                    Simple Network Management Protocol
                     Distributed Protocol Interface
                            Version 2.0

Status of this Memo

   This memo defines an Experimental Protocol for the Internet
   community. This memo does not specify an Internet standard of any
   kind. Discussion and suggestions for improvement are requested.
   Distribution of this memo is unlimited.

Table of Contents

1. INTRODUCTION

    This RFC describes version 2.0 of a protocol that International
    Business Machines Corporation (IBM) has been implementing in most of
    its SNMP agents to allow dynamic extension of supported MIBs. Bell
    Northern Research (BNR) has also implemented a version of this
    protocol in some of its SNMP agents for the same reason.

    The Simple Network Management Protocol (SNMP [1]) Distributed
    Protocol Interface (DPI) is an extension to SNMP agents that permits
    end-users to dynamically add, delete or replace management variables
    in the local Management Information Base without requiring
    recompilation of the SNMP agent. This is achieved by writing a so-
    called sub-agent that communicates with the agent via the SNMP-DPI.

    For the author of a sub-agent, the SNMP-DPI eliminates the need to
    know the details of ASN.1 [2] or SNMP PDU (Protocol Data Unit)
    encoding/decoding [1, 3].

    Versions 1.0 and 1.1 of this protocol have been in use within IBM

Wijnen, Carpenter, Curran, Sehgal & Waters   [Page 2]

since 1989 and is included in the SNMP agents for VM, MVS and OS/2.
Version 1.2 of this protocol has been in use within BNR since 1992.

1.1 MOTIVATION

The Simple Network Management Protocol [1] defines a protocol that
permits operations on a collection of variables. This set of
variables is called the Management Information Base (MIB) and a core
set of variables has previously been defined [4, 5]; however, the
design of the MIB makes provision for extension of this core set.
Thus, an enterprise or individual can define variables of their own
which represent information of use to them. An example of a
potentially interesting variable which is not in the core MIB would
be CPU utilization (percent busy). Unfortunately, conventional SNMP
agent implementations provide no means for an end-user to make
available new variables.

Besides this, today there are many MIBs that people want to implement
on a system. Without a capability for sub-agents, this requires all
the MIBs to be implemented in one big monolithic agent, which is in
many cases undesirable.

The SNMP DPI addresses these issues by providing a light-weight
mechanism by which a process can register the existence of a MIB
variable or a MIB sub-tree with the SNMP agent. Requests for the
variable(s) that are received by the SNMP agent are passed to the
process acting as a sub-agent. The sub-agent then returns an
appropriate answer to the SNMP agent. The SNMP agent eventually
packages an SNMP response packet and sends the answer back to the
remote network management station that initiated the request.

Remote network management stations have no knowledge that the SNMP
agent calls on other processes to obtain an answer. As far as they
can tell, there is only one network management application (agent)
running on the host.

At the San Diego IETF (March 1992) a BOF was held on multiplexing
SNMP agent's requirements. Both the SMUX [6] and DPI [7] protocols
were discussed, as well as other unpublished approaches. There was
also discussion regarding a need for a standard for multiplexing SNMP
agents or sub-agent support. At the end of the BOF, however, there
was not enough support for defining a standard. This was due, at
least partially, to a few well known SNMP authors who stated that the
proxy and party support for SNMPv2 (SMP at the time) would solve the
problem.

Nevertheless, questions continue to be raised about sub-agent support
(both in SNMP and SNMP2 mail lists) in spite of both SNMPv2 [8] being
on the standard's track and SMUX being changed to a historic RFC.
Furthermore, within IBM and BNR we continue to see a substantial and
expanding use of the DPI protocol. with positive results.

Therefore, we believe that there is a place for a sub-agent protocol
and we again offer this new version as an experimental protocol. We
encourage people to try it and send us feedback. Depending on that
feedback, we may decide to try to get onto the standards track at a
later time.

During discussions about sub-agent interfaces at the San Diego BOF it
also became clear that we should reduce the focus on the API for the
sub-agent programmers. This RFC, therefore, specifies only the
protocol to distribute SNMP requests from the main SNMP agent to the
sub-agents. Programmers can build one or more Programming APIs on
top of that protocol as needed, and sample API code is available from
the authors of this document.

1.2 SUMMARY OF CHANGES

The following changes have been made since the initial definition of
SNMP-DPI [7]. Some of these resulted from comparing the SMUX [6] and
DPI [7] protocols.

   o   Documentation changes to cleanup and be more specific in some
       areas. Among other things, this includes:

       -   Defining that integers are in network byte order
       -   Defining the character set used for strings
       -   Defining how DisplayStrings are handled.
       -   Including DPI20 MIB definition.

   o   Removal of the Programming API from the document.

   o   Addition of new DPI packet types:

       -   SNMP_DPI_OPEN for a sub-agent to open a "connection" with
           the DPI SNMP capable agent. The sub-agent must now
           identify itself and optionally provide a "password" for the
           connection.
       -   SNMP_DPI_CLOSE for the agent or sub-agent to close the
           connection in a graceful way.
       -   SNMP_DPI_ARE_YOU_THERE for the sub-agent to verify that the
           agent still knows about the sub-agent.
       -   SNMP_DPI_UNREGISTER for the agent or sub-agent to terminate
           the registration of a MIB variable or MIB sub-tree.

- SNMP_DPI_COMMIT which instructs the sub-agent to actually commit a previous SNMP_DPI_SET request. This, together with the UNDO, allows DPI sub-agents to be compliant with SNMP in the sense that we can now handle the "as if simultaneous" requirement.
- SNMP_DPI_UNDO which instructs the sub-agent to UNDO a SET or COMMIT if such is needed.

o   Changes to DPI packets:

- Multiple varBinds can now be exchanged in one DPI packet (for GET, GETNEXT, SET, TRAP). The sub-agent can specify the maximum it wants to handle per packet.
- The packet headers now contain a packet-ID (similar to SNMP request ID in SNMP PDU). This allows to match RESPONSE packets to REQUESTS, which is important for UDP based DPI-connections.
- The SNMP_DPI_REGISTER packet has new fields for time_out and for requested priority.
- The SNMP_DPI_TRAP packet allows to specify an enterprise OID. In addition, the generic and specific trap types are now 4 octets, so that we can pass the types correctly.
- In general, the packets have a more consistent layout.

o   The agent now sends a RESPONSE to a REGISTER request

o   Addition of SNMPv2 error codes and value types.

2. THEORY OF OPERATION

2.1 CONNECTION ESTABLISHMENT AND TERMINATION

   Communication between the SNMP Agent and its clients (sub-agents)
   takes place via a communication mechanism. The communication type
   can be either a logical stream connection (via TCP, for instance) or
   an unreliable datagram connection (UDP, for instance). It should be
   noted that other stream oriented transport communication mechanisms
   can also be used. For example, the VM SNMP agent allows DPI
   connections over IUCV (Inter-User Communications Vehicle) [9, 10].
   Other than the connection establishment procedure, the protocol used
   is identical in these environments.

   In Unix the number of processes is limited by the number of file-
   descriptors that can be opened. Since each TCP socket represents a
   file-descriptor, restricting SNMP-DPI protocol to TCP only
   connections would limit the number of sub-agents an agent could
   support. As a result, the some SNMP-DPI agents support both TCP and
   UDP socket type communication mechanisms for the SNMP-DPI protocol.

Please note that in the following portion of this text the SNMP-DPI
agent is referred simply as the agent.

Once the transport connection has been set up, the sub-agent must
also initialize the logical connection with the agent. To do so it
issues an OPEN request to the agent in which the sub-agent uniquely
identifies itself and passes some other parameters to the agent, such
as, the maximum number of varBinds per interaction it is prepared to
handle, and the timeout the agent should use when waiting for a
response from the sub-agent.

When the sub-agent prepares to stop or cease operations, it first
issues a CLOSE to shut down the logical connection with the agent,
and then closes the transport connection.

2.2 REGISTRATION

A sub-agent supports a collection of MIB variables or object
identifiers (object IDs) that constitute its MIB (sub)tree. Each of
these object IDs consists of a group ID and an instance ID. The
group ID is the root of the sub-agent's MIB tree that it supports and
the point of registration to the agent's MIB tree. The instance ID
is the piece of the Object Identifier that follows the group ID
(registration point), so it is not an instance in the terms of the
SNMP definition of an instance.

Regardless of the transport mechanism used, after establishing a
connection to the agent, the sub-agent registers a branch (group ID)
to the Agent's MIB tree. With the registration request, the sub-
agent passes some parameters, such as, requested priority and a
timeout value for this specific sub-tree.

The agent sends back a response to indicate success or failure of the
registration request.

2.3 NORMAL OPERATION

Once the sub-agent has set up both the physical and logical
connection to the agent, and once it has successfully registered the
sub-tree(s) of the MIB(s) that it supports, it waits for requests
from the SNMP agent or generates traps as required.

2.4 DPI ARCHITECTURE

These are the requests that can be initiated by the SNMP agent:

     GET, GETNEXT, GETBULK, SET, COMMIT, UNDO, UNREGISTER, and CLOSE.

The first four of these correspond directly to SNMP requests that a
network management station can make (By default a GETBULK request
will be translated into multiple GETNEXT requests by the agent, but a
sub-agent may request that the GETBULK be passed to it). The COMMIT,
UNDO, UNREGISTER, ARE_YOU_THERE and CLOSE requests are specific
SNMP-DPI requests. The sub-agent normally responds to a request with
a RESPONSE packet. The CLOSE request is an exception for which the
sub-agent only closes the physical connection.

These are the requests that can be initiated by a sub-agent:

     OPEN, REGISTER, TRAP, UNREGISTER, ARE_YOU_THERE and CLOSE.

The agent responds to OPEN, REGISTER, UNREGISTER and ARE_YOU_THERE
with a RESPONSE packet. The TRAP packet is just accepted and
forwarded by the agent without returning any information to the sub-
agent. The CLOSE packet is also just accepted by the agent upon
which it closes the physical connection.

