOutPkts  Counter32
    }

inetIcmpMsgAFType OBJECT-TYPE
    SYNTAX     InetAddressType
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
          "The IP address family of the statistics."
    ::= { inetIcmpMsgEntry 1 }

inetIcmpMsgIfIndex OBJECT-TYPE
    SYNTAX     InterfaceIndexOrZero
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
          "The ifindex of the interface, or zero for system-wide
           stats."
    ::= { inetIcmpMsgEntry 2 }

inetIcmpMsgType OBJECT-TYPE
    SYNTAX     Integer32 (0..255)
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
          "The ICMP type field of the message type being counted by
           this row."
    ::= { inetIcmpMsgEntry 3 }

inetIcmpMsgCode OBJECT-TYPE
    SYNTAX     Integer32 (0..256)
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
          "The ICMP code field of the message type being counted by
           this row, or the special value 256 if no specific ICMP code
           is counted by this row."
    ::= { inetIcmpMsgEntry 4 }

inetIcmpMsgInPkts OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
          "The number of input packets for this AF, ifindex, type,
           code."
```

```
    ::= { inetIcmpMsgEntry 5 }

inetIcmpMsgOutPkts OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of output packets for this AF, ifindex, type,
            code."
    ::= { inetIcmpMsgEntry 6 }



-- XXX
-- To do: move current conformance information here.

--
-- Deprecated objects
--

ipInReceives OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The total number of input datagrams received from
            interfaces, including those received in error."
    ::= { ip 3 }

ipInHdrErrors OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of input datagrams discarded due to errors in
            their IPv4 headers, including bad checksums, version number
            mismatch, other format errors, time-to-live exceeded, errors
            discovered in processing their IPv4 options, etc."
    ::= { ip 4 }

ipInAddrErrors OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of input datagrams discarded because the IPv4
            address in their IPv4 header's destination field was not a
            valid address to be received at this entity.  This count
            includes invalid addresses (e.g., 0.0.0.0) and addresses of
            unsupported Classes (e.g., Class E).  For entities which are
            not IPv4 routers and therefore do not forward datagrams,
            this counter includes datagrams discarded because the
            destination address was not a local address."
    ::= { ip 5 }

ipForwDatagrams OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of input datagrams for which this entity was not
            their final IPv4 destination, as a result of which an
            attempt was made to find a route to forward them to that
            final destination.  In entities which do not act as IPv4
            routers, this counter will include only those packets which
            were Source-Routed via this entity, and the Source-Route
            option processing was successful."
    ::= { ip 6 }

ipInUnknownProtos OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of locally-addressed datagrams received
            successfully but discarded because of an unknown or
            unsupported protocol."
    ::= { ip 7 }

ipInDiscards OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
```

```
           "The number of input IPv4 datagrams for which no problems
            were encountered to prevent their continued processing, but
            which were discarded (e.g., for lack of buffer space).  Note
            that this counter does not include any datagrams discarded
            while awaiting re-assembly."
    ::= { ip 8 }

ipInDelivers OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The total number of input datagrams successfully delivered
            to IPv4 user-protocols (including ICMP)."
    ::= { ip 9 }

ipOutRequests OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The total number of IPv4 datagrams which local IPv4 user
            protocols (including ICMP) supplied to IPv4 in requests for
            transmission.  Note that this counter does not include any
            datagrams counted in ipForwDatagrams."
    ::= { ip 10 }

ipOutDiscards OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of output IPv4 datagrams for which no problem was
            encountered to prevent their transmission to their
            destination, but which were discarded (e.g., for lack of
            buffer space).  Note that this counter would include
            datagrams counted in ipForwDatagrams if any such packets met
            this (discretionary) discard criterion."
    ::= { ip 11 }

ipOutNoRoutes OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of IPv4 datagrams discarded because no route
            could be found to transmit them to their destination.  Note
            that this counter includes any packets counted in
            ipForwDatagrams which meet this 'no-route' criterion.  Note
            that this includes any datagrams which a host cannot route
            because all of its default routers are down."
    ::= { ip 12 }

ipReasmReqds OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of IPv4 fragments received which needed to be
            reassembled at this entity."
    ::= { ip 14 }

ipReasmOKs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of IPv4 datagrams successfully re-assembled."
    ::= { ip 15 }

ipReasmFails OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of failures detected by the IPv4 re-assembly
            algorithm (for whatever reason: timed out, errors, etc).
            Note that this is not necessarily a count of discarded IPv4
            fragments since some algorithms (notably the algorithm in
            RFC 815) can lose track of the number of fragments by
            combining them as they are received."
    ::= { ip 16 }

ipFragOKs OBJECT-TYPE
```

```
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of IPv4 datagrams that have been successfully
            fragmented at this entity."
    ::= { ip 17 }

ipFragFails OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of IPv4 datagrams that have been discarded
            because they needed to be fragmented at this entity but
            could not be, e.g., because their Don't Fragment flag was
            set."
    ::= { ip 18 }

ipFragCreates OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of IPv4 datagram fragments that have been
            generated as a result of fragmentation at this entity."
    ::= { ip 19 }

ipRoutingDiscards OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of routing entries which were chosen to be
            discarded even though they are valid.  One possible reason
            for discarding such an entry could be to free-up buffer
            space for other routing entries."
    ::= { ip 23 }

-- the deprecated IPv4 address table

ipAddrTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF IpAddrEntry
    MAX-ACCESS not-accessible
    STATUS     deprecated
    DESCRIPTION
           "The table of addressing information relevant to this
            entity's IPv4 addresses."
    ::= { ip 20 }

ipAddrEntry OBJECT-TYPE
    SYNTAX     IpAddrEntry
    MAX-ACCESS not-accessible
    STATUS     deprecated
    DESCRIPTION
           "The addressing information for one of this entity's IPv4
            addresses."
    INDEX      { ipAdEntAddr }
    ::= { ipAddrTable 1 }


IpAddrEntry ::= SEQUENCE {
        ipAdEntAddr          IpAddress,
        ipAdEntIfIndex       INTEGER,
        ipAdEntNetMask       IpAddress,
        ipAdEntBcastAddr     INTEGER,
        ipAdEntReasmMaxSize  INTEGER
    }


ipAdEntAddr OBJECT-TYPE
    SYNTAX     IpAddress
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The IPv4 address to which this entry's addressing
            information pertains."
    ::= { ipAddrEntry 1 }

ipAdEntIfIndex OBJECT-TYPE
    SYNTAX     INTEGER (1..2147483647)
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
```

```
           "The index value which uniquely identifies the interface to
            which this entry is applicable.  The interface identified by
            a particular value of this index is the same interface as
            identified by the same value of RFC 2863's ifIndex."
    ::= { ipAddrEntry 2 }

ipAdEntNetMask OBJECT-TYPE
    SYNTAX     IpAddress
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The subnet mask associated with the IPv4 address of this
            entry.  The value of the mask is an IPv4 address with all
            the network bits set to 1 and all the hosts bits set to 0."
    ::= { ipAddrEntry 3 }

ipAdEntBcastAddr OBJECT-TYPE
    SYNTAX     INTEGER (0..1)
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The value of the least-significant bit in the IPv4 broadcast
            address used for sending datagrams on the (logical)
            interface associated with the IPv4 address of this entry.
            For example, when the Internet standard all-ones broadcast
            address is used, the value will be 1.  This value applies to
            both the subnet and network broadcasts addresses used by the
            entity on this (logical) interface."
    ::= { ipAddrEntry 4 }

ipAdEntReasmMaxSize OBJECT-TYPE
    SYNTAX     INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The size of the largest IPv4 datagram which this entity can
            re-assemble from incoming IPv4 fragmented datagrams received
            on this interface."
    ::= { ipAddrEntry 5 }



-- the deprecated IPv4 Address Translation table

-- The Address Translation tables contain the IpAddress to
-- "physical" address equivalences.  Some interfaces do not
-- use translation tables for determining address
-- equivalences (e.g., DDN-X.25 has an algorithmic method);
-- if all interfaces are of this type, then the Address
-- Translation table is empty, i.e., has zero entries.

ipNetToMediaTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF IpNetToMediaEntry
    MAX-ACCESS not-accessible
    STATUS     deprecated
    DESCRIPTION
           "The IPv4 Address Translation table used for mapping from
            IPv4 addresses to physical addresses."
    ::= { ip 22 }

ipNetToMediaEntry OBJECT-TYPE
    SYNTAX     IpNetToMediaEntry
    MAX-ACCESS not-accessible
    STATUS     deprecated
    DESCRIPTION
           "Each entry contains one IpAddress to `physical' address
            equivalence."
    INDEX      { ipNetToMediaIfIndex,
                 ipNetToMediaNetAddress }
    ::= { ipNetToMediaTable 1 }

IpNetToMediaEntry ::= SEQUENCE {
        ipNetToMediaIfIndex      INTEGER,
        ipNetToMediaPhysAddress  PhysAddress,
        ipNetToMediaNetAddress   IpAddress,
        ipNetToMediaType         INTEGER
    }

ipNetToMediaIfIndex OBJECT-TYPE
    SYNTAX     INTEGER (1..2147483647)
    MAX-ACCESS read-create
    STATUS     deprecated
    DESCRIPTION
```

```
           "The interface on which this entry's equivalence is
            effective.  The interface identified by a particular value
            of this index is the same interface as identified by the
            same value of RFC 2863's ifIndex."
    ::= { ipNetToMediaEntry 1 }

ipNetToMediaPhysAddress OBJECT-TYPE
    SYNTAX     PhysAddress
    MAX-ACCESS read-create
    STATUS     deprecated
    DESCRIPTION
           "The media-dependent 'physical' address."
    ::= { ipNetToMediaEntry 2 }

ipNetToMediaNetAddress OBJECT-TYPE
    SYNTAX     IpAddress
    MAX-ACCESS read-create
    STATUS     deprecated
    DESCRIPTION
           "The IpAddress corresponding to the media-dependent
            'physical' address."
    ::= { ipNetToMediaEntry 3 }

ipNetToMediaType OBJECT-TYPE
    SYNTAX     INTEGER {
                 other(1),        -- none of the following
                 invalid(2),      -- an invalidated mapping
                 dynamic(3),
                 static(4)
               }
    MAX-ACCESS read-create
    STATUS     deprecated
    DESCRIPTION
           "The type of mapping.

            Setting this object to the value invalid(2) has the effect
            of invalidating the corresponding entry in the
            ipNetToMediaTable.  That is, it effectively disassociates
            the interface identified with said entry from the mapping
            identified with said entry.  It is an implementation-
            specific matter as to whether the agent removes an
            invalidated entry from the table.  Accordingly, management
            stations must be prepared to receive tabular information
            from agents that corresponds to entries not currently in
            use.  Proper interpretation of such entries requires
            examination of the relevant ipNetToMediaType object."
    ::= { ipNetToMediaEntry 4 }



-- the deprecated ICMP group

icmpInMsgs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The total number of ICMP messages which the entity received.
            Note that this counter includes all those counted by
            icmpInErrors."
    ::= { icmp 1 }

icmpInErrors OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP messages which the entity received but
            determined as having ICMP-specific errors (bad ICMP
            checksums, bad length, etc.)."
    ::= { icmp 2 }

icmpInDestUnreachs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Destination Unreachable messages
            received."
    ::= { icmp 3 }

icmpInTimeExcds OBJECT-TYPE
    SYNTAX     Counter32
```

```
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Time Exceeded messages received."
    ::= { icmp 4 }

icmpInParmProbs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Parameter Problem messages received."
    ::= { icmp 5 }

icmpInSrcQuenchs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Source Quench messages received."
    ::= { icmp 6 }

icmpInRedirects OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Redirect messages received."
    ::= { icmp 7 }

icmpInEchos OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Echo (request) messages received."
    ::= { icmp 8 }

icmpInEchoReps OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Echo Reply messages received."
    ::= { icmp 9 }

icmpInTimestamps OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Timestamp (request) messages received."
    ::= { icmp 10 }

icmpInTimestampReps OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Timestamp Reply messages received."
    ::= { icmp 11 }

icmpInAddrMasks OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Address Mask Request messages received."
    ::= { icmp 12 }

icmpInAddrMaskReps OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Address Mask Reply messages received."
    ::= { icmp 13 }

icmpOutMsgs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The total number of ICMP messages which this entity
```

```
            attempted to send.  Note that this counter includes all
            those counted by icmpOutErrors."
    ::= { icmp 14 }

icmpOutErrors OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP messages which this entity did not send
            due to problems discovered within ICMP such as a lack of
            buffers.  This value should not include errors discovered
            outside the ICMP layer such as the inability of IP to route
            the resultant datagram.  In some implementations there may
            be no types of error which contribute to this counter's
            value."
    ::= { icmp 15 }

icmpOutDestUnreachs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Destination Unreachable messages sent."
    ::= { icmp 16 }

icmpOutTimeExcds OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Time Exceeded messages sent."
    ::= { icmp 17 }

icmpOutParmProbs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Parameter Problem messages sent."
    ::= { icmp 18 }

icmpOutSrcQuenchs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Source Quench messages sent."
    ::= { icmp 19 }

icmpOutRedirects OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Redirect messages sent.  For a host, this
            object will always be zero, since hosts do not send
            redirects."
    ::= { icmp 20 }

icmpOutEchos OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Echo (request) messages sent."
    ::= { icmp 21 }

icmpOutEchoReps OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Echo Reply messages sent."
    ::= { icmp 22 }

icmpOutTimestamps OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Timestamp (request) messages sent."
    ::= { icmp 23 }
```

```
icmpOutTimestampReps OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Timestamp Reply messages sent."
    ::= { icmp 24 }

icmpOutAddrMasks OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Address Mask Request messages sent."
    ::= { icmp 25 }

icmpOutAddrMaskReps OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of ICMP Address Mask Reply messages sent."
    ::= { icmp 26 }

-- conformance information

ipMIBConformance OBJECT IDENTIFIER ::= { ipMIB 2 }

ipMIBCompliances OBJECT IDENTIFIER ::= { ipMIBConformance 1 }
ipMIBGroups      OBJECT IDENTIFIER ::= { ipMIBConformance 2 }

ipv6Conformance OBJECT IDENTIFIER ::= { ipv6MIB 3 }

ipv6Compliances OBJECT IDENTIFIER ::= { ipv6Conformance 1 }
ipv6Groups      OBJECT IDENTIFIER ::= { ipv6Conformance 2 }


-- ipv6IcmpConformance OBJECT IDENTIFIER ::= { ipv6IcmpMIB 2 }
--
-- ipv6IcmpCompliances OBJECT IDENTIFIER ::= { ipv6IcmpConformance 1 }
-- ipv6IcmpGroups      OBJECT IDENTIFIER ::= { ipv6IcmpConformance 2 }


-- compliance statements

ipMIBCompliance MODULE-COMPLIANCE
    STATUS     deprecated
    DESCRIPTION
           "The compliance statement for systems which implement only
            IPv4.  For version-independence, this compliance statement
            is deprecated in favor of ipMIBCompliance2."
    MODULE  -- this module
        MANDATORY-GROUPS { ipGroup,
                           icmpGroup }
    ::= { ipMIBCompliances 1 }

-- ipv6Compliance MODULE-COMPLIANCE
-- .ST c
-- .(D
-- "The compliance statement for systems which
-- implement ipv6 MIB."
-- .)D
--             MODULE  -- -- this module
--                 MANDATORY-GROUPS { ipv6GeneralGroup,
--                                    ipv6NotificationGroup }
--                   OBJECT    ipv6Forwarding
--                     MIN-ACCESS  read-only
-- .(D
-- "An agent is not required to provide write
-- access to this object"
-- .)D
--                   OBJECT    ipv6DefaultHopLimit
--                     MIN-ACCESS  read-only
-- .(D

-- "An agent is not required to provide write
-- access to this object"
-- .)D
--                   OBJECT    ipv6IfDescr
--                     MIN-ACCESS  read-only
-- .(D
-- "An agent is not required to provide write
-- access to this object"
-- .)D
```

```
--                  OBJECT    ipv6IfIdentifier
--                    MIN-ACCESS  read-only
-- .(D
-- "An agent is not required to provide write
-- access to this object"
-- .)D
--                  OBJECT    ipv6IfIdentifierLength
--                    MIN-ACCESS  read-only
-- .(D
-- "An agent is not required to provide write
-- access to this object"
-- .)D
--
--                  OBJECT    ipv6IfAdminStatus
--                    MIN-ACCESS  read-only
-- .(D
-- "An agent is not required to provide write
-- access to this object"
-- .)D
--                  OBJECT    ipv6RouteValid
--                    MIN-ACCESS  read-only
-- .(D
-- "An agent is not required to provide write
-- access to this object"
-- .)D
--                  OBJECT    ipv6NetToMediaValid
--                    MIN-ACCESS  read-only
-- .(D
-- "An agent is not required to provide write
-- access to this object"
-- .)D
--            ::= { ipv6Compliances 1 }


-- units of conformance

ipGroup2 OBJECT-GROUP
    OBJECTS   { ipForwarding, ipDefaultTTL }
    STATUS     current
    DESCRIPTION
           "The group of IPv4-specific objects for basic management of
            IPv4 entities."
    ::= { ipMIBGroups 3 }

ipIfStatsGroup OBJECT-GROUP
    OBJECTS   { ipIfStatsInReceives, ipIfStatsInHdrErrors,
                                ipIfStatsInTooBigErrors, ipIfStatsInNoRoutes,
                                ipIfStatsInAddrErrors, ipIfStatsInUnknownProtos,
                                ipIfStatsInTruncatedPkts, ipIfStatsInDiscards,
                                ipIfStatsInDelivers, ipIfStatsOutForwDatagrams,
                                ipIfStatsOutRequests, ipIfStatsOutDiscards,
                                ipIfStatsOutFragOKs, ipIfStatsOutFragFails,
                                ipIfStatsOutFragCreates, ipIfStatsReasmReqds,
                                ipIfStatsReasmOKs, ipIfStatsReasmFails,
                                ipIfStatsInMcastPkts, ipIfStatsOutMcastPkts,
                                ipIfStatsInOctets, ipIfStatsOutOctets,
                                ipIfStatsInBcastPkts, ipIfStatsOutBcastPkts,
                                ipIfStatsInMcastOctets, ipIfStatsOutMcastOctets }
    STATUS     current
    DESCRIPTION
           "IP per-interface or per-system statistics."
    ::= { ipMIBGroups 4 }

-- XXX some HC statistics groups

ipv6ScopeGroup OBJECT-GROUP
    OBJECTS   { ipv6ScopeIdLinkLocal, ipv6ScopeIdSubnetLocal,
                ipv6ScopeIdAdminLocal, ipv6ScopeIdSiteLocal,
                                ipv6ScopeId6, ipv6ScopeId7,
                                ipv6ScopeIdOrganizationLocal, ipv6ScopeId9,
                                ipv6ScopeIdA, ipv6ScopeIdB,
                                ipv6ScopeIdC, ipv6ScopeIdD }
    STATUS     current
    DESCRIPTION
           "The group of objects for managing IPv6 scope zones."
    ::= { ipMIBGroups 5 }

ipGroup OBJECT-GROUP
    OBJECTS   { ipForwarding, ipDefaultTTL, ipInReceives,
                ipInHdrErrors, ipInAddrErrors,
                ipForwDatagrams, ipInUnknownProtos,
                ipInDiscards, ipInDelivers, ipOutRequests,
                ipOutDiscards, ipOutNoRoutes,
```

```
                ipReasmTimeout, ipReasmReqds, ipReasmOKs,
                ipReasmFails, ipFragOKs,
                ipFragFails, ipFragCreates,
                ipAdEntAddr, ipAdEntIfIndex, ipAdEntNetMask,
                ipAdEntBcastAddr, ipAdEntReasmMaxSize,
                ipNetToMediaIfIndex, ipNetToMediaPhysAddress,
                ipNetToMediaNetAddress, ipNetToMediaType,
                ipRoutingDiscards }
    STATUS     deprecated
    DESCRIPTION
           "The ip group of objects providing for basic management of IP
            entities, exclusive of the management of IP routes."
    ::= { ipMIBGroups 1 }

icmpGroup OBJECT-GROUP
    OBJECTS   { icmpInMsgs, icmpInErrors,
                icmpInDestUnreachs, icmpInTimeExcds,
                icmpInParmProbs, icmpInSrcQuenchs,
                icmpInRedirects, icmpInEchos,
                icmpInEchoReps, icmpInTimestamps,
                icmpInTimestampReps, icmpInAddrMasks,
                icmpInAddrMaskReps, icmpOutMsgs,
                icmpOutErrors, icmpOutDestUnreachs,
                icmpOutTimeExcds, icmpOutParmProbs,
                icmpOutSrcQuenchs, icmpOutRedirects,
                icmpOutEchos, icmpOutEchoReps,
                icmpOutTimestamps, icmpOutTimestampReps,
                icmpOutAddrMasks, icmpOutAddrMaskReps }
    STATUS     deprecated
    DESCRIPTION
           "The icmp group of objects providing ICMP statistics."
    ::= { ipMIBGroups 2 }

ipv6GeneralGroup2 OBJECT-GROUP
    OBJECTS { ipv6Forwarding,
              ipv6DefaultHopLimit }
    STATUS     current
    DESCRIPTION
           "The IPv6 group of objects providing for basic management of
            IPv6 entities."
    ::= { ipv6Groups 3 }