See Figure 1 for an overview of the DPI packet flow.
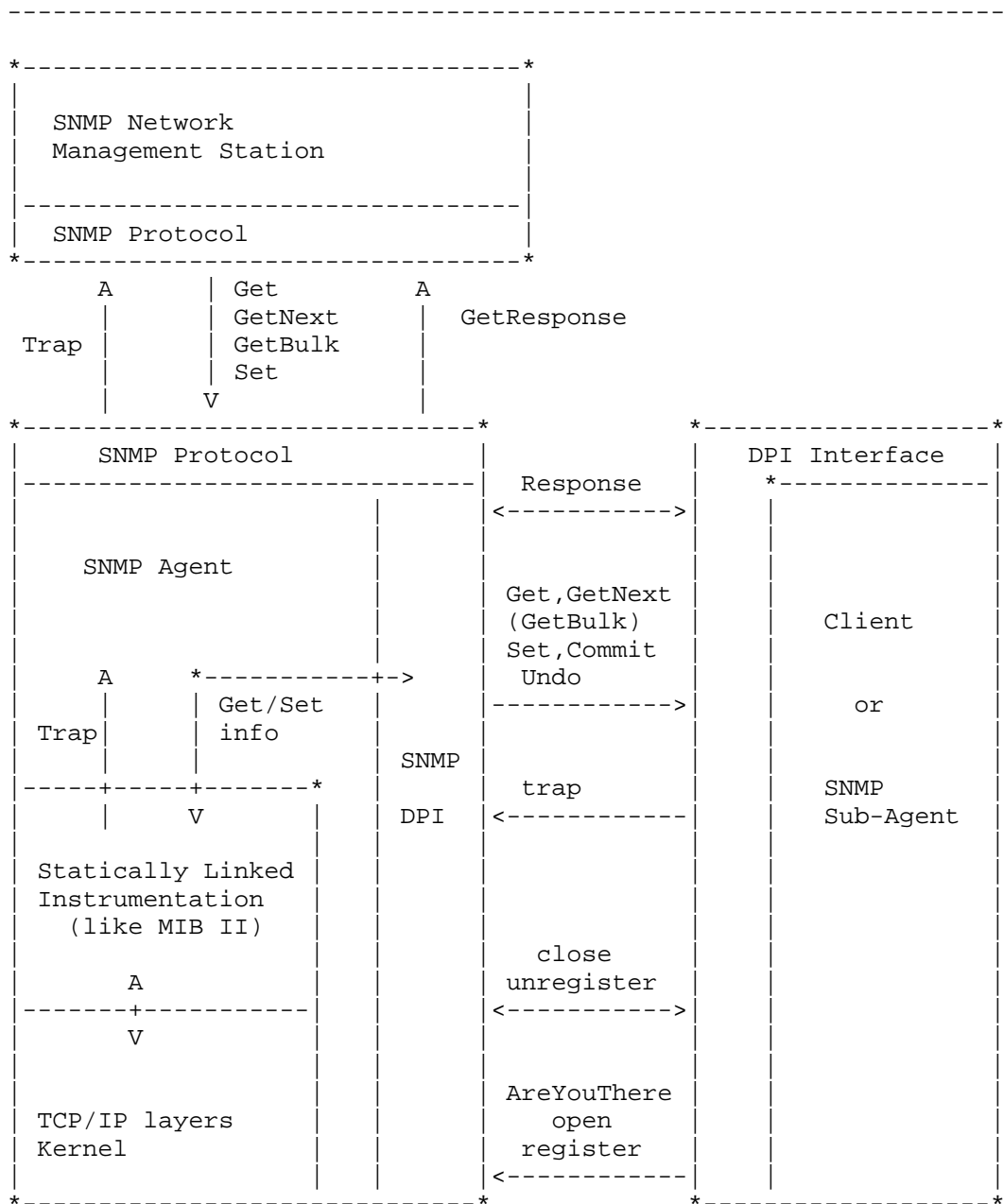
```
        -------------------------------------------------------------------

        *-------------------------------*       |
        |                               |       |
        |   SNMP Network                |       |
        |  Management Station           |       |
        |                               |       |
        |-------------------------------|       |
        |   SNMP Protocol               |       |
        *-------------------------------*       |
             A       | Get           A       |
             |       | GetNext       |    GetResponse
        Trap |       | GetBulk       |       |
             |       | Set           |       |
             |       V               |       |
        *------------------------------*       *-------------------*
        |     SNMP Protocol            |       |  DPI Interface    |
        |------------------------------|  Response  |  *-------------|
        |                       |      | |<---------->|  |         |
        |                       |      | |            |  |         |
        |    SNMP Agent         |      | |            |  |         |
        |                       |      | | Get,GetNext|  |         |
        |                       |      | | (GetBulk)  |  |  Client |
        |                       |      | | Set,Commit |  |         |
        |    A      *-----------+->    | |  Undo      |  |         |
        |    |      | Get/Set   |      | |----------->|  |   or    |
        | Trap|     | info      |      | |            |  |         |
        |    |      |           |   SNMP|            |  |         |
        |-----+-----+-------*   |      | | trap       |  |  SNMP   |
        |    |      V    |      | DPI  |<-----------|  | Sub-Agent|
        |    |           |      |      |            |  |         |
        | Statically Linked |  |      |            |  |         |
        | Instrumentation   |  |      |            |  |         |
        |   (like MIB II)   |  |      |            |  |         |
        |                   |  |      | close      |  |         |
        |    A              |  |      | unregister |  |         |
        |-------+-----------|  |      |<---------->|  |         |
        |       V           |  |      |            |  |         |
        |                   |  |      |            |  |         |
        | TCP/IP layers     |  |      | AreYouThere|  |         |
        | Kernel            |  |      |  open      |  |         |
        |                   |  |      |  register  |  |         |
        |                   |  |      |<-----------|  |         |
        *------------------------------*       *-------------------*

        -------------------------------------------------------------------
```
Figure 1. SNMP DPI overview

Remarks for Figure 1:

o   The SNMP agent communicates with the SNMP manager via the
     standard SNMP protocol.
o   The SNMP agent communicates with some statically linked-in
     instrumentation (potentially for the MIB II), which in turn
     talks to the TCP/IP layers and kernel (operating system) in an
     implementation-dependent manner.
o   An SNMP sub-agent, running as a separate process (potentially
     on another machine), can set up a connection with the agent.
     The sub-agent has an option to communicate with the SNMP agent
     through UDP or TCP sockets, or even through other mechanisms.
o   Once the connection is established, the sub-agent issues a DPI
     OPEN and one or more REGISTER requests to register one or more
     MIB sub-trees with the SNMP agent.
o   The SNMP agent responds to DPI OPEN and REGISTER requests with
     a RESPONSE packet, indicating success or failure.
o   The SNMP agent will decode SNMP packets.
     If such a packet contains a Get or GetNext request for an
     object in a sub-tree registered by a sub-agent, it sends a
     corresponding DPI packet to the sub-agent.
     If the request is for a GetBulk, then the agent translates it
     into multiple DPI GETNEXT packets and sends those to the
     sub-agent. However, the sub-agent can request (in the REGISTER
     packet) that a GETBULK be passed to the sub-agent.
     If the request is for a Set, then the agent uses a 2-phase
     commit scheme and sends the sub-agent a sequence of SET/COMMIT,
     SET/UNDO or SET/COMMIT/UNDO DPI packets.
o   The SNMP sub-agent sends responses back via a RESPONSE packet.
o   The SNMP agent then encodes the reply into an SNMP packet and
     sends it back to the requesting SNMP manager.
o   If the sub-agent wants to report an important state change, it
     sends a DPI TRAP packet to the SNMP agent which will encode it
     into an SNMP trap packet and send it to the manager(s).
o   If the sub-agent wants to stop operations, it sends a DPI
     UNREGISTER and a DPI CLOSE packet to the agent. The agent
     sends a response to an UNREGISTER request.
o   There is no RESPONSE to a CLOSE, the agent just closes the DPI
     connection. A CLOSE implies an UNREGISTER for all
     registrations that exist for the DPI connection being CLOSED.
o   An agent can send DPI UNREGISTER (if a higher priority
     registration comes in or for other reasons) to the sub-agent,
     the sub-agent then responds with a DPI RESPONSE packet.
o   An agent can also (for whatever reason) send a DPI CLOSE to
     indicate it is terminating the DPI connection.
o   A sub-agent can send an ARE_YOU_THERE to verify that the
     "connection" is still open. If so, the agent sends a RESPONSE
     with no error, otherwise, it may send a RESPONSE with an error

          indication, or not react at all.

3. SNMP DPI PROTOCOL

   This section describes the actual protocol used between the SNMP
   agent and sub-agents.

3.1 CONNECTION ESTABLISHMENT

   In a TCP/IP environment, the SNMP agent listens on an arbitrary
   TCP/UDP port for a connection request from a sub-agent. It is
   important to realize that a well-known port is not used: every
   invocation of the SNMP agent will potentially result in a different
   TCP/UDP port being used.

   A sub-agent needs to determine this port number to establish a
   connection. The sub-agent learns the port number from the agent by
   sending it one conventional SNMP get-request PDU. The port numbers
   are maintained by the SNMP agent as the objects whose identifiers
   are:

        1.3.6.1.4.1.2.2.1.1.0   dpiPort.0   (old DPI 1.x form)
        1.3.6.1.4.1.2.2.1.1.1.0 dpiPortForTCP.0
        1.3.6.1.4.1.2.2.1.1.2.0 dpiPortForUDP.0

   These variables are registered under the IBM enterprise-specific
   tree. See 4, "DPI 2.0 MIB definition" for more information. The
   SNMP agent replies with a conventional SNMP response PDU that
   contains the port number to be used. This response is examined by
   the sub-agent and the port number is extracted. The sub-agent then
   establishes the connection to the specified port.

   On the surface, this procedure appears to mean that the sub-agent
   must be able to create and parse SNMP packets, but this is not the
   case. A DPI Application Programming Interface (API) normally
   provides a library routine, query_DPI_port(), which can be used to
   generate and parse the required SNMP packets. This very small
   routine (under 100 lines of C), does not greatly increase the size of
   any sub-agent.

   NOTE: Since this RFC does not define an API, the actual code of and
   interface to a query_DPI_port() type of function depends on the
   implementation.

   For completeness, byte-by-byte descriptions of the packets to be
   generated by an SNMP DPI API routine query_DPI_port() are provided
   below. This is probably of little interest to most readers and
   reading the source of a query_DPI_port() function provides much of

the same information.

3.1.1 SNMP PDU TO GET THE AGENT'S DPI PORT

As noted, before a TCP/UDP connection to the SNMP agent can be made,
the sub-agent must learn which port that the agent is listening on.
To do so, it can issue an SNMP GET for the variable dpiPortForTCP.0
(1.3.6.1.4.1.2.2.1.1.1.0) or variable dpiPortForUDP.0
(1.3.6.1.4.1.2.2.1.1.2.0).

The SNMP PDU can be constructed as shown below. This PDU must be
sent to UDP port 161 on the host where the agent runs (probably the
same host where the sub-agent runs).

The (SNMPv1) packet shown below is for the TCP port.

```
+-----------------------------------------------------------------+
| Table 1 (Page 1 of 2). SNMP GET PDU for dpiPortForTCP.0         |
+--------------+---------------+----------------------------------+
| OFFSET       | VALUE         | FIELD                            |
+--------------+---------------+----------------------------------+
| 0            | 0x30          | ASN.1 header                     |
+--------------+---------------+----------------------------------+
| 1            | 37 + len      | PDU_length, see formula below    |
+--------------+---------------+----------------------------------+
| 2            | 0x02 0x01 0x00| SNMP version:                    |
|              |               | (integer,length=1,value=0)       |
+--------------+---------------+----------------------------------+
| 5            | 0x04          | community name (string)          |
+--------------+---------------+----------------------------------+
| 6            | len           | length of community name         |
+--------------+---------------+----------------------------------+
| 7            | community name| varies                           |
+--------------+---------------+----------------------------------+
| 7 + len      | 0xa0 0x1c     | SNMP GET request:                |
|              |               | request_type=0xa0,length=0x1c    |
+--------------+---------------+----------------------------------+
| 7 + len + 2  | 0x02 0x01 0x01| SNMP request ID:                 |
|              |               | integer,length=1,ID=1            |
+--------------+---------------+----------------------------------+
| 7 + len + 5  | 0x02 0x01 0x00| SNMP error status:               |
|              |               | integer,length=1,error=0         |
+--------------+---------------+----------------------------------+
| 7 + len + 8  | 0x02 0x01 0x00| SNMP index:                      |
|              |               | integer,length=1,index=0         |
+--------------+---------------+----------------------------------+
| 7 + len + 11 | 0x30 0x11     | varBind list, length=0x11        |
+--------------+---------------+----------------------------------+
| 7 + len + 13 | 0x30 0x0f     | varBind, length=0x0f             |
+--------------+---------------+----------------------------------+
| 7 + len + 15 | 0x06 0x0b     | Object ID, length=0x0b           |
+--------------+---------------+----------------------------------+
```

```
+---------------------------------------------------------------+
| Table 1 (Page 2 of 2). SNMP GET PDU for dpiPortForTCP.0       |
+--------------+---------------+--------------------------------+
| OFFSET       | VALUE         | FIELD                          |
+--------------+---------------+--------------------------------+
| 7 + len + 17 | 0x2b 0x06 0x01| Object-ID:                     |
|              | 0x04 0x01 0x02| 1.3.6.1.4.1.2.2.1.1.1          |
|              | 0x02 0x01 0x01| Object-instance: 0             |
|              | 0x01 0x00     |                                |
+--------------+---------------+--------------------------------+
| 7 + len + 28 | 0x05 0x00     | null value, length=0           |
+--------------+---------------+--------------------------------+
| NOTE:  Formula to calculate "PDU_length":                     |
|                                                               |
|    PDU_length =  length of version field and string tag (4 bytes)|
|              +  length of community length field (1 byte)     |
|              +  length of community name (depends...)         |
|              +  length of SNMP GET request (32 bytes)         |
|                                                               |
|              =  37 + length of community name                 |
+---------------------------------------------------------------+
```

3.1.2 SNMP PDU CONTAINING THE RESPONSE TO THE GET

   Assuming that no errors occurred, the port is returned in the last
   few octets of the received packet. In the simple case, where the
   port number will be between 1024 and 16,385, the format of the packet
   is shown below.

   Note: In practice, the port number can be any positive number in the
   range from 1 through 65,535. A port number of 0 means that the agent
   does not have a dpiPort defined for the requested protocol. So the
   actual port value maybe in the last 1, 2 or 3 octets. The sample
   implementation code shows how to handle the response to cover all
   those cases, including error conditions.

   Note: The (SNMPv1) packet shown below is for the TCP port.

```
+---------------------------------------------------------------+
| Table 2 (Page 1 of 3). SNMP RESPONSE PDU for dpiPortForTCP.0  |
+--------------+---------------+--------------------------------+
| OFFSET       | VALUE         | FIELD                          |
+--------------+---------------+--------------------------------+
| 0            | 0x30          | ASN.1 header                   |
+--------------+---------------+--------------------------------+
| 1            | 39 + len      | length, see formula below      |
+--------------+---------------+--------------------------------+
```

```
+----------------------------------------------------------------------+
| Table 2 (Page 2 of 3). SNMP RESPONSE PDU for dpiPortForTCP.0          |
+---------------+---------------+--------------------------------------+
| OFFSET        | VALUE         | FIELD                                |
+---------------+---------------+--------------------------------------+
| 2             | 0x02 0x01 0x00| version                              |
|               |               | (integer,length=1,value=0)           |
+---------------+---------------+--------------------------------------+
| 5             | 0x04          | community name (string)              |
+---------------+---------------+--------------------------------------+
| 6             | len           | length of community name             |
+---------------+---------------+--------------------------------------+
| 7             | community name|                                      |
+---------------+---------------+--------------------------------------+
| 7 + len       | 0xa2 0x1e     | SNMP RESPONSE:                       |
|               |               | request_type=0xa2,length=0x1e        |
+---------------+---------------+--------------------------------------+
| 7 + len + 2   | 0x02 0x01 0x01| SNMP request ID:                     |
|               |               | integer,length=1,ID=1                |
+---------------+---------------+--------------------------------------+
| 7 + len + 5   | 0x02 0x01 0x00| SNMP error status:                   |
|               |               | integer,length=1,error=0             |
+---------------+---------------+--------------------------------------+
| 7 + len + 8   | 0x02 0x01 0x00| SNMP index:                          |
|               |               | integer,length=1,index=0             |
+---------------+---------------+--------------------------------------+
| 7 + len + 11  | 0x30 0x13     | varBind list, length=0x13            |
+---------------+---------------+--------------------------------------+
| 7 + len + 13  | 0x30 0x11     | varBind, length=0x11                 |
+---------------+---------------+--------------------------------------+
| 7 + len + 15  | 0x06 0x0b     | Object ID, length=0x0b               |
+---------------+---------------+--------------------------------------+
| 7 + len + 17  | 0x2b 0x06 0x01| Object-ID:                           |
|               | 0x04 0x01 0x02| 1.3.6.1.4.1.2.2.1.1.1                |
|               | 0x02 0x01 0x01| Object-instance: 0                   |
|               | 0x01 0x00     |                                      |
+---------------+---------------+--------------------------------------+
| 7 + len + 28  | 0x02 0x02     | integer, length=2                    |
+---------------+---------------+--------------------------------------+
| 7 + len + 30  | MSB LSB       | port number (MSB, LSB)               |
+---------------+---------------+--------------------------------------+
```

```
+----------------------------------------------------------------+
| Table 2 (Page 3 of 3). SNMP RESPONSE PDU for dpiPortForTCP.0    |
+---------------+---------------+--------------------------------+
| NOTE:  Formula to calculate "PDU_length":                      |
|                                                                |
|    PDU_length =  length of version field and string tag (4 bytes)|
|               +  length of community length field (1 byte)     |
|               +  length of community name (depends...)         |
|               +  length of SNMP RESPONSE (34 bytes)            |
|                                                                |
|               =  39 + length of community name                 |
+----------------------------------------------------------------+
```

3.2 SNMP DPI PACKET FORMATS

   Each request to, or response from, the agent or sub-agent is
   constructed as a "packet" and is written to the stream.

   Each packet is prefaced with the length of the data remaining in the
   packet. The length is stored in network byte order, the most
   significant byte (MSB) first, least significant byte (LSB) last. If
   we consider a stream connection (like TCP), the receiving side will
   read the packet by doing something similar to:

```
        unsigned char len_bfr[2];
        unsigned char *bfr;
        int len;

        read(fd,len_bfr,2);
        len = len_bfr[0] * 256 + len_bfr[1];
        bfr = malloc(len);
        read(fd,bfr,len);
```

   Note: The above example makes no provisions for error handling or a
   read returning less than the requested amount of data,and it is not
   intended to be used literally.

3.2.1 DPI PACKET HEADER

   The first part of every packet identifies the application protocol
   being used as well as some version information. The protocol major
   version is intended to indicate, in broad terms, what version of the
   protocol is used. The protocol minor version is intended to identify
   major incompatible versions of the protocol. The protocol release is
   intended to indicate incremental modifications to the protocol. The
   constants that are valid for these fields are defined in Table 15.

The next field, present in all packets, is the packet ID. It
contains packet identification that can help an agent or sub-agent
match responses with request. This is useful with UDP connections
over which packets can be lost. The packet ID is a monotonically
increasing unsigned 16-bit integer which wraps at its maximum value.

The next field, present in all packets, is the packet type. It
indicates what kind of packet we're dealing with (OPEN, REGISTER,
GET, GETNEXT, GETBULK, SET, COMMIT, UNDO, TRAP, RESPONSE, UNREGISTER,
or CLOSE). The permitted values for this field are defined in Table
16.

```
+------------------------------------------------------------------+
| Table 3. SNMP DPI packet header.  Present in all packets.         |
+------------+-----------------------------------------------------+
| OFFSET     | FIELD                                               |
+------------+-----------------------------------------------------+
| 0          | packet length to follow (MSB to LSB)                |
+------------+-----------------------------------------------------+
| 2          | protocol major version                              |
+------------+-----------------------------------------------------+
| 3          | protocol minor version                              |
+------------+-----------------------------------------------------+
| 4          | protocol release                                    |
+------------+-----------------------------------------------------+
| 5          | packet id (MSB to LSB)                               |
+------------+-----------------------------------------------------+
| 7          | packet type                                         |
+------------+-----------------------------------------------------+
```

From this point onwards, the contents of the packet are defined by
the protocol being used. The remainder of this section describes:

   o   Layout of packets for the SNMP DPI protocol, version 2.0.

   o   Constants as defined with this version of the protocol.

3.2.2 OPEN

In order for a sub-agent to communicate with a DPI capable SNMP
agent, it must first send an SNMP DPI OPEN request to the agent to
setup the "connection" with that agent.

Such a packet contains the standard SNMP DPI header plus OPEN
specific data. This data consists of:

o   a timeout value (in seconds).
     This is a requested timeout value to be used for all requests
     for objects for which there is no timeout value specified for
     the sub-tree under which the object is registered. If you
     specify a zero timeout value, then the agent will use its own
     default timeout value. If you want a larger value than the
     default value, then you can specify it here. However, the agent
     may have a maximum value that you can never exceed. If you do
     ask for a larger timeout than that maximum, the agent will set
     it at the maximum it accepts.
o   the maximum number of varBinds per DPI packet that the
     sub-agent is prepared to handle.
o   Selected character set to be used for the representation of the
     OBJECT ID strings and DisplayStrings.
     The choices are the native character set (0) or the ASCII
     character set (1). See 3.3.5, "Character set selection"
     for more information in character set selection.
     An agent may choose to support only the native character set.
o   null terminated sub-agent ID, which is a unique ASN.1 OBJECT
     identifier, so in dotted ASN.1 notation. This string is
     represented in the selected character set.
o   null terminated sub-agent description, which is a DisplayString
     describing the sub-agent. This string is represented in the
     selected character set. This may be the null-string if there
     is no description.
o   optionally a password that the agent uses to validate the
     sub-agent. It depends on the agent implementation if a
     password is required. If no password is passed, the length
     must be specified as zero.

The sub-agent must expect a response indicating success or failure.
See Table 19 for the valid codes in a DPI RESPONSE to a DPI OPEN
request.

If the error_code in the RESPONSE is not SNMP_ERROR_DPI_noError, then
the agent closes the connection.

```
+------------------------------------------------------------------+
| Table 4. Layout SNMP DPI OPEN packet                             |
+-----------+------------------------------------------------------+
| OFFSET    | FIELD                                                |
+-----------+------------------------------------------------------+
| 0         | packet length to follow (MSB to LSB)                 |
+-----------+------------------------------------------------------+
| 2         | protocol major version                               |
+-----------+------------------------------------------------------+
| 3         | protocol minor version                               |
+-----------+------------------------------------------------------+
| 4         | protocol release                                     |
+-----------+------------------------------------------------------+
| 5         | packet id (MSB to LSB)                               |
+-----------+------------------------------------------------------+
| 7         | packet type = SNMP_DPI_OPEN                          |
+-----------+------------------------------------------------------+
| 8         | requested overall timeout (seconds, MSB to LSB)     |
+-----------+------------------------------------------------------+
| 10        | max varBinds per DPI packet (MSB to LSB)            |
+-----------+------------------------------------------------------+
| 12        | Selected character set (0=Native, 1=ASCII)          |
+-----------+------------------------------------------------------+
| 13        | null terminated sub-agent ID (OID)                  |
+-----------+------------------------------------------------------+
| 13+L1     | null terminated sub-agent Description                |
+-----------+------------------------------------------------------+
| 13+L2     | password length (zero if no password, MSB to LSB)   |
+-----------+------------------------------------------------------+
| 15+L2     | password (if any)                                   |
+-----------+------------------------------------------------------+
| NOTE:                                                            |
|                                                                  |
| o   L1 = strlen(sub-agent ID) + 1                                |
| o   L2 = L1 + strlen(sub-agent Description) + 1                  |
| o   OID and Description strings use selected character set       |
+------------------------------------------------------------------+
```

3.2.3 CLOSE

   In order for a sub-agent to close the "connection" with the DPI
   capable SNMP agent, it must send an SNMP DPI CLOSE request to the
   agent. The agent will not send a response, but closes the physical
   connection and implicitly unregisters any sub-trees related to the
   connection.

   An agent may also send to the sub-agent an SNMP DPI CLOSE packet that
   contains the standard SNMP DPI header plus CLOSE specific data. This

data consists of:

    o   a reason code for closing. See Table 21 for a list
        of valid reason codes.

    +--+
    | Table 5. Layout SNMP DPI CLOSE packet |
    +-----------+--+
    | OFFSET | FIELD |
    +-----------+--+
    | 0 | packet length to follow (MSB to LSB) |
    +-----------+--+
    | 2 | protocol major version |
    +-----------+--+
    | 3 | protocol minor version |
    +-----------+--+
    | 4 | protocol release |
    +-----------+--+
    | 5 | packet id (MSB to LSB) |
    +-----------+--+
    | 7 | packet type = SNMP_DPI_CLOSE |
    +-----------+--+
    | 8 | reason code (1 octet) |
    +-----------+--+

3.2.4 ARE_YOU_THERE

   An ARE_YOU_THERE packet allows a sub-agent to determine if it still
   has a DPI connection with the agent. This packet is necessary
   because a sub-agent passively awaits requests from an agent and
   normally will not detect problems with an agent connection in a
   timely manner. (In contrast, an agent becomes aware of any sub-agent
   connection problem in a timely manner because it sets a timeout when
   sending request).

   A sub-agent can send a SNMP DPI ARE_YOU_THERE packet to an agent
   which will then return a RESPONSE with a zero error code and a a zero
   error index if the connection is healthy. Otherwise, the agent may
   return a RESPONSE with an error indication. If the connection is
   broken, the sub-agent will see no response at all.

   An ARE_YOU_THERE packet contains the standard SNMP DPI header with no
   additional data.

```
+-------------------------------------------------------------------+
| Table 6. Layout SNMP DPI ARE_YOU_THERE packet                     |
+------------+------------------------------------------------------+
| OFFSET     | FIELD                                                |
+------------+------------------------------------------------------+
| 0          | packet length to follow (MSB to LSB)                 |
+------------+------------------------------------------------------+
| 2          | protocol major version                               |
+------------+------------------------------------------------------+
| 3          | protocol minor version                               |
+------------+------------------------------------------------------+
| 4          | protocol release                                     |
+------------+------------------------------------------------------+
| 5          | packet id (MSB to LSB)                                |
+------------+------------------------------------------------------+
| 7          | packet type = SNMP_DPI_ARE_YOU_THERE                 |
+------------+------------------------------------------------------+
```

3.2.5 REGISTER

   In order to register a branch in the MIB tree, an SNMP sub-agent
   sends an SNMP DPI REGISTER packet to the agent.