END
```

A.2 draft-ietf-ipngwg-RFC2012-update-01

```
TCP-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, Integer32, Unsigned32,
    Gauge32, Counter32, Counter64, IpAddress, mib-2
                                        FROM SNMPv2-SMI
    TimeStamp                           FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP     FROM SNMPv2-CONF
    InetAddress, InetAddressType,
    InetPortNumber                      FROM INET-ADDRESS-MIB;

tcpMIB MODULE-IDENTITY
    LAST-UPDATED "200111140000Z"
    ORGANIZATION "IETF IPv6 MIB Revision Team"
    CONTACT-INFO
           "Bill Fenner (editor)
 AT&T Labs -- Research
            75 Willow Rd.
            Menlo Park, CA 94025

            Phone: +1 650 330-7893
            Email: <fenner@research.att.com>"
    DESCRIPTION
           "The MIB module for managing TCP implementations."
    REVISION      "200111140000Z"
    DESCRIPTION
           "IP version neutral revision, published as RFC XXXX."
    REVISION      "9411010000Z"
    DESCRIPTION
           "Initial SMIv2 version, published as RFC 2012."
    REVISION      "9103310000Z"
    DESCRIPTION
           "The initial revision of this MIB module was part of MIB-II."
    ::= { mib-2 49 }


-- the TCP base variables group

tcp      OBJECT IDENTIFIER ::= { mib-2 6 }

-- Scalars

tcpRtoAlgorithm OBJECT-TYPE
    SYNTAX      INTEGER {
                    other(1),    -- none of the following
                    constant(2), -- a constant rto
                    rsre(3),     -- MIL-STD-1778, Appendix B
                    vanj(4)      -- Van Jacobson's algorithm [1]
                }

 MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The algorithm used to determine the timeout value used for
            retransmitting unacknowledged octets."
    ::= { tcp 1 }

tcpRtoMin OBJECT-TYPE
    SYNTAX     Integer32
    UNITS       "milliseconds"
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The minimum value permitted by a TCP implementation for the
            retransmission timeout, measured in milliseconds.  More
            refined semantics for objects of this type depend upon the
            algorithm used to determine the retransmission timeout.  In
            particular, when the timeout algorithm is rsre(3), an object
            of this type has the semantics of the LBOUND quantity
            described in RFC 793."
    ::= { tcp 2 }

tcpRtoMax OBJECT-TYPE
    SYNTAX     Integer32
    UNITS       "milliseconds"
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The maximum value permitted by a TCP implementation for the
            retransmission timeout, measured in milliseconds.  More
            refined semantics for objects of this type depend upon the
            algorithm used to determine the retransmission timeout.  In
```

```
            particular, when the timeout algorithm is rsre(3), an object
            of this type has the semantics of the UBOUND quantity
            described in RFC 793."
    ::= { tcp 3 }

tcpMaxConn OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The limit on the total number of TCP connections the entity
            can support.  In entities where the maximum number of
            connections is dynamic, this object should contain the value
            -1."
    ::= { tcp 4 }


tcpActiveOpens OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of times TCP connections have made a direct
            transition to the SYN-SENT state from the CLOSED state."
    ::= { tcp 5 }

tcpPassiveOpens OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of times TCP connections have made a direct
            transition to the SYN-RCVD state from the LISTEN state."
    ::= { tcp 6 }

tcpAttemptFails OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of times TCP connections have made a direct
            transition to the CLOSED state from either the SYN-SENT
            state or the SYN-RCVD state, plus the number of times TCP
            connections have made a direct transition to the LISTEN
            state from the SYN-RCVD state."
    ::= { tcp 7 }

tcpEstabResets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of times TCP connections have made a direct
            transition to the CLOSED state from either the ESTABLISHED
            state or the CLOSE-WAIT state."
    ::= { tcp 8 }

tcpCurrEstab OBJECT-TYPE
    SYNTAX     Gauge32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of TCP connections for which the current state is
            either ESTABLISHED or CLOSE-WAIT."
    ::= { tcp 9 }

tcpInSegs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of segments received, including those
            received in error.  This count includes segments received on
            currently established connections."
    ::= { tcp 10 }

tcpOutSegs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of segments sent, including those on
            current connections but excluding those containing only
            retransmitted octets."
```

```
    ::= { tcp 11 }

tcpRetransSegs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of segments retransmitted - that is, the
            number of TCP segments transmitted containing one or more
            previously transmitted octets."
    ::= { tcp 12 }

tcpInErrs OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of segments received in error (e.g., bad
            TCP checksums)."
    ::= { tcp 14 }

tcpOutRsts OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of TCP segments sent containing the RST flag."
    ::= { tcp 15 }

tcpHCInSegs OBJECT-TYPE
 SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of segments received, including those
            received in error, on systems that can receive more than 1
            million TCP packets per second.  This count includes
            segments received on currently established connections."
    ::= { tcp 17 }

tcpHCOutSegs OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of segments sent, including those on
            current connections but excluding those containing only
            retransmitted octets, on systems that can transmit more than
            1 million TCP packets per second."
    ::= { tcp 18 }

-- The TCP Connection table

tcpConnectionTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF TcpConnectionEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "A table containing information about existing TCP
            connections.  Note that unlike earlier TCP MIBs, there is a
            seperate table for connections in the LISTEN state."
    ::= { tcp 19 }

tcpConnectionEntry OBJECT-TYPE
    SYNTAX     TcpConnectionEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "A conceptual row of the tcpConnectionTable containing
            information about a particular current TCP connection.  Each
            row of this table is transient, in that it ceases to exist
            when (or soon after) the connection makes the transition to
            the CLOSED state."
    INDEX   { tcpConnectionLocalAddressType,
              tcpConnectionLocalAddress,
              tcpConnectionLocalPort,
              tcpConnectionRemAddress,
              tcpConnectionRemPort }
    ::= { tcpConnectionTable 1 }

TcpConnectionEntry ::= SEQUENCE {
        tcpConnectionLocalAddressType  InetAddressType,
        tcpConnectionLocalAddress      InetAddress,
        tcpConnectionLocalPort         InetPortNumber,
```

```
        tcpConnectionRemAddress        InetAddress,
        tcpConnectionRemPort           InetPortNumber,
        tcpConnectionState             INTEGER,
        tcpConnectionInPackets         Counter32,
        tcpConnectionOutPackets        Counter32,
        tcpConnectionInOctets          Counter32,
        tcpConnectionOutOctets         Counter32,
        tcpConnectionHCInPackets       Counter64,
        tcpConnectionHCOutPackets      Counter64,
        tcpConnectionHCInOctets        Counter64,
        tcpConnectionHCOutOctets       Counter64,
        tcpConnectionStartTime         TimeStamp,
        tcpConnectionProcessID         Unsigned32
    }

tcpConnectionLocalAddressType OBJECT-TYPE
    SYNTAX     InetAddressType
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The address type of tcpConnectionLocalAddress.  Only IPv4
            and IPv6 addresses are expected."
    ::= { tcpConnectionEntry 1 }

tcpConnectionLocalAddress OBJECT-TYPE
    SYNTAX     InetAddress (SIZE(0..36))
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The local IP address for this TCP connection.  In the case
            of a connection in the listen state which is willing to
            accept connections for any IP interface associated with the
            node, a value of all zeroes is used."
    ::= { tcpConnectionEntry 2 }

tcpConnectionLocalPort OBJECT-TYPE
    SYNTAX     InetPortNumber
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The local port number for this TCP connection."
    ::= { tcpConnectionEntry 3 }
tcpConnectionRemAddress OBJECT-TYPE
    SYNTAX     InetAddress (SIZE(0..36))
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The remote IP address for this TCP connection."
    ::= { tcpConnectionEntry 4 }

tcpConnectionRemPort OBJECT-TYPE
    SYNTAX     InetPortNumber
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The remote port number for this TCP connection."
    ::= { tcpConnectionEntry 5 }

tcpConnectionState OBJECT-TYPE
    SYNTAX     INTEGER {
                    closed(1),
                    listen(2),
                    synSent(3),
                    synReceived(4),
                    established(5),
                    finWait1(6),
                    finWait2(7),
                    closeWait(8),
                    lastAck(9),
                    closing(10),
                    timeWait(11),
                    deleteTCB(12)
                }
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
           "The state of this TCP connection.

            The value listen(2) is included only for parallelism to the
            old tcpConnTable, and should not be used.  A connection in
            LISTEN state should be present in the tcpListenerTable.

            The only value which may be set by a management station is
            deleteTCB(12).  Accordingly, it is appropriate for an agent
```

```
            to return a 'badValue' response if a management station
            attempts to set this object to any other value.

            If a management station sets this object to the value
            deleteTCB(12), then this has the effect of deleting the TCB
            (as defined in RFC 793) of the corresponding connection on
            the managed node, resulting in immediate termination of the
            connection.

            As an implementation-specific option, a RST segment may be
            sent from the managed node to the other TCP endpoint (note
            however that RST segments are not sent reliably)."
    ::= { tcpConnectionEntry 6 }

tcpConnectionInPackets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of packets received on this connection.  This
            count includes retransmitted data."
    ::= { tcpConnectionEntry 7 }

tcpConnectionOutPackets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of packets transmitted on this connection.  This
            count includes retransmitted data."
    ::= { tcpConnectionEntry 8 }

tcpConnectionInOctets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of octets received on this connection.  This
            count includes retransmitted data."
    ::= { tcpConnectionEntry 9 }

tcpConnectionOutOctets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of octets transmitted on this connection.  This
            count includes retransmitted data."
    ::= { tcpConnectionEntry 10 }

tcpConnectionHCInPackets OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
 DESCRIPTION
           "The number of packets received on this connection.  This
            count includes retransmitted data."
    ::= { tcpConnectionEntry 11 }

tcpConnectionHCOutPackets OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of packets transmitted on this connection.  This
            count includes retransmitted data."
    ::= { tcpConnectionEntry 12 }

tcpConnectionHCInOctets OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of octets received on this connection.  This
            count includes retransmitted data."
    ::= { tcpConnectionEntry 13 }

tcpConnectionHCOutOctets OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of octets transmitted on this connection.  This
            count includes retransmitted data."
```

```
    ::= { tcpConnectionEntry 14 }

tcpConnectionStartTime OBJECT-TYPE
    SYNTAX     TimeStamp
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The value of sysUpTime at the time this connection was
            established"
    ::= { tcpConnectionEntry 15 }

tcpConnectionProcessID OBJECT-TYPE
    SYNTAX     Unsigned32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The system's process ID for the process associated with this
            connection, or zero if there is no such process.  This value
            is expected to be the same as HOST-RESOURCES-
            MIB::hrSWRunIndex or SYSAPPL-MIB::sysApplElmtRunIndex for
            some row in the appropriate tables."
    ::= { tcpConnectionEntry 16 }



-- The TCP Listener table

tcpListenerTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF TcpListenerEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "A table containing information about TCP listeners."
    ::= { tcp 20 }

tcpListenerEntry OBJECT-TYPE
    SYNTAX     TcpListenerEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "A conceptual row of the tcpListenerTable containing
            information about a particular TCP listener."
    INDEX   { tcpListenerLocalAddressType,
              tcpListenerLocalAddress,
              tcpListenerLocalPort,
              tcpListenerRemAddressType }
    ::= { tcpListenerTable 1 }

TcpListenerEntry ::= SEQUENCE {
        tcpListenerLocalAddressType       InetAddressType,
        tcpListenerLocalAddress           InetAddress,
        tcpListenerLocalPort              InetPortNumber,
        tcpListenerRemAddressType         InetAddressType,
        tcpListenerConnectionsTimedOut    Counter32,
        tcpListenerHCConnectionsTimedOut  Counter64,
        tcpListenerConnectionsAccepted    Counter32,
        tcpListenerHCConnectionsAccepted  Counter64,
        tcpListenerStartTime              TimeStamp,
        tcpListenerProcessID              Unsigned32
    }

tcpListenerLocalAddressType OBJECT-TYPE
    SYNTAX     InetAddressType
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The address type of tcpListenerLocalAddress.  Only IPv4 and
            IPv6 addresses are expected."
    ::= { tcpListenerEntry 1 }

tcpListenerLocalAddress OBJECT-TYPE
    SYNTAX     InetAddress (SIZE(0..36))
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The local IP address for this TCP connection.  In the case
            of a connection in the listen state which is willing to
            accept connections for any IP interface associated with the
            node, a value of all zeroes is used."
    ::= { tcpListenerEntry 2 }

tcpListenerLocalPort OBJECT-TYPE
    SYNTAX     InetPortNumber
    MAX-ACCESS not-accessible
```

```
    STATUS     current
    DESCRIPTION
           "The local port number for this TCP connection."
    ::= { tcpListenerEntry 3 }

tcpListenerRemAddressType OBJECT-TYPE
    SYNTAX     InetAddressType
MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The address type of connections that will be accepted by
            this listener.  Only IPv4 and IPv6 addresses are expected,
            or unknown to indicate an endpoint willing to accept both
            IPv4 and IPv6 connections."
    ::= { tcpListenerEntry 4 }

tcpListenerConnectionsTimedOut OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of connection attempts to this endpoint which
            have failed due to timeout of the three-way handshake, i.e.
            the row was removed from the tcpConnectionTable but
            tcpConnectionState never moved from synReceived to
            established."
    ::= { tcpListenerEntry 5 }

tcpListenerHCConnectionsTimedOut OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of connection attempts to this endpoint which
            have failed due to timeout of the three-way handshake, i.e.
            the row was removed from the tcpConnectionTable but
            tcpConnectionState never moved from synReceived to
            established."
    ::= { tcpListenerEntry 6 }

tcpListenerConnectionsAccepted OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of connections which have been established to
            this endpoint."
    ::= { tcpListenerEntry 7 }

tcpListenerHCConnectionsAccepted OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of connections which have been established to
            this endpoint."
    ::= { tcpListenerEntry 8 }

tcpListenerStartTime OBJECT-TYPE
    SYNTAX     TimeStamp
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The value of sysUpTime at the time this listener was
            established."
    ::= { tcpListenerEntry 9 }

tcpListenerProcessID OBJECT-TYPE
    SYNTAX     Unsigned32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The system's process ID for the process associated with this
            listener, or zero if there is no such process.  This value
            is expected to be the same as HOST-RESOURCES-
            MIB::hrSWRunIndex or SYSAPPL-MIB::sysApplElmtRunIndex for
            some row in the appropriate tables."
    ::= { tcpListenerEntry 10 }
-- The deprecated TCP Connection table

tcpConnTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF TcpConnEntry
    MAX-ACCESS not-accessible
    STATUS     deprecated
```

```
    DESCRIPTION
           "A table containing information about existing IPv4-specific
            TCP connections or listeners.  This table has been
            deprecated in favor of the version neutral
            tcpConnectionTable."
    ::= { tcp 13 }

tcpConnEntry OBJECT-TYPE
    SYNTAX     TcpConnEntry
    MAX-ACCESS not-accessible
    STATUS     deprecated
    DESCRIPTION
           "A conceptual row of the tcpConnTable containing information
            about a particular current IPv4 TCP connection.  Each row of
            this table is transient, in that it ceases to exist when (or
            soon after) the connection makes the transition to the
            CLOSED state."
    INDEX   { tcpConnLocalAddress,
              tcpConnLocalPort,
              tcpConnRemAddress,
              tcpConnRemPort }
    ::= { tcpConnTable 1 }

TcpConnEntry ::= SEQUENCE {
        tcpConnState         INTEGER,
        tcpConnLocalAddress  IpAddress,
        tcpConnLocalPort     INTEGER,
        tcpConnRemAddress    IpAddress,
        tcpConnRemPort       INTEGER
    }

tcpConnState OBJECT-TYPE
    SYNTAX     INTEGER {
                    closed(1),
                    listen(2),
                    synSent(3),
                    synReceived(4),
                    established(5),
                    finWait1(6),
                    finWait2(7),
                    closeWait(8),
                    lastAck(9),
                    closing(10),
                    timeWait(11),
                    deleteTCB(12)
                }
    MAX-ACCESS read-write
    STATUS     deprecated
    DESCRIPTION
           "The state of this TCP connection.

            The only value which may be set by a management station is
            deleteTCB(12).  Accordingly, it is appropriate for an agent
            to return a 'badValue' response if a management station
            attempts to set this object to any other value.

            If a management station sets this object to the value
            deleteTCB(12), then this has the effect of deleting the TCB
            (as defined in RFC 793) of the corresponding connection on
            the managed node, resulting in immediate termination of the
            connection.

            As an implementation-specific option, a RST segment may be
            sent from the managed node to the other TCP endpoint (note
            however that RST segments are not sent reliably)."
    ::= { tcpConnEntry 1 }

tcpConnLocalAddress OBJECT-TYPE
    SYNTAX     IpAddress
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The local IP address for this TCP connection.  In the case
            of a connection in the listen state which is willing to
            accept connections for any IP interface associated with the
            node, the value 0.0.0.0 is used."
    ::= { tcpConnEntry 2 }

tcpConnLocalPort OBJECT-TYPE
    SYNTAX     INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The local port number for this TCP connection."
```

```
    ::= { tcpConnEntry 3 }

tcpConnRemAddress OBJECT-TYPE
    SYNTAX     IpAddress
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The remote IP address for this TCP connection."
    ::= { tcpConnEntry 4 }

tcpConnRemPort OBJECT-TYPE
    SYNTAX     INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The remote port number for this TCP connection."
    ::= { tcpConnEntry 5 }



-- conformance information

tcpMIBConformance OBJECT IDENTIFIER ::= { tcpMIB 2 }

tcpMIBCompliances OBJECT IDENTIFIER ::= { tcpMIBConformance 1 }
tcpMIBGroups      OBJECT IDENTIFIER ::= { tcpMIBConformance 2 }


-- compliance statements

tcpMIBCompliance2 MODULE-COMPLIANCE
    STATUS     current
    DESCRIPTION
           "The compliance statement for systems which implement TCP."
    MODULE  -- this module
        MANDATORY-GROUPS { tcpBaseGroup, tcpConnectionGroup, tcpListenerGroup }
        GROUP       tcpHCGroup
        DESCRIPTION
           "This group is mandatory for those systems which are capable
                of receiving or transmitting more than 1 million TCP
                packets per second.  1 million packets per second will
                cause a Counter32 to wrap in just over an hour."
        GROUP       tcpStatisticsGroup
        DESCRIPTION
           "This group is optional.  It provides visibility for counters
                that some systems already implement."
        GROUP       tcpHCStatisticsGroup
        DESCRIPTION
           "This group is mandatory for those systems which implement
                the tcpStatisticsGroup and are capable of receiving or
                transmitting more than 1 million TCP packets per second.
                1 million packets per second will cause a Counter32 to
                wrap in just over an hour."
        OBJECT      tcpConnectionState
        MIN-ACCESS  read-only
        DESCRIPTION
           "Write access is not required."
    ::= { tcpMIBCompliances 2 }

tcpMIBCompliance MODULE-COMPLIANCE
    STATUS     deprecated
    DESCRIPTION
           "The compliance statement for IPv4-only systems which
            implement TCP.  In order to be IP version independent, this
            compliance statement is deprecated in favor of
            tcpMIBCompliance2.  However, agents are still encouraged to
            implement these objects in order to interoperate with the
            deployed base of managers."
    MODULE  -- this module
        MANDATORY-GROUPS { tcpGroup }
        OBJECT      tcpConnState
        MIN-ACCESS  read-only
        DESCRIPTION
           "Write access is not required."
    ::= { tcpMIBCompliances 1 }

-- units of conformance

tcpGroup OBJECT-GROUP
    OBJECTS   { tcpRtoAlgorithm, tcpRtoMin, tcpRtoMax,
                tcpMaxConn, tcpActiveOpens,
                tcpPassiveOpens, tcpAttemptFails,
                tcpEstabResets, tcpCurrEstab, tcpInSegs,
```

```
                tcpOutSegs, tcpRetransSegs, tcpConnState,
                tcpConnLocalAddress, tcpConnLocalPort,
                tcpConnRemAddress, tcpConnRemPort,
                tcpInErrs, tcpOutRsts }
    STATUS     deprecated
    DESCRIPTION
           "The tcp group of objects providing for management of TCP
            entities."
    ::= { tcpMIBGroups 1 }

tcpBaseGroup OBJECT-GROUP
    OBJECTS   { tcpRtoAlgorithm, tcpRtoMin, tcpRtoMax,
                tcpMaxConn, tcpActiveOpens,
                tcpPassiveOpens, tcpAttemptFails,
                tcpEstabResets, tcpCurrEstab, tcpInSegs,
                tcpOutSegs, tcpRetransSegs,
                tcpInErrs, tcpOutRsts }
    STATUS     current
    DESCRIPTION
           "The group of counters common to TCP entities."
    ::= { tcpMIBGroups 2 }

tcpHCGroup OBJECT-GROUP
    OBJECTS    { tcpHCInSegs, tcpHCOutSegs }
    STATUS     current
    DESCRIPTION
           "The group of objects providing for counters of high speed
            TCP implementations."
    ::= { tcpMIBGroups 3 }

tcpConnectionGroup OBJECT-GROUP
    OBJECTS   { tcpConnectionState }
    STATUS     current
    DESCRIPTION
           "The table of TCP connections."
    ::= { tcpMIBGroups 4 }

tcpListenerGroup OBJECT-GROUP
    OBJECTS    { tcpListenerRemAddressType }
    STATUS     current
    DESCRIPTION
           "The table of TCP listeners."
    ::= { tcpMIBGroups 5 }

tcpStatisticsGroup OBJECT-GROUP
    OBJECTS    { tcpConnectionInPackets, tcpConnectionOutPackets,
                 tcpConnectionInOctets, tcpConnectionOutOctets,
                             tcpConnectionStartTime, tcpConnectionProcessID,
                 tcpListenerConnectionsTimedOut,
                 tcpListenerConnectionsAccepted,
                 tcpListenerStartTime, tcpListenerProcessID }
    STATUS     current
    DESCRIPTION
           "The packet and octet counters and other statistics specific
            to a TCP connection or listener."
    ::= { tcpMIBGroups 6 }

tcpHCStatisticsGroup OBJECT-GROUP
    OBJECTS    { tcpConnectionHCInPackets, tcpConnectionHCOutPackets,
                 tcpConnectionHCInOctets, tcpConnectionHCOutOctets,
                 tcpListenerHCConnectionsTimedOut,
                 tcpListenerHCConnectionsAccepted }
    STATUS     current
    DESCRIPTION
           "The group of objects providing for statistics for listeners
            or connections on high speed TCP implementations."
    ::= { tcpMIBGroups 7 }


END
```

A.3 draft-ietf-ipngwg-RFC2013-update-01

```
UDP-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, Counter32, Counter64, Unsigned32,
    IpAddress, mib-2                    FROM SNMPv2-SMI
    MODULE-COMPLIANCE, OBJECT-GROUP     FROM SNMPv2-CONF
    InetAddress, InetAddressType,
    InetPortNumber                      FROM INET-ADDRESS-MIB;


udpMIB MODULE-IDENTITY
    LAST-UPDATED "200111150000Z"
    ORGANIZATION "IETF IPv6 MIB Revision Team"
    CONTACT-INFO
           "Bill Fenner (editor)

            AT&T Labs -- Research
            75 Willow Rd.
            Menlo Park, CA 94025

            Phone: +1 650 330-7893
            Email: <fenner@research.att.com>"
    DESCRIPTION
           "The MIB module for managing UDP implementations."
    REVISION      "200111150000Z"
    DESCRIPTION
           "IP version neutral revision, published as RFC XXXX."
    REVISION      "9411010000Z"
    DESCRIPTION
           "Initial SMIv2 version, published as RFC 2013."
    REVISION      "9103310000Z"
    DESCRIPTION
           "The initial revision of this MIB module was part of MIB-II."
    ::= { mib-2 50 }

-- the UDP group

udp      OBJECT IDENTIFIER ::= { mib-2 7 }

udpInDatagrams OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of UDP datagrams delivered to UDP users."
    ::= { udp 1 }

udpNoPorts OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of received UDP datagrams for which there
            was no application at the destination port."
    ::= { udp 2 }

udpInErrors OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of received UDP datagrams that could not be
            delivered for reasons other than the lack of an application
            at the destination port."
    ::= { udp 3 }

udpOutDatagrams OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of UDP datagrams sent from this entity."
    ::= { udp 4 }

udpHCInDatagrams OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of UDP datagrams delivered to UDP users,
            for devices which can receive more than 1 million UDP
            packets per second."
```

```
    ::= { udp 26 }

udpHCOutDatagrams OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The total number of UDP datagrams sent from this entity, for
            devices which can transmit more than 1 million UDP packets
            per second."
    ::= { udp 27 }

-- The UDP Listener table

udpListenerTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF UdpListenerEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "A table containing UDP information about this entity's UDP
            endpoints on which a local application is currently
            accepting or sending datagrams.