   Such a packet contains the standard SNMP DPI header plus REGISTER
   specific data. This data consists of:

   o   a requested priority.
        There are 2 special values, namely minus one (-1, requests best
        available priority) and zero (0, requests next better priority
        than the highest priority in use). Any other value requests a
        specific priority or the next best priority if already in use).
        The lower the number, the better the priority. An agent will
        send requests to only the one sub-agent that has registered
        with the best priority. The agent returns the actual priority
        assigned in the RESPONSE packet in the error_index field.
   o   a requested timeout.
        If a zero value is specified, then the agent uses the timeout
        value specified in the DPI OPEN request.
        If you want a shorter or longer timeout value for this specific
        sub-tree, then you specify it here. The agent has a maximum
        timeout it will allow in this field. The agent will use this
        value (or its maximum) to await a response to requests for this
        sub-tree.
   o   an indication as to whether the sub-agent wishes to handle MIB
        view selection (SNMPv1 community string authentication)
        in subsequent GET, GETNEXT or SET, COMMIT, UNDO requests. Not
        all DPI capable agents need to support this feature, but they
        must at least recognize this indication and give an appropriate

Wijnen, Carpenter, Curran, Sehgal & Waters   [Page 20]

          response if they do not support it.
     o   an indication as to whether the sub-agent wishes to handle the
          GETBULK itself. If not, then the agent will translate a
          GETBULK into multiple GETNEXT requests.
          Not all DPI capable agents need to support this feature. They
          may opt to always translate a GETBULK into multiple GETNEXT
          requests. In this case the agent will send the appropriate
          RESPONSE to indicate this.
     o   the group ID (sub-tree) to be registered (with trailing dot).
          The group ID is represented in the selected character set as
          specified in DPI OPEN packet.

   The agent will respond with an SNMP DPI RESPONSE packet indicating
   registration error or success. The packet ID of the response will be
   the same as that for the REGISTER request to which this is a
   response.

   The group ID will be the same as that specified in the REGISTER
   request. There will be no instance returned (e.g. NULL string for
   instance ID). The value will be an SNMP_TYPE_NULL value with a zero
   length.

```
+----------------------------------------------------------------+
| Table 7. Layout SNMP DPI REGISTER packet                       |
+------------+---------------------------------------------------+
| OFFSET     | FIELD                                             |
+------------+---------------------------------------------------+
| 0          | packet length to follow (MSB to LSB)              |
+------------+---------------------------------------------------+
| 2          | protocol major version                            |
+------------+---------------------------------------------------+
| 3          | protocol minor version                            |
+------------+---------------------------------------------------+
| 4          | protocol release                                  |
+------------+---------------------------------------------------+
| 5          | packet id (MSB to LSB)                             |
+------------+---------------------------------------------------+
| 7          | packet type = SNMP_DPI_REGISTER                   |
+------------+---------------------------------------------------+
| 8          | requested priority (MSB to LSB)                   |
+------------+---------------------------------------------------+
| 12         | timeout in seconds (MSB to LSB)                   |
+------------+---------------------------------------------------+
| 14         | view selection (0 = you (agent) do, 1 = I do)     |
+------------+---------------------------------------------------+
| 15         | getbulk selection (0=use GetNext, 1=use GetBulk)  |
+------------+---------------------------------------------------+
| 16         | null terminated group ID (with trailing dot)      |
+------------+---------------------------------------------------+
| NOTE:                                                          |
|                                                                |
| o   group ID string uses selected character set               |
+----------------------------------------------------------------+
```

3.2.6 UNREGISTER

   In order to unregister a branch in the MIB tree, an SNMP sub-agent
   sends an SNMP DPI UNREGISTER packet to the agent.

   Such a packet contains the standard SNMP DPI header plus UNREGISTER
   specific data: a null terminated string (represented in the selected
   character set) representing the group ID in ASN.1 dotted notation and
   an indication as to the reason for the unregister (see table 14).

   The agent will respond with an SNMP DPI RESPONSE packet indicating
   error or success. The packet ID of the response will be the same as
   that for the UNREGISTER request to which this is a response.

   The group ID will be the same as that specified in the UNREGISTER
   request. There will be no instance returned (e.g. NULL string for

instance ID). The value will be an SNMP_TYPE_NULL value with a zero
length.

```
+----------------------------------------------------------------+
| Table 8. Layout SNMP DPI UNREGISTER packet                     |
+------------+---------------------------------------------------+
| OFFSET     | FIELD                                             |
+------------+---------------------------------------------------+
| 0          | packet length to follow (MSB to LSB)              |
+------------+---------------------------------------------------+
| 2          | protocol major version                            |
+------------+---------------------------------------------------+
| 3          | protocol minor version                            |
+------------+---------------------------------------------------+
| 4          | protocol release                                  |
+------------+---------------------------------------------------+
| 5          | packet id (MSB to LSB)                             |
+------------+---------------------------------------------------+
| 7          | packet type = SNMP_DPI_UNREGISTER                 |
+------------+---------------------------------------------------+
| 8          | reason code                                       |
+------------+---------------------------------------------------+
| 9          | null terminated group ID (with trailing dot)      |
+------------+---------------------------------------------------+
| NOTE:                                                          |
|                                                                |
| o   group ID string uses selected character set               |
+----------------------------------------------------------------+
```

3.2.7 GET

   When the SNMP agent receives a PDU containing an SNMP GET request for
   a variable that resides in a sub-tree registered by a sub-agent, it
   passes an SNMP DPI GET packet to the sub-agent.

   Such a packet contains the standard SNMP DPI header plus GET specific
   data:

   o   the community name used in the SNMP PDU. The length is zero
        unless view handling was selected by the sub-agent. The length
        is also zero if the SNMP PDU was not in SNMPv1 format.
   o   per varBind two null terminated strings (in the selected
        character set) representing the group and instance ID in ASN.1
        dotted notation.

```
+----------------------------------------------------------------+
| Table 9. Layout SNMP DPI GET packet                            |
+-----------+----------------------------------------------------+
| OFFSET    | FIELD                                              |
+-----------+----------------------------------------------------+
| 0         | packet length to follow (MSB to LSB)               |
+-----------+----------------------------------------------------+
| 2         | protocol major version                             |
+-----------+----------------------------------------------------+
| 3         | protocol minor version                             |
+-----------+----------------------------------------------------+
| 4         | protocol release                                   |
+-----------+----------------------------------------------------+
| 5         | packet id (MSB to LSB)                             |
+-----------+----------------------------------------------------+
| 7         | packet type = SNMP_DPI_GET                         |
+-----------+----------------------------------------------------+
| 8         | community name length (MSB to LSB)                |
+-----------+----------------------------------------------------+
| 10        | community name (if any)                            |
+-----------+----------------------------------------------------+
| 10+L1     | null terminated group ID (with trailing dot)       |
+-----------+----------------------------------------------------+
| 10+L2     | null terminated instance ID (no trailing dot)      |
+-----------+----------------------------------------------------+
| 10+L3     | optionally more varBinds (group/instance ID pairs) |
+-----------+----------------------------------------------------+
| NOTE:                                                          |
|                                                                |
| o   L1 = length of community name                              |
| o   L2 = L1 +  strlen(group ID) + 1                            |
| o   L3 = L2 +  strlen(instance ID) + 1                         |
| o   group and instance ID strings use selected character set   |
+----------------------------------------------------------------+
```

3.2.8 GETNEXT

   When the SNMP agent receives a PDU containing an SNMP GETNEXT request
   for a variable for which a sub-agent may be authoritative, it passes
   an SNMP DPI GETNEXT packet to the sub-agent.

   Such a packet contains the standard SNMP DPI header plus GETNEXT
   specific data:

   o   the community name used in the SNMP PDU. The length is zero
        unless view handling was selected by the sub-agent. The length
        is also zero if the SNMP PDU was not in SNMPv1 format.
   o   per varBind two null terminated strings (in the selected

character set) representing the group and instance ID in ASN.1
dotted notation.

```
+---------------------------------------------------------------------+
| Table 10. Layout SNMP DPI GETNEXT packet                            |
+-----------+---------------------------------------------------------+
| OFFSET    | FIELD                                                   |
+-----------+---------------------------------------------------------+
| 0         | packet length to follow (MSB to LSB)                    |
+-----------+---------------------------------------------------------+
| 2         | protocol major version                                  |
+-----------+---------------------------------------------------------+
| 3         | protocol minor version                                  |
+-----------+---------------------------------------------------------+
| 4         | protocol release                                        |
+-----------+---------------------------------------------------------+
| 5         | packet id (MSB to LSB)                                   |
+-----------+---------------------------------------------------------+
| 7         | packet type = SNMP_DPI_GETNEXT                           |
+-----------+---------------------------------------------------------+
| 8         | community name length (MSB to LSB)                      |
+-----------+---------------------------------------------------------+
| 10        | community name                                          |
+-----------+---------------------------------------------------------+
| 10+L1     | null terminated group ID (with trailing dot)            |
+-----------+---------------------------------------------------------+
| 10+L2     | null terminated instance ID (no trailing dot)           |
+-----------+---------------------------------------------------------+
| 10+L3     | optionally more varBinds (group/instance ID pairs)      |
+-----------+---------------------------------------------------------+
| NOTE:                                                               |
|                                                                     |
| o   L1 = length of community name                                   |
| o   L2 = L1 +  strlen(group ID) + 1                                 |
| o   L3 = L2 +  strlen(instance ID) + 1                              |
| o   group and instance ID strings use selected character set        |
+---------------------------------------------------------------------+
```

3.2.9 GETBULK

   When the SNMP agent receives a PDU containing an SNMP GETBULK request
   that includes variables for which a sub-agent may be authoritative,
   it checks if the sub-agent wants to handle the GETBULK itself (as
   specified at registration time). If so, it sends an SNMP DPI GETBULK
   packet to the sub-agent.

   Such a packet contains the standard SNMP DPI header plus GETBULK
   specific data:

- o   non-repeaters
- o   max repetitions
- o   per varBind two null terminated strings (in the selected
        character set) representing the group and instance ID in ASN.1
        dotted notation.

```
+---------------------------------------------------------------------+
| Table 11. Layout SNMP DPI GETBULK packet                            |
+-----------+---------------------------------------------------------+
| OFFSET    | FIELD                                                   |
+-----------+---------------------------------------------------------+
| 0         | packet length to follow (MSB to LSB)                    |
+-----------+---------------------------------------------------------+
| 2         | protocol major version                                  |
+-----------+---------------------------------------------------------+
| 3         | protocol minor version                                  |
+-----------+---------------------------------------------------------+
| 4         | protocol release                                        |
+-----------+---------------------------------------------------------+
| 5         | packet id (MSB to LSB)                                   |
+-----------+---------------------------------------------------------+
| 7         | packet type = SNMP_DPI_GETBULK                           |
+-----------+---------------------------------------------------------+
| 8         | non-repeaters (4 octets, MSB to LSB)                    |
+-----------+---------------------------------------------------------+
| 12        | max-repetitions (4 octets, MSB to LSB)                  |
+-----------+---------------------------------------------------------+
| 16        | null terminated group ID (with trailing dot)            |
+-----------+---------------------------------------------------------+
| 16+L1     | null terminated instance ID (no trailing dot)           |
+-----------+---------------------------------------------------------+
| 16+L2     | optionally more varBinds (group/instance ID pairs)      |
+-----------+---------------------------------------------------------+
| NOTE:                                                               |
|                                                                     |
| o   L1 = strlen(group ID) + 1                                       |
| o   L2 = L1 +  strlen(instance ID) + 1                              |
| o   group and instance ID strings use selected character set        |
+---------------------------------------------------------------------+
```

3.2.10 SET, COMMIT AND UNDO

When the SNMP agent receives a PDU containing an SNMP SET request for
a variable that is in a sub-tree registered by a sub-agent, it passes
one of 3 sequences of SNMP DPI packets to the sub-agent:

       o   SET, COMMIT
            This is the normal sequence. The SET request is the first
            phase. The sub-agent must verify that the SET request is valid
            and that the resources needed are available. The COMMIT
            request comes next. The sub-agent must now effectuate the SET
            request.
       o   SET, UNDO
            If an SNMP packet has a SET request for multiple varBinds that
            reside in different sub-trees, then the agent first sends a SET
            to all sub-agents. If any sub-agent returns an error on the
            SET, then the agent sends UNDO to those sub-agents that
            returned no error on the SET, meaning the SET is being
            canceled.
       o   SET, COMMIT, UNDO
            In the very rare circumstance where all sub-agents have
            responded error-free to a SET and where one of them fails to
            perform the COMMIT, then the agent sends an UNDO to all
            involved sub-agents (also those who completed COMMIT).
            Sub-agents should try, to the best of their ability, to never
            let a commit fail and to undo an already committed set if asked
            to do so.

   Such packets contain the standard SNMP DPI header plus SET specific
   data:

     o   the community name used in the SNMP PDU. The length is zero
          unless view handling was selected by the sub-agent. The length
          is also zero if the SNMP PDU was not in SNMPv1 format.
     o   per varBind:

          -    two null terminated strings (in the selected character set)
               representing the group and instance ID in ASN.1 dotted
               notation.
          -    the type, value length and value to be set.

          The permitted types for the type field are defined in Table 17.

          See 3.3.4, "Value Representation" for information on how the
          value data is represented in the packet value field.

```
+----------------------------------------------------------------+
| Table 12. Layout SNMP DPI SET, COMMIT, UNDO packet             |
+------------+---------------------------------------------------+
| OFFSET     | FIELD                                             |
+------------+---------------------------------------------------+
| 0          | packet length to follow (MSB to LSB)              |
+------------+---------------------------------------------------+
| 2          | protocol major version                            |
+------------+---------------------------------------------------+
| 3          | protocol minor version                            |
+------------+---------------------------------------------------+
| 4          | protocol release                                  |
+------------+---------------------------------------------------+
| 5          | packet id (MSB to LSB)                             |
+------------+---------------------------------------------------+
| 7          | packet type = SNMP_DPI_SET/COMMIT/UNDO            |
+------------+---------------------------------------------------+
| 8          | community name length (MSB to LSB)                |
+------------+---------------------------------------------------+
| 10         | community name                                    |
+------------+---------------------------------------------------+
| 10+L1      | null terminated group ID (with trailing dot)      |
+------------+---------------------------------------------------+
| 10+L2      | null terminated instance ID (no trailing dot)     |
+------------+---------------------------------------------------+
| 10+L3      | SNMP Variable Type Value                           |
+------------+---------------------------------------------------+
| 10+L3+1    | Length of value (2 octets, MSB to LSB)            |
+------------+---------------------------------------------------+
| 10+L3+3    | Value                                             |
+------------+---------------------------------------------------+
| 10+L4      | optionally more varBinds (sequences of group ID,  |
|            | instance ID, Type, Length and Value)              |
+------------+---------------------------------------------------+
| NOTE:                                                          |
|                                                                |
| o   L1 = length of community name                              |
| o   L2 = L1 + strlen(group ID) + 1                             |
| o   L3 = L2 + strlen(instance ID) + 1                          |
| o   L4 = L3 + 3 + length of value                              |
| o   group and instance ID strings use selected character set   |
| o   OID and DisplayString values use selected character set    |
| o   Integer values are in network byte order                   |
+----------------------------------------------------------------+
```

3.2.11 RESPONSE

   An SNMP sub-agent must respond to a GET, GETNEXT, GETBULK, SET,
   COMMIT, UNDO or UNREGISTER request that it has received from the
   agent (unless it fails or has a bug ;-)). To do so, it sends an SNMP
   DPI RESPONSE packet to the agent.

   Such a packet contains the standard SNMP DPI header plus RESPONSE
   specific data:

   o   an error_code,
   o   an error_index,
   o   plus for a successful GET, GETNEXT, or GETBULK, the
        name/type/length/value tuple(s) representing the returned
        object(s). For each varBind this is described as:

        -    two null terminated strings (in the selected character set)
             representing the group and instance ID in ASN.1 dotted
             notation.
        -    the type, value length and value of the object that is
             returned.

        The permitted types for the type field are defined in Table 17.

        See 3.3.4, "Value Representation" for information on how the
        value data is represented in the packet value field.

   For an unsuccessful GET, GETNEXT or GETBULK, the sub-agent does not
   need to return any name/type/length/value tuple(s), because by
   definition, the varBind information is the same as in the request to
   which this is a response, and the agent still has that information.

   The group ID and the packet ID must always be the same as the
   corresponding fields in request PDU which has prompted the RESPONSE.

   If the response is to a SET, COMMIT or UNDO request, there is no need
   to return any varBind information, because by definition, the varBind
   information is the same as in the request to which this is a
   response, and the agent still has that information.

   If the response is to a REGISTER or UNREGISTER, no variable
   (instance) is being returned, so the instance ID is the NULL string
   (one 0x00 byte). In the response to a REGISTER request indicating
   success, the error index contains the priority assigned by the agent.

   If the response is to an OPEN, ARE_YOU_THERE or CLOSE, no varBind
   data will be passed, so no group ID, instance ID or value data. The
   packet will only include the header, the error code and the error

index.

```
+----------------------------------------------------------------+
| Table 13. Layout SNMP DPI RESPONSE packet                      |
+-----------+----------------------------------------------------+
| OFFSET    | FIELD                                              |
+-----------+----------------------------------------------------+
| 0         | packet length to follow (MSB to LSB)               |
+-----------+----------------------------------------------------+
| 2         | protocol major version                             |
+-----------+----------------------------------------------------+
| 3         | protocol minor version                             |
+-----------+----------------------------------------------------+
| 4         | protocol release                                   |
+-----------+----------------------------------------------------+
| 5         | packet id (MSB to LSB)                             |
+-----------+----------------------------------------------------+
| 7         | packet type = SNMP_DPI_RESPONSE                    |
+-----------+----------------------------------------------------+
| 8         | error code (1 octet)                               |
+-----------+----------------------------------------------------+
| 9         | error index (4 octets, MSB to LSB)                 |
+-----------+----------------------------------------------------+
| 15        | null terminated group ID (with trailing dot)       |
+-----------+----------------------------------------------------+
| 15+L1     | null terminated instance ID (no trailing dot)      |
+-----------+----------------------------------------------------+
| 15+L2     | SNMP Variable Type Value                           |
+-----------+----------------------------------------------------+
| 15+L2+1   | Length of value (MSB to LSB)                       |
+-----------+----------------------------------------------------+
| 15+L2+3   | Value                                              |
+-----------+----------------------------------------------------+
| 15+L3     | optionally more varBinds (sequences of group ID,   |
|           | instance ID, Type, Length and Value)               |
+-----------+----------------------------------------------------+
| NOTE:                                                          |
|                                                                |
| o   L1 = strlen(group ID) + 1                                  |
| o   L2 = L1 + strlen(instance ID) + 1                          |
| o   L3 = L2 + 3 + length of value                              |
| o   group and instance ID strings use selected character set   |
| o   OID and DisplayString values use selected character set    |
| o   Integer values are in network byte order                   |
+----------------------------------------------------------------+
```

3.2.12 TRAP

   An SNMP sub-agent can request the agent to generate an SNMPv1 or
   SNMPv2 TRAP (depending on the trap destinations defined at the agent)
   by sending an SNMP DPI TRAP packet to the agent.

   Such a packet contains the standard SNMP DPI header plus TRAP
   specific data:

     o   the generic and specific trap codes
     o   optionally a null terminated string (in the selected character
         set) representing the enterprise ID in ASN.1 dotted notation.
         This enterprise ID will be sent with the TRAP. If the null
         string is passed, then the agent uses the sub-agent Identifier
         (OID as passed with the DPI OPEN packet) as the Enterprise ID.
     o   optionally a set of one or more name/type/length/value tuples.
         representing varBinds to be sent with the trap. Each varBind
         consists of:

         -   two null terminated strings (in the selected character set)
             representing the group and instance ID in ASN.1 dotted
             notation.
         -   the type, value length and value of the object that is
             returned.

         The permitted types for the type field are defined in Table 17.

         See 3.3.4, "Value Representation" for information on how the
         value data is represented in the packet value field.

```
+------------------------------------------------------------------+
| Table 14. Layout SNMP DPI TRAP packet                            |
+-----------+------------------------------------------------------+
| OFFSET    | FIELD                                                |
+-----------+------------------------------------------------------+
| 0         | packet length to follow (MSB to LSB)                 |
+-----------+------------------------------------------------------+
| 2         | protocol major version                               |
+-----------+------------------------------------------------------+
| 3         | protocol minor version                               |
+-----------+------------------------------------------------------+
| 4         | protocol release                                     |
+-----------+------------------------------------------------------+
| 5         | packet id (MSB to LSB)                               |
+-----------+------------------------------------------------------+
| 7         | packet type - SNMP_DPI_TRAP                          |
+-----------+------------------------------------------------------+
| 8         | SNMP generic trap code                               |
+-----------+------------------------------------------------------+
| 12        | SNMP specific trap code                              |
+-----------+------------------------------------------------------+
| 14        | null terminated enterprise ID (no trailing dot)      |
+-----------+------------------------------------------------------+
| 14+L1     | null terminated group ID (with trailing dot)         |
+-----------+------------------------------------------------------+
| 14+L2     | null terminated instance ID (no trailing dot)        |
+-----------+------------------------------------------------------+
| 14+L3     | SNMP Variable Type Value                             |
+-----------+------------------------------------------------------+
| 14+L3+1   | Length of value (MSB to LSB)                         |
+-----------+------------------------------------------------------+
| 14+L3+3   | Value                                                |
+-----------+------------------------------------------------------+
| 14+L4     | optionally more varBinds (sequences of group ID,     |
|           | instance ID, Type, Length and Value)                 |
+-----------+------------------------------------------------------+
| NOTE:                                                            |
|                                                                  |
| o   L1 = strlen(enterprise ID) + 1                               |
| o   L2 = L1 + strlen(group ID) + 1                               |
| o   L3 = L1 + L2 + strlen(instance ID) + 1                       |
| o   L4 = L1 + L2 + L3 + 3 + length of Value                      |
| o   enterprise, group and instance ID strings use selected       |
|     character set                                                |
| o   OID and DisplayString values use selected character set      |
| o   Integer values are in network byte order                     |
+------------------------------------------------------------------+
```

3.3 CONSTANTS AND VALUES

    This section describes the constants that have been defined for this
    version of the SNMP DPI Protocol.

3.3.1 PROTOCOL VERSION AND RELEASE VALUES

```
    +-------------------------------------------------------------------+
    | Table 15. Protocol version and release values                     |
    +-------------------------------+-----------------------------------+
    | FIELD                         | VALUE                             |
    +-------------------------------+-----------------------------------+
    | protocol major version        | 2 (SNMP DPI protocol)             |
    +-------------------------------+-----------------------------------+
    | protocol minor version        | 2 (version 2)                     |
    +-------------------------------+-----------------------------------+
    | protocol release              | 0 (release 0)                     |
    +-------------------------------+-----------------------------------+
```

    Previous versions of this protocol exist and should preferably be
    supported by an agent:

        o   version 1, release 0, described in [7]

    Previous internal versions of this protocol exist and may or may not
    be supported by an agent:

        o   version 1, release 1, experimental within IBM.
        o   version 1, release 2, experimental within BNR.