            Note that despite the name, UDP listeners are not restricted
            to receiving packets; they may also send packets sourced
            from the listening address and port.
            An entity may have multiple applications listening to the
            same UDP endpoint; these are discriminated using the
            udpListenerInstance object."
    ::= { udp 7 }

udpListenerEntry OBJECT-TYPE
    SYNTAX     UdpListenerEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "Information about a particular current UDP listener."
    INDEX   { udpListenerLocalAddressType,
              udpListenerLocalAddress,
              udpListenerLocalPort,
              udpListenerRemoteAddressType,
              udpListenerInstance }
    ::= { udpListenerTable 1 }

UdpListenerEntry ::= SEQUENCE {
        udpListenerLocalAddressType   InetAddressType,
        udpListenerLocalAddress       InetAddress,
        udpListenerLocalPort          InetPortNumber,
        udpListenerRemoteAddressType  InetAddressType,
        udpListenerInstance           Unsigned32,
        udpListenerInPackets          Counter32,
        udpListenerHCInPackets        Counter64,
        udpListenerOutPackets         Counter32,
        udpListenerHCOutPackets       Counter64,
        udpListenerInOctets           Counter32,
        udpListenerHCInOctets         Counter64,
        udpListenerOutOctets          Counter32,
        udpListenerHCOutOctets        Counter64,
        udpListenerProcessID          Unsigned32
    }


udpListenerLocalAddressType OBJECT-TYPE
    SYNTAX     InetAddressType
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The address type of udpListenerLocalAddress.  Only IPv4 and
            IPv6 addresses are expected."
    ::= { udpListenerEntry 1 }

udpListenerLocalAddress OBJECT-TYPE
    SYNTAX     InetAddress (SIZE(0..36))
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The local IP address for this UDP listener.  In the case of
            a UDP listener which is willing to accept datagrams for any
            IP interface associated with the node, a value of all zeroes
            is used."
    ::= { udpListenerEntry 2 }

udpListenerLocalPort OBJECT-TYPE
    SYNTAX     InetPortNumber
```

```
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The local port number for this UDP listener."
    ::= { udpListenerEntry 3 }

udpListenerRemoteAddressType OBJECT-TYPE
    SYNTAX     InetAddressType
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The address type of packets that will be accepted by this
            listener.  Only IPv4 and IPv6 addresses are expected, or
            unknown to indicate an endpoint willing to accept both IPv4
            and IPv6 packets."
    ::= { udpListenerEntry 4 }

udpListenerInstance OBJECT-TYPE
    SYNTAX     Unsigned32 (1..'ffffffff'h)
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The instance of this tuple.  This object is used to
            distinguish between multiple processes listening on the same
            UDP endpoint."
    ::= { udpListenerEntry 5 }

udpListenerInPackets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of packets received for this listener."
    ::= { udpListenerEntry 6 }

udpListenerHCInPackets OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of packets received for this listener."
    ::= { udpListenerEntry 7 }

udpListenerOutPackets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of packets sent by this listener."
    ::= { udpListenerEntry 8 }

udpListenerHCOutPackets OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of packets sent by this listener."
    ::= { udpListenerEntry 9 }

udpListenerInOctets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of octets received for this listener."
    ::= { udpListenerEntry 10 }

udpListenerHCInOctets OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of octets received for this listener."
    ::= { udpListenerEntry 11 }

udpListenerOutOctets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of octets sent by this listener."
    ::= { udpListenerEntry 12 }

udpListenerHCOutOctets OBJECT-TYPE
```

```
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of octets sent by this listener."
    ::= { udpListenerEntry 13 }

udpListenerProcessID OBJECT-TYPE
    SYNTAX     Unsigned32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The system's process ID for the process associated with this
            listener, or zero if there is no such process.  This value
            is expected to be the same as HOST-RESOURCES-
            MIB::hrSWRunIndex or SYSAPPL-MIB::sysApplElmtRunIndex for
            some row in the appropriate tables."
    ::= { udpListenerEntry 14 }



-- The UDP "Connection" table.

udpConnectionTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF UdpConnectionEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "A table containing UDP 'connection' information.

            Although UDP is a connectionless protocol, a system might
            demultiplex UDP packets based upon the local and remote
            addresses and ports."
    ::= { udp 8 }

udpConnectionEntry OBJECT-TYPE
    SYNTAX     UdpConnectionEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "Information about a particular current UDP connection."
    INDEX   { udpConnectionLocalAddressType,
              udpConnectionLocalAddress,
              udpConnectionLocalPort,
              udpConnectionRemoteAddress,
              udpConnectionRemotePort,
              udpConnectionInstance }
    ::= { udpConnectionTable 1 }

UdpConnectionEntry ::= SEQUENCE {
        udpConnectionLocalAddressType  InetAddressType,
        udpConnectionLocalAddress      InetAddress,
        udpConnectionLocalPort         InetPortNumber,
        udpConnectionRemoteAddress     InetAddress,
        udpConnectionRemotePort        InetPortNumber,
        udpConnectionInstance          Unsigned32,
        udpConnectionInPackets         Counter32,
        udpConnectionHCInPackets       Counter64,
        udpConnectionOutPackets        Counter32,
        udpConnectionHCOutPackets      Counter64,
        udpConnectionInOctets          Counter32,
        udpConnectionHCInOctets        Counter64,
        udpConnectionOutOctets         Counter32,
        udpConnectionHCOutOctets       Counter64,
        udpConnectionProcessID         Unsigned32
    }


udpConnectionLocalAddressType OBJECT-TYPE
    SYNTAX     InetAddressType
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The address type of udpConnectionLocalAddress.  Only IPv4
            and IPv6 addresses are expected."
    ::= { udpConnectionEntry 1 }

udpConnectionLocalAddress OBJECT-TYPE
    SYNTAX     InetAddress (SIZE(0..36))
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The local IP address for this UDP connection."
    ::= { udpConnectionEntry 2 }
```

```
udpConnectionLocalPort OBJECT-TYPE
    SYNTAX     InetPortNumber
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The local port number for this UDP connection."
    ::= { udpConnectionEntry 3 }

udpConnectionRemoteAddress OBJECT-TYPE
    SYNTAX     InetAddress (SIZE(0..36))
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The remote IP address for this UDP connection.  It has the
            same type as udpConnectionLocalAddress."
    ::= { udpConnectionEntry 4 }

udpConnectionRemotePort OBJECT-TYPE
    SYNTAX     InetPortNumber
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The remote port number for this UDP connection."
    ::= { udpConnectionEntry 5 }

udpConnectionInstance OBJECT-TYPE
    SYNTAX     Unsigned32 (1..'ffffffff'h)
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The instance of this tuple.  This object is used to
            distinguish between multiple processes 'connected' to the
            same UDP endpoint."
    ::= { udpConnectionEntry 6 }

udpConnectionInPackets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of packets received for this connection."
    ::= { udpConnectionEntry 7 }

udpConnectionHCInPackets OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of packets received for this connection."
    ::= { udpConnectionEntry 8 }

udpConnectionOutPackets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of packets sent on this connection."
    ::= { udpConnectionEntry 9 }

udpConnectionHCOutPackets OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of packets sent on this connection."
    ::= { udpConnectionEntry 10 }

udpConnectionInOctets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of octets received for this connection."
    ::= { udpConnectionEntry 11 }

udpConnectionHCInOctets OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of octets received for this connection."
    ::= { udpConnectionEntry 12 }
```

```
udpConnectionOutOctets OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of octets sent on this connection."
    ::= { udpConnectionEntry 13 }

udpConnectionHCOutOctets OBJECT-TYPE
    SYNTAX     Counter64
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The count of octets sent on this connection."
    ::= { udpConnectionEntry 14 }

udpConnectionProcessID OBJECT-TYPE
    SYNTAX     Unsigned32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The system's process ID for the process associated with this
            connection, or zero if there is no such process.  This value
            is expected to be the same as HOST-RESOURCES-
            MIB::hrSWRunIndex or SYSAPPL-MIB::sysApplElmtRunIndex for
            some row in the appropriate tables."
    ::= { udpConnectionEntry 15 }



-- The deprecated UDP Listener table

-- The UDP listener table contains information about this
-- entity's IPv4 UDP end-points on which a local application is
-- currently accepting datagrams.

udpTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF UdpEntry
    MAX-ACCESS not-accessible
    STATUS     deprecated
    DESCRIPTION
           "A table containing IPv4-specific UDP listener information.
            It contains information about all local IPv4 UDP end-points
            on which an application is currently accepting datagrams.
            This table has been deprecated in favor of the version
            neutral udpListenerTable."
    ::= { udp 5 }

udpEntry OBJECT-TYPE
    SYNTAX     UdpEntry
    MAX-ACCESS not-accessible
    STATUS     deprecated
    DESCRIPTION
           "Information about a particular current UDP listener."
    INDEX   { udpLocalAddress, udpLocalPort }
    ::= { udpTable 1 }

UdpEntry ::= SEQUENCE {
        udpLocalAddress  IpAddress,
        udpLocalPort     INTEGER
    }


udpLocalAddress OBJECT-TYPE
    SYNTAX     IpAddress
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The local IP address for this UDP listener.  In the case of
            a UDP listener which is willing to accept datagrams for any
            IP interface associated with the node, the value 0.0.0.0 is
            used."
    ::= { udpEntry 1 }

udpLocalPort OBJECT-TYPE
    SYNTAX     INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The local port number for this UDP listener."
    ::= { udpEntry 2 }
```

```
-- conformance information

udpMIBConformance OBJECT IDENTIFIER ::= { udpMIB 2 }

udpMIBCompliances OBJECT IDENTIFIER ::= { udpMIBConformance 1 }
udpMIBGroups      OBJECT IDENTIFIER ::= { udpMIBConformance 2 }

-- compliance statements

udpMIBCompliance2 MODULE-COMPLIANCE
    STATUS     current
    DESCRIPTION
           "The compliance statement for systems which implement UDP."
    MODULE  -- this module
        MANDATORY-GROUPS { udpBaseGroup, udpListenerGroup }
        GROUP       udpHCGroup
        DESCRIPTION
           "This group is mandatory for those systems which are capable
                of receiving or transmitting more than 1 million UDP
                packets per second.  1 million packets per second will
                cause a Counter32 to wrap in just over an hour."
        GROUP       udpListenerStatsGroup
        DESCRIPTION
           "This group is optional."
        GROUP       udpListenerHCStatsGroup
        DESCRIPTION
           "This group is mandatory for systems which implement
                udpListenerStatsGroup and are capable of receiving or
                transmitting more than 1 million UDP packets per second.
                1 million packets per second will cause a Counter32 to
                wrap in just over an hour."
        GROUP       udpConnectionGroup
        DESCRIPTION
           "This group is mandatory for systems which implement the UDP
                'connection' abstraction."
        GROUP       udpConnectionStatsGroup
        DESCRIPTION
           "This group is optional."
        GROUP       udpConnectionHCStatsGroup
        DESCRIPTION
           "This group is mandatory for systems which implement
                udpConnectionStatsGroup and are capable of receiving or
                transmitting more than 1 million UDP packets per second.
                1 million packets per second will cause a Counter32 to
                wrap in just over an hour."

    ::= { udpMIBCompliances 2 }

udpMIBCompliance MODULE-COMPLIANCE
    STATUS     deprecated
    DESCRIPTION
           "The compliance statement for IPv4-only systems which
            implement UDP.  For IP version independence, this compliance
            statement is deprecated in favor of udpMIBCompliance2.
            However, agents are still encouraged to implement these
            objects in order to interoperate with the deployed base of
            managers."
    MODULE  -- this module
        MANDATORY-GROUPS { udpGroup }
    ::= { udpMIBCompliances 1 }

-- units of conformance

udpGroup OBJECT-GROUP
    OBJECTS   { udpInDatagrams, udpNoPorts,
                udpInErrors, udpOutDatagrams,
                udpLocalAddress, udpLocalPort }
    STATUS     deprecated
    DESCRIPTION
           "The deprecated group of objects providing for management of
            UDP over IPv4."
    ::= { udpMIBGroups 1 }

udpBaseGroup OBJECT-GROUP
    OBJECTS   { udpInDatagrams, udpNoPorts, udpInErrors, udpOutDatagrams }
    STATUS     current
    DESCRIPTION
           "The group of objects providing for counters of UDP
            statistics."
    ::= { udpMIBGroups 2 }

udpHCGroup OBJECT-GROUP
    OBJECTS   { udpHCInDatagrams, udpHCOutDatagrams }
    STATUS     current
```

```
    DESCRIPTION
           "The group of objects providing for counters of high speed
            UDP implementations."
    ::= { udpMIBGroups 3 }

udpListenerGroup OBJECT-GROUP
    OBJECTS   { udpListenerInstance }
    STATUS    current
    DESCRIPTION
           "The group of objects providing for the IP version
            independent management of UDP listeners."
    ::= { udpMIBGroups 4 }

udpListenerStatsGroup OBJECT-GROUP
    OBJECTS    { udpListenerInPackets, udpListenerOutPackets,
                 udpListenerInOctets, udpListenerOutOctets,
                 udpListenerProcessID }
    STATUS     current
    DESCRIPTION
           "The group of objects to provide statistics about UDP
            listeners."
    ::= { udpMIBGroups 5 }

udpListenerHCStatsGroup OBJECT-GROUP
    OBJECTS    { udpListenerHCInPackets, udpListenerHCOutPackets,
                 udpListenerHCInOctets, udpListenerHCOutOctets }
    STATUS     current
    DESCRIPTION
           "The group of objects to provide statitics about UDP
            listeners on high speed UDP implementations."
    ::= { udpMIBGroups 6 }

udpConnectionGroup OBJECT-GROUP
    OBJECTS    { udpConnectionInstance }
    STATUS     current
    DESCRIPTION
           "The group of objects providing for the IP version
            independent management of UDP 'connections'."
    ::= { udpMIBGroups 7 }

udpConnectionStatsGroup OBJECT-GROUP
    OBJECTS    { udpConnectionInstance, udpConnectionInPackets,
                 udpConnectionOutPackets, udpConnectionInOctets,
                 udpConnectionOutOctets, udpConnectionProcessID }
    STATUS     current
    DESCRIPTION
           "The group of objects providing statistics about UDP
            'connections'."
    ::= { udpMIBGroups 8 }

udpConnectionHCStatsGroup OBJECT-GROUP
    OBJECTS    { udpConnectionHCInPackets, udpConnectionHCOutPackets,
                 udpConnectionHCInOctets, udpConnectionHCOutOctets }
    STATUS     current
    DESCRIPTION
           "The group of objects to provide statitics about UDP
            'connections' on high speed UDP implementations."
    ::= { udpMIBGroups 9 }


END
```

A.4 draft-ietf-ipngwg-RFC2096-update-00

```
IP-FORWARD-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    IpAddress, Integer32, Gauge32,
    Unsigned32, Counter32               FROM SNMPv2-SMI
    RowStatus                           FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP     FROM SNMPv2-CONF
    InterfaceIndex                      FROM IF-MIB
    ip                                  FROM IP-MIB
    IANAipRouteProtocol                 FROM IANA-RTPROTO-MIB
    InetAddress, InetAddressType,
    InetAddressPrefixLength,
    InetAutonomousSystemNumber          FROM INET-ADDRESS-MIB;




ipForward MODULE-IDENTITY
    LAST-UPDATED "200107130000Z"
    ORGANIZATION "IETF IPv6 MIB Revision Team"
    CONTACT-INFO
           "Editor:
            Bill Fenner
            AT&T Labs - Research
            75 Willow Rd
            Menlo Park, CA

            Phone: +1 650 330-7893
            Email: <fenner@research.att.com>"
    DESCRIPTION
           "The MIB module for the management of CIDR multipath IP
            Routes."
    REVISION      "200107130000Z"
    DESCRIPTION
           "IP version neutral revision, published as RFC XXXX."
    REVISION      "9609190000Z"
    DESCRIPTION
           "Revised to support CIDR routes."
    ::= { ip 24 }

inetCidrRouteNumber OBJECT-TYPE
    SYNTAX     Gauge32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of current inetCidrRouteTable entries that are
            not invalid."
    ::= { ipForward 6 }

inetCidrRouteDiscards OBJECT-TYPE
    SYNTAX     Counter32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of routing entries which were chosen to be
            discarded even though they are valid.  One possible reason
            for discarding such an entry could be to free-up buffer
            space for other routing entries."
    ::= { ipForward 8 }

--  Inet CIDR Route Table

--  The Inet CIDR Route Table deprecates and replaces the ipCidrRoute
--  Table currently in the IP Forwarding Table MIB.
--  It adds IP protocol independence.




inetCidrRouteTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF InetCidrRouteEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "This entity's IP Routing table."
    REFERENCE
       "RFC 1213 Section 6.6, The IP Group"
    ::= { ipForward 7 }
```

```
inetCidrRouteEntry OBJECT-TYPE
    SYNTAX     InetCidrRouteEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "A particular route to a particular destination, under a
            particular policy."
    INDEX {
        inetCidrRouteInstance,
        inetCidrRouteDestType,
        inetCidrRouteDest,
        inetCidrRoutePfxLen,
        inetCidrRouteNextHopType,
        inetCidrRouteNextHop
        }
    ::= { inetCidrRouteTable 1 }

InetCidrRouteEntry ::= SEQUENCE {
        inetCidrRouteInstance     Unsigned32,
        inetCidrRouteDestType     InetAddressType,
        inetCidrRouteDest         InetAddress,
        inetCidrRoutePfxLen       InetAddressPrefixLength,
        inetCidrRouteNextHopType  InetAddressType,
        inetCidrRouteNextHop      InetAddress,
        inetCidrRouteIfIndex      InterfaceIndex,
        inetCidrRouteType         INTEGER,
        inetCidrRouteProto        IANAipRouteProtocol,
        inetCidrRouteAge          Integer32,
        inetCidrRouteNextHopAS    InetAutonomousSystemNumber,
        inetCidrRouteMetric1      Integer32,
        inetCidrRouteMetric2      Integer32,
        inetCidrRouteMetric3      Integer32,
        inetCidrRouteMetric4      Integer32,
        inetCidrRouteMetric5      Integer32,
        inetCidrRouteStatus       RowStatus
    }

inetCidrRouteInstance OBJECT-TYPE




    SYNTAX     Unsigned32
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The instance identifier of the (conceptual) routing table
            containing this route.  This identifier may be used to
            represent multiple routing tables, type-of-service routing,
            scopes, or any other use of multiple tables.

            XXX This needs more discussion."
    ::= { inetCidrRouteEntry 1 }

inetCidrRouteDestType OBJECT-TYPE
    SYNTAX     InetAddressType
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The type of inetCidrRouteDest.  Only IPv4 and IPv6 addresses
            are expected."
    ::= { inetCidrRouteEntry 2 }

inetCidrRouteDest OBJECT-TYPE
    SYNTAX     InetAddress (SIZE(0..36))
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The destination IP address of this route.

            Any assignment (implicit or otherwise) of an instance of
            this object to a value x must be rejected if the bitwise
            logical-AND of x with the value of the mask formed from the
            corresponding instance of the inetCidrRoutePfxLen object is
            not equal to x."
    ::= { inetCidrRouteEntry 3 }

inetCidrRoutePfxLen OBJECT-TYPE
    SYNTAX     InetAddressPrefixLength
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "Indicate the number of leading one bits which form the mask
```

```
            to be logical-ANDed with the destination address before
            being compared to the value in the inetCidrRouteDest field.

            Any assignment (implicit or otherwise) of an instance of
            this object to a value x must be rejected if the bitwise
            logical-AND of the mask formed from x with the value of the
            corresponding instance of the inetCidrRouteDest object is




            not equal to inetCidrRouteDest."
    ::= { inetCidrRouteEntry 4 }

inetCidrRouteNextHopType OBJECT-TYPE
    SYNTAX     InetAddressType
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "The address type of inetCidrRouteNextHop.  Must be the same
            as that of inetCidrRouteDestType, or unknown if there is no
            next hop."
    ::= { inetCidrRouteEntry 5 }

inetCidrRouteNextHop OBJECT-TYPE
    SYNTAX     InetAddress (SIZE(0..36))
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
           "On remote routes, the address of the next system en route;
            Otherwise, a zero-length string."
    ::= { inetCidrRouteEntry 6 }

inetCidrRouteIfIndex OBJECT-TYPE
    SYNTAX     InterfaceIndex
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
           "The ifIndex value which identifies the local interface
            through which the next hop of this route should be reached."
    ::= { inetCidrRouteEntry 7 }

inetCidrRouteType OBJECT-TYPE
    SYNTAX     INTEGER {
                 other    (1), -- not specified by this MIB
                 reject   (2), -- route which discards traffic and
                               --  returns notification
                 local    (3), -- local interface
                 remote   (4), -- remote destination
                 blackhole(5)  -- route which discards traffic silently
               }
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
           "The type of route.  Note that local(3) refers to a route for
            which the next hop is the final destination; remote(4)
            refers to a route for which the next hop is not the final
            destination.




            Routes which do not result in traffic forwarding or
            rejection should not be displayed even if the implementation
            keeps them stored internally.

            reject(2) refers to a route which, if matched, discards the
            message as unreachable and returns a notification (e.g. ICMP
            error) to the message sender.  This is used in some
            protocols as a means of correctly aggregating routes.
            blackhole(5) refers to a route which, if matched, discards
            the message silently."
    ::= { inetCidrRouteEntry 8 }

inetCidrRouteProto OBJECT-TYPE
    SYNTAX     IANAipRouteProtocol
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The routing mechanism via which this route was learned.
            Inclusion of values for gateway routing protocols is not
```

```
            intended to imply that hosts should support those
            protocols."
    ::= { inetCidrRouteEntry 9 }

-- XXX new type?  TimeTicks?
inetCidrRouteAge OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The number of seconds since this route was last updated or
            otherwise determined to be correct.  Note that no semantics
            of 'too old' can be implied except through knowledge of the
            routing protocol by which the route was learned."
    ::= { inetCidrRouteEntry 10 }

inetCidrRouteNextHopAS OBJECT-TYPE
    SYNTAX     InetAutonomousSystemNumber
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
           "The Autonomous System Number of the Next Hop.  The semantics
            of this object are determined by the routing-protocol
            specified in the route's inetCidrRouteProto value. When this
            object is unknown or not relevant its value should be set to
            zero."
    DEFVAL { 0 }
    ::= { inetCidrRouteEntry 11 }




inetCidrRouteMetric1 OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
           "The primary routing metric for this route.  The semantics of
            this metric are determined by the routing-protocol specified
            in the route's inetCidrRouteProto value.  If this metric is
            not used, its value should be set to -1."
    DEFVAL { -1 }
    ::= { inetCidrRouteEntry 12 }

inetCidrRouteMetric2 OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
           "An alternate routing metric for this route.  The semantics
            of this metric are determined by the routing-protocol
            specified in the route's inetCidrRouteProto value.  If this
            metric is not used, its value should be set to -1."
    DEFVAL { -1 }
    ::= { inetCidrRouteEntry 13 }

inetCidrRouteMetric3 OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
           "An alternate routing metric for this route.  The semantics
            of this metric are determined by the routing-protocol
            specified in the route's inetCidrRouteProto value.  If this
            metric is not used, its value should be set to -1."
    DEFVAL { -1 }
    ::= { inetCidrRouteEntry 14 }

inetCidrRouteMetric4 OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
           "An alternate routing metric for this route.  The semantics
            of this metric are determined by the routing-protocol
            specified in the route's inetCidrRouteProto value.  If this
            metric is not used, its value should be set to -1."
    DEFVAL { -1 }
    ::= { inetCidrRouteEntry 15 }
```

```
inetCidrRouteMetric5 OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
           "An alternate routing metric for this route.  The semantics
            of this metric are determined by the routing-protocol
            specified in the route's inetCidrRouteProto value.  If this
            metric is not used, its value should be set to -1."
    DEFVAL { -1 }
    ::= { inetCidrRouteEntry 16 }

inetCidrRouteStatus OBJECT-TYPE
    SYNTAX     RowStatus
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
           "The row status variable, used according to row installation
            and removal conventions."
    ::= { inetCidrRouteEntry 17 }



--  Conformance information

ipForwardConformance OBJECT IDENTIFIER ::= { ipForward 5 }

ipForwardGroups      OBJECT IDENTIFIER ::= { ipForwardConformance 1 }
ipForwardCompliances OBJECT IDENTIFIER ::= { ipForwardConformance 2 }

--  Compliance statements

ipForwardCompliance2 MODULE-COMPLIANCE
    STATUS     current
    DESCRIPTION
           "The compliance statement for systems which have routing
            tables.  XXX is this right?"
   MODULE  -- this module
   MANDATORY-GROUPS { inetForwardCidrRouteGroup }
   ::= { ipForwardCompliances 3 }

-- units of conformance

inetForwardCidrRouteGroup OBJECT-GROUP
    OBJECTS { inetCidrRouteNumber, inetCidrRouteDiscards,
              inetCidrRouteIfIndex, inetCidrRouteType,
              inetCidrRouteProto, inetCidrRouteAge,
              inetCidrRouteNextHopAS, inetCidrRouteMetric1,




              inetCidrRouteMetric2, inetCidrRouteMetric3,
              inetCidrRouteMetric4, inetCidrRouteMetric5, inetCidrRouteStatus
        }
    STATUS     current
    DESCRIPTION
           "The IP version independent CIDR Route Table."
    ::= { ipForwardGroups 4 }

--  Deprecated Objects

ipCidrRouteNumber OBJECT-TYPE
    SYNTAX     Gauge32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of current ipCidrRouteTable entries that are not
            invalid.  This object is deprecated in favor of
            inetCidrRouteNumber and the inetCidrRouteTable."
    ::= { ipForward 3 }

--  IP CIDR Route Table

--  The IP CIDR Route Table obsoletes and replaces the ipRoute
--  Table current in MIB-I and MIB-II and the IP Forwarding Table.
--  It adds knowledge of the autonomous system of the next hop,
--  multiple next hops, and policy routing, and Classless
--  Inter-Domain Routing.

ipCidrRouteTable OBJECT-TYPE
```

```
    SYNTAX     SEQUENCE OF IpCidrRouteEntry
    MAX-ACCESS not-accessible
    STATUS     deprecated
    DESCRIPTION
           "This entity's IP Routing table.  This table has been
            deprecated in favor of the IP version neutral
            inetCidrRouteTable."
    REFERENCE
       "RFC 1213 Section 6.6, The IP Group"
    ::= { ipForward 4 }

ipCidrRouteEntry OBJECT-TYPE
    SYNTAX     IpCidrRouteEntry
    MAX-ACCESS not-accessible
    STATUS     deprecated
    DESCRIPTION
           "A particular route to a particular destination, under a
            particular policy."
    INDEX {



        ipCidrRouteDest,
        ipCidrRouteMask,
        ipCidrRouteTos,
        ipCidrRouteNextHop
        }
    ::= { ipCidrRouteTable 1 }

IpCidrRouteEntry ::= SEQUENCE {
        ipCidrRouteDest       IpAddress,
        ipCidrRouteMask       IpAddress,
        ipCidrRouteTos        Integer32,
        ipCidrRouteNextHop    IpAddress,
        ipCidrRouteIfIndex    Integer32,
        ipCidrRouteType       INTEGER,
        ipCidrRouteProto      INTEGER,
        ipCidrRouteAge        Integer32,
        ipCidrRouteInfo       OBJECT IDENTIFIER,
        ipCidrRouteNextHopAS  Integer32,
        ipCidrRouteMetric1    Integer32,
        ipCidrRouteMetric2    Integer32,
        ipCidrRouteMetric3    Integer32,
        ipCidrRouteMetric4    Integer32,
        ipCidrRouteMetric5    Integer32,
        ipCidrRouteStatus     RowStatus
    }

ipCidrRouteDest OBJECT-TYPE
    SYNTAX     IpAddress
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The destination IP address of this route.

            This object may not take a Multicast (Class D) address
            value.

            Any assignment (implicit or otherwise) of an instance of
            this object to a value x must be rejected if the bitwise
            logical-AND of x with the value of the corresponding
            instance of the ipCidrRouteMask object is not equal to x."
    ::= { ipCidrRouteEntry 1 }

ipCidrRouteMask OBJECT-TYPE
    SYNTAX     IpAddress
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "Indicate the mask to be logical-ANDed with the destination




            address before being compared to the value in the
            ipCidrRouteDest field.  For those systems that do not
            support arbitrary subnet masks, an agent constructs the
            value of the ipCidrRouteMask by reference to the IP Address
            Class.

            Any assignment (implicit or otherwise) of an instance of
            this object to a value x must be rejected if the bitwise
            logical-AND of x with the value of the corresponding
```

```
            instance of the ipCidrRouteDest object is not equal to
            ipCidrRouteDest."
    ::= { ipCidrRouteEntry 2 }

-- The following convention is included for specification
-- of TOS Field contents.  At this time, the Host Requirements
-- and the Router Requirements documents disagree on the width
-- of the TOS field.  This mapping describes the Router
-- Requirements mapping, and leaves room to widen the TOS field
-- without impact to fielded systems.

ipCidrRouteTos OBJECT-TYPE
    SYNTAX     Integer32 (0..2147483647)
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The policy specifier is the IP TOS Field.  The encoding
            of IP TOS is as specified by the following convention.
            Zero indicates the default path if no more specific
            policy applies.

            +-----+-----+-----+-----+-----+-----+-----+-----+
            |                 |                       |     |
            |   PRECEDENCE    |    TYPE OF SERVICE    |  0  |
            |                 |                       |     |
            +-----+-----+-----+-----+-----+-----+-----+-----+

                                              IP TOS                IP TOS
               Field     Policy      Field     Policy
               Contents    Code      Contents    Code
               0 0 0 0  ==>   0      0 0 0 1  ==>   2
               0 0 1 0  ==>   4      0 0 1 1  ==>   6
               0 1 0 0  ==>   8      0 1 0 1  ==>  10
               0 1 1 0  ==>  12      0 1 1 1  ==>  14
               1 0 0 0  ==>  16      1 0 0 1  ==>  18
               1 0 1 0  ==>  20      1 0 1 1  ==>  22
               1 1 0 0  ==>  24      1 1 0 1  ==>  26
               1 1 1 0  ==>  28      1 1 1 1  ==>  30"
    ::= { ipCidrRouteEntry 3 }



ipCidrRouteNextHop OBJECT-TYPE
    SYNTAX     IpAddress
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "On remote routes, the address of the next system en route;
            Otherwise, 0.0.0.0."
    ::= { ipCidrRouteEntry 4 }

ipCidrRouteIfIndex OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     deprecated
    DESCRIPTION
           "The ifIndex value which identifies  the local interface
            through which the next hop of this route should be reached."
    DEFVAL { 0 }
    ::= { ipCidrRouteEntry 5 }

ipCidrRouteType OBJECT-TYPE
    SYNTAX     INTEGER {
                other    (1), -- not specified by this MIB
                reject   (2), -- route which discards traffic
                local    (3), -- local interface
                remote   (4)  -- remote destination
               }
    MAX-ACCESS read-create
    STATUS     deprecated
    DESCRIPTION
           "The type of route.  Note that local(3) refers to a route for
            which the next hop is the final destination; remote(4)
            refers to a route for which the next hop is not the final
            destination.

            Routes which do not result in traffic forwarding or
            rejection should not be displayed even if the implementation
            keeps them stored internally.

            reject (2) refers to a route which, if matched, discards the
            message as unreachable. This is used in some protocols as a
            means of correctly aggregating routes."
```

```
    ::= { ipCidrRouteEntry 6 }

ipCidrRouteProto OBJECT-TYPE
    SYNTAX     INTEGER {
                 other     (1),  -- not specified
                 local     (2),  -- local interface
                 netmgmt   (3),  -- static route




                 icmp      (4),  -- result of ICMP Redirect

                        -- the following are all dynamic
                        -- routing protocols
                 egp         (5),  -- Exterior Gateway Protocol
                 ggp         (6),  -- Gateway-Gateway Protocol
                 hello       (7),  -- FuzzBall HelloSpeak
                 rip         (8),  -- Berkeley RIP or RIP-II
                 isIs        (9),  -- Dual IS-IS
                 esIs        (10), -- ISO 9542
                 ciscoIgrp   (11), -- Cisco IGRP
                 bbnSpfIgp   (12), -- BBN SPF IGP
                 ospf        (13), -- Open Shortest Path First
                 bgp         (14), -- Border Gateway Protocol
                 idpr        (15), -- InterDomain Policy Routing
                 ciscoEigrp (16)  -- Cisco EIGRP
               }
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The routing mechanism via which this route was learned.
            Inclusion of values for gateway routing protocols is not
            intended to imply that hosts should support those
            protocols."
    ::= { ipCidrRouteEntry 7 }

ipCidrRouteAge OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-only
    STATUS     deprecated
    DESCRIPTION
           "The number of seconds since this route was last updated or
            otherwise determined to be correct.  Note that no semantics
            of `too old' can be implied except through knowledge of the
            routing protocol by which the route was learned."
    DEFVAL  { 0 }
    ::= { ipCidrRouteEntry 8 }

ipCidrRouteInfo OBJECT-TYPE
    SYNTAX     OBJECT IDENTIFIER
    MAX-ACCESS read-create
    STATUS     deprecated
    DESCRIPTION
           "A reference to MIB definitions specific to the particular
            routing protocol which is responsible for this route, as
            determined by the value specified in the route's
            ipCidrRouteProto value.  If this information is not present,
            its value should be set to the OBJECT IDENTIFIER { 0 0 },



            which is a syntactically valid object identifier, and any
            implementation conforming to ASN.1 and the Basic Encoding
            Rules must be able to generate and recognize this value."
    ::= { ipCidrRouteEntry 9 }

ipCidrRouteNextHopAS OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     deprecated
    DESCRIPTION
           "The Autonomous System Number of the Next Hop.  The semantics
            of this object are determined by the routing-protocol
            specified in the route's ipCidrRouteProto value. When this
            object is unknown or not relevant its value should be set to
            zero."
    DEFVAL { 0 }
    ::= { ipCidrRouteEntry 10 }

ipCidrRouteMetric1 OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
```

```
    STATUS     deprecated
    DESCRIPTION
           "The primary routing metric for this route.  The semantics of
            this metric are determined by the routing-protocol specified
            in the route's ipCidrRouteProto value.  If this metric is
            not used, its value should be set to -1."
    DEFVAL { -1 }
    ::= { ipCidrRouteEntry 11 }

ipCidrRouteMetric2 OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     deprecated
    DESCRIPTION
           "An alternate routing metric for this route.  The semantics
            of this metric are determined by the routing-protocol
            specified in the route's ipCidrRouteProto value.  If this
            metric is not used, its value should be set to -1."
    DEFVAL { -1 }
    ::= { ipCidrRouteEntry 12 }

ipCidrRouteMetric3 OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     deprecated
    DESCRIPTION
           "An alternate routing metric for this route.  The semantics




            of this metric are determined by the routing-protocol
            specified in the route's ipCidrRouteProto value.  If this
            metric is not used, its value should be set to -1."
    DEFVAL { -1 }
    ::= { ipCidrRouteEntry 13 }

ipCidrRouteMetric4 OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     deprecated
    DESCRIPTION
           "An alternate routing metric for this route.  The semantics
            of this metric are determined by the routing-protocol
            specified in the route's ipCidrRouteProto value.  If this
            metric is not used, its value should be set to -1."
    DEFVAL { -1 }
    ::= { ipCidrRouteEntry 14 }

ipCidrRouteMetric5 OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     deprecated
    DESCRIPTION
           "An alternate routing metric for this route.  The semantics
            of this metric are determined by the routing-protocol
            specified in the route's ipCidrRouteProto value.  If this
            metric is not used, its value should be set to -1."
    DEFVAL { -1 }
    ::= { ipCidrRouteEntry 15 }

ipCidrRouteStatus OBJECT-TYPE
    SYNTAX     RowStatus
    MAX-ACCESS read-create
    STATUS     deprecated
    DESCRIPTION
           "The row status variable, used according to row installation
            and removal conventions."
    ::= { ipCidrRouteEntry 16 }



-- compliance statements

ipForwardCompliance MODULE-COMPLIANCE
    STATUS     deprecated
    DESCRIPTION
           "The compliance statement for SNMPv2 entities which implement
            the ipForward MIB."
```

```
   MODULE  -- this module
   MANDATORY-GROUPS { ipForwardCidrRouteGroup }

   ::= { ipForwardCompliances 1 }

-- units of conformance

ipForwardCidrRouteGroup OBJECT-GROUP
    OBJECTS { ipCidrRouteNumber,
              ipCidrRouteDest, ipCidrRouteMask, ipCidrRouteTos,
              ipCidrRouteNextHop, ipCidrRouteIfIndex, ipCidrRouteType,
              ipCidrRouteProto, ipCidrRouteAge, ipCidrRouteInfo,
              ipCidrRouteNextHopAS, ipCidrRouteMetric1,
              ipCidrRouteMetric2, ipCidrRouteMetric3,
              ipCidrRouteMetric4, ipCidrRouteMetric5, ipCidrRouteStatus
        }
    STATUS     deprecated
    DESCRIPTION
           "The CIDR Route Table."
    ::= { ipForwardGroups 3 }

-- Obsoleted Definitions - Objects

ipForwardNumber OBJECT-TYPE
    SYNTAX     Gauge32
    MAX-ACCESS read-only
    STATUS     obsolete
    DESCRIPTION
           "The number of current ipForwardTable entries that are not
            invalid."
    ::= { ipForward 1 }

--  IP Forwarding Table

--  The IP Forwarding Table obsoletes and replaces the ipRoute
--  Table current in MIB-I and MIB-II.  It adds knowledge of
--  the autonomous system of the next hop, multiple next hop
--  support, and policy routing support.

ipForwardTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF IpForwardEntry
    MAX-ACCESS not-accessible
    STATUS     obsolete
    DESCRIPTION
           "This entity's IP Routing table."
    REFERENCE
       "RFC 1213 Section 6.6, The IP Group"
    ::= { ipForward 2 }



ipForwardEntry OBJECT-TYPE
    SYNTAX     IpForwardEntry
    MAX-ACCESS not-accessible
    STATUS     obsolete
    DESCRIPTION
           "A particular route to a particular destination, under a
            particular policy."
    INDEX {
        ipForwardDest,
        ipForwardProto,
        ipForwardPolicy,
        ipForwardNextHop
        }
    ::= { ipForwardTable 1 }

IpForwardEntry ::= SEQUENCE {
        ipForwardDest       IpAddress,
        ipForwardMask       IpAddress,
        ipForwardPolicy     Integer32,
        ipForwardNextHop    IpAddress,
        ipForwardIfIndex    Integer32,
        ipForwardType       INTEGER,
        ipForwardProto      INTEGER,
        ipForwardAge        Integer32,
        ipForwardInfo       OBJECT IDENTIFIER,
        ipForwardNextHopAS  Integer32,
        ipForwardMetric1    Integer32,
        ipForwardMetric2    Integer32,
        ipForwardMetric3    Integer32,
        ipForwardMetric4    Integer32,
        ipForwardMetric5    Integer32
    }
```

```
ipForwardDest OBJECT-TYPE
    SYNTAX     IpAddress
    MAX-ACCESS read-only
    STATUS     obsolete
    DESCRIPTION
           "The destination IP address of this route.  An entry with a
            value of 0.0.0.0 is considered a default route.

            This object may not take a Multicast (Class D) address
            value.

            Any assignment (implicit or otherwise) of an instance of
            this object to a value x must be rejected if the bitwise
            logical-AND of x with the value of the corresponding
            instance of the ipForwardMask object is not equal to x."


    ::= { ipForwardEntry 1 }

ipForwardMask OBJECT-TYPE
    SYNTAX     IpAddress
    MAX-ACCESS read-create
    STATUS     obsolete
    DESCRIPTION
           "Indicate the mask to be logical-ANDed with the destination
            address before being compared to the value in the
            ipForwardDest field.  For those systems that do not support
            arbitrary subnet masks, an agent constructs the value of the
            ipForwardMask by reference to the IP Address Class.