3.3.2 PACKET TYPE VALUES

```
+-------------------------------------------------------------------+
| Table 16. Valid values for the packet type field                 |
+-------+-----------------------------------------------------------+
| VALUE | PACKET TYPE                                               |
+-------+-----------------------------------------------------------+
| 1     | SNMP_DPI_GET                                              |
+-------+-----------------------------------------------------------+
| 2     | SNMP_DPI_GETNEXT                                          |
+-------+-----------------------------------------------------------+
| 3     | SNMP_DPI_SET                                              |
+-------+-----------------------------------------------------------+
| 4     | SNMP_DPI_TRAP                                             |
+-------+-----------------------------------------------------------+
| 5     | SNMP_DPI_RESPONSE                                         |
+-------+-----------------------------------------------------------+
| 6     | SNMP_DPI_REGISTER                                        |
+-------+-----------------------------------------------------------+
| 7     | SNMP_DPI_UNREGISTER                                      |
+-------+-----------------------------------------------------------+
| 8     | SNMP_DPI_OPEN                                             |
+-------+-----------------------------------------------------------+
| 9     | SNMP_DPI_CLOSE                                            |
+-------+-----------------------------------------------------------+
| 10    | SNMP_DPI_COMMIT                                          |
+-------+-----------------------------------------------------------+
| 11    | SNMP_DPI_UNDO                                            |
+-------+-----------------------------------------------------------+
| 12    | SNMP_DPI_GETBULK                                         |
+-------+-----------------------------------------------------------+
| 13    | SNMP_DPI_TRAPV2 (not yet used)                           |
+-------+-----------------------------------------------------------+
| 14    | SNMP_DPI_INFORM (not yet used)                           |
+-------+-----------------------------------------------------------+
| 15    | SNMP_DPI_ARE_YOU_THERE                                    |
+-------+-----------------------------------------------------------+
```

3.3.3 VARIABLE TYPE VALUES

```
+----------------------------------------------------------------+
| Table 17. Valid values for the Value Type field                |
+-------+--------------------------------------------------------+
| VALUE | VALUE TYPE                                             |
+-------+--------------------------------------------------------+
| 129   | SNMP_TYPE_Integer32                                    |
+-------+--------------------------------------------------------+
| 2     | SNMP_TYPE_OCTET_STRING                                 |
+-------+--------------------------------------------------------+
| 3     | SNMP_TYPE_OBJECT_IDENTIFIER                            |
+-------+--------------------------------------------------------+
| 4     | SNMP_TYPE_NULL (empty, no value)                      |
+-------+--------------------------------------------------------+
| 5     | SNMP_TYPE_IpAddress                                    |
+-------+--------------------------------------------------------+
| 134   | SNMP_TYPE_Counter32                                    |
+-------+--------------------------------------------------------+
| 135   | SNMP_TYPE_Gauge32                                      |
+-------+--------------------------------------------------------+
| 136   | SNMP_TYPE_TimeTicks (1/100ths seconds)                |
+-------+--------------------------------------------------------+
| 9     | SNMP_TYPE_DisplayString                               |
+-------+--------------------------------------------------------+
| 10    | SNMP_TYPE_BIT_STRING                                   |
+-------+--------------------------------------------------------+
| 11    | SNMP_TYPE_NsapAddress                                  |
+-------+--------------------------------------------------------+
| 140   | SNMP_TYPE_UInteger32                                   |
+-------+--------------------------------------------------------+
| 13    | SNMP_TYPE_Counter64                                    |
+-------+--------------------------------------------------------+
| 14    | SNMP_TYPE_Opaque                                       |
+-------+--------------------------------------------------------+
| 15    | SNMP_TYPE_noSuchObject                                 |
+-------+--------------------------------------------------------+
| 16    | SNMP_TYPE_noSuchInstance                               |
+-------+--------------------------------------------------------+
| 17    | SNMP_TYPE_endOfMibView                                 |
+-------+--------------------------------------------------------+
```

Notes:

1. A 32-bit integer value has its base base type ORed with 128.
2. DisplayString is a textual convention. An SNMP PDU shows a
    type of OCTET_STRING for the value. An agent can handle such
    an object as DisplayString if the object is included in some

form of a compiled MIB for the agent. If not, the agent passes
the value as an OCTET_STRING.

3.3.4 VALUE REPRESENTATION

   Values in the DPI packets are represented as follows:

   o   32-bit integers are 4-byte elements in network byte order, MSB
       (most significant byte) first, LSB (least significant byte)
       last. Example: '00000001'h represents 1.
   o   64-bit integers are 8-byte elements in network byte order, MSB
       first, LSB last.
       Example: '0000000100000001'h represents 4,294,967,297.
   o   Object Identifiers are NULL terminated strings in the selected
       character set, representing the OID in ASN.1 dotted notation.
       The length includes the terminating NULL.
       Example ASCII: '312e332e362e312e322e312e312e312e3000'h
       represents "1.3.6.1.2.1.1.1.0" which is sysDescr.0.
       Example EBCDIC: 'f14bf34bf64bf14bf24bf14bf14bf14bf000'h
       represents "1.3.6.1.2.1.1.1.0" which is sysDescr.0.
   o   DisplayStrings are in the selected character set. The length
       specifies the length of the string.
       Example ASCII: '6162630d0a'h represents "abc\r\n", no NULL.
       Example EBCDIC: '8182830d25'h represents "abc\r\n", no NULL.
   o   IpAddress, NsapAddress and Opaque are implicit OCTET_STRING, so
       they are octets (e.g. IpAddress in network byte order).
   o   NULL has a zero length for the value, no value data.
   o   noSuchObject, noSuchInstance and endOfMibView are implicit NULL
       and represented as such.
   o   BIT_STRING is an OCTET_STRING of the form uubbbb...bb, where
       the first octet (uu) is 0x00-0x07 and indicates the number of
       unused bits in the last octet (bb). The bb octets represent the
       bit string itself, where bit zero (0) comes first and so on.

3.3.5 CHARACTER SET SELECTION

   In the DPI OPEN packet, the sub-agent can specify the character set
   to be used for the representation of:

   o   group and instance ID in the DPI REGISTER, UNREGISTER, GET,
       GETNEXT, GETBULK, SET, UNDO, COMMIT, RESPONSE and TRAP packets.
   o   sub-agent ID and sub-agent Description in DPI OPEN packet.
   o   Object Identifiers in the value field for a value of type
       SNMP_TYPE_OBJECT_IDENTIFIER.
   o   DisplayString in the value field for a value of type
       SNMP_TYPE_DPI_DisplayString.

   The choice is the native character set or the ASCII character set.

The native set is the set native to the platform where the agent
runs. If the native set is ASCII, then character set selection is a
moot point. On non-ASCII based platforms, the agent must convert
between native and ASCII if the native character set is chosen.

3.3.6 ERROR CODE VALUES FOR SNMP DPI RESPONSE PACKETS

When the RESPONSE packet is a response to a GET, GETNEXT, GETBULK,
SET, COMMIT, or UNDO, then the error code can have one of the
following values:

```
+----------------------------------------------------------------+
| Table 18. Valid SNMP_ERROR values for RESPONSE error code      |
+-------+--------------------------------------------------------+
| VALUE | ERROR CODE                                             |
+-------+--------------------------------------------------------+
| 0     | SNMP_ERROR_noError                                     |
+-------+--------------------------------------------------------+
| 1     | SNMP_ERROR_tooBig                                      |
+-------+--------------------------------------------------------+
| 2     | SNMP_ERROR_noSuchName (SNMPv1, do not use)             |
+-------+--------------------------------------------------------+
| 3     | SNMP_ERROR_badValue (SNMPv1, do not use)               |
+-------+--------------------------------------------------------+
| 4     | SNMP_ERROR_readOnly (SNMPv1 do not use)                |
+-------+--------------------------------------------------------+
| 5     | SNMP_ERROR_genErr                                      |
+-------+--------------------------------------------------------+
| 6     | SNMP_ERROR_noAccess                                    |
+-------+--------------------------------------------------------+
| 7     | SNMP_ERROR_wrongType                                   |
+-------+--------------------------------------------------------+
| 8     | SNMP_ERROR_wrongLength                                 |
+-------+--------------------------------------------------------+
| 9     | SNMP_ERROR_wrongEncoding                               |
+-------+--------------------------------------------------------+
| 10    | SNMP_ERROR_wrongValue                                  |
+-------+--------------------------------------------------------+
| 11    | SNMP_ERROR_noCreation                                  |
+-------+--------------------------------------------------------+
| 12    | SNMP_ERROR_inconsistentValue                           |
+-------+--------------------------------------------------------+
| 13    | SNMP_ERROR_resourceUnavailable                         |
+-------+--------------------------------------------------------+
| 14    | SNMP_ERROR_commitFailed                                |
+-------+--------------------------------------------------------+
| 15    | SNMP_ERROR_undoFailed                                  |
+-------+--------------------------------------------------------+
| 16    | SNMP_ERROR_authorizationError                          |
+-------+--------------------------------------------------------+
| 17    | SNMP_ERROR_notWritable                                 |
+-------+--------------------------------------------------------+
| 18    | SNMP_ERROR_inconsistentName                            |
+-------+--------------------------------------------------------+
```

When the RESPONSE packet is a response to an OPEN, REGISTER or
UNREGISTER, then the error code can have one of the following values:

```
+----------------------------------------------------------------+
| Table 19. Valid SNMP_ERROR_DPI values for RESPONSE error code  |
+-------+--------------------------------------------------------+
| VALUE | ERROR CODE                                             |
+-------+--------------------------------------------------------+
| 0     | SNMP_ERROR_DPI_noError                                 |
+-------+--------------------------------------------------------+
| 101   | SNMP_ERROR_DPI_otherError                              |
+-------+--------------------------------------------------------+
| 102   | SNMP_ERROR_DPI_notFound                                |
+-------+--------------------------------------------------------+
| 103   | SNMP_ERROR_DPI_alreadyRegistered                       |
+-------+--------------------------------------------------------+
| 104   | SNMP_ERROR_DPI_higherPriorityRegistered                |
+-------+--------------------------------------------------------+
| 105   | SNMP_ERROR_DPI_mustOpenFirst                           |
+-------+--------------------------------------------------------+
| 106   | SNMP_ERROR_DPI_notAuthorized                           |
+-------+--------------------------------------------------------+
| 107   | SNMP_ERROR_DPI_viewSelectionNotSupported               |
+-------+--------------------------------------------------------+
| 108   | SNMP_ERROR_DPI_getBulkSelectionNotSupported            |
+-------+--------------------------------------------------------+
| 109   | SNMP_ERROR_DPI_duplicateSubAgentIdentifier             |
+-------+--------------------------------------------------------+
| 110   | SNMP_ERROR_DPI_invalidDisplayString                    |
+-------+--------------------------------------------------------+
| 111   | SNMP_ERROR_DPI_characterSetSelectionNotSupported       |
+-------+--------------------------------------------------------+
```

3.3.7 UNREGISTER REASON CODES

   The following are valid reason codes in an UNREGISTER packet.

```
   +---------------------------------------------------------------+
   | Table 20. Valid UNREGISTER reason codes                       |
   +-------+-------------------------------------------------------+
   | VALUE | REASON CODE                                           |
   +-------+-------------------------------------------------------+
   | 1     | SNMP_UNREGISTER_otherReason                           |
   +-------+-------------------------------------------------------+
   | 2     | SNMP_UNREGISTER_goingDown                             |
   +-------+-------------------------------------------------------+
   | 3     | SNMP_UNREGISTER_justUnregister                        |
   +-------+-------------------------------------------------------+
   | 4     | SNMP_UNREGISTER_newRegistration                       |
   +-------+-------------------------------------------------------+
   | 5     | SNMP_UNREGISTER_higherPriorityRegistered              |
   +-------+-------------------------------------------------------+
   | 6     | SNMP_UNREGISTER_byManager                             |
   +-------+-------------------------------------------------------+
   | 7     | SNMP_UNREGISTER_timeout                               |
   +-------+-------------------------------------------------------+
```

3.3.8 CLOSE REASON CODES

   The following are valid reason codes in a CLOSE packet.

```
   +------------------------------------------------------------------+
   | Table 21. Valid CLOSE reason codes                               |
   +-------+----------------------------------------------------------+
   | VALUE | REASON CODE                                              |
   +-------+----------------------------------------------------------+
   | 1     | SNMP_CLOSE_otherReason                                   |
   +-------+----------------------------------------------------------+
   | 2     | SNMP_CLOSE_goingDown                                     |
   +-------+----------------------------------------------------------+
   | 3     | SNMP_CLOSE_unsupportedVersion                            |
   +-------+----------------------------------------------------------+
   | 4     | SNMP_CLOSE_protocolError                                 |
   +-------+----------------------------------------------------------+
   | 5     | SNMP_CLOSE_authenticationFailure                         |
   +-------+----------------------------------------------------------+
   | 6     | SNMP_CLOSE_byManager                                     |
   +-------+----------------------------------------------------------+
   | 7     | SNMP_CLOSE_timeout                                       |
   +-------+----------------------------------------------------------+
   | 8     | SNMP_CLOSE_openError                                     |
   +-------+----------------------------------------------------------+
```

4. DPI 2.0 MIB DEFINITION

   DPI20-MIB DEFINITIONS ::= BEGIN

   -- Objects in this MIB are implemented in the local SNMP agent.