            Any assignment (implicit or otherwise) of an instance of
            this object to a value x must be rejected if the bitwise
            logical-AND of x with the value of the corresponding
            instance of the ipForwardDest object is not equal to
            ipForwardDest."
    DEFVAL { '00000000'h }      -- 0.0.0.0
    ::= { ipForwardEntry 2 }

-- The following convention is included for specification
-- of TOS Field contents.  At this time, the Host Requirements
-- and the Router Requirements documents disagree on the width
-- of the TOS field.  This mapping describes the Router
-- Requirements mapping, and leaves room to widen the TOS field
-- without impact to fielded systems.

ipForwardPolicy OBJECT-TYPE
    SYNTAX     Integer32 (0..2147483647)
    MAX-ACCESS read-only
    STATUS     obsolete
    DESCRIPTION
           "The general set of conditions that would cause
            the selection of one multipath route (set of
            next hops for a given destination) is referred
            to as 'policy'.

            Unless the mechanism indicated by ipForwardProto
            specifies otherwise, the policy specifier is
            the IP TOS Field.  The encoding of IP TOS is as
            specified by the following convention.  Zero
            indicates the default path if no more specific
            policy applies.

            +-----+-----+-----+-----+-----+-----+-----+-----+
            |                 |                       |     |
            |   PRECEDENCE    |    TYPE OF SERVICE    |  0  |



            |                 |                       |     |
            +-----+-----+-----+-----+-----+-----+-----+-----+

                                        IP TOS                IP TOS
               Field     Policy      Field     Policy
               Contents    Code      Contents    Code
               0 0 0 0  ==>   0      0 0 0 1  ==>   2
               0 0 1 0  ==>   4      0 0 1 1  ==>   6
               0 1 0 0  ==>   8      0 1 0 1  ==>  10
               0 1 1 0  ==>  12      0 1 1 1  ==>  14
               1 0 0 0  ==>  16      1 0 0 1  ==>  18
               1 0 1 0  ==>  20      1 0 1 1  ==>  22
               1 1 0 0  ==>  24      1 1 0 1  ==>  26
```

```
              1 1 1 0  ==>  28      1 1 1 1  ==>  30

            Protocols defining 'policy' otherwise must either
            define a set of values which are valid for
            this object or must implement an integer-instanced
            policy table for which this object's
            value acts as an index."
    ::= { ipForwardEntry 3 }

ipForwardNextHop OBJECT-TYPE
    SYNTAX     IpAddress
    MAX-ACCESS read-only
    STATUS     obsolete
    DESCRIPTION
           "On remote routes, the address of the next system en route;
            Otherwise, 0.0.0.0."
    ::= { ipForwardEntry 4 }

ipForwardIfIndex OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     obsolete
    DESCRIPTION
           "The ifIndex value which identifies the local interface
            through which the next hop of this route should be reached."
    DEFVAL { 0 }
    ::= { ipForwardEntry 5 }

ipForwardType OBJECT-TYPE
    SYNTAX     INTEGER {
                other    (1), -- not specified by this MIB
                invalid  (2), -- logically deleted
                local    (3), -- local interface
                remote   (4)  -- remote destination
             }



    MAX-ACCESS read-create
    STATUS     obsolete
    DESCRIPTION
           "The type of route.  Note that local(3) refers to a route for
            which the next hop is the final destination; remote(4)
            refers to a route for which the next hop is not the final
            destination.

            Setting this object to the value invalid(2) has the effect
            of invalidating the corresponding entry in the
            ipForwardTable object.  That is, it effectively
            disassociates the destination identified with said entry
            from the route identified with said entry.  It is an
            implementation-specific matter as to whether the agent
            removes an invalidated entry from the table.  Accordingly,
            management stations must be prepared to receive tabular
            information from agents that corresponds to entries not
            currently in use.  Proper interpretation of such entries
            requires examination of the relevant ipForwardType object."
    DEFVAL { invalid }
    ::= { ipForwardEntry 6 }

ipForwardProto OBJECT-TYPE
    SYNTAX     INTEGER {
                other     (1),  -- not specified
                local     (2),  -- local interface
                netmgmt   (3),  -- static route
                icmp      (4),  -- result of ICMP Redirect

                        -- the following are all dynamic
                        -- routing protocols
                egp       (5),  -- Exterior Gateway Protocol
                ggp       (6),  -- Gateway-Gateway Protocol
                hello     (7),  -- FuzzBall HelloSpeak
                rip       (8),  -- Berkeley RIP or RIP-II
                is-is     (9),  -- Dual IS-IS
                es-is     (10), -- ISO 9542
                ciscoIgrp (11), -- Cisco IGRP
                bbnSpfIgp (12), -- BBN SPF IGP
                ospf      (13), -- Open Shortest Path First
                bgp       (14), -- Border Gateway Protocol
                idpr      (15)  -- InterDomain Policy Routing
              }
    MAX-ACCESS read-only
    STATUS     obsolete
```

```
    DESCRIPTION
           "The routing mechanism via which this route was learned.
            Inclusion of values for gateway routing protocols is not



            intended to imply that hosts should support those
            protocols."
    ::= { ipForwardEntry 7 }

ipForwardAge OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-only
    STATUS     obsolete
    DESCRIPTION
           "The number of seconds since this route was last updated or
            otherwise determined to be correct.  Note that no semantics
            of 'too old' can be implied except through knowledge of the
            routing protocol by which the route was learned."
    DEFVAL  { 0 }
    ::= { ipForwardEntry 8 }

ipForwardInfo OBJECT-TYPE
    SYNTAX     OBJECT IDENTIFIER
    MAX-ACCESS read-create
    STATUS     obsolete
    DESCRIPTION
           "A reference to MIB definitions specific to the particular
            routing protocol which is responsible for this route, as
            determined by the value specified in the route's
            ipForwardProto value.  If this information is not present,
            its value should be set to the OBJECT IDENTIFIER { 0 0 },
            which is a syntactically valid object identifier, and any
            implementation conforming to ASN.1 and the Basic Encoding
            Rules must be able to generate and recognize this value."
    ::= { ipForwardEntry 9 }

ipForwardNextHopAS OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     obsolete
    DESCRIPTION
           "The Autonomous System Number of the Next Hop.  When this is
            unknown or not relevant to the protocol indicated by
            ipForwardProto, zero."
    DEFVAL { 0 }
    ::= { ipForwardEntry 10 }

ipForwardMetric1 OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     obsolete
    DESCRIPTION
           "The primary routing metric for this route.  The semantics of


            this metric are determined by the routing-protocol specified
            in the route's ipForwardProto value.  If this metric is not
            used, its value should be set to -1."
    DEFVAL { -1 }
    ::= { ipForwardEntry 11 }

ipForwardMetric2 OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     obsolete
    DESCRIPTION
           "An alternate routing metric for this route.  The semantics
            of this metric are determined by the routing-protocol
            specified in the route's ipForwardProto value.  If this
            metric is not used, its value should be set to -1."
    DEFVAL { -1 }
    ::= { ipForwardEntry 12 }

ipForwardMetric3 OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     obsolete
    DESCRIPTION
           "An alternate routing metric for this route.  The semantics
            of this metric are determined by the routing-protocol
            specified in the route's ipForwardProto value.  If this
            metric is not used, its value should be set to -1."
    DEFVAL { -1 }
```

```
    ::= { ipForwardEntry 13 }

ipForwardMetric4 OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     obsolete
    DESCRIPTION
           "An alternate routing metric for this route.  The semantics
            of this metric are determined by the routing-protocol
            specified in the route's ipForwardProto value.  If this
            metric is not used, its value should be set to -1."
    DEFVAL { -1 }
    ::= { ipForwardEntry 14 }

ipForwardMetric5 OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     obsolete
    DESCRIPTION
           "An alternate routing metric for this route.  The semantics


            of this metric are determined by the routing-protocol
            specified in the route's ipForwardProto value.  If this
            metric is not used, its value should be set to -1."
    DEFVAL { -1 }
    ::= { ipForwardEntry 15 }



-- Obsoleted Definitions - Groups
-- compliance statements

ipForwardOldCompliance MODULE-COMPLIANCE
    STATUS     obsolete
    DESCRIPTION
           "The compliance statement for SNMP entities which implement
            the ipForward MIB."

   MODULE  -- this module
   MANDATORY-GROUPS { ipForwardMultiPathGroup }

   ::= { ipForwardCompliances 2 }

ipForwardMultiPathGroup OBJECT-GROUP
    OBJECTS { ipForwardNumber,
              ipForwardDest, ipForwardMask, ipForwardPolicy,
              ipForwardNextHop, ipForwardIfIndex, ipForwardType,
              ipForwardProto, ipForwardAge, ipForwardInfo,
              ipForwardNextHopAS,
              ipForwardMetric1, ipForwardMetric2, ipForwardMetric3,
              ipForwardMetric4, ipForwardMetric5
        }
    STATUS     obsolete
    DESCRIPTION
           "IP Multipath Route Table."
    ::= { ipForwardGroups 2 }

END
```

B Code of the drafts implementation

Here is added the code which is not describe in detail into the main core of the document.

B.1 Implementation of the ipIfStatsTable

The get_iflist6 function into the ipIfStatsTable is the following :

```
static void
get_iflist6(void)
{
  int  bit;
  static int mib[6] = { CTL_NET, PF_ROUTE, 0, AF_INET6 , NET_RT_IFLIST, 0 };

  char *cp, *ifbuf;
  size_t len;
  struct rt_msghdr *rtm;
  struct if_msghdr *ifm;
  struct ifa_msghdr *ifam, *ifam2;

  if (ifs2)
    free(ifs2);
  ifs2 = 0;
  nindex2 = 0;
  nifs = 0;
  len = 0;

  if (sysctl(mib, 6, 0, &len, 0, 0) < 0)
    return;

  ifbuf = malloc(len);
  if (ifbuf == 0)
    return;

  if (sysctl(mib, 6, ifbuf, &len, 0, 0) < 0) {
    syslog(LOG_WARNING, "sysctl net-route-iflist: %m");
    free(ifbuf);
    return;
  }

loop:
  nindex2 = 0;
  cp = ifbuf;

  while (cp < &ifbuf[len]) {
    int gotindex;
```

```
    gotindex = 0;
    rtm = (struct rt_msghdr *)cp;

    if (rtm->rtm_version != RTM_VERSION || rtm->rtm_type != RTM_IFINFO) {
      free(ifs2);
      ifs = 0;
      nifs = 0;
      free(ifbuf);
      return;
    }
    ifm = (struct if_msghdr *)rtm;
    cp += ifm->ifm_msglen;
    rtm = (struct rt_msghdr *)cp;

    while (cp < &ifbuf[len] && rtm->rtm_type == RTM_NEWADDR) {

      if (ifs2) {
ifs2[nindex2].in_recieve=ifm->ifm_data.ifi_ipackets;
ifs2[nindex2].in_errors=ifm->ifm_data.ifi_ierrors;
ifs2[nindex2].in_noproto=ifm->ifm_data.ifi_noproto;
ifs2[nindex2].in_octets=ifm->ifm_data.ifi_ibytes;
ifs2[nindex2].out_octets=ifm->ifm_data.ifi_obytes;
ifs2[nindex2].in_mcast=ifm->ifm_data.ifi_imcasts;
ifs2[nindex2].out_mcast=ifm->ifm_data.ifi_omcasts;
ifs2[nindex2].index  = ifm->ifm_index;
ifs2[nindex2].addrtype = 2;
      }
      gotindex = 1;

      cp = (char *)rtm + rtm->rtm_msglen;
      rtm = (struct rt_msghdr *)cp;
    }
    if (gotindex)
      nindex2++;

  }
  if (ifs2) {
    nifs = nindex2 + nindex1;
    free(ifbuf);
    return;
  }
  ifs2 = malloc(nindex2 * sizeof(*ifs2));

  if (ifs2 == 0) {
    free(ifbuf);
    return;
```

```
  }
  nindex2 = 0;
  goto loop;

}
```

The var_ipIfStatsEntry function is:

```
u_char *
var_ipIfStatsEntry(struct variable *vp,
oid *name,
size_t *length,
int exact,
size_t *var_len,
WriteMethod **write_method)
{
  oid lowest[12];
  oid current[12], *op;
  u_char *cp;
  int lowinterface = -1;
  int i;
  char vide[]="0.0.0.0";

/* fill in object part of name for current (less sizeof instance part) */
memcpy(current, vp->name, (int)vp->namelen * sizeof(oid));


/*
 * Get interface table from kernel.
 */
get_iflist();
get_iflist6();

  ifs = malloc(nifs * sizeof(*ifs));

for (i=0 ; i <nindex1 ; i++)
  ifs[i] = ifs1[i];


for (i=nindex1; i < nifs; i++)
  ifs[i]=ifs2[i-nindex1];


for (i = 0; i < nifs; i++) {

  op = &current[10];
```

```
  *op++=ifs[i].addrtype;
  *op++=ifs[i].index;

  if (exact) {
    if (snmp_oid_compare(current, 12, name, *length) == 0) {
      memcpy(lowest, current, 12 * sizeof(oid));
      lowinterface = i;
      break; /* no need to search further */
    }
  } else {
    if ((snmp_oid_compare(current, 12, name, *length) > 0) &&
(lowinterface < 0
 || (snmp_oid_compare(current, 12, lowest, 12) < 0))) {
      /*
       * if new one is greater than input
       * and closer to input than previous
       * lowest, save this one as the "next"
       * one.
       */
      lowinterface = i;
      memcpy(lowest, current, 12 * sizeof(oid));
    }
  }

}

if (lowinterface < 0)
  return NULL;

i = lowinterface;


memcpy(name, lowest, 12 * sizeof(oid));
*length = 12;

*write_method = 0;

*var_len = sizeof(long_return);

switch (vp->magic) {

case IPIFSTATSAFTYPE:
  long_return = ifs[i].addrtype;
  return (u_char *)&long_return;

case IPIFSTATSIFINDEX:
```

```
  long_return = ifs[i].index;
  return (u_char *)&long_return;

case IPIFSTATSINRECEIVES:
  long_return = ifs[i].in_recieve;
  return (u_char *)&long_return;

case IPIFSTATSINHDRERRORS:
  long_return = ifs[i].in_errors;
  return (u_char *)&long_return;

case IPIFSTATSINUNKNOWNPROTOS:
  long_return = ifs[i].in_noproto;
  return (u_char *)&long_return;

case IPIFSTATSINOCTETS:
  long_return =ifs[i].in_octets;
  return (u_char *)&long_return;

case IPIFSTATSOUTOCTETS:
  long_return =ifs[i].in_octets;
  return (u_char *)&long_return;

case IPIFSTATSINMCASTPKTS:
  long_return =ifs[i].in_mcast;
  return (u_char *)&long_return;

case IPIFSTATSOUTMCASTPKTS:
  long_return =ifs[i].out_mcast;
  return (u_char *)&long_return;

default:
DEBUGMSGTL(("snmpd", "unknown sub-id %d in
var_ipIfStatsEntry\n",
vp->magic));
}
return NULL;
}
```

B.2 Implementation of ipv4IfTable

We have added below the exact code added to manage the element available into the ipv4IfTable, defined into the ipv4If.c file :

```
struct iflist {

        int index;

};

static struct iflist *ifs;
static int nifs;
static struct sockaddr *sa;

static void
get_iflist(void)
{
  int bit;
  static int mib[6] = { CTL_NET, PF_ROUTE, 0, AF_INET , NET_RT_IFLIST, 0 };

  char *cp, *ifbuf;
  size_t len;
  struct rt_msghdr *rtm;
  struct if_msghdr *ifm;
  struct ifa_msghdr *ifam;

  if (ifs)
    free(ifs);

  ifs = 0;
  nifs = 0;
  len = 0;

  if (sysctl(mib, 6, 0, &len, 0, 0) < 0)
    return;

  ifbuf = malloc(len);

  if (ifbuf == 0)
    return;

  if (sysctl(mib, 6, ifbuf, &len, 0, 0) < 0) {
    syslog(LOG_WARNING, "sysctl net-route-iflist: %m");
    free(ifbuf);
    return;
  }
```

```
loop:

  nifs = 0;
  cp = ifbuf;

  while (cp < &ifbuf[len]) {
    int gotindex;

    gotindex = 0;
    rtm = (struct rt_msghdr *)cp;

    if (rtm->rtm_version != RTM_VERSION || rtm->rtm_type != RTM_IFINFO) {
      free(ifs);
      ifs = 0;
      nifs = 0;
      free(ifbuf);
      return;
    }

    ifm = (struct if_msghdr *)rtm;
    cp += ifm->ifm_msglen;
    rtm = (struct rt_msghdr *)cp;

    while (cp < &ifbuf[len] && rtm->rtm_type == RTM_NEWADDR) {

      if (ifs)
ifs[nifs].index = ifm->ifm_index;

      gotindex = 1;

      cp = (char *)rtm + rtm->rtm_msglen;
      rtm = (struct rt_msghdr *)cp;
    }

    if (gotindex)
      nifs++;

  }
  if (ifs) {
    free(ifbuf);
    return;
  }

  ifs = malloc(nifs * sizeof(*ifs));
```

```
  if (ifs == 0) {
    free(ifbuf);
    return;
  }

  nifs = 0;
  goto loop;

}

u_char *
var_ipv4IfEntry(struct variable *vp,
oid *name,
size_t *length,
int exact,
size_t *var_len,
WriteMethod **write_method)
{
  oid lowest[11];
  oid current[11], *op;
  u_char *cp;
  int lowinterface = -1;
  int i;

/* fill in object part of name for current (less sizeof instance part) */
  memcpy(current, vp->name, (int)vp->namelen * sizeof(oid));

/*
 * Get interface table from kernel.
 */
  get_iflist();

  for (i = 0; i < nifs; i++) {
    op = &current[10];
    cp = (u_char *)&ifs[i].index;
    *op++ = *cp++;

    if (exact) {
      if (snmp_oid_compare(current, 11, name, *length) == 0) {
memcpy(lowest, current, 11 * sizeof(oid));
lowinterface = i;
break; /* no need to search further */
      }
    } else {
      if ((snmp_oid_compare(current, 11, name, *length) > 0) &&
          (lowinterface < 0 || (snmp_oid_compare(current, 11, lowest, 11) < 0))) {
```

```
/*
 * if new one is greater than input
 * and closer to input than previous
 * lowest, save this one as the "next"
 * one.
 */
lowinterface = i;
memcpy(lowest, current, 11 * sizeof(oid));
      }
    }
  }

  if (lowinterface < 0)
    return NULL;

  i = lowinterface;

  memcpy(name, lowest, 11 * sizeof(oid));
  *length = 11;
  *write_method = 0;
  *var_len = sizeof(long_return);

  switch (vp->magic) {

  case IPV4IFINDEX:
    long_return = ifs[i].index;
    return (u_char *)&long_return;

  case IPV4IFREASMMAXSIZE:
    return NULL;

#if NO_DUMMY_VALUES
    return NULL;
#else
    long_return = -1;
    return (u_char *)&long_return;
#endif

  default:
    DEBUGMSGTL(("snmpd", "unknown sub-id %d in var_ipv4IfEntry\n",
vp->magic));
  }
  return NULL;

}
```

B.3 Code for the ipv6ScopeIdtable

As this table is tipicaly an IPv6 table, only the get_iflist6 function is defined:

```
0   struct iflist {
1     int notavailable;

            int index;

5   };
    static struct iflist *ifs,*ifs1,*ifs2;
    static int nifs,naddrs1,naddrs2;

    static  char buf[INET_ADDRSTRLEN], buf6[INET6_ADDRSTRLEN];
10
    static void
    get_iflist6(void)
    {
    int  bit;
15  static int mib[6] = { CTL_NET, PF_ROUTE, 0, AF_INET6 , NET_RT_IFLIST, 0 };

    char *cp, *ifbuf;
    char *temp1,*temp2;
    size_t len;
20  struct rt_msghdr *rtm;
    struct if_msghdr *ifm;
    struct ifa_msghdr *ifam, *ifam2;
    int lastindex=0;
    if (ifs2)
25          free(ifs2);
    ifs2 = 0;
    naddrs2 = 0;
    nifs = 0;
    len = 0;
30  if (sysctl(mib, 6, 0, &len, 0, 0) < 0)
            return;

    ifbuf = malloc(len);
    if (ifbuf == 0)
35          return;
    if (sysctl(mib, 6, ifbuf, &len, 0, 0) < 0) {
            syslog(LOG_WARNING, "sysctl net-route-iflist: %m");
            free(ifbuf);
            return;
40  }

    loop:
```

```
        cp = ifbuf;

45      while (cp < &ifbuf[len]) {
                int gotaddr;

                gotaddr = 0;
                rtm = (struct rt_msghdr *)cp;
50              if (rtm->rtm_version != RTM_VERSION
                    || rtm->rtm_type != RTM_IFINFO) {
                        free(ifs2);
                        ifs = 0;
                        nifs = 0;
55                      free(ifbuf);
                        return;
                }
                ifm = (struct if_msghdr *)rtm;

60              cp += ifm->ifm_msglen;
                rtm = (struct rt_msghdr *)cp;

                while (cp < &ifbuf[len] && rtm->rtm_type == RTM_NEWADDR) {
                        ifam = (struct ifa_msghdr *)rtm;
65                      cp += sizeof(*ifam);
                        cp = (char *)rtm + rtm->rtm_msglen;
                        rtm = (struct rt_msghdr *)cp;
                        if (ifs2) {
                                ifs2[naddrs2].notavailable=0;
70                              ifs2[naddrs2].index  = ifam->ifam_index;
                                }
                        if(lastindex !=  ifam->ifam_index)
                        naddrs2++;
                        lastindex=ifam->ifam_index;
75                      }

        }
        if (ifs2) {
                nifs = naddrs2 + naddrs1;
80              free(ifbuf);
                return;
        }
        ifs2 = malloc(naddrs2 * sizeof(*ifs2));
        if (ifs2 == 0) {
85              free(ifbuf);
                return;
        }
        naddrs2 = 0;
```