      IMPORTS
              MODULE-IDENTITY, OBJECT-TYPE, snmpModules, enterprises
                   FROM SNMPv2-SMI

      ibm   OBJECT IDENTIFIER ::= { enterprises 2 }
      ibmDPI OBJECT IDENTIFIER ::= { ibm 2 }
      dpi20MIB OBJECT IDENTIFIER ::= { ibmDPI 1 }

   -- dpi20MIB MODULE-IDENTITY
   --   LAST-UPDATED "9401210000Z"
   --   ORGANIZATION "IBM Research - T.J. Watson Research Center"
   --   CONTACT-INFO "   Bert Wijnen
   --   Postal: IBM International Operations
   --   Watsonweg 2
   --   1423 ND Uithoorn
   --   The Netherlands

```
      --                   Tel:        +31 2975 53316
      --                   Fax:        +31 2975 62468
      --                   E-mail:     wijnen@vnet.ibm.com"
      --    DESCRIPTION "MIB module describing DPI objects."
      --    ::= { snmpModules x }

          dpiPort   OBJECT IDENTIFIER ::= { dpi20MIB 1 }


          dpiPortForTCP   OBJECT-TYPE
                  SYNTAX   INTEGER (0..65535)
                  ACCESS   read-only
                  STATUS   mandatory
                  DESCRIPTION "The TCP port number on which the agent
                               listens for DPI connections. A zero value
                               means the agent has no DPI TCP port."
                  ::= { dpiPort 1 }


          dpiPortForUDP   OBJECT-TYPE
                  SYNTAX   INTEGER (0..65535)
                  ACCESS   read-only
                  STATUS   mandatory
                  DESCRIPTION "The UDP port number on which the agent
                               listens for DPI packets. A zero value
                               means the agent has no DPI UDP port."
                  ::= { dpiPort 2 }
      END
```

5. SUBAGENT CONSIDERATIONS

   When implementing a sub-agent, it is strongly recommended to use the
   DPI version 2 approach (SNMPv2 based). This means:

   o   Use SNMPv2 error codes only (even though we have definitions
       for the old SNMPv1 error codes).
   o   Do implement SET, COMMIT, UNDO processing properly.
   o   For GET requests, use the SNMPv2 approach and pass back
       noSuchInstance or noSuchObject value if such is the case.
       Continue to process all remaining varBinds in this case.
   o   For GETNEXT, use the SNMPv2 approach and pass back endOfMibView
       value if such is the case. Continue to process all remaining
       varBinds in this case.
   o   When you are processing a request from the agent (GET, GETNEXT,
       GETBULK, SET, COMMIT, UNDO) you are supposed to respond within
       the timeout period (which you can specify in the OPEN and
       REGISTER packets). If you fail to respond within that timeout
       period, the agent will most probably close your DPI connection
       and then discard your RESPONSE packet if it comes in later. If
       you can detect that the response is not going to make it in

          time, then you might decide to abort the request and return an
          SNMP_ERROR_genErr in the RESPONSE.
     o   If you have a UDP "connected" sub-agent, or one that uses
          another unreliable protocol, you may want to issue an SNMP DPI
          ARE_YOU_THERE request once in a while to ensure that the agent
          is still alive and still knows about you.
     o   When you are running on an EBCDIC based machine, and you use
          the (default) native character set, then all OID strings (as
          used for things like group ID, instance ID, Enterprise ID,
          sub-agent ID) and also all variable values of type
          OBJECT_IDENTIFIER or DisplayString will be passed to you in
          EBCDIC format. When you return a response, you should then
          also use EBCDIC FORMAT.
     o   When you are running on an EBCDIC based machine, and you use
          the ASCII character set (specified in DPI OPEN), then all OID
          strings (as used for things like group ID, instance ID,
          Enterprise ID, sub-agent ID) and also all variable values of
          type OBJECT_IDENTIFIER or DisplayString will be passed to you
          in ASCII format. When you return a response, you should then
          also use ASCII FORMAT.
     o   When you are running on an ASCII machine, then the character
          set selection for you basically is moot. Except maybe when you
          connect to an EBCDIC based agent, in which case you may want to
          specify in the DPI OPEN packet that you want to use ASCII
          character set. After that, all this is transparent to you and
          the burden of conversion is on the EBCDIC based agent.
     o   Please realize that DisplayString is only a textual convention.
          In the SNMP PDU (SNMP packet), the type is just an
          OCTET_STRING, and from that it is not clear if this is a
          DisplayString or any arbitrary data. This means that the agent
          can only know about an object being a DisplayString if the
          object is included in some sort of a compiled MIB. If it is,
          then the agent will use SNMP_TYPE_DisplayString in the type
          field of the varBind in a DPI SET packet. When you send a
          DisplayString in a RESPONSE packet, the agent will handle it as
          such (e.g. translate EBCDIC to ASCII if needed).

5.1 DPI API

   The primary goal of this document is to specify the SNMP DPI, a
   protocol by which sub-agents can exchange SNMP related information
   with an agent. On top of this protocol, one can imagine one or
   possibly many Application Programming Interfaces, but those are not
   addressed in this document.

However, in order to provide an environment that is more or less
platform independent, we strongly suggest to also define a DPI API.
We have a sample DPI API available, see 9, "Sample Sources for
Anonymous FTP" for a place to obtain that sample DPI API.

5.2 OVERVIEW OF REQUEST PROCESSING

5.2.1 GET PROCESSING

A GET request is the easiest to process. The DPI GET packet holds
one or more varBinds that the sub-agent has taken responsibility for.

If the sub-agent encounters an error while processing the request, it
creates a DPI RESPONSE packet with an appropriate error indication in
the error_code field and sets the error_index to the position of the
varBind at which the error occurs (first varBind is index 1, second
varBind is index 2, and so on). No name/type/length/value
information needs to be provided in the packet, because by
definition, the varBind information is the same as in the request to
which this is a response, and the agent still has that information.

If there are no errors, then the sub-agent creates a DPI RESPONSE
packet in which the error_code is set to SNMP_ERROR_noError (zero)
and error_index is set to zero. The packet must also include the
name/type/length/value of each varBind requested. When you get a
request for a non-existing object or a non-existing instance of an
object, then you must return a NULL value with a type of
SNMP_TYPE_noSuchObject or SNMP_TYPE_noSuchInstance respectively.

These two values are not considered errors, so the error_code and
error_index should be zero.

The DPI RESPONSE packet is then sent back to the agent.

5.2.2 SET PROCESSING

Processing a DPI SET request is more difficult than a DPI GET
request. In the case of a DPI SET packet, additional information is
available in the packet, namely the value type, value length and
value to be set.

If the sub-agent encounters an error while processing the request, it
creates a DPI RESPONSE packet with an appropriate error indication in
the error_code field and an error_index listing the position of the
varBind at which the error occurs (first varBind is index 1, second
varBind is index 2, and so on). No name/type/length/value
information needs to provided in the packet, because by definition,
the varBind information is the same as in the request to which this

is a response, and the agent still has that information.

If there are no errors, then the sub-agent creates a DPI RESPONSE
packet in which the error_code is set to SNMP_ERROR_noError (zero)
and error_index is set to zero. No name/type/length/value
information is needed; by definition the RESPONSE to a SET should
contain exactly the same varBind data as the data present in the
request, so the agent can use the values it already has. (This
suggests that the agent must keep state information, and that is
indeed the case. It needs to do that anyway in order to be able to
later pass the data with a DPI COMMIT or DPI UNDO packet). The sub-
agent must have allocated the required resources and prepared itself
for the SET. It does not yet effectuate the set, that will be done
at COMMIT time.

The sub-agent sends a DPI RESPONSE packet (indicating success or
failure for the preparation phase) back to the agent.

The agent will then issue a SET request for all other varBinds in the
same original SNMP request it received. This may be to the same or
to one or more different sub-agents. Once all SET requests have
returned a "no error" condition, the agent starts sending DPI COMMIT
packets to the sub-agent(s). If any SET request returns an error,
then the agent sends DPI UNDO packets to those sub-agents that
indicated successful processing of the SET preparation phase.

When the sub-agent receives the DPI COMMIT packet, again all the
varBind information will be available in the packet. The sub-agent
can now effectuate the SET request.

If the sub-agent encounters an error while processing the COMMIT
request, it creates a DPI RESPONSE packet with value
SNMP_ERROR_commitFailed in the error_code field and an error_index
that lists at which varBind the error occurs (first varBind is index
1 and so on). No name/type/length/value information is needed. The
fact that a commitFailed error exists does not mean that this error
should be returned easily. A sub-agent should do all that is
possible to make a COMMIT succeed.

If there are no errors, and the SET/COMMIT has been effectuated with
success, then the sub-agent creates a DPI RESPONSE packet in which
the error_code is set to SNMP_ERROR_noError (zero) and error_index is
set to zero. No name/type/length/value information is needed.

So far we have discussed a SET, COMMIT sequence. That happens if all
goes well. However, after a successful SET, the sub-agent may
receive a DPI UNDO packet. The sub-agent must now undo any
preparations it made during the SET processing (like free allocated

memory and such). Even after a COMMIT, a sub-agent may still receive
a DPI UNDO packet. This is the case if some other sub-agent could
not complete a COMMIT request. Because of the SNMP-requirement that
all varBinds in a single SNMP SET request must be changed "as if
simultaneous", all committed changes must be undone if any of the
COMMIT requests fail. In this case the sub-agent must try and undo
the committed SET operation.

If the sub-agent encounters an error while processing the UNDO
request, it creates a DPI RESPONSE packet with value
SNMP_ERROR_undoFailed in the error_code field and an error_index that
lists at which varBind the error occurs (first varBind is index 1 and
so on). No name/type/length/value information is needed. The fact
that an undoFailed error exists does not mean that this error should
be returned easily. A sub-agent should do all that is possible to
make an UNDO succeed.

If there are no errors, and the UNDO has been effectuated with
success, then the sub-agent creates a DPI RESPONSE packet in which
the error_code is set to SNMP_ERROR_noError (zero) and error_index is
set to zero. No name/type/length/value information is needed.

5.2.3 GETNEXT PROCESSING

GETNEXT requests are a bit more complicated to process than a GET.
The DPI GETNEXT packet contains the object(s) on which the GETNEXT
operation must be performed. The semantics of the operation are that
the sub-agent is to return the name/type/length/value of the next
variable it supports whose (ASN.1) name lexicographically follows the
one passed in the group ID (sub-tree) and instance ID.

In this case, the instance ID may not be present (NULL) implying that
the NEXT object must be the first instance of the first object in the
sub-tree that was registered.

It is important to realize that a given sub-agent may support several
discontiguous sections of the MIB tree. In such a situation it would
be incorrect to jump from one section to another. This problem is
correctly handled by examining the group ID in the DPI packet. This
group ID represents the "reason" why the sub-agent is being called.
It holds the prefix of the tree that the sub-agent had indicated it
supported (registered).

If the next variable supported by the sub-agent does not begin with
that prefix, the sub-agent must return the same object instance as in
the request (e.g. group ID and instance ID) with a value of
SNMP_TYPE_endOfMibView (implied NULL value). This endOfMibView is
not considered an error, so the error_code and error_index should be

zero. If required, the SNMP agent will call upon the sub-agent
again, but pass it a different group ID (prefix). This is
illustrated in the discussion below.

Assume there are two sub-agents. The first sub-agent registers two
distinct sections of the tree, A and C. In reality, the sub-agent
supports variables A.1 and A.2, but it correctly registers the
minimal prefix required to uniquely identify the variable class it
supports.

The second sub-agent registers a different section, B, which appears
between the two sections registered by the first agent.