```
        goto loop;
}
```

The main function of the file is the var_ipv6ScopeIdEntry, which is defined as follows:

```

u_char *
var_ipv6ScopeIdEntry(struct variable *vp,
                oid *name,
                size_t *length,
                int exact,
                size_t *var_len,
                WriteMethod **write_method)
{
        /*
         * object identifier is of form:
         * 1.3.6.1.2.1.4.20.1.?.A.B.C.D,  where A.B.C.D is IP address.
         * IPADDR starts at offset 10.
         */
  oid lowest[28],prefix[128];
  oid current[28], *op;
  u_char *cp;
  int lowinterface = -1;
  int i;
  int start;
  char vide[]="0.0.0.0";
  int oid_compare_length;


      /* fill in object part of name for current (less sizeof instance part) */
  memcpy(current, vp->name, (int)vp->namelen * sizeof(oid));
  memcpy(prefix, vp->name, (int)vp->namelen * sizeof(oid));


  /*
   * Get interface table from kernel.
   */
  naddrs1=0;
  naddrs2=0;


  get_iflist6();
  //printf("nifs: %d\n",nifs);

  ifs = malloc(nifs * sizeof(*ifs));

  for (i=naddrs1; i < nifs; i++)
```

```
        ifs[i]=ifs2[i-naddrs1];

      oid_compare_length=11;
45
      for (i = 0; i < nifs; i++) {
        op = &current[10];
        *op++=ifs[i].index;
        if (exact) {
50          if (snmp_oid_compare(current, oid_compare_length, name, *length) == 0) {
                memcpy(lowest, current,oid_compare_length  * sizeof(oid));
                lowinterface = i;
                break;      /* no need to search further */
                }
55           } else {
                if ((snmp_oid_compare(current, oid_compare_length, name, *length) > 0)
                    &&
                    (lowinterface < 0
                     ||
60                   (snmp_oid_compare(current,oid_compare_length , lowest,oid_compare_length ) < 0)
                    )
                   ) {
                  /*
                   * if new one is greater than input
65                 * and closer to input than previous
                   * lowest, save this one as the "next"
                   * one.
                   */
                  lowinterface = i;
70                memcpy(lowest, current, oid_compare_length * sizeof(oid));
                  }
             }
            }

75          if (lowinterface < 0)
                   return NULL;

            i = lowinterface;

80          memcpy(name, lowest, oid_compare_length * sizeof(oid));
            *length =oid_compare_length ;

            *write_method = 0;

85          *var_len = sizeof(long_return);

            switch (vp->magic) {
```

```
            case 1: long_return = ifs[i].index;
                    return (u_char *)&long_return;
            case 2: long_return = ifs[i].index;
                    return (u_char *)&long_return;
            case 3: long_return = ifs[i].notavailable;
                    return (u_char *)&long_return;
            case 4: long_return = ifs[i].notavailable;
                    return (u_char *)&long_return;
            case 5: long_return = ifs[i].notavailable;
                    return (u_char *)&long_return;
            case 6: long_return = ifs[i].notavailable;
                    return (u_char *)&long_return;
            case 7: long_return = ifs[i].notavailable;
                    return (u_char *)&long_return;
            case 8: long_return = ifs[i].notavailable;
                    return (u_char *)&long_return;
            case 9: long_return = ifs[i].notavailable;
                    return (u_char *)&long_return;
            case 10: long_return = ifs[i].notavailable;
                    return (u_char *)&long_return;
            case 11: long_return = ifs[i].notavailable;
                    return (u_char *)&long_return;
            case 12: long_return = ifs[i].notavailable;
                    return (u_char *)&long_return;
            case 13: long_return = ifs[i].notavailable;
                    return (u_char *)&long_return;
            default:
                    DEBUGMSGTL(("snmpd", "unknown sub-id %d in var_ipAddressEntry\n", vp->magic));
            }
            return NULL;
    }
```

B.4 Implementation of the draft-ietf-ipngwg-RFC2096-update-00

The function var_ipForwardEntry is:

```
0   u_char *var_ipForwardEntry(struct variable * vp,
1                   oid * name,
                    size_t * length,
                    int exact, size_t * var_len, WriteMethod ** write_method)
    {
5       /*
         * object identifier is of form:
         * 1.3.6.1.2.1.4.21.1.1.A.B.C.D,  where A.B.C.D is IP address.
         * IPADDR starts at offset 10.
         */
10      int             Save_Valid, result;
```

```
        u_char         *cp;
        oid            *op;
        struct snmprt  *rt;
        static struct snmprt *savert;
15      static int      saveNameLen, saveExact;
        static oid      saveName[25], Current[49],lowest[49];
        int type=1;
        int oid_compare_length;
        int lowinterface;
20      int nooid;

        lowinterface=-1;

            /*
25           * fill in object part of name for current
             * (less sizeof instance part)
             */

            memcpy(Current, vp->name, (int) (vp->namelen) * sizeof(oid));
30          //printf("Dumped Earlier\n");
            suck_krt(0);
            //printf("Dumped\n");
            for (rt = rthead.tqh_first; rt; rt = rt->link.tqe_next) {
                op = Current + 11;
35              if(rt->type==1)
                  oid_compare_length=25;
                else
                  oid_compare_length=49;
                if(rt->type==1)
40                {
                cp = (u_char *) & rt->dest;
                *op++=1;
                *op++=1;
                *op++=32;
45              *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++=calcul_prefix(&rt->netmask,sizeof(struct in_addr));
50              *op++=1;
                 if (rt->gateway.s_addr == 0 && rt->ifa.s_addr == 0)

                   {
                     *op++=32;
55                    *op++ = 0;
                      *op++ = 0;
```

```
                *op++ = 0;
                *op++ = 0;
              }
            else
              {
                if (rt->gateway.s_addr == 0)
                  cp=(u_char *)& rt->ifa;
                else
                  cp = (u_char *) & rt->gateway;
                *op++=32;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
                *op++ = *cp++;
              }
             }
           else
             {
                cp = (u_char *) & rt->dest6;
           *op++=1;
           *op++=2;
           *op++=128;
           *op++ = *cp++;
           *op++ = *cp++;
           *op++ = *cp++;
           *op++ = *cp++;

           *op++ = *cp++;
           *op++ = *cp++;
           *op++ = *cp++;
           *op++ = *cp++;

           *op++ = *cp++;
           *op++ = *cp++;
           *op++ = *cp++;
           *op++ = *cp++;

             *op++ = *cp++;
           *op++ = *cp++;
           *op++ = *cp++;
           *op++ = *cp++;
           *op++=calcul_prefix(&rt->netmask6,sizeof(struct in6_addr));
           *op++=2;
            if (check_all_zero(&rt->gateway6,sizeof(struct in6_addr)) == 1
               &&
               check_all_zero(&rt->ifa6,sizeof(struct in6_addr)) == 1 )
```

```
                  )
                  {
105                 *op++=128;
                     *op++ = 0;
                     *op++ = 0;
                     *op++ = 0;
                     *op++ = 0;
110
                     *op++ = 0;
                     *op++ = 0;
                     *op++ = 0;
                     *op++ = 0;
115
                     *op++ = 0;
                     *op++ = 0;
                     *op++ = 0;
                     *op++ = 0;
120
                     *op++ = 0;
                     *op++ = 0;
                     *op++ = 0;
                     *op++ = 0;
125               }
                 else
                   {
                     if (check_all_zero(&rt->gateway6,sizeof(struct in6_addr)) == 1)
                       cp=(u_char *)& rt->ifa6;
130                  else
                       cp = (u_char *) & rt->gateway6;
                     *op++=128;
                     *op++ = *cp++;
                     *op++ = *cp++;
135                  *op++ = *cp++;
                     *op++ = *cp++;

                      *op++ = *cp++;
               *op++ = *cp++;
140            *op++ = *cp++;
               *op++ = *cp++;

               *op++ = *cp++;
               *op++ = *cp++;
145            *op++ = *cp++;
               *op++ = *cp++;

                 *op++ = *cp++;
```

```
                *op++ = *cp++;
150             *op++ = *cp++;
                *op++ = *cp++;
                    }
                  }
                 if(rt->type==1)
155               oid_compare_length=25;
                else
                  oid_compare_length=49;

                result = snmp_oid_compare(Current,oid_compare_length,  name,*length);
160
                  if ((exact && (result == 0))
                    || (!exact && (result > 0)))
                    break;
            }
165         if (rt == NULL)
                return NULL;
             if(rt->type==1)
                  oid_compare_length=25;
                else
170               oid_compare_length=49;
                     memcpy(name, Current, oid_compare_length * sizeof(oid));
             *length = oid_compare_length;

    *write_method = write_rte;
175     *var_len = sizeof long_return;
        switch (vp->magic) {

        case INETCIDROUTEINSTANCE: long_return = 1;
           return (u_char *) & long_return;
180
        case INETCIDROUTEDESTTYPE: long_return = rt->type;
           return (u_char *) & long_return;

        case INETCIDROUTEDEST:
185        switch (rt->type){
              case 1:
                                free_buf(buf);

                                if (inet_ntop(AF_INET,(void *)&rt->dest, buf, sizeof(buf)) != NULL)
190                                *var_len = (size_t)longueur_buf(buf);

                                long_return = (long int)buf;

                                 break;
```

```
195
             case 2:
                             free_buf6(buf6);
                             if (inet_ntop(AF_INET6,(void *)&rt->dest6, buf6, sizeof(buf6)) != NULL)
                                *var_len = (size_t)longueur_buf6(buf6);
200
                              long_return = (long int)buf6;

                              break;
             }
205
            return (u_char *)long_return;

         case INETCIDROUTEIFINDEX:
             long_return = rt->index;
             return (u_char *) & long_return;

         case INETCIDROUTEMETRIC1:
               long_return = rt->hdr->rtm_rmx.rmx_hopcount + 1;
               return (u_char *) & long_return;
215
         case INETCIDROUTEMETRIC2:
           long_return=-1;
             return (u_char *) & long_return;

220      case INETCIDROUTEMETRIC3:
           long_return=-1;
             return (u_char *) & long_return;

         case INETCIDROUTEMETRIC4:
225        long_return=-1;
             return (u_char *) & long_return;

         case INETCIDROUTEMETRIC5:
             long_return=-1;
             return (u_char *) & long_return;
230

         case INETCIDROUTENEXTHOP:
           switch(rt->type)
             {
235          case 1:

               if (rt->gateway.s_addr == 0 && rt->ifa.s_addr == 0)
                 {
                    free_buf(buf);
240                 *var_len = (size_t)0;
```

```
                long_return = (long int)buf;
            }
          else{ if (rt->gateway.s_addr == 0)
            {
                free_buf(buf);
                if (inet_ntop(AF_INET,(void *)&rt->ifa, buf, sizeof(buf)) != NULL)
                *var_len = (size_t)longueur_buf(buf);
                long_return = (long int)buf;
            }
          else
            {
                free_buf(buf);
                if (inet_ntop(AF_INET,(void *)&rt->gateway, buf, sizeof(buf)) != NULL)
                *var_len = (size_t)longueur_buf(buf);
                long_return = (long int)buf;
            }
          }
          break;

        case 2:
            if (check_all_zero(&rt->gateway6,sizeof(struct in6_addr)) == 1
               &&
               check_all_zero(&rt->ifa6,sizeof(struct in6_addr)) == 1
                )
            {
               free_buf6(buf6);
                *var_len = (size_t)0;
                long_return = (long int)buf6;
            }
          else{ if (check_all_zero(&rt->gateway6,sizeof(struct in6_addr)) == 1)
            {
               free_buf6(buf6);
               if (inet_ntop(AF_INET6,(void *)&rt->ifa6, buf6, sizeof(buf6)) != NULL)
                           *var_len = (size_t)longueur_buf6(buf6);
                           long_return = (long int)buf6;
            }
          else
            {
                free_buf6(buf6);
                if (inet_ntop(AF_INET6,(void *)&rt->gateway6, buf6, sizeof(buf6)) != NULL)
                           *var_len = (size_t)longueur_buf6(buf6);
                           long_return = (long int)buf6;
            }
          }
        }
```

```
            return (u_char *) long_return;


290       case INETCIDROUTENEXTHOPTYPE:
            if(rt->type==1)
              long_return=1;
            else
              long_return=2;
295           return (u_char *) &long_return;


          case INETCIDROUTETYPE:
              if (rt->hdr->rtm_flags & RTF_UP) {
300               if (rt->hdr->rtm_flags & RTF_GATEWAY) {
                      long_return = 4;        /*  indirect(4)  */
                  } else {
                      long_return = 3;        /*  direct(3)  */
                  }
305           } else {
                  long_return = 2;    /*  invalid(2)  */
              }
              return (u_char *) & long_return;

310       case INETCIDROUTEPROTO:
              long_return = (rt->hdr->rtm_flags & RTF_DYNAMIC) ? 4 : 2;
              return (u_char *) & long_return;

          case INETCIDROUTEAGE:
315   #if NO_DUMMY_VALUES
              return NULL;
      #endif
              long_return = 0;
              return (u_char *) & long_return;
320
          case INETCIDROUTEPFXLEN:
            //  long_return = rt->netmask.s_addr;
            if(rt->type==1)
            long_return =calcul_prefix(&rt->netmask,sizeof(struct in_addr));
325         else
               long_return =calcul_prefix(&rt->netmask6,sizeof(struct in6_addr));
              return (u_char *) & long_return;

          case INETCIDROUTESTATUS:
330          return NULL;
              *var_len = (size_t)nullOidLen;
              return (u_char *) nullOid;
```

```
        case INETCIDROUTENEXTHOPAS:
335         long_return = (u_int32_t) 0;
            return (u_char *) & long_return;

        default:
            DEBUGMSGTL(("snmpd", "unknown sub-id %d in var_ipForwardEntry\n",
340                     vp->magic));
        }
        return NULL;
    }
```

C Outputs

C.1 Output for the draft-ietf-ipngwg-RFC2011-update-00

C.1.1 Output for the ipAddressTable

```
IP-MIB::ipAddressAddrType.ipv4.32.127.0.0.1 = INTEGER: ipv4(1)
IP-MIB::ipAddressAddrType.ipv4.32.152.81.48.2 = INTEGER: ipv4(1)
IP-MIB::ipAddressAddrType.ipv4.32.152.81.49.1 = INTEGER: ipv4(1)
IP-MIB::ipAddressAddrType.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 =
INTEGER: ipv6(2)
IP-MIB::ipAddressAddrType.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0 =
INTEGER: ipv6(2)
IP-MIB::ipAddressAddrType.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 =
INTEGER: ipv6(2)
IP-MIB::ipAddressAddrType.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0 =
INTEGER: ipv6(2)
IP-MIB::ipAddressAddrType.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 =
INTEGER: ipv6(2)
IP-MIB::ipAddressAddrType.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.1 =
INTEGER: ipv6(2)
IP-MIB::ipAddressAddrType.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.93 =
INTEGER: ipv6(2)
IP-MIB::ipAddressAddrType.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.138
= INTEGER: ipv6(2)
IP-MIB::ipAddressAddr.ipv4.32.127.0.0.1 = STRING: "127.0.0.1"
IP-MIB::ipAddressAddr.ipv4.32.152.81.48.2 = STRING: "152.81.48.2"
IP-MIB::ipAddressAddr.ipv4.32.152.81.49.1 = STRING: "152.81.49.1"
IP-MIB::ipAddressAddr.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = STRING: "::1"
IP-MIB::ipAddressAddr.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0 =
STRING: "2001:660:301:32::"
IP-MIB::ipAddressAddr.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 =
STRING: "2001:660:301:32:201:2ff:fee3:608a"
IP-MIB::ipAddressAddr.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0 =
 STRING: "2001:660:301:33::"
IP-MIB::ipAddressAddr.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 =
STRING: "2001:660:301:33:201:2ff:fee3:605d"
IP-MIB::ipAddressAddr.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.1 =
STRING: "fe80::1"
IP-MIB::ipAddressAddr.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.93 =
STRING: "fe80::201:2ff:fee3:605d"
IP-MIB::ipAddressAddr.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.138 =
STRING: "fe80::201:2ff:fee3:608a"
IP-MIB::ipAddressIfIndex.ipv4.32.127.0.0.1 = INTEGER: 3
IP-MIB::ipAddressIfIndex.ipv4.32.152.81.48.2 = INTEGER: 1
IP-MIB::ipAddressIfIndex.ipv4.32.152.81.49.1 = INTEGER: 2
IP-MIB::ipAddressIfIndex.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 3
IP-MIB::ipAddressIfIndex.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0 =
INTEGER: 1
IP-MIB::ipAddressIfIndex.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 =
INTEGER: 1
IP-MIB::ipAddressIfIndex.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0 =
INTEGER: 2
IP-MIB::ipAddressIfIndex.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 =
INTEGER: 2
IP-MIB::ipAddressIfIndex.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.1 =
INTEGER: 3
IP-MIB::ipAddressIfIndex.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.93 =
INTEGER: 2
IP-MIB::ipAddressIfIndex.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.138 =
INTEGER: 1
IP-MIB::ipAddressType.ipv4.32.127.0.0.1 = INTEGER: unicast(1)
IP-MIB::ipAddressType.ipv4.32.152.81.48.2 = INTEGER: unicast(1)
IP-MIB::ipAddressType.ipv4.32.152.81.49.1 = INTEGER: unicast(1)
IP-MIB::ipAddressType.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 =
INTEGER: unicast(1)
IP-MIB::ipAddressType.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0 =
INTEGER: anycast(2)
IP-MIB::ipAddressType.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 =
INTEGER: unicast(1)
IP-MIB::ipAddressType.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0 =
INTEGER: anycast(2)
IP-MIB::ipAddressType.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 =
INTEGER: unicast(1)
IP-MIB::ipAddressType.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.1 =
INTEGER: unicast(1)
IP-MIB::ipAddressType.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.93 =
INTEGER: unicast(1)
IP-MIB::ipAddressType.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.138 =
INTEGER: unicast(1)
IP-MIB::ipAddressPrefix.ipv4.32.127.0.0.1 =
```

```
OID: IP-MIB::ipAddressPrefixPrefix.3.ipv4.32.127.0.0.0.8
IP-MIB::ipAddressPrefix.ipv4.32.152.81.48.2 =
OID: IP-MIB::ipAddressPrefixPrefix.1.ipv4.32.152.81.48.0.24
IP-MIB::ipAddressPrefix.ipv4.32.152.81.49.1 =
OID: IP-MIB::ipAddressPrefixPrefix.2.ipv4.32.152.81.49.0.24
IP-MIB::ipAddressPrefix.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 =
OID: IP-MIB::ipAddressPrefixPrefix.3.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.128
IP-MIB::ipAddressPrefix.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0 =
OID: IP-MIB::ipAddressPrefixPrefix.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.64
IP-MIB::ipAddressPrefix.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 =
OID: IP-MIB::ipAddressPrefixPrefix.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.64
IP-MIB::ipAddressPrefix.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0 =
OID: IP-MIB::ipAddressPrefixPrefix.2.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.64
IP-MIB::ipAddressPrefix.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 =
OID: IP-MIB::ipAddressPrefixPrefix.2.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.64
IP-MIB::ipAddressPrefix.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.1 =
OID: IP-MIB::ipAddressPrefixPrefix.3.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64
IP-MIB::ipAddressPrefix.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.93 =
OID: IP-MIB::ipAddressPrefixPrefix.2.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64
IP-MIB::ipAddressPrefix.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.138 =
OID: IP-MIB::ipAddressPrefixPrefix.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64
```

C.1.2 Output for the ipAddressPrefixTable

```
IP-MIB::ipAddressPrefixIfIndex.1.ipv4.32.152.81.48.0.24 = INTEGER: 1
IP-MIB::ipAddressPrefixIfIndex.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.64 =
INTEGER: 1
IP-MIB::ipAddressPrefixIfIndex.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
INTEGER: 1
IP-MIB::ipAddressPrefixIfIndex.2.ipv4.32.152.81.49.0.24 = INTEGER: 2
IP-MIB::ipAddressPrefixIfIndex.2.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.64 =
INTEGER: 2
IP-MIB::ipAddressPrefixIfIndex.2.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
INTEGER: 2
IP-MIB::ipAddressPrefixIfIndex.3.ipv4.32.127.0.0.0.8 = INTEGER: 3
IP-MIB::ipAddressPrefixIfIndex.3.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.128 =
INTEGER: 3
IP-MIB::ipAddressPrefixIfIndex.3.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
INTEGER: 3
IP-MIB::ipAddressPrefixType.1.ipv4.32.152.81.48.0.24 = INTEGER: ipv4(1)
IP-MIB::ipAddressPrefixType.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.64 =
INTEGER: ipv6(2)
IP-MIB::ipAddressPrefixType.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
INTEGER: ipv6(2)
IP-MIB::ipAddressPrefixType.2.ipv4.32.152.81.49.0.24 = INTEGER: ipv4(1)
IP-MIB::ipAddressPrefixType.2.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.64 =
INTEGER: ipv6(2)
IP-MIB::ipAddressPrefixType.2.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
INTEGER: ipv6(2)
IP-MIB::ipAddressPrefixType.3.ipv4.32.127.0.0.0.8 = INTEGER: ipv4(1)
IP-MIB::ipAddressPrefixType.3.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.128 =
INTEGER: ipv6(2)
IP-MIB::ipAddressPrefixType.3.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
INTEGER: ipv6(2)
IP-MIB::ipAddressPrefixPrefix.1.ipv4.32.152.81.48.0.24 = STRING: "152.81.48.0"
IP-MIB::ipAddressPrefixPrefix.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.64 =
STRING: "2001:660:301:32::"
IP-MIB::ipAddressPrefixPrefix.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
STRING: "fe80::"
IP-MIB::ipAddressPrefixPrefix.2.ipv4.32.152.81.49.0.24 = STRING: "152.81.49.0"
IP-MIB::ipAddressPrefixPrefix.2.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.64 =
STRING: "2001:660:301:33::"
IP-MIB::ipAddressPrefixPrefix.2.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
STRING: "fe80::"
IP-MIB::ipAddressPrefixPrefix.3.ipv4.32.127.0.0.0.8 = STRING: "127.0.0.0"
IP-MIB::ipAddressPrefixPrefix.3.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.128 =
STRING: "::1"
IP-MIB::ipAddressPrefixPrefix.3.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
STRING: "fe80::"
IP-MIB::ipAddressPrefixLength.1.ipv4.32.152.81.48.0.24 = Gauge32: 24
IP-MIB::ipAddressPrefixLength.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.64 =
Gauge32: 64
IP-MIB::ipAddressPrefixLength.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
Gauge32: 64
IP-MIB::ipAddressPrefixLength.2.ipv4.32.152.81.49.0.24 = Gauge32: 24
IP-MIB::ipAddressPrefixLength.2.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.64 =
Gauge32: 64
IP-MIB::ipAddressPrefixLength.2.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
Gauge32: 64
IP-MIB::ipAddressPrefixLength.3.ipv4.32.127.0.0.0.8 = Gauge32: 8
IP-MIB::ipAddressPrefixLength.3.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.128 =
Gauge32: 128
IP-MIB::ipAddressPrefixLength.3.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
```

```
Gauge32: 64
IP-MIB::ipAddressPrefixAutonomousFlag.1.ipv4.32.152.81.48.0.24 = INTEGER: false(2)
IP-MIB::ipAddressPrefixAutonomousFlag.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.64 =
INTEGER: true(1)
IP-MIB::ipAddressPrefixAutonomousFlag.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
INTEGER: true(1)
IP-MIB::ipAddressPrefixAutonomousFlag.2.ipv4.32.152.81.49.0.24 = INTEGER: false(2)
IP-MIB::ipAddressPrefixAutonomousFlag.2.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.64 =
INTEGER: true(1)
IP-MIB::ipAddressPrefixAutonomousFlag.2.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
INTEGER: true(1)
IP-MIB::ipAddressPrefixAutonomousFlag.3.ipv4.32.127.0.0.0.8 = INTEGER: false(2)
IP-MIB::ipAddressPrefixAutonomousFlag.3.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.1.128 =
INTEGER: true(1)
IP-MIB::ipAddressPrefixAutonomousFlag.3.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
INTEGER: true(1)
IP-MIB::ipAddressPrefixAdvPreferredLifetime.1.ipv4.32.152.81.48.0.24 =
Gauge32: 4294967295 seconds
IP-MIB::ipAddressPrefixAdvPreferredLifetime.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.64 =
Gauge32: 86400 seconds
IP-MIB::ipAddressPrefixAdvPreferredLifetime.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
Gauge32: 86400 seconds
IP-MIB::ipAddressPrefixAdvPreferredLifetime.2.ipv4.32.152.81.49.0.24 =
Gauge32: 4294967295 seconds
IP-MIB::ipAddressPrefixAdvPreferredLifetime.2.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.64 =
Gauge32: 86400 seconds
IP-MIB::ipAddressPrefixAdvPreferredLifetime.2.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
Gauge32: 86400 seconds
IP-MIB::ipAddressPrefixAdvPreferredLifetime.3.ipv4.32.127.0.0.0.8 =
Gauge32: 4294967295 seconds
IP-MIB::ipAddressPrefixAdvPreferredLifetime.3.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.1.128 =
Gauge32: 86400 seconds
IP-MIB::ipAddressPrefixAdvPreferredLifetime.3.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
Gauge32: 86400 seconds
IP-MIB::ipAddressPrefixAdvValidLifetime.1.ipv4.32.152.81.48.0.24 =
Gauge32: 4294967295 seconds
IP-MIB::ipAddressPrefixAdvValidLifetime.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.64 =
Gauge32: 604800 seconds
IP-MIB::ipAddressPrefixAdvValidLifetime.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
Gauge32: 604800 seconds
IP-MIB::ipAddressPrefixAdvValidLifetime.2.ipv4.32.152.81.49.0.24 =
Gauge32: 4294967295 seconds
IP-MIB::ipAddressPrefixAdvValidLifetime.2.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.64 =
Gauge32: 604800 seconds
IP-MIB::ipAddressPrefixAdvValidLifetime.2.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
Gauge32: 604800 seconds
IP-MIB::ipAddressPrefixAdvValidLifetime.3.ipv4.32.127.0.0.0.8 =
Gauge32: 4294967295 seconds
IP-MIB::ipAddressPrefixAdvValidLifetime.3.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.1.128 =
Gauge32: 604800 seconds
IP-MIB::ipAddressPrefixAdvValidLifetime.3.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 =
Gauge32: 604800 seconds
```

C.1.3 Output for ipIfStatstable

```
IP-MIB::ipIfStatsAFType.ipv4.1 = INTEGER: ipv4(1)
IP-MIB::ipIfStatsAFType.ipv4.2 = INTEGER: ipv4(1)
IP-MIB::ipIfStatsAFType.ipv4.3 = INTEGER: ipv4(1)
IP-MIB::ipIfStatsAFType.ipv6.1 = INTEGER: ipv6(2)
IP-MIB::ipIfStatsAFType.ipv6.2 = INTEGER: ipv6(2)
IP-MIB::ipIfStatsAFType.ipv6.3 = INTEGER: ipv6(2)
IP-MIB::ipIfStatsIfIndex.ipv4.1 = INTEGER: 1
IP-MIB::ipIfStatsIfIndex.ipv4.2 = INTEGER: 2
IP-MIB::ipIfStatsIfIndex.ipv4.3 = INTEGER: 3
IP-MIB::ipIfStatsIfIndex.ipv6.1 = INTEGER: 1
IP-MIB::ipIfStatsIfIndex.ipv6.2 = INTEGER: 2
IP-MIB::ipIfStatsIfIndex.ipv6.3 = INTEGER: 3
IP-MIB::ipIfStatsInReceives.ipv4.1 = Counter32: 4874108
IP-MIB::ipIfStatsInReceives.ipv4.2 = Counter32: 2265286
IP-MIB::ipIfStatsInReceives.ipv4.3 = Counter32: 1447081
IP-MIB::ipIfStatsInReceives.ipv6.1 = Counter32: 4874108
IP-MIB::ipIfStatsInReceives.ipv6.2 = Counter32: 2265286
IP-MIB::ipIfStatsInReceives.ipv6.3 = Counter32: 1447087
IP-MIB::ipIfStatsInHdrErrors.ipv4.1 = Counter32: 0
IP-MIB::ipIfStatsInHdrErrors.ipv4.2 = Counter32: 0
IP-MIB::ipIfStatsInHdrErrors.ipv4.3 = Counter32: 0
IP-MIB::ipIfStatsInHdrErrors.ipv6.1 = Counter32: 0
IP-MIB::ipIfStatsInHdrErrors.ipv6.2 = Counter32: 0
IP-MIB::ipIfStatsInHdrErrors.ipv6.3 = Counter32: 0
IP-MIB::ipIfStatsInUnknownProtos.ipv4.1 = Counter32: 0
IP-MIB::ipIfStatsInUnknownProtos.ipv4.2 = Counter32: 0
IP-MIB::ipIfStatsInUnknownProtos.ipv4.3 = Counter32: 0
IP-MIB::ipIfStatsInUnknownProtos.ipv6.1 = Counter32: 0
```

```
IP-MIB::ipIfStatsInUnknownProtos.ipv6.2 = Counter32: 0
IP-MIB::ipIfStatsInUnknownProtos.ipv6.3 = Counter32: 0
IP-MIB::ipIfStatsInMcastPkts.ipv4.1 = Counter32: 195147
IP-MIB::ipIfStatsInMcastPkts.ipv4.2 = Counter32: 56850
IP-MIB::ipIfStatsInMcastPkts.ipv4.3 = Counter32: 0
IP-MIB::ipIfStatsInMcastPkts.ipv6.1 = Counter32: 195147
IP-MIB::ipIfStatsInMcastPkts.ipv6.2 = Counter32: 56850
IP-MIB::ipIfStatsInMcastPkts.ipv6.3 = Counter32: 0
IP-MIB::ipIfStatsOutMcastPkts.ipv4.1 = Counter32: 62628
IP-MIB::ipIfStatsOutMcastPkts.ipv4.2 = Counter32: 46877
IP-MIB::ipIfStatsOutMcastPkts.ipv4.3 = Counter32: 0
IP-MIB::ipIfStatsOutMcastPkts.ipv6.1 = Counter32: 62628
IP-MIB::ipIfStatsOutMcastPkts.ipv6.2 = Counter32: 46877
IP-MIB::ipIfStatsOutMcastPkts.ipv6.3 = Counter32: 0
IP-MIB::ipIfStatsInOctets.ipv4.1 = Counter32: 3822874128
IP-MIB::ipIfStatsInOctets.ipv4.2 = Counter32: 3077937641
IP-MIB::ipIfStatsInOctets.ipv4.3 = Counter32: 183696160
IP-MIB::ipIfStatsInOctets.ipv6.1 = Counter32: 3822874128
IP-MIB::ipIfStatsInOctets.ipv6.2 = Counter32: 3077937641
IP-MIB::ipIfStatsInOctets.ipv6.3 = Counter32: 183696618
IP-MIB::ipIfStatsOutOctets.ipv4.1 = Counter32: 3822874128
IP-MIB::ipIfStatsOutOctets.ipv4.2 = Counter32: 3077937641
IP-MIB::ipIfStatsOutOctets.ipv4.3 = Counter32: 183697076
IP-MIB::ipIfStatsOutOctets.ipv6.1 = Counter32: 3822874128
IP-MIB::ipIfStatsOutOctets.ipv6.2 = Counter32: 3077937641
IP-MIB::ipIfStatsOutOctets.ipv6.3 = Counter32: 183697534
```

C.1.4 Output for the ipv6InterfaceTable

```
IP-MIB::ipv6InterfaceIfIndex.1 = INTEGER: 1
IP-MIB::ipv6InterfaceIfIndex.2 = INTEGER: 2
IP-MIB::ipv6InterfaceIfIndex.3 = INTEGER: 3
IP-MIB::ipv6InterfaceEffectiveMtu.1 = Gauge32: 1496 octets
IP-MIB::ipv6InterfaceEffectiveMtu.2 = Gauge32: 1496 octets
IP-MIB::ipv6InterfaceEffectiveMtu.3 = Gauge32: 16384 octets
IP-MIB::ipv6InterfaceIdentifier.1 = STRING: 201:2ff:fee3:608a
IP-MIB::ipv6InterfaceIdentifier.2 = STRING: 201:2ff:fee3:605d
IP-MIB::ipv6InterfaceIdentifier.3 = STRING: 0:0:0:1
IP-MIB::ipv6InterfaceIdentifierLength.1 = INTEGER: 64 bits
IP-MIB::ipv6InterfaceIdentifierLength.2 = INTEGER: 64 bits
IP-MIB::ipv6InterfaceIdentifierLength.3 = INTEGER: 64 bits
IP-MIB::ipv6InterfacePhysicalAddress.1 = STRING: 0:1:2:e3:60:8a
IP-MIB::ipv6InterfacePhysicalAddress.2 = STRING: 0:1:2:e3:60:5d
IP-MIB::ipv6InterfacePhysicalAddress.3 = STRING:
```

C.1.5 Output for the ipv4IfTable

```
IP-MIB::ipv4IfIndex.1 = INTEGER: 1
IP-MIB::ipv4IfIndex.2 = INTEGER: 2
IP-MIB::ipv4IfIndex.3 = INTEGER: 3
```

C.1.6 Output for the inetNetToMediaTable

```
% snmpwalk -v 2c -c public aria inetNetToMediaTable

IP-MIB::inetNetToMediaNetAddressType.1.ipv4.32.152.81.48.2 = INTEGER: ipv4(1)
IP-MIB::inetNetToMediaNetAddressType.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0 =
INTEGER: ipv6(2)
IP-MIB::inetNetToMediaNetAddressType.1.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 =
INTEGER: ipv6(2)
IP-MIB::inetNetToMediaNetAddressType.1.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.138 =
INTEGER: ipv6(2)
IP-MIB::inetNetToMediaNetAddressType.2.ipv4.32.152.81.49.1 = INTEGER: ipv4(1)
IP-MIB::inetNetToMediaNetAddressType.2.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0 = INTEGER: ipv6(2)
IP-MIB::inetNetToMediaNetAddressType.2.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 =
INTEGER: ipv6(2)
IP-MIB::inetNetToMediaNetAddressType.2.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.93 =
INTEGER: ipv6(2)
IP-MIB::inetNetToMediaNetAddressType.3.ipv4.32.127.0.0.1 = INTEGER: ipv4(1)
IP-MIB::inetNetToMediaNetAddressType.3.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: ipv6(2)
IP-MIB::inetNetToMediaNetAddressType.3.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: ipv6(2)
IP-MIB::inetNetToMediaNetAddress.1.ipv4.32.152.81.48.2 = STRING: "152.81.48.2"
IP-MIB::inetNetToMediaNetAddress.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0 =
STRING: "2001:660:301:32::"
IP-MIB::inetNetToMediaNetAddress.1.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 =
STRING: "2001:660:301:32:201:2ff:fee3:608a"
IP-MIB::inetNetToMediaNetAddress.1.ipv6.128.254.128.0.0.0.0.0.2.1.2.255.254.227.96.138 =
STRING: "fe80::201:2ff:fee3:608a"
IP-MIB::inetNetToMediaNetAddress.2.ipv4.32.152.81.49.1 = STRING: "152.81.49.1"
IP-MIB::inetNetToMediaNetAddress.2.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0 =
```

```
STRING: "2001:660:301:33::"
IP-MIB::inetNetToMediaNetAddress.2.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 =
STRING: "2001:660:301:33:201:2ff:fee3:605d"
IP-MIB::inetNetToMediaNetAddress.2.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.93 =
STRING: "fe80::201:2ff:fee3:605d"
IP-MIB::inetNetToMediaNetAddress.3.ipv4.32.127.0.0.1 = STRING: "127.0.0.1"
IP-MIB::inetNetToMediaNetAddress.3.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = STRING: "::1"
IP-MIB::inetNetToMediaNetAddress.3.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = STRING: "fe80::1"
IP-MIB::inetNetToMediaPhysAddress.1.ipv4.32.152.81.48.2 = STRING: 0:1:2:e3:60:8a
IP-MIB::inetNetToMediaPhysAddress.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0 =
STRING: 0:1:2:e3:60:8a
IP-MIB::inetNetToMediaPhysAddress.1.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 =
STRING: 0:1:2:e3:60:8a
IP-MIB::inetNetToMediaPhysAddress.1.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.138 =
STRING: 0:1:2:e3:60:8a
IP-MIB::inetNetToMediaPhysAddress.2.ipv4.32.152.81.49.1 = STRING: 0:1:2:e3:60:5d
IP-MIB::inetNetToMediaPhysAddress.2.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0 =
STRING: 0:1:2:e3:60:5d
IP-MIB::inetNetToMediaPhysAddress.2.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 =
STRING: 0:1:2:e3:60:5d
IP-MIB::inetNetToMediaPhysAddress.2.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.93 =
STRING: 0:1:2:e3:60:5d
IP-MIB::inetNetToMediaPhysAddress.3.ipv4.32.127.0.0.1 = STRING:
IP-MIB::inetNetToMediaPhysAddress.3.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = STRING:
IP-MIB::inetNetToMediaPhysAddress.3.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = STRING:
IP-MIB::inetNetToMediaType.1.ipv4.32.152.81.48.2 = INTEGER: other(1)
IP-MIB::inetNetToMediaType.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0 = INTEGER: other(1)
IP-MIB::inetNetToMediaType.1.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 =
INTEGER: other(1)
IP-MIB::inetNetToMediaType.1.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.138 =
INTEGER: other(1)
IP-MIB::inetNetToMediaType.2.ipv4.32.152.81.49.1 = INTEGER: other(1)
IP-MIB::inetNetToMediaType.2.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0 = INTEGER: other(1)
IP-MIB::inetNetToMediaType.2.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 =
INTEGER: other(1)
IP-MIB::inetNetToMediaType.2.ipv6.128.254.128.0.0.0.0.0.0.2.1.2.255.254.227.96.93 =
INTEGER: other(1)
IP-MIB::inetNetToMediaType.3.ipv4.32.127.0.0.1 = INTEGER: other(1)
IP-MIB::inetNetToMediaType.3.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: other(1)
IP-MIB::inetNetToMediaType.3.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: other(1)
```

C.1.7 Output for the ipv6ScopeIdTable

```
% snmpwalk -v 2c -c public aria ipv6ScopeIdTable

IP-MIB::ipv6ScopeIdIfIndex.1 = INTEGER: 1
IP-MIB::ipv6ScopeIdIfIndex.2 = INTEGER: 2
IP-MIB::ipv6ScopeIdIfIndex.3 = INTEGER: 3
IP-MIB::ipv6ScopeIdLinkLocal.1 = Gauge32: 1
IP-MIB::ipv6ScopeIdLinkLocal.2 = Gauge32: 2
IP-MIB::ipv6ScopeIdLinkLocal.3 = Gauge32: 3
IP-MIB::ipv6ScopeIdSubnetLocal.1 = Gauge32: 0
IP-MIB::ipv6ScopeIdSubnetLocal.2 = Gauge32: 0
IP-MIB::ipv6ScopeIdSubnetLocal.3 = Gauge32: 0
IP-MIB::ipv6ScopeIdAdminLocal.1 = Gauge32: 0
IP-MIB::ipv6ScopeIdAdminLocal.2 = Gauge32: 0
IP-MIB::ipv6ScopeIdAdminLocal.3 = Gauge32: 0
IP-MIB::ipv6ScopeIdSiteLocal.1 = Gauge32: 0
IP-MIB::ipv6ScopeIdSiteLocal.2 = Gauge32: 0
IP-MIB::ipv6ScopeIdSiteLocal.3 = Gauge32: 0
IP-MIB::ipv6ScopeId6.1 = Gauge32: 0
IP-MIB::ipv6ScopeId6.2 = Gauge32: 0
IP-MIB::ipv6ScopeId6.3 = Gauge32: 0
IP-MIB::ipv6ScopeId7.1 = Gauge32: 0
IP-MIB::ipv6ScopeId7.2 = Gauge32: 0
IP-MIB::ipv6ScopeId7.3 = Gauge32: 0
IP-MIB::ipv6ScopeIdOrganizationLocal.1 = Gauge32: 0
IP-MIB::ipv6ScopeIdOrganizationLocal.2 = Gauge32: 0
IP-MIB::ipv6ScopeIdOrganizationLocal.3 = Gauge32: 0
IP-MIB::ipv6ScopeId9.1 = Gauge32: 0
IP-MIB::ipv6ScopeId9.2 = Gauge32: 0
IP-MIB::ipv6ScopeId9.3 = Gauge32: 0
IP-MIB::ipv6ScopeIdA.1 = Gauge32: 0
IP-MIB::ipv6ScopeIdA.2 = Gauge32: 0
IP-MIB::ipv6ScopeIdA.3 = Gauge32: 0
IP-MIB::ipv6ScopeIdB.1 = Gauge32: 0
IP-MIB::ipv6ScopeIdB.2 = Gauge32: 0
IP-MIB::ipv6ScopeIdB.3 = Gauge32: 0
IP-MIB::ipv6ScopeIdC.1 = Gauge32: 0
IP-MIB::ipv6ScopeIdC.2 = Gauge32: 0
IP-MIB::ipv6ScopeIdC.3 = Gauge32: 0
IP-MIB::ipv6ScopeIdD.1 = Gauge32: 0
IP-MIB::ipv6ScopeIdD.2 = Gauge32: 0
IP-MIB::ipv6ScopeIdD.3 = Gauge32: 0
```

C.1.8 Output for the InetIcmpTable

```
% snmpwalk -v 2c -c public aria inetIcmpTable

IP-MIB::inetIcmpAFType.ipv4.0 = INTEGER: ipv4(1)
IP-MIB::inetIcmpAFType.ipv6.0 = INTEGER: ipv6(2)
IP-MIB::inetIcmpIfIndex.ipv4.0 = INTEGER: 0
IP-MIB::inetIcmpIfIndex.ipv6.0 = INTEGER: 0
IP-MIB::inetIcmpInMsgs.ipv4.0 = Counter32: 8510
IP-MIB::inetIcmpInMsgs.ipv6.0 = Counter32: 137986
IP-MIB::inetIcmpInErrors.ipv4.0 = Counter32: 0
IP-MIB::inetIcmpInErrors.ipv6.0 = Counter32: 0
IP-MIB::inetIcmpOutMsgs.ipv4.0 = Counter32: 503
IP-MIB::inetIcmpOutMsgs.ipv6.0 = Counter32: 14278
IP-MIB::inetIcmpOutErrors.ipv4.0 = Counter32: 0
IP-MIB::inetIcmpOutErrors.ipv6.0 = Counter32: 0
```

C.1.9 Output for the inetIcmpMsgTable