If a management station begins dumping the MIB, starting from A, the
following sequence of queries of the form get-next(group ID, instance
ID) would be performed:

```
    Sub-agent 1 gets called:
            get-next(A,none) = A.1
            get-next(A,1)    = A.2
            get-next(A,2)    = endOfMibView

    Sub-agent 2 is then called:
            get-next(B,none) = B.1
            get-next(B,1)    = endOfMibView

    Sub-agent 1 gets called again:
            get-next(C,none) = C.1
```

5.2.4 GETBULK PROCESSING

   You can ask the agent to translate GETBULK requests into multiple
   GETNEXT requests. This is basically the default and it is specified
   in the DPI REGISTER packet. In principle, we expect the majority of
   DPI sub-agents to run on the same machine as the agent (or otherwise,
   on the same physical network), so repetitive GETNEXT requests stay
   local and in general should not be a problem.

   If experience tells us different, the sub-agent can tell the agent to
   pass on a DPI GETBULK packet.

   When a GETBULK request is received, the sub-agent must process the
   request and send a RESPONSE that sends back as many varBinds as
   requested by the request, as long as they fit with in the buffers.

   The GETBULK requires similar processing as a GETNEXT with regard to
   endOfMibView handling.

5.2.5 OPEN REQUEST

   As the very first step, a DPI sub-agent must open a "connection" with
   the agent. To do so, it must send a DPI OPEN packet in which these
   things must be specified:

   o   The max timeout value in seconds. The agent is requested to
        wait this long for a response to any request for an object
        being handled by this sub-agent. The agent may have an
        absolute maximum timeout value which will be used if the
        sub-agent asks for too big a timeout value. A value of zero
        can be used to indicate that the agent's own default timeout
        value should be used. A sub-agent is advised to use a
        reasonably short interval of a few seconds or so. If a
        specific sub-tree needs a (much) longer time, then a specific
        REGISTER can be done for that sub-tree with a longer timeout
        value.
   o   The maximum number of varBinds that the sub-agent is prepared
        to handle per DPI packet. Specifying 1 would result in DPI
        version 1 behavior of one varBind per DPI packet that the agent
        sends to the sub-agent.
   o   The character set you want to use. By default (value 0) this is
        the native character set of the machine (platform) where the
        agent runs.
        Since the sub-agent and agent normally run on the same system
        or platform, you want to use the native character set (which on
        many platforms is ASCII anyway).
        If your platform is EBCDIC based, then using the native
        character set of EBCDIC makes it easy to recognize the string
        representations of the fields like group ID, instance ID, etc.

        At the same time, the agent will translate the value from ASCII
        NVT to EBCDIC (and vice versa) for objects that it knows (from
        a compiled MIB) to have a textual convention of DisplayString.
        Be aware that this fact cannot be determined from the SNMP PDU
        encoding because in the PDU the object is only known to be an
        OCTET_STRING.
        If your sub-agent runs on an ASCII based platform and the agent
        runs on an EBCDIC based platform (or the other way around),
        then you can specify that you want to use the ASCII character
        set, and so you both know how to handle the string-based data.
        Beware that not all agents need to support other than native
        character set selection. See 5, "Subagent Considerations"
        and 3.3.5, "Character set selection" for more information on
        character set usage.
   o   The sub-agent ID. This an ASN.1 Object Identifier that
        uniquely identifies the sub-agent. This OID is represented as
        a null terminated string using the selected character set.

              Example: "1.3.5.1.2.3.4.5".
       o   The sub-agent Description. This is a DisplayString describing
             the sub-agent. This is a character string using the selected
             character set. Example: "DPI sample sub-agent version 2.0"

   Once a sub-agent has sent a DPI OPEN packet to an agent, it should
   expect a DPI RESPONSE packet that informs the sub-agent about the
   result of the request. The packet ID of the RESPONSE packet should
   be the same as that of the OPEN request to which the RESPONSE packet
   is the response. See Table 19 for a list of valid DPI RESPONSE error
   codes that may be expected. If you receive an error RESPONSE on the
   OPEN packet, then you will also receive a DPI CLOSE packet with an
   SNMP_CLOSE_openError code, and then the agent closes the
   "connection".

   If the OPEN is accepted, then the next step is to REGISTER one or
   more MIB sub-trees.

5.2.6 CLOSE REQUEST

   When a sub-agent is finished and wants to terminate it should first
   UNREGISTER its sub-trees and then close the "connection" with the
   agent. To do so, it must send a DPI CLOSE packet in which it
   specifies a reason for the closing. See Table 21 for a list of valid
   CLOSE reason codes. You should not expect a response to the CLOSE
   request.

   A sub-agent should also be prepared to handle an incoming DPI CLOSE
   packet from the agent. Again, the packet will contain a reason code
   for the CLOSE request. A sub-agent need not send a response to a
   CLOSE request. The agent just assumes that the sub-agent will handle
   it appropriately. The close takes place, no matter what the sub-
   agent does with it.

5.2.7 REGISTER REQUEST

   Before a sub-agent will receive any requests for MIB variables it
   must first register the variables or sub-tree it supports with the
   SNMP agent. The sub-agent must specify a number of things in the
   REGISTER request:

   o   The sub-tree to be registered. This is a null terminated
         string in the selected character set. The sub-tree must have a
         trailing dot (example: "1.3.6.1.2.3.4.5.").
   o   The requested priority for the registration, one of:
         -1 Request for best available priority.
         0   Request for next better available priority than highest
             priority currently registered for this sub-tree.

        NNN Any other positive value requests that specific priority if
            available or the first worse priority that is available.
   o   The max timeout value in seconds. The agent is requested to
        wait this long for a response to any request for an object in
        this sub-tree. The agent may have an absolute maximum timeout
        value which will be used if the sub-agents asks for too big a
        timeout value. A value of zero can be used to indicate that
        the DPI OPEN value should be used for timeout.
   o   A specification if the sub-agent wants to do view selection.
        If it does, then the community name (from SNMPv1 packets) will
        be passed in the DPI GET, GETNEXT, SET packets).
   o   A specification if the sub-agent wants to receive GETBULK
        packets or if it just prefers that the agent converts a GETBULK
        into multiple GETNEXT requests.

   Once a sub-agent has sent a DPI REGISTER packet to the agent, it
   should expect a DPI RESPONSE packet that informs the sub-agent about
   the result of the request. The packet ID of the RESPONSE packet
   should be the same as that of the REGISTER packet to which the
   RESPONSE packet is the response. If the response indicates success,
   then the error_index field in the RESPONSE packet contains the
   priority that the agent assigned to the sub-tree registration. See
   Table 19 for a list of valid DPI RESPONSE error codes that may be
   expected.

5.2.8 UNREGISTER REQUEST

   A sub-agent may unregister a previously registered sub-tree. The
   sub-agent must specify a few things in the UNREGISTER request:

   o   The sub-tree to be unregistered. This is a null terminated
        string in the selected character set. The sub-tree must have a
        trailing dot (example: "1.3.6.1.2.3.4.5.").
   o   The reason for the unregister. See Table 20 for a
        list of valid reason codes.

   Once a sub-agent has sent a DPI UNREGISTER packet to the agent, it
   should expect a DPI RESPONSE packet that informs the sub-agent about
   the result of the request. The packet ID of the RESPONSE packet
   should be the same as that of the REGISTER packet to which the
   RESPONSE packet is the response. See Table 19 for a list of valid
   DPI RESPONSE error codes that may be expected.

   A sub-agent should also be prepared to handle incoming DPI UNREGISTER
   packets from the agent. Again, the DPI packet will contain a reason
   code for the UNREGISTER. A sub-agent need not send a response to an
   UNREGISTER request. The agent just assumes that the sub-agent will
   handle it appropriately. The registration is removed, no matter what

the sub-agent returns.

5.2.9 TRAP REQUEST

   A sub-agent can request that the SNMP agent generates a trap for it.
   The sub-agent must provide the desired values for the generic and
   specific parameters of the trap. It may optionally provide a set of
   one or more name/type/length/value tuples that will be included in
   the trap packet. Also, it may optionally specify an Enterprise ID
   (Object Identifier) for the trap to be generated. If a NULL value is
   specified for the Enterprise ID, then the agent will use the sub-
   agent Identifier (from the DPI OPEN packet) as the Enterprise ID to
   be sent with the trap.

5.2.10 ARE_YOU_THERE REQUEST

   A sub-agent can send an ARE_YOU_THERE packet to the agent. This may
   be useful to do if you have a DPI "connection" over an unreliable
   transport protocol (like UDP).

   If the "connection" is in a healthy state, the agent responds with a
   RESPONSE packet with SNMP_ERROR_DPI_noError.

   If the "connection" is not in a healthy state, the agent may respond
   with a RESPONSE packet with an error indication, but the agent might
   not react at all, so you would timeout while waiting for a response.

5.2.11 HOW TO QUERY THE DPI PORT.

   The DPI API implementations are encouraged to provide a facility that
   helps DPI sub-agent programmers to dynamically find the port that the
   agent is using for the TCP and/or UDP DPI port(s). A suggested name
   for such a function is: query_DPI_port().

6. REFERENCES

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

   [2] Information processing systems - Open Systems Interconnection,
       "Specification of Abstract Syntax Notation One (ASN.1)",
       International Organization for Standardization, International
       Standard 8824, December 1987.

   [3] Information processing systems - Open Systems Interconnection,
       "Specification of Basic Encoding Rules for Abstract Syntax
       Notation One (ASN.1)", International Organization for
       Standardization, International Standard 8825, December 1987.

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

   [5] Rose, M., and K. McCloghrie, "Structure and Identification of
       Management Information for TCP/IP-based internets", STD 16, RFC
       1155, Performance Systems International, Hughes LAN Systems, May
       1990.

   [6] Rose, M., "SNMP MUX Protocol and MIB", RFC 1227, Performance
       Systems International, RFC 1227, May 1991.

   [7] Carpenter G., and B. Wijnen, "SNMP-DPI, Simple Network Management
       Protocol Distributed Program Interface", RFC 1228, International
       Business Machines, Inc., May 1991.

   [8] Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "SNMPv2
       RFCs (RFC 1441 through RFC 1452)", SNMP Research Inc, Hughes LAN
       Systems, Dover Beach Consulting Inc, Carnegie Mellon University,
       Trusted Information Systems, April 1993.

   [9] International Business Machines, Inc., TCP/IP for VM:
       Programmer's Reference, SC31-6084-0, 1990.

  [10] International Business Machines, Inc., Virtual Machine System
       Facilities for Programming, Release 6, SC24-5288-01, 1988.

7. SECURITY CONSIDERATIONS

    Security issues are not discussed in this memo.

8. AUTHORS' ADDRESSES

Bert Wijnen
IBM International Operations
Watsonweg 2
1423 ND Uithoorn
The Netherlands

Phone: +31-2975-53316
Fax: +31-2975-62468
EMail: wijnen@vnet.ibm.com

Geoffrey C. Carpenter
IBM T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
USA

Phone: +1-914-945-1970
EMail: gcc@watson.ibm.com

Kim Curran
Bell Northern Research Ltd.
P.O. Box 3511 Station C
Ottawa, Ontario K1Y 4HY
Canada

Phone: +1-613-763-5283
EMail: kcurran@bnr.ca

Aditya Sehgal
Bell Northern Research Ltd.
P. O. Box 3511 Station C
Ottawa, Ontario K1Y 4HY
Canada

Phone: +1-613-763-8821
EMail: asehgal@bnr.ca

       Glen Waters
       Bell Northern Research Ltd.
       P.O. Box 3511 Station C
       Ottawa, Ontario K1Y 4HY
       Canada

       Phone: +1-613-763-3933
       EMail: gwaters@bnr.ca

9.   SAMPLE SOURCES FOR ANONYMOUS FTP

       An implementation sample of a DPI API (as used at the agent and sub-
       agent side) plus sample sub-agent code and documentation are
       available for anonymous FTP from:

           software.watson.ibm.com (129.34.139.5)

       To obtain the source, perform the following steps:

           ftp software.watson.ibm.com
           user: anonymous
           password: your_e-mail_address
           cd /public/dpi
           get README
           binary
           get dpi_api.tar (or compressed dpi_api.tar.Z)
           quit