```
% snmpwalk -v 2c -c public aria inetIcmpMsgTable

IP-MIB::inetIcmpMsgAFType.ipv4.0.1.256 = INTEGER: ipv4(1)
IP-MIB::inetIcmpMsgAFType.ipv4.0.3.256 = INTEGER: ipv4(1)
IP-MIB::inetIcmpMsgAFType.ipv4.0.4.256 = INTEGER: ipv4(1)
IP-MIB::inetIcmpMsgAFType.ipv4.0.134.0 = INTEGER: ipv4(1)
IP-MIB::inetIcmpMsgAFType.ipv4.0.137.0 = INTEGER: ipv4(1)
IP-MIB::inetIcmpMsgAFType.ipv6.0.1.0 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.1.1 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.1.2 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.1.3 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.1.4 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.1.256 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.2.0 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.3.0 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.3.1 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.3.256 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.4.0 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.4.1 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.4.2 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.4.256 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.131.0 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.132.0 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.134.0 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.135.0 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.136.0 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgAFType.ipv6.0.137.0 = INTEGER: ipv6(2)
IP-MIB::inetIcmpMsgIfIndex.ipv4.0.1.256 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv4.0.3.256 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv4.0.4.256 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv4.0.134.0 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv4.0.137.0 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.1.0 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.1.1 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.1.2 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.1.3 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.1.4 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.1.256 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.2.0 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.3.0 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.3.1 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.3.256 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.4.0 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.4.1 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.4.2 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.4.256 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.131.0 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.132.0 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.134.0 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.135.0 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.136.0 = INTEGER: 0
IP-MIB::inetIcmpMsgIfIndex.ipv6.0.137.0 = INTEGER: 0
IP-MIB::inetIcmpMsgType.ipv4.0.1.256 = INTEGER: 1
IP-MIB::inetIcmpMsgType.ipv4.0.3.256 = INTEGER: 3
IP-MIB::inetIcmpMsgType.ipv4.0.4.256 = INTEGER: 4
IP-MIB::inetIcmpMsgType.ipv4.0.134.0 = INTEGER: 134
IP-MIB::inetIcmpMsgType.ipv4.0.137.0 = INTEGER: 137
IP-MIB::inetIcmpMsgType.ipv6.0.1.0 = INTEGER: 1
IP-MIB::inetIcmpMsgType.ipv6.0.1.1 = INTEGER: 1
IP-MIB::inetIcmpMsgType.ipv6.0.1.2 = INTEGER: 1
IP-MIB::inetIcmpMsgType.ipv6.0.1.3 = INTEGER: 1
IP-MIB::inetIcmpMsgType.ipv6.0.1.4 = INTEGER: 1
IP-MIB::inetIcmpMsgType.ipv6.0.1.256 = INTEGER: 1
IP-MIB::inetIcmpMsgType.ipv6.0.2.0 = INTEGER: 2
IP-MIB::inetIcmpMsgType.ipv6.0.3.0 = INTEGER: 3
```

```
IP-MIB::inetIcmpMsgType.ipv6.0.3.1 = INTEGER: 3
IP-MIB::inetIcmpMsgType.ipv6.0.3.256 = INTEGER: 3
IP-MIB::inetIcmpMsgType.ipv6.0.4.0 = INTEGER: 4
IP-MIB::inetIcmpMsgType.ipv6.0.4.1 = INTEGER: 4
IP-MIB::inetIcmpMsgType.ipv6.0.4.2 = INTEGER: 4
IP-MIB::inetIcmpMsgType.ipv6.0.4.256 = INTEGER: 4
IP-MIB::inetIcmpMsgType.ipv6.0.131.0 = INTEGER: 131
IP-MIB::inetIcmpMsgType.ipv6.0.132.0 = INTEGER: 132
IP-MIB::inetIcmpMsgType.ipv6.0.134.0 = INTEGER: 134
IP-MIB::inetIcmpMsgType.ipv6.0.135.0 = INTEGER: 135
IP-MIB::inetIcmpMsgType.ipv6.0.136.0 = INTEGER: 136
IP-MIB::inetIcmpMsgType.ipv6.0.137.0 = INTEGER: 137
IP-MIB::inetIcmpMsgCode.ipv4.0.1.256 = INTEGER: 256
IP-MIB::inetIcmpMsgCode.ipv4.0.3.256 = INTEGER: 256
IP-MIB::inetIcmpMsgCode.ipv4.0.4.256 = INTEGER: 256
IP-MIB::inetIcmpMsgCode.ipv4.0.134.0 = INTEGER: 0
IP-MIB::inetIcmpMsgCode.ipv4.0.137.0 = INTEGER: 0
IP-MIB::inetIcmpMsgCode.ipv6.0.1.0 = INTEGER: 0
IP-MIB::inetIcmpMsgCode.ipv6.0.1.1 = INTEGER: 1
IP-MIB::inetIcmpMsgCode.ipv6.0.1.2 = INTEGER: 2
IP-MIB::inetIcmpMsgCode.ipv6.0.1.3 = INTEGER: 3
IP-MIB::inetIcmpMsgCode.ipv6.0.1.4 = INTEGER: 4
IP-MIB::inetIcmpMsgCode.ipv6.0.1.256 = INTEGER: 256
IP-MIB::inetIcmpMsgCode.ipv6.0.2.0 = INTEGER: 0
IP-MIB::inetIcmpMsgCode.ipv6.0.3.0 = INTEGER: 0
IP-MIB::inetIcmpMsgCode.ipv6.0.3.1 = INTEGER: 1
IP-MIB::inetIcmpMsgCode.ipv6.0.3.256 = INTEGER: 256
IP-MIB::inetIcmpMsgCode.ipv6.0.4.0 = INTEGER: 0
IP-MIB::inetIcmpMsgCode.ipv6.0.4.1 = INTEGER: 1
IP-MIB::inetIcmpMsgCode.ipv6.0.4.2 = INTEGER: 2
IP-MIB::inetIcmpMsgCode.ipv6.0.4.256 = INTEGER: 256
IP-MIB::inetIcmpMsgCode.ipv6.0.131.0 = INTEGER: 0
IP-MIB::inetIcmpMsgCode.ipv6.0.132.0 = INTEGER: 0
IP-MIB::inetIcmpMsgCode.ipv6.0.134.0 = INTEGER: 0
IP-MIB::inetIcmpMsgCode.ipv6.0.135.0 = INTEGER: 0
IP-MIB::inetIcmpMsgCode.ipv6.0.136.0 = INTEGER: 0
IP-MIB::inetIcmpMsgCode.ipv6.0.137.0 = INTEGER: 0
IP-MIB::inetIcmpMsgInPkts.ipv4.0.1.256 = Counter32: 493
IP-MIB::inetIcmpMsgInPkts.ipv4.0.3.256 = Counter32: 42
IP-MIB::inetIcmpMsgInPkts.ipv4.0.4.256 = Counter32: 0
IP-MIB::inetIcmpMsgInPkts.ipv4.0.134.0 = Counter32: 7951
IP-MIB::inetIcmpMsgInPkts.ipv4.0.137.0 = Counter32: 0
IP-MIB::inetIcmpMsgInPkts.ipv6.0.1.256 = Counter32: 104
IP-MIB::inetIcmpMsgInPkts.ipv6.0.2.0 = Counter32: 0
IP-MIB::inetIcmpMsgInPkts.ipv6.0.3.256 = Counter32: 1189
IP-MIB::inetIcmpMsgInPkts.ipv6.0.4.256 = Counter32: 0
IP-MIB::inetIcmpMsgInPkts.ipv6.0.131.0 = Counter32: 0
IP-MIB::inetIcmpMsgInPkts.ipv6.0.132.0 = Counter32: 0
IP-MIB::inetIcmpMsgInPkts.ipv6.0.134.0 = Counter32: 126335
IP-MIB::inetIcmpMsgInPkts.ipv6.0.135.0 = Counter32: 5060
IP-MIB::inetIcmpMsgInPkts.ipv6.0.136.0 = Counter32: 5277
IP-MIB::inetIcmpMsgInPkts.ipv6.0.137.0 = Counter32: 1
IP-MIB::inetIcmpMsgOutPkts.ipv4.0.1.256 = Counter32: 484
IP-MIB::inetIcmpMsgOutPkts.ipv4.0.3.256 = Counter32: 3
IP-MIB::inetIcmpMsgOutPkts.ipv4.0.4.256 = Counter32: 0
IP-MIB::inetIcmpMsgOutPkts.ipv4.0.134.0 = Counter32: 0
IP-MIB::inetIcmpMsgOutPkts.ipv4.0.137.0 = Counter32: 0
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.1.0 = Counter32: 29
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.1.1 = Counter32: 0
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.1.2 = Counter32: 0
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.1.3 = Counter32: 70
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.1.4 = Counter32: 0
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.2.0 = Counter32: 1
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.3.0 = Counter32: 11
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.3.1 = Counter32: 0
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.4.0 = Counter32: 0
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.4.1 = Counter32: 0
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.4.2 = Counter32: 0
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.131.0 = Counter32: 18
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.132.0 = Counter32: 0
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.134.0 = Counter32: 3454
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.135.0 = Counter32: 5492
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.136.0 = Counter32: 5060
IP-MIB::inetIcmpMsgOutPkts.ipv6.0.137.0 = Counter32: 1280
```

C.2 Output of the draft-ietf-ipngwg-RFC2012-update-00

```
TCP-MIB::tcpRtoAlgorithm.0 = INTEGER: vanj(4)
TCP-MIB::tcpRtoMin.0 = INTEGER: 1000 milliseconds
TCP-MIB::tcpRtoMax.0 = INTEGER: 64000 milliseconds
TCP-MIB::tcpActiveOpens.0 = Counter32: 4270
TCP-MIB::tcpPassiveOpens.0 = Counter32: 262
TCP-MIB::tcpAttemptFails.0 = Counter32: 21
```

```
TCP-MIB::tcpEstabResets.0 = Counter32: 27
TCP-MIB::tcpCurrEstab.0 = Gauge32: 0
TCP-MIB::tcpInSegs.0 = Counter32: 43228741
TCP-MIB::tcpOutSegs.0 = Counter32: 29547347
TCP-MIB::tcpRetransSegs.0 = Counter32: 121
TCP-MIB::tcpInErrs.0 = Counter32: 0
TCP-MIB::tcpOutRsts.0 = Counter32: 3642
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.0.32.0.0.0.0.0 = INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60108.32.0.0.0.0.226 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60108.32.0.0.0.0.243 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60108.32.0.0.0.0.8437 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60108.32.0.0.0.0.16631 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60108.32.0.0.0.0.24825 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60108.32.0.0.0.0.33019 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60108.32.0.0.0.0.41179 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60108.32.0.0.0.0.41213 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60108.32.0.0.0.0.49356 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60108.32.0.0.0.0.49373 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60108.32.0.0.0.0.57567 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60108.32.0.0.0.0.57584 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60364.32.0.0.0.0.16409 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60364.32.0.0.0.0.24603 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60364.32.0.0.0.0.32848 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60364.32.0.0.0.0.49168 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60364.32.0.0.0.0.49219 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60364.32.0.0.0.0.57362 =
INTEGER: closed(1)
TCP-MIB::tcpConnectionState.ipv4.32.0.0.0.0.60364.32.0.0.0.0.57481 =
INTEGER: closed(1)
```

C.3 Output of the draft-ietf-ipngwg-RFC2013-update-00

```
UDP-MIB::udpNoPorts.0 = Counter32: 889
UDP-MIB::udpInErrors.0 = Counter32: 43
UDP-MIB::udpListenerInstance.ipv4.32.0.0.0.0.0.1.0 = Gauge32: 0
UDP-MIB::udpListenerInstance.ipv4.32.0.0.0.0.57036.1.0 = Gauge32: 0
UDP-MIB::udpListenerInstance.ipv4.32.0.0.0.0.57292.1.0 = Gauge32: 0
```

C.4 Output of the draft-ietf-ipngwg-RFC2096-update-00

```
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv4.32.0.0.0.0.32.1.32.0.0.0.0 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.255.255.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.0.10.128.2.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.10 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138.128.2.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93.128.2.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.52.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.52.2.16.75.255.254.205.226.153.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.53.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.55.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.55.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.56.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.56.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.64.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.64.2.1.2.255.254.227.96.137.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.65.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.65.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.32.1.6.96.3.1.0.66.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 3
```

```
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.254.128.0.1.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138.128.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.254.128.0.2.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.1 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.255.1.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.255.2.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.255.2.0.1.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.255.2.0.2.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteIfIndex.1.ipv6.128.255.2.0.3.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteType.1.ipv4.32.0.0.0.0.32.1.32.0.0.0.0 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.255.255.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.10.128.2.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.10 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138.128.2.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93.128.2.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.52.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.52.2.16.75.255.254.205.226.153.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.53.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.55.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.55.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.56.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.56.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.64.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.64.2.1.2.255.254.227.96.137.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.65.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.65.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.32.1.6.96.3.1.0.66.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.254.128.0.1.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138.128.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.254.128.0.2.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.1 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.255.1.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.255.2.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: remote(4)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.255.2.0.1.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.255.2.0.2.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteType.1.ipv6.128.255.2.0.3.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: local(3)
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv4.32.0.0.0.0.32.1.32.0.0.0.0 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.255.255.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.10.128.2.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.10 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138.128.2.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93.128.2.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.52.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.52.2.16.75.255.254.205.226.153.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.53.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.55.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.55.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.56.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.56.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.64.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.64.2.1.2.255.254.227.96.137.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.65.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.65.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.32.1.6.96.3.1.0.66.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.254.128.0.1.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138.128.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.254.128.0.2.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.1 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.255.1.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.255.2.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.255.2.0.1.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.255.2.0.2.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteProto.1.ipv6.128.255.2.0.3.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv4.32.0.0.0.0.32.1.32.0.0.0.0 = Gauge32: 0
```

```
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.255.255.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.0.0.10.128.2.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.10 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138.128.2.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93.128.2.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.52.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.52.2.16.75.255.254.205.226.153.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.53.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.55.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.55.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.56.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.56.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.64.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.64.2.1.2.255.254.227.96.137.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.65.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.65.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.32.1.6.96.3.1.0.66.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.254.128.0.1.0.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138.128.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.254.128.0.2.0.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.1 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.254.128.0.3.0.0.0.0.0.0.0.0.1 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.255.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.255.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.255.2.0.1.0.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.255.2.0.2.0.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteNextHopAS.1.ipv6.128.255.2.0.3.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = Gauge32: 0
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv4.32.0.0.0.0.32.1.32.0.0.0.0 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.255.255.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.0.0.10.128.2.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.10 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138.128.2.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93.128.2.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.52.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.52.2.16.75.255.254.205.226.153.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.53.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.55.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.55.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.56.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 4
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.56.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 4
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.64.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.64.2.1.2.255.254.227.96.137.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 3
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.65.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.65.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.32.1.6.96.3.1.0.66.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: 2
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.254.128.0.1.0.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138.128.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.254.128.0.2.0.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.254.128.0.3.0.0.0.0.0.0.0.0.1 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.255.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.255.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.255.2.0.1.0.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.255.2.0.2.0.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric1.1.ipv6.128.255.2.0.3.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: 1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv4.32.0.0.0.0.32.1.32.0.0.0.0 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.255.255.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.0.0.10.128.2.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.10 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138.128.2.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93.128.2.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.52.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
```

```
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.52.2.16.75.255.254.205.226.153.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.53.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.55.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.55.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.56.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.56.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.64.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.64.2.1.2.255.254.227.96.137.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.65.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.65.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.32.1.6.96.3.1.0.66.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.254.128.0.1.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138.128.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.254.128.0.2.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.255.1.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.255.2.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.255.2.0.1.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.255.2.0.2.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric2.1.ipv6.128.255.2.0.3.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv4.32.0.0.0.0.32.1.32.0.0.0.0 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.255.255.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.0.10.128.2.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.10 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138.128.2.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93.128.2.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.52.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.52.2.16.75.255.254.205.226.153.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.53.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.55.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.55.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.56.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.56.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.64.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.64.2.1.2.255.254.227.96.137.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.65.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.65.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.32.1.6.96.3.1.0.66.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.254.128.0.1.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138.128.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.254.128.0.2.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.255.1.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.255.2.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.255.2.0.1.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.255.2.0.2.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric3.1.ipv6.128.255.2.0.3.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv4.32.0.0.0.0.32.1.32.0.0.0.0 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.255.255.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.0.10.128.2.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.10 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.0 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138.128.2.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.0 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93.128.2.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.52.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.52.2.16.75.255.254.205.226.153.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.53.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.55.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.55.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.56.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.56.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.64.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.64.2.1.2.255.254.227.96.137.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.65.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.65.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.32.1.6.96.3.1.0.66.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
```

```
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.254.128.0.1.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138.128.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.254.128.0.2.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.1.128.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.255.1.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.255.2.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.255.2.0.1.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.255.2.0.2.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric4.1.ipv6.128.255.2.0.3.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv4.32.0.0.0.0.32.1.32.0.0.0.0 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.128.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.0.0.0.0.0.0.0.0.0.255.255.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.0.0.0.0.0.0.0.10.128.2.128.32.1.6.96.3.1.0.0.0.0.0.0.0.10 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.50.0.0.0.0.0.0.0 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138.128.2.128.32.1.6.96.3.1.0.50.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0.128.2.128.32.1.6.96.3.1.0.51.0.0.0.0.0.0.0 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93.128.2.128.32.1.6.96.3.1.0.51.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.52.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.52.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.52.2.16.75.255.254.205.226.153.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.53.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.55.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.55.2.1.2.255.254.227.96.25.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.56.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.56.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.64.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.64.2.1.2.255.254.227.96.137.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.65.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.65.2.1.2.255.254.227.96.18.128.2.128.254.128.0.2.0.0.0.0.2.5.50.255.254.100.39.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.32.1.6.96.3.1.0.66.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.95.204 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.254.128.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.254.128.0.1.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138.128.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.254.128.0.2.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93.128.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.254.128.0.3.0.0.0.0.0.0.0.0.0.0.1.128.2.128.254.128.0.3.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.255.1.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.255.2.0.0.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.255.2.0.1.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.1.0.0.0.0.2.1.2.255.254.227.96.138 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.255.2.0.2.0.0.0.0.0.0.0.0.0.0.2.128.254.128.0.2.0.0.0.0.2.1.2.255.254.227.96.93 = INTEGER: -1
IP-FORWARD-MIB::inetCidrRouteMetric5.1.ipv6.128.255.2.0.3.0.0.0.0.0.0.0.0.0.0.2.128.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1 = INTEGER: -1
```

Contents

References

[Bak97] F. Baker. Ip forwarding table. Technical report, IETF, January 1997. RFC 2096.

[CD98] A. Conta and S. Deering. Internet control message protocol (icmpv6) for the internet protocol version 6 (ipv6) specification. Technical report, IETF, December 1998. RFC 2463.

[CFSD90] J.D. Case, M. Fedor, M.L. Schoffstall, and C. Davin. Simple network management protocol (snmp). Technical report, IETF, May 1990. RFC 1157.

[CMRW96a] J. Case, K. McCloghrie, M.T. Rose, and S. Waldbusser. Structure of management information for version 2 of the simple network management protocol (snmpv2). Technical report, IETF, January 1996. RFC 1902.

[CMRW96b] J. Case, K. McCloghrie, M.T. Rose, and S. Waldbusser. Textual conventions for version 2 of the simple network management protocol (snmpv2). Technical report, IETF, January 1996. RFC 1903.

[Con01] A. Conta. Extensions to ipv6 neighbor discovery for inverse discovery specification. Technical report, IETF, June 2001. RFC 3122.

[Cra02] M. Crawford. "ipv6 node information queries, draft-ietf-ipngwg-name-lookups-09.txt". Technical report, IETF, 2002.

[DH98] S. Deering and R. Hinden. Internet protocol, version 6 (ipv6) specifications. Technical report, IETF, December 1998. RFC 2460.

[DHRS00] M. Daniele, B. Haberman, S. Routhier, and J Schoenwaelder. Textual conventions for internet network addresses. Technical report, IETF, June 2000. RFC 2851.

[DHRS02] M. Daniele, B. Haberman, S. Routhier, and J Schoenwaelder. Textual conventions for internet network addresses. Technical report, IETF, 2002. RFC 3291.

[FLYV93] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless inter-domain routing (cidr): an address assignment and aggregation strategy. Technical report, IETF, September 1993. RFC 1519.

[HD98] R. Hinden and S. Deering. Ip version 6 addressing architecture. Technical report, IETF, July 1998. RFC 2373.

[HO98] D. Haskin and S. Onishi. Management information base for ip version 6 : Textuel conventions and general group. Technical report, IETF, December 1998. RFC 2465.

[McC96a] K. McCloghrie. Management information base for the internet protocol (ip). Technical report, IETF, November 1996. RFC 2011.

[McC96b] K. McCloghrie. Management information base for the transmission control protocol (tcp). Technical report, IETF, November 1996. RFC 2012.

[McC96c] K. McCloghrie. Management information base for the user datagram protocol (udp). Technical report, IETF, November 1996. RFC 2013.

[MPS99] K. McCloghrie, D. Perkins, and J. Schoenwalder. Structure of management information v2 (smiv2). Technical report, IETF, April 1999. RFC 2578, STD 58.

[MR91] K. McCloghrie and M.T. Rose. Management information base for the network management of tcp/ipbased internets - mib ii. Technical report, IETF, March 1991. RFC 1213, STD 17.

[NNS98] T. Narten, E. Nordmark, and W. Simpson. Neighbor discovery for ip version 6 (ipv6). Technical report, IETF, December 1998. RFC 2461.

[RM91] M.T. Rose and K. McCloghrie. Concise mib definitions. Technical report, IETF, March 1991. RFC 1212, STD 16.

[Ste98] Richard W. Stevens. *"UNIX Network programming - Networking APIs : Sockets and XTI"*, volume 2. Prentice Hall, 1998.

[VCF+02] R. Vida, L. Costa, S. Fdida, S. Deering, B. Fenner, I. Kouvelas, and B. haberman. "multicast listener discovery version 2 (mldv2) for ipv6, draft-vida-mld-v2-05.txt". Technical report, IETF, 2002.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803